

ALTAIR

Altair PBS Professional 2021.1

Big Book

You are reading the Altair PBS Professional 2021.1

Big Book (IG, AG, HG, RG, UG, PG, CG, BG, SG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Main Table of Contents

About PBS Documentation Main-xi

Installation & Upgrade Guide (IG)

Contents IG-v

1 PBS Architecture IG-1

2 Pre-Installation Steps IG-7

3 Installation IG-19

4 Communication IG-45

5 Initial Configuration IG-63

6 Upgrading IG-65

7 Installing and Upgrading on Cray IG-139

8 Starting & Stopping PBS on Linux IG-159

9 Starting & Stopping MoM on Windows IG-171

Index IG-177

Administrator's Guide (AG)

Contents AG-v

1 New Features AG-1

2 Configuring the Server and Queues AG-17

3 Configuring MoMs and Vnodes AG-33

4 Scheduling AG-53

5 Using PBS Resources AG-229

6 Managing Power Usage AG-315
7 Provisioning AG-327
8 Security AG-359
9 Making Your Site More Robust AG-399
10 Using MPI with PBS AG-451
11 Configuring PBS for Cray AG-471
12 Support for HPE AG-499
13 Support for NEC SX-Aurora TSUBASA AG-501
14 Managing Jobs AG-505
15 Administration AG-539
16 Configuring and Using PBS with Cgroups AG-571
17 Configuring PBS for SELinux AG-609
18 Configuring PBS for Containers AG-615
19 Accounting AG-627
20 Mixed Linux-Windows Operation AG-659
21 Problem Solving AG-663
Index AG-677

Hooks Guide (HG)

Contents HG-v
1 New Hook Features HG-1
2 Introduction to Hooks HG-5
3 Quick Start with Hooks HG-11
4 Hook Basics HG-15

5 Creating and Configuring Hooks HG-29
6 Hook Objects and Methods HG-75
7 Built-in Hooks HG-155
8 Debugging Hooks HG-159
9 Hook Examples HG-231
Index HG-289

Reference Guide (RG)

Contents RG-v
1 Glossary of Terms RG-1
2 PBS Commands RG-21
3 MoM Parameters RG-239
4 Scheduler Parameters RG-249
5 List of Built-in Resources RG-259
6 Attributes RG-277
7 Formats RG-353
8 States RG-361
9 The PBS Configuration File RG-369
10 Log Levels RG-375
11 Job Exit Status RG-377
12 Example Configurations RG-379
13 Run Limit Error Messages RG-387
14 Error Codes RG-389
15 Request Codes RG-395

16 PBS Environment Variables RG-399
17 File Listing RG-403
18 Introduction to PBS RG-411
Index RG-413

User's Guide (UG)

Contents UG-v
1 Getting Started with PBS UG-1
2 Submitting a PBS Job UG-11
3 Job Input & Output Files UG-31
4 Allocating Resources & Placing Jobs UG-49
5 Multiprocessor Jobs UG-77
6 Controlling How Your Job Runs UG-107
7 Reserving Resources UG-135
8 Job Arrays UG-149
9 Working with PBS Jobs UG-163
10 Checking Job & System Status UG-171
11 Submitting Cray Jobs UG-191
12 Submitting Jobs to NEC SX-Aurora TSUBASA UG-201
13 Using MLS with PBS Professional UG-211
14 Using Provisioning UG-215
15 Using Accounting UG-221
Index UG-223

Programmer's Guide (PG)

Contents PG-v
List of APIs PG-vii
20 PBS Architecture PG-1
21 Server Functions PG-5
22 Developer Headers and Libraries PG-19
23 Batch Interface Library (IFL) PG-21
24 TM Library PG-93
25 RM Library PG-99
26 TCL/tk Interface PG-103
27 Hooks PG-109
28 Custom Authentication and Encryption Library APIs PG-121
Index PG-133

Cloud Guide (CG)

Contents CG-i
1 Introduction to PBS Cloud CG-1
2 Installing the PBS Cloud Module CG-5
3 Configuring PBS Cloud CG-11
4 Configuring the Cloud Bursting Hook CG-33
5 Using Cloud Provider Services CG-43
6 The Cloud Node Startup Script CG-107
7 Managing Cloud Bursting CG-115
8 Managing Cloud Jobs CG-121
9 Running Cloud Jobs CG-125

10 Example Azure Head Node CG-127
11 Command Reference CG-131
Index CG-143

Budget Guide (BG)

Contents BG-v
1 Introduction to Budget BG-1
2 Installing and Upgrading Budget BG-19
3 Configuring Budget BG-35
4 Budget Commands BG-47
5 Basic Install and Configure BG-97
6 Using Budget BG-103
Index BG-109

Simulate Guide (SG)

Contents SG-v
1 Installing Simulate SG-1
2 Using Simulate SG-3
Index SG-11
Main Index Main-1

About PBS Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

Document Conventions

`Abbr`eviation

The shortest acceptable abbreviation of a command or subcommand is underlined

Attribute

Attributes, parameters, objects, variable names, resources, types

`Command`

Commands such as `qmgr` and `scp`

Definition

Terms being defined

`File name`

File and path names

`Input`

Command-line instructions

Method

Method or member of a class

`Output`

Output, example code, or file contents

Syntax

Syntax, template, synopsis

`Utility`

Name of utility, such as a program

Value

Keywords, instances, states, values, labels

Notation

Optional arguments are enclosed in square brackets. For example:

```
qstat [-E]
```

Variables are enclosed in angle brackets. A variable is something the user must fill in with the correct value. In the following example, the user replaces *vnode name* with the name of the vnode:

```
pbsnodes -v <vnode name>
```

Optional variables are enclosed in angle brackets inside square brackets. For example:

```
qstat [<job ID>]
```

Literal terms appear exactly as they should be used. For example, to get the version of the `qstat` command, type the following exactly:

```
qstat --version
```

Multiple alternative choices are enclosed in curly braces. For example, if you can use either "-n" or "--name":

```
{-n | --name}
```

List of PBS Professional Documentation

The PBS Professional guides and release notes apply to the *commercial* releases of PBS Professional.

PBS Professional Release Notes

Supported platforms, what's new and/or unexpected in this release, deprecations and interface changes, open and closed bugs, late-breaking information. For administrators and job submitters.

PBS Professional Big Book

All your favorite PBS guides in one place: *Installation & Upgrade, Administrator's, Hooks, Reference, User's*, *Programmer's, Cloud, Budget,* and *Simulate* guides in a single book.

PBS Professional Installation & Upgrade Guide

How to install and upgrade PBS Professional. For the administrator.

PBS Professional Administrator's Guide

How to configure and manage PBS Professional. For the PBS administrator.

PBS Professional Hooks Guide

How to write and use hooks for PBS Professional. For the PBS administrator.

PBS Professional Reference Guide

Covers PBS reference material: the PBS commands, resource, attributes, configuration files, etc.

PBS Professional User's Guide

How to submit, monitor, track, delete, and manipulate jobs. For the job submitter.

PBS Professional Programmer's Guide

Discusses the PBS application programming interface (API). For integrators.

PBS Professional Manual Pages

PBS commands, resources, attributes, APIs.

PBS Professional Licensing Guide

How to configure licensing for PBS Professional. For the PBS administrator.

PBS Professional Cloud Guide

How to configure and use the PBS Professional Cloud feature.

PBS Professional Budget Guide

How to configure Budget and use it to track and manage resource usage by PBS jobs.

PBS Professional Simulate Guide

How to configure and use the PBS Professional Simulate feature.

Where to Keep the Documentation

To make cross-references work, put all of the PBS guides in the same directory.

Ordering Software and Licenses

To purchase software packages or additional software licenses, contact your Altair sales representative at pbssales@altair.com.



Altair PBS Professional 2021.1

Installation & Upgrade Guide

You are reading the Altair PBS Professional 2021.1

Installation & Upgrade Guide (IG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation vii

1 PBS Architecture 1

1.1 What is PBS? . . . 1
1.2 PBS Daemons . . . 1
1.3 PBS Commands . . . 4
1.4 Scheduling Jobs . . . 5

2 Pre-Installation Steps 7

2.1 Prerequisites for Running PBS . . . 7
2.2 Important Considerations . . . 12
2.3 PBS Configurations for Windows . . . 13

3 Installation 19

3.1 Overview of Installation . . . 19
3.2 Licenses . . . 19
3.3 Major Steps for Installing PBS Professional . . . 20
3.4 All Installations . . . 20
3.5 Installing via RPM on Linux Systems . . . 23
3.6 Installing via dpkg on Ubuntu . . . 37
3.7 Installing PBS on Windows Hosts . . . 37

4 Communication 45

4.1 Communication Within a PBS Complex . . . 45
4.2 Terminology . . . 45
4.3 Prerequisites . . . 45
4.4 Communication Parameters . . . 45
4.5 Inter-daemon Communication Using TPP . . . 49
4.6 Ports Used by PBS . . . 58
4.7 PBS with Multihomed Systems . . . 60

5 Initial Configuration 63

5.1 Validate the Installation . . . 63
5.2 Support PBS Features . . . 63

6 Upgrading 65
6.1 Types of Upgrades . . . 65
6.2 Differences from Previous Versions . . . 66
6.3 Caveats and Advice . . . 67
6.4 Introduction to Upgrading Under Linux . . . 70
6.5 Overlay Upgrade Under Linux . . . 70
6.6 Overlay Upgrade on One or More Machines Running Cpuset MoM . . . 82
6.7 Migration Upgrade Under Linux . . . 93
6.8 Upgrading a Windows/Linux Complex . . . 109
6.9 Upgrading from an All-Windows Complex . . . 125
6.10 After Upgrading . . . 137

7 Installing and Upgrading on Cray 139
7.1 Installing PBS with Shasta . . . 139
7.2 Prerequisites for Using Power Profiles with Cray XC . . . 139
7.3 Support for IMPS and CLE 6 and 7 . . . 139
7.4 Caveats and Advice for Installing and Upgrading on the Cray XC . . . 140
7.5 Installing PBS on the Cray XC . . . 142
7.6 After Installing on the Cray XC . . . 152
7.7 Upgrading on the Cray XC . . . 153
7.8 After Upgrading on Cray XC . . . 157
7.9 Cray XC References . . . 158

8 Starting & Stopping PBS on Linux 159
8.1 Automatic Start on Bootup . . . 159
8.2 When to Restart PBS Daemons . . . 159
8.3 Methods for Starting, Stopping, or Restarting PBS . . . 160
8.4 Starting, Stopping, and Restarting PBS Daemons . . . 163
8.5 Impact of Stop-Restart on Running Linux Jobs . . . 169

9 Starting & Stopping MoM on Windows 171
9.1 Automatic Start on Bootup . . . 171
9.2 When to Restart PBS MoMs . . . 171
9.3 Starting, Stopping, and Restarting PBS . . . 171
9.4 Stopping PBS Using the `qterm` Command . . . 174
9.5 Impact of Stop-Restart on Running Windows Jobs . . . 174

Index 177

1 PBS Architecture

1.1 What is PBS?

PBS Professional is a distributed workload management system for managing and monitoring your computational workload. PBS consists of daemons and commands that you use to manage jobs on one or more machines. You can use PBS to do tasks such as submitting, querying, altering, monitoring, moving, and deleting jobs.

1.2 PBS Daemons

You use one PBS server to manage a group of machines. The server coordinates with one or more schedulers to schedule where and when jobs run. Each machine where jobs run is managed by a MoM. Communication between server, schedulers, and MoMs is handled by one or more communication daemons. We call each instance of server, schedulers, MoMs, and communication daemons a *PBS complex*.

PBS daemons live in `PBS_EXEC/sbin`.

1.2.1 Server

The PBS server receives incoming job submissions, holds jobs that are waiting for execution, sends jobs for execution when it's their turn, and ensures that work is completed by monitoring the complex for failures and rerunning jobs when necessary. Commands communicate with the server, even if they affect other daemons. The server executable is named `pbs_server`; it is located in `$PBS_EXEC/sbin/pbs_server`.

The server contains a licensing client which communicates with the licensing server for licensing PBS jobs.

For more about the server, see "Configuring the Server and Queues" on page 17 in the PBS Professional Administrator's Guide.

1.2.2 Schedulers

PBS has a default scheduler; if you want to schedule individual partitions separately, you can add any number of additional schedulers, called *multischeds*. Each PBS scheduler follows its own scheduling policy.

Each scheduler daemon implements a policy that you define that controls when each job is run and on which resources. See "About Schedulers" on page 89 in the PBS Professional Administrator's Guide.

1.2.3 MoM

The MoM daemon places each job into execution when it receives a copy of the job from the server. MoM creates a new session that is as identical to a user login session as is possible. For example, if the user's login shell is `csh`, then MoM creates a session in which `.login` is run as well as `.cshrc`. MoM also returns the job's output to the user. One MoM runs on each computer executing PBS jobs. These computers are called *execution hosts*.

For a complete description of configuring MoM, see "Configuring MoMs and Vnodes" on page 33 in the PBS Professional Administrator's Guide.

1.2.4 Communication Daemon

The *communication daemon*, `pbs_comm`, handles communication between the other PBS daemons. For a complete description, see section 4.5, "Inter-daemon Communication Using TPP", on page 49.

1.2.5 Typical Daemon Placements

1.2.5.1 Linux Layouts

The PBS server, scheduler, and communication daemons run on a Linux host. One or more communication daemons run on other Linux hosts, if there are enough MoMs in the complex to require additional comm daemons. Typical layouts:

- A cluster of MPI-connected execution hosts where each host runs a MoM
- One or more Cray computers
- One or more HPE execution hosts, where each host is managed by a MoM and is made up of multiple blades
- Individual execution hosts on a network
- Any or all of the above

1.2.5.2 Windows Layouts

1.2.5.2.i Linux-Windows Complex

A Linux-Windows complex has a Linux server/scheduler/communication host and Windows execution and client hosts.

1.2.5.2.ii Mixed-mode Complex

A mixed-mode complex has a Linux server/scheduler/communication host, Linux execution and client hosts, and Windows execution and client hosts.

1.2.6 Single Execution System

You can install and run all PBS components on a single machine. The following illustration shows how communication works when PBS is on a single host:

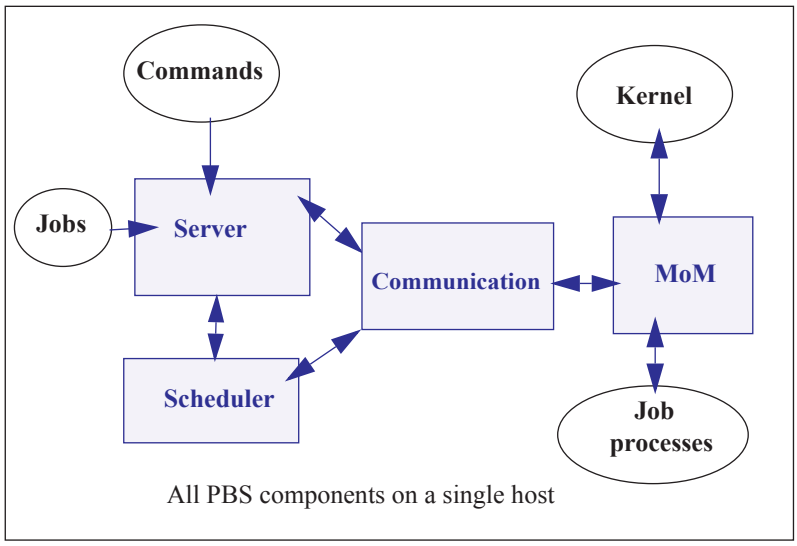


Figure 1-1:PBS daemons on a single execution host

1.2.7 Single Execution System with Front End

The PBS server and scheduler (`pbs_server` and `pbs_sched`) can run on one system and jobs can execute on another. The following illustration shows how communication works when the PBS server and scheduler are on a front-end system and MoM is on a separate host:

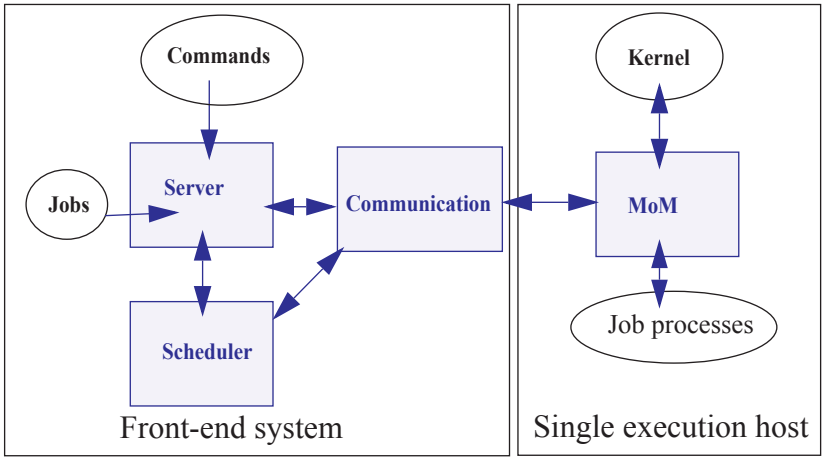


Figure 1-2:PBS daemons on single execution system with front end

1.2.8 Multiple Execution Systems

When you run PBS on several systems, the server (`pbs_server`), the scheduler (`pbs_sched`), and the communication daemon (`pbs_comm`) are installed on a front end system, and a MoM (`pbs_mom`) is installed and run on each execution host. The following diagram illustrates this:

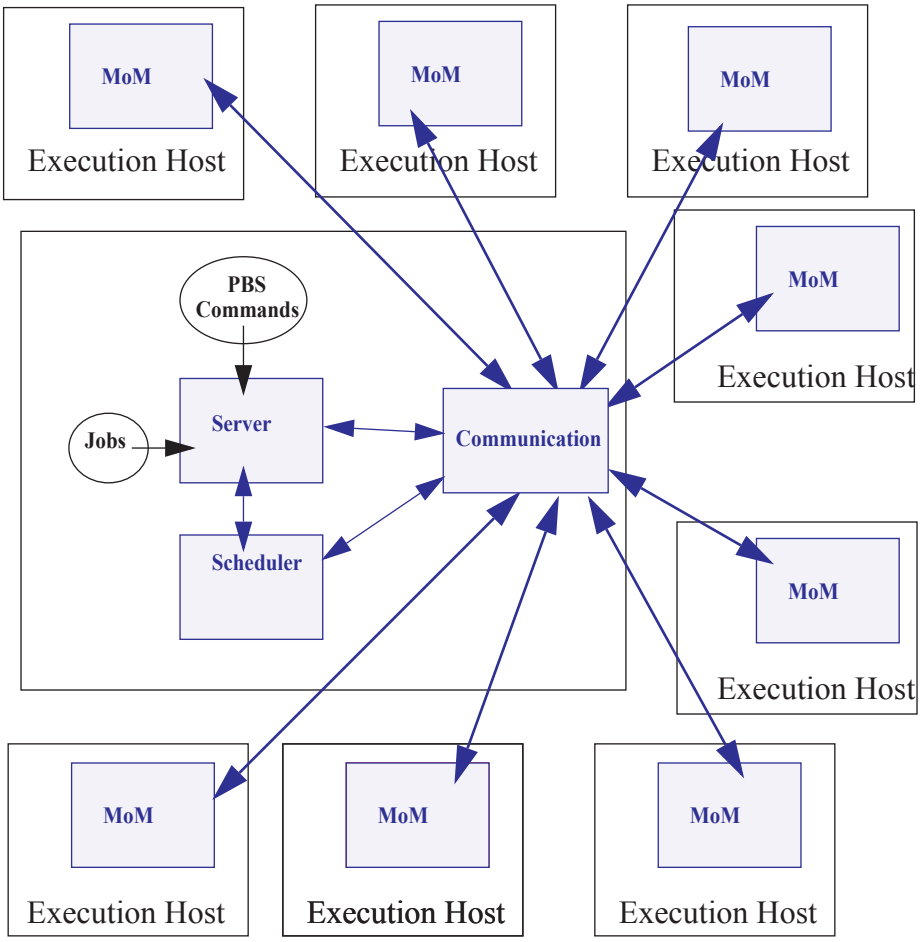


Figure 1-3:Typical PBS daemon locations for multiple execution hosts

1.3 PBS Commands

PBS supplies command-line client commands that are used to submit, monitor, modify, and delete jobs. These *client commands* can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

The privilege required to run each command varies with that command; see each command's description. PBS commands are described in "PBS Commands" on page 21 of the PBS Professional Reference Guide.

1.4 Scheduling Jobs

PBS runs jobs only on the execution hosts in the complex (hosts running a MoM). Each job is placed on a host or hosts according to the job's request. The scheduler matches jobs with available resources such as CPUs, memory, required software, licenses, etc. The scheduler follows rules for selecting hosts and parts of hosts that match each job's request. Once the scheduler finds the resources that match a job's request, it allocates hosts or parts of hosts to the job, according to how the host is configured and what the job requested.

Each task from a job can be placed on a different host, or a different part of a host. Alternatively, all tasks can be run on a single host. The job can request exclusive use of each host or part of a host, or shared use with other jobs. For details, see "Specifying Job Placement", on page 64 of the PBS Professional User's Guide.

Each scheduler can be configured so that it follows its own scheduling policy. Scheduling policy dictates which jobs are allowed to run where, who can use how much of what, etc. See "Scheduling" on page 53 in the PBS Professional Administrator's Guide.

2
Pre-Installation Steps

This chapter describes the steps to take before installing PBS. Make sure that your setup meets the requirements described here, and that you take the required steps to prepare for installing PBS.

2.1 Prerequisites for Running PBS

2.1.1 Run Same Version Within Complex

Do not mix different versions of PBS within a PBS complex. All machines using a particular PBS server (all machines in the same PBS complex) must run the exact same version of PBS, except for platform differences. Do not mix major, minor, or patch versions for any element of PBS such as daemons or commands. For example, do not run 2020.1.1 and 2020.1.2 in the same complex.

Do not mix different versions of PBS across PBS complexes, either.

2.1.2 Resources Required by PBS

The amount of memory required by the PBS server and scheduler depends on the number of hosts and the number of jobs to be queued or running. You will need less than 512 bytes per host. The number of jobs is the important factor, since each job needs about 10 KB at server startup and 5 KB when the server is running. The number of processors in the complex is not a factor.

2.1.2.1 Memory Required By Server Running Hooks

A PBS server executing hook scripts can consume a larger amount of memory than one not executing hook scripts. For example, a system consisting of a server and a MoM on a Linux machine handling 10,000 short-running jobs being submitted, modified, and moved causing execution of qsub, qalter, and movejob hooks will use around 40 MB of memory in a span of 24 hours.

2.1.2.2 Memory Required for Job History

Enabling job history requires additional memory for the server. When the server is keeping job history, it needs 8k-12k of memory per job, instead of the 5k it needs without job history. Make sure you have enough memory: multiply the number of jobs being tracked by this much memory. For example, if you are starting 100 jobs per day, and tracking history for two weeks, you're tracking 1400 jobs at a time. On average, this will require 14.3M of memory.

If the server is shut down abruptly, there is no loss of job information. However, the server will require longer to start up when keeping job history, because it must read in more information.

2.1.2.3 Amount of Memory in Complex

If the sum of all memory on all vnodes in a PBS complex is greater than 2 terabytes, then the server (`pbs_server`) and scheduler (`pbs_sched`) must be run on a 64-bit architecture host, using a 64-bit binary.

2.1.2.4 Adequate Space for Logfiles

PBS logging can fill up a filesystem. For customers running a large number of array jobs, we recommend that the filesystem where `$PBS_HOME` is located has at least 2 GB of free space for log files. It may also be necessary to rotate and archive log files frequently to ensure that adequate space remains available. (A typical PBS Professional complex will generate about 2 GB of log files for every 1,000,000 subjobs and/or jobs.)

2.1.2.5 Installation Disk Space

Make sure you have adequate disk space to install PBS. It is recommended to have at least 350 MB available, for installation alone.

2.1.2.6 Disk and Memory for Communication Daemon

By default, the communication daemon is installed on the server host.

Disk space used by the communication daemon is only for logfiles; make sure that your logging does not fill up the disk.

On any host running a communication daemon handling up to 5000 MoMs, make sure you have 500MB to 1GB of memory for the daemon.

2.1.2.7 Memory for Data Store

The data store itself requires around 100MB, but its size depends on the amount of memory required to store each job script. The total memory required is the size of all job scripts plus 100MB.

2.1.3 Name Resolution and Network Configuration

Do NOT skip this section. PBS cannot function if your hostname resolution or network is configured incorrectly.

2.1.3.1 Firewalls

PBS needs to be able to use any port for outgoing connections, but only specific ports for incoming connections. If you have firewalls running on the server or execution hosts, be sure to allow incoming connections on the appropriate ports for each host. By default, the PBS server and MoM daemons use ports 15001 through 15004 for incoming connections, the PBS communication daemon listens on port 17001, and daemons use any port below 1024 for outgoing connections. See section 4.6, "Ports Used by PBS", on page 58 for a list of ports.

Firewall-based issues are often associated with server-MoM communication failures and messages such as 'premature end of message' in the log files.

To allow interactive jobs, make sure that the ephemeral port range in your firewall is open (make sure that MoMs can connect to an ephemeral port on submission hosts). Check your OS documentation for the correct range.

2.1.3.2 Network Tuning

Depending on your network, you may need to tune kernel settings or other configuration parameters. Make sure that your kernel settings support PBS. For example, check your IP tuning parameters, including UDP and TCP, and check your ARP, routing, and name resolution settings.

2.1.3.3 Planning for Number of Machines Connected to Complex

Configure your server host with sufficient ARP cache entries in order to allow at least one connection per ethernet address that will connect to the server or to which the server will connect. This includes execution hosts, client hosts, peered servers, storage machines, or machines where the scheduler may execute scripts. Check your ARP table tuning settings.

2.1.3.4 Required Name Resolution

Make sure that the following are true:

- Use only one canonical name per host. The canonical name must be unambiguous.
- On the server/scheduler/communication host, the short name must resolve to the correct IP address.
- On the server/scheduler/communication host, the IP address must reverse resolve to the canonical name.
- Make sure that different resolvers cannot disagree when resolving the server host, whether you are using `/etc/hosts`, DNS, LDAP, NIS, or something else.
- Every MoM must resolve each MoM to the same IP address that the server recognizes for that MoM. So if the server recognizes MoM A at IP address w.x.y.z, all other MoMs must resolve MoM A to w.x.y.z.
- Make sure that the IP address of each machine in the complex resolves to the fully qualified domain name for that machine, and vice versa. Forward and reverse hostname resolution must work consistently between all machines.
- The server must be able to look up the IP addresses for any execution host, any client host, and itself.
- Make sure that forward and reverse name lookup operate according to the IETF standard. The network on which you will be deploying PBS must be configured according to IETF standards.

2.1.3.5 Required Network Configuration

- PBS can use a static address mapping only.
- Communications between daemons must be robust and must have sufficient capacity. Make sure that your network does not present any limitations to PBS. For example, the ARP table size limit must not interfere when you have a large number of MoMs. Configure your server with sufficient ARP cache entries to allow at least one connection per ethernet address that will connect to the server or to which the server will connect. This includes execution hosts, client hosts, peered servers, storage machines, or machines where the scheduler may execute scripts. See section 2.1.3.1, "Firewalls", on page 8.

2.1.3.6 Recommendations for Name Resolution and Network Configuration

- Test name resolution using the `ping` command.
- Test the connections between server and MoM daemons on every physical network. You should test TCP and UDP, and make sure that the connection can handle large packets. You can use a tool such as `ttcp`, with packets of size16k, for testing.
- For multihomed MoMs, keep all PBS traffic on the same control network or subnet.
- Keep different types of traffic on separate interfaces to reduce jitter.
- When configuring `/etc/hosts`, do the following:
  - Use the server's FQDN as the first item on the first line on the PBS-to-PBS interface
  - Use different FQDNs as the first item on other lines
  - Use a name on only one line
- If you want redundancy in your network interface, consider using bonding. Aside from presenting a transparent interface, this can allow you to load-balance network traffic across different networks.
- If name resolution is a problem in a network that should be working, tell `nscd` not to cache the host name of the machine with the problem.
- If you are using `nscd` and you change an IP address or hostname, restart `nscd` on all hosts.

2.1.3.6.i Recommendations for Name Resolution and Network Configuration on Windows

- On Windows, make sure the first nameserver resolves all the needed hostnames, including the server hostname and the domain controller host for active directory queries.
- On Windows, put explicit IP-to-hostname addresses in the `C:\windows\system32\drivers\etc\hosts` file. Otherwise your site will experience extreme slowdowns. If you make these changes to a running PBS complex, you must then restart all the PBS daemons (services).

2.1.3.7 Order of Operations for Name Resolution and Network Configuration

You can take care of some of the name resolution testing before you install PBS. However, you must do some testing using the `pbs_hostn` command, after you install PBS. The "Initial Configuration" chapter follows the "Installation" chapter, and includes steps to test name resolution. We include an overview of the whole process here for clarity:

1. Set up firewall
2. Set up name resolution
3. Test name resolution by using `ping` command; if necessary, fix & re-test
4. Install PBS
5. Test name resolution by using `pbs_hostn` command.
6. If name resolution does not work correctly:
   a. Uninstall PBS
   b. Fix name resolution
   c. Install PBS
   d. Test using `pbs_hostn`

2.1.3.8 Server Hostname

The PBS_SERVER entry in `pbs.conf` cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER.

2.1.3.9 Sockets

Some PBS processes cause network sockets to be opened between submission and execution hosts. For more information about these processes, see "Sockets and Checkpointing" on page 431 in the PBS Professional Administrator's Guide. Make sure your network and firewalls are set up to handle sockets correctly.

2.1.3.10 Mounting NFS File Systems

Asynchronous writes to an NFS server can cause reliability problems. If using an NFS file system, mount the NFS file system synchronously (without caching.)

2.1.3.11 Making Ports Available

The ports used by the PBS daemons must be available during the installation. See section 4.6, "Ports Used by PBS", on page 58.

2.1.4 HPE Prerequisites

2.1.4.1 HPE MPI Recommendation

For HPE MC990X, HPE Superdome Flex, and HPE 8600 machines, we recommend using HPE MPI.

As of PBS version 2020.1, `pbs_mom.cpuset` is no longer available. Instead, use standard MoM, and use the cgroups hook to manage cgroups.

2.1.4.2 Power File Requirement

When using PBS Power Provisioning on HPE, ensure that the following file exists:

```
/opt/clmgr/power-service
```

2.1.5 License Server Requirement

Make sure that the ALM license server is at version 14.5 before installing PBS.

2.1.6 System Clocks in Sync

We recommend that clocks on all participating systems be in sync.

2.1.7 User Requirements on Linux

2.1.7.1 User Accounts

Users who will submit jobs must have accounts at the server and at each execution host.

2.1.7.2 Linux User Authorization

When the user submits a job from a system other than the one on which the PBS server is running, system-level user authorization is required. This authorization is needed for submitting the job and for PBS to return output files (see also "Managing Output and Error Files", on page 39 of the PBS Professional User's Guide and "Input/Output File Staging", on page 31 of the PBS Professional User's Guide).

The username under which the job is to be executed is selected according to the rules listed under the "-u" option to qsub. The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not).

Such authorization is provided by any of the following methods:

1. The host on which qsub is run (i.e. the submission host) is trusted by the server. This permission may be granted at the system level by having the submission host as one of the entries in the server's hosts.equiv file naming the submission host. For file delivery and file staging, the host representing the source of the file must be in the receiving host's hosts.equiv file. Such entries require system administrator access.
2. The host on which qsub is run (i.e. the submission host) is explicitly trusted by the server via the user's .rhosts file in his/her home directory. The .rhosts must contain an entry for the system from which the job is submitted, with the user name portion set to the name under which the job will run. For file delivery and file staging, the host representing the source of the file must be in the user's .rhosts file on the receiving host. It is recommended to have two lines per host, one with just the "base" host name and one with the full hostname, e.g.: *host.domain.name*.
3. PBS may be configured to use the Secure Copy (scp) for file transfers. The administrator sets up SSH keys as described in "Enabling Passwordless Authentication" on page 565 in the PBS Professional Administrator's Guide. See also "Setting File Transfer Mechanism" on page 559 in the PBS Professional Administrator's Guide.
4. User authentication may also be enabled by setting the server's flatuid attribute to *True*. See the pbs_server_attributes(7B) man page and "Flatuid and Access" on page 376 in the PBS Professional Administrator's Guide. Note that flatuid may open a security hole in the case where a vnode has been logged into by someone impersonating a genuine user.

2.2 Important Considerations

2.2.1 Avoiding Datastore Corruption from Job Spool Files

Job spool files can fill up the PBS_HOME filesystem. This can corrupt the datastore and cause a failure that requires recovering from backups. Consider moving the spool directory to a dedicated file system, or using quotas. Note that if you use quotas on Cray ext3fs filesystems, the kernel enforces quotas correctly even though commands that display quotas may show stale data (when files are not closed.)

Job spool files are saved on the server on job rerun, and on the MoM for running jobs.

2.2.2 Using noexec on /tmp

If you need to have noexec on your /tmp, do one of the following:

- Set the TMPDIR environment variable; the shared library that is extracted to /tmp/xf-dll follows TMPDIR if it is set
- Install a soft link from /tmp/xf-dll pointing to a location on a filesystem that does not have the "noexec" mount flag

Why? The ALSDK `liblmx-altair.so` self-extracts a DSO into `/tmp/xf-dll`, and then tries to map it. If it fails to do so because `noexec` is set, the ALSDK routines simply perform an `exit(1)`,which terminates the server, without any log message in the server log.

2.3 PBS Configurations for Windows

2.3.1 Definitions

Active Directory

Active Directory is an implementation of LDAP directory services by Microsoft to use in Windows environments. It is a directory service used to store information about the network resources (e.g. user accounts and groups) across a domain. Active Directory is fully integrated with DNS and TCP/IP; DNS is required. To be fully functional, the DNS server must support SRV resource records or service records.

Admin (Windows)

As referred to in various parts of this document, this is a user logged in from an account who is a member of any group that has full control over the local computer, domain controller, or is allowed to make domain and schema changes to the Active directory.

Administrators

A group that has built-in capabilities that give its members full control over the local system, or the domain controller host itself.

Delegation

A capability provided by Active Directory that allows granular assignment of privileges to a domain account or group. So for instance, instead of adding an account to the "Account Operators" group which might give too much access, then delegation allows giving the account read access only to all domain users and groups information. This is done via the Delegation wizard.

Domain Admin Account

This is a domain account on Windows that is a member of the "Domain Admins" group.

Domain Admins

A global group whose members are authorized to administer the domain. By default, the Domain Admins group is a member of the Administrators group on all computers that have joined a domain, including the domain controllers.

Domain User Account

It is a domain account on Windows that is a member of the "Domain Users" group.

Domain Users

A global group that, by default, includes all user accounts in a domain. When you create a user account in a domain, it is added to this group automatically.

Enterprise Admins

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make forest-wide changes in Active Directory, such as adding child domains.

Install Account, Installation Account

The account used by the person who installs PBS.

Schema Admins

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make schema changes in Active Directory.

PBS service account

The account that is used to execute `pbs_mom` via the Service Control Manager on Windows. This account can have any name. The default name is *pbsadmin*.

2.3.2 Domained Environment Required

All Windows hosts and users must be in a domained environment.

2.3.3 Permission Requirement

On Windows 7 and later with UAC enabled, if you will use the `cmd` prompt to operate on hooks, or for any privileged command such as `qmgr`, you must run the `cmd` prompt with option *Run as Administrator*.

2.3.4 Daemon Layout for Windows

As of PBS 19.4.1, all PBS complexes run the PBS server, scheduler, and comm daemons on Linux hosts. You can run all MoMs and client commands on Windows hosts, or some on Windows and some on Linux.

2.3.5 Windows Configuration in a Domained Environment

2.3.5.1 Machines

- Any Windows client commands and MoMs must run on a set of Windows machines networked in a single domain.
- The machines must be members of this one domain, and they must be dependent on a centralized database located on the primary/secondary domain controllers.
- The domain controllers must be running on a Server type of Windows host, using Active Directory configured in "native" mode.
- The choice of DNS must be compatible with Active Directory.
- The PBS server and scheduler run on a Linux host.
- PBS must not be installed or run on a Windows machine that is serving as the domain controller (running Active Directory) to the PBS hosts.

2.3.5.2 User Accounts

- Windows job submitters must have an account at all PBS hosts involved in a job: the server, the execution hosts, and the client host.
- All user accounts must be in the same domain as the Windows client and execution hosts.
- Each user must explicitly be assigned a HomeDirectory sitting on some network path. PBS does not support a HomeDirectory that is not network-mounted. PBS currently supports network-mounted directories that are using the Windows network share facility.
- If a user was not assigned a HomeDirectory, then PBS uses `PROFILE_PATH\My Documents\PBS Pro`, where `PROFILE_PATH` could be, for example, "`\Documents and Settings\`*`username`*".

2.3.5.3 User Jobs

- All users must submit and run PBS jobs using only their domain accounts (no local accounts), and domain groups. If a user has both a domain account and local account, then PBS will ensure that the job runs under the domain account.
- Each user must always supply an initial password in order to submit jobs. This is done by running the pbs_login command at least once to supply the password that PBS will use to run the user's jobs.
- Access by jobs to network resources, such as a network drive, requires a password.
- All job scripts, as well as input, output, error, and intermediate files of a PBS job must reside in an NTFS directory.

2.3.6 User Authorization Under Windows

Windows job submitters must cache a password for authorization. To do this, each job submitter must run pbs_login at each client host initially and for each password change.

The user name under which the job is to be executed is selected according to the rules listed under the "`-u`" option to `qsub`. See "qsub" on page 213 of the PBS Professional Reference Guide. The user submitting the job must be authorized to run the job under the execution user name (whether explicitly specified or not). Authorization is provided by either of the following methods:

2.3.6.1 Requirements for Non-admin Users

Under Windows, if a user has a non-admin account, the server `hosts.equiv` file is used to determine whether that user can run a job at a given server.

The Windows `hosts.equiv` file determines the list of non-Administrator accounts that are allowed access to the local host, that is, the host containing this file. This file also determines whether a remote user is allowed to submit jobs to the local PBS server, with the user on the local host being a non-Administrator account.

This file is usually: `%WINDIR%\system32\drivers\etc\hosts.equiv`.

The format of the `hosts.equiv` file is as follows:

[+|-] hostname username

'+' means enable access, whereas '`-`' means to disable access. If '+' or '`-`' is not specified, then this implies enabling of access. If only *hostname* is given, then users logged into that host are allowed access to like-named accounts on the local host. If only *username* is given, then that user has access to all accounts (except Administrator-type users) on the local host. Finally, if both *hostname* and *username* are given, then user at that host has access to like-named account on local host.

The `hosts.equiv` file must be owned by an admin-type user or group, with write access granted to an admin-type user or group.

Table 2-1: Requirements for Non-admin User to Submit Job

File	Submission Host Username vs. Server Host Username	
	UserS Same as UserS	UserS Different from UserA
`hosts.equiv` on ServerA	<HostS>	<HostS> UserS

2.3.6.2 Requirements for Admin Users

For an admin account, `[PROFILE_PATH].\rhosts` is used, and the server's acl_roots attribute must be set to allow job submissions.

Table 2-2: Requirements for Admin User to Submit Job

Location/Action	Submission Host Username vs. Server Host Username	
	UserS Same as UserS	UserS Different from UserA
`[PROFILE_PATH]\.rhosts contains`	For UserS on ServerA, add <HostS> UserS	For UserA on ServerA, add <HostS> UserS
set ServerA's acl_roots attribute	`qmgr> set server acl_roots=UserS`	`qmgr> set server acl_roots=UserA`

2.3.7 Windows User HOMEDIR

Each Windows user must have a home directory (`HOMEDIR`) where their PBS job will initially be started. For jobs that do not have their staging and execution directories created by PBS, the home directory is also the starting location of file transfers when users specify relative path arguments to `qsub/qalter -W stagein/stageout` options.

PBS supports network mounted home directories.

2.3.7.1 Configuring User HOMEDIR

The home directory can be configured by an Administrator by setting the user's HomeDirectory field in the user database, via the User Management Tool. It is important to include the drive letter when specifying the home directory path. The directory specified for the home folder must be accessible to the user. If the directory has incorrect permissions, PBS will be unable to run jobs for the user.

2.3.7.2 Directory Must Exist Already

You must specify an already existing directory for home folder. If you don't, the system will create it for you, but set the permissions to that which will make it inaccessible to the user.

2.3.7.3 Default Directory

If a user has not been explicitly assigned a home directory, then PBS will use this Windows-assigned default, local home directory as base location for its default home directory. More specifically, the actual home path will be:

```
[PROFILE_PATH]\My Documents\PBS Pro
```

For instance, if a *userA* has not been assigned a home directory, it will default to a local home directory of:

```
\Documents and Settings\userA\My Documents\PBS Pro
```

UserA's job will use the above path as working directory, and for jobs that do not have their staging and execution directories created by PBS, any relative pathnames in stagein, stageout, output, error file delivery will resolve to the above path.

Note that Windows can return as PROFILE_PATH one of the following forms:

\Documents and Settings*username*

\Documents and Settings*username*.*local-hostname*

\Documents and Settings*username*.*local-hostname*.00*N* where *N* is a number

\Documents and Settings*username*.*domain-name*

2.3.8 Windows Caveats

2.3.8.1 Installation of Microsoft Redistributable Pack

The PBS installer installs the Microsoft redistributable pack of vc++ redistributable binaries into the system root (C:\Windows) directory.

2.3.8.2 Make Sure ComSpec Environment Variable Is Set

Check that in the pbs_environment file, the environment variable ComSpec is set to *C:\WINDOWS\system32\cmd.exe*. If it is not, set it to that value:

1. Change directory:

   cmd.admin> cd \Program Files\PBS\home

2. Edit the pbs_environment file:

   cmd.admin> edit pbs_environment

3. Add the following entry to the pbs_environment file:

   ComSpec=C:\WINDOWS\system32\cmd.exe

4. Restart the MoM:

   net stop pbs_mom

   net start pbs_mom

Simply setting this variable inside a job script doesn't work. The ComSpec variable must be set before PBS executes cmd. cmd invokes the user's submission script.

2.3.8.3 Unsupported Windows Configurations

The following Windows configurations are currently unsupported:

- Using NIS/NIS+ for authentication on non-domain accounts.
- Using RSA SecurID module with Windows logons as a means of authenticating non-domain accounts.

3 Installation

3.1 Overview of Installation

3.1.1 Prerequisite Reading

This chapter shows how to install PBS Professional. You should read the Release Notes and Chapter 2, "Pre-Installation Steps", on page 7 before installing the software.

3.1.2 Replacing an Older Version of PBS

If you are installing on a system where PBS is already running, follow the instructions for an upgrade. Go to Chapter 6, "Upgrading", on page 65.

3.1.3 Package Naming

Download the package for your platform from our website, and uncompress it. Packages are named like this:

```
PBSPro_<version>-<platform>_<hardware>.tar.gz.
```

For example, the PBS 19.2.2 package for CentOS 7 is named `PBSPro_19.2.2-CentoOS7.tar.gz`. When you uncompress it, you'll find the following sub-package RPMs:

- Server/scheduler/MoM/communication/commands:
  `pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm`
- MoM/commands:
  `pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm`
- Commands:
  `pbspro-client-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm`

For example, for CentOS 7, the sub-packages are:

```
pbspro-server-19.2.2-<date etc.>-0.el7.x86_64.rpm
pbspro-execution-19.2.2-<date etc.>-0.el7.x86_64.rpm
pbspro-client-19.2.2-<date etc.>-0.el7.x86_64.rpm
```

3.2 Licenses

In order for a job to run, it must be running on a licensed host. Make sure that you have access to an Altair License Manager (ALM) license server that is hosting the licenses you need. Your license server can host either of these:

- Node licenses, which license a certain amount of hardware. Node licenses are obtained from Altair.
- Socket licenses, which are tied to hosts.

Each PBS complex can be licensed using PBSProNodes licenses or PBSProSockets licenses, but not both, so the ALM license server will provide one or the other. See the *PBS Works Licensing Guide*.

3.2.1 Licensing Caveats

If you do not tell PBS where to find the license server, the pbs_license_info attribute is left as is, which could be set to some previous value or unset. It is usually set to some previous value when doing an overlay or migration upgrade.

If the license server location is incorrectly initialized (e.g. the hostname or port number is incorrect), PBS may not be able to pinpoint the misconfiguration as the cause of the failure to reach a license server. The PBS server's first attempt to contact the license server results in the following message on the server's log file:

```
"unable to connect to license server at ..."
```

3.3 Major Steps for Installing PBS Professional

1. Set up your ALM license server with enough licenses for your site. See the *PBS Works Licensing Guide*.
2. Create accounts used by PBS. See section 3.5.1.3, "Create PBS Data Service Management Account", on page 23 and section 3.7.8, "Create Installation and Service Accounts", on page 39.
3. Download the correct PBS Professional package for each host. The PBS Professional package is available on the PBS download page at `https://secure.altair.com/UserArea/`.
4. Please read section 3.4, "All Installations", on page 20. Then install PBS Professional on the server host and all execution hosts, without starting any daemons. For instructions, see section 3.5, "Installing via RPM on Linux Systems", on page 23 or section 3.7, "Installing PBS on Windows Hosts", on page 37.
5. Optionally, install additional communication daemons.
6. If you have additional communication daemons, start them using `systemd` or the PBS start/stop script. See section 8.3, "Methods for Starting, Stopping, or Restarting PBS", on page 160.
7. Install PBS commands on any client hosts.
8. Start PBS on each execution host using `systemd` or the PBS start/stop script. See section 8.3, "Methods for Starting, Stopping, or Restarting PBS", on page 160.
9. Start PBS on the server host using `systemd` or the PBS start/stop script. See section 8.3, "Methods for Starting, Stopping, or Restarting PBS", on page 160.
10. Set the server's pbs_license_info attribute to point to the license server:

    ```
    # qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
    ```

11. Using the `qmgr` command, define the vnodes that the server will manage. See "Creating Vnodes" on page 38 in the PBS Professional Administrator's Guide.
12. Perform post-installation tasks such as validation. See Chapter 5, "Initial Configuration", on page 63.

3.4 All Installations

3.4.1 Automatic Installation of Database

Installing PBS automatically installs (and upgrades) the database used by PBS for its data store.

3.4.2 Choosing Installation Sub-package

On each PBS host, install the sub-package corresponding to the task(s) that host will perform. The task you give a host determines what we call the host. For example, a host that runs job tasks is called an "execution host". Sometimes there is more than one title that means the same thing; for example, some people call the server host the "headnode". Select the sub-package (or, for Windows, the installation option) that matches the desired task:

Table 3-1: Choosing Installation Type

Option	Host Role	Task	Package Contents	Parameters in pbs.conf For Default Start
1	Server host, headnode, front end machine	Runs server, scheduler, and communication daemons. Optionally runs MoM daemon. Client commands are included. If using failover, install on both server hosts.	Server/scheduler/communication/MoM/client commands	PBS_START_SERVER=1 PBS_START_SCHED=1 PBS_START_COMM=1 To run MoM, add: PBS_START_MOM =1
2	Execution host, MoM host	Runs MoM. Executes job tasks. Client commands are included. Install on each execution host.	Execution/client commands	PBS_START_MOM =1
3	Client host, submit host, submission host	Users can run PBS commands and view man pages. Install on each client host.	Client commands	None

3.4.2.1 Pathname Conventions

The term *PBS_HOME* refers to the location where the daemon/service configuration files, accounting logs, etc. are installed.

The term *PBS_EXEC* refers to the location where the executable programs are installed.

3.4.3 Installing Additional Communication Daemons

By default, one communication daemon is installed on each server host. If you are configuring failover, your site will automatically have two communication daemons and all PBS daemons will automatically connect to them.

You may want to install additional communication daemons. For some rough guidelines on when you might want additional communication daemons, see section 4.5.4, "Recommendations for Maximizing Communication Performance", on page 51.

To install just the communication daemon:

1. Download the appropriate PBS package
2. Uncompress the package
3. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
4. Install the server sub-package:

   ```
   rpm -i
       <path/to/sub-package>pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

5. Edit pbs.conf to run only the communication daemon:

   ```
   PBS_START_COMM=1
   PBS_START_MOM=0
   PBS_START_SCHED=0
   PBS_START_SERVER=0
   ```

6. Start PBS:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to script>/pbs start
   ```

7. Check to see that the communication daemon is running:

   ```
   ps -ef | grep pbs
   ```

   You should see that the pbs_comm daemon is running.

3.4.4 Deciding to Run a MoM After Installation

When you initially start PBS on a host that is configured not to run a MoM, PBS does not create MoM's home directory. If you later decide to run a MoM on this host:

1. Edit pbs.conf on that host and set PBS_START_MOM=1
2. You may find it helpful to source your /etc/pbs.conf file.
3. Run the pbs_habitat script:

   ```
   $PBS_EXEC/libexec/pbs_habitat
   ```

4. Start PBS on the host:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to start/stop script>/pbs start
   ```

3.4.5 Installation Method and Instructions by Platform

The procedure for installing PBS is the same on most platforms. Some platforms have a few minor differences, and some require special instructions. The following table lists instructions by platform:

Table 3-2: Installation Method and Instructions by Platform

Platform	Installation Method	Installation Instructions
RHEL	package manager, e.g. RPM	section 3.5, "Installing via RPM on Linux Systems", on page 23
CentOS	package manager, e.g. RPM	section 3.5, "Installing via RPM on Linux Systems", on page 23
HPE MC990X, Superdome Flex	package manager, e.g. RPM	section 3.5.3, "Installing on MC990X or Superdome Flex", on page 28
HPE 8600	package manager, e.g. RPM	section 3.5.4, "Installing PBS on the HPE 8600", on page 30
SuSE	package manager, e.g. RPM	section 3.5, "Installing via RPM on Linux Systems", on page 23
CLE	RPM	"Installing and Upgrading on Cray" on page 139
Ubuntu	deb	section 3.6, "Installing via dpkg on Ubuntu", on page 37
Windows	PBS installation program provided by Altair	section 3.7, "Installing PBS on Windows Hosts", on page 37

3.5 Installing via RPM on Linux Systems

3.5.1 Prerequisites for Installing on Linux Systems

3.5.1.1 Prerequisite Reading

Please do not jump straight to this section in your reading. Before downloading and installing PBS, please make sure that you have read the following and taken any required steps:

- Prerequisites: All of Section 2.1, "Prerequisites for Running PBS", and Section 2.1.7, "User Requirements on Linux" and their subsections.
- Please read Section 3.1, "Overview of Installation".
- Make sure that you know how you will proceed by reading Section 3.3, "Major Steps for Installing PBS Professional".
- Please check all of Section 3.4, "All Installations" and its subsections to make sure you have prepared properly.

3.5.1.2 Permissions

The location for the installation of the PBS Professional software binaries (`PBS_EXEC`) and private directories (`PBS_HOME`) must be owned and writable by root, and must not be writable by other users.

3.5.1.3 Create PBS Data Service Management Account

Before you install PBS, you must create the PBS data service management account.

Note that there are two accounts related to the data service. Both have the same account name, but one is a Linux account and one is internal to the data service:

PBS data service management account

Created by administrator. Linux account with a Linux system password.

Data service account

Created by PBS on installation. Account that is internal to the data service, with its own data service password. Used by PBS to log into and do operations on the data service. PBS maps this account to the PBS data service management account. Must have same name as PBS data service management account.

Create the PBS data service management account with the following characteristics:

- Non-root account
- Account must be for a system user; the UID must be less than 1000. Otherwise, the data service may be killed at inopportune times.
- Account is enabled
- If you are using failover, the UID of this account must be the same on both primary and secondary server hosts
- We recommend that the account is called *pbsdata*.
  - The installer looks for an account called *pbsdata*. If this account exists, the installer does not need to prompt for a username, and can install silently.
  - If you choose to use an account named something other than *pbsdata,* make sure you export an environment variable named PBS_DATA_SERVICE_USER with the value set to the desired existing PBS data service management account name.
- Root must be able to `su` to the PBS data service management account and run commands as that user. Do not add lines such as '`exec bash`' to the `.profile` of the PBS data service management account. If you want to use `bash` or similar, set this in the `/etc/passwd` file, via the OS tools for user management.
- The PBS data service management account must have a home directory.
- Do not put a CPU time limit on the data service Linux account. If you do, the datastore will die and kill the server.

3.5.1.4 Unset PBS_EXEC Environment Variable

Unset the PBS_EXEC environment variable.

3.5.2 Generic Installation on Linux

For all platforms except those listed here, follow the generic instructions. The following platforms require their own steps:

- HPE MC990X and Superdome Flex: Go to section 3.5.3, "Installing on MC990X or Superdome Flex", on page 28
- HPE 8600: Go to section 3.5.4, "Installing PBS on the HPE 8600", on page 30

3.5.2.1 Downloading PBS

1. Download the PBS `tar.gz` package
2. Extract the `tar` file. For example:

```
tar zxvf PBSPro_<version>-linux26_i686.tar.gz
```

3.5.2.2 Setting Installation Parameters

Make sure that the PBS_EXEC, PBS_HOME, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER parameters are specified at install time. PBS has default locations for PBS_EXEC and PBS_HOME, and a default value for PBS_DATA_SERVICE_USER, but you must specify the others.

You can override defaults at install time, in this order of precedence:

1. Via arguments to the package manager
2. Via environment variables
3. By specifying the desired parameters in `/etc/pbs.conf`. For details see "The PBS Configuration File" on page 539 in the PBS Professional Administrator's Guide

This table lists each parameter, its default value, and how it can be set at install time:

Table 3-3: Setting Installation Parameters

Parameter	Default Value	Specify via `pbs.conf`	Specify via Environment Variable	Specify via `rpm` Command
PBS_DATA_SERVICE_USER	*pbsdata*	No - ignored	Yes - environment variable only	No
PBS_EXEC	*/opt/pbs*	No - value in `pbs.conf` is overridden at install time. Note that changing this in `pbs.conf` breaks `rpm`	No - ignored	`--prefix <location>`
PBS_HOME	*/var/spool/pbs*	Yes	Yes	No
PBS_LICENSE_INFO	None	No - ignored	Yes - environment variable only. Can set pbs_license_info server attribute via `qmgr`	No
PBS_SERVER	For server installation: output of `hostname` command up to first period. For all other installations: "*CHANGE_THIS_TO_PBS_PRO_SERVER_HOSTNAME*"	Yes	Yes	No

3.5.2.2.i Caveats for Installation Parameters

Any PBS_START_* parameters set in the environment are not picked up and set in `pbs.conf`. You must specify these in `pbs.conf`; do not export them.

3.5.2.3 Installing on a Standalone Linux Machine

Make sure that you have covered the prerequisites in section 3.5.1, "Prerequisites for Installing on Linux Systems", on page 23. The following example shows an installation on a single host on which all PBS components will run, and from which users will also submit jobs. The process may vary depending on the native package installer on your system.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
5. Install the server sub-package:

   ```
   rpm -i
       <path/to/sub-package>pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

6. Edit pbs.conf to set PBS_START_MOM=1
7. Start PBS:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to script>/pbs start
   ```

8. Check to see that the server, scheduler, MoM, and communication daemons are running:

   ```
   ps -ef | grep pbs
   ```

   You should see that the following daemons are running: pbs_mom, pbs_server, pbs_sched, pbs_comm
9. Make sure that user paths work, and submit sleep jobs. See section 3.5.5, "Making User Paths Work", on page 36.
10. Verify that the jobs are running:

    ```
    /opt/pbs/bin/qstat -a
    ```

11. Verify that you are running the correct version:

    ```
    /opt/pbs/bin/qstat --version
    ```

12. Set the pbs_license_info server attribute to the location of the license server:

    ```
    # qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
    ```

3.5.2.4 Installing on a Linux Cluster

Make sure that you have covered the prerequisites in section 3.5.1, "Prerequisites for Installing on Linux Systems", on page 23.

You may or may not want to run batch jobs on the server/scheduler/communication host. First, install and start PBS on each execution host. Then install PBS on the server host. Follow these steps:

3.5.2.4.i Install PBS on Execution Hosts

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
5. Install the PBS execution sub-package on each execution host:

   ```
   rpm -i
       <path/to/sub-package>pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

6. Start PBS:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to script>/pbs start
   ```

Instead of running the installer by hand on each machine, you can use a command such as `pdsh`. The one-line format for a non-default install is:

```
PBS_SERVER=<server name> PBS_HOME=<new home location> rpm -i --prefix <new exec location>
    pbspro-<sub-package>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

3.5.2.4.ii Install PBS on Server Host

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
5. If you want to run batch jobs on the front-end host, create or edit the `pbs.conf` file on the front-end machine so that a MoM runs there:

   ```
   PBS_START_MOM=1
   ```

6. Install the server sub-package:

   ```
   rpm -i
       <path/to/sub-package>pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

3.5.2.4.iii Start PBS on Server Host

Start PBS on the server machine by running `systemd` or the PBS start/stop script. If `/etc/init.d` exists, the script is in `/etc/init.d/pbs`, otherwise `/etc/rc.d/init.d/pbs`:

```
systemctl start pbs
```

or

```
<path to script>/pbs start
```

3.5.2.4.iv Configure Licensing

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

3.5.2.4.v Install PBS on Client Hosts

Install PBS on each client host.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
5. Install the PBS client sub-package on each execution host:

```
rpm -i
    <path/to/sub-package>pbspro-client-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

3.5.2.4.vi Define Vnodes

Using the `qmgr` command, define the vnodes that the server will manage. See "Creating Vnodes" on page 38 in the PBS Professional Administrator's Guide.

3.5.2.4.vii Check User Paths

Make sure that user paths work. See section 3.5.5, "Making User Paths Work", on page 36.

3.5.3 Installing on MC990X or Superdome Flex

3.5.3.1 Prerequisites for Installing on a MC990X or Superdome Flex

Make sure that you have covered the prerequisites in section 3.5.1, "Prerequisites for Installing on Linux Systems", on page 23. On these machines, you install the PBS server package and use cgroups to manage cpusets.

3.5.3.2 Download and Install the New PBS

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
5. Install the server sub-package:

```
rpm -i
    <path/to/sub-package>pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

3.5.3.3 Start PBS

1. Edit `pbs.conf` to set `PBS_START_MOM=1`
2. Start the PBS daemons by running `systemd` or the PBS start/stop script. The location of the script varies depending on system configuration.

```
systemctl start pbs
```

or

```
<path to script>/pbs start
```

3.5.3.4 Configure Licensing

Set the pbs_license_info server attribute to the location of the license server(s):

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

3.5.3.5 Test the New PBS

1. Check to see that the PBS daemons are running. You should see that there are four daemons running: pbs_mom, pbs_server, pbs_sched, pbs_comm:

   ```
   ps -ef | grep pbs
   ```

2. Submit jobs as a normal user.

   Submit a job to the default queue:

   ```
   echo "sleep 60" | /opt/pbs/bin/qsub
   ```

3. Verify that the jobs are running:

   ```
   /opt/pbs/bin/qstat -an
   ```

3.5.3.6 Configure Cgroups to Manage Cpusets

1. Make sure that your cgroups hook is enabled and that you can use cgroups. See "Configuring and Using PBS with Cgroups" on page 571 in the PBS Professional Administrator's Guide.

1. Export the cgroups hook configuration file to pbs_cgroups.json:

   ```
   # qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
   ```

2. You can make the cgroups hook mimic the behavior of the cpuset MoM in previous versions:

   a. Create one vnode for each NUMA node. Edit pbs_cgroups.json as follows (important):

      ```
      "vnode_per_numa_node" : true,
      ```

   b. Edit pbs_cgroups.json as follows (recommended):

      ```
      "use_hyperthreads" : true,
      ```

3. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

      ```
      # mount | grep cgroup | grep memory
      ```

      If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

      ```
      "cgroup": {
        ...
      "memory": {
      "enabled": false,
      ```

4. Import the modified configuration (make sure you use "**x-config**"):

   ```
   # qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
   ```

3.5.3.7 Restart MoMs

On each execution host, restart MoM :

```
ps -eaf | grep pbs_mom
kill <MoM PID>
/opt/pbs/sbin/pbs_mom
```

3.5.4 Installing PBS on the HPE 8600

3.5.4.1 HPE 8600 Components

An 8600 system consists of one Admin node, one or more Service (login) nodes, and a set of one or more compute racks. Each compute rack consists of one or more IRU nodes and one or more compute nodes per IRU. The racks are diskless. The root file system of the IRU and compute nodes are mounted read-only from a NAS managed by the Admin node. There is a single image of the root file system for all of the compute nodes and a separate image for all of the IRU nodes. HPE Performance Cluster Manager node management commands are used to publish the image to the various nodes in a process that involves powering down the nodes, pushing a new image, and re-powering the nodes.

In a typical configuration, user home file systems are mounted from NAS, and each node has a separately mounted file system for `/var/spool`.

HPE follows a naming convention when preparing a system for shipment. Service nodes are named "service0", "service1", … Compute nodes are named "rRiLnN" where 'R' is the rack number starting with 1; 'L' is the IRU node number within a rack starting with 0 in each rack; N is the node number, starting with 0, under the specific Rack Leader. For example, two racks with 2 IRUs per rack and 4 nodes per IRU are named:

Table 3-4: Node Names

IRU	Rack 1	Rack 2
IRU 0	*r1i0n0*	*r2i0n0*
	r1i0n1	*r2i0n1*
	r1i0n2	*r2i0n2*
	r1i0n3	*r2i0n3*
IRU 1	*r1i1n0*	*r2i1n0*
	r1i1n1	*r2i1n1*
	r1i1n2	*r2i1n2*
	r1i1n3	*r2i1n3*

3.5.4.2 Requirements for the HPE 8600 with HPE MPI

- Make sure that you have covered the prerequisites in section 3.5.1, "Prerequisites for Installing on Linux Systems", on page 23.
- In order to run PBS on the HPE 8600 with HPE MPI, HPE Performance Cluster Manager node management tools must already be installed. You will be using the following HPE Performance Cluster Manager commands:

Table 3-5: Performance Cluster Manager Commands

Performance Cluster Manager Command	Description
`cnodes --ice-compute`	List the compute node names; useful in scripting operations
`cpower node off <node name>`	Powers down
`cpower node on <node name>`	Powers up the named nodes
`cimage --…`	Manages the file system image for the various nodes

- You must use the correct names for the Admin and Service nodes in any commands.

3.5.4.3 Choosing Whether PBS Will Manage Cpusets with HPE 8600 Running HPE MPI

You can use cpusets on an HPE 8600 running PBS, whether or not PBS manages the cpusets. If PBS manages the cpusets for you, that means that PBS dynamically creates a cpuset for each job and confines job processes to that cpuset. If PBS does not manage the cpusets for you, then jobs are not confined to cpusets. You can use the PBS cgroups hook to manage the cpusets on the 8600; see section 3.5.4.10, "Configure Cgroups to Manage Cpusets", on page 35.

3.5.4.4 Installation of the PBS Server, Scheduler, and Communication Daemons

Install the PBS server, scheduler, communication daemon, and commands on a single service node; here we assume this node is "service0":

1. Log on to service0 as root.
2. Unzip and untar the appropriate package.
3. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
4. Install the server sub-package:

   ```
   rpm -i
       <path/to/sub-package>pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

5. Do not start PBS

3.5.4.5 Installation of the PBS MoM

You install and configure MoM once on the root file system, then you push the image to all of the compute nodes by propagating it to the rack leaders. Then you reboot each node with the new image.

1. Log on to the Admin node as root.
2. Determine which image file is being used on the compute nodes. To list the nodes on rack 1:

   ```
   cimage --list-nodes r1
   ```

   It will show output in the form "*node: image_name kernel*" similar to

   ```
   r1i0n0: compute-sles15sp1 2.6.26.46-0.12-smp
   ```

   Thus node r1i0n0 is running the image "`compute-sles15sp1`" and the kernel version "2.6.26.46-0.12-smp". For the remaining steps, it is assumed that those are the images and kernel available.
3. List the available images:

   ```
   cimage --list-images
   ```

   which will list the images available for the compute nodes. Each image may have multiple kernels.
4. Unless you are experienced in managing the image files, we suggest that you create a copy of the image in use and install PBS in that copy. To copy an image:

   ```
   cinstallman --create-image --clone --source compute-sles15sp1 --image compute-sles15sp1pbs
   ```
5. The image file lives in the directory `/opt/clmgr/image/images`, so change into the `tmp` directory found in the new image just cloned:

   ```
   cd /opt/clmgr/image/images/compute-sles15sp1pbs/tmp
   ```
6. Chroot to the new image file:

   ```
   chroot /opt/clmgr/image/images/compute-sles15sp1pbs /bin/sh
   ```

   The new root is in effect.
7. Download, unzip and untar the PBS package
8. Make sure that parameters for PBS_HOME, PBS_EXEC and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
9. Install the PBS execution sub-package in the normal execution directory, `/opt/pbs`, in this system image:

   ```
   rpm -i <path/to/sub-package>pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hard-
       ware>.rpm
   ```
10. Do not start PBS
11. Exit from the chroot shell and return to root's normal home directory.
12. Power down each rack of compute nodes:

    ```
    for n in `cnodes --ice-compute` ; do
        cpower node off $n
    done
    ```
13. Publish the new system image to the compute nodes:

    ```
    cimage --push-rack compute-sles15sp1pbs r\*
    ```

This instruction will take several minutes to finish.

14. Set the new image and kernel to be booted. This need not be done if: (1) rather than cloning a new image, you have installed PBS into the image already running on the compute nodes; or (2) you are using an image that was already pushed to the nodes.

```
cimage --set compute-sles15sp1pbs 2.6.26.46-0.12-smp r\*i\*n\*
```

15. Power up the compute nodes:

```
for n in `cnodes --ice-compute` ; do
    cpower node on $n
done
```

It will take several minutes for the compute nodes to reboot.

3.5.4.6 Start PBS Server

1. Log on to the Service node as root
2. On the Service node, start the PBS server, scheduler, and communication daemons:

```
systemctl start pbs
```

or

```
<path to script>/pbs start
```

3.5.4.7 Configure Licensing

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

3.5.4.8 Add Compute Nodes

Using `qmgr`, add the compute nodes to the PBS configuration:

```
for N in `cnodes --ice-compute`
do
    qmgr -c "create node $N"
done
```

If you use the IP address for the name of the vnode:

1. Add `PBS_MOM_NODE_NAME=<IP address>` to `pbs.conf` on the execution host
2. Restart MoM

3.5.4.9 Configuring Placement Sets on the HPE 8600

Placement sets improve job placement on execution nodes. If you want to use cgroups, you can generate placement set information. See "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

Placement sets can be defined only after you have defined the compute nodes as in the previous section. Put placement set resource information in a Version 2 configuration file for each host. Make sure that the vnode names you use in your Version 2 configuration file are exactly the same as the names generated by the cgroups hook.

Steps to generate placement sets:

1. Shut down the server.
2. Add a resource named "router" (the script uses this exact name):

   ```
   Qmgr: create resource router type=string_array, flag=h
   ```

3. Restart the server
4. Generate your placement sets and set their resource values at vnodes; you can use the `sgiICEplacement.sh` script, which is in the `unsupported` directory, as an example
5. Verify the result:
   a. Run the `pbsnodes -a` command
   b. Look for the line "`resources_available.router`" at each vnode. The value assigned to the "router" resource should be in the form "*r#,r#i#*", where *r* identifies the rack number and *i* identifies the IRU number.

3.5.4.10 Configure Cgroups to Manage Cpusets

Do the following steps as root on the server node (service0).

1. Make sure that cgroups subsystems including cpuset are mounted on the compute nodes. See "Configuring and Using PBS with Cgroups" on page 571 in the PBS Professional Administrator's Guide.
2. Modify the cgroups hook configuration file:
   a. Export the cgroups hook configuration file:

      ```
      qmgr -c "export hook pbs_cgroups application/x-config default" > pbs_cgroups.json
      ```

   b. Edit the cgroups configuration file. To get default cpuset behavior, set these:

      ```
      "vnode_per_numa_node"   : true,
      "use_hyperthreads"      : true,
      "ncpus_are_cores"       : false,
      ```

      We describe how to manage hyperthreading behavior in "Configuring Hyperthreading Support" on page 582 in the PBS Professional Administrator's Guide.

   c. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file. Check to see whether it is mounted:

      ```
      # mount | grep cgroup | grep memory
      ```

      If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   d. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

      ```
      "cgroup": {
        ...
      "memory": {
      "enabled": false,
      ```

   e. Read in the updated cgroups hook configuration:

      ```
      qmgr -c "import hook pbs_cgroups application/x-config default pbs_cgroups.json"
      ```

3. Enable the cgroups hook:

   ```
   # qmgr -c "set hook pbs_cgroups enabled=true"
   ```

4. Restart the MoMs, using either `systemctl` or the start/stop script:

   ```
   # for n in `cnodes --ice-compute`; do
         ssh $n "systemctl restart pbs"
   done
   or
   # for n in `cnodes --ice-compute`; do
         ssh $n "<path to script>/pbs restart"
   done
   ```

5. Check that you have created one vnode for each NUMA node, and that the vnode state is *free*:

   ```
   # pbsnodes -av
   ```

3.5.5 Making User Paths Work

If you're installing PBS for the first time, make sure that user PATHs include the location of the PBS commands. If users already have paths to PBS commands, you can either make symbolic links so that users don't have to change their PATHs, or users can set their PATHs to the locations of the commands.

3.5.5.1 Setting User Paths to Location of Commands

Users should set their path to include PBS_EXEC/bin and PBS_EXEC/sbin. For example, if PBS_EXEC is /opt/pbs, by including /opt/pbs/bin, users will have PBS executables in their path.

3.5.5.2 Making Existing User Paths Work with New Location

You may need to make users' PATH variable point to the new PBS_EXEC directory, especially if PBS_EXEC is in a non-default location, or if you're using a new location. You can use symbolic links to enable users to access PBS commands via their current PATH:

```
<user PATH>/bin -> <PBS_EXEC>/bin
<user PATH>/sbin -> <PBS_EXEC>/sbin
```

For example if the old location was /usr/pbs_bin, create the link /usr/pbs_bin/bin -> /opt/pbs/bin.

3.5.5.3 Testing User Paths

- Test that a normal user can submit a job. As a normal user, type:

  echo "sleep 60" | /opt/pbs/bin/qsub

  This submits a job to the queue named 'workq' (the queue that is automatically defined as the default queue)
- If you've changed the location of PBS commands and used symbolic links to allow users to keep their old PATHs, verify that the old paths work:

  echo "sleep 60" | <old user path>/bin/qsub

3.5.6 Caveats for Uninstalling on Linux

Using rpm -e, even on an older package than the one you are currently using, will cause any currently running PBS daemons to shut down, and will also remove the system V init and/or systemd service startup files. This will prevent PBS daemons from starting automatically at system boot time. If you wish to remove an older RPM without these effects, use rpm -e --noscripts.

3.6 Installing via dpkg on Ubuntu

To install PBS Professional on Ubuntu, use the following steps:

1. Choose the .deb package to install. Make sure it is appropriate for the host's function, which could be server, execution, or client host.
2. Use `dpkg -i` to install the .deb package:

   `dpkg -i <.deb package>`
3. Update `/etc/pbs.conf`: set the PBS_START_* parameters to the appropriate values. Here is an example where one host will run all daemons:

   ```
   PBS_EXEC=/opt/pbs
   PBS_SERVER=<hostname>
   PBS_START_SERVER=1
   PBS_START_SCHED=1
   PBS_START_COMM=1
   PBS_START_MOM=1
   PBS_HOME=/var/spool/pbs
   PBS_CORE_LIMIT=unlimited
   PBS_SCP=/usr/bin/scp
   ```
4. Each hostname **must** resolve to at least one non-loopback IP address. Typically, the default `/etc/hosts` file does not conform to this prerequisite, so you probably need to do additional network configuration to make PBS work on Ubuntu. You can do this by using DNS or by adding a new entry into `/etc/hosts` that associates the hostname with a non-loopback IP address. To update `/etc/hosts`:

   Update the IP address for the server host:

   ```
   127.0.0.1         localhost
   192.168.238.135   <server hostname>
   ```

   The following lines are desirable for IPv6-capable hosts:

   ```
   ::1     ip6-localhost ip6-loopback
   fe00::0 ip6-localnet
   ff00::0 ip6-mcastprefix
   ff02::1 ip6-allnodes
   ff02::2 ip6-allrouters
   ```
5. If the PBS Data service management account, usually called *pbsdata,* does not already exist, create it. See section 3.5.1.3, "Create PBS Data Service Management Account", on page 23.
6. Start PBS:

   `/etc/init/pbs start`

3.7 Installing PBS on Windows Hosts

3.7.1 Daemon Layout

MoMs and client commands can run on Windows machines, but all other PBS components are installed on Linux hosts. Install the Windows MoM and client packages, and install your chosen Linux server/scheduler/comm package.

3.7.2 Prerequisites

Please do not jump straight to this section in your reading. Before downloading and installing PBS, please make sure that you have read the following and taken any required steps:

- Prerequisites: All of Section 2.1, "Prerequisites for Running PBS", Section 2.3, "PBS Configurations for Windows", Section 2.3.6, "User Authorization Under Windows", and Section 2.3.8, "Windows Caveats" and their subsections.
- Please read Section 3.1, "Overview of Installation".
- Please start your installation by following the steps in Section 3.3, "Major Steps for Installing PBS Professional".
- Please check all of Section 3.4, "All Installations" and its subsections to make sure you have prepared properly.

3.7.3 Default Installation Locations

On Windows systems, PBS is installed in `\Program Files (x86)\PBS\`.

Default installation directories:

`PBS_HOME`: `C:\Program Files (x86)\PBS\home`

`PBS_EXEC`: `C:\Program Files (x86)\PBS\exec`

3.7.4 Where to Run Daemons (Services)

When PBS is installed on a complex, the MoM must be run on each execution host. The server, scheduler, and communication daemons are installed on a Linux front-end system. The PBS Windows package contains the following:

- PBS Professional software
- Supporting text files (README etc.)

3.7.5 PBS Requirements on Windows

All Windows hosts in a PBS complex must be in the same domain.

PBS Professional is supported if the domain controller server is configured "native". Running PBS in an environment where the domain controllers are configured in "mixed-mode" is not supported.

You must install PBS Professional from an Administrator account.

Before you install PBS on Windows, make sure you are using the correct type of account. See section 2.3.5, "Windows Configuration in a Domained Environment", on page 14.

PBS Professional requires that the drive that PBS was installed under (e.g. `\Program Files\PBS` or `\Program Files (x86)\PBS`) be configured as an NTFS filesystem.

Before installing PBS Professional, be sure to uninstall any old PBS Professional files. For details see "Uninstalling PBS Professional on Windows" on page 44.

You can specify the destination folder for PBS using the "Ask Destination Path" dialog during setup.

3.7.6 Make Sure Hostnames Resolve Correctly

Make sure that all of your hosts consistently resolve to the correct IP addresses. Wrong IP address to hostname translation will cause errors for PBS.

Configure your system to talk to a properly configured and functioning DNS server.

On each host, add the correct host entries to the following files:

c:\windows\system32\drivers\etc\hosts

hosts.equiv

Make each etc\hosts file identical on each host, and make each hosts.equiv file identical on each host.

Example 3-1: Your server is serverA, your execution host is exec01, and your client hosts are client001 and client002. Hostnames and IP addresses look like this:

Table 3-6: Example Host Names and Addresses

Hostname	Host IP Address
serverA	192.168.0.101
exec01	192.168.0.102
client001	192.168.0.103
client002	192.168.0.104

Here's what etc\hosts should look like at each host:

```
192.168.0.101 server
192.168.0.102 mom
192.168.0.103 client001
192.168.0.104 client002
```

Here's what hosts.equiv should look like at each host:

```
server
mom
client001
client002
```

3.7.7 Create Job Submission Accounts

Set up any user accounts that will be used to run PBS jobs. All job submission accounts must be part of the same domain as any Windows hosts. The accounts should not be Administrator-type accounts, that is, not a member of the "Domain Administrators" or local "Administrators" group, so that basic authentication using hosts.equiv can be used.

Once the accounts have been set up, list all PBS hosts (server, execution, client, file storage) in the hosts.equiv or job submitters' .rhosts files. Do this on all the hosts, to allow accounts on these hosts to access PBS services such as job submission and remote file copying.

The hosts.equiv file can usually be found in the following location:

C:\windows\system32\drivers\etc\hosts.equiv

3.7.8 Create Installation and Service Accounts

Before you install PBS, you must create the accounts that PBS requires.

On Windows, the PBS data service management account is the same as the PBS Windows service account. You do not need to create a separate data service account.

You need to create the installation and service accounts. We give instructions below.

You do not need to create the following accounts:

PBS data service management account

On Windows, the PBS data service management account is the same as the PBS Windows service account. You do not need to create a separate PBS data service management account.

Data service account

Account that is internal to the data service, and has its own data service password. On installation, PBS creates the internal data service account, and maps it to the PBS service account. The data service account name must be the same as the PBS service account.

You do need to create the installation and service accounts, and we give instructions below.

3.7.8.1 Creating Installation Account in Domained Environment

The installation account is the account from which PBS is installed. The installation account must be the only account that will be used for all steps of PBS installation including modifying configuration files, setting up failover, and so on. If any of the PBS configuration files are modified by an account that is not the installation account, permissions/ownerships of the files could be reset, rendering them inaccessible to PBS. For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

3.7.8.2 Creating PBS Service Account in Domained Environment

The PBS service account is the account under which the PBS service (`pbs_mom`) will run.

- This account can have any name.
- The name of the account defaults to *pbsadmin*.
- This account must exist while any PBS services are running.
- The password for this account should not be changed while PBS is running.
- Create the PBS service account before installing PBS.
- For domained environments, the PBS service account must:
  a. be a domain account
  b. be a member of the "Domain Users" group, and **only** this group
  c. have "domain read" privilege to all users and groups.
- For a domained environment, delegate "read access to all users and groups information" to the PBS service account. See section 3.7.8.2.i, "Delegating Read Access to PBS Service Account in Domained Environment", on page 41.
- If the PBS service account is set up with no explicit domain read privilege, MoM may hang. The hang happens when users submit jobs from a network mapped drive without the `-o/-e` option for redirecting files. When this happens, bring up Task manager, look for a "`cmd`" process by the user who owned the job, and kill it. After the first `cmd` process is killed, you may have to look for a second one (the first one copies the output file, the second one does the error file). This should un-hang the MoM.
- The PBS service account must be a member of the local Administrators group. Add the PBS service account to the local Administrators group:
  ```
  net localgroup Administrators <domain name>\<service account name> /add
  ```
- Do not put a CPU time limit on the service account . If you do, the datastore will die and kill the server.

3.7.8.2.i Delegating Read Access to PBS Service Account in Domained Environment

- To delegate "read access to users and groups information" to the PBS service account:
  a. On the domain controller host, bring up Active Directory Users and Computers.
  b. Select <domain name>, right mouse click, and choose "Delegate Control". This will bring up the "Delegation of Control Wizard".
  c. When it asks for a user or group to which to delegate control, select the name of the PBS service account.
  d. When it asks for a task to delegate, specify "Create a custom task to delegate".
  e. For active directory object type, select the "this folder, existing objects in this folder, and creation of objects in this folder" button.
  f. For permissions, select "Read" and "Read All Properties".
  g. Exit out of Active Directory.

3.7.8.2.ii Service Account Caveats

If you change the name of the PBS service account:

- You must restart the daemons on that host
- On Windows, you must re-register the MoM service

3.7.9 Installation Notes for Domained Environment

3.7.9.1 Installation Path

- The destination/installation path of PBS must be NTFS. All PBS configuration files must reside on an NTFS filesystem.

3.7.9.2 Notes on Installation

- The installation account must be used in all future invocations of the install program when setting up a complex of PBS hosts.
- The install program requires the installer to supply the password for the PBS service account. This same password must be supplied to future invocations of the install program on other hosts.
- The install program will enable the following rights to the PBS service account: "Create Token Object", "Replace Process Level Token", "Log On As a Service", and "Act As Part of the Operating System".
- The install program will enable Full Control permission to local "Administrators" group on the install host for all PBS-related files.
- The install program will give you a specific error if the PBS service account is not a member of the local Administrators group on the local computer. It will quit at this point, and you must go back:
  a. Make the PBS service account be a member of the local Administrators group on the local computer:

     ```
     net localgroup Administrators <name of PBS service account> /add
     ```

  b. Re-run the install program.

3.7.10 Steps to Install PBS on Windows

1. On each execution and client host, do the following:
   a. Log in with the installation account.
   b. Install the KB2999226 update for Windows on all Windows Server 2012 execution and client machines.
   c. Download the MSI installer (the .msi file).
   d. Double-click the MSI installer; the splash screen is displayed.
   e. Click the *Next* button to move to the license page. Accept the license.
   f. Click the *Next* button and choose the path where you will install the PBS executable. By default this path points to "C:\Program Files (x86)\PBS\".
   g. Using "Run As Administrator", open a Command prompt.
2. Install the server/scheduler package on a Linux host. See section 3.5.2.4.ii, "Install PBS on Server Host", on page 27.

3.7.11 Post-installation Steps

3.7.11.1 Configuring MoMs

On each execution host, manually execute the win_postinstall.py script as shown below. When you specify the PBS service account, whether or not you are on a domain machine, include only the username, not the domain. For example, if the full username on a domain machine is <domain>\<username>, pass only *username* as an argument:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account> -p <PBS
    service account password> -s <server name> -t execution -c <path to scp.exe>
```

3.7.11.2 Configuring Client Hosts

On each client host, manually execute the win_postinstall.py script as shown below. When you specify the PBS service account, whether or not you are on a domain machine, include only the username, not the domain. For example, if the full username on a domain machine is <domain>\<username>, pass only *username* as an argument:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account> -p <PBS
    service account password> -s <server name> -t client -c <path to scp.exe>
```

3.7.11.3 Defining Vnodes

Using the qmgr command, define the vnodes that the server will manage. See "Creating Vnodes" on page 38 in the PBS Professional Administrator's Guide.

3.7.11.4 Configuring Remote File Copy

If you will use scp for your remote file copy mechanism, configure passwordless ssh. If you will use $usecp to specify your remote file copy mechanism, you do not need to configure passwordless ssh. See "Configuring MoM for Remote Copy" on page 562 in the PBS Professional Administrator's Guide.

3.7.12 Post-installation Considerations on Windows

3.7.12.1 File Creation

The installation process automatically creates the following file:

```
[PBS Destination folder]\pbs.conf
```

containing at least the following entries:

```
PBS_EXEC=[PBS Destination Folder]\exec
PBS_HOME=[PBS Destination Folder]\home
PBS_SERVER=<server name>
PBS_START_SERVER=<value>
PBS_START_SCHED=<value>
PBS_START_MOM=<value>
PBS_START_COMM=<value>
PBS_AUTH_METHOD=pwd
```

where `PBS_EXEC` will contain subdirectories where the executable and scripts reside, `PBS_HOME` will house the log files, job files, and other processing files, and *server-name* will reference the system running the PBS server. The `pbs.conf` file can be edited by calling the PBS program "`pbs-config-add`". For example:

```
\Program Files (x86)\PBS\exec\bin\pbs-config-add "PBS_SCP=C:\Windows\System32\OpenSSH\scp.exe"
```

Don't edit `pbs.conf` directly as the permission on the file could get reset causing other users to have a problem running PBS.

3.7.12.2 File Access on Windows

Upon installation, some PBS directories have restricted access. The following directories have files that are readable by the \\Everyone group but writable only by Administrators-type accounts:

```
PBS_HOME/mom_logs/
PBS_HOME/spool/
```

The following directories have files that are only accessible to Administrators-type accounts:

```
PBS_HOME/mom_priv/
```

3.7.13 Startup on Windows

- The auto-startup of the MoM service is controlled by the PBS `pbs.conf` file as well as the *Services* dialog. You invoke this via *Settings->Control Panel->Administrative Tools->Services*. If the service fails to start up with the message, "incorrect environment", it means that the PBS_START_MOM `pbs.conf` variable is set to *0* (*False*).
- On Windows, sometimes PBS may fail to start automatically after the boot. We recommend that you change the startup mode from " *[Startup type: Automatic]*" to "*[Startup type: Automatic (Delayed Start)]*", which means "shortly after boot".

  Open `regedit` to change the registry keys. These are, in some versions of Windows: *HKLM\SYSTEM\CurrentControlSet\services\<PBS Professional>\DelayedAutostart*.

  When startup is delayed, this has the value *1*. When not delayed, its value is *0*.

3.7.13.1 Setting Up User Accounts and Directories

You should review the recommended steps for setting up user accounts and home directories, as documented in section 2.3.6, "User Authorization Under Windows", on page 15.

3.7.14 Uninstalling PBS Professional on Windows

For uninstalling versions 5.4.2 through 8.0, use a domain admin account. For later versions, use an Administrator account. Note that as of 19.4.1, the only PBS service on Windows is PBS_MOM.

1. Use the Task Manager to stop/kill the services: PBS_SERVER, PBS_SCHED, PBS_COMM, PBS_MOM, and PBS_RSHD.
2. Manually de-register the PBS services:

```
pbs_account --unreg pbs_server
pbs_account --unreg pbs_sched
pbs_account --unreg pbs_comm
pbs_account --unreg pbs_mom
pbs_account --unreg pbs_rshd
```

3. Use the MSI installer to uninstall the PBS package. At the second double click, you get the "*Remove*" option.
4. Manually delete the PBS directory at "`C:\\program Files (x86)\PBS`"

4 Communication

4.1 Communication Within a PBS Complex

There are two primary communication methods in PBS: TCP, where a client sends a request to a server using a non-permanent TCP connection, and TPP, in which daemons establish permanent TCP connections to one or more `pbs_comm` daemons and use these permanent connections to reach other daemons. TPP stands for "TCP-based Packet Protocol".

A PBS complex using TPP can handle much greater throughput than in previous versions of PBS, and the scheduler can start jobs much faster. A PBS complex using TPP does not need as many reserved ports as previous versions.

4.2 Terminology

Endpoint

A PBS server, scheduler, or MoM daemon.

Communication daemon, comm

The daemon which handles communication between the server, scheduler, and MoMs. Executable is `pbs_comm`.

Leaf

An endpoint (a server, scheduler, or MoM daemon.)

TPP

TCP-based Packet Protocol. Protocol used by `pbs_comm`.

4.3 Prerequisites

Each hostname must resolve to at least one non-loopback IP address.

4.4 Communication Parameters

4.4.1 Location of Communication Daemon for Endpoint

You can tell each endpoint which communication daemon it should talk to. Specifying the port is optional.

PBS_LEAF_ROUTERS

Parameter in `/etc/pbs.conf`. Tells an endpoint where to find its communication daemon.

Format: *PBS_LEAF_ROUTERS=<host>[:<port>][,<host>[:>port>]]*

4.4.2 Location of Other Communication Daemons

When you add a communication daemon, you must tell it about the other pbs_comms in the complex. When you inform communication daemons about each other, you only tell one of each pair about the other. Do not tell both about each other. We recommend that an easy way to do this is to tell each new pbs_comm about each existing pbs_comm, and leave it at that.

PBS_COMM_ROUTERS

Parameter in /etc/pbs.conf. Tells a pbs_comm where to find its fellow communication daemons.

Format: *PBS_COMM_ROUTERS=<host>[:<port>][,<host>[:<port>]]*

4.4.3 Number of Threads for Communication Daemon

By default, each pbs_comm process starts four threads. You can configure the number of threads that each pbs_comm uses. Usually, you want no more threads than the number of processors on the host.

PBS_COMM_THREADS

Parameter in /etc/pbs.conf. Tells pbs_comm how many threads to start.

Maximum allowed value: *100*

Format: *Integer*

Example:

```
PBS_COMM_THREADS=8
```

4.4.4 Daemon Log Mask

By default, pbs_comm produces few log messages. You can choose more logging, usually for troubleshooting. See section 4.5.10, "Logging and Errors with TPP", on page 54 for logging details. The daemon rereads this parameter when HUPed.

PBS_COMM_LOG_EVENTS

Parameter in /etc/pbs.conf. Tells pbs_comm which log mask to use.

Format: *Integer*

Default: *511*

Example:

```
PBS_COMM_LOG_EVENTS=<log level>
```

4.4.5 Name of Endpoint Host

By default, the name of the endpoint's host is the hostname of the machine. You can set the name that the endpoint uses for its host. This is useful when you have multiple networks configured, and you want PBS to use a particular network. TPP internally resolves the name to a set of IP addresses, so you do not affect how pbs_comm works.

PBS_LEAF_NAME

Parameter in /etc/pbs.conf. Tells endpoint what name to use for network. The value does not include a port, since that is usually set by the daemon.

Canonicalized value of this becomes the value of resources_available.host.

By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network.

The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in /etc/pbs.conf to something else, make sure you set the Mom attribute at vnode creation.

TPP internally resolves the name to a set of IP addresses, so you do not affect how pbs_comm works.

Format: *String*

Example:

```
PBS_LEAF_NAME=host1
```

4.4.6 Whether Host Runs Communication Daemon

Just as with the other PBS daemons, you can specify whether each host should start pbs_comm.

PBS_START_COMM

Parameter in /etc/pbs.conf. Tells PBS init script whether to start a pbs_comm on this host if one is installed. When set to *1*, pbs_comm is started.

Format: *Boolean*

Default: *0*

Example:

```
PBS_START_COMM=1
```

4.4.7 Scheduler Throughput Mode

You can tell the scheduler to run asynchronously, so it doesn't wait for each job to be accepted by MoM, which means it also doesn't wait for an execjob_begin hook to finish. Especially for short jobs, this can give you better scheduling performance. You can run the scheduler asynchronously only when the complex is using TPP mode.

throughput_mode

Scheduler attribute. When set to *True*, the scheduler runs asynchronously and can start jobs faster. Only available when complex is in TPP mode.

Format: *Boolean*

Default: *True*

Example:

```
qmgr -c "set sched throughput_mode=<Boolean value>"
```

Trying to set the value to a non-Boolean value generates the following error message:

```
qmgr obj= svr=default: Illegal attribute or resource value
qmgr: Error (15014) returned from server
```

4.4.8 Managing Communication Behavior

rpp_highwater

Server attribute.

This is the maximum number of messages per stream (meaning the maximum number of messages between each pair of endpoints).

Format: *Integer*

Valid values: *Greater than or equal to one*

Default: *1024*

Python type: *int*

rpp_max_pkt_check

Server attribute.

Maximum number of TPP messages processed by the main server thread per iteration.

Format: *Integer*

Default: *64*

Python type: *int*

rpp_retry

Server attribute.

In a fault-tolerant setup (multiple `pbs_comms`), when the first `pbs_comm` fails partway through a message, this is number of times TPP tries to use any other remaining `pbs_comms` to send the message.

Format: *Integer*

Valid values: *Greater than or equal to zero*

Default: *10*

Python type: *int*

4.5 Inter-daemon Communication Using TPP

The PBS server, scheduler, and MoM daemons communicate with each other using TPP through the communication daemon pbs_comm, except for scheduler-server and server-server communication, which uses TCP. The server, scheduler, and MoMs are communication endpoints, connected by one or more pbs_comm daemons. The following figure illustrates communication within a PBS complex using TPP.

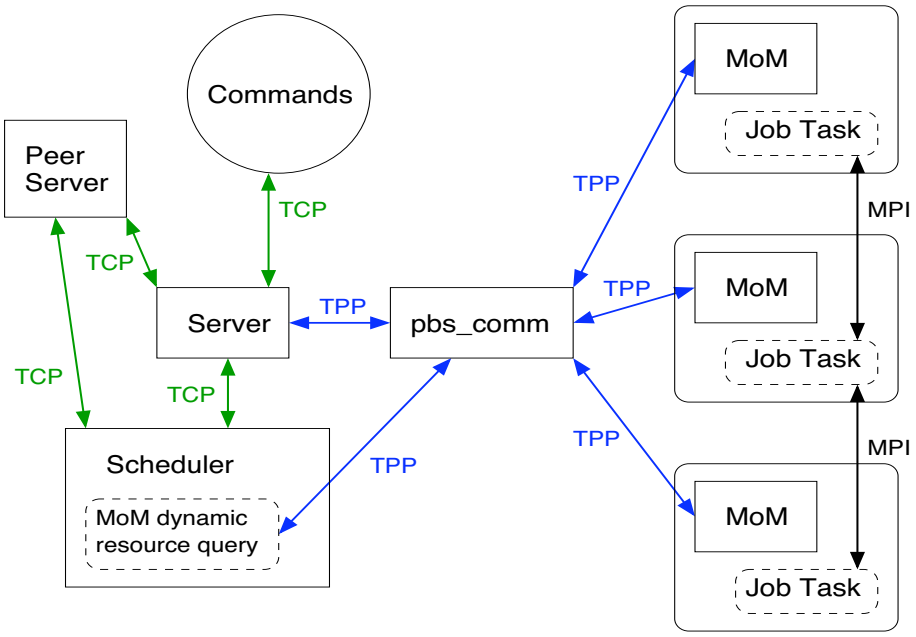


Figure 4-1:Communication Within PBS Complex Using TPP

Communication daemons are connected to each other. If there are multiple pbs_comms, and two endpoints on different pbs_comms transmit data, communication between endpoints goes from the first endpoint, to the endpoint's configured pbs_comm daemon, to the pbs_comm configured for the receiving endpoint, to the receiving endpoint.

4.5.1 Inter-daemon Connection Behavior Using TPP

When each endpoint starts up, it automatically attempts to connect to the configured or default pbs_comm daemon. If the pbs_comm daemon is available, the connection attempt succeeds; if not, the endpoint continues to attempt to connect to the pbs_comm daemon using a background thread. The order in which endpoints and pbs_comms are started is not important. Connections are completed when the pbs_comm daemon becomes available. If you have configured multiple pbs_comms, each endpoint continues to periodically attempt to connect to each one until all connections succeed.

If the connection from an endpoint to a pbs_comm daemon fails, the endpoint attempts to find another already-connected pbs_comm daemon to send data via that connection. When the original failed connection is reestablished (via automatic periodic background attempts to connect to the failed daemon) data exchange switches over to the original connection.

When a pbs_comm daemon is configured to talk to other pbs_comms, it behaves exactly the same way as an endpoint.

Just after you start a MoM, it may not appear to be up, because there is a delay between endpoint connection attempts. The MoM may need up to 30 seconds to show up.

If you have only one communication daemon installed (failover is not configured), and that communication daemon is killed, vnodes become unreachable.

4.5.1.1 Sending and Receiving

Endpoints have a built-in retry mechanism to re-send information that has not been acknowledged by the receiver. The receiving endpoint can determine whether it has received duplicate data packets.

4.5.1.2 Data Compression

Some jobs cause the server and MoMs to exchange a very large amount of data. The communication daemon automatically compresses the data before communication. In communications, there is usually benefit from compression, because communication is usually CPU-bound, not I/O-bound.

4.5.2 Communication Daemon Syntax

4.5.2.1 Usage on Linux

On Linux, the `pbs_comm` executable takes the following options:

pbs_comm [-N] [-r <other routers>] [-t <number of threads>]

-r

Used to specify the list of other `pbs_comm` daemons to which this `pbs_comm` must connect. This is equivalent to the `pbs.conf` variable PBS_COMM_ROUTERS. The command line overrides the variable. Format:

<host>[:<port>][,<host>[:<port>]]

-t

Used to specify the number of threads the `pbs_comm` daemon uses. This is equivalent to the `pbs.conf` variable PBS_COMM_THREADS. The command line overrides the variable. Format:

Integer

-N

The communication daemon runs in standalone mode.

4.5.3 Adding Communication Daemons

4.5.3.1 Installation Location of Communication Daemons

The `pbs_comm` daemon can be installed on any host that is connected to the PBS complex. By default, a `pbs_comm` is installed on the server host(s), and all endpoints will connect to it (them) by default.

4.5.3.2 Configuring Communication Daemons

Make sure that when you configure additional communication daemons, you only point one of each pair of `pbs_comm`s to the other; do not point both at each other. We recommend that an easy way to do this is to tell each new `pbs_comm` about each existing `pbs_comm`, and leave it at that.

Steps to configure additional pbs_comms:

1. Tell each endpoint that goes with the new pbs_comm where to find the new pbs_comm. Edit the pbs.conf file on the endpoint's host, and add:

   PBS_LEAF_ROUTERS=<host>[:<port>][,<host>[:>port>]]

2. For each new pbs_comm, tell each new pbs_comm about previous pbs_comms. Do not tell existing pbs_comms about new pbs_comms. So if you have an existing pbs_comm C1 and add a new pbs_comm C2, only point C2 to C1. In pbs.conf on C2's host, add:

   PBS_COMM_ROUTERS=<C1 host>[:<C1 port>]

   If you add C3, point C3 to both C1 and C2. On C3's host, add:

   PBS_COMM_ROUTERS=<C1 host>[:<C1 port>],<C2 host>[:<C2 port>]

3. Optionally, set the number of threads the new pbs_comm will use. The default is *4*. We recommend not specifying more threads than processors on the host. In pbs.conf, add:

   PBS_COMM_THREADS=<number of threads>

4. Optionally, set the desired log level for the new pbs_comm. In pbs.conf, add:

   PBS_COMM_LOG_EVENTS=<log level>

5. On the new pbs_comm host, tell the init script to start pbs_comm. In pbs.conf, add:

   PBS_START_COMM=1

1. If you are running a PBS complex that contains both Linux and Windows execution hosts, on any hosts running comms, configure sssd so that the users of the Windows domain can log in to the Linux host on which pbs_server and sssd run. See "Mixed Linux-Windows Operation" on page 659 in the PBS Professional Administrator's Guide.

   For an example, see section 8.4.5, "Configuring SSSD", on page 380. For information on configuring sssd, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/windows_integration_guide/index#sssd-ad-proc and https://access.redhat.com/articles/3023951.

   If you want the Linux host to automatically create a home directory for an Active Directory user if that home directory does not exist at login, you may have to set SELinux to permissive mode. This is optional.

4.5.3.2.i Caveats for Configuring Communication Daemons

When you HUP the communication daemon, it reads only PBS_COMM_LOG_EVENTS from pbs.conf. If you change any of its other parameters, you must restart the communication daemon:

```
<path to start/stop script>/pbs restart
```

4.5.4 Recommendations for Maximizing Communication Performance

You can partition your endpoints so that each group of endpoints has its own pbs_comm(s). Keeping the workload for each pbs_comm below the level that degrades performance will speed up your complex. Your site's characteristics determine how many pbs_comms you need. Here are some rules of thumb for adding pbs_comms:

- One pbs_comm per 2000 MoMs, where communication is light
- One pbs_comm per rack of ~150 - 200 MoMs, where communication is heavier
- If server start time doubles, add a pbs_comm
- If the CPU usage for a pbs_comm is above 10 or 15 percent, add a pbs_comm
- If performance drops, consider adding a pbs_comm

4.5.5 Robust Communication with TPP

4.5.5.1 Failover and Communication Daemons

When failover is configured, endpoints automatically connect to the pbs_comm daemons running on either the primary or secondary PBS server hosts, allowing for communication failover. If both pbs_comms are available, communication goes through the pbs_comm on the primary server host. If the primary server host fails, communication automatically goes through the pbs_comm on the secondary server host. When the primary server host comes back up, communication is automatically resumed by the pbs_comm on the primary server host. If failover is configured and the only pbs_comms are on the primary and secondary server hosts, and both of those hosts fail, communication between endpoints is unavailable.

4.5.5.2 Fault Tolerance

By default, endpoints automatically connect to the pbs_comm daemon running at the server host.

You can configure each endpoint to connect to multiple communication daemons. If one of the communication daemons fails, the endpoint can still communicate with the rest of the complex using the alternate communication daemons. When a failed pbs_comm comes back online, it automatically resumes handling communications.

If you have configured failover, you have communication fault tolerance to the extent of one of the pbs_comms on the primary or secondary server host failing. If you want fault tolerance beyond or instead of failover, you must explicitly install and configure additional pbs_comm daemons.

4.5.6 Extending Your Complex

To add a new rack to a PBS complex using TCP, take the following steps:

1. Install MoMs as usual on the new execution hosts.
2. Optionally, edit the configuration file on the new MoM hosts to include failover settings.
3. You can configure new MoMs to communicate via existing pbs_comms. However, if you are adding many MoMs, we recommend deploying additional pbs_comms. Follow the steps in "Adding Communication Daemons" on page 50.
4. Start the daemons in the new rack, and tell the server about the new vnodes:

   ```
   qmgr -c "create node <vnode name>"
   ```

   If you use the IP address for the name of the vnode:

   a. Add PBS_MOM_NODE_NAME=<IP address> to pbs.conf on the execution host

   b. Restart MoM

4.5.7 Changing IP Address of pbs_comm Host

To change the IP address of a pbs_comm host:

1. Change the IP address of the host
2. Update the DNS
3. Restart pbs_comm on that host

Each endpoint or pbs_comm periodically retries the connection to each pbs_comm that it knows about. When a pbs_comm becomes unavailable, all connections to it are automatically retried until they succeed. Endpoint and pbs_comm IP addresses to target pbs_comms are internally cached for a short time, so if you change the IP address of a target, they will not be able to connect for this time. When this time runs out, endpoints and pbs_comms refresh their IP addresses, and connections are reestablished.

4.5.8 Configuring Communication for Internal and External Networks

PBS complexes often use an internal network and an external network. PBS clients such as qsub and qstat communicate to the server over the external network. The daemons communicate with each other over the internal network. In this case, the server host is configured with multiple network interfaces, one for each of the different networks.

The default value of the endpoint's name is the hostname. The TPP network resolves the endpoint's name to the IP address of the machine, and could end up using the external IP address of the host. When this endpoint, for example the server, sends a message to another endpoint, say the MoM, it would embed this external IP address in the message. The MoM detects that this message has arrived from an external IP address and could reject the message, since the MoM is typically configured to use only the internal network and is unaware of the external IP address.

Instead of letting the endpoint use the machine hostname as the endpoint's name (which is the default), set the endpoint's name to a variable that resolves to only the internal network address(es) of the server host. To do that, set the PBS_LEAF_NAME pbs.conf variable to the internal network name of the host.

The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in /etc/pbs.conf to something else, make sure you set the Mom attribute at vnode creation.

4.5.9 Troubleshooting Communication with TPP

New connections are being dropped at a pbs_comm

Check whether the pbs_comm log has messages saying that the process has exceeded the configured nfiles (the open file limit). If so, increase the allowed max open files limit, and restart the pbs_comm daemon.

Message saying NOROUTE to destination xxx:nnn

The "noroute" message shows the destination address and the pbs_comm daemon or endpoint which generated the error. Example:

```
Received noroute to dest ::1:15003, msg:pbs_comm:::1:17001: Dest not found
```

The above message means that the pbs_comm running at address ::1:17001 has responded that the destination address (MoM, in this case) ::1:15003 is not known to it. This means the MoM at localhost:15003 was not started (it is down) and/or did not register its address with this pbs_comm. Check the MoM logs for that MoM, and see whether it was started, and if so, what addresses it registered and to which pbs_comm daemon. These log lines from the pbs_mom logs may be useful:

```
Registering address ::1:15003 to pbs_comm
Registering address 192.168.184.156:15003 to pbs_comm
…
….
Connecting to pbs_comm hostname:port
```

The above messages list the actual pbs_comm daemon that the MoM or any endpoint is connected to, and when it connected. After connection, it registered the endpoint with the addresses as listed in the "Registering address" messages, before the connect message.

Corresponding to the above messages in the endpoint log, (in this case, MoM), there should be messages in the associated pbs_comm daemon's logs, as follows:

```
tfd=14: Leaf registered address ::1:15003
tfd=14: Leaf registered address 192.168.184.156:15003
```

The above messages mean that a connection from socket file descriptor 14 at the pbs_comm daemon received data to register the endpoint with addresses ::1:15003 and 192.168.184.156:15003.

The above messages from the endpoint and the associated pbs_comm daemon tell us whether there are address mismatches, or the endpoints never connected, or connected to the wrong MoMs, or the endpoints are not configured to use TCP.

MoM down/stale on pbsnodes -av output

- Check whether the respective MoM is actually up.
- Check that the MoM that is showing as down is actually pointing to the correct pbs_comm daemon, by checking whether it is the default or PBS_LEAF_ROUTERS is set.
- Check that the pbs_comms that are handling the pbs_server and the MoM in question are running, and that none of them have a system error in their logs such as no files etc.
- Check the connection settings between this pair of pbs_comms is as intended. Check each of the pbs_comm's PBS_COMM_ROUTERS settings.
- Follow a "noroute" message to trace where the "noroute" is originating, and troubleshoot why the route is not being found .

4.5.10 Logging and Errors with TPP

4.5.10.1 Communication Daemon Logfiles

The pbs_comm daemon creates its log files under $PBS_HOME/comm_logs. This directory is automatically created by the PBS installer.

In a failover configuration, this directory is shared as part of the shared PBS_HOME by the pbs_comm daemons running on both the primary and secondary servers. This directory must never be shared across multiple pbs_comm daemons in any other case.

The log filename format is *yyyymmdd* (the same as for other PBS daemons).

Whenever a new log file is opened, the communication daemon logs PBS_LEAF_NAME, PBS_MOM_NODE_NAME, and the hostname. The daemon also logs all network interfaces, listing each interface and all of the hostnames associated with that interface. In addition, it logs the PBS version and the build information.

The log record format is the same as used by other PBS daemons, with the addition of the thread number and the daemon name in the log record. The log record format is as follows:

date-time;event_code;daemon_name(thread number);object_type;object_name;message

An example is as follows:

```
03/25/2014 15:13:39;0d86;host1.example.com;TPP;host1.example.com(Thread 2);Connection from leaf
    192.168.184.156:19331, tfd=81 down
```

4.5.10.2 Messages from Endpoints

`Connected to pbs_comm %s`

Endpoint was able to connect to the named `pbs_comm` daemon.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Connection to pbs_comm %s down`

The endpoint's connection to the specified `pbs_comm` daemon is down.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Connection to pbs_comm %s failed`

The endpoint failed to connect to the specified `pbs_comm` daemon. A system/socket error message may accompany this message.

Log level: *PBSEVENT_ERROR*

`Registering address %s to pbs_comm`

The endpoint logs the list of IP addresses it is registering with the `pbs_comm` daemon.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`sd %d, Received noroute to dest %s, msg:%s`

Specified stream sd (stream descriptor) has received a "noroute" message from the `pbs_comm` daemon indicating that the destination is not known to the `pbs_comm` daemon. An additional message from `pbs_comm` is also printed.

Log level: *PBSEVENT_ERROR*

`Single pbs_comm configured, TPP Fault tolerant mode disabled`

Only one `pbs_comm` daemon was configured, so fault tolerant mode is disabled.

Log level: *PBSEVENT_SYSTEM | PBSEVENT_ADMIN*

4.5.10.3 Messages from Communication Daemons

`tfd=%d: endpoint registered address %s`

Endpoint registered this address.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Connection from leaf %s, tfd=%d down`

The connection from an endpoint just went down.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`pbs_comm %s connected`

Another `pbs_comm` daemon connected to this `pbs_comm` daemon.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`pbs_comm %s accepted connection`

Specified `pbs_comm` daemon accepted connection from this `pbs_comm`.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`pbs_comms should have at least 2 threads`

Number of threads configured for the daemon is too few. There should be a minimum of two threads. The daemon will abort.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

`Received TPP_CTL_NOROUTE for message, %s(sd=%d) -> %s: %s`

The `pbs_comm` daemon received a "noroute" message from a destination endpoint. This means that the destination stream was not found in that endpoint.

Log level: PBSEVENT_ERROR

`Connection from non-reserved port, rejected`

The `pbs_comm` received a connection request from an endpoint or another `pbs_comm` or an endpoint, but since the connection originated from a non-reserved port, it was not accepted.

Log level: PBSEVENT_ERROR

`Failed initiating connection to pbs_comm %s`

This `pbs_comm` daemon failed to initiate a connection to another `pbs_comm`.

Log level: PBSEVENT_ERROR

4.5.10.4 Important Messages from Communication or Other Daemons

`Compression failed`

Compression routine failed. Usually due to memory constraints.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

`Decompression failed`

Decompression routine failed due to bad input data. Usually a transmission/network error.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

`Error %d resolving %s`

There was an error in name resolution of a hostname.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

`Error %d while binding to port %d`

There was an error in binding to the specified port. Usually this means the address is already in use.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

`No reserved ports available`

No more reserved ports are available. Cannot initiate connection to a `pbs_comm` daemon. Not applicable on Windows.

Log level: PBSEVENT_ERROR

`Out of memory <in an operation>`

An out-of-memory condition occurred.

Log level: PBSEVENT_SYSTEM | PBSEVENT_ADMIN | PBSEVENT_FORCE

4.5.10.5 Informational Messages from Communication or Other Daemons

`Initializing TPP transport Layer`

Starting the initialization of the TPP layer: starting threads etc.; creating internal data structures.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`TPP initialization done`

Initialization completed successfully; system ready to transmit data.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Shutting down TPP transport Layer`

TPP was asked to shut down.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Max files allowed = %ld`

Logs the nfiles currently configured.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

`Max files too low - you may want to increase it`

If nfiles is <1024, the `pbs_comm` daemon emits the message. If nfiles configured is <100, the startup aborts. Usually nfiles must be configured to allow the number of connections (usually the number of MoMs) the `pbs_comm` process is going to handle.

Log level: *PBSEVENT_SYSTEM | PBSEVENT_ADMIN*

`Thread exiting, had %d connections`

Each thread in the TPP layer logs the number of connections it was handling. For `pbs_comm`, this is usually the number of MoMs that were handled by each thread. This gives you information useful for deciding when to increase the threads in order to distribute the load.

Log level: *PBSEVENT_DEBUG | PBSEVENT_DEBUG2*

4.6 Ports Used by PBS

PBS daemons listen for inbound connections at specific network ports. These ports have defaults, but can be configured if necessary. PBS daemons use any ports numbered less than 1024 for outbound communication. For PBS daemon-to-daemon communication over TCP, the originating daemon will request a privileged port for its end of the communication.

PBS makes use of fully-qualified host names for identifying jobs and their locations. A PBS installation is known by the host name on which the server is running. The canonical host name is used to authenticate messages, and is taken from the primary name field, `h_name`, in the structure returned by the library call `gethostbyaddr()`. According to the IETF RFCs, this name must be fully qualified and consistent for any IP address assigned to that host.

Port numbers can be set via `/etc/services`, the command line, or in `pbs.conf`. If not set by any of these means, they will be set to the default values. The PBS components and the commands will attempt to use the system `services` file to identify the standard port numbers to use for communication. If the port number for a PBS daemon can't be found in the system file, a default value for that daemon will be used. The server, scheduler, and MoM daemons have startup options for setting port numbers. In the PBS Professional Reference Guide, see "pbs_mom" on page 71, "pbs_sched" on page 105, and "pbs_server" on page 108.

For port settings in `pbs.conf`, see "Contents of Configuration File" on page 369 in the PBS Professional Installation & Upgrade Guide.

The scheduler uses any privileged port (less than 1024) as the outgoing port to talk to the server.

Under Linux, the `services` file is named `/etc/services`.

Under Windows, it is named `%WINDIR%\system32\drivers\etc\services`.

The port numbers listed are the default numbers used by PBS. If you change them, be careful to use the same numbers on all systems. The port number for `pbs_resmon` must be one higher than for `pbs_mom`.

4.6.1 Ports Used by PBS in TPP Mode

The table below lists the default port numbers for PBS daemons in TPP mode:

Table 4-1: Ports Used by PBS Daemons in TPP Mode

Daemon Listening at Port	Port Number	Protocol	Type of Communication
`pbs_server`	*15001*	TPP (TCP)	All communication to server
`pbs_mom`	*15002*	TPP (TCP)	All communication to MoM
`pbs_resmon`	*15003*	TPP (TCP)	Scheduler-MoM resource requests (`pbs_resmon` listens on this port)
`pbs_sched`	*15004*	TPP (TCP)	All communication to scheduler
`pbs_datastore`	*15007*	proprietary	PBS information storage and retrieval
License server	*6200*	proprietary	All communication to license server
`pbs_comm`	*17001*	TPP (TCP)	All communication to `pbs_comm`

4.6.2 Port Settings in pbs.conf

You can set the following in `pbs.conf`:

Table 4-2: Port Parameters in `pbs.conf`

Parameter	Description
PBS_BATCH_SERVICE_PORT	Port server listens on
PBS_BATCH_SERVICE_PORT_DIS	DIS port server listens on
PBS_DATA_SERVICE_PORT	Used to specify non-default port for connecting to data service. Default is 15007.
PBS_MANAGER_SERVICE_PORT	Port MoM listens on
PBS_MOM_SERVICE_PORT	Port MoM listens on
PBS_SCHEDULER_SERVICE_PORT	Port scheduler listens on

4.7 PBS with Multihomed Systems

PBS expects the network to function according to IETF standards. Please make sure that your addresses resolve correctly. You can set host name parameters in pbs.conf to disambiguate addresses for contacting the server, sending mail, delivering output and error files, and establishing outgoing connections.

When setting these parameters, use fully qualified host names where you could have host name collisions, for example master.foo.example.com and master.bar.example.com. See the following sections for details.

Before tackling this section, make sure that you have taken care of everything listed in section 2.1.3, "Name Resolution and Network Configuration", on page 8.

PBS uses only IPv4, so all names must resolve to IPv4 addresses.

4.7.1 Contacting the Server

Use the PBS_SERVER_HOST_NAME parameter in pbs.conf on each host in the complex to specify the FQDN of the server, under these circumstances:

- The host on which the PBS server runs has multiple interfaces and some of these interfaces are limited to a private network that might not be addressable outside of the immediate complex
- The server name to be used in Job IDs needs to be different from the PBS_SERVER parameter. It might become impossible for a client to contact the server where this option is not used or is misconfigured. Take extreme care when using PBS_SERVER_HOST_NAME for this reason.

You can specify the server name with the following order of precedence, highest first:

- Specifying server name at the client
  - Specifying server name at the command line, e.g. pbsnodes -s <server name>
  - Setting the PBS_PRIMARY and PBS_SECONDARY environment variables
  - Setting the PBS_SERVER_HOST_NAME environment variable
  - Setting the PBS_SERVER environment variable
- Setting PBS_PRIMARY and PBS_SECONDARY in pbs.conf
- Setting PBS_SERVER_HOST_NAME in pbs.conf
- Setting PBS_SERVER in pbs.conf

4.7.2 Delivering Output and Error Files

You can specify the host name portions of paths for standard output and standard error for jobs. To specify the host where the job's standard output and error files are delivered, use the PBS_OUTPUT_HOST_NAME parameter in pbs.conf on the server host. It is useful when submission and execution hosts are not visible to each other.

- If the job submitter specifies an output or error path with both file path and host name, PBS uses that path.
- If the job submitter specifies an output or error path containing only a file path:
  - If PBS_OUTPUT_HOST_NAME is set, PBS uses that as the host name portion of the path
  - If PBS_OUTPUT_HOST_NAME is not set, PBS follows the rules in "Default Behavior For Output and Error Files", on page 39 of the PBS Professional User's Guide.
- If the job submitter does not specify an output or error path, PBS uses the current working directory of qsub, following the naming rules in "Default Paths for Output and Error Files", on page 42 of the PBS Professional User's Guide, and appends an at sign ("@") and the value of PBS_OUTPUT_HOST_NAME.

4.7.3 When Installing and Upgrading

During installation or upgrade:

1. When asked whether you want to start the new version of PBS, reply "*no*"
2. Edit pbs.conf to set the desired network address parameters
3. Start the new version of PBS:

   systemctl start pbs

   or

   <path to script>/pbs start

You may see differences in new job IDs. For example, if the prior value of PBS_SERVER was set to the fully qualified host name, the existing jobs will have IDs containing the full hostname, for example *123.server.example.com*. If the current value of PBS_SERVER is a short name, then new jobs will have IDs with the short form of the host name, in this case, *123.server*.

With version 13.0, PBS supports host names up to 255 characters. The format of the job files written by pbs_mom has changed due to this. If there are existing job files during an overlay upgrade, PBS prints a summary message showing the number of job files successfully upgraded and the total number of job files. For each job file that was not successfully upgraded, PBS prints a message that the job file was not successfully upgraded and gives the full path to that job file.

4.7.4 Hostname Parameters in pbs.conf

The following table describes the hostname parameters in the pbs.conf configuration file:

Table 4-3: Hostname Parameters in pbs.conf

Parameter	Description
PBS_LEAF_NAME	Tells endpoint what hostname to use for network. The value does not include a port, since that is usually set by the daemon. By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in /etc/pbs.conf to something else, make sure you set the Mom attribute at vnode creation. TPP internally resolves the name to a set of IP addresses, so you do not affect how pbs_comm works.
PBS_MAIL_HOST_NAME	Optional. Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. See "Specifying Mail Delivery Domain" on page 20 in the PBS Professional Administrator's Guide. Should be a fully qualified domain name. Cannot contain a colon (":").
PBS_MOM_NODE_NAME	Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by gethostname(). If you use the IP address for a vnode name, set PBS_MOM_NODE_NAME=<IP address> in pbs.conf on the execution host. This parameter cannot contain dots unless it is for an IP address.
PBS_OUTPUT_HOST_NAME	Optional. Host to which all job standard output and standard error are delivered. See section 4.7.2, "Delivering Output and Error Files", on page 60. Should be a fully qualified domain name. Cannot contain a colon (":").
PBS_PRIMARY	Hostname of primary server. Overrides PBS_SERVER_HOST_NAME.
PBS_SECONDARY	Hostname of secondary server. Overrides PBS_SERVER_HOST_NAME.
PBS_SERVER	Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in pbs.conf. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER. Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.
PBS_SERVER_HOST_NAME	Optional. The FQDN of the server host. Used by clients to contact server. See section 4.7.1, "Contacting the Server", on page 60. Should be a fully qualified domain name. Cannot contain a colon (":").

5 Initial Configuration

After you have installed PBS Professional, perform the following steps:

5.1 Validate the Installation

- Check files and directories: To validate the installation of PBS Professional, at any time, run the `pbs_probe` command. It will review the installation (installed files, directory and file permissions, etc) and report any problems found. For details, see "pbs_probe" on page 81 of the PBS Professional Reference Guide.

  The `pbs_probe` command is not available under Windows.
- Check PBS version. Use the `qstat` command to find out what version of PBS Professional you have:

  ```
  qstat -fB
  ```
- Check hostname resolution:
  - At the server, use the `pbs_hostn` command with the name of each host in the complex. This should complain if hostname resolution is not working correctly. See "pbs_hostn" on page 63 of the PBS Professional Reference Guide.
  - Make sure that rcp and/or scp work correctly. They must work outside of PBS before PBS can use them Run rcp and/or scp between machines in the complex to make sure they work. If there are problems, see section 2.1.3, "Name Resolution and Network Configuration", on page 8.
- Windows: turn firewall off for execution hosts: see "Windows Firewall" on page 395 in the PBS Professional Administrator's Guide

5.2 Support PBS Features

- Configure PBS inter-daemon communication. See Chapter 4, "Communication", on page 45.
- Define PATHs for users: set paths for all users to include PBS commands and man pages. For paths, see section 3.5.2.2, "Setting Installation Parameters", on page 25. Administrator commands are in `PBS_EXEC/sbin`, and user commands are in `PBS_EXEC/bin`. Man pages are `PBS_HOME/man`.
- Support X forwarding:
  - Edit each MoM's PATH variable to include the directory containing the `xauth` utility.
  - Add the path to the `xauth` binary to each MoM's `pbs_environment` file. For example, if you start with this path:

    ```
    /bin:/user/bin
    ```

    and the `xauth` utility is here:

    ```
    /usr/bin/X11/xauth
    ```

    The entry in the `pbs_environment` file would be the following:

    ```
    PATH=/bin:/usr/bin:/usr/bin/X11
    ```
  - In the `ssh_config` file for each machine that will use X forwarding, put this line:

    ```
    ForwardX11Trusted yes
    ```

X forwarding is not available under Windows.

- Allow interactive jobs. For interactive jobs, MoMs establish a connection back to the submission host:
  - Make sure that the ephemeral port range in your firewall is open (make sure that MoMs can connect to an ephemeral port on submission hosts). Check your OS documentation for the correct range.
  - Allow interactive jobs under Windows: see "Allowing Interactive Jobs on Windows" on page 534 in the PBS Professional Administrator's Guide
- Create and configure vnodes: see "About Vnodes: Virtual Nodes" on page 37 in the PBS Professional Administrator's Guide
- Create and configure queues: see "Queues" on page 21 in the PBS Professional Administrator's Guide
- Manage cgroups and cpusets: see "Configuring and Using PBS with Cgroups" on page 571 in the PBS Professional Administrator's Guide
- Configure resources: see "Using PBS Resources" on page 229 in the PBS Professional Administrator's Guide
- Set up resource limits: see "Managing Resource Usage" on page 287 in the PBS Professional Administrator's Guide
- Define scheduling policy: see "Scheduling" on page 53 in the PBS Professional Administrator's Guide
- Create hooks: see the PBS Professional Hooks Guide.
- Integrate with an MPI: see "Using MPI with PBS" on page 451 in the PBS Professional Administrator's Guide
- Use containers: see "Configuring PBS for Containers" on page 615 in the PBS Professional Administrator's Guide
- Use provisioning: see "Provisioning" on page 327 in the PBS Professional Administrator's Guide
- Set up failover: see "Failover" on page 399 in the PBS Professional Administrator's Guide
- Set up checkpointing: see "Checkpoint and Restart" on page 418 in the PBS Professional Administrator's Guide
- Minimize communication problems: see "Preventing Communication and Timing Problems" on page 441 in the PBS Professional Administrator's Guide
- Manage security features, including authentication and encryption: see "Security" on page 359 in the PBS Professional Administrator's Guide
  - Required on Windows: set up encryption via TLS. See "Encrypting PBS Communication" on page 387 in the PBS Professional Administrator's Guide.
- Set up desired file transfer mechanism: see "Setting File Transfer Mechanism" on page 559 in the PBS Professional Administrator's Guide
- Configure where PBS components will put temporary files: see "Temporary File Location for PBS Components" on page 567 in the PBS Professional Administrator's Guide

6 Upgrading

This chapter shows how to upgrade from a previous version of PBS Professional. If PBS Professional is not installed on your system, go instead to Chapter 3, "Installation", on page 19.

6.1 Types of Upgrades

There are two types of upgrades available for PBS Professional:

overlay upgrade

Installs the new PBS_HOME and PBS_EXEC on top of the old ones. Jobs cannot be running during an overlay upgrade.

migration upgrade

You install the new PBS_HOME and PBS_EXEC in a separate location from the old PBS_HOME and PBS_EXEC. The new PBS_HOME can be in the standard location if the old version has been moved. Jobs are moved from the old server to the new one, and cannot be running during the move.

6.1.1 Choosing Upgrade Type on Linux

Usually, you can do an overlay upgrade on Linux systems. However, the following require migration upgrades:

- When moving between hosts
- When upgrading from an open-source version of PBS Professional
- When certain European or Japanese characters are stored in the data store

For specific upgrade recommendations and updates, see the Release Notes.

6.1.2 Upgrading Existing All-Windows Complex

If your existing complex runs a PBS server on a Windows host, "upgrading" means doing a fresh install for the server/schedulers/comms, and upgrading your Windows MoMs. You cannot preserve any jobs in any state during the upgrade. See Chapter 6, "Upgrading from an All-Windows Complex", on page 125.

6.1.3 Upgrading from Windows/Linux Combination to Windows/Linux Combination

Upgrading on Windows/Linux requires a migration upgrade; see section 6.8, "Upgrading a Windows/Linux Complex", on page 109.

6.2 Differences from Previous Versions

6.2.1 New Way to Manage Vnode Attributes

As of version 2020.1, PBS can use the cgroups hook to manage cpusets and create child vnodes on multi-vnode machines.

If you use the cgroups hook on a host where you want to set the sharing attribute or define the placement sets, you can use an exechost_startup hook or a Version 2 configuration file for this, but make sure that you refer precisely to the vnodes that were created by the cgroups hook. Do not accidentally create new vnodes by defining them (that is, using a vnode name unknown to the cgroups hook).

6.2.2 New Scheduler Attributes

The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in `$PBS_HOME/sched_priv/sched_config` in older versions of PBS. They are now scheduler attributes with the same names and formats. Schedulers now have a log_events attribute that replaces the log_filter scheduler parameter. You use `qmgr` to set these attributes.

6.2.3 Using RPM Instead of `INSTALL` (14.2)

You use RPM or another native package manager such as `yum` or `zypper` to install PBS, instead of the `INSTALL` script.

6.2.4 Using `systemd` Instead of Start/stop Script (14.2)

PBS uses `systemd` instead of the PBS start/stop script on Linux platforms that support `systemd`. On Linux platforms that do not support `systemd`, PBS still uses the start/stop script. You will see a choice of instructions for starting or stopping PBS.

6.2.5 Automatic Upgrade of Database (13.0)

The PBS installer automatically upgrades the database used by PBS for its data store.

6.2.6 Installing Communication Daemon (13.0)

As of 13.0, PBS uses a new daemon, `pbs_comm`, for communication. One communication daemon is automatically installed on each server host, and all daemons automatically connect to it. If you require additional communication daemons, you must install and configure them. See section 4.5.3, "Adding Communication Daemons", on page 50.

6.2.7 Default Location of `PBS_EXEC` and `PBS_HOME`

`PBS_EXEC` is the directory that contains the PBS binaries. The default location for `PBS_EXEC` is `/opt/pbs`. `PBS_HOME` is the directory where PBS information is stored. The default location for `PBS_HOME` is `/var/spool/pbs`.

6.2.8 Use PBS Start Script or `systemd` During Overlay Upgrade

During an overlay upgrade, you must start the PBS server using `systemd` on platforms that support it, or the start/stop script where `systemd` is not supported, so that the server is initialized correctly. The instructions in this manual for overlay upgrading specify using `systemd` or the start script.

6.3 Caveats and Advice

6.3.1 Licensing

PBS starts faster if you install, configure, and start the Altair license server before starting PBS. We recommend that you follow the steps for installing and starting the license server before upgrading. See the *Altair License Management System Installation and Operations Guide*, available at www.altair.com. Do not attempt to use any license server other than the Altair license server.

6.3.2 Making Time to Upgrade

If you want to avoid having to work around running jobs when you perform an upgrade, you can set PBS up so that there are no running jobs when you want to do the upgrade. Follow these steps:

1. Figure out how much walltime the longest-running jobs are likely to need, e.g. two weeks
2. Pick a time further into the future than that, e.g. 3 weeks
3. On all PBS hosts, create dedicated time or a reservation for the amount of time you think the upgrade will require, e.g. a day
   - You can use a dedicated time slot, making it so that no jobs will be scheduled for that dedicated time. The system can be shut down all at once at the start of the dedicated time. See "Dedicated Time" on page 125 in the PBS Professional Administrator's Guide.
   - You can create a reservation that reserves an entire host by using `-l place=exclhost`. The following reservation creates a reservation for the host mars, from 10am to 10pm:

     ```
     pbs_rsub -R 1000 -D 12:00:00 -l select = host=mars -l place=exclhost
     ```

     For more on creating reservations, see "pbs_rsub" on page 96 of the PBS Professional Reference Guide.

6.3.3 Upgrading Database

PBS automatically upgrades the database used for its data store. If the process of upgrading the database fails, you must restore the database to its pre-upgrade state in order to upgrade PBS.

6.3.4 Data Service Account Must Be Same as When Installed

The data service account you use when upgrading PBS must be the same as when you installed the old version of PBS, otherwise the upgrade will fail. The workaround is to change the data service user ID to the ID used for installation of the old PBS data service, perform the upgrade, then change the ID back.

1. Identify the user who originally created the data store:
   a. Log in to the data store:

      su - <data service account> -s /bin/sh -c "LD_LIBRARY_PATH=$PBS_EXEC/pgsql/lib $PBS_EXEC/pgsql/bin/psql -U <data service account> -p <data service port> -d pbs_datastore"

      The default data service port is *15007*

      The default data service account is *pbsdata*

   b. Run a query to get the list of users in the database:

      ```
      pbs_datastore=# select pg_authid.oid, rolname from pg_authid;
      oid | rolname
      ------+--------
      10 | pbsdata
      16541 | <username>
      (2 rows)
      ```

   c. Find the original user who created the database:

      ```
      pbs_datastore=# select pg_authid.oid, rolname from pg_authid where pg_authid.oid=10;
      oid | rolname
      ----+--------
      10 | pbsdata
      (1 row)
      ```

2. Exit the database
3. Create the original data service account in system if it is not available.
4. Update the current database user to the original data service account.

   pbs_ds_password -C <original username>

5. Perform the overlay upgrade
6. Reset the current database user to desired username:

   pbs_ds_password -C <later username>

See "Setting Data Service Account Name and Password" on page 558 in the PBS Professional Administrator's Guide.

6.3.5 Updating Hooks for New Python Version

As of version 19.4.1, PBS uses Python 3.6, so you need to make sure that your hooks and their configuration files are compatible with Python 3.6. To do this, you export each hook and configuration file in ASCII format, make sure it is compliant with Python 3.6, then import the 3.6-ready hook and configuration file in ASCII format. We include a link to a site with instructions for making your Python code compatible with version 3.6. We include all of these steps in the instructions.

6.3.6 New Server Requires New MoMs

As of version 12.0, you must **not** attempt to run a newer server with older MoMs. You must start the new server only when all MoMs have been upgraded. Follow the steps in this chapter.

6.3.7 Do Not Unset default_chunk.ncpus

Do not unset the value for the default_chunk.ncpus server attribute. It is set by the server to *1*. You can set it to another non-zero value, but a value of *0* will produce undefined behavior. When the PBS server initializes and the default_chunk server attribute has not been specified during a prior run, the server will internally set the following:

```
default_chunk.ncpus=1
```

This ensures that each "chunk" of a job's select specification requests at least one CPU.

If you explicitly set the default_chunk server attribute, that setting will be retained across server restarts.

6.3.8 Unset PBS_EXEC Environment Variable

Make sure that the PBS_EXEC environment variable is unset.

6.3.9 Saving and Re-creating Vnode Configuration

For an overlay upgrade, you do not need to save and re-create vnodes. For a migration upgrade, you can save your vnode configuration and re-create it using this sequence:

```
qmgr -c 'print node @default' > nodes.new
```

<clean up nodes.new>

```
qmgr < nodes.new
```

Why clean up nodes.new before reading it back in?

- PBS (the cgroups hook or MoM) should create all child vnodes (vnodes that are not parent vnodes). If you create these child vnodes using qmgr, you can end up with duplicate vnode objects.
- The state attribute and the arch, and host, and vnode resources are set automatically while creating vnodes. Do not set them explicitly. Doing so can get you into trouble especially if you are changing how hostname resolution works.
- The qmgr command overrides resource settings in Version 2 configuration files. If you use qmgr to set vnode resources, you can't set them later in Version 2 configuration files.
- MoM reports mem, vmem and ncpus. You can use qmgr to set these if they need to be explicitly set; otherwise, don't include these lines in nodes.new.
- Leave only the creation lines for parent vnodes and any resources you want managed on the server side through qmgr.

We include this step in the upgrading instructions; we explain why here.

6.3.10 Upgrading with Failover

If you are upgrading and using failover, do not start the new secondary server until the new primary has finished starting.

If your secondary server has a STONITH script, before you perform an upgrade, prevent the STONITH script from running by setting its permissions to 0644. After the upgrade, you can set the permissions back to 0755. We include these steps in the upgrade instructions.

6.4 Introduction to Upgrading Under Linux

When you get your new version of PBS, unpack it (unzip, untar) as a non-privileged user. When you follow the upgrading instructions below, all of the steps should be performed as root.

6.4.1 Directories

The location of PBS_HOME is specified in the file /etc/pbs.conf, but defaults to /var/spool/pbs if not specified. The default for PBS_EXEC is /opt/pbs. You can specify a non-default location for PBS_EXEC via the --prefix option to rpm when installing the new PBS.

6.4.2 Upgrading on Multiple Machines

Instead of running the installer by hand on each machine, you can use a command such as pdsh. The one-line format for a non-default install is:

```
PBS_SERVER=<server name> PBS_HOME=<new/home/location/pbs> rpm -i --prefix <new/exec/location/pbs>
    pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

6.4.3 Upgrading on a Machine Running the Cpuset MoM

Machines running the cpuset MoM typically include HPE MC990X, HPE Superdome Flex, or HPE 8600.

When upgrading on a machine running the cpuset MoM, follow the instructions in section 6.6, "Overlay Upgrade on One or More Machines Running Cpuset MoM", on page 82.

6.5 Overlay Upgrade Under Linux

The steps in this section are for machines that are not running a cpuset MoM. Machines running the cpuset MoM typically include HPE MC990X, HPE Superdome Flex, or HPE 8600. When upgrading on a machine running the cpuset MoM, follow the instructions in section 6.6, "Overlay Upgrade on One or More Machines Running Cpuset MoM", on page 82.

The following commands must be run as root.

6.5.1 Prevent Jobs From Being Started

Prevent the scheduler(s) from starting jobs. Set scheduling to *false* for the default scheduler and each multisched:

```
qmgr -c "set sched <scheduler name> scheduling = false"
```

6.5.2 Allow Running Jobs to Finish, or Requeue Them

You cannot perform an upgrade while jobs are running. Either let running jobs finish, or requeue them. (You can also delete them.)

To requeue any running jobs:

1. List the jobs. This will list some jobs more than once. You only need to requeue each job once:
   pbsnodes <hostname> | grep jobs
2. Requeue the jobs:

   qrerun <job ID> <job ID> ...

To kill the jobs:

1. List the jobs. This will list some jobs more than once. You only need to kill each job once:
   pbsnodes <hostname> | grep jobs
2. Use the `qdel` command to kill each job by job ID:

   qdel <job ID> <job ID> ...

To drain the host, wait until any running jobs have finished.

Make sure that there are no old job files on any execution hosts. Remove any of the following:

```
$PBS_HOME/mom_priv/jobs/*.JB
```

6.5.3 Disable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, disable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.5.4 Disable STONITH Script

If your secondary server has a STONITH script, prevent the STONITH script from running by setting its permissions to 0644.

6.5.5 Unwrap Any Wrapped MPIs

If you used the `pbsrun_wrap` mechanism with your old version of PBS, you must first unwrap any MPIs that you wrapped. This includes MPICH-GM, MPICH-MX, MPICH2, etc. You can re-wrap your MPIs after upgrading PBS.

For example, you can unwrap an MPICH2 MPI:

```
# pbsrun_unwrap pbsrun.mpich2_64
```

See "pbsrun_unwrap" on page 50 of the PBS Professional Reference Guide.

6.5.6 Save Execution Host Configuration Information

On each PBS execution host, copy the Version 1 and Version 2 configuration files:

1. Make a backup directory:
   ```
   mkdir /tmp/pbs_mom_backup
   ```
2. Make a copy of the Version 1 configuration file:
   ```
   cp $PBS_HOME/mom_priv/config /tmp/pbs_mom_backup/config.backup
   ```
3. Make a copy of the Version 2 configuration files:
   ```
   mkdir /tmp/pbs_mom_backup/mom_configs
   pbs_mom -s list | egrep -v '^PBS' | while read file
   do
     pbs_mom -s show file > /tmp/pbs_mom_backup/mom_configs/$file
   done
   ```

6.5.7 Save Hooks and Hook Configuration Files

Save your hooks and hook configuration files in ASCII format so you can check them and import them later. The new version of PBS includes a new pbs_cgroups hook with a new configuration file. If you use the cgroups hook, you must use the new hook and configuration file, but you may want to modify the configuration file, so if you have made any changes to your existing pbs_cgroups hook configuration file, you need to save it before you upgrade. Later, you can use the saved information to modify the new configuration file.

For **each** hook:

1. Save the hook. Export the hook:
   ```
   # qmgr -c 'export hook <hook name> application/x-python default /tmp/<hook name>.old2.7'
   ```
2. Save your hook configuration file. Export the configuration file:
   ```
   # qmgr -c 'export hook <hook name> application/x-config default /tmp/<hook name>.configcheck'
   ```

6.5.8 Update Hooks and Hook Configuration Files for New Python

PBS 19.4.1 and later uses Python 3.6, so if you have not already, update all of your site-defined hooks (not the built-in hooks) to Python 3.6. For **each** hook except for the pbs_cgroups hook:

1. Update your hook to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your updated hook file differently; use something like "/tmp/<hook name>.new3.6"
2. Check that the contents of the configuration file are correct for Python 3.6

6.5.9 Shut Down Your Existing PBS

1. Shut down the server(s), default scheduler, and MoMs:
   ```
   qterm -t immediate -m -s -f
   ```

If your server is not running in a failover environment, the "-f" option is not required.

2. Shut down any multischeds. On each multisched host:

   a. Find the PID you want:

      ps -ef | grep pbs_sched

      For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

   b. Stop the scheduler or multisched:

      kill <multisched PID>

3. On the server host and any other comm hosts, shut down the communication daemon:

   systemctl stop pbs

   or

   <path to script>/pbs stop

4. Verify that PBS daemons are not running in the background:

   ps -ef | grep pbs

   If you see the pbs_server, pbs_sched, pbs_mom, or pbs_comm process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

   kill -9 <daemon PID>

6.5.10 Back Up Existing PBS Files

On each PBS host, make a tar file of the PBS_HOME and PBS_EXEC directories.

1. Make a backup directory:

   mkdir /tmp/pbs_backup

2. Make a tar file of PBS_HOME:

   cd $PBS_HOME/..
   tar -cvf /tmp/pbs_backup/PBS_HOME_tarbackup.tar $PBS_HOME

3. Make a tar file of PBS_EXEC:

   cd $PBS_EXEC/..
   tar -cvf /tmp/pbs_backup/PBS_EXEC_tarbackup.tar $PBS_EXEC

4. Make a copy of your configuration file:

   cp /etc/pbs.conf /tmp/pbs_backup/pbs.conf.backup

5. If this is a scheduler or multisched host, make a copy of the scheduler's directory to modify:

   cp -r $PBS_HOME/sched_priv /tmp/pbs_backup/sched_priv.work

   or

   cp -r $PBS_HOME/sched_priv_<multisched name> /tmp/pbs_backup/sched_priv_<multisched name>.work

6.5.11 Install the New Version of PBS

For an overlay upgrade, you install the new PBS in the same location as the existing PBS. The default location for PBS_HOME is /var/spool/pbs, and the default for PBS_EXEC is /opt/pbs.

6.5.11.1 Install New PBS Server(s)

Install the new version of PBS without uninstalling the previous version. If you are using failover, do not upgrade the primary and secondary servers simultaneously. Upgrade the primary first, then once that is complete, upgrade the secondary.

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25.

   If you are using failover, pay special attention to your configuration parameters, including PBS_HOME and PBS_MOM_HOME, when installing the server sub-package on the secondary server host. See "Configuring the pbs.conf File for Failover" on page 409 in the PBS Professional Administrator's Guide.
4. Install the server sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:

     rpm -i <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   - When upgrading from 14.2 or a later version:

     rpm -U <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm

Do **not** start PBS now.

6.5.11.2 Install New PBS MoMs

Install the new version of PBS on all execution hosts without uninstalling the previous version:

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
4. Install the execution sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:

     rpm -i <path/to/execution sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   - When upgrading from 14.2 or a later version:

     rpm -U <path/to/execution sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm

Do **not** start PBS now.

6.5.11.3 Install New PBS Client Commands

Install the new version of PBS on all hosts without uninstalling the previous version:

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25.
4. Install the client command sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:
     rpm -i <path/to/client command sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   - When upgrading from 14.2 or a later version:
     rpm -U <path/to/client command sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm

6.5.11.4 Install New PBS Communication Daemons

If you are installing a communication daemon on a communication-only host, install the server-scheduler-communication-MoM sub-package, and disable the server, scheduler, and MoM on that host. (MoM is disabled by default.) Install the new version of PBS without uninstalling the previous version.

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25
4. Disable the server, scheduler, and MoM. In pbs.conf:

   ```
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_MOM=0
   ```

5. Install the server sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:
     rpm -i <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   - When upgrading from 14.2 or a later version:
     rpm -U <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm

Do **not** start PBS now.

6.5.12 Prepare Configuration File for New Scheduler(s)

If you were running one or more multischeds with your old version of PBS, make sure you update their configuration files along with that of the default scheduler. Note that the preempt_order, preempt_prio, preempt_queue_prio, preempt_sort, and log_events scheduler attributes are new; some were parameters in sched_config with the same names. In a later step (after the server is running), you will use qmgr to set the attributes. We explicitly list the step; don't worry.

For each scheduler:

1. Make a copy of the new sched_config, which is in PBS_EXEC/etc/pbs_sched_config.
   cp $PBS_EXEC/etc/pbs_sched_config $PBS_EXEC/etc/pbs_sched_config.new
2. Update PBS_EXEC/etc/pbs_sched_config**.new** with any modifications that were made to the old %PBS_HOME/sched_priv/sched_config or %PBS_HOME/sched_priv_<multisched name>/sched_config. This is saved in the backup directory /tmp/pbs_backup/sched_priv.work.
3. If you were using vmem at the queue or server level before the upgrade, then after upgrading you must add vmem to the resource_unset_infinite sched_config option. Otherwise jobs requesting vmem will not run.
4. Move PBS_EXEC/etc/pbs_sched_config.new to the correct name and location, i.e. $PBS_HOME/sched_priv/sched_config or $PBS_HOME/sched_priv_**<multisched name>**/sched_config:

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv/sched_config

   or

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv_<multisched name>/sched_config

6.5.13 Update Holidays File

Make sure your new holidays file is up to date.

6.5.14 Modify the New PBS Configuration File

Your new pbs.conf needs to reflect any changes that you made to the old file.

If you will use failover:

- Edit pbs.conf on the primary server host to include failover settings. See "Configuring Failover For the Primary Server on Linux" on page 411 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf.
- Edit pbs.conf on the secondary server host to include failover settings. See "Configuring Failover For the Secondary Server on Linux" on page 413 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf. You can use the following steps:
  - Copy pbs.conf from primary to secondary
  - Modify pbs.conf on secondary for failover (PBS_START_SCHED = *0*)
- Edit pbs.conf on all execution and client hosts to include failover settings. See "Configuring Failover For Execution and Client Hosts on Linux" on page 414 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf.

If you will not use failover, edit pbs.conf on each host to include changes that you made to the old pbs.conf.

6.5.15 Configure Communication Daemons

If you are using additional communication daemons (more than those automatically installed on server hosts), configure them. See section 4.5.3.2, "Configuring Communication Daemons", on page 50.

6.5.16 Start Then Stop New PBS Servers (If Using Failover)

6.5.16.1 Start New Servers

If you are not using failover, skip this step. If you are using failover, this pair of start and stop steps really is necessary. Bear with us.

1. If you will run a MoM on each server host, disable MoM start in `pbs.conf`, so that it contains this:
   `PBS_START_MOM=0`
2. Start PBS on the primary server host:

   `systemctl start pbs`

   or

   `<path to init.d>/init.d/pbs start`
3. Once the primary is finished starting, start PBS on the secondary server host:

   `systemctl start pbs`

   or

   `<path to init.d>/init.d/pbs start`

6.5.16.2 Stop the Servers

If you are not using failover, skip this step.

1. On the primary server host:
   a. Stop PBS:

   `systemctl stop pbs`

   or

   `<path to init.d>/init.d/pbs stop`

   b. If a MoM is installed, enable it by setting `PBS_START_MOM=1` in `pbs.conf`
2. On the secondary server host:
   a. Stop PBS:

   `systemctl stop pbs`

   or

   `<path to init.d>/init.d/pbs stop`

   b. If a MoM is installed, enable it by setting `PBS_START_MOM=1` in `pbs.conf`

6.5.17 Start New PBS MoMs, Schedulers, Servers, and Comms

6.5.17.1 Start PBS on Execution Hosts

On each execution host, first update PBS_HOME by running the start/stop script or `systemctl start`, then start the MoMs:

1. Prevent the script from starting MoMs by setting PBS_START_MOM=0 in pbs.conf
2. Start PBS:

   systemctl start pbs

   or

   <path to init.d>/init.d/pbs start
3. Stop PBS:

   systemctl stop pbs

   or

   <path to init.d>/init.d/pbs stop
4. Enable starting MoMs by setting PBS_START_MOM=1 in pbs.conf
5. Start MoM:

   $PBS_EXEC/sbin/pbs_mom

6.5.17.2 Start PBS on Server Hosts

If failover is configured, start PBS on the primary server host before the secondary.

1. Prevent the script from starting MoMs by setting PBS_START_MOM=0 in pbs.conf
2. Start PBS on the primary server host:

   systemctl start pbs

   or

   <path to init.d>/init.d/pbs start
3. Once the primary is finished starting, start PBS on the secondary server host:

   systemctl start pbs

   or

   <path to init.d>/init.d/pbs start
4. If a MoM will run on the server host(s):
   a. Enable starting MoMs by setting PBS_START_MOM=1 in pbs.conf
   b. Start MoM:

      $PBS_EXEC/sbin/pbs_mom

6.5.17.3 Restart Multischeds

To start a multisched, call pbs_sched and specify the name and port you already gave it. For each multisched:

pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

6.5.17.4 Start PBS on Communication-only Hosts

Start PBS on any communication-only hosts. On each communication-only host, type:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.5.18 Import and Configure Hooks

Make sure you do not overwrite the new pbs_cgroups hook or its configuration file by importing the old ones. Instead, use the saved information from your old hook to modify the new hook and configuration file.

6.5.18.1 Import Old Hooks Except for Cgroups Hook

1. Do not import your old pbs_cgroups hook. Import your other hooks and their configuration files. For each hook **except** for pbs_cgroups:

```
# qmgr -c 'import hook <hook name> application/x-python default  /tmp/<hook name>.new3.6'
# qmgr -c 'import hook <hook name> application/x-config default  /tmp/<hook name>.configcheck'
```

6.5.18.2 Modify Cgroups Hook Configuration File

If you will use the cgroups hook:

1. Export the cgroups hook configuration file to pbs_cgroups.json:

```
# qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
```

2. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

```
# mount | grep cgroup | grep memory
```

   If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

```
"cgroup": {
  ...
"memory": {
"enabled": false,
```

3. If you made changes to the old cgroups configuration file, you may want to make those changes in the new configuration file. Use the information saved in /etc/pbs_cgroups.old2.7

4. Import the modified configuration (make sure you use "**x-config**"):

```
# qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
```

6.5.18.3 Enable Cgroups Hook

If you will use the cgroups hook, enable the pbs_cgroups hook:

```
qmgr -c "set hook pbs_cgroups enabled=true"
```

6.5.18.4 Write and Deploy New Hooks

If you have written new hooks for the new version of PBS, deploy them now. See the *PBS Professional Hooks Guide*.

6.5.18.5 Restart MoMs

On each execution host, restart MoM :

```
ps -eaf | grep pbs_mom
kill <MoM PID>
/opt/pbs/sbin/pbs_mom
```

6.5.19 Set License Location Server Attribute

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

6.5.20 Configure Sharing and Placement Sets

6.5.20.1 Configuration with Cgroups Hook

As of version 2020.1, the cgroups hook creates the child vnodes on a multi-vnode machine if you set vnode_per_numa_node to *true*; in this case, it is important that any Version 2 configuration files refer only to these vnodes. Use Version 2 configuration files only to set the sharing attribute and optionally to set resources that will be used for placement sets. The default value for the sharing attribute of the vnodes is "sharing=default_shared". You can change this, for example to "sharing=default_excl".

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs" that has MoM's list of vnodes; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide

   ```
   # pbsnodes -av | awk -F'=' '{printf "%s:\tsharing = default_excl\n", $2}' > vnodedefs
   ```

2. Edit the file to reflect what you want for the sharing attribute and placement sets. Use the information saved in **/tmp/pbs_mom_backup/mom_configs/** in step "Save Execution Host Configuration Information" on page 72

3. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

4. Restart pbs_mom:

   ```
   # pbs -eaf | grep pbs_mom
   # kill <MoM PID>
   # /opt/pbs/sbin/pbs_mom
   ```

6.5.20.2 Configuration without Cgroups Hook

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs"; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide
2. Create your new Version 2 configuration file and name it for example "vnodedefs":

```
# pbs_mom -s insert vnodedefs vnodedefs
```

3. Restart pbs_mom:

```
# pbs -eaf | grep pbs_mom
# kill <MoM PID>
# /opt/pbs/sbin/pbs_mom
```

6.5.21 Set New Scheduler Attributes

For the default scheduler and all multischeds:

- The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
- The scheduler's log_filter configuration parameter is **obsolete**. The scheduler's log filter now uses the same bitmask system as the other daemons. The new default value is 767. Use qmgr to set the scheduler's log_events attribute to the value you want. See "Specifying Scheduler Log Events" on page 548 in the PBS Professional Administrator's Guide.

6.5.22 Re-wrap Any MPIs

If you want any wrapped MPIs, wrap them. See "Integration by Wrapping" on page 455 in the PBS Professional Administrator's Guide.

6.5.23 Enable STONITH Script

If your secondary server has a STONITH script, allow the STONITH script to run by setting its permissions to 0755.

6.5.24 Enable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, enable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.5.25 Enable Scheduling

If you disabled scheduling earlier, enable it for the default scheduler and any multischeds:

```
qmgr -c 'set sched <scheduler name> scheduling = true'
```

6.5.26 Shut Down and Restart Servers

1. Shut down both servers:
   ```
   qterm -f
   ```
2. Restart PBS on the server hosts. On each server host, primary first:
   ```
   systemctl start pbs
   ```
   or
   ```
   <path to init.d>/init.d/pbs start
   ```

6.5.27 Removing Old PBS

If you decide to remove the old version of PBS after upgrading, **be sure to use the `--noscripts` option** when using `rpm -e`. Using `rpm -e` without this option, even on an older package than the one you are currently using, will cause any currently running PBS daemons to shut down, and will also remove the system V init and/or systemd service startup files. This will prevent PBS daemons from starting automatically at system boot time. If you wish to remove an older RPM without these effects, use `rpm -e --noscripts`.

6.6 Overlay Upgrade on One or More Machines Running Cpuset MoM

Machines running the cpuset MoM typically included HPE MC990X, HPE Superdome Flex, or HPE 8600, for versions of PBS before 2020.1.

As of 2020.1, we no longer provide `pbs_mom.cpuset`; instead, we use standard `pbs_mom`, and the cgroups hook manages the cpusets for jobs. We include the instructions on making the change from the cpuset MoM to the cgroups hook below.

You must run the following commands as root.

6.6.1 Prevent Jobs From Being Started

Prevent the scheduler(s) from starting jobs. Set scheduling to *false* for the default scheduler and each multisched:

```
qmgr -c 'set sched <scheduler name> scheduling = false'
```

6.6.2 Allow Running Jobs to Finish, or Requeue Them

You cannot perform an upgrade while jobs are running. Either let running jobs finish, or requeue them. (You can also delete them.)

To requeue any running jobs:

1. List the jobs. This will list some jobs more than once. You only need to requeue each job once:
   ```
   pbsnodes <hostname> | grep jobs
   ```
2. Requeue the jobs:
   ```
   qrerun <job ID> <job ID> ...
   ```

To kill the jobs:

1. List the jobs. This will list some jobs more than once. You only need to kill each job once:
   pbsnodes <hostname> | grep jobs
2. Use the qdel command to kill each job by job ID:

   qdel <job ID> <job ID> ...

To drain the host, wait until any running jobs have finished.

Make sure that there are no old job files on any execution hosts. Remove any of the following:

```
$PBS_HOME/mom_priv/jobs/*.JB
```

6.6.3 Disable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, disable cloud bursting. See the *Altair Control Administrator's Guide*.

6.6.4 Disable STONITH Script

If your secondary server has a STONITH script, prevent the STONITH script from running by setting its permissions to 0644.

6.6.5 Unwrap Any Wrapped MPIs

If you used the pbsrun_wrap mechanism with your old version of PBS, you must first unwrap any MPIs that you wrapped. This includes MPICH-GM, MPICH-MX, MPICH2, etc. You can re-wrap your MPIs after upgrading PBS.

For example, you can unwrap an MPICH2 MPI:

```
# pbsrun_unwrap pbsrun.mpich2_64
```

See "pbsrun_unwrap" on page 50 of the PBS Professional Reference Guide.

6.6.6 Save Execution Host Configuration Information

On each PBS execution host, copy the Version 1 and Version 2 configuration files:

1. Make a backup directory:
   mkdir /tmp/pbs_mom_backup
2. Make a copy of the Version 1 configuration file:

   cp $PBS_HOME/mom_priv/config /tmp/pbs_mom_backup/config.backup
3. Make a copy of the Version 2 configuration files:

```
mkdir /tmp/pbs_mom_backup/mom_configs
pbs_mom -s list | egrep -v '^PBS' | while read file
do
    pbs_mom -s show file > /tmp/pbs_mom_backup/mom_configs/$file
done
```

6.6.7 Save Hooks and Hook Configuration Files

Save your hooks and hook configuration files in ASCII format so you can check them and import them later. The new version of PBS includes a new pbs_cgroups hook with a new configuration file. You must use the new hook and configuration file, but you may want to modify the configuration file, so if you have made any changes to your existing pbs_cgroups hook configuration file, you need to save it before you upgrade. Later, you can use the saved information to modify the new configuration file.

For **each** hook:

1. Save the hook. Export the hook:
   ```
   # qmgr -c 'export hook <hook name> application/x-python default /tmp/<hook name>.old2.7'
   ```
2. Save your hook configuration file. Export the configuration file:
   ```
   # qmgr -c 'export hook <hook name> application/x-config default /tmp/<hook name>.configcheck'
   ```

6.6.8 Update Hooks and Hook Configuration Files for New Python

PBS 19.4.1 and later uses Python 3.6, so if you have not already, update all of your site-defined hooks (not the built-in hooks) to Python 3.6. For **each** hook except for the pbs_cgroups hook:

1. Update your hook to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your updated hook file differently; use something like "/tmp/<hook name>.new3.6"
2. Check that the contents of the configuration file are correct for Python 3.6

6.6.9 Remove Old PBS Configuration and Resource Conflicts

1. Ensure that each cpuset MoM host has its values for resources_available.(mem|vmem|ncpus) unset:
   ```
   Qmgr: unset node <hostname>  resources_available.mem
   Qmgr: unset node <hostname>  resources_available.ncpus
   Qmgr: unset node <hostname>  resources_available.vmem
   ```
2. Remove the old PBS reserved files. On each execution host:
   ```
   # rm /var/spool/pbs/mom_priv/config.d/PBSvnodedefs
   ```
3. Delete the old default vnodes. On the server host:
   ```
   # qmgr -c "delete node @default"
   ```

6.6.10 Shut Down Your Existing PBS

1. Shut down the server(s), default scheduler, and MoMs:
   ```
   qterm -t immediate -m -s -f
   ```

If your server is not running in a failover environment, the "-f" option is not required.

2. Shut down any multischeds. On each multisched host:

   a. Find the PID you want:

      ```
      ps -ef | grep pbs_sched
      ```

      For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

   b. Stop the scheduler or multisched:

      ```
      kill <multisched PID>
      ```

3. On the server host and any other comm hosts, shut down the communication daemon:

   ```
   systemctl stop pbs
   ```

   or

   ```
   <path to script>/pbs stop
   ```

4. Verify that PBS daemons are not running in the background:

   ```
   ps -ef | grep pbs
   ```

   If you see the pbs_server, pbs_sched, pbs_mom, or pbs_comm process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

   ```
   kill -9 <daemon PID>
   ```

6.6.11 Back Up Existing PBS Files

On each PBS host, make a tar file of the PBS_HOME and PBS_EXEC directories. On the MC990X, make sure you copy your backups to the server host, because otherwise they will be lost during the upgrade.

1. Make a backup directory:

   ```
   mkdir /tmp/pbs_backup
   ```

2. Make a tar file of PBS_HOME:

   ```
   cd $PBS_HOME/..
   tar -cvf /tmp/pbs_backup/PBS_HOME_tarbackup.tar  $PBS_HOME
   ```

3. Make a tar file of PBS_EXEC:

   ```
   cd $PBS_EXEC/..
   tar -cvf /tmp/pbs_backup/PBS_EXEC_tarbackup.tar $PBS_EXEC
   ```

4. Make a copy of your configuration file:

   ```
   cp /etc/pbs.conf /tmp/pbs_backup/pbs.conf.backup
   ```

5. If this is a scheduler host, make a copy of the scheduler's directory to modify:

   ```
   cp -r $PBS_HOME/sched_priv /tmp/pbs_backup/sched_priv.work
   ```

   or

   ```
   cp -r $PBS_HOME/sched_priv_<multisched name> /tmp/pbs_backup/sched_priv_<multisched name>.work
   ```

6.6.12 Install the New Version of PBS

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25.

   If you are using failover, pay special attention to your configuration parameters, including PBS_HOME and PBS_MOM_HOME, when installing the server sub-package on the secondary server host. See "Configuring the pbs.conf File for Failover" on page 409 in the PBS Professional Administrator's Guide.
4. Install the server sub-package:
   - When upgrading from 13.2 or an earlier version:

     ```
     rpm -i <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```
   - When upgrading from 14.2 or a later version:

     ```
     rpm -Uhv <path/to/server sub-package>pbspro-<daemon>-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```

6.6.12.1 Installing MoM on non-HPE 8600

On execution-only hosts, install the MoM sub-package:

- When upgrading from 13.2 or an earlier version:

  ```
  rpm -i <path/to/MoM sub-package>pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
  ```
- When upgrading from 14.2 or a later version:

  ```
  rpm -Uhv <path/to/MoM sub-package>pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
  ```

6.6.12.2 Installing MoM on HPE 8600

You install and configure MoM once on the root file system, then you push the image to all of the compute nodes by propagating it to the rack leaders. Then you reboot each node with the new image.

1. Log on to the Admin node as root.
2. Determine which image file is being used on the compute nodes. To list the nodes on rack 1:

   ```
   cimage --list-nodes r1
   ```

   It will show output in the form "*node: image_name kernel*" similar to

   ```
   r1i0n0: compute-sles15sp1 2.6.26.46-0.12-smp
   ```

   Thus node r1i0n0 is running the image "`compute-sles15sp1`" and the kernel version "2.6.26.46-0.12-smp". For the remaining steps, it is assumed that those are the images and kernel available.
3. List the available images:

   ```
   cimage --list-images
   ```

which will list the images available for the compute nodes. Each image may have multiple kernels.

4. Unless you are experienced in managing the image files, we suggest that you create a copy of the image in use and install PBS in that copy. To copy an image:

   ```
   cinstallman --create-image --clone --source compute-sles15sp1 --image compute-sles15sp1pbs
   ```

5. The image file lives in the directory `/opt/clmgr/image/images`, so change into the `tmp` directory found in the new image just cloned:

   ```
   cd /opt/clmgr/image/images/compute-sles15sp1pbs/tmp
   ```

6. Chroot to the new image file:

   ```
   chroot /opt/clmgr/image/images/compute-sles15sp1pbs /bin/sh
   ```

   The new root is in effect.

7. Download, unzip and untar the PBS package

8. Make sure that parameters for PBS_HOME, PBS_EXEC and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25

9. Install the PBS execution sub-package in the normal execution directory, `/opt/pbs`, in this system image:

   ```
   rpm -U <path/to/sub-package>pbspro-execution-<version>-0.<platform-specific-dist-tag>.<hard-
       ware>.rpm
   ```

10. Do not start PBS

11. Exit from the chroot shell and return to root's normal home directory.

12. Power down each rack of compute nodes:

    ```
    for n in `cnodes --ice-compute` ; do
        cpower node off $n
    done
    ```

13. Publish the new system image to the compute nodes:

    ```
    cimage --push-rack compute-sles15sp1pbs r\*
    ```

    This instruction will take several minutes to finish.

14. Set the new image and kernel to be booted. This need not be done if: (1) rather than cloning a new image, you have installed PBS into the image already running on the compute nodes; or (2) you are using an image that was already pushed to the nodes.

    ```
    cimage --set compute-sles15sp1pbs 2.6.26.46-0.12-smp r\*i\*n\*
    ```

15. Power up the compute nodes:

    ```
    for n in `cnodes --ice-compute` ; do
        cpower node on $n
    done
    ```

    It will take several minutes for the compute nodes to reboot.

6.6.13 Prepare Configuration File for New Scheduler(s)

If you were running one or more multischeds with your old version of PBS, make sure you update their configuration files along with that of the default scheduler. Note that the preempt_order, preempt_prio, preempt_queue_prio, preempt_sort, and log_events scheduler attributes are new; some were parameters in sched_config with the same names. In a later step (after the server is running), you will use qmgr to set the attributes. We explicitly list the step; don't worry.

For each scheduler:

1. Make a copy of the new sched_config, which is in PBS_EXEC/etc/pbs_sched_config.

   cp $PBS_EXEC/etc/pbs_sched_config $PBS_EXEC/etc/pbs_sched_config.new

2. Update PBS_EXEC/etc/pbs_sched_config.new with any modifications that were made to the old %PBS_HOME/sched_priv/sched_config or %PBS_HOME/sched_priv_<multisched name>/sched_config. This is saved in the backup directory /tmp/pbs_backup/sched_priv.work.

3. If you were using vmem at the queue or server level before the upgrade, then after upgrading you must add vmem to the resource_unset_infinite sched_config option. Otherwise jobs requesting vmem will not run.

4. Move PBS_EXEC/etc/pbs_sched_config.new to the correct name and location, i.e. $PBS_HOME/sched_priv/sched_config or $PBS_HOME/sched_priv_**<multisched name>**/sched_config:

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv/sched_config

   or

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv_<multisched name>/sched_config

6.6.14 Update Holidays File

Make sure your new holidays file is up to date.

6.6.15 Modify the New PBS Configuration File

Your new pbs.conf needs to reflect any changes that you made to the old file.

If you will use failover:

- Edit pbs.conf on the primary server host to include failover settings. See "Configuring Failover For the Primary Server on Linux" on page 411 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf.
- Edit pbs.conf on the secondary server host to include failover settings. See "Configuring Failover For the Secondary Server on Linux" on page 413 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf. You can use the following steps:
  - Copy pbs.conf from primary to secondary
  - Modify pbs.conf on secondary for failover (PBS_START_SCHED = *0*)
- Edit pbs.conf on all execution and client hosts to include failover settings. See "Configuring Failover For Execution and Client Hosts on Linux" on page 414 in the PBS Professional Administrator's Guide. Make any other changes to this file that you made to the old pbs.conf.

If you will not use failover, edit pbs.conf on each host to include changes that you made to the old pbs.conf.

6.6.16 Configure Communication Daemons

If you are using additional communication daemons (more than those automatically installed on server hosts), configure them. See section 4.5.3.2, "Configuring Communication Daemons", on page 50.

6.6.17 Start Then Stop New PBS Servers (If Using Failover)

6.6.17.1 Start New Servers

If you are not using failover, skip this step. If you are using failover, this pair of start and stop steps really is necessary. Bear with us.

Start PBS on the server host. The start/stop script is located here:

If `/etc/init.d` exists

`/etc/init.d/pbs`

Else

`/etc/rc.d/init.d/pbs`

1. If you will run a MoM on each server host, disable MoM start in `pbs.conf`, so that it contains this:
   `PBS_START_MOM=0`
2. Start PBS on the primary server host and then the secondary server host:

   `systemctl start pbs`

   or

   `<path to init.d>/init.d/pbs start`

6.6.17.2 Stop the Servers

If you are not using failover, skip this step.

1. On the primary server host:
   a. Stop PBS:

   `systemctl stop pbs`

   or

   `<path to init.d>/init.d/pbs stop`

   b. If a MoM is to run, enable it by setting `PBS_START_MOM=1` in `pbs.conf`
2. On the secondary server host:
   a. Stop PBS:

   `systemctl stop pbs`

   or

   `<path to init.d>/init.d/pbs stop`

   b. If a MoM is to run, enable it by setting `PBS_START_MOM=1` in `pbs.conf`

6.6.18 Start New PBS MoMs, Schedulers, Servers, and Comms

6.6.18.1 Start PBS on Execution Hosts

On each execution host, start MoM :

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.6.18.2 Start PBS on Server Hosts

If failover is configured, start the primary server host, wait until the primary is finished starting, then start the secondary:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.6.18.3 Restart Multischeds

To start a multisched, call `pbs_sched` and specify the name and port you already gave it. For each multisched:

```
pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>
```

6.6.18.4 Start PBS on Communication-only Hosts

Start PBS on any communication-only hosts. On each communication-only host, type:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.6.19 Import and Configure Hooks

Make sure you do not overwrite the new pbs_cgroups hook or its configuration file by importing the old ones. Instead, use the saved information from your old hook to modify the new hook and configuration file.

6.6.19.1 Import Old Hooks Except for Cgroups Hook

1. Do not import your old pbs_cgroups hook. Import your other hooks and their configuration files. For each hook **except** for pbs_cgroups:

```
# qmgr -c 'import hook <hook name> application/x-python default  /tmp/<hook name>.new3.6'
# qmgr -c 'import hook <hook name> application/x-config default  /tmp/<hook name>.configcheck'
```

6.6.19.2 Modify Cgroups Hook Configuration File

1. Export the cgroups hook configuration file to pbs_cgroups.json:

   ```
   # qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
   ```

2. You can make the cgroups hook mimic the behavior of the cpuset MoM in previous versions:

   a. Create one vnode for each NUMA node. Edit pbs_cgroups.json as follows (important):

      ```
      "vnode_per_numa_node" : true,
      ```

   b. Edit pbs_cgroups.json as follows (recommended):

      ```
      "use_hyperthreads" : true,
      ```

   c. Set the value of the ncpus_are_cores parameter; see "Configuring Hyperthreading Support" on page 582 in the PBS Professional Administrator's Guide

3. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

      ```
      # mount | grep cgroup | grep memory
      ```

      If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

      ```
      "cgroup": {
        ...
      "memory": {
      "enabled": false,
      ```

4. If you made changes to the old cgroups configuration file, you may want to make those changes in the new configuration file. Use the information saved in /etc/pbs_cgroups.old2.7

5. Import the modified configuration (make sure you use "**x-config**"):

   ```
   # qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
   ```

6.6.19.3 Enable Cgroups Hook

6. Enable the pbs_cgroups hook:

   ```
   qmgr -c "set hook pbs_cgroups enabled=true"
   ```

6.6.19.4 Write and Deploy New Hooks

If you have written new hooks for the new version of PBS, deploy them now. See the *PBS Professional Hooks Guide*.

6.6.19.5 Restart MoMs

On each execution host, restart MoM :

```
ps -eaf | grep pbs_mom
kill <MoM PID>
/opt/pbs/sbin/pbs_mom
```

6.6.20 Set License Location Server Attribute

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

6.6.21 Configure Sharing and Placement Sets

As of version 2020.1, the cgroups hook creates the child vnodes on a multi-vnode machine; it is important that any Version 2 configuration files refer only to these vnodes. Use Version 2 configuration files only to set the sharing attribute and optionally to set resources that will be used for placement sets. The default value for the sharing attribute of the vnodes is "sharing=default_shared". You can change this, for example to "sharing=default_excl".

Make sure that a Version 2 configuration file matches your available vnodes every time MoM is started. If your machine reboots with a hardware change, your earlier placement set information will not make sense because child vnode names will not match the available hardware. You can use a script to regenerate this file each time the machine starts, and run the script before MoM is restarted.

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs" that has MoM's list of vnodes; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide:

   ```
   # pbsnodes -av | awk -F'=' '{printf "%s:\tsharing = default_excl\n", $2}' > vnodedefs
   ```

2. Edit the file to reflect what you want for the sharing attribute and placement sets. Use the information saved in **/tmp/pbs_mom_backup/mom_configs/** in step "Save Execution Host Configuration Information" on page 83

3. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

4. Restart pbs_mom:

   ```
   # pbs -eaf | grep pbs_mom
   # kill <MoM PID>
   # /opt/pbs/sbin/pbs_mom
   ```

6.6.22 Re-Wrap Any MPIs

If you want any wrapped MPIs, wrap them. See "Integration by Wrapping" on page 455 in the PBS Professional Administrator's Guide.

6.6.23 Shut Down and Restart Servers

1. Shut down both servers:

   ```
   qterm -f
   ```

2. Restart PBS on the server hosts. On each server host, primary first:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to init.d>/init.d/pbs start
   ```

6.6.24 Set New Scheduler Attributes

The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.

The scheduler's log_filter configuration parameter is **obsolete**. The scheduler's log filter now uses the same bitmask system as the other daemons. The new default value is 767. Use qmgr to set the scheduler's log_events attribute to the value you want. See "Specifying Scheduler Log Events" on page 548 in the PBS Professional Administrator's Guide.

6.6.25 Enable STONITH Script

If your secondary server has a STONITH script, allow the STONITH script to run by setting its permissions to 0755.

6.6.26 Enable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, enable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.6.27 Enable Scheduling

If you disabled scheduling earlier, enable it for the default scheduler and any multischeds:

```
qmgr -c 'set sched <scheduler name> scheduling = true'
```

6.6.28 Removing Old PBS

If you decide to remove the old version of PBS after upgrading, **be sure to use the `--noscripts` option** when using `rpm -e`. Using `rpm -e` without this option, even on an older package than the one you are currently using, will cause any currently running PBS daemons to shut down, and will also remove the system V init and/or systemd service startup files. This will prevent PBS daemons from starting automatically at system boot time. If you wish to remove an older RPM without these effects, use `rpm -e --noscripts`.

6.7 Migration Upgrade Under Linux

Use these instructions:

- When moving between hosts
- When upgrading from an open-source version of PBS Professional
- When certain European or Japanese characters are stored in the data store

For specific upgrade recommendations and updates, see the Release Notes.

For a migration upgrade, you kill or requeue all jobs, install the new PBS with PBS_EXEC and PBS_HOME in different locations from those of the old version of PBS, run both the old and new instances of PBS at the same time, and qmove the jobs from the old server to the new one.

During a migration upgrade, jobs cannot be running. You can let any jobs finish before the upgrade. You can checkpoint, terminate and requeue all possible jobs and requeue non-checkpointable but rerunnable jobs. Your options with non-rerunnable jobs are to either let them finish or kill them.

In the instructions below, file and directory pathnames are the PBS defaults. If you installed PBS in different locations, use your locations instead. PBS_EXEC_OLD refers to your existing, pre-upgrade location for PBS_EXEC.

The following commands must be run as root.

6.7.1 Set Paths for Old PBS

To use the following commands without having to substitute actual paths, on the server host, source your /etc/pbs.conf file.

We recommend using /opt as the location where you'll run your old PBS during the job transfer phase, rather than /tmp.

- Choose where you want to copy your old PBS_EXEC; set PBS_EXEC_OLD to this location, and export it
- Choose where you want to copy your old PBS_HOME; set PBS_HOME_OLD to this location, and export it

6.7.2 Prevent Jobs From Being Enqueued or Started

You must deactivate the scheduler(s) and queues. When the scheduling attribute is false, jobs are not started by the scheduler. When the queues' enabled attribute is false, jobs cannot be enqueued.

1. Prevent the scheduler(s) from starting jobs. Set scheduling to *false* for the default scheduler and each multisched:
   qmgr -c "set sched <scheduler name> scheduling = false"
2. Print a list of all queues managed by the server. Save the list of queue names for the next step:
   qstat -q
3. Disable queues to stop jobs from being enqueued. Do this for each queue in your list from the previous step:
   qdisable <queue name>

6.7.3 Allow Running Jobs to Finish, or Requeue Them

You cannot perform a migration upgrade while jobs are running. Either let running jobs finish, or requeue them. (You can also delete them.)

To requeue any running jobs:

1. List the jobs. This will list some jobs more than once. You only need to requeue each job once:
   pbsnodes <hostname> | grep jobs
2. Requeue the jobs:
   qrerun <job ID> <job ID> ...

To kill the jobs:

1. List the jobs. This will list some jobs more than once. You only need to kill each job once:
   pbsnodes <hostname> | grep jobs
2. Use the qdel command to kill each job by job ID:
   qdel <job ID> <job ID> ...

To drain the host, wait until any running jobs have finished.

Make sure that there are no old job files on any execution hosts. Remove any of the following:

$PBS_HOME/mom_priv/jobs/*.JB

6.7.4 Disable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, disable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.7.5 Disable STONITH Script

If your secondary server has a STONITH script, prevent the STONITH script from running by setting its permissions to 0644.

6.7.6 Unwrap Any Wrapped MPIs

If you used the `pbsrun_wrap` mechanism with your old version of PBS, you must first unwrap any MPIs that you wrapped. This includes MPICH-GM, MPICH-MX, MPICH2, etc. You can re-wrap your MPIs after upgrading PBS.

For example, you can unwrap an MPICH2 MPI:

```
# pbsrun_unwrap pbsrun.mpich2_64
```

See "pbsrun_unwrap" on page 50 of the PBS Professional Reference Guide.

6.7.7 Save Server Host Information To Be Used for New PBS

At the server:

1. Create a backup directory called `/tmp/pbs_backup`

   ```
   mkdir /tmp/pbs_backup
   ```

2. Make a copy of the server's configuration for the new PBS:

   ```
   qmgr -c "print server" > /tmp/pbs_backup/server.new
   ```

3. Make a copy of the vnode attributes for the new PBS:

   ```
   qmgr -c "print node @default" > /tmp/pbs_backup/nodes.new
   ```

4. Make a copy of all scheduler attributes for the new PBS (this prints all settable attributes for the default and multischeds):

   ```
   qmgr -c "print sched" > /tmp/pbs_backup/sched_attrs.new
   ```

5. Print reservation information to a file:

   ```
   pbs_rstat -f > /tmp/pbs_backup/reservations
   ```

6. Make a copy of `pbs.conf` for the new PBS. This command is all one line:

   ```
   cp /etc/pbs.conf /tmp/pbs_backup/pbs.conf.backup
   ```

7. Make a copy of each scheduler's directory for the new PBS. For the default scheduler and each multisched:

   ```
   cp -rp $PBS_HOME/sched_priv /tmp/pbs_backup/sched_priv.new
   ```

   or

   ```
   cp -rp $PBS_HOME/sched_priv_<multisched name> /tmp/pbs_backup/sched_priv_<multisched name>.new
   ```

6.7.8 Save Execution Host Configuration Files

On each PBS execution host, copy the Version 1 and Version 2 configuration files:

1. Make a backup directory:
   ```
   mkdir /tmp/pbs_mom_backup
   ```
2. Make a copy of the Version 1 configuration file:
   ```
   cp $PBS_HOME/mom_priv/config /tmp/pbs_mom_backup/config.backup
   ```
3. Make a copy of the Version 2 configuration files:
   ```
   mkdir /tmp/pbs_mom_backup/mom_configs
   $PBS_EXEC_OLD/sbin/pbs_mom -s list | egrep -v '^PBS' | while read file
   do
     $PBS_EXEC_OLD/sbin/pbs_mom -s show file > /tmp/pbs_mom_backup/mom_configs/$file
   done
   ```

6.7.9 Save Hooks and Hook Configuration Files

Save your hooks and hook configuration files in ASCII format so you can check them and import them later. The new version of PBS includes a new pbs_cgroups hook with a new configuration file. If you use the cgroups hook, you must use the new hook and configuration file, but you may want to modify the configuration file, so if you have made any changes to your existing pbs_cgroups hook configuration file, you need to save it before you upgrade. Later, you can use the saved information to modify the new configuration file.

For **each** hook:

1. Save the hook. Export the hook:
   ```
   # qmgr -c 'export hook <hook name> application/x-python default /tmp/<hook name>.old2.7'
   ```
2. Save your hook configuration file. Export the configuration file:
   ```
   # qmgr -c 'export hook <hook name> application/x-config default /tmp/<hook name>.configcheck'
   ```

6.7.10 Update Hooks and Hook Configuration Files for New Python

PBS 19.4.1 and later uses Python 3.6, so if you have not already, update all of your site-defined hooks (not the built-in hooks) to Python 3.6. For **each** hook except for the pbs_cgroups hook:

1. Update your hook to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your updated hook file differently; use something like "/tmp/<hook name>.new3.6"
2. Check that the contents of the configuration file are correct for Python 3.6

6.7.11 Shut Down Your Existing PBS

Use the `-t immediate` option to `qterm` so that all possible running jobs will be requeued. If you are using failover, this will stop the secondary server as well:

1. Shut down the server, scheduler, and MoMs:
   ```
   qterm -t immediate -m -s -f
   ```

If your server is not running in a failover environment, the "-f" option is not required.

2. Shut down any multischeds. On each multisched host:

   a. Find the PID you want:

      ps –ef | grep pbs_sched

      For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

   b. Stop the scheduler or multisched:

      kill <multisched PID>

3. On the server host and any other comm hosts, shut down the communication daemon:

   systemctl stop pbs

   or

   <path to script>/pbs stop

4. Verify that PBS daemons are not running in the background:

   ps -ef | grep pbs

   If you see the pbs_server, pbs_sched, pbs_mom, or pbs_comm process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

   kill -9 <daemon PID>

6.7.12 Back Everything Up to Transfer Location

Later, you will run the old PBS server from the backup location while you are moving jobs to the new server. You must do a copy, not a move, because the installation software depends on the old version of PBS being available for it to remove. You'll be running commands from the backup directory, so we recommend a directory under /opt.

6.7.12.1 Back Up Server/scheduler/communication Host

On the server host, copy the existing PBS_HOME and PBS_EXEC hierarchies to the backup location.

1. Copy PBS_HOME to the backup directory:

   cp -rp $PBS_HOME $PBS_HOME_OLD

2. Copy PBS_EXEC to the backup directory:

   cp -rp $PBS_EXEC $PBS_EXEC_OLD

6.7.12.2 Back Up Execution Host Information

On each execution host, copy the existing PBS_HOME and PBS_EXEC hierarchies to the backup location. This is just for safekeeping.

1. Copy PBS_HOME to the backup directory:

   cp -rp $PBS_HOME /tmp/pbs_mom_backup/pbs_mom_home_backup

2. Copy PBS_EXEC to the backup directory:

   cp -rp PBS_EXEC /tmp/pbs_mom_backup/pbs_mom_exec_backup

6.7.13 Install the New Version of PBS

For a migration upgrade, use `rpm -i` so that the old version of PBS can still be used to move the jobs. You might think that you'd use `rpm -U`, but that removes the old PBS, and you still need it until the jobs are moved.

6.7.13.1 Install New PBS Server

On the server host, install the new version of PBS without uninstalling the previous version.

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC are in locations that are different from your existing PBS.

   If you are using failover, pay special attention to your configuration parameters, including PBS_HOME and PBS_MOM_HOME, when installing the server sub-package on the secondary server host. See section 3.5.2.2, "Setting Installation Parameters", on page 25 and "Configuring the pbs.conf File for Failover" on page 409 in the PBS Professional Administrator's Guide.
4. Install the server sub-package:

```
rpm -i --prefix=<new PBS_EXEC location> <path/to/server sub-package>/pbspro-server-<ver-
    sion>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

Do **not** start PBS now.

6.7.13.2 Install New PBS MoMs

On each execution host, install the new version of PBS without uninstalling the previous version. You can install new MoMs in the same locations as the old MoMs.

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25.
4. Install the execution sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:

     ```
     rpm -i <path/to/execution sub-package>/pbspro-execution-<version>-0.<platform-spe-
         cific-dist-tag>.<hardware>.rpm
     ```
   - When upgrading from 14.2 or a later version:

     ```
     rpm -U <path/to/execution sub-package>/pbspro-execution-<version>-0.<platform-spe-
         cific-dist-tag>.<hardware>.rpm
     ```

Do **not** start PBS now.

6.7.13.3 Install New PBS Client Commands

On each client command host, install the new version of PBS without uninstalling the previous version:

1. Download the appropriate PBS package
2. Uncompress the package as an unprivileged user
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC point to the locations you're using for the new PBS.
4. Install the client command sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:
     ```
     rpm -i <path/to/client command sub-package>/pbspro-client-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```
   - When upgrading from 14.2 or a later version:
     ```
     rpm -U <path/to/client command sub-package>/pbspro-client-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```

6.7.13.4 Install New PBS Communication Daemons

If you are installing a communication daemon on a communication-only host, install the server-scheduler-communication-MoM sub-package, and disable the server, scheduler, and MoM on that host. (MoM is disabled by default.) Install the new version of PBS without uninstalling the previous version.

1. Download the appropriate PBS package
2. Uncompress the package
3. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC point to the locations you are using for the new PBS.
4. Disable the server, scheduler, and MoM. In `pbs.conf`:

   ```
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_MOM=0
   ```

5. Install the server sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:
     ```
     rpm -i <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```
   - When upgrading from 14.2 or a later version:
     ```
     rpm -U <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```

Do **not** start PBS now.

6.7.14 Switch To New PBS_EXEC Path

Source your new `/etc/pbs.conf` file.

6.7.15 Create PBS_HOME

Create the subdirectories under PBS_HOME by running pbs_habitat. On the new PBS server host and on each execution host:

```
$PBS_EXEC/libexec/pbs_habitat
```

6.7.16 Start and Stop the New Server (If Using Failover)

If you are not using failover, skip this step. If you are using failover, this pair of start and stop steps really is necessary. Bear with us.

When the new server starts up it will have default queue "workq" and the server host already defined. You want to start the new server with empty configurations so that you can import your old settings.

1. Start the new server with empty queue and vnode configurations:

   ```
   $PBS_EXEC/sbin/pbs_server -t create
   ```

   A message will appear saying "Create mode and server database exists, do you wish to continue?"

   Type "**y**" to continue.

   Because of the new licensing scheme an additional message may appear:

   "One or more PBS license keys are invalid, jobs may not run"

   This message is expected. Continue to the next step in these instructions.

2. Shut down PBS:

   ```
   qterm -t immediate -m -s -f
   ```

3. Verify that PBS daemons are not running in the background:

   ```
   ps -ef | grep pbs
   ```

   If you see the pbs_server, pbs_sched, pbs_comm, or pbs_mom process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

   ```
   kill -9 <daemon PID>
   ```

6.7.17 Start the New Server Without Defined Queues or Vnodes

When the new server starts up it will have default queue "workq" and the server host already defined. You want to start the new server with empty configurations so that you can import your old settings.

Start the new server with empty queue and vnode configurations:

```
$PBS_EXEC/sbin/pbs_server -t create
```

A message will appear saying "Create mode and server database exists, do you wish to continue?"

Type "**y**" to continue.

Because of the new licensing scheme an additional message may appear:

"One or more PBS license keys are invalid, jobs may not run"

This message is expected. Continue to the next step in these instructions.

6.7.18 Re-wrap Any MPIs

If you want any wrapped MPIs, wrap them. See "Integration by Wrapping" on page 455 in the PBS Professional Administrator's Guide.

6.7.19 Set License Location Server Attribute

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

6.7.20 Clean Up Configuration Information

6.7.20.1 Clean Up Scheduler Configuration Files

If you were running one or more multischeds with your old version of PBS, make sure you update their configuration files along with that of the default scheduler. Note that the preempt_order, preempt_prio, preempt_queue_prio, preempt_sort, and log_events scheduler attributes are new; some were parameters in sched_config with the same names. In a later step (after the server is running), you will use qmgr to set the attributes. For each scheduler:

1. Make a copy of the new sched_config, which is in PBS_EXEC/etc/pbs_sched_config.

   ```
   cp $PBS_EXEC/etc/pbs_sched_config $PBS_EXEC/etc/pbs_sched_config.new
   ```

2. Update PBS_EXEC/etc/pbs_sched_config.new with any modifications that were made to your old scheduler configuration file, saved in %PBS_HOME/sched_priv/sched_config or %PBS_HOME/sched_priv_<multisched name>/sched_config.

3. If you were using vmem at the queue or server level before the upgrade, then after upgrading you must add vmem to the resource_unset_infinite sched_config option. Otherwise jobs requesting vmem will not run.

4. Move PBS_EXEC/etc/pbs_sched_config.new to the correct name and location, i.e. $PBS_HOME/sched_priv/sched_config or $PBS_HOME/sched_priv_<multisched name>/sched_config:

   ```
   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv/sched_config
   ```

   or

   ```
   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv_<multisched name>/sched_config
   ```

6.7.20.2 Clean Up Scheduler Attributes

For each scheduler, clean up the attributes saved in /tmp/pbs_backup/<scheduler name>/sched_attrs.new. When you read in multisched attributes, you'll re-create the multischeds, so make sure your new multischeds are what you want:

- Remove read-only attributes
- Remove lines containing the following:
  pbs_version

For the new default scheduler and all new multischeds:

- The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
- The scheduler's log_filter configuration parameter is **obsolete**. The scheduler's log filter now uses the same bitmask system as the other daemons. The new default value is 767. Use qmgr to set the scheduler's log_events attribute to the value you want. See "Specifying Scheduler Log Events" on page 548 in the PBS Professional Administrator's Guide.

6.7.20.3 Clean Up Server Configuration

Remove read-only attributes from the server's configuration information in server.new. For example, remove lines containing the following:

```
license_count
pbs_version
```

Remove creation commands for any reservation queues. You will create reservations and their queues separately.

6.7.20.4 Copy User Credentials to New Server

PBS caches user credentials in $PBS_HOME/server_priv/users. PBS stores the credential for each user in a file named <username>.CR. Normally this directory is created by PBS when users log in. If you installed the new version of PBS in the same location as the old one, you do not need to copy user credentials.

However, if the new version of PBS is in a different location, you need to create the directory and copy the credential files, keeping the permissions the same:

1. Create the user credential directory:
   ```
   mkdir -p $PBS_HOME/server_priv/users/
   ```
2. Copy the user credential files to the new directory:
   ```
   cp -rpu $PBS_HOME_OLD/server_priv/users/*  $PBS_HOME/server_priv/users/
   ```

6.7.20.5 Clean up Vnode Configuration

Here you prepare the vnode attribute input to the new qmgr.

If your system has multi-vnode hosts:

1. Copy your saved node configuration file /tmp/pbs_backup/nodes.new into two files:
   - qmgr_parent_vnode.out, which contains all the configuration information for parent vnodes
   - qmgr_child_vnode.out, which contains all the configuration information for vnodes that aren't parent vnodes
2. Continue by preparing configuration information for parent vnodes. You will prepare the configuration information for the other vnodes after they have been created, because the vnode names in your file must be precisely the same as the ones created by PBS.

If your system has only single-vnode hosts, follow the steps below for preparing configuration information for parent vnodes only.

6.7.20.5.i Prepare Configuration Information for Parent Vnodes

Edit qmgr_parent_vnode.out:

Leave only the the following creation lines:

- Those for parent vnodes
- Any resources you want managed on the server side through qmgr
- Custom resources on the parent vnodes

Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

- Child vnodes, that should be created by MoM (vnodes that are NOT parent vnodes)
- Lines that set the sharing attribute
- The ncpus, mem, and vmem resources, unless they should explicitly be set via qmgr

6.7.21 Create and Configure New Multischeds

Create the directories required for each new multisched, and configure each multisched. See "Creating and Configuring a Multisched" on page 55 in the PBS Professional Administrator's Guide.

6.7.22 Start New Server and New Schedulers

1. Start the new server and new default scheduler. On the server host:

   ```
   systemctl restart pbs
   ```

   or

   ```
   <path to init.d>/init.d/pbs restart
   ```

2. Start multischeds. To start a multisched, call pbs_sched and specify the name and port you already gave it. For each multisched:

   ```
   pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>
   ```

6.7.23 Replicate Queue, Server, Scheduler, and Vnode Configurations

6.7.23.1 Replicate Server and Queue Attributes

1. Give the new server the old server's configuration, but modified for the new PBS:

   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/server.new
   ```

2. Verify the configuration was read in properly:

   ```
   $PBS_EXEC/bin/qmgr -c "print server"
   ```

6.7.23.2 Replicate Scheduler Attributes

1. Give the new default scheduler the old default scheduler's attributes, and re-create your multischeds:

   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/<scheduler name>/sched_attrs.new
   ```

2. Verify the configurations were read in properly.

   You can see all schedulers at once:

   ```
   $PBS_EXEC/bin/qmgr -c "print sched"
   ```

Or for each scheduler:

```
$PBS_EXEC/bin/qmgr -c "print sched default"
or
$PBS_EXEC/bin/qmgr -c "print sched <multisched name>"
```

6.7.23.3 Replicate Vnode Attributes

Replicate vnode configuration, also modified for the new PBS:

1. Read in the parent vnode configuration file:
   ```
   $PBS_EXEC/bin/qmgr < qmgr_parent_vnode.out
   ```
2. Wait until MoM or the cgroups hook creates any vnodes that are not parent vnodes. Check:
   ```
   pbsnodes -av
   ```
3. Prepare configuration information for child vnodes:

   Edit qmgr_child_vnode.out. Make sure that the vnode names in this file are exactly what MoM or the cgroups hook created. It's easiest to put all resource information into a Version 2 configuration file, rather than using qmgr.

   Leave only the the following creation lines:

   - Any resources you want managed on the server side through qmgr
   - Custom resources on the other vnodes (but this may be easier in a Version 2 configuration file)

   Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

   - Vnodes that should be created by the cgroups hook or MoM (vnodes that are NOT parent vnodes)
   - Lines that set the sharing attribute
   - The ncpus, mem, and vmem resources, unless they should explicitly be set via qmgr
4. Read in the configuration file for child vnodes (not parent vnodes):
   ```
   $PBS_EXEC/bin/qmgr < qmgr_child_vnode.out
   ```
5. Verify the configurations were read in properly:
   ```
   $PBS_EXEC/bin/pbsnodes -a
   ```

6.7.24 Import and Configure Hooks

Make sure you do not overwrite the new pbs_cgroups hook or its configuration file by importing the old ones. Instead, use the saved information from your old hook to modify the new hook and configuration file.

6.7.24.1 Import Old Hooks Except for Cgroups Hook

1. Do not import your old pbs_cgroups hook. Import your other hooks and their configuration files. For each hook **except** for pbs_cgroups:
   ```
   # qmgr -c 'import hook <hook name> application/x-python default  /tmp/<hook name>.new3.6'
   # qmgr -c 'import hook <hook name> application/x-config default  /tmp/<hook name>.configcheck'
   ```

6.7.24.2 Modify Cgroups Hook Configuration File

If you will use the cgroups hook:

1. Export the cgroups hook configuration file to pbs_cgroups.json:

   ```
   # qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
   ```

2. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

      ```
      # mount | grep cgroup | grep memory
      ```

      If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

      ```
      "cgroup": {
        ...
      "memory": {
      "enabled": false,
      ```

3. If you made changes to the old cgroups configuration file, you may want to make those changes in the new configuration file. Use the information saved in /etc/pbs_cgroups.old2.7

4. Import the modified configuration (make sure you use "**x-config**"):

   ```
   # qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
   ```

6.7.24.3 Enable Cgroups Hook

If you will use the cgroups hook, enable the pbs_cgroups hook:

```
qmgr -c "set hook pbs_cgroups enabled=true"
```

6.7.24.4 Write and Deploy New Hooks

If you have written new hooks for the new version of PBS, deploy them now. See the *PBS Professional Hooks Guide*.

6.7.25 Start New MoMs

You can start the MoMs in any order.

- On each execution host:

  ```
  systemctl start pbs
  ```

  or

  ```
  <path to init.d>/init.d/pbs start
  ```

- Optionally start a MoM on the new server host. If your old configuration had a MoM running on the server host, and you wish to replicate the configuration, you can start a MoM on that machine:

  ```
  $PBS_EXEC/sbin/pbs_mom
  ```

6.7.26 Configure Sharing and Placement Sets

6.7.26.1 Configuration with Cgroups Hook

As of version 2020.1, the cgroups hook creates the child vnodes on a multi-vnode machine; if you will use the cgroups hook, it is important that any Version 2 configuration files refer only to these vnodes. Use Version 2 configuration files only to set the sharing attribute and optionally to set resources that will be used for placement sets. The default value for the sharing attribute of the vnodes is "sharing=default_shared". You can change this, for example to "sharing=default_excl".

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs" that has MoM's list of vnodes; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide

   ```
   # pbsnodes -av | awk -F'=' '{printf "%s:\tsharing = default_excl\n", $2}' > vnodedefs
   ```

2. Edit the file to reflect what you want for the sharing attribute and placement sets. Use the information saved in **/tmp/pbs_mom_backup/mom_configs/** in step "Save Execution Host Configuration Files" on page 96

3. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

4. Restart pbs_mom:

   ```
   # pbs -eaf | grep pbs_mom
   # kill <MoM PID>
   # /opt/pbs/sbin/pbs_mom
   ```

6.7.26.2 Configuration without Cgroups Hook

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs"; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide

2. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

3. Restart pbs_mom:

   ```
   # pbs -eaf | grep pbs_mom
   # kill <MoM PID>
   # /opt/pbs/sbin/pbs_mom
   ```

6.7.27 Start New Communication Daemons

Start PBS on any communication-only hosts. On each communication-only host, type:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.7.28 Verify Communication Between Server and MoMs

All new MoMs on all execution hosts should be running and communicating with the new server. Run `pbsnodes -a` on the new server host to see if it can communicate with the execution hosts in your complex. If a host is down, go to the problem host and restart the MoM:

```
# pbs -eaf | grep pbs_mom
# kill <MoM PID>
# /opt/pbs/sbin/pbs_mom
```

6.7.29 Re-create Reservations

You must re-create each reservation that was on the old server, using the `pbs_rsub` command. Each reservation is created as a new reservation. You can use all of the information about the old reservation except for its start time. Be sure to give each reservation a start time in the future. Use the information stored in `/tmp/pbs_backup/reservations`.

6.7.30 Change Ports and PBS_EXEC Path in `pbs.conf` for Old PBS

You must edit the `pbs.conf` file of the old PBS so that all old services use ports that won't clash with those of the new PBS. Edit **`/tmp/pbs_backup/pbs.conf.backup.`**

You must change the port numbers for these PBS daemons: server and data service. You do not need to change the port number for the comm, MoM, or scheduler.

You must also make sure that the PBS_EXEC entry in the old `pbs.conf` points to the path for the old `PBS_EXEC`.

Edit **`/tmp/pbs_backup/pbs.conf.backup`** so that the entries look like those in the following table:

Table 6-1: Entries in Old PBS Configuration File

New Entry in `pbs.conf`	Description
PBS_EXEC=**`<path to PBS_EXEC_OLD>`**	Location where `PBS_EXEC` for your old PBS was copied
PBS_HOME=**`<path to PBS_HOME_OLD>`**	Location where `PBS_HOME` for your old PBS was copied
PBS_START_SERVER=1	Unchanged
PBS_START_MOM=1	Unchanged
PBS_START_SCHED=1	Unchanged
PBS_SERVER=<hostname>	Unchanged
PBS_BATCH_SERVICE_PORT=13001	This is the changed port number for the old server
PBS_DATA_SERVICE_PORT=13007	This is the changed port number for the old data service

6.7.31 Start the Old Server

You must start the old server in order to move jobs to the new server. The old server must be started on **alternate** ports. These are specified in **`/tmp/pbs_backup/pbs.conf.backup`**.

Start the old server daemon and point it to the old configuration file:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/sbin/pbs_server
```

6.7.32 Verify Old Server is Running on Alternate Ports

Verify that the old `pbs_server` is running on the alternate ports by running the following:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qstat @<old server host>:13001
```

6.7.33 Move Existing Jobs to the New Server

You must move existing jobs from the old server to the new server. To do this, you run the `qmove` commands from the old server, and give the new server's port number, 15001, in the destination. See "qmove" on page 172 of the PBS Professional Reference Guide or the `qmove(1B)` man page. When moving jobs from reservation queues, be sure to move them into the equivalent new reservation queues.

If your jobs have dependencies, move them according to the order in which they appear in the dependency chain. If job A depends on the outcome of job B, move job B first.

If your old server host also ran a MoM, you will need to delete that vnode from the old server.

Delete the vnode on the old server host:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qmgr -c "d n <old server host>"
    <old server host>:13001
```

Move jobs from the old server to the new one:

1. Print the list of jobs on the old server:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qstat @<old server host>:13001
```

2. Move each job from each queue. Make sure that you move jobs in old reservation queues to their counterparts on the new server:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qmove <new queue name>@<new
    server host>:15001 <job id>@<old server host>:13001
```

   You can use `qselect` to select all the jobs in a queue instead of moving each job individually.

3. Move all jobs in a queue:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup
for jobname in $($PBS_EXEC_OLD/bin/qselect -q <queue name>@<old server host>:13001);
    do
        $PBS_EXEC_OLD/bin/qmove <queue name>@<new server host>:15001 ${jobname}@<old server
            host>:13001;
    done
```

   If you see the error message "Too many arguments...", there are too many jobs to fit in the shell's command line buffer. You can continue moving jobs one at a time until there are few enough.

6.7.34 Shut Down Old Server

Shut down the old server daemon:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qterm -t quick <old server
    host>:13001
```

6.7.35 Enable STONITH Script

If your secondary server has a STONITH script, allow the STONITH script to run by setting its permissions to 0755.

6.7.36 Enable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, enable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.7.37 Enable Scheduling

If you disabled scheduling earlier, enable it for the default scheduler and any multischeds:

```
qmgr -c 'set sched <scheduler name> scheduling = true'
```

6.7.38 Removing Old PBS

If you decide to remove the old version of PBS after upgrading, **be sure to use the `--noscripts` option** when using `rpm -e`. Using `rpm -e` without this option, even on an older package than the one you are currently using, will cause any currently running PBS daemons to shut down, and will also remove the system V init and/or systemd service startup files. This will prevent PBS daemons from starting automatically at system boot time. If you wish to remove an older RPM without these effects, use `rpm -e --noscripts`.

6.8 Upgrading a Windows/Linux Complex

As of version 19.4.1, Windows MoMs and client commands run with a Linux server, scheduler(s), and comm(s). PBS servers, schedulers, and comms run on Linux only. These instructions are for upgrading from a Windows execution host/Linux server complex to a Windows execution host/Linux server complex. If your existing complex is all Windows, see section 6.9, "Upgrading from an All-Windows Complex", on page 125.

You must use a migration upgrade with a Windows/Linux complex. During the migration upgrade, you can install the new version of PBS in the same place or in a new location, which can be the default location or a non-default location.

You will probably want to move jobs from the old system to the new. During a migration upgrade, jobs cannot be running. You can requeue rerunnable jobs. Your can let non-rerunnable jobs finish, or you can kill them.

On the Windows hosts, the account from which you install PBS (the installation account) must be a local account that is a member of the local Administrators group on the local computer.

In the instructions below, file and directory pathnames are the PBS defaults. If you installed PBS in different locations, use your locations instead. Where you see `%WINDIR%`, it will be automatically replaced by the correct directory.

The name of the default server host is specified in `/etc/pbs.conf`.

The default installation location on Windows systems is `\Program Files (x86)\PBS\`.

You perform a migration upgrade by copying your old PBS to a temporary location and running it from that temporary location so that you can migrate jobs to the new PBS.

6.8.1 Set Paths for Old PBS

To use the following commands without having to substitute actual paths, on the server host, source your `/etc/pbs.conf` file.

We recommend using /opt as the location where you'll run your old PBS during the job transfer phase, rather than /tmp.

- Choose where you want to copy your old PBS_EXEC; set PBS_EXEC_OLD to this location, and export it
- Choose where you want to copy your old PBS_HOME; set PBS_HOME_OLD to this location, and export it

6.8.2 Prevent Jobs From Being Enqueued or Started

You must deactivate the scheduler(s) and queues. When the scheduling attribute is false, jobs are not started by the scheduler. When the queues' enabled attribute is false, jobs cannot be enqueued.

1. Prevent the scheduler(s) from starting jobs. Set scheduling to *false* for the default scheduler and each multisched:
   ```
   qmgr -c "set sched <scheduler name> scheduling = false"
   ```
2. Print a list of all queues managed by the server. Save the list of queue names. You will need it in the next step and when moving jobs:
   ```
   qstat -q
   ```
3. Disable queues to stop jobs from being enqueued. Do this for each queue in your list from the previous step:
   ```
   qdisable <queue name>
   ```

6.8.3 Allow Running Jobs to Finish, or Requeue Them

You cannot perform a migration upgrade while jobs are running. Either let running jobs finish, or requeue them. (You can also delete them.)

To requeue any running jobs:

1. List the jobs. This will list some jobs more than once. You only need to requeue each job once:
   ```
   pbsnodes <hostname> | findstr jobs
   ```
2. Requeue the jobs:
   ```
   qrerun <job ID> <job ID> ...
   ```

To kill the jobs:

1. List the jobs. This will list some jobs more than once. You only need to kill each job once:
   ```
   pbsnodes <hostname> | grep jobs
   ```
2. Use the qdel command to kill each job by job ID:
   ```
   qdel <job ID> <job ID> ...
   ```

To drain the host, wait until any running jobs have finished.

Make sure that there are no old job files on any execution hosts. Remove any of the following:

```
C:\Program Files (x86)\PBS\home\mom_priv\jobs\*.JB
```

6.8.4 Disable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, disable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.8.5 Disable STONITH Script

If your secondary server has a STONITH script, prevent the STONITH script from running by setting its permissions to 0644.

6.8.6 Save Server Host Information To Be Used for New PBS

At the server:

1. Create a backup directory called `/tmp/pbs_backup`

   ```
   mkdir /tmp/pbs_backup
   ```

2. Make a copy of the server's configuration for the new PBS:

   ```
   qmgr -c "print server" > /tmp/pbs_backup/server.new
   ```

3. Make a copy of the vnode attributes for the new PBS:

   ```
   qmgr -c "print node @default" > /tmp/pbs_backup/nodes.new
   ```

4. Make a copy of all scheduler attributes for the new PBS (this prints all settable attributes for the default and multi-scheds):

   ```
   qmgr -c "print sched" > /tmp/pbs_backup/sched_attrs.new
   ```

5. Print reservation information to a file**:**

   ```
   pbs_rstat -f > /tmp/pbs_backup/reservations
   ```

6. Make a copy of `pbs.conf` for the new PBS. This command is all one line:

   ```
   cp /etc/pbs.conf /tmp/pbs_backup/pbs.conf.backup
   ```

7. Make a copy of each scheduler's directory for the new PBS. For the default scheduler and each multisched:

   ```
   cp -rp $PBS_HOME/sched_priv /tmp/pbs_backup/sched_priv.new
   ```

   or

   ```
   cp -rp $PBS_HOME/sched_priv_<multisched name> /tmp/pbs_backup/sched_priv_<multisched name>.new
   ```

6.8.7 Save Execution Host Configuration Files

On each PBS execution host, copy the Version 1 and Version 2 configuration files:

1. Make a backup directory:

   ```
   mkdir "%WINDIR%\TEMP\PBS_MoM_Backup"
   ```

2. Make a copy of the Version 1 configuration file:

   ```
   copy "C:\Program Files (x86)\PBS\home\mom_priv\config" "%WINDIR%\TEMP\PBS_MoM_Backup\con-
       fig.backup"
   ```

3. Make a copy of the Version 2 configuration files:

   ```
   mkdir "%WINDIR%\TEMP\PBS_MoM_Backup\mom_config"
   for /f %a in (' "C:\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe" -N -s list') do
       "C:\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe" -N -s show %a >
       "%WINDIR%\TEMP\PBS_MoM_Backup\mom_config\%a"
   ```

6.8.8 Save Hooks and Hook Configuration Files

Save your hooks and hook configuration files in ASCII format so you can check them and import them later. The new version of PBS includes a new pbs_cgroups hook with a new configuration file. If you use the cgroups hook, you must use the new hook and configuration file, but you may want to modify the configuration file, so if you have made any changes to your existing pbs_cgroups hook configuration file, you need to save it before you upgrade. Later, you can use the saved information to modify the new configuration file.

For **each** hook:

1. Save the hook. Export the hook:

   ```
   # qmgr -c 'export hook <hook name> application/x-python default /tmp/<hook name>.old2.7'
   ```

2. Save your hook configuration file. Export the configuration file:

   ```
   # qmgr -c 'export hook <hook name> application/x-config default /tmp/<hook name>.configcheck'
   ```

6.8.9 Update Hooks and Hook Configuration Files for New Python

PBS 19.4.1 and later uses Python 3.6, so if you have not already, update all of your site-defined hooks (not the built-in hooks) to Python 3.6. For **each** hook except for the pbs_cgroups hook:

1. Update your hook to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your updated hook file differently; use something like "/tmp/<hook name>.new3.6"
2. Check that the contents of the configuration file are correct for Python 3.6

6.8.10 Shut Down Your Existing PBS

Use the `-t immediate` option to `qterm` so that all possible running jobs will be requeued. If you are using failover, this will stop the secondary server as well:

1. Shut down the server, scheduler, and MoMs:

   ```
   qterm -t immediate -m -s -f
   ```

   If your server is not running in a failover environment, the "-f" option is not required.

2. Shut down any multischeds. On each multisched host:

   a. Find the PID you want:

      ```
      ps -ef | grep pbs_sched
      ```

      For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

   b. Stop the scheduler or multisched:

      ```
      kill <multisched PID>
      ```

3. On the server host and any other comm hosts, shut down the communication daemon:

   ```
   systemctl stop pbs
   ```

   or

   ```
   <path to script>/pbs stop
   ```

4. Verify that PBS daemons are not running in the background:

   ```
   ps -ef | grep pbs
   ```

If you see the pbs_server, pbs_sched, pbs_mom, or pbs_comm process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

```
kill -9 <daemon PID>
```

or

```
net stop pbs_mom
```

6.8.11 Back Everything Up to Transfer Location

Later, you will run the old PBS server from the backup location while you are moving jobs to the new server. You must do a copy, not a move, because the installation software depends on the old version of PBS being available for it to remove. You'll be running commands from the backup directory, so we recommend a directory under /opt.

6.8.11.1 Back Up Server/scheduler/communication Host

On the server host, copy the existing PBS_HOME and PBS_EXEC hierarchies to the backup location.

1. Copy PBS_HOME to the backup directory:

   ```
   cp -rp $PBS_HOME $PBS_HOME_OLD
   ```

2. Copy PBS_EXEC to the backup directory:

   ```
   cp -rp $PBS_EXEC $PBS_EXEC_OLD
   ```

6.8.11.2 Back Up Execution Host Information

On each execution host, copy the existing PBS_HOME and PBS_EXEC hierarchies to the backup location. This is just for safekeeping.

1. Copy PBS_HOME to the backup directory:

   ```
   xcopy /o /E /C "C:\Program Files (x86)\PBS\home" %WINDIR%\TEMP\PBS_MoM_Backup
   ```

2. Copy PBS_EXEC to the backup directory:

   ```
   xcopy /o /E /C "C:\Program Files (x86)\PBS\exec" %WINDIR%\TEMP\PBS_MoM_Backup
   ```

6.8.12 Install the New Version of PBS

For a migration upgrade, use rpm -i so that the old version of PBS can still be used to move the jobs. You might think that you'd use rpm -U, but that removes the old PBS, and you still need it until the jobs are moved.

6.8.12.1 Install New PBS Server

On the server host, install the new version of PBS without uninstalling the previous version.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package as an unprivileged user
4. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC are in locations that are different from your existing PBS.

If you are using failover, pay special attention to your configuration parameters, including PBS_HOME and PBS_MOM_HOME, when installing the server sub-package on the secondary server host. See section 3.5.2.2, "Setting Installation Parameters", on page 25 and "Configuring the pbs.conf File for Failover" on page 409 in the PBS Professional Administrator's Guide.

5. Install the server sub-package:

```
rpm -i --force --prefix=<new PBS_EXEC location> <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
```

Do **not** start PBS now.

6.8.12.2 Install New PBS Communication Daemons

If you are installing a communication daemon on a communication-only host, install the server-scheduler-communication-MoM sub-package, and disable the server, scheduler, and MoM on that host. (MoM is disabled by default.) Install the new version of PBS without uninstalling the previous version.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC point to the locations you are using for the new PBS.
5. Disable the server, scheduler, and MoM. In pbs.conf:

```
PBS_START_SERVER=0
PBS_START_SCHED=0
PBS_START_MOM=0
```

6. Install the server sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:

     ```
     rpm -i <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```
   - When upgrading from 14.2 or a later version:

     ```
     rpm -U <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
     ```

Do **not** start PBS now.

6.8.12.3 Switch To New PBS_EXEC Path

On the server host, source your new /etc/pbs.conf file.

6.8.12.4 Create PBS_HOME

Create the subdirectories under PBS_HOME by running pbs_habitat. On the new PBS server host:

```
$PBS_EXEC/libexec/pbs_habitat
```

6.8.12.5 Install New PBS MoMs and Client Commands

On each execution and client host, do the following:

1. Log in with the installation account.
2. Install the KB2999226 update for Windows on all Windows Server 2012 execution and client machines.
3. Download the MSI installer (the `.msi` file).
4. Double-click the MSI installer; the splash screen is displayed.
5. Click the *Next* button to move to the license page. Accept the license.
6. Click the *Next* button and choose the path where you will install the PBS executable. By default this path points to "`C:\Program Files (x86)\PBS\`".
7. Using "Run As Administrator", open a Command prompt.

6.8.12.6 Configure New PBS MoMs and Client Hosts

On each execution and client host, manually execute the `win_postinstall.py` script as shown below. When you specify the PBS service account, whether or not you are on a domain machine, include only the username, not the domain. For example, if the full username on a domain machine is <domain>\<username>, pass only *username* as an argument.

On each execution host:

- Delete the "home" folder inside "`C:\Program Files (x86)\PBS\`" if it exists
- Run `win_postinstall`:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account> -p
    <PBS service account password> -s <server name> -t execution -c <path to scp.exe>
```

On each client host:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account>  -p <PBS
    service account password> -s <server name> -t client -c <path to scp.exe>
```

6.8.13 Start and Stop the New Server (If Using Failover)

If you are not using failover, skip this step. If you are using failover, this pair of start and stop steps really is necessary. Bear with us.

When the new server starts up it will have default queue "workq" and the server host already defined. You want to start the new server with empty configurations so that you can import your old settings.

1. Start the new server with empty queue and vnode configurations:

   ```
   $PBS_EXEC/sbin/pbs_server -t create
   ```

   A message will appear saying "Create mode and server database exists, do you wish to continue?"

   Type "**y**" to continue.

   Because of the new licensing scheme an additional message may appear:

   "`One or more PBS license keys are invalid, jobs may not run`"

This message is expected. Continue to the next step in these instructions.

2. Shut down PBS:

   qterm -t immediate -m -s -f

3. Verify that PBS daemons are not running in the background:

   ps -ef | grep pbs

   If you see the pbs_server, pbs_sched, pbs_comm, or pbs_mom process running, manually terminate that process. If using failover, check both primary and secondary server hosts:

   kill -9 <daemon PID>

6.8.14 Start the New Server Without Defined Queues or Vnodes

When the new server starts up it will have default queue "workq" and the server host already defined. You want to start the new server with empty configurations so that you can import your old settings.

Start the new server with empty queue and vnode configurations:

$PBS_EXEC/sbin/pbs_server -t create

A message will appear saying "Create mode and server database exists, do you wish to continue?"

Type "**y**" to continue.

Because of the new licensing scheme an additional message may appear:

"One or more PBS license keys are invalid, jobs may not run"

This message is expected. Continue to the next step in these instructions.

6.8.15 Set License Location Server Attribute

Set the pbs_license_info server attribute to the location of the license server:

 # qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'

6.8.16 Clean Up Configuration Information

6.8.16.1 Clean Up Scheduler Configuration Files

If you were running one or more multischeds with your old version of PBS, make sure you update their configuration files along with that of the default scheduler. Note that the preempt_order, preempt_prio, preempt_queue_prio, preempt_sort, and log_events scheduler attributes are new; some were parameters in sched_config with the same names. In a later step (after the server is running), you will use qmgr to set the attributes. For each scheduler:

1. Make a copy of the new sched_config, which is in PBS_EXEC/etc/pbs_sched_config.

   cp $PBS_EXEC/etc/pbs_sched_config $PBS_EXEC/etc/pbs_sched_config.new

2. Update PBS_EXEC/etc/pbs_sched_config**.new** with any modifications that were made to your old scheduler configuration file, saved in (Windows) "%WINDIR%\TEMP\PBS_Backup\sched_priv.sched_config" or "%WINDIR%\TEMP\PBS_Backup\sched_priv_<multisched name>.sched_config", or in (Linux) %PBS_HOME/sched_priv/sched_config or %PBS_HOME/sched_priv_<multisched name>/sched_config.

3. If you were using vmem at the queue or server level before the upgrade, then after upgrading you must add vmem to the resource_unset_infinite sched_config option. Otherwise jobs requesting vmem will not run.

4. Move PBS_EXEC/etc/pbs_sched_config**.new** to the correct name and location, i.e. $PBS_HOME/sched_priv/sched_config or $PBS_HOME/sched_priv_**<multisched name>**/sched_config:

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv/sched_config

   or

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv_<multisched name>/sched_config

6.8.16.2 Clean Up Scheduler Attributes

For each scheduler, clean up the attributes saved in /tmp/pbs_backup/<scheduler name>/sched_attrs**.new**. When you read in multisched attributes, you'll re-create the multischeds, so make sure your new multischeds are what you want:

- Remove read-only attributes
- Remove lines containing the following:

  pbs_version

For the new default scheduler and all new multischeds:

- The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
- The scheduler's log_filter configuration parameter is **obsolete**. The scheduler's log filter now uses the same bitmask system as the other daemons. The new default value is 767. Use qmgr to set the scheduler's log_events attribute to the value you want. See "Specifying Scheduler Log Events" on page 548 in the PBS Professional Administrator's Guide.

6.8.16.3 Clean Up Server Configuration

Remove read-only attributes from the server's configuration information in server**.new**. For example, remove lines containing the following:

```
license_count
pbs_version
```

Remove creation commands for any reservation queues. You will create reservations and their queues separately.

6.8.16.4 Copy User Credentials to New Server

PBS caches user credentials in `$PBS_HOME/server_priv/users`. PBS stores the credential for each user in a file named <username>.CR. Normally this directory is created by PBS when users log in. If you installed the new version of PBS in the same location as the old one, you do not need to copy user credentials.

However, if the new version of PBS is in a different location, you need to create the directory and copy the credential files, keeping the permissions the same:

1. Create the user credential directory:
   ```
   mkdir -p $PBS_HOME/server_priv/users/
   ```
2. Copy the user credential files to the new directory:
   ```
   cp -rpu $PBS_HOME_OLD/server_priv/users/*  $PBS_HOME/server_priv/users/
   ```

6.8.16.5 Clean up Vnode Configuration

Here you prepare the vnode attribute input to the new `qmgr`.

If your system has multi-vnode hosts:

- Copy your saved node configuration file `"%WINDIR%\TEMP\PBS_Backup\nodes.new"` into two files:
  - `qmgr_parent_vnode.out`, which contains all the configuration information for parent vnodes
  - `qmgr_child_vnode.out`, which contains all the configuration information for vnodes that aren't parent vnodes
- Continue by preparing configuration information for parent vnodes. You will prepare the configuration information for the child vnodes after they have been created, because the vnode names in your file must be precisely the same as the ones created by PBS.

If your system has only single-vnode hosts, follow the steps below for preparing configuration information for parent vnodes only.

6.8.16.5.i Prepare Configuration Information for Parent Vnodes

Edit `qmgr_parent_vnode.out`:

Leave only the the following creation lines:

- Those for parent vnodes
- Any resources you want managed on the server side through `qmgr`
- Custom resources on the parent vnodes

Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

- Child vnodes, that should be created by MoM (vnodes that are NOT parent vnodes)
- Lines that set the sharing attribute
- The ncpus, mem, and vmem resources, unless they should explicitly be set via `qmgr`

6.8.17 Create and Configure New Multischeds

Create the directories required for each new multisched, and configure each multisched. See "Creating and Configuring a Multisched" on page 55 in the PBS Professional Administrator's Guide.

6.8.18 Start New Server and New Schedulers

1. Start the new server and new default scheduler. On the server host:

   ```
   systemctl restart pbs
   ```

   or

   ```
   <path to init.d>/init.d/pbs restart
   ```

2. Start multischeds. To start a multisched, call `pbs_sched` and specify the name and port you already gave it. For each multisched:

   ```
   pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>
   ```

6.8.19 Replicate Queue, Server, Scheduler, and Vnode Configurations

6.8.19.1 Replicate Server and Queue Attributes

1. Give the new server the old server's configuration, but modified for the new PBS:

   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/server.new
   ```

2. Verify the configuration was read in properly:

   ```
   $PBS_EXEC/bin/qmgr -c "print server"
   ```

6.8.19.2 Replicate Scheduler Attributes

1. Give the new default scheduler the old default scheduler's attributes, and re-create your multischeds:

   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/<scheduler name>/sched_attrs.new
   ```

2. Verify the configurations were read in properly.

   You can see all schedulers at once:

   ```
   $PBS_EXEC/bin/qmgr -c "print sched"
   ```

   Or for each scheduler:

   ```
   $PBS_EXEC/bin/qmgr -c "print sched default"
   or
   $PBS_EXEC/bin/qmgr -c "print sched <multisched name>"
   ```

6.8.19.3 Replicate Vnode Attributes

Replicate vnode configuration, also modified for the new PBS:

1. Read in the parent vnode configuration file:

   ```
   $PBS_EXEC/bin/qmgr < qmgr_natural_vnode.out
   ```

2. Wait until MoM or the cgroups hook creates any child vnodes. Check:

   ```
   pbsnodes -av
   ```

3. Prepare configuration information for child vnodes:

   Edit `qmgr_child_vnode.out`. Make sure that the vnode names in this file are exactly what MoM or the cgroups hook created. It's easiest to put all resource information into a Version 2 configuration file, rather than using `qmgr`.

Leave only the the following creation lines:

- Any resources you want managed on the server side through qmgr
- Custom resources on the child vnodes (but this may be easier in a Version 2 configuration file)

Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

- Child vnodes, that should be created by the cgroups hook or MoM (vnodes that are NOT parent vnodes)
- Lines that set the sharing attribute
- The ncpus, mem, and vmem resources, unless they should explicitly be set via qmgr

4. Read in the configuration file for child vnodes (not parent vnodes):

```
$PBS_EXEC/bin/qmgr < qmgr_not_natural_vnode.out
```

5. Verify the configurations were read in properly:

```
$PBS_EXEC/bin/pbsnodes -a
```

6.8.20 Import and Configure Hooks

Make sure you do not overwrite the new pbs_cgroups hook or its configuration file by importing the old ones. Instead, use the saved information from your old hook to modify the new hook and configuration file.

6.8.20.1 Import Old Hooks Except for Cgroups Hook

1. Do not import your old pbs_cgroups hook. Import your other hooks and their configuration files. For each hook **except** for pbs_cgroups:

```
# qmgr -c 'import hook <hook name> application/x-python default  /tmp/<hook name>.new3.6'
# qmgr -c 'import hook <hook name> application/x-config default  /tmp/<hook name>.configcheck'
```

6.8.20.2 Modify Cgroups Hook Configuration File

If you will use the cgroups hook:

1. Export the new cgroups hook configuration file to pbs_cgroups.json:

   ```
   # qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
   ```

2. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

      ```
      # mount | grep cgroup | grep memory
      ```

      If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

      ```
      "cgroup": {
        ...
      "memory": {
      "enabled": false,
      ```

3. If you made changes to the old cgroups configuration file, you may want to make those changes in the new configuration file. Use the information saved in /etc/pbs_cgroups.old2.7

4. Import the modified configuration (make sure you use "**x-config**"):

   ```
   # qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
   ```

6.8.20.3 Enable Cgroups Hook

If you will use the cgroups hook, enable the pbs_cgroups hook:

```
qmgr -c "set hook pbs_cgroups enabled=true"
```

6.8.20.4 Write and Deploy New Hooks

If you have written new hooks for the new version of PBS, deploy them now. See the *PBS Professional Hooks Guide*.

6.8.20.5 Start MoMs

On each execution host, start MoM :

```
net start pbs_mom
```

6.8.21 Configure Sharing and Placement Sets

6.8.21.1 Configuration with Cgroups Hook

As of version 2020.1, the cgroups hook creates the child vnodes on a multi-vnode machine; if you will use the cgroups hook, it is important that any Version 2 configuration files refer only to these vnodes. Use Version 2 configuration files only to set the sharing attribute and optionally to set resources that will be used for placement sets. The default value for the sharing attribute of the vnodes is "sharing=default_shared". You can change this, for example to "sharing=default_excl".

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs" that has MoM's list of vnodes; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide

   ```
   # pbsnodes -av | awk -F'=' '{printf "%s:\tsharing = default_excl\n", $2}' > vnodedefs
   ```

2. Edit the file to reflect what you want for the sharing attribute and placement sets. Use the information saved in **"%WINDIR%\TEMP\PBS_MoM_Backup\mom_config"** in step "Save Execution Host Configuration Files" on page 111

3. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

4. Restart pbs_mom:

   ```
   net stop pbs_mom
   net start pbs_mom
   ```

6.8.21.2 Configuration without Cgroups Hook

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs"; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide

2. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

3. Restart pbs_mom:

   ```
   net stop pbs_mom
   net start pbs_mom
   ```

6.8.22 Start New Communication Daemons

Start PBS on any communication-only hosts. On each communication-only host, type:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.8.23 Verify Communication Between Server and MoMs

All new MoMs on all execution hosts should be running and communicating with the new server. Run `pbsnodes -a` on the new server host to see if it can communicate with the execution hosts in your complex. If a host is down, go to the problem host and restart the MoM:

```
net stop pbs_mom
net start pbs_mom
```

6.8.24 Re-create Reservations

You must re-create each reservation that was on the old server, using the pbs_rsub command. Each reservation is created as a new reservation. You can use all of the information about the old reservation except for its start time. Be sure to give each reservation a start time in the future. Use the information stored in /tmp/pbs_backup/reservations.

6.8.25 Change Ports and PBS_EXEC Path in pbs.conf for Old PBS

You must edit the pbs.conf file of the old PBS so that all old services use ports that won't clash with those of the new PBS. Edit **/tmp/pbs_backup/pbs.conf.backup.**

You must change the port numbers for the PBS server and data service. You do not need to change the port numbers for the comm, MoM, or scheduler.

You must also make sure that the PBS_EXEC entry in the old pbs.conf points to the path for the old PBS_EXEC.

Edit **/tmp/pbs_backup/pbs.conf.backup** so that the entries look like those in the following table:

Table 6-2: Entries in Old PBS Configuration File

New Entry in pbs.conf	Description
PBS_EXEC=**<path to PBS_EXEC_OLD>**	Location where PBS_EXEC for your old PBS was copied
PBS_HOME=**<path to PBS_HOME_OLD>**	Location where PBS_HOME for your old PBS was copied
PBS_START_SERVER=1	Unchanged
PBS_START_MOM=1	Unchanged
PBS_START_SCHED=1	Unchanged
PBS_SERVER=<hostname>	Unchanged
PBS_BATCH_SERVICE_PORT=13001	This is the changed port number for the old server
PBS_DATA_SERVICE_PORT=13007	This is the changed port number for the old data service

6.8.26 Start the Old Server

You must start the old server in order to move jobs to the new server. The old server must be started on **alternate** ports. These are specified in **/tmp/pbs_backup/pbs.conf.backup**.

Start the old server daemon and point it to the old configuration file:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/sbin/pbs_server
```

6.8.27 Verify Old Server is Running on Alternate Ports

Verify that the old pbs_server is running on the alternate ports by running the following:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qstat @<old server host>:13001
```

6.8.28 Move Existing Jobs to the New Server

You must move existing jobs from the old server to the new server. To do this, you run the `qmove` commands from the old server, and give the new server's port number, 15001, in the destination. See "qmove" on page 172 of the PBS Professional Reference Guide or the `qmove(1B)` man page. When moving jobs from reservation queues, be sure to move them into the equivalent new reservation queues.

If your jobs have dependencies, move them according to the order in which they appear in the dependency chain. If job A depends on the outcome of job B, move job B first.

If your old server host also ran a MoM, you will need to delete that vnode from the old server.

Delete the vnode on the old server host:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qmgr -c "d n <old server host>"
    <old server host>:13001
```

Move jobs from the old server to the new one:

1. Print the list of jobs on the old server:

   ```
   PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qstat @<old server host>:13001
   ```

2. Move each job from each queue. Make sure that you move jobs in old reservation queues to their counterparts on the new server:

   ```
   PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qmove <new queue name>@<new
       server host>:15001 <job id>@<old server host>:13001
   ```

   You can use `qselect` to select all the jobs in a queue instead of moving each job individually.

3. Move all jobs in a queue:

   ```
   export PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup
   for jobname in $($PBS_EXEC_OLD/bin/qselect -q <queue name>@<old server host>:13001);
       do
           $PBS_EXEC_OLD/bin/qmove <queue name>@<new server host>:15001 ${jobname}@<old server
               host>:13001;
       done
   ```

   If you see the error message "Too many arguments...", there are too many jobs to fit in the shell's command line buffer. You can continue moving jobs one at a time until there are few enough.

6.8.29 Shut Down Old Server

Shut down the old server daemon:

```
PBS_CONF_FILE=/tmp/pbs_backup/pbs.conf.backup $PBS_EXEC_OLD/bin/qterm -t quick <old server
    host>:13001
```

6.8.30 Enable STONITH Script

If your secondary server has a STONITH script, allow the STONITH script to run by setting its permissions to 0755.

6.8.31 Enable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, enable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.8.32 Enable Scheduling

If you disabled scheduling earlier, enable it for the default scheduler and any multischeds:

```
qmgr -c 'set sched <scheduler name> scheduling = true'
```

6.8.33 Removing Old PBS

If you decide to remove the old version of PBS after upgrading, **be sure to use the `--noscripts` option** when using `rpm -e`. Using `rpm -e` without this option, even on an older package than the one you are currently using, will cause any currently running PBS daemons to shut down, and will also remove the system V init and/or systemd service startup files. This will prevent PBS daemons from starting automatically at system boot time. If you wish to remove an older RPM without these effects, use `rpm -e --noscripts`.

6.9 Upgrading from an All-Windows Complex

As of version 19.4.1, Windows MoMs and client commands run with a Linux server, scheduler(s), and comm(s). PBS servers, schedulers, and comms run on Linux only. If you are already using a Linux server with Windows MoMs, see .section 6.8, "Upgrading a Windows/Linux Complex", on page 109.

These instructions are for upgrading from a Windows/Windows complex to a Windows/Linux complex.

If your existing complex runs a PBS server on a Windows host, "upgrading" means doing a fresh install for the server/schedulers/comms, and upgrading your Windows MoMs. You cannot preserve any jobs in any state during the upgrade. Your can let jobs finish, or you can kill them.

On the Windows hosts, the account from which you install PBS (the installation account) must be a local account that is a member of the local Administrators group on the local computer.

In the instructions below, file and directory pathnames are the PBS defaults. If you installed PBS in different locations, use your locations instead. Where you see `%WINDIR%`, it will be automatically replaced by the correct directory.

The name of the old default server host is specified in `\Program Files (x86)\PBS\pbs.conf`.

On Windows systems, PBS is installed in `\Program Files (x86)\PBS\`.

6.9.1 Prevent Jobs From Being Enqueued or Started

You must deactivate the scheduler(s) and queues. When the scheduling attribute is false, jobs are not started by the scheduler. When the queues' enabled attribute is false, jobs cannot be enqueued.

1. Prevent the scheduler(s) from starting jobs. Set scheduling to *false* for the default scheduler and each multisched:

   ```
   qmgr -c "set sched <scheduler name> scheduling = false"
   ```

2. Print a list of all queues managed by the server. Save the list of queue names. You will need it in the next step and when moving jobs:

   ```
   qstat -q
   ```

3. Disable queues to stop jobs from being enqueued. Do this for each queue in your list from the previous step:

   ```
   qdisable <queue name>
   ```

6.9.2 Allow Running Jobs to Finish, or Kill Them

You cannot perform this upgrade while jobs are running or queued. Either let running jobs finish, or kill them.

To drain the host, wait until any running jobs have finished. To kill the jobs:

1. List the jobs. This will list some jobs more than once. You only need to kill each job once:
   ```
   pbsnodes <hostname> | findstr jobs
   ```
2. Use the qdel command to kill each job by job ID:
   ```
   qdel <job ID> <job ID> ...
   ```

Make sure that there are no old job files on any execution hosts. Remove any of the following:

```
C:\Program Files (x86)\PBS\home\mom_priv\jobs\*.JB
```

6.9.3 Disable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, disable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.9.4 Disable STONITH Script

If your secondary server has a STONITH script, prevent the STONITH script from running.

6.9.5 Save Server Host Information To Be Used for New PBS

At the server:

1. Make a backup directory:
   ```
   mkdir "%WINDIR%\TEMP\PBS_Backup"
   ```
2. Make a copy of the server's configuration for the new PBS:
   ```
   qmgr -c "print server" > "%WINDIR%\TEMP\PBS_Backup\server.new"
   ```
3. Make a copy of the vnode attributes for the new PBS:
   ```
   qmgr -c "print node @default" > "%WINDIR%\TEMP\PBS_Backup\nodes.new"
   ```
4. Make a copy of all scheduler configurations for the new PBS (this prints settable attributes for default and multischeds):
   ```
   qmgr -c "print sched" > "%WINDIR%\TEMP\PBS_Backup\sched_attrs.new"
   ```
5. Print reservation information to a file:
   ```
   pbs_rstat -f > "%WINDIR%\TEMP\PBS_Backup\reservations"
   ```
6. Make a copy of `pbs.conf` for the new PBS. This command is all one line:
   ```
   copy "\Program Files (x86)\PBS\pbs.conf"  "%WINDIR%\TEMP\PBS_Backup\pbs.conf.new"
   ```
7. Make a copy of each scheduler's directory for the new PBS. For the default scheduler and each multisched:
   ```
   xcopy /o /E /C "C:\Program Files (x86)\PBS\home\sched_priv"
       "%WINDIR%\TEMP\PBS_Backup\sched_priv.work"
   ```
   or
   ```
   xcopy /o /E /C "C:\Program Files (x86)\PBS\home\sched_priv_<multisched name>"
       "%WINDIR%\TEMP\PBS_Backup\sched_priv_<multisched name>.work"
   ```

When you see this message:

```
Does C:\Windows\TEMP\PBS_Backup\sched_priv.work specify a file name or directory name on the
    target (F = file, D = directory)?
```

Type this:

```
D
```

6.9.6 Save Execution Host Configuration Files

On each PBS execution host, copy the Version 1 and Version 2 configuration files:

1. Make a backup directory:

```
mkdir "%WINDIR%\TEMP\PBS_MoM_Backup"
```

2. Make a copy of the Version 1 configuration file:

```
copy "C:\Program Files (x86)\PBS\home\mom_priv\config" "%WINDIR%\TEMP\PBS_MoM_Backup\con-
    fig.backup"
```

3. Make a copy of the Version 2 configuration files:

```
mkdir "%WINDIR%\TEMP\PBS_MoM_Backup\mom_config"
for /f %a in (' "C:\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe" -N -s list') do
    "C:\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe" -N -s show %a >
    "%WINDIR%\TEMP\PBS_MoM_Backup\mom_config\%a"
```

6.9.7 Save Hooks and Hook Configuration Files

Save your hooks and hook configuration files in ASCII format so you can check them and import them later. The new version of PBS includes a new pbs_cgroups hook with a new configuration file. If you use the cgroups hook, you must use the new hook and configuration file, but you may want to modify the configuration file, so if you have made any changes to your existing pbs_cgroups hook configuration file, you need to save it before you upgrade. Later, you can use the saved information to modify the new configuration file.

For **each** hook:

1. Save the hook. Export the hook:

```
# qmgr -c 'export hook <hook name> application/x-python default %WINDIR%\TEMP\PBS_Backup\<hook
    name>.old2.7'
```

2. Save your hook configuration file. Export the configuration file:

```
# qmgr -c 'export hook <hook name> application/x-config default %WINDIR%\TEMP\PBS_Backup\<hook
    name>.configcheck'
```

3. Run `dos2unix` to convert the hooks and hook configuration files from DOS to UNIX format:

```
dos2unix /tmp/pbs_backup/<saved file>
```

6.9.8 Update Hooks and Hook Configuration Files for New Python

PBS 19.4.1 and later uses Python 3.6, so if you have not already, update all of your site-defined hooks (not the built-in hooks) to Python 3.6. For **each** hook except for the pbs_cgroups hook:

1. Update your hook to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your updated hook file differently; use something like "WINDIR%\TEMP\PBS_Backup\<hook name>.new3.6"
2. Check that the contents of the configuration file are correct for Python 3.6

6.9.9 Shut Down Your Existing PBS

Use the `-t immediate` option to `qterm` so that all possible running jobs will be requeued. If you are using failover, this will stop the secondary server as well:

1. Shut down the server, scheduler, and MoMs:

   ```
   qterm -t immediate -m -s -f
   ```

   If your server is not running in a failover environment, the "-f" option is not required.

2. Shut down any multischeds. On each multisched host:

   ```
   net stop pbs_sched
   ```

3. On the server host and any other comm hosts, shut down the communication daemon:

   ```
   net stop pbs_comm
   ```

6.9.10 Install the New Version of PBS

6.9.10.1 Install New PBS Server

On the server host, install the new version of PBS without uninstalling the previous version.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package as an unprivileged user
4. Make sure that parameters for PBS_HOME, PBS_EXEC, PBS_LICENSE_INFO, PBS_SERVER and PBS_DATA_SERVICE_USER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC are in locations that are different from your existing PBS.

   If you are using failover, pay special attention to your configuration parameters, including PBS_HOME and PBS_MOM_HOME, when installing the server sub-package on the secondary server host. See section 3.5.2.2, "Setting Installation Parameters", on page 25 and "Configuring the pbs.conf File for Failover" on page 409 in the PBS Professional Administrator's Guide.

5. Install the server sub-package:

   ```
   rpm -i --prefix=<new PBS_EXEC location> <path/to/server sub-package>/pbspro-server-<ver-
       sion>-0.<platform-specific-dist-tag>.<hardware>.rpm
   ```

Do **not** start PBS now.

6.9.10.2 Install New PBS Communication Daemons

If you are installing a communication daemon on a communication-only host, install the server-scheduler-communication-MoM sub-package, and disable the server, scheduler, and MoM on that host. (MoM is disabled by default.) Install the new version of PBS without uninstalling the previous version.

1. Log in as root
2. Download the appropriate PBS package
3. Uncompress the package
4. Make sure that parameters for PBS_HOME, PBS_EXEC, and PBS_SERVER are set correctly; see section 3.5.2.2, "Setting Installation Parameters", on page 25. Make sure that PBS_HOME and PBS_EXEC point to the locations you are using for the new PBS.
5. Disable the server, scheduler, and MoM. In pbs.conf:

   ```
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_MOM=0
   ```

6. Install the server sub-package. The method you use depends on the version you are upgrading from.
   - When upgrading from 13.2 or an earlier version:

     rpm -i <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm
   - When upgrading from 14.2 or a later version:

     rpm -U <path/to/server sub-package>/pbspro-server-<version>-0.<platform-specific-dist-tag>.<hardware>.rpm

Do **not** start PBS now.

6.9.10.3 Create PBS_HOME

Create the subdirectories under PBS_HOME by running pbs_habitat. On the new PBS server host:

$PBS_EXEC/libexec/pbs_habitat

6.9.10.4 Install New PBS MoMs and Client Commands

On each execution and client host, do the following:

1. Log in with the installation account.
2. Install the KB2999226 update for Windows on all Windows Server 2012 execution and client machines.
3. Download the MSI installer (the .msi file).
4. Double-click the MSI installer; the splash screen is displayed.
5. Click the *Next* button to move to the license page. Accept the license.
6. Click the *Next* button and choose the path where you will install the PBS executable. By default this path points to "C:\Program Files (x86)\PBS\".
7. Using "Run As Administrator", open a Command prompt.

6.9.10.5 Configure New PBS MoMs and Client Hosts

On each execution and client host, manually execute the win_postinstall.py script as shown below. When you specify the PBS service account, whether or not you are on a domain machine, include only the username, not the domain. For example, if the full username on a domain machine is <domain>\<username>, pass only *username* as an argument.

On each execution host:

- Delete the "home" folder inside "C:\Program Files (x86)\PBS\" if it exists
- Run win_postinstall:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account> -p
    <PBS service account password> -s <server name> -t execution -c <path to scp.exe>
```

On each client host:

```
<PBS_EXEC>\python\python.exe <PBS_EXEC>\etc\win_postinstall.py -u <PBS service account>  -p <PBS
    service account password> -s <server name> -t client -c <path to scp.exe>
```

6.9.11 Start the New Server Without Defined Queues or Vnodes

When the new server starts up it will have default queue "workq" and the server host already defined. You want to start the new server with empty configurations so that you can import your old settings.

Start the new server with empty queue and vnode configurations:

```
$PBS_EXEC/sbin/pbs_server -t create
```

A message will appear saying "Create mode and server database exists, do you wish to continue?"

Type "**y**" to continue.

Because of the new licensing scheme an additional message may appear:

"One or more PBS license keys are invalid, jobs may not run"

This message is expected. Continue to the next step in these instructions.

6.9.12 Set License Location Server Attribute

Set the pbs_license_info server attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

6.9.13 Clean Up Configuration Information

6.9.13.1 Clean Up Scheduler Configuration Files

If you were running one or more multischeds with your old version of PBS, make sure you update their configuration files along with that of the default scheduler. Note that the preempt_order, preempt_prio, preempt_queue_prio, preempt_sort, and log_events scheduler attributes are new; some were parameters in sched_config with the same names. In a later step (after the server is running), you will use qmgr to set the attributes. For each scheduler:

1. Make a copy of the new sched_config, which is in PBS_EXEC/etc/pbs_sched_config.

   cp $PBS_EXEC/etc/pbs_sched_config $PBS_EXEC/etc/pbs_sched_config.new

2. Update PBS_EXEC/etc/pbs_sched_config**.new** with any modifications that were made to your old scheduler configuration file, saved in (Windows) "%WINDIR%\TEMP\PBS_Backup\sched_priv.sched_config" or "%WINDIR%\TEMP\PBS_Backup\sched_priv_<multisched name>.sched_config", or in (Linux) %PBS_HOME/sched_priv/sched_config or %PBS_HOME/sched_priv_<multisched name>/sched_config.

3. If you were using vmem at the queue or server level before the upgrade, then after upgrading you must add vmem to the resource_unset_infinite sched_config option. Otherwise jobs requesting vmem will not run.

4. Move PBS_EXEC/etc/pbs_sched_config**.new** to the correct name and location, i.e. $PBS_HOME/sched_priv/sched_config or $PBS_HOME/sched_priv_**<multisched name>**/sched_config:

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv/sched_config

   or

   mv $PBS_EXEC/etc/pbs_sched_config.new $PBS_HOME/sched_priv_<multisched name>/sched_config

6.9.13.2 Clean Up Scheduler Attributes

For each scheduler, clean up the attributes saved in **"%WINDIR%\TEMP\PBS_Backup\sched_attrs.new"**. When you read in multisched attributes, you'll re-create the multischeds, so make sure your new multischeds are what you want:

- Remove read-only attributes
- Remove lines containing the following:

  pbs_version

For the new default scheduler and all new multischeds:

- The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
- The scheduler's log_filter configuration parameter is **obsolete**. The scheduler's log filter now uses the same bitmask system as the other daemons. The new default value is 767. Use qmgr to set the scheduler's log_events attribute to the value you want. See "Specifying Scheduler Log Events" on page 548 in the PBS Professional Administrator's Guide.

6.9.13.3 Clean Up Server Configuration

Remove read-only attributes from the server's configuration information in server.new. For example, remove lines containing the following:

```
license_count
pbs_version
```

Remove creation commands for any reservation queues. You will create reservations and their queues separately.

6.9.13.4 Clean up Vnode Configuration

Here you prepare the vnode attribute input to the new qmgr.

If your system has multi-vnode hosts:

- Copy your saved node configuration file "%WINDIR%\TEMP\PBS_Backup\nodes.new" into two files:
  - qmgr_parent_vnode.out, which contains all the configuration information for parent vnodes
  - qmgr_child_vnode.out, which contains all the configuration information for vnodes that aren't parent vnodes
- Continue by preparing configuration information for parent vnodes. You will prepare the configuration information for the child vnodes after they have been created, because the vnode names in your file must be precisely the same as the ones created by PBS.

If your system has only single-vnode hosts, follow the steps below for preparing configuration information for parent vnodes only.

6.9.13.4.i Prepare Configuration Information for Parent Vnodes

Edit qmgr_natural_vnode.out:

Leave only the the following creation lines:

- Those for parent vnodes
- Any resources you want managed on the server side through qmgr
- Custom resources on the parent vnodes

Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

- Child vnodes, that should be created by MoM (vnodes that are NOT parent vnodes)
- Lines that set the sharing attribute
- The ncpus, mem, and vmem resources, unless they should explicitly be set via qmgr

6.9.14 Create and Configure New Multischeds

Create the directories required for each new multisched, and configure each multisched. See "Creating and Configuring a Multisched" on page 55 in the PBS Professional Administrator's Guide.

6.9.15 Start New Server and New Schedulers

1. Start the new server and new default scheduler. On the server host:

   systemctl restart pbs

   or

   <path to init.d>/init.d/pbs restart

2. Start multischeds. To start a multisched, call pbs_sched and specify the name and port you already gave it. For each multisched:

   pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

6.9.16 Replicate Queue, Server, Scheduler, and Vnode Configurations

6.9.16.1 Replicate Server and Queue Attributes

1. Give the new server the old server's configuration, but modified for the new PBS:
   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/server.new
   ```
2. Verify the configuration was read in properly:
   ```
   $PBS_EXEC/bin/qmgr -c "print server"
   ```

6.9.16.2 Replicate Scheduler Attributes

1. Give the new default scheduler the old default scheduler's attributes, and re-create your multischeds:
   ```
   $PBS_EXEC/bin/qmgr < /tmp/pbs_backup/<scheduler name>/sched_attrs.new
   ```
2. Verify the configurations were read in properly.

   You can see all schedulers at once:
   ```
   $PBS_EXEC/bin/qmgr -c "print sched"
   ```
   Or for each scheduler:
   ```
   $PBS_EXEC/bin/qmgr -c "print sched default"
   or
   $PBS_EXEC/bin/qmgr -c "print sched <multisched name>"
   ```

6.9.16.3 Replicate Vnode Attributes

Replicate vnode configuration, also modified for the new PBS:

1. Read in the parent vnode configuration file:
   ```
   $PBS_EXEC/bin/qmgr < qmgr_natural_vnode.out
   ```
2. Wait until MoM or the cgroups hook creates any child vnodes. Check:
   ```
   pbsnodes -av
   ```
3. Prepare configuration information for child vnodes:

   Edit `qmgr_not_natural_vnode.out`. Make sure that the vnode names in this file are exactly what MoM or the cgroups hook created. It's easiest to put all resource information into a Version 2 configuration file, rather than using `qmgr`.

   Leave only the the following creation lines:
   - Any resources you want managed on the server side through `qmgr`
   - Custom resources on the child vnodes (but this may be easier in a Version 2 configuration file)

Delete any lines for resources managed through Version 2 configuration files or that MoM reports from what the vnode's host OS is reporting. For example, delete:

- Child vnodes, that should be created by the cgroups hook or MoM (vnodes that are NOT parent vnodes)
- Lines that set the sharing attribute
- The ncpus, mem, and vmem resources, unless they should explicitly be set via qmgr

4. Read in the configuration file for child vnodes (not parent vnodes):

```
$PBS_EXEC/bin/qmgr < qmgr_not_natural_vnode.out
```

5. Verify the configurations were read in properly:

```
$PBS_EXEC/bin/pbsnodes -a
```

6.9.17 Import and Configure Hooks

Make sure you do not overwrite the new pbs_cgroups hook or its configuration file by importing the old ones. Instead, use the saved information from your old hook to modify the new hook and configuration file.

6.9.17.1 Import Old Hooks Except for Cgroups Hook

1. Do not import your old pbs_cgroups hook. Import your other hooks and their configuration files. For each hook **except** for pbs_cgroups:

```
# qmgr -c 'import hook <hook name> application/x-python default  /tmp/<hook name>.new3.6'
# qmgr -c 'import hook <hook name> application/x-config default  /tmp/<hook name>.configcheck'
```

6.9.17.2 Modify Cgroups Hook Configuration File

If you will use the cgroups hook:

1. Export the new cgroups hook configuration file to pbs_cgroups.json:

```
# qmgr -c 'export hook pbs_cgroups application/x-config default' > pbs_cgroups.json
```

2. If the cgroups memory subsystem is not mounted on the system, disable 'memory' in the cgroups hook configuration file:

   a. Check to see whether it is mounted:

   ```
   # mount | grep cgroup | grep memory
   ```

   If the memory subsystem is mounted, the command returns something like "cgroup on /sys/fs/cgroup/memory type cgroup (rw,nosuid,nodev,noexec,relatime,memory".

   b. If this returns empty, edit the pbs_cgroups.json file so that 'enabled' parameter for 'memory' under cgroup is *false*:

   ```
   "cgroup": {
     ...
   "memory": {
   "enabled": false,
   ```

3. If you made changes to the old cgroups configuration file, you may want to make those changes in the new configuration file. Use the information saved in /etc/pbs_cgroups.old2.7

4. Import the modified configuration (make sure you use "**x-config**"):

```
# qmgr -c 'import hook pbs_cgroups application/x-config default pbs_cgroups.json'
```

6.9.17.3 Enable Cgroups Hook

If you will use the cgroups hook, enable the pbs_cgroups hook:

```
qmgr -c "set hook pbs_cgroups enabled=true"
```

6.9.17.4 Write and Deploy New Hooks

If you have written new hooks for the new version of PBS, deploy them now. See the *PBS Professional Hooks Guide*.

6.9.17.5 Start MoMs

On each execution host, start MoM :

```
net start pbs_mom
```

6.9.18 Configure Sharing and Placement Sets

6.9.18.1 Configuration with Cgroups Hook

As of version 2020.1, the cgroups hook creates the child vnodes on a multi-vnode machine; if you will use the cgroups hook, it is important that any Version 2 configuration files refer only to these vnodes. Use Version 2 configuration files only to set the sharing attribute and optionally to set resources that will be used for placement sets. The default value for the sharing attribute of the vnodes is "sharing=default_shared". You can change this, for example to "sharing=default_excl".

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs" that has MoM's list of vnodes; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide
2. Edit the file to reflect what you want for the sharing attribute and placement sets. Use the information saved in **"%WINDIR%\TEMP\PBS_MoM_Backup\mom_config"** in step "Save Execution Host Configuration Files" on page 127
3. Create your new Version 2 configuration file and name it for example "vnodedefs":

   ```
   # pbs_mom -s insert vnodedefs vnodedefs
   ```

4. Restart pbs_mom:

   ```
   net stop pbs_mom
   net start pbs_mom
   ```

6.9.18.2 Configuration without Cgroups Hook

Do **not** set resources_available.mem, resources_available.ncpus, or resources_available.vmem in the Version 2 configuration file.

On each execution host:

1. Create a file named "vnodedefs"; see "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide
2. Insert your new Version 2 configuration file and name it for example "vnodedefs":

```
# pbs_mom -s insert vnodedefs vnodedefs
```

3. Restart pbs_mom:

```
net stop pbs_mom
net start pbs_mom
```

6.9.19 Start New Communication Daemons

Start PBS on any communication-only hosts. On each communication-only host, type:

```
systemctl start pbs
```

or

```
<path to init.d>/init.d/pbs start
```

6.9.20 Verify Communication Between Server and MoMs

All new MoMs on all execution hosts should be running and communicating with the new server. Run `pbsnodes -a` on the new server host to see if it can communicate with the execution hosts in your complex. If a host is down, go to the problem host and restart the MoM:

```
net stop pbs_mom
net start pbs_mom
```

6.9.21 Re-create Reservations

You must re-create each reservation that was on the old server, using the `pbs_rsub` command. Each reservation is created as a new reservation. You can use all of the information about the old reservation except for its start time. Be sure to give each reservation a start time in the future. Use the information stored in `/tmp/pbs_backup/reservations`.

6.9.22 Enable STONITH Script

If your secondary server has a STONITH script, allow the STONITH script to run by setting its permissions to 0755.

6.9.23 Enable Cloud Bursting

If you are using Altair Control for cloud bursting with PBS, enable cloud bursting. See the *Altair Control Administrator's Guide*, at www.pbsworks.com.

6.9.24 Enable Scheduling

If you disabled scheduling earlier, enable it for the default scheduler and any multischeds:

```
qmgr -c 'set sched <scheduler name> scheduling = true'
```

6.10 After Upgrading

6.10.1 Making Upgrade Transparent for Users

You may wish to make the upgrade transparent for users, if the installation program hasn't done that already. See section 3.5.5, "Making User Paths Work", on page 36.

7 Installing and Upgrading on Cray

7.1 Installing PBS with Shasta

To install the PBS server and client packages on Shasta, follow the instructions supplied by Cray.

To install the PBS MoM and comm packages on Shasta, follow the standard Linux instructions. You can install comms on compute nodes. See Chapter 3, "Installation", on page 19.

7.1.1 Prerequisites for PBS on Shasta

If you want to be able to use `pbs_snapshot`, install the `file` command.

7.2 Prerequisites for Using Power Profiles with Cray XC

- The Cray XC `capmc` package must be running on XC30 or XC40 hardware with SMW software release 7.2.UP02 or later.
- Make sure that the `capmc` utility is available and working correctly in your PBS complex.

7.3 Support for IMPS and CLE 6 and 7

With CLE 6 and 7, Cray XC introduces a new mechanism to manage system images, because Cray XC is no longer using `xtopview` on a shared root. The IMPS (Image Management and Provisioning System) interface includes new commands to install and boot on a Cray XC. PBS Professional uses this new Cray XC installation mechanism to include PBS in the images.

This version of PBS allows a PBS SLES12 package to be added to a Cray XC image recipe, as well as building and installing an image using Cray XC's new IMPS command-line interfaces.

7.3.1 Prerequisites for IMPS

This version of PBS relies on the following Cray XC interfaces:

recipe

image

cfgset

image customization via `postbuild_copy` and `postbuild_chroot`

cnode

xtbootsys

7.3.2 Where to Run PBS Daemons for IMPS

MoMs may be run on any nodes where ALPS is configured, and the PBS server may be run on any nodes that have access to the license server.

7.4 Caveats and Advice for Installing and Upgrading on the Cray XC

- If you install the server/scheduler on a non-CLE machine, any automatic installation behavior on that machine is not Cray XC-specific. Therefore you need to make sure that all of your settings on that host are appropriate. See section 7.5.2.1, "Installing PBS on a Cray XC esLogin Node or Repurposed Compute Node", on page 142.
- When upgrading PBS on a Cray XC, you can use an overlay upgrade.
- All MoMs on X* series Cray machines must be the same version.
- For a list of supported operating systems, see the release notes.
- On CLE 5.2UP03, if you are using preemption via suspend/resume, you may see that after a SWITCH request, ALPS continues to start new applications, so that the system is never ready for PBS to suspend job processes. The system can go from RUNNING to EMPTY to RUNNING.
- If you start PBS with PBS_START_MOM = 0 for the first time, PBS does not create MoM directories on that host. If you want to use the machine as an execution host (running a MoM on it), set *PBS_START_MOM* = 1, run `pbs_habitat`, then start MoM.
- Beware of overflow: PBS stores fairshare allocations in a signed integer (32-bit on Linux x86_64 platforms), and fairshare usage in a long (64-bit on Linux x86_64 platforms).
- If you are upgrading and using failover, do not upgrade the primary and secondary servers simultaneously. Upgrade the primary first, then once that is complete, upgrade the secondary.

7.4.1 Where to Run PBS Server, Scheduler, Comm Daemons

The PBS Professional server, scheduler, and communication daemons can be run on an external host; they do not have to run on your Cray system. One instance of the `pbs_mom` daemon is run on each login node of the Cray XC system, allowing PBS Professional to select the login node to which a job is assigned.

The PBS Professional scheduler, server, and communication daemons can be run on any UNIX/Linux machine or node that fits the requirements. We include instructions for all options.

- Make sure you choose the correct PBS package for the server/scheduler/communication host or node. If you are installing PBS components on a Cray XC service node, install from the Cray XC package. If you are installing on a non-Cray XC host, choose the package for that host's OS.
- Make sure that the server/scheduler/communication host or node can reach the license server host.
- Make sure that the server/scheduler/communication host or node can communicate with the login nodes where PBS MoMs are running.
- Make sure that the server/scheduler/communication host meets the minimum requirements to run PBS. See section 2.1.2, "Resources Required by PBS", on page 7.

7.4.2 Licensing PBS

If you intend to use socket licenses for your Cray XC, you must have enough socket licenses for all of the sockets on the entire machine. Each PBS complex is licensed using either PBSProNodes or PBSProSockets licenses, but not a mix.

Install and configure the Altair license server before you install PBS. See the *Altair License Management System Installation and Operations Guide*, available at www.altair.com for information on installing and configuring the Altair license server(s).

To license a PBS complex, set the server pbs_license_info attribute to the location of the license server:

```
# qmgr -c 'set server pbs_license_info=<port>@<license server hostname>'
```

7.4.3 Hardware and Software Requirements

Confirm the following before beginning installation:

- You must have root permission.
- Make sure that all configuration files have been updated appropriately for your site.
- On the login nodes and on the host or node on which the PBS server and Scheduler will run, make sure that the `/var` directory is owned by root.
- Make sure your ALM license server is up and running and has enough licenses for your site.
- Compute nodes must be in the "batch mode" before they can be used by jobs under the control of PBS Professional. To set all of the compute nodes in the batch mode:

  On the boot node execute:

  ```
  xtprocadmin -k m batch
  ```

7.4.4 Caveats When Upgrading to PBS 2021.1

- If you have an existing queue that has a dot (".") in its name, PBS 2021.1 will not accept the queue name. The queue name specification in previous versions of PBS did not include a dot, but this was not enforced, and names with a dot could slide by. However, PBS 2021.1 enforces the specification and does not accept dots in queue names. The format for the name of a queue is a "PBS NAME"; for a description of PBS NAME, see the "Formats" chapter in the PBS Professional Reference Guide.
- PBS vnodes representing some types of Xeon Phi nodes do not have values for the PBScrayorder and PBScraylabel resources. Vnodes representing other kinds of nodes do have values for those resources.
- Unlike prior Cray CLE versions (Pearl and earlier), with CLE 6 and 7 `/var/spool` is not persistent by default. When you reboot a node, any `PBS_HOME/*_priv` configuration changes you may have made will be gone unless you use Cray's `cfgset` command to make particular directories (such as `PBS_HOME`) persistent. Please refer to Cray's `cfgset` man page for more information.
- This version of PBS supports IMPS but only so far as is necessary to support CLE 6 and 7.
- The provisioning hook called `PBS_xeon_phi_provision` that is shipped with this version of PBS is disabled by default. You must enable it according to the instructions in the PBS Professional Administrator's Guide:

  ```
  Qmgr: set pbshook PBS_xeon_phi_provision enabled=true
  ```
- If you installed PBS on a Cray XC or a non-Cray headnode, make sure that the inventory hook, PBS_alps_inventory_check, is enabled. List it:

  ```
  qmgr -c "list pbshook PBS_alps_inventory_check"
  ```

  If the hook is not enabled:

  ```
  qmgr -c "set pbshook PBS_alps_inventory_check enabled=true"
  ```

7.5 Installing PBS on the Cray XC

7.5.1 Changes for Cray XC Installation

- The default installation directory is now /opt/pbs. When PBS is installed on systems using the modules package (e.g. module load pbs), PATH is updated to include /opt/pbs/bin and MANPATH is updated to include /opt/pbs/man. The PBS module file is in PBS_EXEC/etc/modulefile.
- The order in which spec file scripts are run during an upgrade is described in https://www.ibm.com/developerworks/library/l-rpm2/. Because the spec file scripts were removing the PBS init script during an upgrade, the spec file scripts have been updated to avoid deleting this file and disabling PBS via chkconfig during an upgrade. When uninstalling (e.g. rpm -e), these instructions are called and PBS is disabled.
- On CLE 5.2, all PBS services are disabled by default in /etc/pbs.conf. On non-Cray XC systems, PBS services are enabled based on the type of installation. On Cray CLE 5.2 systems, additional configuration steps are required to specialize /etc/pbs.conf on various nodes where the services will run. As a result, PBS_START_SERVER, PBS_START_SCHED, PBS_START_COMM, and PBS_START_MOM are all set to zero by default on Cray CLE 5.2 systems only.
- The $alps_client entry in the MoM config file is automatically added for Cray X* series systems. The lines "$vnodedef_additive 0" and "$alps_client /opt/cray/alps/default/bin/apbasil" are added automatically on Cray XC systems only.

7.5.2 Installation Notes

7.5.2.1 Installing PBS on a Cray XC esLogin Node or Repurposed Compute Node

To install the MoM on a Cray XC esLogin node or a repurposed compute node, you should follow the appropriate install procedure for that operating system. You may use either:

- The "PBS for Cray XC" binaries package
  - You should not set $alps_client (because there is no ALPS client to connect to)
  - Because this binary expects to be able to talk to ALPS, you will see warnings and/or errors in the MoM logs because PBS complains in the logs when it can't talk to ALPS
- The appropriate Linux package

If the scheduler is installed on an esLogin node or repurposed compute node:

- Modify the $<sched_priv directory>/sched_config file resources: line to add "vntype"
- Set restrict_res_to_release_on_suspend to "*ncpus*"

7.5.3 Installing PBS on CLE 6 and 7

7.5.3.1 Caveats for Installing PBS on CLE 6 and 7

When you install PBS on CLE 6 and 7, some directories and files are not created or configured until PBS is started for the first time. These include PBS_HOME and /etc/pbs.conf.

PBS will report only the compute nodes that are up and in batch mode.

7.5.3.2 Installation Notes for CLE 6 and 7

These instructions rely on Cray's Image Management and Provisioning System (IMPS) features, and are based on the information in the section titled "Install Third-Party Software with a Custom Image" of the Cray document S-2559. See section "Miscellaneous Installation and Configuration Procedures", sub-section "Install Third-Party Software with a Custom Image Recipe". We list this and other useful Cray documents in section 7.9, "Cray XC References", on page 158. Please use the newest versions of the Cray documents.

7.5.3.3 Installation Steps for CLE 6 and 7

1. Create a repository. Follow Cray's instructions in the section titled "Install Third-Party Software with a Custom Image" in the *Software Installation and Configuration Guide* in Cray document S-2559.
2. Create a package collection. Follow Cray's instructions in Cray document S-2559. Use the RPM named "pbspro_server-*".
3. Create a new recipe and update it with the PBS Professional RPM. Follow Cray's instructions in Cray document S-2559.

   Each recipe must be based on the type of node where the image will run. For example, a service node is in the service image as a base sub-recipe named "service_cle_6.0up01_sles_12_x86-64_ari".

   If you will run the PBS Professional server/scheduler/comm on the sdb node and the MoM on the login nodes, you will need at least two different recipes, one for the server/scheduler/comm and one for the MoMs.

   a. Add the other RPMs required by PBS Professional to the recipe. Syntax:

      1. Provide `/bin/sh`. Add the repo named "sle-server_<version>_x86-64"

         smw# recipe update -r sle-server_<version>_x86-64 <recipe name>

         For example:

         smw# recipe update -r sle-server_12sp3_x86-64 <recipe name>

      2. Cray XC requires adding the `base` and `base_updates` repository pairs; add "sle-server_<version>_x86-64_updates". Syntax:

         smw# recipe update -r sle-server_<server>_x86-64_updates <recipe name>

         For example:

         smw# recipe update -r sle-server_12sp3_x86-64_updates <recipe name>

   b. Validate the image recipe. Follow Cray's instructions in Cray document S-2559.

   c. Repeat step 3 including parts a and b for each different type of node.

4. Build and push/package the image; follow Cray's instructions in Cray document S-2559.
5. If you are upgrading PBS, make sure that `PBS_EXEC = /opt/pbs`, and that `PBS_HOME` is set to the same path as before the upgrade.
6. Make `/etc/pbs.conf` persistent by creating any desired configuration sets. See Cray's documentation on configuration sets and the configurator.

   On each host where a PBS daemon or command will run, edit the Cray XC Ansible play named "simple sync" to include the information for each `/etc/pbs.conf`.

   a. It may be necessary to create different persistent `/etc/pbs.conf` files depending on how the system will be configured. When making the `/etc/pbs.conf` file persistent, include the following (at a minimum):

      ```
      PBS_EXEC=/opt/pbs
      PBS_SERVER=<hostname of the machine hosting the server>
      PBS_START_SERVER=<set this to 1 if server is to run on this host, 0 otherwise>
      PBS_START_SCHED=<set this to 1 if scheduler is to run on this host, 0 otherwise>
      PBS_START_COMM=<set this to 1 if a communication daemon is to run on this host, 0 otherwise>
      PBS_START_MOM=<set this to 1 if MoM is to run on this host, 0 otherwise>
      PBS_HOME=/var/spool/pbs
      PBS_CORE_LIMIT=unlimited
      PBS_SCP=/usr/bin/scp
      ```

   b. Add any other configuration changes to `/etc/pbs.conf` that need to persist across reboots.

7. Assign the new boot image to the nodes where PBS Professional is to be installed. This includes all nodes running

PBS Professional daemons (e.g. `pbs_server`, `pbs_mom`, etc.) and those requiring access to PBS commands (e.g. `qsub`, `qstat`, etc.). Include the configuration set from step 6 that contains the information that needs to be persistent. Follow Cray's instructions in Cray document S-2559. General syntax:

cnode update -i /var/opt/cray/imps/boot_images/<image name>.cpio -c <configuration set name> <cname-of-node-to-update or group-to-update>

8. Reboot the system. Follow Cray's instructions in Cray document S-2559.
9. If you are upgrading PBS and `PBS_HOME` is already persistent, you can skip this step. Log on to each node that hosts a PBS Professional daemon:
   a. Start PBS Professional on that node. For example, on the server node:

      ```
      # PBS_DATA_SERVICE_USER=<non-root user for the database> /etc/init.d/pbs start
      ```

   b. Stop PBS Professional (in order to make the `PBS_HOME` directory persistent in the next step)
10. Make `PBS_HOME` (e.g. `/var/spool/pbs`) persistent:

    See Cray's documentation on configuration sets and the configurator. Edit the Cray XC Ansible play named "Persistent Dirs". Modify the same configuration set that you created earlier in step 6.

    Unlike CLE 5.2 and earlier, with CLE 6 and 7 **`/var/spool` is not persistent by default**. Any configuration changes are lost when nodes are rebooted, unless you use Cray's `cfgset` command to make particular directories persistent. You may want to make `/var/spool/pbs`, `/etc/pbs.conf`, and others persistent.
11. Start PBS Professional. At this point PBS Professional will be up and running, connected to ALPS, and ready to be configured. You can add MoMs to the server, etc. See "Configuring PBS for Cray" on page 471 in the PBS Professional Administrator's Guide.
12. Optional step: PBS is shipped with a module file in `PBS_EXEC/etc/modulefile`. If your system uses modules, you can copy this module file to the appropriate location for your system configuration.

7.5.4 Installing PBS on CLE 5.2 and Older

7.5.4.1 Installation and Configuration Overview for CLE 5.2

1. If you will run the server, scheduler, and communication daemons on a non-Cray XC host, install the server there. Follow the steps in section 7.5.4.3, "Installing PBS Server, Scheduler, and Communication Daemons on Non-Cray XC Host", on page 148

   The new version of the PBS Professional software is now installed, and you are ready to proceed with configuration.
2. If you will run the server, scheduler, and communication daemons on a service node, configure the service node to run the server, scheduler, and communication daemons. Follow the steps in section 7.5.4.4, "Configuring Service Node to Run Server, Scheduler, and Communication", on page 149
3. Configure the PBS server, scheduler, and communication daemons. Follow the steps in section 7.5.4.5, "Configuring PBS Server, Scheduler, and Communication Daemons", on page 150
4. Configure the PBS MoMs. Follow the steps in section 7.5.4.6, "Configuring MoMs on the Cray XC", on page 151
5. This step is optional. Configure startup for PBS Professional only if you need to change your startup configuration. Follow the steps in section 7.5.4.7, "Configuring PBS Professional Startup and Shutdown", on page 152

7.5.4.2 Installation Steps on CLE 5.2

1. Log in to the boot node.
2. If the /rr/current/software/pbspro directory is not present, create it.
3. Copy the PBS server RPM to the /rr/current/software/pbspro directory on the boot node. Make sure it is the only RPM file present in the directory.
4. Set the compute nodes to *batch*:

   ```
   boot# xtprocadmin -k m batch
   ```

5. Install the PBS RPM by running the following command on the boot node. Adjust the value of PBS_SERVER to the name of the node where the scheduler/server/comm services will run. For a fresh install run the following:

   ```
   boot# xtopview -d /rr/current/software/pbspro -m "Installing PBS Pro" -e "PBS_SERVER=sdb rpm -i
       /mnt/pbspro-server-*.rpm"
   ```

6. Check to make sure PBS_SERVER is set correctly in /etc/pbs.conf and edit if necessary. All PBS services should be disabled:

   ```
   boot# xtopview -e "cat /etc/pbs.conf"
   PBS_EXEC=/opt/pbs
   PBS_SERVER=sdb
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_COMM=0
   PBS_START_MOM=0
   PBS_HOME=/var/spool/pbs
   PBS_CORE_LIMIT=unlimited
   PBS_SCP=/usr/bin/scp
   ```

7. Update /etc/pbs.conf for the login nodes: set the value of PBS_START_MOM to *1*:

   ```
   boot# xtopview -c login -e "xtspec /etc/pbs.conf"
   boot# xtopview -c login -e "vi /etc/pbs.conf"
   ***File /etc/pbs.conf was MODIFIED
   boot# xtopview -c login -e "cat /etc/pbs.conf"
   PBS_EXEC=/opt/pbs
   PBS_SERVER=sdb
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_COMM=0
   PBS_START_MOM=1
   PBS_HOME=/var/spool/pbs
   PBS_CORE_LIMIT=unlimited
   PBS_SCP=/usr/bin/scp
   ```

8. Determine the NID of the node that will run the PBS server and scheduler. In this example, the SDB node will be used:

   ```
   boot# ssh sdb cat /proc/cray_xt/nid
   5
   ```

9. Use the value returned (in this example, the value is *5*) as the argument to the -n parameter and update the

/etc/pbs.conf settings for the PBS server. The values of PBS_START_SERVER, PBS_START_SCHED, and PBS_START_COMM should all be set to *1*:

```
boot# xtopview -n 5 -e "xtspec /etc/pbs.conf"
boot# xtopview -n 5 -e "vi /etc/pbs.conf"
***File /etc/pbs.conf was MODIFIED
boot# xtopview -n 5 -e "cat /etc/pbs.conf"
PBS_EXEC=/opt/pbs
PBS_SERVER=sdb
PBS_START_SERVER=1
PBS_START_SCHED=1
PBS_START_COMM=1
PBS_START_MOM=0
PBS_HOME=/var/spool/pbs
PBS_CORE_LIMIT=unlimited
PBS_SCP=/usr/bin/scp
```

10. The PBS data service runs on the PBS server node. The processes must be owned by an account other than root (e.g. pbsdata, postgres, etc.). If you are performing these steps while upgrading a version of PBS prior to 17, PBS_HOME already exists on the PBS server node and the file PBS_HOME/server_priv/db_user is already populated with the name of the account. If this is a clean install, there are two options.

    - Create a Linux user account named *pbsdata* on the PBS server host. That is the default account name used by the PBS data service:

      ```
      boot# xtopview -n 5 -e "useradd -c 'PBS Pro Dataservice' -d /home/users/pbsdata -m pbsdata"
      boot# ssh sdb
      === Welcome to sdb ===
      sdb# /etc/init.d/pbs start
      ```

    OR

    - Specify the data service account name when starting PBS for the first time. In this example, the SDB node will be running the PBS data service as the user *postgres*. The postgres account must already exist prior to starting PBS for the first time:

      ```
      boot# ssh sdb
      === Welcome to sdb ===
      sdb# PBS_DATA_SERVICE_USER=postgres /etc/init.d/pbs start
      ```

11. Start the PBS service on each execution host:

    ```
    sdb# ssh nid00030 /etc/init.d/pbs start
    ```

12. Enable flatuid on the server:

    ```
    sdb# qmgr -c "set server flatuid = true"
    ```

13. Tell the PBS server where to find the license server. Set the pbs_license_info server attribute via qmgr. For example:

    ```
    sdb# qmgr -c "set server pbs_license_info = 6200@licenseserver"
    ```

14. With licensing now configured, restart PBS on the SDB node:

    ```
    sdb# /etc/init.d/pbs restart
    ```

15. Working on the server node, configure the execution hosts. Create a node in PBS for each login/service node that will be running pbs_mom:

```
sdb# qmgr -c "create node nid00030"
```

The installation of PBS should now be complete. The PBS services should be up and running, and the system should be ready for jobs to be submitted.

7.5.4.3 Installing PBS Server, Scheduler, and Communication Daemons on Non-Cray XC Host

If you will run the PBS server, scheduler, and communication daemons on the Cray XC, skip this section. If you will run the PBS server, scheduler, and communication daemons on a non-Cray XC host, do the following:

1. Log on to the server host as root.
2. Unzip and untar the install package.
3. Change to the directory created when the package was untarred.
4. Run the INSTALL script: accept the default locations, and select the number *1* (server, scheduler, communication, execution and commands) option.
5. When asked if you wish to start PBS, answer "*no*".
6. Make sure that MoM does not run on the server/scheduler/communication host. Modify `/etc/pbs.conf` by changing the line "`PBS_START_MOM=1`" to "`PBS_START_MOM=0`".
7. Ensure that the `PBS_SERVER` entry in `pbs.conf` names the machine on which you plan to run the server, scheduler, and communication daemons and that `PBS_EXEC` is set to `/opt/pbs`.
8. Log in as root to the host on which the PBS server `pbs_server` will be running.

   ```
   # ssh root@<server host>
   ```

9. At this point, you **must** start the PBS server with the `init.d/pbs` script. Do not start the server by any other means.

   ```
   <host name> # /etc/init.d/pbs start
   ```

   This will create or update the PBS working directories, `PBS_HOME`, in `/var/spool`.

7.5.4.4 Configuring Service Node to Run Server, Scheduler, and Communication

If you will run the PBS server, scheduler, and communication daemons on a non-Cray XC host, skip this section. If you will run the PBS server, scheduler, and communication daemons on a service node, configure the service node to run the PBS server, scheduler, and communication daemons:

1. Log into the boot node.
2. Determine the node ID, or NID, of the node on which you plan to run the PBS server, scheduler, and communication daemons.
3. Tailor the install to run the PBS server, scheduler, and communication daemons on the service node:

   Use the `xtopview` command:

   ```
   # xtopview -m "pbs.conf for Server" -n <NID>
   ```

   Run the `xtspec` command to specialize the change to `/etc/pbs.conf` for the service node:

   ```
   <nodeID>: # xtspec /etc/pbs.conf
   ```

4. Enable the server, scheduler, and communication daemons on the service node. Edit the `/etc/pbs.conf` file or verify that it contains the following lines:

   ```
   PBS_START_SERVER=1
   PBS_START_SCHED=1
   PBS_START_COMM=1
   ```

5. Verify that the `MoM` start line in `/etc/pbs.conf` is:

   ```
   PBS_START_MOM=0
   ```

6. Ensure that the `PBS_SERVER` entry in `pbs.conf` names the node on which you plan to run the server, scheduler, and communication daemons and that `PBS_EXEC` is set to `/opt/pbs`.
7. Exit from `xtopview`.

7.5.4.5 Configuring PBS Server, Scheduler, and Communication Daemons

1. If you are on a Cray XC node, load the pbs module:

   ```
   <service node>: # module load pbs
   ```

2. Using the `qmgr` command, create the PBS execution node or nodes, which are the login nodes.

   When creating a vnode to represent a login node, use the short name returned by the `hostname` command on the login node. For example, if `hostname` returns *HostA*, do the following:

   ```
   Qmgr: create node HostA
   ```

   If you create a vnode with a different name from the short name returned by `hostname`, the following happens:

   - MoM creates a vnode whose name is the short name returned by `hostname`
   - The vnode you created is not recognized by MoM, and is marked *stale*

3. The execution queue *workq* is automatically created during the normal PBS post-install procedure.

4. Make sure that flatuid is set to *True* in the PBS server configuration:

   ```
   <server host>: # qmgr -c "list server"
   ```

   If flatuid is not set to true, set it to true:

   ```
   <server host>: # qmgr -c "set server flatuid=true"
   ```

5. To ensure that scheduling of jobs is initiated when PBS is restarted, set the server's scheduling attribute to true:

   ```
   <server host>: # qmgr -c "set server scheduling=true"
   ```

6. Shut down the PBS daemons using the following command:

   ```
   /etc/init.d/pbs stop
   ```

7.5.4.6 Configuring MoMs on the Cray XC

Configure each login node to run a PBS MoM:

1. As root, log in to the boot node:
   # ssh root@boot
2. Tailor the install to run MoMs on the login nodes:

   Use the `xtopview` command:

   xtopview -m "pbs.conf for login" -c login

   Run the `xtspec` command to specialize the change to `/etc/pbs.conf` for the login nodes:

   class/login/**: # xtspec /etc/pbs.conf**
3. Enable MoM. Edit `/etc/pbs.conf` ensuring the following lines are present:

   ```
   PBS_START_MOM=1
   ```
4. Make sure that the server, scheduler, and communication daemons do not start:

   ```
   PBS_START_SERVER=0
   PBS_START_SCHED=0
   PBS_START_COMM=0
   ```
5. If you want each login node to run a `pbs_comm`, configure the `pbs_comms` and MoMs.

   a. Enable the communication daemon. Edit `pbs.conf` to set this variable:

      ```
      PBS_START_COMM=1
      ```

   b. Tell each MoM about its `pbs_comm` by adding this line to its `pbs.conf`:

      ```
      PBS_LEAF_ROUTERS=<host>[:<port>][,<host>[:>port>]]
      ```

   c. Tell each `pbs_comm` where to find any already-configured `pbs_comm`, by adding this to its `pbs.conf`:

      ```
      PBS_COMM_ROUTERS=<host>[,<host>]
      ```

   For details, see section 4.5.3, "Adding Communication Daemons", on page 50.
6. Ensure that the `PBS_SERVER` entry names the host or node on which you plan to run the server, scheduler, and communication daemons.
7. Ensure that `PBS_EXEC` is set to `/opt/pbs.`
8. Ensure that `/opt/pbs` is a symbolic link to the location you specified for the binaries.
9. Exit from `xtopview`.

On each login node, log in and perform the following steps:

1. Start the PBS MoM daemon with the `init.d/pbs` script. Do not use any other means to start `pbs_mom`. This will create or update the PBS working directories, `PBS_HOME`, in `/var/spool`.

   ```
   Login:~ # /etc/init.d/pbs start
   ```

2. Stop the MoM daemon by running the `init.d/pbs` script:

   ```
   Login:~ # /etc/init.d/pbs stop
   ```

3. Add the following directives in the `pbs_mom` configuration file, `PBS_HOME/mom_priv/config`:

   For CLE:

   ```
   $clienthost <server host or node>
   $alps_client /usr/bin/apbasil (CLE 4.1 and earlier)
   $alps_client /opt/cray/alps/default/bin/apbasil (CLE 5.0 and later)
   ```

   The sample configuration is now complete.

4. HUP the MoM.

7.5.4.7 Configuring PBS Professional Startup and Shutdown

The PBS start/stop script is located in `/etc/init.d/pbs`. To use the `init.d` script and the `chkconfig` command to start up PBS Professional, execute the following steps:

1. If you will run the PBS server, scheduler, and communication daemons on a Cray XC service node, find the node ID (NID) of the node on which the PBS server, scheduler, and communication daemons will run. For example, on the sdb node:

   ```
   grep sdb /etc/hosts
   ```

2. If you will run the PBS server, scheduler, and communication daemons on a Cray XC service node, enable PBS startup for the server, scheduler, and communication daemons on the service node. This example uses an NID of 3.

   ```
   boot001: # xtopview -n 3
   node/3:/ # chkconfig pbs on
   node/3:/ # exit
   ```

3. Enable PBS startup for the MoM on each login node:

   ```
   # xtopview -c login
   class/login/: # chkconfig pbs on
   class/login/: # exit
   ```

7.6 After Installing on the Cray XC

- We recommend that you set node_fail_requeue to *0* (zero) on any complex that manages Cray X* series machines. See "Prevent Jobs from Being Requeued on Cray XC" on page 481 in the PBS Professional Administrator's Guide.
- If you installed PBS on a Cray XC or a non-Cray XC headnode, make sure that the inventory hook, PBS_alps_inventory_check, is enabled. List it:

  ```
  qmgr -c "list pbshook PBS_alps_inventory_check"
  ```

  If the hook is not enabled:

  ```
  qmgr -c "set pbshook PBS_alps_inventory_check enabled=true"
  ```

7.7 Upgrading on the Cray XC

7.7.1 Upgrading on CLE 6 and 7

7.7.1.1 Overlay when Existing PBS Version is 13.0.40x or Lower

Prior to version 18, PBS used a script named INSTALL to install and configure the RPMs. The INSTALL script provided support for installing multiple versions simultaneously, using a symbolic link (/opt/pbs/default) to select the "active" version on the system. As of version 18, the INSTALL script is no longer supported. The new package allows you to install and upgrade PBS as you would any other RPM-based package.

1. Make sure that PBS_HOME is persistent.
2. Drain the system of running jobs. Retain queued jobs.
3. Remove existing vnodes. Prior to 18.x, PBS created one vnode per NUMA node, but from 18.x forward, PBS creates one vnode per compute node. We remove all older vnodes because they will be marked as stale and may be difficult to remove later.

   ```
   # qmgr -c "delete node @default"
   ```

4. Shut down PBS on all the nodes where a PBS daemon is running:

   ```
   # /etc/init.d/pbs stop
   ```

5. Install PBS: use the same configuration set that was used for the previous installation of PBS. Use the installation steps in section 7.5.3.3, "Installation Steps for CLE 6 and 7", on page 144.
6. The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
7. PBS 19.4.1 and later uses Python 3.6. Update all of your site-defined hooks (not the built-in hooks) to Python 3.6.
   a. Export your hooks in ASCII format. For **each** hook:

      ```
      # qmgr -c 'export hook <hook name> application/x-python default  <hook name>.old2.7'
      ```

   b. Update your hooks to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your new hook file differently; use something like "<hook name>.new3.6".
   c. Import your hooks. For **each** hook:

      ```
      # qmgr -c 'import hook <hook name> application/x-python default  <hook name>.new3.6'
      ```

7.7.1.2 Overlay When Existing PBS Version is 18.2 or Higher

For the steps below, we will use the original repository, package collection, and recipes that were used for the existing version of PBS (the one that we are about to replace.)

1. Make sure that `PBS_HOME` is already persistent.
2. Drain the system of running jobs. Retain queued jobs.
3. Shut down PBS on all the nodes where a PBS daemon is running:

   ```
   # /etc/init.d/pbs stop
   ```

4. Update the original repository. Follow Cray's instructions for updating a repository; refer to the section titled "Install Third-Party Software with a Custom Image" in the *Software Installation and Configuration Guide* in Cray document S-2559. Use the same repository that was used for the existing installed image. General syntax:

   repo update -a "<path to new PBS server package> <repository name>

   For example, if the original repository was called *my_repo*, the command might look something like this:

   ```
   smw# repo update -a "./pbspro-server-18.2.0.20171109010636-0.x86_64.rpm" my_repo
   ```

5. Update the original package collection. Follow Cray's instructions in Cray document S-2559.

   a. Use the `pkgcoll update` command. General syntax:

      pkgcoll update -p <new PBS server package> <package collection name>

      For example, if the original package collection was called *my_collection*, the command might look something like this:

      ```
      smw# pkgcoll update -p pbspro-server-18.2.0.20171109010636-0.x86_64 my_collection
      ```

   b. See what's in the package collection. Use the `pkgcoll` command. General syntax:

      pkgcoll show <package collection name>

      Following our example:

      ```
      smw# pkgcoll show my_collection
      my_collection:
      name: my_collection
      packages:
      pbspro-server-18.2.0.20171018010818-0.x86_64
      pbspro-server-18.2.0.20171109010636-0.x86_64
      ```

   c. Remove the older version of PBS. Use the package collection `update` option. General syntax:

      pkgcoll update -P <PBS package> <package collection name>

      Following our example:

      ```
      smw# pkgcoll update -P pbspro-server-18.2.0.20171018010818-0.x86_64 my_collection
      ```

   d. Check to see that there is only one PBS RPM version. Use the package collection `show` command. General syntax:

      pkgcoll show <package collection name>

      This command should now show only one PBS RPM version.

6. Validate the recipe; follow Cray's instructions in Cray document S-2559. Even though changes have been made to the repository and the package collection, the original recipe should still validate at this point.
7. Build and push/package each type of image that you need such as server, login, etc.

   Build and push/package the image using the original recipe; follow Cray's instructions in Cray document S-2559. Use a new image name in order to keep the original image untouched. General syntax:

image create -r <original recipe> <new image name>

image export <new image name>

For example, if the original recipe name was *my_recipe* and the original image created was *my_recipe_image*, then the new image could be *my_new_image* and the command might look something like this:

```
smw# image create -r my_recipe my_new_image
smw# image export my_new_image
```

8. Assign the new boot image to each node where PBS Professional should be installed; follow Cray's instructions in Cray document S-2559. This includes all nodes running PBS Professional services such as `pbs_server`, `pbs_mom`, etc., and those requiring access to PBS commands, e.g. `qsub`, `qstat`, etc. Include the original configuration set that contains the persistent information. General syntax:

   smw# cnode update -i /var/opt/cray/imps/boot_images/<image name>.cpio -c <configuration set name> <cname-of-node-to-update or group-to-update>

9. Reboot the system.

10. Log on to each node hosting a PBS Professional daemon, and start PBS.

    ```
    # /etc/init.d/pbs start
    ```

11. At this point PBS Professional should be up and running, and connected to ALPS. Prior queued jobs should still be queued.

12. Turn on scheduling so PBS can resume normal operation.

13. The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in `$PBS_HOME/sched_priv/sched_config` in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use `qmgr` to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.

14. PBS 19.4.1 and later uses Python 3.6. Update all of your site-defined hooks (not the built-in hooks) to Python 3.6.

    a. Export your hooks in ASCII format. For **each** hook:

       ```
       # qmgr -c 'export hook <hook name> application/x-python default  <hook name>.old2.7'
       ```

    b. Update your hooks to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your new hook file differently; use something like "<hook name>.new3.6".

    c. Import your hooks. For **each** hook:

       ```
       # qmgr -c 'import hook <hook name> application/x-python default  <hook name>.new3.6'
       ```

7.7.2 Upgrading on CLE 5.2

7.7.2.1 Overlay when Existing PBS Version is 13.0.40x or Lower

Prior to version 18, PBS used a script named INSTALL to install and configure the RPMs. The INSTALL script provided support for installing multiple versions simultaneously, using a symbolic link (/opt/pbs/default) to select the "active" version on the system. As of version 18, the INSTALL script is no longer supported. The new package allows you to install and upgrade PBS as you would any other RPM based package.

Due to the significant packaging changes, we recommend that you uninstall the old version of PBS prior to installing the new version. Uninstalling the old version after installing the new version will prevent PBS from starting automatically at boot. The /etc/pbs.conf file and contents of the PBS_HOME directories will not be affected when PBS is uninstalled.

The following instructions assume that the PBS server is running on the SDB node and that the PBS MoMs are running on the login nodes. Please adjust the commands according to your local configuration.

1. Drain the system of running jobs. Retain queued jobs.
2. Remove all existing vnodes:

   ```
   sdb# qmgr -c "delete node @default"
   ```

3. Shut down PBS on all login nodes:

   ```
   login# /etc/init.d/pbs stop
   ```

4. Shut down PBS on the server:

   ```
   /etc/init.d/pbs stop
   ```

5. Remove specialization of the existing /etc/pbs.conf file:

   ```
   boot# xtopview -e "xtunspec -N /etc/pbs.conf"
   boot# xtopview -e "xtunspec -C /etc/pbs.conf"
   ```

6. There may be more than one version of PBS installed. Get the list of all currently installed PBS RPMs:

   ```
   boot# xtopview -e "rpm -qa" | grep pbs
   ```

7. Remove each installed version of PBS:

   ```
   boot# xtopview -e "rpm -e pbspro-<version>"
   ```

8. Remove the /opt/pbs directory:

   ```
   xtopview -e "rm -rf /opt/pbs"
   ```

9. PBS is now completely uninstalled, but the /etc/pbs.conf and PBS_HOME directories remain. Leave these files in place and follow the steps to perform a fresh install of PBS in section 7.5.4, "Installing PBS on CLE 5.2 and Older", on page 145.
10. The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
11. PBS 19.4.1 and later uses Python 3.6. Update all of your site-defined hooks (not the built-in hooks) to Python 3.6.
    a. Export your hooks in ASCII format. For **each** hook:

       ```
       # qmgr -c 'export hook <hook name> application/x-python default  <hook name>.old2.7'
       ```

    b. Update your hooks to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your new hook file differently; use something like "<hook name>.new3.6".

c. Import your hooks. For **each** hook:

```
# qmgr -c 'import hook <hook name> application/x-python default  <hook name>.new3.6'
```

7.7.2.2 Overlay When Existing PBS Version is 18.2 or Higher

The following instructions assume that the PBS server is running on the SDB node and that the PBS MoMs are running on the login nodes. Please adjust the commands according to your local configuration.

1. Log in to the boot node.
2. If the /rr/current/software/pbspro directory is not present, create it.
3. Copy the new PBS server RPM to /rr/current/software/pbspro on the boot node.
4. Drain the system of running jobs. Retain queued jobs.
5. Shut down PBS on all login nodes:

```
login# /etc/init.d/pbs stop
```

6. Shut down PBS on the server:

```
sdb# /etc/init.d/pbs stop
```

7. On the boot node, run this command:

```
boot# xtopview -d /rr/current/software/pbspro -m "Upgrading PBS Pro" -e "rpm -U
    /mnt/pbspro-server-*.rpm"
```

8. Start the PBS server:

```
sdb# /etc/init.d/pbs start
```

9. The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in $PBS_HOME/sched_priv/sched_config in older versions of PBS. They are now scheduler attributes with the same names and formats. Make sure that you use qmgr to set the attributes as desired. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.
10. On each machine that will host a MoM, start the MoM:

```
login# /etc/init.d/pbs start
```

11. PBS 19.4.1 and later uses Python 3.6. Update all of your site-defined hooks (not the built-in hooks) to Python 3.6.
    a. Export your hooks in ASCII format. For **each** hook:

```
# qmgr -c 'export hook <hook name> application/x-python default  <hook name>.old2.7'
```

    b. Update your hooks to Python 3.6. See https://docs.python.org/3.6/howto/pyporting.html. Name your new hook file differently; use something like "<hook name>.new3.6".
    c. Import your hooks. For **each** hook:

```
# qmgr -c 'import hook <hook name> application/x-python default  <hook name>.new3.6'
```

7.8 After Upgrading on Cray XC

7.8.1 Check qstat Wrappers and Scripts

You must check any qstat wrappers or scripts. Either replace any mpp* resources, or modify the wrapper or script to use values in the job's Submit_arguments attribute. This attribute contains the original job submission line.

7.8.2 Configure PBS to Support Cray XC

PBS provides tight integration with Cray XC. To take advantage of the tight integration, go to "Configuring PBS for Cray" on page 471 in the PBS Professional Administrator's Guide.

7.8.3 Enable Inventory Hook

If you upgraded PBS on a Cray XC or a non-Cray XC headnode, make sure that the inventory hook, PBS_alps_inventory_check, is enabled.

7.9 Cray XC References

XC Series Software Installation and Configuration Guide S-2559

https://pubs.cray.com/content/S-2559/CLE%206.0.UP04/xctm-series-software-installation-and-configuration-guide-cle-60up04-s-2559-rev-b/install-third-party-software-with-a-custom-image-recipe

XC Series Configurator User Guide S-2560

https://pubs.cray.com/content/S-2560/CLE%206.0.UP04/xctm-series-configurator-user-guide-cle-60up04-s-2560/about-variable-names-in-the-configurator-and-configuration-worksheets

XC Series DVS Administration Guide S-0005

https://pubs.cray.com/content/S-0005/CLE%206.0.UP05/xctm-series-dvs-administration-guide

8 Starting & Stopping PBS on Linux

8.1 Automatic Start on Bootup

On installation, PBS is configured to start automatically. Under Linux, PBS starts on bootup using `init` or `systemd`. PBS uses `systemd` for automatic startup on platforms that support `systemctl`; for platforms that support only `init`, PBS uses `init` for automatic startup.

You specify which PBS daemons start on each host on bootup in that host's `/etc/pbs.conf`. The table below lists the parameters that control startup of daemons:

Table 8-1: Daemon Start Parameters in pbs.conf

Parameter	Description
PBS_START_COMM	Set this to *1* if a communication daemon is to run on this host.
PBS_START_MOM	Default is *0*. Set this to *1* if a MoM is to run on this host.
PBS_START_SCHED	Set this to *1* if a scheduler is to run on this host.
PBS_START_SERVER	Set this to *1* if server is to run on this host.

8.1.1 Shutting Down Host

When a host running PBS is shut down or rebooted, PBS is shut down via the start/stop script or `systemd`.

8.2 When to Restart PBS Daemons

- Restart PBS if you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS. You should restart PBS by typing the following:

  ```
  <path-to-script>/pbs restart
  ```

- Restart PBS after making changes to the `/etc/hosts` file. See section 2.1.3, "Name Resolution and Network Configuration", on page 8
- Restart PBS after changing the name of the PBS service account
- Restart the scheduler(s) if you added a new custom resource to the `resources`: line in `sched_config`

8.3 Methods for Starting, Stopping, or Restarting PBS

The PBS daemons can be started by different types of methods. These types are not equivalent. You can use `init`, `systemd`, or the PBS command that starts the daemon.

The following table shows how to start, stop, restart, or status PBS on the local host:

Table 8-2: Commands to Start, Stop, Restart, Status PBS

Effect	`init`	`systemd`	Command

Start PBS	`/etc/init.d/pbs start`
	or
	`/etc/rc.d/init.d/pbs start`
		`systemctl start pbs`	`pbs_server`
			`pbs_sched`
			`pbs_mom`
			`pbs_comm`

Stop PBS	`/etc/init.d/pbs stop`
	or
	`/etc/rc.d/init.d/pbs stop`
		`systemctl stop pbs`	`qterm` (stops server, scheduler(s), MoM)
			`kill -INT <PID of pbs_comm>`

Status PBS	`/etc/init.d/pbs status`
	or
	`/etc/rc.d/init.d/pbs status`
		`systemctl status pbs`	`qstat`

Restart PBS	`/etc/init.d/pbs restart`
	or
	`/etc/rc.d/init.d/pbs restart`
		`systemctl restart pbs`	`---`

8.3.1 Using systemd

When you use `systemctl` to start PBS, it uses a PBS unit file. PBS supports `systemd` where it's available.

8.3.1.1 Required Privilege

You must be root to run `systemctl`.

8.3.1.2 Effect of `systemctl` on Jobs

When you use `systemctl` to start or stop PBS, any running jobs and subjobs are killed. When you use `systemd` to stop PBS, MoM kills her jobs and exits. When you use it to restart PBS, jobs are requeued.

When you use `systemd` by typing "systemctl stop pbs", the following take place:

- The server gets a `qterm -t quick`
- MoM gets a SIGTERM: MoM terminates all running children and exits
- The communication daemon gets a SIGTERM and exits

8.3.1.3 Caveats for Using systemctl

PBS supports most `systemctl` options, including `start`, `stop`, `restart`, and `status`. However, PBS does not support the `reload` option.

`systemd` uses the settings in `pbs.conf` to determine which daemons to start and stop. If you specify in `pbs.conf` that a daemon should not start, `systemd` also will not stop it if it is running. For example, setting PBS_START_MOM to *0* effectively makes `systemd` ignore the MoM, and if you do the following steps, the `pbs_mom` process is **not** stopped:

1. Start `pbs_mom`
2. Set PBS_START_MOM to *0*
3. Run `systemd` with *stop* as the argument

8.3.2 Using init with PBS Start/Stop Script

When you use `init` to start PBS, `init` runs the PBS start/stop script. PBS supports `init` on all Linux systems.

The script starts, stops, or restarts PBS daemons on the local machine. It can also be used to report the PID of any PBS daemon on the local machine. The PBS start/stop script reads the `pbs.conf` file to determine which components should be started. The start/stop script runs automatically at boot time, starting PBS upon bootup. The start/stop script runs on and affects only the local host.

The PBS start/stop script is named `pbs`. To run it, you type the following:

```
<path to script>/pbs [start|stop|restart|status]
```

See "pbs" on page 28 of the PBS Professional Reference Guide.

8.3.2.1 Required Privilege

You must be root to run the start/stop script.

8.3.2.2 Using Start/Stop Script to Check Status of Daemons

You can check whether or not each daemon is running by using the PBS start/stop script with the status option. To check the status of MoM, do the following on MoM's host:

```
<path to script>/pbs status
```

8.3.2.3 Location of the PBS Start/Stop Script

If `/etc/init.d` exists

```
/etc/init.d/pbs
```

Else

```
/etc/rc.d/init.d/pbs
```

8.3.2.4 Effect of Start/Stop Script on Jobs

When you use the PBS start/stop script to start or stop PBS, any running jobs and subjobs are killed on the host where you run the script. When you use the PBS start/stop script to stop PBS on the local host, MoM kills her jobs and exits. When you use it to restart PBS, jobs are requeued; note that there is a short but non-zero amount of time after MoM and the server are restarted, when jobs from MoM's previous session are visible via `qstat` but not running, before the server requeues them. If you stop one MoM for a multihost job, that job will probably be killed.

When you use the PBS start/stop script by typing "pbs stop", the following take place:

- The server gets a `qterm -t quick`
- MoM gets a SIGTERM: MoM terminates all running children and exits
- The communication daemon gets a SIGTERM and exits

8.3.2.5 Start/Stop Script Caveats

- The PBS start/stop script uses the settings in `pbs.conf` to determine which daemons to start and stop. If you specify in `pbs.conf` that a daemon should not start, the script also will not stop it if it is running. For example, setting PBS_START_MOM to *0* effectively makes the start/stop script or `systemd` ignore the MoM, and if you do the following steps, the `pbs_mom` process is **not** stopped:
  a. Start *`pbs_mom`*
  b. Set PBS_START_MOM to *0*
  c. Run the PBS start/stop script or `systemd` with *stop* as the argument
- If you start PBS using the start/stop script, you cannot use `systemctl` to status PBS.

8.3.3 Using the `qterm` Command to Stop PBS

You use the `qterm` command to shut down your choice of the following PBS daemons:

- Primary server
- Secondary server
- Whichever default scheduler is running (primary or secondary)
- All MoMs

The `qterm` command does not shut down `pbs_comm`.

If you have failover configured, you can choose to shut down either or both servers, or you can shut down the primary and leave the secondary idle.

You can specify how running jobs and subjobs are treated during shutdown by specifying the type of shutdown. The type of shutdown performed by the `qterm` command defaults to "-t quick", which preserves running jobs and subjobs:

```
qterm -t quick
```

The following command shuts down the primary server, the scheduler(s), and all MoMs in the complex. If configured, the secondary server becomes active. Running jobs and subjobs continue to run:

```
qterm -s -m
```

The following command shuts down the primary server, the secondary server, the scheduler(s), and all MoMs in the complex. Running jobs and subjobs continue to run:

```
qterm -s -m -f
```

See "qterm" on page 233 of the PBS Professional Reference Guide.

8.3.3.0.i `qterm` Caveats

- The `qterm` command does not stop the `pbs_comm` daemon. You must stop `pbs_comm` using the start/stop script, `systemd`, the service command, or the `kill` command.
- Shutting PBS down using the `qterm` command does not perform any of the other cleanup operations that are performed by the PBS start/stop script.

8.4 Starting, Stopping, and Restarting PBS Daemons

8.4.1 Daemon Execution Requirements

The server, scheduler(s), communication, and MoM processes must run with the real and effective UID of root.

8.4.2 Required Privilege

You must be root to run `pbs_server`, `pbs_mom`, `pbs_comm`, and `pbs_sched`.

8.4.3 Recommendation for Daemon Start Order

We recommend starting the communication daemon before starting the MoMs, but you can also start it after the MoMs and before the server.

We recommend starting MoMs before starting the server. This way, MoM will be ready to respond to the server's "are you there?" ping, preventing the server from attempting to contact a MoM that is still down. This will cut down on inter-daemon traffic, especially in larger complexes.

8.4.4 Creation of MoM Home Directory

When you run `systemctl` or the PBS start/stop script on an execution host, PBS creates MoM's home directory if it does not already exist.

8.4.5 Server: Starting, Stopping, Restarting

8.4.5.1 Starting Server Without Failover

On the local host:

```
PBS_EXEC/sbin/pbs_server [options]
```

8.4.5.2 Starting Servers With Failover

You can start the servers in any order. If you want to let running jobs and subjobs continue running, use the `pbs_server` command to start the servers. Starting via the start/stop script or `systemctl` kills running jobs and subjobs. If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it will inform the secondary that the primary is taking over and the secondary can become idle.

- On the primary host, start the primary server:

  ```
  pbs_server
  ```

- You can start the secondary server while it is the active server. On the secondary server host:

  ```
  pbs_server -F -l
  ```

  The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.

If there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

```
qterm -F
pbs_server
```

8.4.5.3 Stopping Server Without Failover

To stop the server and leave running jobs and subjobs running:

```
qterm
```

8.4.5.3.i Stopping Server via Signals

If you send the server a SIGTERM, the server does a quick shutdown, equivalent to receiving a `qterm -t quick`.

See "pbs_server" on page 108 of the PBS Professional Reference Guide and "qterm" on page 233 of the PBS Professional Reference Guide.

8.4.5.4 Stopping Servers With Failover

If you have failover configured, and want to stop the servers but allow running jobs and subjobs to continue running, use the `qterm` command. Both the start/stop script and `systemctl` kill running jobs and subjobs.

- To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

  ```
  qterm -f
  ```

- To stop the primary server and leave the secondary server idle:

  ```
  qterm -i
  ```

- To stop the secondary server only:

  ```
  qterm -F
  ```

8.4.5.5 Restarting Server Without Failover

```
qterm -t quick
PBS_EXEC/sbin/pbs_server
```

8.4.5.6 Restarting Servers with Failover

8.4.5.6.i Stopping Servers

If you have failover configured, and want to stop the servers but allow running jobs and subjobs to continue running, use the `qterm` command. Both the start/stop script and `systemctl` kill running jobs and subjobs.

- To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

  ```
  qterm -f
  ```

- To stop the primary server and leave the secondary server idle:

  ```
  qterm -i
  ```

- To stop the secondary server only:

  ```
  qterm -F
  ```

8.4.5.6.ii Starting Servers

You can start the servers in any order. If you want to let running jobs and subjobs continue running, use the `pbs_server` command to start the servers. Starting via the start/stop script or `systemctl` kills running jobs and subjobs. If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it will inform the secondary that the primary is taking over and the secondary can become idle.

- On the primary host, restart the primary server:
  ```
  pbs_server
  ```
- To restart the secondary server while it is the active server:
  ```
  pbs_server -F -1
  ```
  The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.

8.4.5.6.iii Network Outage

If there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

```
qterm -F
pbs_server
```

8.4.5.7 Restarting Server To Resume Previously-running Jobs

If, when the server was shut down, running jobs and subjobs were killed and requeued, then starting the server with the `-t hot` option puts those jobs back in the *Running* state first. See "pbs_server" on page 108 of the PBS Professional Reference Guide for details and the options to the `pbs_server` command.

8.4.6 Scheduler(s): Starting, Stopping, Restarting

8.4.6.1 Starting Default Scheduler

To start the default scheduler directly, do the following:

PBS_EXEC/sbin/pbs_sched [options]

8.4.6.2 Starting Multisched

To start a multisched, call `pbs_sched` and specify the name and port you already gave it:

pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

For example:

```
pbs_sched -I multisched_1 -S 15050
```

When you start a multisched, you must specify its name.

See "pbs_sched" on page 105 of the PBS Professional Reference Guide for more information and a description of available options.

8.4.6.3 Stopping Scheduler or Multisched

1. Find the PID you want:
   ```
   ps -ef | grep pbs_sched
   ```

For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

2. Stop the scheduler or multisched:

```
kill <scheduler PID>
```

8.4.6.4 Stopping Scheduler(s) via Signals

You can stop a scheduler by sending it SIGTERM or SIGINT. These result in an orderly shutdown of the scheduler.

8.4.6.5 Restarting and Reinitializing Scheduler or Multisched

8.4.6.5.i When to Restart or Reinitialize Scheduler or Multisched

- Restart the scheduler(s) after you change pbs.conf.
- HUP the scheduler(s) if you added any custom resources to the resources: line in <sched_priv directory>/sched_config.

8.4.6.5.ii Restarting Scheduler or Multisched

1. Find the PID you want:

```
ps -ef | grep pbs_sched
```

For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

2. Stop the scheduler or multisched:

```
kill <scheduler PID>
```

3. Start the scheduler or multisched:
   - To start the default scheduler:

     PBS_EXEC/sbin/pbs_sched [options]
   - To start a multisched, call pbs_sched and specify the name and port you already gave it:

     pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

8.4.6.5.iii Reinitializing Scheduler or Multisched

Find the PID you want:

```
ps -ef | grep pbs_sched
```

For the default scheduler, you'll see "pbs_sched", but for multischeds, you'll see "pbs_sched -I <multisched name>".

```
kill -HUP <scheduler PID>
```

8.4.7 MoMs: Starting, Stopping, Restarting

8.4.7.1 Starting MoM

You start the PBS MoM directly via the pbs_mom command. See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

8.4.7.2 Stopping MoM

8.4.7.2.i Stopping MoM via Signals

You can stop MoM using the following signals:

Table 8-3: Signals Handled by MoM

Signal	Effect
SIGTERM	If a MoM is killed with the signal SIGTERM, jobs are killed before MoM exits. Notification of the terminated jobs is not sent to the server until the MoM is restarted. Jobs will still appear to be in the "R" (running) state.
SIGINT	If a MoM is killed with this signal, jobs are not killed before the MoM exits. MoM exits after cleanly closing network connections.
SIGKILL	If a MoM is killed with this signal, jobs are not killed before the MoM exits.

8.4.7.2.ii Recommendation to Offline Vnodes Before Stopping MoM

We recommend that you offline vnodes before stopping the MoM. The server tries to keep continual contact with each MoM. If you offline the vnode before stopping the MoM, the server does not try to stay in contact with the MoM. This reduces network traffic.

8.4.7.3 Restarting and Reinitializing MoM

8.4.7.4 Whether to Restart or Reinitialize MoM

When you change configuration files on Linux, whether the MoM must be restarted or reinitialized depends on which MoM configuration file has been changed.

- If only the Version 1 MoM configuration file was changed, you only need to HUP the MoM.
- If you used the `pbs_mom -s insert` command to add to or change anything in the Version 2 MoM config file, you can HUP the MoM.
- If you used the `pbs_mom -s insert` command to remove anything from the Version 2 MoM config file, you must **restart** the MoM.

8.4.7.5 Restarting MoM

You can restart MoM with the following options:

Table 8-4: MoM Restart Options

Option	Effect on Jobs
`pbs_mom`	Job processes continue to run, but the jobs themselves are requeued.
`pbs_mom -r`	Running processes associated with jobs that were running before MoM was terminated are killed. Running jobs and subjobs are requeued or deleted. Do not use this option after a reboot, because process numbers will be incorrect and processes unrelated to jobs may be killed.
`pbs_mom -p`	Jobs which are running when MoM is terminated remain running. Do not use after reboot.

See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

8.4.7.5.i Preserving Existing Jobs When Restarting MoM

By default, when MoM is started, she allows running processes to continue to run, but tells the server to requeue her jobs. You can direct MoM to preserve running jobs and subjobs and to track them, by using the -p option to the pbs_mom command. If you have not just rebooted, you can preserve existing jobs:

1. Use the ps command to determine MoM's process ID. Note that ps arguments vary among Linux systems, thus "-ef" may need to be replaced by "-aux".

   ```
   ps -ef | grep pbs_mom
   ```

2. Terminate MoM using the kill command, with MoM's PID as an argument. The syntax will vary depending on your system:

   ```
   kill -INT <MoM PID>
   ```

   or

   ```
   kill -s INT <MoM PID>
   ```

3. Restart MoM, allowing running jobs and subjobs to continue running through the restart. If your custom resource query script/program takes longer than the default ten seconds, you can change the alarm timeout via the -a alarm command line start option:

   ```
   PBS_EXEC/sbin/pbs_mom -p [ -a timeout]
   ```

8.4.7.5.ii Caveats for Restarting MoM After a Reboot

Never restart pbs_mom with the -p or the -r option following a reboot of the host system.

When a Linux operating system is first booted, it begins to assign process IDs (PIDs) to processes as they are created. PID 1 is always assigned to the system "init" process. As new processes are created, they are either assigned the next PID in sequence or the first empty PID found, which depends on the operating system implementation. Generally, the session ID of a session is the PID of the top process in the session.

The PBS MoM keeps track of the session IDs of the jobs. If only MoM is restarted on a system, those session IDs/PIDs have not changed and apply to the correct processes.

If the entire system is rebooted, the assignment of PIDs by the system will start over. Therefore the PID which MoM thinks belongs to an earlier job will now belong to a different later process. If you restart MoM with -p, she will believe the jobs are still valid jobs and the PIDs belong to those jobs. When she kills the processes she believes to belong to one of her earlier jobs, she will now be killing the wrong processes, those created much later but with the same PID as she recorded for that earlier job.

8.4.7.5.iii Killing Existing Jobs When Restarting MoM

If you wish to kill all existing processes, use the -r option to pbs_mom.

To kill existing jobs, start MoM with the command line:

```
PBS_EXEC/sbin/pbs_mom -r
```

8.4.7.5.iv Starting MoM on the HPE MC990X, HPE Superdome Flex, or HPE 8600

For a cpusetted MC990X, Superdome Flex, or 8600, start MoM using the PBS start/stop script or systemd.

8.4.7.5.v Using Existing CPU and Memory for cpusets

By default, MoM removes existing cpusets when she starts. You can specify that MoM is to use existing CPU and memory allocations for cpusets by using the -p option to the pbs_mom command. This option also preserves running jobs and subjobs. See "Options to pbs_mom" on page 72 of the PBS Professional Reference Guide.

Vnode definition files are not created when the pbs_mom command is used; use it only when you know that they are already up to date.

8.4.7.5.vi Effect of Stopping Sister MoM on Multihost Jobs

Stopping a sister MoM for a multi-vnode job may cause the job to be requeued if the primary MoM loses contact with the sister MoM.

8.4.7.6 Reinitializing MoM

1. Use the `ps` command to determine MoM's process ID. Note that `ps` arguments vary among Linux systems, thus "`-ef`" may need to be replaced by "`-aux`".

   ```
   ps -ef | grep pbs_mom
   ```

2. HUP MoM using the `kill` command, with MoM's PID as an argument:

   ```
   kill -HUP <MoM PID>
   ```

See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

8.4.8 Comms: Starting, Stopping, Restarting

8.4.8.1 Starting Communication Daemon

To start the communication daemon directly, do the following on the local host:

```
PBS_EXEC/sbin/pbs_comm [-N] [ -r <other routers>] [-t <number of threads>]
```

See "pbs_comm" on page 57 of the PBS Professional Reference Guide.

8.4.8.2 Stopping Communication Daemon via Signals

You can stop the communication daemon using a SIGTERM.

8.5 Impact of Stop-Restart on Running Linux Jobs

8.5.1 Whether to Use Script, Command, or Signal for Shutdown and Restart

Use the `qterm` command to shut the server down when running jobs and subjobs must be checkpointed before shutdown, allowed to run to completion before shutdown, or preserved through shutdown and restart. To preserve running jobs and subjobs, stop MoM using KILL -INT and use the `pbs_mom -p` command when restarting MoM.

When you use the PBS start/stop script or `systemd` to stop PBS, MoM kills her jobs and exits. When you use it to restart MoM, jobs are requeued.

8.5.2 Scenarios for Stopping Then Restarting Daemons

Choose one of the following recommended sequences, based on the desired impact on jobs, to stop and restart PBS:

- To allow running jobs and subjobs to continue to run:

  Shutdown:

  ```
  qterm -t quick -m -s
  <path to start/stop script>/pbs stop
  ```
  (on communication-only host)

Restart:

pbs_server -t warm

pbs_mom -p

pbs_sched

pbs_comm (on server host)

<path to start/stop script>/pbs start (on communication-only host)

- To checkpoint and requeue checkpointable jobs, requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs that were previously running:

  Shutdown:

  qterm -t immediate -m -s

  <path to start/stop script>/pbs stop (on communication-only host)

  Restart:

  pbs_mom

  pbs_server -t hot

  pbs_sched

  pbs_comm (on server host)

  <path to start/stop script>/pbs start (on communication-only host)

- To checkpoint and requeue checkpointable jobs, requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs without taking prior state into account:

  Shutdown:

  qterm -t immediate -m -s

  <path to start/stop script>/pbs stop (on communication-only host)

  Restart:

  pbs_mom

  pbs_server -t warm

  pbs_sched

  pbs_comm (on server host)

  <path to start/stop script>/pbs start (on communication-only host)

9 Starting & Stopping MoM on Windows

9.1 Automatic Start on Bootup

On Windows, the PBS MoM daemons are registered as system services, and are automatically started and stopped when the system boots and shuts down.

- The auto-startup of MoM is controlled by the PBS `pbs.conf` file and the *Services* dialog. You invoke this via *Settings->Control Panel->Administrative Tools->Services*. Make sure that in `pbs.conf` your setting for PBS_START_MOM is correct. If this is set to *0*, the service will fail to start up with the message, "incorrect environment".
- On Windows, sometimes MoM may fail to start automatically after the boot. We recommend that you change the startup mode from " *[Startup type: Automatic]*" to "*[Startup type: Automatic (Delayed Start)]*", which means "shortly after boot".

  At the command prompt:

  ```
  sc config <service name> start= delayed-auto
  ```

9.2 When to Restart PBS MoMs

Restart MoM:

- If you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS
- After creating a Version 2 configuration file
- After changing the name of the PBS service account
- After changing the PBS service account to a non-domain administrator account
- After making changes to the `%WINDIR%\system32\dirvers\etc\hosts` file

9.3 Starting, Stopping, and Restarting PBS

9.3.1 Required Privilege

To stop or start MoM, you must have Administrator privilege.

9.3.2 Recommendation for Service Start Order

We recommend starting the communication daemon before starting the MoMs, but you can also start it after the MoMs and before the server.

We recommend starting MoMs before starting the server. This way, MoM will be ready to respond to the server's "are you there?" ping, preventing the server from attempting to contact a MoM that is still down. This will cut down on inter-daemon traffic, especially in larger complexes.

9.3.3 Creation of MoM Home Directory

When you run `systemctl` or the PBS start/stop script on an execution host, PBS creates MoM's home directory if it does not already exist.

9.3.4 Windows-specific Service Options

The Windows MoM has the following Windows-only option:

-N

The service runs in standalone mode, not as a Windows service.

9.3.5 Configuring Startup Options to MoM

You can use the startup options to the `pbs_mom` command when starting the MoM.

The procedure to specify startup options to the MoM is as follows:

1. Go to the *Services* menu.
2. Select "*PBS_MOM*". The MoM service dialog box comes up.
3. Enter the desired options in the "*Start parameters*" entry line. For example, to specify an alternate MoM configuration file, you might specify the following input:

   On Windows systems:

   ```
   -c "\Program Files (x86)\PBS\home\mom_priv\config2"
   ```

4. Click on "*Start*" to start the MoM service.

9.3.5.1 Saving Startup Options

You can save your options for the future. If PBS_EXEC and PBS_HOME are set:

```
sc config pbs_mom binpath="%PBS_EXEC%\sbin\pbs_mom.exe -c ""%PBS_HOME%\mom_priv\config2"""
```

If you don't save your startup options, the Windows services dialog does not remember the "*Start parameters*" value when you close the dialog. You will have to specify the "*Start parameters*" value for each future restart.

9.3.6 MoMs: Starting, Stopping, Restarting

On Windows, you must restart MoM when any MoM configuration file has been changed.

9.3.6.1 Starting MoM as a Service

On the local host:

```
net start pbs_mom
```

9.3.6.2 Starting MoM in Standalone Mode

On the local host:

```
pbs_mom -N <options>
```

9.3.6.3 Stopping MoMs

On the local host:

```
net stop pbs_mom
```

9.3.6.3.i Effect of Stopping Sister MoM on Multihost Jobs

Stopping a sister MoM for a multi-vnode job may cause the job to be requeued if the primary MoM loses contact with the sister MoM.

9.3.6.3.ii Recommendation: Offline Vnodes Before Stopping MoM

We recommend that you offline vnodes before stopping the MoM. The server tries to keep continual contact with each MoM. If you offline the vnode before stopping the MoM, the server does not try to stay in contact with the MoM. This reduces network traffic.

9.3.6.4 Restarting MoMs

You can restart MoM with the following options:

Table 9-1: MoM Restart Options

Option	Effect on Jobs
pbs_mom	Job processes will continue to run, but the jobs themselves are requeued.
pbs_mom -p	Jobs which were running when MoM terminated remain running.
pbs_mom -r	Processes associated with the job are killed. Running jobs and subjobs are returned to the server to be requeued or deleted. This option should not be used if the system has just been rebooted as the process numbers will be incorrect and a process not related to the job would be killed.

See section 9.3.5, "Configuring Startup Options to MoM", on page 172.

On the local host:

```
Admin> net stop pbs_mom
Admin> net start pbs_mom
```

9.3.6.4.i Preserving Existing Jobs When Restarting MoM

By default, when MoM is started, she allows running processes to continue to run, but tells the server to requeue her jobs. You can direct MoM to preserve running jobs and subjobs and to track them, by using the -p option to the pbs_mom command.

9.3.6.4.ii Caveats for Preserving Existing Jobs When Restarting MoM

- If you restart a sister MoM for a multi-vnode job, the job may be killed because the primary MoM may lose contact with the sister MoM and requeue the job.
- Never use the -p option to pbs_mom after a reboot.

9.4 Stopping PBS Using the qterm Command

The qterm command is used to shut down, selectively or inclusively, the PBS server, scheduler(s), and MoMs. The qterm command does not shut down pbs_comm. If you have a failover server configured, then when the primary server is shut down, the secondary server becomes active unless you shut it down as well. The qterm command can be run at any PBS host.

You can specify how running jobs and subjobs are treated during shutdown by specifying the type of shutdown. The type of shutdown performed by the qterm command defaults to "-t quick", which preserves running jobs and subjobs:

```
qterm -t quick
```

The following command shuts down the primary server, the scheduler(s), and all MoMs in the complex. If configured, the secondary server becomes active. Running jobs and subjobs continue to run:

```
qterm -s -m
```

The following command shuts down the primary server, the secondary server, the scheduler(s), and all MoMs in the complex. Running jobs and subjobs continue to run:

```
qterm -s -m -f
```

See "qterm" on page 233 of the PBS Professional Reference Guide.

9.4.0.0.i qterm Caveats

- The qterm command does not stop the pbs_comm service. You must stop pbs_comm using the start/stop script or the kill command.
- Shutting PBS down using the qterm command does not perform any of the other cleanup operations that are performed by the net stop command.

9.5 Impact of Stop-Restart on Running Windows Jobs

The methods you can use to shut down PBS, and which daemons are shut down, will affect running jobs and subjobs differently. You can leave jobs and subjobs running during shutdown.

The impact of a shutdown (and subsequent restart) on running jobs and subjobs depends on whether you use net stop or the qterm command to shut down PBS, and how pbs_mom is restarted.

You can use the qterm command to shut the server down.

Jobs are not killed when pbs_mom is stopped via net stop; whether they are killed depends on how MoM is restarted.

Use the qterm command to shut the server down when running jobs and subjobs must be checkpointed before shutdown, allowed to run to completion before shutdown, or preserved through shutdown and restart.

To preserve running jobs and subjobs, use the -p option to the pbs_mom command when restarting MoM.

9.5.1 Scenarios for Stopping Then Restarting Services

Choose one of the following recommended sequences, based on the desired impact on jobs, to stop and restart PBS.

The start/stop script is located in /etc/init.d/pbs or /etc/rc.d/init.d/pbs.

- To allow running jobs and subjobs to continue to run:

Shutdown:

qterm -t quick -m -s

<path to start/stop script>/pbs stop (on communication-only host)

Restart:

PBS_EXEC/sbin/pbs_server -t warm

pbs_mom -p

PBS_EXEC/sbin/pbs_sched

PBS_EXEC/sbin/pbs_comm (on server host)

<path to start/stop script>/pbs start (on communication-only host)

net start pbs_mom (with **-p** startup option set)

- To checkpoint and requeue checkpointable jobs, requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs that were previously running:

  qterm -t immediate -m -s

  <path to start/stop script>/pbs stop (on communication-only host)

  Restart:

  net start pbs_mom

  PBS_EXEC/sbin/pbs_server -t hot

  PBS_EXEC/sbin/pbs_sched

  PBS_EXEC/sbin/pbs_comm (on server host)

  <path to start/stop script>/pbs start (on communication-only host)

- To checkpoint and requeue checkpointable jobs, requeue rerunnable jobs, kill any non-rerunnable jobs, then restart and run jobs without taking prior state into account:

  Shutdown:

  qterm -t immediate -m -s

  <path to start/stop script>/pbs stop (on communication-only host)

  Restart:

  net start pbs_mom

  PBS_EXEC/sbin/pbs_server -t warm

  PBS_EXEC/sbin/pbs_sched

  PBS_EXEC/sbin/pbs_comm (on server host)

  <path to start/stop script>/pbs start (on communication-only host)

Index

A

account
- installation IG-13
- PBS service IG-14

Active Directory IG-13
Admin IG-13
administrators IG-13
authorization IG-12

B

backup directory
- overlay upgrade IG-72, IG-73, IG-83, IG-85, IG-96
- Windows upgrade IG-111, IG-126, IG-127

C

capmc IG-139
CentOS IG-23
CLE 6 and 7 IG-139
client commands IG-4
commands IG-4

D

delegation IG-13
DIS IG-59
DNS IG-38
Domain Admin Account IG-13
Domain Admins IG-13
Domain User Account IG-13
Domain Users IG-13
domains
- mixed IG-17

E

empty queue, node configurations
- migration under Linux IG-100, IG-115, IG-116, IG-130

Enterprise Admins IG-13

F

failover
- migration IG-73, IG-85, IG-97, IG-112, IG-128

file
- .rhosts IG-12
- .shosts IG-12
- hosts.equiv IG-15, IG-39
- pbs.conf IG-43
- services IG-59

G

gethostbyaddr IG-58

H

headnode IG-21

I

IETF IG-9, IG-58
IMPS IG-139
installation
- Windows MoMs IG-37

installation account IG-13

M

migration upgrade IG-65
- Linux IG-93
- Windows IG-109, IG-125

mixed domains IG-17
MoM IG-4
moving jobs
- migration upgrade under Linux IG-107, IG-123

N

network
- ports IG-58
- services IG-58

NTFS IG-41

O

output files IG-12
overlay upgrade IG-65
- backup directory IG-72, IG-73, IG-83, IG-85, IG-96
- Linux IG-70

P

PBS service account IG-14
PBS_BATCH_SERVICE_PORT IG-59
PBS_BATCH_SERVICE_PORT_DIS IG-59
PBS_DATA_SERVICE_PORT IG-59
PBS_EXEC IG-21, IG-43
PBS_EXEC/pbs_sched_config

overlay upgrade IG-76, IG-88, IG-101, IG-117, IG-131
PBS_HOME IG-21, IG-43
PBS_LEAF_NAME IG-62
PBS_MAIL_HOST_NAME IG-62
PBS_MANAGER_SERVICE_PORT IG-59
pbs_mom IG-4
starting during overlay IG-78
PBS_MOM_HOST_NAME IG-62
PBS_MOM_SERVICE_PORT IG-59
PBS_OUTPUT_HOST_NAME IG-62
PBS_PRIMARY IG-62
pbs_probe IG-63
pbs_sched IG-3, IG-4
PBS_SCHEDULER_SERVICE_PORT IG-59
PBS_SECONDARY IG-62
PBS_SERVER IG-62
pbs_server IG-3, IG-4
PBS_SERVER_HOST_NAME IG-62
PBS_START_COMM IG-159
PBS_START_MOM IG-159
PBS_START_SCHED IG-159
PBS_START_SERVER IG-159
primary server IG-62

Q

qalter IG-16
qsub IG-16

R

Red Hat Enterprise Linux IG-23
Release Notes
upgrade recommendations IG-65, IG-93

S

scheduler IG-4
Schema Admins IG-14
scp IG-12
secondary server IG-62
secure copy IG-12
server IG-4
primary IG-62
secondary IG-62
service account
PBS IG-14
ssh IG-12
starting
MoM IG-166
SuSE IG-23

T

tar file
overlay upgrade IG-73, IG-85

U

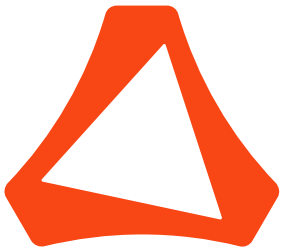
upgrade
migration IG-65
migration under Linux IG-93
migration under Windows IG-109, IG-125
overlay IG-65
upgrading
Linux IG-70
Windows IG-109, IG-125

W

Windows IG-15, IG-17, IG-23

X

X forwarding IG-63
xauth IG-63



Altair PBS Professional 2021.1

Administrator's Guide

You are reading the Altair PBS Professional 2021.1

Administrator's Guide (AG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation xi

1 New Features 1

1.1 New Features in PBS 2020.1 . . . 1
1.2 Changes in Previous Releases . . . 1
1.3 Commercial-only Features . . . 15
1.4 Backward Compatibility . . . 16

2 Configuring the Server and Queues 17

2.1 The Server . . . 17
2.2 How PBS Uses Mail . . . 19
2.3 Queues . . . 21

3 Configuring MoMs and Vnodes 33

3.1 About MoMs . . . 33
3.2 About Vnodes: Virtual Nodes . . . 37
3.3 Creating Vnodes . . . 38
3.4 Configuring Vnodes . . . 41
3.5 Deleting Vnodes . . . 49

4 Scheduling 53

4.1 Chapter Contents . . . 53
4.2 Scheduling Each Partition Separately . . . 55
4.3 Scheduling Policy Basics . . . 62
4.4 Choosing a Policy . . . 79
4.5 About Schedulers . . . 89
4.6 Using Queues in Scheduling . . . 98
4.7 Scheduling Restrictions and Caveats . . . 99
4.8 Errors and Logging . . . 100
4.9 Scheduling Tools . . . 100

5 Using PBS Resources 229

5.1 Chapter Contents . . . 229
5.2 Introduction to PBS Resources . . . 230
5.3 Glossary . . . 230
5.4 Categories of Resources . . . 233
5.5 Resource Types . . . 237
5.6 Resource Formats . . . 237
5.7 Setting Values for Resources . . . 239
5.8 Overview of Ways Resources Are Used . . . 243
5.9 Resources Allocated to Jobs and Reservations . . . 244
5.10 Using Resources to Track and Control Allocation . . . 252
5.11 Using Resources for Topology and Job Placement . . . 253
5.12 Using Resources to Prioritize Jobs . . . 254
5.13 Using Resources to Restrict Server or Queue Access . . . 254
5.14 Custom Resources . . . 255
5.15 Managing Resource Usage . . . 287
5.16 Where Resource Information Is Kept . . . 309
5.17 Viewing Resource Information . . . 311
5.18 Resource Recommendations and Caveats . . . 313

6 Managing Power Usage 315

6.1 Monitoring and Controlling Job Power Usage . . . 315
6.2 Managing Node Power on Cray XC . . . 320
6.3 Power Management Attributes, Resources, Etc. . . . 322
6.4 Caveats and Restrictions for Power Management . . . 325

7 Provisioning 327

7.1 Introduction . . . 327
7.2 Definitions . . . 327
7.3 How Provisioning Can Be Used . . . 327
7.4 How Provisioning Works . . . 328
7.5 Configuring Provisioning . . . 335
7.6 Viewing Provisioning Information . . . 340
7.7 Requirements and Restrictions . . . 343
7.8 Defaults and Backward Compatibility . . . 345
7.9 Example Scripts . . . 345
7.10 Advice and Caveats . . . 353
7.11 Errors and Logging . . . 355

8 Security 359

8.1 Configurable Features . . . 359
8.2 User Roles and Required Privilege . . . 359
8.3 Using Access Control Lists . . . 362
8.4 Authentication for Daemons & Users . . . 378
8.5 Encrypting PBS Communication . . . 387
8.6 Restricting Execution Host Access . . . 391
8.7 Access to Schedulers . . . 392
8.8 Changing the PBS Service Account Password . . . 392
8.9 Paths and Environment Variables . . . 393
8.10 File and Directory Permissions . . . 393
8.11 Root-owned Jobs . . . 394
8.12 Passwords . . . 394
8.13 Windows Firewall . . . 395
8.14 Logging Security Events . . . 395
8.15 Securing Containers . . . 397

9 Making Your Site More Robust 399

9.1 Robustness . . . 399
9.2 Failover . . . 399
9.3 Checkpoint and Restart . . . 418
9.4 Reservation Fault Tolerance . . . 432
9.5 Vnode Fault Tolerance for Job Start and Run . . . 434
9.6 Preventing Communication and Timing Problems . . . 441
9.7 Preventing File System Problems . . . 448
9.8 OOM Killer Protection . . . 449

10 Using MPI with PBS 451

10.1 Integration with MPI . . . 451
10.2 Prerequisites . . . 451
10.3 Types of Integration . . . 451
10.4 Transparency to the User . . . 453
10.5 Integrating Intel MPI 4.0.3 On Linux Using Environment Variables . . . 453
10.6 Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script . . . 454
10.7 Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script . . . 454
10.8 Integration Using the TM Interface . . . 454
10.9 Integration on the Fly using the `pbs_tmrsh` Command . . . 454
10.10 Integration by Wrapping . . . 455
10.11 Wrapping an MPI Using the `pbsrun_wrap` Script . . . 457
10.12 Unwrapping MPIs Using the `pbsrun_unwrap` Script . . . 460
10.13 Integration By Hand . . . 460
10.14 How Processes are Started Using MPI and PBS . . . 466
10.15 Limit Enforcement with MPI . . . 468
10.16 Restrictions and Caveats for MPI Integration . . . 469

11 Configuring PBS for Cray 471
11.1 Support for Shasta . . . 471
11.2 Configuring PBS for Cray XC Series . . . 472
11.3 Introduction to PBS on Cray XC . . . 473
11.4 Relationship of PBS Vnodes to Cray XC Nodes . . . 473
11.5 Requirements for Cray XC . . . 474
11.6 Restrictions for Cray XC . . . 474
11.7 Resources, Parameters, etc. for Cray XC . . . 474
11.8 Automatic Configuration for Cray XC . . . 475
11.9 Recommended Manual Configuration for Cray XC . . . 477
11.10 Improving Server/MoM Inventory Performance for Cray XC . . . 481
11.11 Synchronizing PBS with ALPS Inventory on Cray XC . . . 483
11.12 Support for Xeon Phi on Cray XC . . . 483
11.13 Using Hyperthreading on Cray XC . . . 490
11.14 Viewing Cray XC Information . . . 490
11.15 Resource Restrictions and Deprecations for Cray XC . . . 492
11.16 Caveats and Advice for Cray XC . . . 492
11.17 Errors and Logging on Cray XC . . . 496

12 Support for HPE 499
12.1 Briefly, How PBS Manages Cpusets . . . 499
12.2 Cpusets and Vnodes . . . 499
12.3 Requirements for Managing Cpusets . . . 499
12.4 Where to Use Cpusets . . . 499
12.5 Settings for sharing Attribute . . . 499
12.6 Comprehensive System Accounting . . . 500

13 Support for NEC SX-Aurora TSUBASA 501
13.1 Vnodes for NEC SX-Aurora TSUBASA . . . 501
13.2 Terminology . . . 501
13.3 Resources for SX-Aurora TSUBASA . . . 502
13.4 Configuring PBS for NEC SX-Aurora TSUBASA . . . 503
13.5 Debugging on NEC SX-Aurora TSUBASA . . . 503
13.6 Suspending and Resuming Jobs . . . 504
13.7 Job Accounting on NEC SX-Aurora TSUBASA . . . 504

14 Managing Jobs 505

14.1 Routing Jobs . . . 505
14.2 Limiting Number of Jobs Considered in Scheduling Cycle . . . 505
14.3 Allocating Resources to Jobs . . . 505
14.4 Grouping Jobs By Project . . . 507
14.5 Job Prologue and Epilogue . . . 508
14.6 Linux Shell Invocation . . . 513
14.7 When Job Attributes are Set . . . 514
14.8 Job Termination . . . 516
14.9 Job Exit Status Codes . . . 519
14.10 Rerunning or Requeueing a Job . . . 521
14.11 Job IDs . . . 522
14.12 Where to Find Job Information . . . 522
14.13 Job Directories . . . 523
14.14 The Job Lifecycle . . . 527
14.15 Managing Job History . . . 530
14.16 Environment Variables . . . 534
14.17 Adjusting Job Running Time . . . 534
14.18 Managing Number of Run Attempts . . . 534
14.19 Managing Amount of Memory for Job Scripts . . . 534
14.20 Allowing Interactive Jobs on Windows . . . 534
14.21 Releasing Unneeded Vnodes from Jobs . . . 537
14.22 Tolerating Vnode Faults . . . 538
14.23 Managing Job Array Behavior . . . 538
14.24 Recommendations . . . 538

15 Administration 539

15.1 The PBS Configuration File . . . 539
15.2 Environment Variables . . . 545
15.3 Event Logging . . . 546
15.4 Managing Machines . . . 553
15.5 Managing the Data Service . . . 557
15.6 Setting File Transfer Mechanism . . . 559
15.7 Some Performance Tips . . . 567
15.8 Temporary File Location for PBS Components . . . 567
15.9 Administration Caveats . . . 569
15.10 Support for Globus . . . 569
15.11 Support for Hyperthreading . . . 569
15.12 How To... . . . 570

16 Configuring and Using PBS with Cgroups 571

16.1 Chapter Contents . . . 571
16.2 Introduction to Cgroups . . . 571
16.3 Why Use Cgroups? . . . 572
16.4 How PBS Uses Cgroups . . . 573
16.5 Configuring Cgroups . . . 576
16.6 Configuring MPI for Cgroups . . . 604
16.7 Managing Jobs with Cgroups . . . 606
16.8 Caveats and Errors . . . 607

17 Configuring PBS for SELinux 609
17.1 Overview of PBS Support for MLS-compliant SELinux . . . 609
17.2 Terminology . . . 609
17.3 How Support for SELinux Works . . . 609
17.4 Enforcement of Permissions . . . 610
17.5 Special Attributes and Directories . . . 610
17.6 Prerequisites . . . 611
17.7 Caveats and Restrictions . . . 611
17.8 Installing PBS For Use With SELinux . . . 611
17.9 Configuring PBS for SELinux . . . 613
17.10 Managing an SELinux System . . . 614

18 Configuring PBS for Containers 615
18.1 Introduction . . . 615
18.2 The PBS Container Hook . . . 620
18.3 Prerequisites . . . 620
18.4 Configuring PBS for Containers . . . 621
18.5 Caveats and Restrictions . . . 626
18.6 Errors and Logging . . . 626

19 Accounting 627
19.1 The Accounting Log File . . . 627
19.2 Viewing Accounting Information . . . 628
19.3 Format of Accounting Log Messages . . . 628
19.4 Types of Accounting Log Records . . . 630
19.5 Timeline for Accounting Messages . . . 643
19.6 Resource Accounting . . . 649
19.7 Options, Attributes, and Parameters Affecting Accounting . . . 654
19.8 Accounting Caveats and Advice . . . 656

20 Mixed Linux-Windows Operation 659
20.1 Introduction to Mixed Linux-Windows Operation . . . 659
20.2 Configuration . . . 659
20.3 Troubleshooting Mixed Linux-Windows Complex . . . 661

21 Problem Solving 663
21.1 Debugging Tools . . . 663
21.2 Security and Permissions Problems . . . 664
21.3 Troubleshooting Jobs . . . 665
21.4 Troubleshooting Daemons . . . 669
21.5 Troubleshooting Vnodes . . . 671
21.6 Troubleshooting Client Commands . . . 671
21.7 Troubleshooting PBS Licenses . . . 673
21.8 Crash Recovery . . . 673
21.9 Other Troubleshooting . . . 674
21.10 Getting Help . . . 675

Index 677

1 New Features

This chapter briefly lists new features by release, with the most recent listed first.

For deprecations, please see the *Release Notes*.

The *Release Notes* included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the *Release Notes*, as they may contain information that was not available when this book was written.

1.1 New Features in PBS 2020.1

New Options for Altering Reservations

You can change a reservation's select specification, and the administrator can override the scheduler to change the start time, end time, and duration of a reservation. See “pbs_ralter” on page 86 of the PBS Professional Reference Guide.

Integration with NEC SX-Aurora TSUBASA

PBS provides topologically aware job resource requests and scheduling. For configuration information, see Chapter 13, "Support for NEC SX-Aurora TSUBASA", on page 501. For job submission instructions, see "Submitting Jobs to NEC SX-Aurora TSUBASA", on page 201 of the PBS Professional User’s Guide.

Integration with Container Access Control

PBS allows you to whitelist container registries, and supports logging into private container registries. You can also manage mount paths for greater security. See Chapter 18, "Configuring PBS for Containers", on page 615.

Managing Shared Job Directory Behavior

You can prevent sister MoMs from prematurely removing shared job directories and files during node release. See section 14.13.1, “Staging and Execution Directories for Job”, on page 523.

Limiting Number of Subjobs Running at One Time

Job submitters can limit the number of simultaneously running subjobs for an array job; see "Limiting Number of Simultaneously Running Subjobs", on page 152 of the PBS Professional User’s Guide.

1.2 Changes in Previous Releases

New Cgroups Hook (2020.1)

PBS has an expanded cgroups hook with many new capabilities. This hook replaces the cpuset MoM. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

Cloud Bursting Feature (2020.1)

PBS now has its own cloud bursting feature. See the PBS Cloud Guide.

Budget Allocation Feature (2020.1)

PBS now has its own budget allocation feature. See the PBS Budget Guide.

Workload Simulation Feature (2020.1)

PBS now has its own workload simulation feature. See the PBS Simulate Guide.

Timeout for Dynamic Server Resource Scripts (2020.1)

By default, PBS allows a dynamic server resource script 30 seconds to run. You can configure the timeout; see section 5.14.3.1, “Dynamic Server-level Resources”, on page 267.

Specifying Hosts or Vnodes to Keep when Releasing Unneeded Vnodes (2020.1)

You can specify how many hosts or which vnodes to keep when releasing unneeded vnodes. See “pbs_release_nodes” on page 92 of the PBS Professional Reference Guide.

Using Undo Live Recorder to Capture Daemon Execution Recordings (2020.1)

You can use Undo Live Recorder to capture execution history for analysis by Altair support. See section 21.1.4, “Sending Daemon Execution Recordings to Altair”, on page 663.

PBS Reconfirms Degraded Reservations (2020.1)

If reservation vnodes become unavailable, PBS looks for replacements. See section 9.4.2, “Finding Replacement Vnodes for Degraded and In-conflict Reservations”, on page 433.

New Default for TPP Message Processing (2020.1)

The default for the number of TPP messages the server can process per thread iteration is now 64. See “rpp_max_pkt_check” on page 295 of the PBS Professional Reference Guide.

Automatic Deletion of Idle Reservations (2020.1)

PBS can automatically delete idle reservations. See "Introduction to Creating and Using Advance and Standing Reservations", on page 136 of the PBS Professional User’s Guide.

Flexible Job-specific Reservations (2020.1)

You can create flexible job-specific reservations for queued or running jobs. See "Job-specific Reservations", on page 140 of the PBS Professional User’s Guide.

Altering Reservation Duration, Authorized Groups, Authorized Users (2020.1)

You can alter the duration of a reservation; see “pbs_ralter” on page 86 of the PBS Professional Reference Guide.

Accounting Record for Job Suspend and Resume (2020.1)

PBS records job suspension and resumption in the accounting log. See Chapter 19, "Accounting", on page 627.

Managing Number of Scheduler Threads (2020.1)

You can set the maximum number of threads used by each scheduler. See section 4.5.7.3, “Setting Number of Scheduler Threads”, on page 98.

Configurable Authentication Methods (2020.1)

You can use various authentication methods with PBS; see section 8.4, “Authentication for Daemons & Users”, on page 378.

Using TLS for Encryption (2020.1)

You can use TLS encryption with PBS. See section 8.5, “Encrypting PBS Communication”, on page 387.

Mixed Operation on Linux and Windows (2020.1)

You can use both Linux and Windows execution and client hosts in the same PBS complex. See Chapter 20, "Mixed Linux-Windows Operation", on page 659.

Run Jobs on First Available Resources (Beta 2020.1)

You can submit a set of jobs that would all accomplish the same thing, but that specify different resources. PBS runs only the first that can run. See "Running Your Job on First Available Resources (2020.1 Beta)", on page 108 of the PBS Professional User’s Guide.

New pbs_login Command (2020.1)

PBS includes a new command for user authentication called `pbs_login`. See "pbs_login" on page 69 in the PBS Professional Installation & Upgrade Guide.

One way to sort jobs for preemption (2020.1)

Jobs are chosen for preemption only by which have been running the shortest time. See section 4.9.33, "Using Preemption", on page 180.

New Threading Option for Schedulers (2020.1)

You can specify the number of threads each scheduler runs. See "pbs_sched" on page 105 of the PBS Professional Reference Guide.

License Server for Node and Socket Licenses (2020.1)

PBS uses a license server to license hosts in the complex. See the *PBS Works Licensing Guide*.

Specifying Additional Arguments for Container Engines (2020.1)

Job submitters can specify additional container engine arguments such as secondary groups and shared memory; see "Specifying Additional Arguments to Container Engine", on page 132 of the PBS Professional User's Guide.

Preemption via Deletion (19.4)

You can use deletion to preempt jobs. See section 4.9.33, "Using Preemption", on page 180.

New Scheduler Attributes for Preemption (19.4)

The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings are now scheduler attributes with the same names and formats. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.

All Groups Included in Group ACLs (19.4)

All of a user's groups are included in the list of groups in group ACLs. See section 8.3.4.5, "Contents of Group ACLs", on page 364.

Changes to qstat Job Output (19.4)

Wide output lines can be displayed for any default or alternate `qstat` job output formats, and when output size is too large for a field, the last character is replaced with an asterisk. See "qstat" on page 197 of the PBS Professional Reference Guide.

Subjob Run Count Tracking (19.4)

PBS tracks the run_count attribute for subjobs, and holds job arrays whose subjobs hit the run count limit. See section 14.18, "Managing Number of Run Attempts", on page 534.

Faster Read of Custom Job Resources by Execution Hooks (19.4)

You can specify which custom resources are cached at MoMs so that execution hooks can read them faster. See section 5.14.2.3.x, "Allowing Execution Hooks to Read Custom Job Resources Faster", on page 261.

Applications Running in Containers Can Use Ports (19.4)

PBS can provide ports for applications running in containers. See "Configuring PBS for Containers".

Support for Singularity Containers (19.4)

You can run PBS jobs in Singularity containers. See "Configuring PBS for Containers".

New Post-suspend and Pre-resume Hooks (19.4)

PBS has two new hook events for just after suspending a job and just before resuming it. See "Event Types" on page 87 in the PBS Professional Hooks Guide.

Scheduler Logging Consistent with Other Daemons (19.4)

Schedulers use the same logging scheme as other daemons. See "Event Logging" on page 546 of the PBS Professional Reference Guide.

Option to Capture Only PBS Configuration Information with pbs_snapshot (19.4)

You can use the new `pbs_snapshot --basic` option to capture just PBS configuration information. See "pbs_snapshot" on page 112 of the PBS Professional Reference Guide.

Support for Cray Shasta Systems (19.4)

PBS is supported on Cray's Shasta systems. See section 11.1, "Support for Shasta", on page 471.

Expanded and New Accounting Records (19.4)

PBS writes a new "a" accounting record when a job is altered, and the "Q" record is expanded to include more information. See section 19.4, "Types of Accounting Log Records", on page 630.

IP Address Can Be Used for Vnode Name (19.4)

You can use the IP address as the vnode name. See "Vnode Name" on page 358 of the PBS Professional Reference Guide.

Developer Libraries and Headers in Developer Package (19.4)

The libraries and headers needed for development but not for running PBS have been moved to a developer package. See "Developer Headers and Libraries" on page 19 in the PBS Professional Programmer's Guide.

PBS Uses Python 3 (19.4)

As of 19.4.1, PBS uses Python 3.

New Basic Option to pbs_snapshot (19.4)

The `pbs_snapshot` command has a new `--basic` option. See "pbs_snapshot" on page 112 of the PBS Professional Reference Guide.

New Maintenance Reservation (19.4)

PBS provides a new type of reservations for performing maintenance. See section 4.9.37, "Reservations", on page 197.

Windows MoMs and Clients Run with Linux Server, Schedulers, Comms (19.4)

As of 19.4.1, PBS complexes that run Windows MoMs and Clients run with Linux server, schedulers, and comms.

Undo Live Recorder Debugger (19.4)

Undo's Live Recorder integration enhances our ability to pinpoint root causes of problematic behavior. (This capability is used under the direction of Altair support staff to speed troubleshooting.)

PBS Defaults to 24/7 Primetime (19.2)

You can use PBS without configuring primetime and/or holidays. See section 4.9.34, "Using Primetime and Holidays", on page 191

Microsecond Logging (19.2)

You can choose to have daemons log with microsecond resolution. See section 15.3.4.1, "Event Logfile Format", on page 549.

Limiting ncpus Count to Cores (19.2)

You can opt not to include hyperthreads when calculating the value for ncpus that MoM reports to the server. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

Change in Enabling Power Provisioning (19.2)

You enable power provisioning by enabling the PBS_power hook. See Chapter 6, "Managing Power Usage", on page 315.

Settable Maximum Job ID (19.2)

You can set the maximum value for job IDs, job array IDs, and reservation IDs, using the max_job_sequence_id server attribute.

Job Vnode Fault Tolerance (19.2)

You can allocate extra vnodes to jobs to allow jobs to successfully start and run despite vnode failures. See section 9.5, "Vnode Fault Tolerance for Job Start and Run", on page 434.

Hooks Support Reliable Job Startup and Run (19.2)

Hooks have been enhanced to allow you to provide jobs with extra vnodes in case of vnode failure. See section 9.5, "Vnode Fault Tolerance for Job Start and Run", on page 434.

New Reservation End Hook (19.2)

You can create hooks for the end of a reservation. See "resv_end: Event when Reservation Ends" on page 90 in the PBS Professional Hooks Guide.

Enhancements to pbs_snapshot (19.2)

You can run `pbs_snapshot` without root privilege, and the command captures JSON output. See "pbs_snapshot" on page 112 of the PBS Professional Reference Guide.

Tunable Job Release Wait Time for Cray (19.2)

You can set the amount of time that PBS waits between sending release requests to ALPS. See section 11.9.7, "Set ALPS Reservation Release Timeout on Cray XC", on page 480.

Managing Power Usage on Cray (18.2)

You can power nodes up and down, limit ramp rate, and use power profiles for jobs. See Chapter 6, "Managing Power Usage", on page 315.

On Cray, PBS Creates One Vnode per Compute Node (18.2)

Default behavior on the Cray has changed to create one vnode per compute node. See section 11.8, "Automatic Configuration for Cray XC", on page 475.

Suspend and Resume on Cray (18.2)

You can use suspend and resume on Cray. See section 11.16.5, "Suspending and Resuming Jobs on Cray XC", on page 493

Installing PBS on Cray CLE 5.2 via RPM (18.2)

PBS is installed on Cray CLE 5.2 via RPM. See "Changes for Cray XC Installation" on page 142 in the PBS Professional Installation & Upgrade Guide.

Performance Enhancement for PBS on Cray via Improved MoM Reporting (18.2)

You can improve the performance of PBS on Cray by using the vnode_pool vnode attribute. This allows only one MoM to report inventory, and reduces communication traffic. See section 11.10, "Improving Server/MoM Inventory Performance for Cray XC", on page 481.

Periodic Synchronization of Inventory on Cray (18.2)

PBS periodically makes sure that its inventory matches what ALPS reports. See section 11.11, "Synchronizing PBS with ALPS Inventory on Cray XC", on page 483.

On Cray, Automatic Creation of One Vnode Per Compute Node (18.2)

PBS automatically creates one vnode for each compute node. See section 11.8, "Automatic Configuration for Cray XC", on page 475.

Installing PBS on CLE 6 via IMPS (18.2)

See "Installing PBS on CLE 6 and 7" on page 142 in the PBS Professional Installation & Upgrade Guide.

Support for Xeon Phi (18.2)

PBS supports Xeon Phi. See Chapter 11, "Support for Xeon Phi on Cray XC", on page 483.

1.2.1 New Scheduling Features

Restricting Placement Set Creation to Resources with Values that Have Been Set (18.2)

See section 4.9.32, “Placement Sets”, on page 168.

Soft Walltimes for Jobs (18.2)

You can set a soft walltime for jobs, and PBS can estimate a job's soft walltime. See section 4.9.44, “Using Soft Walltime”, on page 219

Formula Uses Fairshare (18.2)

You can use fairshare in the job sorting formula. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

Manage Partitions with Multischeds (18.2)

You can schedule each partition separately. See section 4.2, “Scheduling Each Partition Separately”, on page 55.

Run Jobs in a Cloud (18.2)

PBS can burst jobs to a cloud. See the *PBS Professional Cloud Guide*.

1.2.2 New Hooks Features

The execjob_prologue Hook Runs on All Sister MoMs (18.2)

The execjob_prologue hook runs on all sister MoMs. See "execjob_prologue: Event Just Before Execution of Top-level Job Process" on page 97 in the PBS Professional Hooks Guide.

Python Version Changed to 2.7.1 (18.2)

PBS 18.2.1 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Periodic Server Hook (18.2)

PBS has a periodic hook that runs at the server. See "periodic: Periodic Event at Server Host" on page 95 in the PBS Professional Hooks Guide.

Hook to Run Job Start Time Estimator (18.2)

PBS has a built-in hook named PBS_est that can run the job start time estimator. See section 4.9.15, “Estimating Job Start Time”, on page 131.

Configurable Python Interpreter Restarts (18.2)

You can configure how often you want the Python interpreter to restart. See "Restarting the Python Interpreter" on page 23 in the PBS Professional Hooks Guide.

PBS Can Report Custom Resources Set in Hooks (18.2)

MoM can accumulate and report custom resources that are set in a hook. See "Setting Job Resources in Hooks" on page 49 in the PBS Professional Hooks Guide

1.2.3 Other New Features

Managing Job Resource Use with Cgroups (18.2)

You can use cgroups to manage the resources used by jobs. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

Running Jobs in Containers (18.2)

Job submitters can run each job in its own container. See Chapter 18, "Configuring PBS for Containers", on page 615 and "Running Your Job in a Container", on page 130 of the PBS Professional User's Guide.

Power Provisioning (18.2)

PBS can monitor and control job power usage. See Chapter 6, "Managing Power Usage", on page 315.

Collecting Diagnostic Information with pbs_snapshot Command (18.2)

See "pbs_snapshot" on page 112 of the PBS Professional Reference Guide.

New pbs_ralter Command (18.2)

You can change reservations using the `pbs_ralter` command. See "Modifying Reservations", on page 142 of the PBS Professional User's Guide.

Privileged Access to Server for MoMs (18.2)

You can give all MoMs privileged access to the server without having to explicitly add their hosts to the acl_hosts server attribute. See section 8.3.7.3, "Access to Server for MoMs", on page 370.

Releasing Unneeded Vnodes from Jobs (18.2)

You can release vnodes that were allocated to jobs when those vnodes are no longer needed. See "Releasing Unneeded Vnodes from Your Job", on page 128 of the PBS Professional User's Guide.

Running Subjobs Survive Server Restart (18.2)

Subjobs of an array job will continue to run during a restart of the server.

Writing Output and Error Files Directly to Final Destination (18.2)

You can have PBS write your standard output and error files directly to their final destination. See "Writing Files Directly to Final Destination", on page 45 of the PBS Professional User's Guide.

Deleting Output and Error Files (18.2)

You can have PBS delete your standard output and error files. See "Avoiding Creation of stdout and/or stderr", on page 43 of the PBS Professional User's Guide.

Output for qstat in JSON and DSV Formats; qstat Attribute Output on Single Line (18.2)

You can get output from `qstat` in JSON or DSV formats. You can also print out attribute information in one unbroken line. See "qstat" on page 197 in the PBS Professional Installation & Upgrade Guide.

Specifying Resources to Release on Suspension (18.2)

You can specify which resources you want released when jobs are suspended. See section 5.9.6.2, "Job Suspension and Resource Usage", on page 250.

Maintenance State for Powered-up Vnodes (18.2)

You can suspend a job and put all the vnodes belonging to a job into the *maintenance* state. See section 15.4.2, "Performing Maintenance on Powered-up Vnodes", on page 554.

Debuginfo RPM Package (18.2)

PBS is packaged with a debuginfo RPM package. See section 21.1.3, "Using the debuginfo RPM Package", on page 663.

Logging Hostname and Interfaces (18.2)

Each time a log file is opened, PBS logs the hostname and interface information. See section 15.3, "Event Logging", on page 546.

Subjobs Survive Server Restarts (18.2)

Subjobs keep running after you stop the server. See "Impact of Stop-Restart on Running Linux Jobs" on page 169 in the PBS Professional Installation & Upgrade Guide and "Impact of Stop-Restart on Running Windows Jobs" on page 174 in the PBS Professional Installation & Upgrade Guide.

You can see all attributes for subjobs; see "Viewing Status of a Job Array", on page 157 of the PBS Professional User's Guide.

Jobs Can Use Provisioning for Some Chunks (18.2)

Jobs can request an AOE for some chunks as long as all chunks use the same AOE. See Chapter 14, "Using Provisioning", on page 215.

Node Licenses (18.2)

You can license your hosts using node licenses. See the *PBS Works Licensing Guide*.

PBS Can Send Mail for Subjobs (18.2)

PBS can send mail for subjobs. See "Specifying Email Notification", on page 25 of the PBS Professional User's Guide.

Server Periodic Hook (14.2)

You can run a hook periodically at the server. See "periodic: Periodic Event at Server Host" on page 95 in the PBS Professional Hooks Guide.

Hook to Run Job Start Time Estimator (14.2)

PBS has a built-in hook named PBS_est that can run the job start time estimator. See section 4.9.15, "Estimating Job Start Time", on page 131.

PBS Can Report Custom Resources Set in Hooks (14.2)

MoM can accumulate and report custom resources that are set in a hook. See section 5.2.4.12, "Setting Job Resources in Hooks", on page 49.

Configurable Python Interpreter Restarts (14.2)

You can configure how often you want the Python interpreter to restart. See "Restarting the Python Interpreter" on page 23 in the PBS Professional Hooks Guide.

Python Version Changed to 2.7.1 (14.2)

PBS 14.2.1 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Name for MoM to Use for Parent Vnode (14.2)

You can specify the name that MoM should use for her parent vnode and child vnodes. See section 3.3.2, "How to Choose Vnode Names", on page 38.

Grouping Jobs and Sorting by ID (14.2)

When getting job status, you can group jobs and sort them by ID. See "Grouping Jobs and Sorting by ID", on page 183 of the PBS Professional User's Guide.

Support for systemd (14.2)

PBS supports using sytemctl commands to start, stop, restart, and status PBS. See "Methods for Starting, Stopping, or Restarting PBS" on page 160 in the PBS Professional Installation & Upgrade Guide.

Support for Native Package Managers on Linux (14.2)

PBS supports use of RPM for installation and upgrading. See "Installation" on page 19 in the PBS Professional Installation & Upgrade Guide and "Upgrading" on page 65 in the PBS Professional Installation & Upgrade Guide.

Server Sets Job Comment on Run or Reject (14.2)

The server sets the job comment when the job is run or rejected. See section 14.7.3.1, "Comment Set When Running Job", on page 515.

Update to Accounting R Record (14.2)

PBS writes the R accounting record when MoM is restarted with -p or -r. See section , “R”, on page 637.

Interactive GUI Jobs on Windows (13.1)

Users can run interactive GUI jobs on Windows. See "Submitting Interactive GUI Jobs on Windows", on page 125 of the PBS Professional User’s Guide.

Administrators can choose a remote viewer for interactive GUI jobs. See section 14.20.1, “Configuring PBS for Remote Viewer on Windows”, on page 535.

MUNGE Integration (13.1)

PBS can use MUNGE to create and validate credentials. See section 8.4.4, “Authentication via MUNGE”, on page 379.

Controlling Backfill Depth at the Queue (13.1)

Administrators can choose the backfilling depth independently at each queue. See section 4.9.3, “Using Backfilling”, on page 105.

Optional Scheduler Cycle Speedup (13.1)

You can optionally speed up the scheduling cycle. See section 4.9.40, “Scheduler Cycle Speedup”, on page 209.

Preventing Some Jobs from Being Top Jobs (13.1)

You can prevent a job from being a top job by setting its topjob_ineligible attribute to True. See section 4.9.17.1, “Making Jobs Ineligible to be Top Jobs”, on page 137.

Improved Mail on Windows (13.1)

Under Windows, you can specify an SMTP server. (As of 19.4.1, PBS does not use an SMTP server.)

New Hook Events (13.0)

PBS provides three new hook events:

- An execjob_launch hook runs just before MoM runs the user's program
- An execjob_attach hook runs when pbs_attach is called
- An exechost_startup hook runs when MoM starts up

See "When Hooks Run" on page 15 in the PBS Professional Hooks Guide, "execjob_launch: Event when Execution Host Receives Job" on page 98 in the PBS Professional Hooks Guide, "execjob_attach: Event when pbs_attach() runs" on page 100 in the PBS Professional Hooks Guide, and "exechost_startup: Event When Execution Host Starts Up" on page 106 in the PBS Professional Hooks Guide.

Configuration Files for Hooks (13.0)

You can use configuration files with hooks. See "Using Hook Configuration Files" on page 32 in the PBS Professional Hooks Guide.

Configuring Vnodes in Hooks (13.0)

You can use hooks to configure vnode attributes and resources. See "Setting and Unsetting Vnode Resources and Attributes" on page 48 in the PBS Professional Hooks Guide.

Adding Custom Resources in Hooks (13.0)

You can use hooks to add custom non-consumable host-level resources. See "Adding Custom Non-consumable Host-level Resources" on page 64 in the PBS Professional Hooks Guide.

Node Health Hook Features (13.0)

PBS has node health checking features for hooks. You can offline and clear vnodes, and restart the scheduling cycle. See "Offlining and Clearing Vnodes Using the fail_action Hook Attribute" on page 66 in the PBS Professional Hooks Guide and "Restarting Scheduler Cycle After Hook Failure" on page 63 in the PBS Professional Hooks Guide.

Hook Debugging Enhancements (13.0)

You can get hooks to produce debugging information, and then read that information in while debugging hooks. See "Debugging Hooks" on page 159 in the PBS Professional Hooks Guide.

Managing Built-in Hooks (13.0)

You can enable and disable built-in hooks. See "Managing Built-in Hooks" on page 155 in the PBS Professional Hooks Guide.

Scheduler Does not Trigger modifyjob Hooks (13.0)

The scheduler does not trigger modifyjob hooks. See the PBS Professional Hooks Guide.

Faster, Asynchronous Communication Between Daemons (13.0)

PBS has a communication daemon that provides faster, asynchronous communication between the server, scheduler, and MoM daemons. See “Communication” on page 45 in the PBS Professional Installation & Upgrade Guide.

Enhanced Throughput of Jobs (13.0)

By default, the scheduler runs asynchronously to speed up job start, and jobs that have been altered via `qalter`, server_dyn_res, or peering can run in the same scheduler cycle in which they were altered. See section 4.5.7.1, “Improving Throughput of Jobs”, on page 97.

Creating Custom Resources via qmgr (13.0)

You can create any custom resources using nothing but the `qmgr` command. See section 5.14.2.4, “Defining Custom Resources via qmgr”, on page 263.

Job Sorting Formula: Python Math Functions and Threshold (13.0)

You can use standard Python math functions in the job sorting formula. You can also set a threshold for job priority, below which jobs cannot run. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

Fairshare: Formula and Decay Factor (13.0)

You can use a mathematical formula for fairshare, and you can set a custom decay factor. See section 4.9.19, “Using Fairshare”, on page 138.

Preempted Jobs can be Top Jobs (13.0)

You can specify that preempted jobs should be classified as top jobs. See section 4.9.16, “Calculating Job Execution Priority”, on page 134. You can use a new scheduler attribute called sched_preempt_enforce_resumption for this; see section 4.9.3, “Using Backfilling”, on page 105.

Limiting Preemption Targets (13.0)

You can specify which jobs can be preempted by a given job. See section 4.9.33.4.i, “Setting Job Preemption Targets”, on page 182.

Limiting Number of Jobs in Execution Queues (13.0)

You can speed up the scheduling cycle by limiting the number of jobs in execution queues. See section 4.5.7.2, “Limiting Number of Jobs Queued in Execution Queues”, on page 98.

Improved Round-robin Behavior (13.0)

The round_robin scheduler parameter produces improved behavior. See section 4.9.38, “Round Robin Queue Selection”, on page 204.

Limiting Resources Allocated to Queued Jobs (13.0)

You can set limits on the amounts of resources allocated to queued jobs specifically. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

Running qsub in the Foreground (13.0)

By default, the `qsub` command runs in the background. You can run it in the foreground using the `-f` option. See "qsub" on page 213 of the PBS Professional Reference Guide.

Windows Users can Use UNC Paths (13.0)

Windows users can use UNC paths for job submission and file staging. See "Set up Paths", on page 9 of the PBS Professional User's Guide and "Using UNC Paths", on page 34 of the PBS Professional User's Guide.

Automatic Installation and Upgrade of Database (13.0)

PBS automatically installs or upgrades its database. See "Automatic Upgrade of Database (13.0)" on page 66 in the PBS Professional Installation & Upgrade Guide.

Longer Job and Reservation Names (13.0)

You can use job and reservation names up to 236 characters in length. See "Formats" on page 353 of the PBS Professional Reference Guide.

Address Disambiguation for Multihomed Systems (13.0)

You can disambiguate addresses for contacting the server, sending mail, sending outgoing traffic, and delivering output and error files. See "PBS with Multihomed Systems" on page 60 in the PBS Professional Installation & Upgrade Guide.

Support for Hydra Process Manager in Intel MPI (13.0)

Intel MPI is integrated with PBS. See "Integrating Intel MPI 4.0.3 On Linux Using Environment Variables" on page 453.

Enhancements to pbsnodes Command (13.0)

You can now use the `pbsnodes` command to edit the comment attribute of a host, to write out host information, and to operate on specific vnodes. See "pbsnodes" on page 35.

Primary Group of Job Owner or Reservation Creator Automatically Added to Job group_list (13.0)

The job submitter's and reservation creator's primary group is automatically added to the job or reservation group_list attribute. See "qsub" on page 213 and "pbs_rsub" on page 96.

Intel MPI Integrated under Windows (13.0)

MPI is integrated with PBS under Windows (as well as Linux). See "Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script" on page 454.

MPICH2 Integrated under Windows (13.0)

MPICH2 is integrated with PBS under Windows (as well as Linux). See "Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script" on page 454.

PBS pbsdsh Command Available under Windows (13.0)

The pbsdsh command is available under Windows. See "pbsdsh" on page 29.

PBS TM APIs Available under Windows (13.0)

The PBS TM APIs are available under Windows. See "TM Library" on page 93 of the PBS Professional Programmer's Guide.

PBS pbs_attach Command Available under Windows (13.0)

The pbs_attach command is available under Windows. See "pbs_attach" on page 55.

Xeon Phi Reported on Cray (13.0)

PBS automatically detects and reports a Xeon Phi in the ALPS inventory. See "Using Xeon Phi Vnodes on Cray XC", on page 195 of the PBS Professional User's Guide.

Command Line Editing in qmgr (12.2)

The `qmgr` command provides a history and allows you to edit command lines. See "Reusing and Editing the qmgr Command Line" on page 150 of the PBS Professional Reference Guide.

Interactive Jobs Available under Windows (12.2)

Job submitters can run interactive jobs under Windows. See "Running Your Job Interactively", on page 121 of the PBS Professional User's Guide.

Job Run Count is Writable (12.2)

Job submitters and administrators can set the value of a job's run count. See section 14.18, "Managing Number of Run Attempts", on page 534 and "Controlling Number of Times Job is Re-run", on page 119 of the PBS Professional User's Guide.

runjob Hook can Modify Job Attributes (12.2)

The runjob hook can modify a job's attributes and resources. See "Using Attributes and Resources in Hooks" on page 44 in the PBS Professional Hooks Guide.

Jobs can be Suspended under Windows (12.2)

You can suspend and resume a job under Windows.

Configuration of Directory for PBS Component Temporary Files (12.2)

You can configure the root directory where you want PBS components to put their temporary files. See section 15.8, "Temporary File Location for PBS Components", on page 567.

Execution Event and Periodic Hooks (12.0)

You can write hooks that run at the execution host when the job reaches the execution host, when the job starts, ends, is killed, and is cleaned up. You can also write hooks that run periodically on all execution hosts. See the PBS Professional Hooks Guide.

Shrink-to-fit Jobs (12.0)

PBS allows users to specify a variable running time for jobs. Job submitters can specify a walltime range for jobs where attempting to run the job in a tight time slot can be useful. Administrators can convert non-shrink-to-fit jobs into shrink-to-fit jobs in order to maximize machine use. See "Adjusting Job Running Time", on page 110 of the PBS Professional User's Guide and section 4.9.42, "Using Shrink-to-fit Jobs", on page 211.

PBS Supports Socket Licensing (11.3)

PBS lets you use socket licenses to license hosts. See the *PBS Works Licensing Guide*.

Deleting Job History (11.3)

You can delete job histories. See section 14.15.9, "Deleting Moved Jobs and Job Histories", on page 533.

Managing Resource Usage by Project (11.2)

You can set resource usage limits for projects, at the server and queue. You can set limits for the amount of each resource being used, or for the number of jobs. Jobs have a new attribute called *project*. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

PBS Daemons Protected from OOM Killer (11.2)

PBS daemons are protected from being terminated by an OOM killer. See section 9.8, "OOM Killer Protection", on page 449.

PBS Supports X Forwarding for Interactive Jobs (11.2)

PBS allows users to receive X output from interactive jobs. See "Receiving X Output from Interactive Linux Jobs", on page 124 of the PBS Professional User's Guide, and section 15.2.1.1, "Contents of Environment File", on page 545.

Support for Accelerators on Cray (11.2)

PBS provides tight integration for accelerators on Cray. See Chapter 11, "Configuring PBS for Cray", on page 471.

Support for Interlagos on Cray (11.1)

No longer supported.

Improved Cray Integration (11.0)

PBS is more tightly integrated with Cray systems. You can use the PBS select and place language when submitting Cray jobs. See section , “Configuring PBS for Cray”, on page 471.

Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See the PBS Professional Hooks Guide.

Enhanced Job Placement (11.0)

PBS allows job submitters to scatter chunks by vnode in addition to scattering by host. PBS also allows job submitters to reserve entire hosts via a job's placement request. See "Specifying Job Placement", on page 64 of the PBS Professional User’s Guide.

Choice in PBS service account Name (11.0)

Under Windows, the PBS service account used to run PBS daemons can have any name. See “Creating PBS Service Account in Domained Environment” on page 40 in the PBS Professional Installation & Upgrade Guide.

Change of Licensing Method (11.0)

As of 11.0, PBS is licensed using a new Altair license server. See the *PBS Works Licensing Guide*.

Change in Data Management (11.0)

PBS uses a new data service. See section 15.5, “Managing the Data Service”, on page 557.

Choice in Job Requeue Timeout (11.0)

You can choose how long the job requeue process should be allowed to run. See section 9.6.3, “Setting Job Requeue Timeout”, on page 445.

Backfilling Around Top N Jobs (10.4)

PBS can backfill around the most deserving jobs. You can configure the number of jobs PBS backfills around. See section 4.9.3, “Using Backfilling”, on page 105.

Estimating Job Start Times (10.4)

PBS can estimate when jobs will run, and which vnodes each job will use. See section 4.9.15, “Estimating Job Start Time”, on page 131.

Unified Job Submission (10.4)

PBS allows users to submit jobs using the same scripts, whether the job is submitted on a Windows or Linux system. See "Python Job Scripts", on page 14 of the PBS Professional User’s Guide.

Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 7, "Provisioning", on page 327.

New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See the PBS Professional Hooks Guide.

New Scheduler Attribute (10.2)

PBS allows the administrator to set the scheduler's cycle time using the new sched_cycle_length scheduler attribute. See the `pbs_sched_attributes(7B)` manual page.

Walltime as Checkpoint Interval Measure (10.2)

PBS allows a job to be checkpointed according to its walltime usage. See the `pbs_job_attributes(7B)` manual page.

Managing Resource Usage (10.1)

You can set separate limits for resource usage by individual users, individual groups, generic users, generic groups, and the total used. You can limit the amount of resources used, and the number of queued and running jobs. These limits can be defined separately for each queue and for the server. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287. These new limits are incompatible with the limit attributes existing before Version 10.1.

Managing Job History (10.1)

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used. PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server. See section 14.15, “Managing Job History”, on page 530.

Reservation Fault Tolerance (10.1)

PBS attempts to reconfirm reservations for which associated vnodes have become unavailable. See section 9.4, “Reservation Fault Tolerance”, on page 432.

Checkpoint Support via Epilogue (10.1)

Checkpointed jobs can be requeued if the epilogue exits with a special value. See section 9.3.7.3, “Requeueing via Epilogue”, on page 429.

Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See the PBS Professional Hooks Guide.

Versioned Installation (10.0)

PBS is now automatically installed in versioned directories. For most platforms, different versions of PBS can coexist, and upgrading is simplified. See Chapter 3, "Installation", on page 19 and Chapter 6, "Upgrading", on page 65 in the PBS Professional Installation and Upgrade Guide.

Resource Permissions for Custom Resources (9.2)

You can set permissions on custom resources so that they are either invisible to users or cannot be requested by users. This also means that users cannot modify a resource request for those resources via `qalter`. See section 5.14.2.3.vi, “Resource Permission Flags”, on page 260.

Extension to Job Sorting Formula (9.2)

The job sorting formula has been extended to include parentheses, exponentiation, division, and unary plus and minus. See section 4.9.3, “Using Backfilling”, on page 105.

Eligible Wait Time for Jobs (9.2)

A job that is waiting to run can be accruing "eligible time". Jobs can accrue eligible time when they are blocked due to a lack of resources. This eligible time can be used in the job sorting formula. Jobs have two new attributes, eligible_time and accrue_type, which indicates what kind of wait time the job is accruing. See section 4.9.13, “Eligible Wait Time for Jobs”, on page 126.

Job Staging and Execution Directories (9.2)

PBS now provides per-job staging and execution directories. Jobs have new attributes sandbox and jobdir, the MoM has a new option `$jobdir_root`, and there is a new environment variable called PBS_JOBDIR. If the job's sandbox attribute is set to `PRIVATE`, PBS creates a job-specific staging and execution directory. If the job's sandbox attribute is unset or is set to `HOME`, PBS uses the user's home directory for staging and execution, which is how previous versions of PBS behaved. If MoM's `$jobdir_root` is set to a specific directory, that is where PBS will create job-specific staging and execution directories. If MoM's `$jobdir_root` is unset, PBS will create the job-specific staging and execution directory under the user's home directory. See section 14.13.1, “Staging and Execution Directories for Job”, on page 523.

Standing Reservations (9.2)

PBS now provides both advance and standing reservation of resources. A standing reservation is a reservation of resources for specific recurring periods of time. See section 4.9.37, “Reservations”, on page 197.

New Server Attribute for Job Sorting Formula (9.1)

The new server attribute "job_sort_formula" is used for sorting jobs according to a site-defined formula. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

Change to sched_config (9.1)

The default for job_sort_key of "cput" is commented out in the default `sched_config` file. It is left in as a usage example.

Change to Licensing (9.0)

PBS now depends on an Altair license server that will hand out licenses to be assigned to PBS jobs. See the *PBS Works Licensing Guide*. PBS Professional versions 8.0 and below will continue to be licensed using the proprietary licensing scheme.

Installing With Altair Licensing (9.0)

If you will use floating licenses, we recommend that you install and configure the Altair license server before installing and configuring PBS. PBS starts up faster. See “Overview of Installation” on page 19 in the PBS Professional Installation & Upgrade Guide.

Unset Host-level Resources Have Zero Value (9.0)

An unset numerical resource at the host level behaves as if its value is zero, but at the server or queue level it behaves as if it were infinite. An unset string or string array resource cannot be matched by a job's resource request. An unset boolean resource behaves as if it is set to "*False*". See section 4.9.28.7, “Matching Unset Resources”, on page 161.

Better Management of Resources Allocated to Jobs (9.0)

The resources allocated to a job from vnodes will not be released until certain allocated resources have been freed by all MoMs running the job. The end of job accounting record will not be written until all of the resources have been freed. The "end" entry in the job end ('E') record will include the time to stage out files, delete files, and free the resources. This will not change the recorded "walltime" for the job.

1.3 Commercial-only Features

PBS is dual-licensed. Altair releases a commercial version and an open-source version. The core of the product is the same, but the commercial version contains additional features available only in the commercial (licensed) version of PBS. For example:

- Estimated start times for non-top jobs via `pbs_est`
- Container integration
- PBS licensing

1.4 Backward Compatibility

1.4.1 New and Old Resource Usage Limits Incompatible

The new resource usage limits are incompatible with the old resource usage limits. See section 5.15.1.15, “Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1”, on page 302, section 5.15.1.13.v, “Do Not Mix Old And New Limits”, on page 301, and section 5.15.1.14.i, “Error When Setting Limit Attributes”, on page 301.

1.4.2 Job Dependencies Affected By Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history, PBS releases j1's dependency, and takes appropriate action. If job j1 depends on a finished job j3 that has been purged from job history, j1 is rejected just as in previous versions of PBS where the job was no longer in the system.

1.4.3 PBS path information no longer saved in AUTOEXEC.BAT

Any value for PATH saved in AUTOEXEC.BAT may be lost after installation of PBS. If there is any path information that needs to be saved, AUTOEXEC.BAT must be edited by hand after the installation of PBS. PBS path information is no longer saved in AUTOEXEC.BAT.

1.4.4 OS-level Checkpointing Not Supported

PBS does not directly support OS-level checkpointing. PBS supports checkpointing using site-supplied methods. See section 9.3, “Checkpoint and Restart”, on page 418.

1.4.5 Scheduler Parameters Changed to Scheduler Attributes (19.4.1)

The preempt_order, preempt_prio, preempt_queue_prio, and preempt_sort preemption settings were scheduler parameters in `$PBS_HOME/sched_priv/sched_config` in older versions of PBS. They are now scheduler attributes with the same names and formats. You cannot use the old parameters. Make sure that you use `qmgr` to set the attributes as desired. See “Scheduler Attributes” on page 298 of the PBS Professional Reference Guide.

2 Configuring the Server and Queues

This chapter describes how to configure the server and any queues.

2.1 The Server

2.1.1 Configuring the Server

You configure the server by setting server attributes via the `qmgr` command:

```
Qmgr: set server <attribute> = <value>
```

For a description of the server attributes, see "Server Attributes" on page 281 of the PBS Professional Reference Guide.

For a description of the `qmgr` command, see "qmgr" on page 149 of the PBS Professional Reference Guide.

2.1.2 Default Server Configuration

The default configuration from the binary installation sets the default server settings. An example server configuration is shown below:

```
qmgr
Qmgr: print server
#
# Create queues and set their attributes.
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server query_other_jobs = True
set server resources_default.ncpus = 1
set server resv_enable = True
set server node_fail_requeue = 310
set server max_array_size = 10000
set server default_chunk.ncpus=1
```

2.1.3 The PBS Node File

The server creates a file of the nodes managed by PBS. This node file is written only by the server. On startup each MoM sends a time-stamped list of her known vnodes to the server. The server updates its information based on that message. If the time stamp on the vnode list is newer than what the server recorded before in the node file, the server will create any vnodes which were not already defined. If the time stamp in the MoM's message is not newer, then the server will not create any missing vnodes and will log an error for any vnodes reported by MoM but not already known.

Whenever new vnodes are created, the server sends a message to each MoM with the list of MoMs and each vnode managed by the MoMs. The server will only delete vnodes when they are explicitly deleted via `qmgr`.

This is different from the node file created for each job. See "The Job Node File", on page 77 of the PBS Professional User's Guide.

2.1.4 Server Configuration Attributes

See "Server Attributes" on page 281 of the PBS Professional Reference Guide for a table of server attributes.

2.1.5 Recording Server Configuration

If you wish to record the configuration of a PBS server for re-use later, you may use the `print` subcommand of `qmgr(8B)`. For example,

```
qmgr -c "print server" > /tmp/server.out
qmgr -c "print node @default" > /tmp/nodes.out
```

will record in the file `/tmp/server.out` the `qmgr` subcommands required to recreate the current configuration including the queues. The second file generated above will contain the vnodes and all the vnode properties. The commands could be read back into `qmgr` via standard input:

```
qmgr < /tmp/server.out
qmgr < /tmp/nodes.out
```

2.1.6 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. The Globus MoM is no longer available.

2.1.7 Configuring the Server for Licensing

The PBS server must be configured for licensing. You must set the location where PBS will look for the license server(s), by setting the server attribute pbs_license_info, then force the server to re-query for licenses by setting the server's scheduling attribute to *True*. The other server licensing attributes have defaults, but you may wish to set them as well. See the *PBS Works Licensing Guide*.

You may also wish to have redundant license servers. See the *Altair License Management System Installation and Operations Guide*, available at `www.pbsworks.com`.

2.2 How PBS Uses Mail

PBS sends mail to the administrator for administration-related issues, and to job submitters for job-related issues. See "Specifying Email Notification", on page 25 of the PBS Professional User's Guide for information about mail PBS sends to job submitters.

PBS sends mail to the administrator under the following circumstances:

- When failover occurs, PBS sends an email is sent to and from the account defined in the server's mail_from attribute.
- When the database is stopped unexpectedly. For example:

  ```
  "Panic shutdown of Server on database error.  Please check PBS_HOME file system for no space
      condition."
  ```

- When your license is expiring, PBS sends mail once a day.

2.2.1 Configuring Server Mail Address

You can configure the account that is used as the address to both send and receive administrative mail. These are the same account. For example, when failover occurs, an email is sent to and from the account defined in the server's mail_from attribute, saying that failover has occurred.

Use the `qmgr` command to set the mail_from server attribute to an address that is monitored regularly:

```
Qmgr: s server mail_from=<address>
```

You cannot configure which mail server PBS uses. PBS uses the default mail server. On Linux, this is `/usr/lib/sendmail`.

2.2.2 Specifying Mail Delivery Domain

You can use the PBS_MAIL_HOST_NAME parameter in `pbs.conf` on the server host to direct mail to a domain in which the user can receive it. For example, if a job is submitted from a cluster node, it may not be possible for mail to be delivered there, especially if the job runs on a different cluster.

You can specify the destination domain for email that is sent by the server to the administrator or to job submitters or reservation creators by setting the PBS_MAIL_HOST_NAME parameter in `pbs.conf`.

2.2.2.1 Delivering Mail to Administrator

The default user name for administrative mail is "adm". The following table shows where PBS sends administrator mail:

Table 2-1: How PBS Sets Administrator Mail

Value of mail_from	Destination
<username>@<hostname>	*<username>@<hostname>*
user	*user* *(Destination depends on mail server configuration)*
unset	*adm* *(Destination depends on mail server configuration)*

2.2.2.2 Delivering Mail to Job Submitter or Reservation Creator

The Mail_Users attribute is a list of one or more user names. For each entry in the list, PBS handles the entry according to the rules in the following table showing where PBS sends job or reservation mail:

Table 2-2: How PBS Sets Job or Reservation Mail

Value of Mail_Users	Value of PBS_MAIL_HOST_NAME	
	Set	Unset
<username>@<hostname>	Linux: *<username>@<hostname>*	*<username>@<hostname>*
user	*user@PBS_MAIL_HOST_NAME*	*user@<server FQDN from Job_Owner attribute of job>*
unset	*<job owner>@ PBS_MAIL_HOST_NAME*	*Linux: <job owner>@ <server FQDN from Job_Owner attribute of job>*

2.2.3 Attributes, Parameters Etc. Affecting Mail

PBS_MAIL_HOST_NAME

Parameter in pbs.conf. Optional. Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. See section 2.2.2, “Specifying Mail Delivery Domain”, on page 20.

Should be a fully qualified domain name. Cannot contain a colon (":").

mail_from

Server attribute. Mail is sent from and to this account when failover occurs.

Mail_Users

Job and reservation attribute. List of users to whom mail about job or reservation is sent.

Mail_Points

Job and reservation attribute. List of events where PBS sends mail to users who are the job or reservation owner, or are listed in the Mail_Users job or reservation attribute.

PBS_O_MAIL

Value of MAIL environment variable, taken from job submitter's environment.

2.3 Queues

When a job is submitted to PBS and accepted, it is placed in a queue. Despite the fact that the name implies first-in, first-out ordering of jobs, this is not the case. Job submission order does not determine job execution order. See Chapter 4, "Scheduling", on page 53.

You can create different queues for different purposes: queues for certain kinds of jobs, queues for specific groups, queues for specific vnodes, etc. You can tell PBS how to automatically route jobs into each queue. PBS has a default execution queue named *workq*, where jobs are placed when no queue is requested; you can specify a different default queue. See section 2.3.14, “Specifying Default Queue”, on page 32.

2.3.1 Kinds of Queues

2.3.1.1 Execution and Routing Queues

There are two main types of PBS queues: *routing* and *execution*.

- A routing queue is used only to move jobs to other queues (destination queues). These destination queues can be routing or execution queues, and can be located at different PBS servers. For more information on creating and using routing queues, see section 2.3.6, “Routing Queues”, on page 25.
- An execution queue is used as the home for a waiting or running job. A job must reside in an execution queue to be eligible to run. The job remains in the execution queue during the time it is running. See section 2.3.5, “Execution Queues”, on page 23.

2.3.1.2 Available Kinds of Queues

PBS supplies the following kinds of execution and routing queues:

Table 2-3: Kinds of Queues

Kind of Queue		Description	Link
Routing queues		Used for moving jobs to another queue	See section 2.3.6, "Routing Queues", on page 25
Execution queues	Reservation queues	Created for reservation. Do not operate on these directly; instead, operate on the reservation.	See section 2.3.5.2.iv, "Reservation Queues", on page 24
	Dedicated time queues	Holds jobs that run only during dedicated time.	See section 2.3.5.2.i, "Dedicated Time Queues", on page 24
	Primetime queues	Holds jobs that run only during primetime.	See section 2.3.5.2.ii, "Primetime and Non-Primetime Queues", on page 24
	Non-primetime queues	Holds jobs that run only during non-primetime.	See section 2.3.5.2.ii, "Primetime and Non-Primetime Queues", on page 24
	Anytime queues	Queue with no dedicated time or primetime restrictions	See section 2.3.5.2.iii, "Anytime Queues", on page 24
	Express queues	High-priority queue; priority is set to the level signifying that it is an express queue	See section 2.3.5.3.i, "Express Queues", on page 24
	Anti-express queue	Low-priority queue designed for work that should run only when no other jobs need the resources	See section 4.9.1, "Anti-Express Queues", on page 102

2.3.2 Basic Queue Use

The simplest form of PBS uses just one queue. The queue is an execution queue named *workq*. This queue is always created, enabled, and started for you during installation. After a basic installation, this queue is ready to hold jobs submitted by users.

2.3.3 Creating Queues

To create a queue, use the `qmgr` command to create it and set its queue_type attribute:

```
Qmgr: create queue <queue name>
Qmgr: set queue <queue_name> queue_type = <execution or route>
```

For example, to create an execution queue named *exec_queue*, set its type, start it, and enable it:

```
Qmgr: create queue exec_queue
Qmgr: set queue exec_queue queue_type = execution
Qmgr: set queue exec_queue enabled = True
Qmgr: set queue exec_queue started = True
```

Now we will create a routing queue, which will send jobs to our execution queue:

```
Qmgr: create queue routing_queue
Qmgr: set queue routing_queue queue_type = route
Qmgr: set queue routing_queue route_destinations = exec_queue
```

2.3.4 Enabling, Disabling, Starting, and Stopping Queues

When you *enable* a queue, you allow it to accept jobs, meaning that jobs can be enqueued in the queue. When you *disable* a queue, you disallow it from accepting jobs. Queues are disabled by default. You enable a queue by setting its enabled attribute to *True*:

```
Qmgr: set queue <queue name> enabled = True
```

When you *start* a queue, you allow the jobs in the queue to be executed. Jobs are selected to be run according to the scheduling policy. When you *stop* a queue, you disallow jobs in that queue from running, regardless of scheduling policy. Except for the default queue, queues are stopped by default. You start a queue by setting its started attribute to *True*:

```
Qmgr: set queue <queue name> started = True
```

2.3.5 Execution Queues

Execution queues are used to run jobs; jobs must be in an execution queue in order to run. PBS does not route from execution queues.

2.3.5.1 Where Execution Queues Get Their Jobs

By default, PBS allows jobs to be moved into execution queues via the `qmove` command, by hooks, from routing queues, and by being submitted to execution queues. You can specify that an execution queue should accept only those jobs that are routed from a routing queue by PBS, by setting the queue's from_route_only attribute to *True*:

```
Qmgr: set queue <queue name> from_route_only = True
```

2.3.5.2 Execution Queues for Specific Time Periods

PBS provides a mechanism that allows you to specify that the jobs in an execution queue can run only during specific time periods. PBS provides a different kind of execution queue for each kind of time period. The time periods you can specify are the following:

Reservations

You can create an advance, standing, job-specific, or maintenance reservation. See section 4.9.37, “Reservations”, on page 197.

Dedicated time

Dedicated time is a period of time with a defined beginning and end. You can define multiple dedicated times.

Primetime

Primetime is a recurring time period with a defined beginning and end. You can define primetime to be different for each day of the week.

Non-primetime

Non-primetime is a recurring time period with a defined beginning and end. Non-primetime begins when primetime ends, and vice versa.

Holidays

Holidays are dates defined in the `<sched_priv directory>/holidays` file. PBS provides an example file with everything commented out, and you define your own holidays and primetime. Holiday time is treated like non-primetime, meaning jobs in non-primetime queues run during holiday time.

Anytime queue

The term "anytime queue" means a queue that is not a primetime or a non-primetime queue.

2.3.5.2.i Dedicated Time Queues

The jobs in a dedicated time execution queue can run only during dedicated time. Dedicated time is defined in `<sched_priv directory>/dedicated_time`. See section 4.9.10, "Dedicated Time", on page 125.

To specify that a queue is a dedicated time queue, you prefix the queue name with the dedicated time keyword. This keyword defaults to "*ded*", but can be defined in the dedicated_prefix scheduler parameter in `<sched_priv directory>/sched_config`. See "dedicated_prefix" on page 250 of the PBS Professional Reference Guide.

2.3.5.2.ii Primetime and Non-Primetime Queues

The jobs in a primetime queue run only during primetime, and the jobs in a non-primetime queue run only during non-primetime. Primetime and non-primetime are defined in `<sched_priv directory>/holidays`. See section 4.9.34, "Using Primetime and Holidays", on page 191.

To specify that a queue is a primetime or non-primetime queue, you prefix the queue name with the primetime or non-primetime keyword. For primetime, this keyword defaults to "*p_*", and for non-primetime, the keyword defaults to "*np_*", but these can be defined in the primetime_prefix and nonprimetime_prefix scheduler parameters in `<sched_priv directory>/sched_config`. See "Scheduler Parameters" on page 249 of the PBS Professional Reference Guide.

2.3.5.2.iii Anytime Queues

An anytime queue is a queue whose jobs can run at any time. An anytime queue is simply a queue that is not a dedicated time, primetime, or non-primetime queue.

2.3.5.2.iv Reservation Queues

When the `pbs_rsub` command is used to create a reservation or to convert a job into a reservation job, PBS creates a reservation queue. Jobs in the queue run only during the reservation. Do not operate on these queues directly; instead, operate on the reservations. See section 4.9.37, "Reservations", on page 197.

2.3.5.3 Prioritizing Execution Queues

You can set the priority of each execution queue as compared to the other queues in this complex by specifying a value for the priority queue attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

A higher value for priority means the queue has greater priority. There is no limit to the priority that you can assign to a queue, however it must fit within integer size. See "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

For how queue priority is used in scheduling, see section 4.9.36, "Queue Priority", on page 196.

2.3.5.3.i Express Queues

A queue is an *express queue* if its priority is greater than or equal to the value that defines an express queue. This value is set in the preempt_queue_prio parameter in `<sched_priv directory>/sched_config`. The default value for preempt_queue_prio is *150*.

You do not need to set by_queue to *True* in order to use express queues.

For how express queues can be used, see section 4.9.18, “Express Queues”, on page 137.

2.3.6 Routing Queues

A routing queue is used only to route jobs to other queues; jobs cannot run from a routing queue.

A routing queue has the following properties:

- Can route to multiple destination queues
- For each job, tries destination queues in the order listed, starting at the top of the list.
- Can route to execution queues
- Can route to other routing queues
- Can route to queues in other complexes (at other servers)

Destinations can be specified in the following ways:

```
route_destinations = Q1
route_destinations = Q1@Server1
route_destinations = "Q1, Q2@Server1, Q3@Server2"
route_destinations += Q1
route_destinations += "Q4, Q5@Server3"
```

2.3.6.1 How Routing Works

Whenever a job is in a started routing queue, PBS immediately attempts to route the job to a destination queue. When PBS routes a job, it starts at the top of the destination list and tries each destination in the order listed. The job's destination is the first queue that accepts it. The result is one of the following:

- The job is routed to one of the destination queues.
- The attempt to route is permanently rejected by each destination queue, and the job is deleted.
- Every destination queue rejects the job, but at least one rejection is temporary. In this case, the destination is tried again later, after the amount of time specified in the routing queue's route_retry_time attribute.
- If the job exceeds the time set in the queue's route_lifetime attribute, the job is deleted.

If there are multiple routing queues containing jobs to be routed, the routing queues are processed in the order in which they are displayed in the output of a `qstat -Q` command.

Queue priority does not play a role in routing jobs.

2.3.6.2 Requirements for Routing Queues

- A routing queue's destination queues must be created before being specified in the routing queue's route_destinations attribute.
- A routing queue's route_destinations attribute must be specified before enabling and starting the routing queue.
- A routing queue must be enabled in order to route jobs.

2.3.6.3 Caveats and Advice for Routing Queues

- Avoid routing loops. If a job makes more than 20 routing hops, it is discarded, and PBS sends mail to the job owner if the job's Mail_Points attribute contains "a" for "abort". Avoid setting a routing queue's destination to be the routing queue itself.
- When routing to a complex that is using failover, it's a good idea to include the names of both primary and secondary servers in a routing destination:

  ```
  route_destinations = "destQ@primary_server, destQ@secondary_server"
  ```
- When routing a job between complexes, the job's owner must be able to submit a job to the destination complex.
- When routing to a destination in another complex, the source and destination complexes should use the same version of PBS. If not, you may need a submission hook to modify incoming jobs.
- It is recommended to list the destination queues in order of the most restrictive first, because the first queue which meets the job's requirements and is enabled will be its destination

2.3.6.4 Using Resources to Route Jobs Between Queues

You can use resources to direct jobs to the desired queues. The server will automatically route jobs that are in routing queues, based on job resource requests. The destination queue can be at the local server or at another server. If you have more than one PBS complex, you may want to route jobs between the complexes, depending on the resources available at each complex.

You can set up queues for specific kinds of jobs, for example jobs requesting very little memory, a lot of memory, or a particular application. You can then route jobs to the appropriate queues.

A routing queue tests destination queues in the order listed in the queue's route_destinations attribute. The job is placed in the first queue that meets the job's request and is enabled.

Please read all of the subsections for this section.

2.3.6.4.i How Queue and Server Limits Are Applied, Except Running Time

The following applies to to all resources except for min_walltime and max_walltime.

You can set a minimum and a maximum for each resource at each queue using the resources_min.<resource name> and resources_max.<resource name> queue attributes. Any time a job is considered for entry into a queue, the job's resource request is tested against resources_min.<resource name> and resources_max.<resource name> for that queue. The job's resource request must be greater than or equal to the value specified in resources_min.<resource name>, and less than or equal to the value specified in resources_max.<resource name>.

The job is tested only against existing resources_min.<resource name> and resources_max.<resource name> for the queue.

Only those resources that are specified in the job's resource request are tested, so if a job does not request a particular resource, and did not inherit a default for that resource, the minimum and maximum tests for that resource are not applied to the job.

If you want jobs requesting only a specific value for a resource to be allowed into a queue, set the queue's resources_min.<resource name> and resources_max.<resource name> to the same value. This resource can be numeric, string, string array, or Boolean.

If you limit queue access using a string array, a job must request one of the values in the string array to be allowed into the queue. For example, if you set resources_min.strarr and resources_max.strarr to "blue,red,black", jobs can request `–l strarr=blue`, `-l strarr=red,` or `–l strarr=black` to be allowed into the queue.

2.3.6.4.ii How Queue and Server Running Time Limits are Applied

For shrink-to-fit jobs, running time limits are applied to max_walltime and min_walltime, not walltime. To set a running time limit for shrink-to-fit jobs, you cannot use resources_max or resources_min for max_walltime or min_walltime. Instead, use resources_max.walltime and resources_min.walltime. See section 4.9.42.6, "Shrink-to-fit Jobs and Resource Limits", on page 213.

2.3.6.4.iii Resources Used for Routing and Admittance

You can route jobs using the following kinds of resources:

- Any server-level or queue-level (job-wide) built-in or custom resource, whether it is numeric, string, or Boolean, for example ncpus and software

  When routing jobs with min_walltime and/or max_walltime, PBS examines the values for resources_min.walltime and resources_max.walltime at the server or queue. See section 2.3.6.4.ii, "How Queue and Server Running Time Limits are Applied", on page 27.
- The following built-in chunk-level resources:

  accelerator_memory

  mem

  mpiprocs

  naccelerators

  ncpus

  nodect

  vmem
- Custom vnode-level (chunk-level) resources that are global and have the n, q, or f flags set
- Any resource in the job's Resource_List attribute; see section 5.9.2, "Resources Requested by Job", on page 245. For string or string array resources, see section 2.3.6.4.iv, "Using String, String Array, and Boolean Values for Routing and Admittance", on page 27.

When jobs are routed using a chunk-level resource, routing is based on the sum of that resource across all chunks.

2.3.6.4.iv Using String, String Array, and Boolean Values for Routing and Admittance

When using strings or string arrays for routing or admittance, you can use only job-wide (server-level or queue-level) string or string array resources. String or string array resources in chunks are ignored. The resources_min and resources_max attributes work as expected with numeric values. In addition, they can be used with string and Boolean values to force an exact match; this is done by setting both to the same value. For example, to limit jobs entering queue big to those that specify `arch=unicos8`, or that do not specify a value for arch:

```
Qmgr: set q App1Queue resources_max.software=App1
Qmgr: set q App1Queue resources_min.software=App1
```

2.3.6.4.v Examples of Routing Jobs

You can force all jobs into a routing queue, or you can allow users to request some queues but not others. If you set up the default queue be a routing queue, and make all execution queues accept jobs only from routing queues, all jobs are initially forced into a routing queue.

Alternatively, you can set up one routing queue and a couple of execution queues which accept jobs only from routing queues, but add other queues which can be requested. Or you could allow jobs to request the execution queues, by making the execution queues also accept jobs that aren't from routing queues.

Example 2-1: Jobs can request one execution queue named *WorkQ*. All jobs that do not request a specific queue are routed according to their walltime:

- Create a routing queue *RouteQ* and make it the default queue:

```
Qmgr: create queue RouteQ queue_type = route
Qmgr: set server default_queue = RouteQ
```

- Create two execution queues, *LongQ* and *ShortQ*. One is for long-running jobs, and one is for short-running jobs:

```
Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution
```

- Set resources_min.walltime and resources_max.walltime on these queues:

```
Qmgr: set queue LongQ resources_min.walltime = 5:00:00
Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
```

- For *LongQ* and *ShortQ*, disallow jobs that are not from a route queue:

```
Qmgr: set queue LongQ from_route_only = True
Qmgr: set queue ShortQ from_route_only = True
```

- Set the destinations for *RouteQ* to be *LongQ* and *ShortQ*:

```
Qmgr: set queue RouteQ route_destinations = "ShortQ, LongQ"
```

- Create a work queue that can be requested:

```
Qmgr: create queue WorkQ queue_type = execution
```

- Enable and start all queues:

```
Qmgr: active queue RouteQ,LongQ,ShortQ,WorkQ
Qmgr: set queue enabled = True
Qmgr: set queue started = True
```

- Set default for walltime at the server so that jobs that don't request it inherit the default, and land in *ShortQ*:

```
Qmgr: set server resources_default.walltime = 4:00:00
```

Example 2-2: Jobs are not allowed to request any queues. All jobs are routed to one of three queues based on the job's walltime request:

- Create a routing queue *RouteQ* and make it the default queue:

```
Qmgr: create queue RouteQ queue_type = route
Qmgr: set server default_queue = RouteQ
```

- Create three execution queues, *LongQ*, *MedQ*, and *ShortQ*. One is for long-running jobs, one is for medium jobs, and one is for short-running jobs:

```
Qmgr: create queue LongQ queue_type = execution
Qmgr: create queue MedQ queue_type = execution
Qmgr: create queue ShortQ queue_type = execution
```

- Set resources_min.walltime and resources_max.walltime on these queues:

```
Qmgr: set queue LongQ resources_min.walltime = 10:00:00
Qmgr: set queue MedQ resources_max.walltime = 9:59:00
Qmgr: set queue MedQ resources_min.walltime = 5:00:00
Qmgr: set queue ShortQ resources_max.walltime = 4:59:00
```

- For *LongQ, MedQ,* and *ShortQ*, disallow jobs that are not from a route queue:

```
Qmgr: set queue LongQ from_route_only = True
Qmgr: set queue MedQ from_route_only = True
Qmgr: set queue ShortQ from_route_only = True
```

- Set the destinations for *RouteQ* to be *LongQ*, *MedQ* and *ShortQ*:

```
Qmgr: set queue RouteQ route_destinations = "ShortQ, MedQ, LongQ"
```

- Enable and start all queues:

```
Qmgr: active queue RouteQ,LongQ,ShortQ,MedQ
Qmgr: set queue enabled = True
Qmgr: set queue started = True
```

2.3.6.4.vi Caveats for Queue Resource Limits

If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has resources_max.<resource name> set, the job inherits that maximum value. If the queue has resources_max.<resource name> set, the job inherits the queue value, and if not, the job inherits the server value.

2.3.6.5 Using Access Control to Route Jobs

You can route jobs based on job ownership by setting access control limits at destination queues. A queue's access control limits specify which users or groups are allowed to have jobs in that queue. Default behavior is to disallow an entity that is not listed, so you need only list allowed entities.

To set the list of allowed users at a queue:

```
Qmgr: set queue <queue name> acl_users = "User1@*.example.com, User2@*.example.com"
```

To enable user access control at a queue:

```
Qmgr: set queue <queue name> acl_user_enable = True
```

To set the list of allowed groups at a queue:

```
Qmgr: set queue <queue name> acl_groups = "Group1, Group2"
```

To enable group access control at a queue:

```
Qmgr: set queue <queue name> acl_group_enable = True
```

For a complete explanation of access control, see section 8.3, “Using Access Control Lists”, on page 362.

2.3.6.6 Allowing Routing of Held or Waiting Jobs

By default, PBS will not route jobs that are held. You can allow a routing queue to route held jobs by setting the queue's route_held_jobs attribute to *True*:

```
Qmgr: set queue <queue name> route_held_jobs = True
```

By default, PBS will not route jobs whose execution_time attribute has a value in the future. You can allow a routing queue to route jobs whose start time is in the future by setting the queue's route_waiting_jobs attribute to *True*:

```
Qmgr: set queue <queue name> route_waiting_jobs = True
```

2.3.6.7 Setting Routing Retry Time

The default time between routing retries is 30 seconds. To set the time between routing retries, set the value of the queue's route_retry_time attribute:

```
Qmgr: set queue <queue name> route_retry_time = <value>
```

2.3.6.8 Specifying Job Lifetime in Routing Queue

By default, PBS allows a job to exist in a routing queue for an infinite amount of time. To change this, set the queue's route_lifetime attribute:

```
Qmgr: set queue <queue name> route_lifetime = <value>
```

2.3.7 Queue Requirements

- Each queue must have a unique name. The name must be alphanumeric, and must begin with an alphabetic character
- A server may have multiple queues of either or both types, but the server must have at least one execution queue defined.

2.3.8 Queue Configuration Attributes

Queue configuration attributes fall into three groups:

- Those which apply to both types of queues
- Those which apply only to execution queues
- Those which apply only to routing queues

If an "execution queue only" attribute is set for a routing queue, or vice versa, it is ignored. However, as this situation might indicate the administrator made a mistake, the server will write a warning message on `stderr` about the conflict. The same message is written when the queue type is changed and there are attributes that do not apply to the new type.

See "Queue Attributes" on page 311 of the PBS Professional Reference Guide for a table of queue attributes.

2.3.9 Viewing Queue Status

To see the status of a queue, including values for attributes, use the `qstat` command:

```
qstat -Qf <queue name>
```

To see the status of all queues:

```
qstat -Qf
```

The status of the queue is reported in the *State* field. The field shows two letters. One is either *E* (enabled) or *D* (disabled.) The other is *R* (running, same as started) or *S* (stopped.) Attributes with non-default values are displayed. See "qstat" on page 197 of the PBS Professional Reference Guide.

The following queue attributes contain queue status information:

total_jobs
state_count
resources_assigned
hasnodes
enabled
started

2.3.10 Deleting Queues

Use the qmgr command to delete queues.

```
Qmgr: delete queue <queue name>
```

2.3.10.1 Caveats for Deleting Queues

- A queue that has queued or running jobs cannot be deleted.
- The vnode queue attribute is **deprecated**. A queue that is associated with a vnode via that vnode's queue attribute cannot be deleted. To remove the association, save the output of pbsnodes -a to a file and search for the queue. Unset the queue attribute for each associated vnode.

2.3.11 Defining Queue Resources

For each queue, you can define the resources you want to have available at that queue. To set the value for an existing resource, use the qmgr command:

```
Qmgr: set queue <queue name> resources_available.<resource name> = <value>
```

For example, to set the value of the Boolean resource RunsMyApp to *True* at *QueueA*:

```
Qmgr: set queue QueueA resources_available.RunsMyApp = True
```

For information on how to define a new resource at a queue, see section 5.14, “Custom Resources”, on page 255.

For information on defining default resources at a queue, see section 5.9.3.3, “Specifying Job-wide Default Resources at Queue”, on page 246 and section 5.9.3.2.ii, “Specifying Chunk Default Resources at Queue”, on page 246.

2.3.12 Setting Queue Resource Defaults

The jobs that are placed in a queue inherit the queue's defaults for any resources not specified by the job's resource request. You can specify each default resource for each queue. This is described in section 5.9.3, “Specifying Job Default Resources”, on page 245. Jobs inherit default resources according to the rules described in section 5.9.4, “Allocating Default Resources to Jobs”, on page 247.

2.3.13 How Default Server and Queue Resources Are Applied When Jobs Move

When a job is moved from one server to another, the following changes happen:

- Any default resources that were applied by the first server are removed
- Default resources from the new server are applied to the job

When a job is moved from one queue to another, the following changes happen:

- Any default resources that were applied by the first queue are removed
- Default resources from the new queue are applied to the job

For more details on how default resources are inherited when a job is moved, see section 5.9.4.3, "Moving Jobs Between Queues or Servers Changes Defaults", on page 248.

2.3.14 Specifying Default Queue

PBS has a default execution queue named *workq*, where jobs are placed when no queue is requested. You can specify which queue should be the default. To specify the queue which is to accept jobs when no queue is requested, set the server's default_queue attribute to the name of the queue:

```
Qmgr: set server default_queue = <queue name>
```

2.3.15 Associating Queues and Vnodes

You can set up vnodes so that they accept jobs only from specific queues. See section 4.9.2, "Associating Vnodes with Queues", on page 103.

2.3.16 Configuring Access to Queues

You can configure each queue so that only specific users or groups can submit jobs to the queue. See section 8.3, "Using Access Control Lists", on page 362.

2.3.17 Setting Limits on Usage at Queues

You can set limits on different kinds of usage at each queue:

- You can limit the size of a job array using the max_array_size queue attribute
- You can limit the number of jobs or the usage of each resource by each user or group, or overall. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287

2.3.18 Queues and Failover

For information on configuring routing queues and failover, see section 9.2.6.1, "Configuring Failover to Work with Routing Queues", on page 415.

2.3.19 Additional Queue Information

For a description of each queue attribute, see "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

For information on using queues for scheduling, see section 4.6, "Using Queues in Scheduling", on page 98.

3 Configuring MoMs and Vnodes

3.1 About MoMs

A MoM runs and manages the jobs on each execution host. The pbs_mom daemon starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, manages job file transfer, and notifies the server when the job is finished. When the MoM starts a job, she creates a new session that is as identical to the user's login session as is possible. For example, under Linux, if the user's login shell is csh, then MoM creates a session in which .login and .cshrc are run. MoM returns the job's output to the user. The MoM performs any communication with job tasks and with other MoMs. The MoM on the first vnode on which a job is running manages communication with the MoMs on the remaining vnodes on which the job runs. The MoM on the first vnode is called the *primary execution host MoM*.

The MoM writes a log file in PBS_HOME/mom_logs. The MoM writes an error message in its log file when it encounters any error. The MoM also writes other miscellaneous information to its log file. If it cannot write to its log file, it writes to standard error.

You start a MoM via the pbs_mom command. The executable for pbs_mom is in PBS_EXEC/sbin, and can be run only by root. For Linux, see "MoMs: Starting, Stopping, Restarting" on page 166 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "MoMs: Starting, Stopping, Restarting" on page 172 in the PBS Professional Installation & Upgrade Guide.

The MoM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

PBS supplies a hook that you can use to manage cgroups on each execution host, and via the hook, cpusets. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571. If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

3.1.1 Configuring MoMs

3.1.1.1 MoM Configuration File

During the installation process, PBS creates a Version 1 configuration file for each MoM. Each parameter in this file controls some aspect of MoM's behavior. To configure MoM's behavior, edit this file, and set each parameter as desired.

The default location for the Version 1 configuration file is on MoM's host, in PBS_HOME/mom_priv/config, or if PBS_MOM_HOME is defined, PBS_MOM_HOME/mom_priv/config. It can be in a different location; in that case, MoM must be started with the -c option. See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

If you add or change anything via a Version 1 configuration file, you can HUP the MoM, but if you remove anything, you must restart the MoM so that the default value is re-applied.

The Version 1 configuration file must be secure. It must be owned by a user ID and group ID both less than 10 and must not be world-writable.

For a complete description of the syntax and contents of the Version 1 configuration file, see "MoM Parameters" on page 239 of the PBS Professional Reference Guide.

3.1.1.2 Editing Version 1 Files

Use your favorite text editor to edit Version 1 configuration files.

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.1.1.3 Caveats and Restrictions for Configuration Files

- The `pbs_mom -d` option changes where MoM looks for `PBS_HOME`, and using this option will change where MoM looks for all configuration files. If you use the `-d` option, MoM will look in the new location for all MoM and vnode configuration files. Instead, we recommend setting the location of `PBS_HOME` or `PBS_MOM_HOME` in `/etc/pbs.conf` on MoM's host.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.1.1.4 When MoM Reads Configuration Files

MoM reads `pbs.conf` at startup, and her own configuration files at startup and reinitialization. On Linux, this is when `pbs_mom` receives a `SIGHUP` signal or is started or restarted, and on Windows, when MoM is started or restarted.

If you make changes to the hardware or a change occurs in the number of CPUs or amount of memory that is available to PBS, such as a non-PBS process releasing a cpuset, you should restart PBS, by typing the following:

```
<path-to-script>/pbs restart
```

The MoM daemon is normally started by the PBS start/stop script.

When MoM is started, it opens its Version 1 configuration file, `mom_priv/config`, in the path specified in `pbs.conf`, if the file exists. If it does not, MoM will continue anyway. The `config` file may be placed elsewhere or given a different name, by starting `pbs_mom` using the `-c` option with the new file and path specified. See "MoMs: Starting, Stopping, Restarting" on page 172 in the PBS Professional Installation & Upgrade Guide.

The files are processed in this order:

1. Version 1 configuration file
2. PBS reserved configuration files
3. Version 2 configuration files

Within each category, the files are processed in lexicographic order.

The contents of a file that is read later will override the contents of a file that is read earlier.

If there is an error in `mom_priv/config`, MoM will not start.

3.1.2 Configuring MoM Polling Cycle

3.1.2.1 Cgroups Hook Can Replace Polling

The cgroups hook (see Chapter 16, "Configuring and Using PBS with Cgroups", on page 571) can provide accurate accounting information and job resource usage management, so that MoM does not need to perform periodic job resource usage polling. If you use the cgroups hook to manage jobs at a host, MoM does not need to poll throughout the life of the job, and the server and the datastore experience less traffic.

Each time a MoM polls, the server rewrites all of the job's data to the datastore, causing traffic to the data store. If you have smaller jobs, MoM needs to poll often in order to get reasonably accurate information. If you have many of these jobs, this slows the server and reduces throughput.

Each job is always polled at the start and end, regardless of periodic polling. MoM polls at job end, before running the epilogue, when she detects that the last job task is done. If the job was spawned with `tm_spawn`, MoM can get an accurate value for cput. If the job was `tm_attach`ed and the cgroups hook is not running on the host, she cannot get an accurate value for cput, because a process other than MoM reaped the job. For example, if memory is reaped by something other than MoM, there is no way to get usage. However, if the cgroups hook is managing that job, the hook can get accurate usage.

If the cgroups hook manages the jobs at a host, MoM does not need to do any periodic job polling at that host.

3.1.2.2 Polling on Linux

MoM's polling cycle is determined by the values of $min_check_poll and $max_check_poll in the Version 1 configuration file. The interval between each poll starts at $min_check_poll and increases with each cycle until it reaches $max_check_poll, after which it remains the same. The amount by which the cycle increases is the following:

(max_check_poll - min_check_poll + 19) / 20

The default value for $max_check_poll is 120 seconds. The minimum is 1 second.

The default value for $min_check_poll is 10 seconds. The minimum is 1 second.

The start of a new job resets the polling for all of the jobs being managed by this MoM.

MoM polls for resource usage for cput, walltime, mem and ncpus.

3.1.2.2.i Linux Polling Caveats

Please note that polling intervals cannot be considered to be exact:

- The polling calculation simply provides a minimum amount of time between one poll and the next.
- The actual time between polls can vary. The actual time taken by MoM also depends on the other tasks MoM is performing, such as starting jobs, running a prologue or epilogue, etc.
- The timing of MoM's activities is not completely under her control, because she is a user process.
- The finest granularity for calculating polling is in seconds.

3.1.2.3 Polling on Windows

On Windows, MoM updates job usage at fixed intervals of 10 seconds. The $min_check_poll and $max_check_poll parameters are not used by MoM on Windows. MoM looks for any job that has exceeded a limit for walltime, mem, or cput, and terminates jobs that have exceeded the limit.

3.1.2.4 How Polling is Used

- Job-wide limits are enforced by MoM using polling. See section 5.15.2.4.i, “Job Memory Limit Enforcement on Linux”, on page 306. MoM can enforce cpuaverage and cpuburst resource usage. See section 5.15.2.5.i, “Average CPU Usage Enforcement”, on page 307 and section 5.15.2.5.ii, “CPU Burst Usage Enforcement”, on page 308.
- MoM enforces the $restrict_user access restrictions on the polling cycle controlled by $min_check_poll and $max_check_poll. See section 15.4.7, “Restricting User Access to Execution Hosts”, on page 556.
- Cycle harvesting has its own polling interval. See “$kbd_idle <idle wait> <min use> <poll interval>” on page 243 of the PBS Professional Reference Guide for information on $kbd_idle.

3.1.2.5 Polling for Multi-host Jobs

Polling cycles are different on the primary execution host MoM and sister MoMs.

- The primary execution host MoM polls immediately when a task is started and again after the minimum polling period, then continues polling at each maximum polling period
- Sister MoMs poll a full cycle after the first task is created there

3.1.2.6 Recommendations for Polling Interval

Consider the workload at the host, and the overall workload at the server, when you set polling intervals. MoM's polling period should depend on the length of the typical job, and the importance for your site of accurate accounting. If you have many small jobs, frequent polling can take up a lot of MoM's cycles, and cause heavy traffic for the datastore and the server.

You may want to set $min_check_poll and $max_check_poll to somewhat higher values than the defaults. For example, for a 1-hour job, you could poll at 10-minute intervals. We do not recommend a value for $max_check_poll of less than 30 seconds. We do not recommend setting $min_check_poll to less than 10 seconds.

3.1.3 Files and Directories Used by MoM

If PBS_MOM_HOME is present in the `pbs.conf` file, `pbs_mom` will use that directory for its "home" instead of `PBS_HOME`.

3.1.3.1 Linux Files and Directories Used by MoM

Under Linux, all files and directories that MoM uses must be owned by root. MoM uses the following files and directories:

Table 3-1: MoM Files and Directories Under Linux

File/Directory	Description	Permissions
`/etc/pbs.conf`	File	*0644*
`aux`	Directory	*0755*
`checkpoint`	Directory	*0700*
checkpoint script	File	*0755*
`mom_logs`	Directory	*0755*
`mom_priv`	Directory	*0751*
`mom_priv/jobs`	Directory	*0751*
`mom_priv/config`	File	*0644*
`mom_priv/prologue`	File	*0755*
`mom_priv/epilogue`	File	*0755*
`pbs_environment`	File	*0644*
`spool`	Directory	*1777 (drwxrwxrwt)*
`undelivered`	Directory	*1777 (drwxrwxrwt)*

Table 3-1: MoM Files and Directories Under Linux

File/Directory	Description	Permissions
Version 2 configuration files (optional)	Files	*0755*
PBS reserved configuration files	Files	----
Job temporary directory	Directory	*1777*

3.1.3.2 Linux Files and Directories Used by MoM

Under Windows, these directories must have at least Full Control permission for the local Administrators group. MoM uses the following files and directories:

Table 3-2: MoM Files and Directories Under Windows

File/Directory	Description	Ownership/Permission
`pbs.conf`	File	
`auxiliary`	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
checkpoint	Directory	At least Full Control permission for the local Administrators group
checkpoint script	File	At least Full Control permission for the local Administrators group
`mom_logs`	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
`mom_priv`	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
`mom_priv/jobs`	Directory	At least Full Control permission for the local Administrators group and read-only access to Everyone
`mom_priv/config`	File	At least Full Control permission for the local Administrators group
`pbs_environment`	File	At least Full Control permission for the local Administrators group and read-only to Everyone
`spool`	Directory	Full access to Everyone
`undelivered`	Directory	Full access to Everyone
Job's temporary directory	Directory	Writable by Everyone

3.2 About Vnodes: Virtual Nodes

A virtual node, or *vnode*, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, a NUMA node, a nodeboard, or a blade. A single host can be represented by one vnode or multiple vnodes. PBS views hosts as being composed of one or more vnodes, and PBS can manage and schedule each vnode independently. One PBS MoM manages all of the vnodes for each host.

3.2.1 Parent Vnodes and Child Vnodes

Each machine is represented by at least one vnode. The main vnode is called the *parent vnode*. Vnodes that represent machine resources such as CPUs are called *child vnodes*.

For single-vnode machines, the parent vnode is also the child vnode, and this vnode represents all of the machine's resources, including its hardware.

For machines with more than one vnode, the parent vnode does not correspond to any actual hardware; instead, it is a collection of information that applies to the host but not the individual vnodes, such as dynamic host-level resources and shared resources. On multi-vnode machines, resources such as CPUs are represented in child vnodes.

3.3 Creating Vnodes

3.3.1 Overview of Creating Vnodes

1. For each machine, you create one parent vnode using `qmgr`. See "Creating the Parent Vnode" on page 40.

   For a single-vnode machine, vnode creation is done.

2. For a machine which will have more than one vnode, after you create the parent vnode, PBS handles creation of the child vnodes:

   - If you run the cgroups hook with vnode_per_numa_node set to *true*, the cgroups hook creates all the local child vnodes. We recommend using the cgroups hook for hosts where you need to fence jobs in or take advantage of the topology to keep job processes on nearby resources. See "Creating Child Vnodes via Cgroups Hook" on page 40.
   - If you are not using the cgroups hook to create child vnodes, you can have PBS create any child vnodes. You tell MoM which vnodes to create and how to set their attributes and resources by specifying them in a Version 2 configuration file. See "Creating Child Vnodes via Version 2 Configuration File" on page 40.

3. After all vnodes have been created, you can set vnode attributes and resources if necessary. See "Configuring Vnodes" on page 41.

3.3.1.1 Overview of Creating Vnodes on Cray XC

On the Cray XC, you create the parent vnode for each login node and MAMU node. After that, MoM creates any child vnodes.

3.3.2 How to Choose Vnode Names

MoM needs to know what name you will use for the parent vnode when she starts up. So if you decide to use a non-default name, define the name before starting MoM.

By default, the cgroups hook and MoM use the non-canonicalized hostname returned by `gethostname()` for the host as the vnode name. If you use the hostname, use the part before the first dot. You can use the `hostname()` command without any extra flags to get the hostname:

```
hostname<return>
```

For example, if this returns "myhost.mydomain", use "myhost".

You can choose the name for the parent vnode, such as an alias, or a name bound to another IP address on the host.

Except for Cray XC, you can use the IP address as the name of the parent vnode.

To use any parent vnode name that is not the default, you must specify the name by setting the PBS_MOM_NODE_NAME parameter in the host's /etc/pbs.conf. For example, if you use a name that has a dot in it and you don't set PBS_MOM_NODE_NAME, hooks will fail.

If you've already started MoM, then in order for MoM to be able to use the non-default name, you need to make the name available to her, then restart her. For example, to use the IP address:

1. Add PBS_MOM_NODE_NAME=<IP address> to pbs.conf on the execution host
2. Restart MoM

When PBS_MOM_NODE_NAME is defined, MoM performs a sanity check to ensure that the value is a resolvable host.

3.3.2.1 Names of Child Vnodes

If the cgroups hook creates child vnodes, it creates them with the same name as the parent vnode, with an index number. For example, on a machine with two NUMA nodes where each NUMA node is represented by a vnode, you'll have the parent vnode plus two child vnodes; if the parent vnode is named "myhost", they are named "myhost[0]" and "myhost[1]".

If you create child vnodes via a Version 2 configuration file, each vnode in your complex must have a unique name within the complex. We recommend using or at least including the name of the parent vnode, to ensure uniqueness and to make vnodes recognizable. Do not use square brackets for anything but the index.

3.3.2.2 Caveats for Vnode Names

- Do not change the name of the parent vnode after you create it
- If there is a dot in the name, you must set resources_available.host by hand; otherwise the part after the dot is stripped
- If you use an IP address as the name of a vnode, you must set it in PBS_MOM_NODE_NAME
- On Cray XC, you cannot use the IP address as the name of the vnode
- You cannot use a vnode attribute as the name of a vnode
- Vnode names are case-insensitive
- If you create a vnode with a different name from the short name returned by hostname, and you don't set it in PBS_MOM_NODE_NAME, the following happens:
  - MoM creates a vnode whose name is the short name returned by hostname()
  - The vnode you created is not recognized by MoM, and is marked stale

3.3.2.3 Errors and Logging for Vnode Names

- If PBS_MOM_NODE_NAME is unset and the call to gethostname() fails, or if PBS_MOM_NODE_NAME is set and the value does not conform to RFCs 952 and 1123, the following message is printed to the MoM log:
  Unable to obtain my host name
- Once the hostname is obtained, MoM ensures the hostname resolves properly. If the hostname fails to resolve, the following message is printed to the MoM log:
  Unable to resolve my host name

3.3.3 Creating the Parent Vnode

1. Make sure MoM can look up the name of the parent vnode when she starts. Follow the rules in section 3.3.2, “How to Choose Vnode Names”, on page 38. Choose the name for the parent vnode:
   - If you will use the default, make sure that PBS_MOM_NODE_NAME is not set, or set it to the default. To get the default name, run this at MoM's host, and use the part before the dot:

     hostname<return>
   - If you will use a non-default name, set it in PBS_MOM_NODE_NAME in /etc/pbs.conf on the MoM host. For example, to use the IP address for the name of the vnode, add this to /etc/pbs.conf on the execution host:

     PBS_MOM_NODE_NAME=<IP address>
2. Start MoM using systemd or the PBS start/stop script:

   systemctl start pbs

   or

   <path to script>/pbs start

   For details on starting and stopping MoM, see “Methods for Starting, Stopping, or Restarting PBS” on page 160 in the PBS Professional Installation & Upgrade Guide.
3. Use the qmgr command to create the parent vnode:

   qmgr -c 'create node <vnode name> [<attribute>=<value>]'

   All comma-separated attribute-value strings must be enclosed in quotes:

   qmgr -c 'create node <vnode name> ["<attribute>=<value>, <attribute>=<value>"]'

   Attributes and their possible values are listed in “Vnode Attributes” on page 320 of the PBS Professional Reference Guide.

3.3.4 Creating Child Vnodes for Multi-vnode Machines

3.3.4.1 Creating Child Vnodes via Cgroups Hook

If you are running the cgroups hook with vnode_per_numa_node set to *true*, the hook creates the local child vnodes.

1. If you have not done so yet, create the parent vnode; see Chapter 3, "Creating the Parent Vnode", on page 40.
2. Configure and enable the cgroups hook. Follow all the instructions in Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.
3. Restart the MoM:

   <path to PBS start/stop script>/pbs restart

   or

   systemctl restart pbs

3.3.4.2 Creating Child Vnodes via Version 2 Configuration File

1. If you have not done so yet, create the parent vnode; see Chapter 3, "Creating the Parent Vnode", on page 40.
2. If you are not using the cgroups hook, you can create any child vnodes by defining the child vnodes you want in a Version 2 configuration file, so MoM creates the vnodes for you.

Note that in prior versions of PBS, a Version 2 configuration file was the preferred method for advanced GPU configuration (see "Advanced GPU Scheduling" on page 284). As of version 2020.1, the cgroups hook makes it much easier for job submitters to request exclusive use of GPUs. However, if you want to continue to use Version 2 configuration files for managing GPUs, you can do so.

See section 3.4.3.1, "Creating Version 2 Configuration Files", on page 43.

3. Restart the MoM:

```
<path to PBS start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

4. Check for stale vnodes. Make sure you spell "Stale" with a capital S:

```
qmgr -c 'print node @default' | grep "Stale"
```

3.3.5 Caveats for Creating Vnodes

When using qmgr to create vnodes, create only the parent vnode on each host. Do not use qmgr to create child vnodes on a multi-vnode host; MoM will not know about these, and cannot use them.

3.4 Configuring Vnodes

Each vnode has an associated set of attributes and resources, such as CPUs, memory, and partition. Vnode attributes are listed and described in "Vnode Attributes" on page 320 of the PBS Professional Reference Guide. Vnode resources can be built-in or custom (defined by you.) See Chapter 5, "Using PBS Resources", on page 229.

3.4.1 Methods for Configuring Vnodes

You may need to configure vnodes after you create them. You can use the following methods:

- Using exechost_startup hooks to set vnode attributes and resources

  This method is powerful and flexible. You can interrogate the host; for example, you can check whether a vnode exists before setting values for it. You can use this to set the sharing attribute and resources_available.host. If the cgroups hook creates your vnodes, make sure that the cgroups hook runs before the hook that configures the vnodes. Your exechost_startup hooks run when MoM is restarted. See "Setting and Unsetting Vnode Resources and Attributes" on page 48 in the PBS Professional Hooks Guide.

- Using Version 2 vnode configuration files, either to modify vnodes created by the cgroups hook, or to tell MoM to create the vnodes you specify. See section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.

  Make sure that a Version 2 configuration file matches your available vnodes every time MoM is started. If your machine reboots with a missing blade, your earlier placement set information will not make sense because child vnode names will not match the available hardware. You can use a script to regenerate this file each time the machine starts, and run the script before MoM is restarted.

  You can use a Version 2 configuration file to set the sharing attribute and the value of resources_available.host (you cannot set these via qmgr).

  An advantage of using a Version 2 configuration file is that if you delete and re-create the parent vnode, you don't have to re-create this file. MoM automatically picks up everything in a Version 2 configuration file, whereas if you use qmgr you have to re-run all your configuration commands.

If you use the cgroups hook to create child vnodes, and you want to modify these child vnodes, make sure you create the Version 2 configuration file after the vnodes are created, and that you use the exact vnode names that the cgroups hook knows about, by checking the output of `pbsnodes -av`.

If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.

Version 2 configuration files are read when MoM is restarted. See section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.

- Using the `qmgr` command to set vnode attribute and resource values

  You can easily use `qmgr` to set values across your complex. You cannot use this to set the sharing attribute or resources_available.host. If you delete and re-create a vnode, the effects of your configuration commands are lost. Changes take place immediately. See "qmgr" on page 149 of the PBS Professional Reference Guide.

- Using the `pbsnodes -o` or `pbsnodes -r` command to mark all vnodes on a host as offline or not offline

  You must use `qmgr` to change the state of a single vnode in a multi-vnode host. Changes take place immediately. See "pbsnodes" on page 35 of the PBS Professional Reference Guide.

3.4.2 Rules for Configuring Vnodes

- If you are using the cgroups hook to create child vnodes and manage subsystems, do not change attribute or resource values that are set by the cgroups hook.
- To set the sharing attribute or resources_available.host, you must use an exechost_startup hook or a Version 2 configuration file. You cannot use `qmgr`. See section 3.4.4, "Configuring the Vnode Sharing Attribute", on page 46.
- Set the Mom attribute for the parent vnode only. You can set the initial value only via `qmgr -c 'create node <vnode name>'` to tell the server at what IP address MoM is located. The server will later update it based on the MoM's response. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation.

3.4.2.1 Rules for Configuring Vnodes on Cray XC

- On Cray XC, we recommend using an exechost_startup hook instead of a Version 2 configuration file; the inventory hook will detect any inconsistency and send a HUP to MoM.
- On Cray XC compute nodes, never change the sharing attribute.
- We do not recommend using Version 2 configuration files on Cray XC. A Version 2 configuration file on ALPS can interfere with the mechanism whereby ALPS updates PBS. For example, changes to node state will make vnodes stale. On Cray XC machines, compute nodes are not hosts; they are only vnodes, which means that they become *stale* and not *down* when ALPS marks them no longer available to PBS. Mentioning such stale nodes in Version 2 configuration files would interfere with this mechanism.

3.4.3 Version 2 Vnode Configuration Files

Version 2 configuration files contain settings for vnode attributes and resources. For example, to change the sharing vnode attribute or resources_available.host, you can use a Version 2 configuration file, or an exechost_startup hook, but not `qmgr`. You can use more than one Version 2 configuration file per host, but make sure they do not conflict.

PBS places Version 2 configuration files in an area that is private to each installed instance of PBS.

It's best to automate updates to Version 2 configuration files so that they are created at boot time to match available hardware, because a change in hardware may create a mismatch with an old Version 2 configuration file. This ensures that a Version 2 configuration file matches your available hardware every time MoM is started. If your machine reboots with a missing blade, your earlier placement set information will not make sense because child vnode names will be not match the available hardware. You can use a script to regenerate this file each time the machine starts, and run the script before MoM is restarted.

An advantage of using a Version 2 configuration file is that if you delete and re-create the parent vnode, you don't have to re-create this file. MoM automatically picks up everything in a Version 2 configuration file, whereas if you use `qmgr` you have to rerun all your configuration commands.

3.4.3.1 Creating Version 2 Configuration Files

Version 2 configuration files are created by PBS, through a process where you write a source file and then PBS copies it to the location where Version 2 files are used. Instead of editing one of these directly, you create an input file and give it as an argument to the `pbs_mom -s insert` option on the local host (`pbs_mom -N -s insert` on Windows), and PBS creates a new configuration file for you.

You use the `pbs_mom -s insert` command to create Version 2 configuration files. On Windows, use the `pbs_mom` command in standalone mode: `pbs_mom -N -s insert.`

First, you create an input file which is to be the contents of the configuration file. Then, you use the `pbs_mom -s insert` command, on the host you want to configure:

Linux:

pbs_mom -s insert <Version 2 configuration file> <input file name>

Windows:

pbs_mom -N -s insert <Version 2 configuration file> <input file name>

After you create the new Version 2 configuration file, restart the MoM.

3.4.3.1.i Syntax of Version 2 Configuration Files

In a Version 2 configuration file, you tell PBS that it's a Version 2 MoM configuration file by putting a special tag on the first line:

```
$configversion 2
```

The rest of the file describes vnodes, with one attribute specification per line.

The format of the remaining contents of the file is the following:

<vnode name> : <attribute name> = <attribute value>

where

<vnode name>

Sequence of characters not including a colon (":"). The vnode name must be unique in this PBS complex. Vnode names are case-insensitive. See "Vnode Name" on page 358 of the PBS Professional Reference Guide.

If you're modifying vnodes created by the cgroups hook, the *vnode name* must exactly match the output of `pbsnodes -av`.

<attribute name>

Name of the attribute being specified. See "Attribute Name" on page 353 of the PBS Professional Reference Guide.

<attribute value>

Value being specified. Sequence of characters not including an equal sign ("="). See "Resource Formats" on page 359 of the PBS Professional Reference Guide.

White space around the colon and equal sign is ignored.

In a Version 2 configuration file, do not use quotes around string array values. This is different from using the qmgr command; in the qmgr command line, you need to put quotes around the value.

Make sure that the first vnode entry is for the parent vnode. If you don't need to set anything, you can set the ntype attribute to "PBS" (the default).

Make sure that there is a newline at the end of the file. Under Windows, the Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

Make sure that entries do not conflict, whether within one file or multiple files.

Do not use Version 1 (MoM configuration file) syntax or contents in Version 2 files, and vice versa.

3.4.3.1.ii Example of Creating Version 2 Configuration File

Example 3-1: If your machine named "myhost" has 4 vnodes, where two are big (myhost[0] and myhost[1]), and two are small (myhost[2] and myhost[3]), and you want big jobs to have exclusive use of myhost[0] and myhost[1], and small jobs to share myhost[2] and myhost[3]:

a. Set sharing for big and small vnodes by creating a file "set_sharing" containing the following:

```
$configversion 2
myhost: ntype = PBS
myhost[0]: sharing = default_excl
myhost[0]: resources_available.nodetype=big
myhost[1]: sharing = default_excl
myhost[1]: resources_available.nodetype=big
myhost[2]: sharing = default_shared
myhost[2]: resources_available.nodetype=small
myhost[3]: sharing = default_shared
myhost[3]: resources_available.nodetype=small
```

b. Use the pbs_mom -s insert <filename> <script> option at myhost to create its configuration file:

Linux:

```
pbs_mom -s insert sharing_config set_sharing
```

Windows:

```
pbs_mom -N -s insert sharing_config set_sharing
```

PBS creates the new Version 2 configuration file called "sharing_config". Its contents will override previously-read sharing settings.

c. Restart the MoM after changing the configuration file:

Linux:

```
kill -INT <MoM PID>
PBS_EXEC/sbin/pbs_mom
```

or

```
systemctl restart pbs
```

or

```
<path to start/stop script>/pbs restart
```

Windows:

```
net stop pbs_mom
net start pbs_mom
```

Jobs can then request nodetype = big or nodetype=small, or you can use a hook to route jobs, etc.

3.4.3.2 Listing and Viewing Version 2 Configuration Files

You can list and view the Version 2 configuration files at each host.

To see the list of Version 2 configuration files:

Linux:

```
pbs_mom -s list
```

Windows:

```
pbs_mom -N -s list
```

To display the contents of a Version 2 configuration file:

Linux:

```
pbs_mom -s show <filename>
```

Windows:

```
pbs_mom -N -s show <filename>
```

See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

3.4.3.3 Moving Version 2 Configuration Files

To move a set of Version 2 configuration files from one MoM host to another:

1. List the Version 2 files at the source instance:
   ```
   pbs_mom -s list
   ```
2. Save a copy of each file at the source instance:
   ```
   pbs_mom -s show <V2 filename> > <new input file>
   ```
3. Create the new Version 2 configuration files at the destination host. For each file:
   ```
   pbs_mom -s insert <Version 2 file> <new input file>
   ```

3.4.3.4 Removing Version 2 Configuration Files

You can remove a Version 2 configuration file:

Linux:

```
pbs_mom -s remove <filename>
```

Windows:

```
pbs_mom -N -s remove <filename>
```

See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

3.4.3.5 Caveats for Version 2 Configuration Files

- If you are using the cgroups hook to create child vnodes at a host, and you use a Version 2 configuration file to modify those child vnodes:
  - Make sure that you use exactly the same vnode names in the Version 2 configuration file as those that the cgroups hook has created; check the output of `pbsnodes -av`.
  - Do not use a Version 2 configuration file to change hardware settings for that host.
- If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.
- The `pbs_mom -d` option changes where MoM looks for `PBS_HOME`, and using this option will change where MoM looks for all configuration files. If you use the `-d` option, MoM will look in the new location for all MoM and vnode configuration files. Instead, we recommend setting the location of `PBS_HOME` or `PBS_MOM_HOME` in `/etc/pbs.conf` on MoM's host.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.4.3.6 PBS Reserved Configuration Files

PBS reserved configuration files are created by PBS and are prefixed with "PBS". You cannot create or modify a configuration file whose name begins with "PBS". Do not move PBS reserved configuration files.

3.4.4 Configuring the Vnode Sharing Attribute

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode's sharing attribute and the placement requested by a job.

You can set each vnode's sharing attribute so that the vnode or host is always shared, is always exclusive, or so that it honors the job's placement request. If the vnode attribute is set to *force_shared* or *force_excl*, the value of a vnode's sharing attribute takes precedence over a job's placement request. If the vnode attribute is set to *default_*, the job request overrides the vnode attribute.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs. If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.

For a complete description of the sharing attribute, and a table showing the interaction between the value of the sharing attribute and the job's placement request, see "sharing" on page 324 of the PBS Professional Reference Guide.

3.4.4.1 Sharing on a Multi-vnode Machine

On a multi-vnode shared-memory machine, a scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

To keep a job in a single vnode, use `-lplace=group=vnode`; if you want to restrict it to larger sets of vnodes, identify those sets using a custom string or string_array resource and use it in `-lplace=group=<resource>`. If you already have resources used in node_group_key you can usually use these.

3.4.4.2 Setting the sharing Vnode Attribute

To set the sharing attribute for a vnode, use either:

- An exechost_startup hook; see "Setting and Unsetting Vnode Resources and Attributes" on page 48 in the PBS Professional Hooks Guide
- A Version 2 configuration file; see section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 46

3.4.4.3 Viewing Sharing Information

You can use the `qmgr` or `pbsnodes` commands to view sharing information. See “qmgr” on page 149 of the PBS Professional Reference Guide and “pbsnodes” on page 35 of the PBS Professional Reference Guide.

3.4.4.4 Sharing Caveats

- On the Cray XC, on *cray_compute* vnodes, the sharing attribute is set to *force_exclhost* by default. Do not change this setting, because ALPS does not support sharing a compute vnode with more than one job.
- The term "sharing" is also used to describe the case where MoM manages a resource that is shared among her vnodes, for example an application license shared by the vnodes of a multi-vnode machine.
- The term "sharing" is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are "sharing" a CPU. See section 9.6.5, “Managing Load Levels on Vnodes”, on page 445
- If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.
- For vnodes with sharing=*default_shared*, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The exec_vnode attribute will show this allocation.

3.4.5 Configuring Vnode Resources

Before configuring vnode (host-level) resources, consider how you will use them. When configuring static resources, it is best to configure global static resources. Even though they are global, they can be configured at the host level. Global resources can be operated on via the `qmgr` command and viewed via the `qstat` command. When configuring dynamic resources, if you need the script to run at the execution host, configure local dynamic resources. These resources cannot be operated on via the `qmgr` command or viewed via the `qstat` command. See section 5.4, “Categories of Resources”, on page 233.

3.4.5.1 Configuring Global Static Vnode Resources

You can create global custom static host-level resources that can be reported by MoM and used for jobs. Follow the instructions in section 5.14.4.2, “Static Host-level Resources”, on page 270.

You can set values for built-in and custom global static vnode resources; see section 3.4.5, “Configuring Vnode Resources”, on page 47.

3.4.5.2 Configuring Local Dynamic Vnode Resources

You can create local custom dynamic host-level resources. The primary use of this feature is to add site-specific resources, such as software application licenses or scratch space. Follow the instructions in section 5.14.4.1, “Dynamic Host-level Resources”, on page 269.

3.4.5.3 Rules for Configuring Vnode Resources

- In general, it is not advisable to set resources_available.ncpus or resources_available.mem to a value greater than PBS has detected on the machine. This is because you do not want MoM to try to allocate more resources than are available. However, if you have lots of I/O-bound jobs, you might get away with oversubscribing CPUs.
- In general, it is safe to set resources_available.ncpus or resources_available.mem to a value less than PBS has detected. If you are using a Version 2 configuration file, consider setting ncpus lower to set aside some of the resource for the operating system.
- For the parent vnode on a multi-vnode machine, set all values for resources_available.<resource name> to *zero* (*0*), unless the resource is being shared among child vnodes via indirection. Here is an example of the vnode definition for a parent vnode:

```
host03: pnames = cbrick, router
host03: sharing = ignore_excl
host03: resources_available.ncpus = 0
host03: resources_available.mem = 0
host03: resources_available.vmem = 0
```

- When MoM creates a vnode, she automatically sets values for the following resources according to information from the host:

  resources_available.ncpus

  resources_available.arch

  resources_available.mem

- If you set the value of a resource via qmgr, that setting is carried forth across server restarts.
- If you add or change a value via a Version 2 configuration file, you can HUP the MoM. If you remove a value, you must restart MoM so that she uses the default. (Hint: to avoid restarting MoM, use the configuration file to set the default.)
- You can set values for the sharing attribute and resources_available.host only in an exechost_startup or exechost_periodic hook, or in a Version 2 configuration file. You cannot use qmgr to set these.
- Version 2 configuration files take effect before exechost_startup hooks.

3.4.6 Configuring Vnodes via the qmgr Command

You can use the qmgr command to set attribute and resource values for individual vnodes, for single-vnode and multi-vnode machines.

To set a vnode's attribute:

qmgr -c 'set node <vnode name> <attribute> = <value>'

We describe the qmgr command in "qmgr" on page 149 of the PBS Professional Reference Guide.

3.4.6.1 Caveats for Setting Values via qmgr Command

- When setting hardware resources, be careful about setting these to values that are higher than what MoM or the cgroups hook did. If you have lots of I/O-bound jobs, you might get away with oversubscribing CPUs.
- It is usually safe to set hardware resources to values lower than what MoM or the cgroups hook did.
- If you are not using the cgroups hook, consider setting ncpus to a slightly lower value than what MoM reports, to give some to the operating system.
- If you set a value using `qmgr`, this value overrides the existing value, and you cannot change the value using another method, such as a Version 2 configuration file. If you want to use a different method to set a value that has been set via `qmgr`, use `qmgr` to unset the value, then HUP the MoM.
- You cannot set the value of resources_available.host via the `qmgr` command.

3.4.7 Configuring Vnodes via the `pbsnodes` Command

You can use the `pbsnodes` command to set the state all of the vnodes on a host to be *offline* or not *offline*. To set the state attribute of one or more hosts to *offline*:

```
pbsnodes -o <hostname [hostname ...]>
```

To remove the *offline* setting from the state attribute of one or more hosts:

```
pbsnodes -r <hostname [hostname ...]>
```

See "pbsnodes" on page 35 of the PBS Professional Reference Guide.

3.4.7.1 Caveats for pbsnodes Command

For multi-vnode hosts, the `pbsnodes` command operates on all of the host's vnodes only. You cannot use it on individual vnodes where those vnodes are on multi-vnode machines. To operate on individual vnodes, use the `qmgr` command:

```
qmgr -c 'set node <vnode name> state = <new state>'
```

When you specify a hostname, the `pbsnodes` command looks for the value of a vnode's resources_available.host resource. If this is different from the PBS_MOM_NODE_NAME parameter, it may be helpful to use a Version 2 configuration file to set resources_available.host to match the PBS_MOM_NODE_NAME parameter (you cannot use `qmgr` for this).

Make sure that resources_available.host is unique for each host in your complex.

The `pbsnodes -o <target host>` command offlines everything with a matching resources_available.<target host>.

3.5 Deleting Vnodes

3.5.1 Deleting the Vnode on a Single-vnode Machine

Use the `qmgr` command to delete the vnode:

```
Qmgr: delete node <vnode name>
```

3.5.2 Deleting Vnodes on a Multi-vnode Machine

3.5.2.1 Deleting Vnodes When Not Using Version 2 Configuration File

4. Use the qmgr command to delete the vnodes:

   Qmgr: delete node <vnode name>

3.5.2.2 Deleting Vnodes When Using Version 2 Configuration File

To delete one or more vnodes on a multi-vnode machine where there is a Version 2 configuration file, you must first remove the configuration file. Then you can delete the vnodes. You may want to save the existing configuration file and edit it down to just the vnodes you want to preserve. On the local host:

1. To see the list of Version 2 configuration files:

   Linux:

   pbs_mom -s list

   Windows:

   pbs_mom -N -s list

2. To save the contents of a Version 2 configuration file in "tempconfig":

   Linux:

   pbs_mom -s show <filename> > tempconfig

   Windows:

   pbs_mom -N -s show <filename> > tempconfig

3. Edit tempconfig so that it describes only the vnodes you want to keep.

4. Use pbs_mom -s remove to remove the old Version 2 configuration file:

   On Linux:

   pbs_mom -s remove <filename>

   On Windows:

   pbs_mom -N -s remove <filename>

5. Use pbs_mom -s insert to create a new Version 2 configuration file describing the vnodes to be retained. If you created tempconfig, it is your input file:

   On Linux:

   pbs_mom -s insert <configuration file target> <input file>

   On Windows:

   pbs_mom -N -s insert <configuration file target> <input file>

6. Restart the MoM:

   <path to start/stop script>/pbs restart

or

```
systemctl restart pbs
```

7. Use the qmgr command to remove the vnodes no longer appearing in your configuration file:

```
Qmgr: delete node <vnode name>
```

8. Check for stale vnodes. Make sure you spell "Stale" with a capital S:

```
qmgr -c 'print node @default' | grep "Stale"
```

3.5.2.3 Deleting Vnodes on Cray XC

See section 11.16.8, “Deleting Vnodes on Cray XC”, on page 494.

4 Scheduling

The "Scheduling Policy Basics" section of this chapter describes what PBS can do, so that you can consider these capabilities when choosing how to schedule jobs. The "Choosing a Policy" section describes how PBS can meet the scheduling needs of various workloads. The "Scheduling Tools" section describes each scheduling tool offered by PBS.

4.1 Chapter Contents

4.1 Chapter Contents . . . 53
4.2 Scheduling Each Partition Separately . . . 55
4.2.1 Creating and Configuring a Multisched . . . 55
4.2.2 Starting a Multisched . . . 57
4.2.3 Configuring Your Partitions for Multischeds . . . 57
4.2.4 Using the Default Scheduler with Multischeds . . . 58
4.2.5 Multisched Caveats and Restrictions . . . 58
4.2.6 Attributes Used with Multischeds . . . 58
4.2.7 Multisched Errors and Logging . . . 60
4.2.8 Multisched Deprecations . . . 62
4.3 Scheduling Policy Basics . . . 62
4.3.1 How Scheduling Can Be Used . . . 62
4.3.2 What Is Scheduling Policy? . . . 63
4.3.3 Basic PBS Scheduling Behavior . . . 63
4.3.4 Sub-goals . . . 63
4.3.5 Job Prioritization and Preemption . . . 64
4.3.6 Resource Allocation to Users, Projects & Groups . . . 70
4.3.7 Time Slot Allocation . . . 72
4.3.8 Job Placement Optimization . . . 73
4.3.9 Resource Efficiency Optimizations . . . 76
4.3.10 Overrides . . . 78
4.4 Choosing a Policy . . . 79
4.4.1 Overview of Kinds of Policies . . . 79
4.4.2 FIFO: Submission Order . . . 79
4.4.3 Prioritizing Jobs by User, Project or Group . . . 80
4.4.4 Allocating Resources by User, Project or Group . . . 80
4.4.5 Scheduling Jobs According to Size Etc. . . . 82
4.4.6 Scheduling Jobs into Time Slots . . . 84
4.4.7 Default Scheduling Policy . . . 86
4.4.8 Examples of Workload and Policy . . . 88
4.5 About Schedulers . . . 89
4.5.1 Configuring a Scheduler . . . 89
4.5.2 Making a Scheduler Read its Configuration . . . 95
4.5.3 Scheduling on Resources . . . 95
4.5.4 Starting, Stopping, and Restarting a Scheduler . . . 95
4.5.5 The Scheduling Cycle . . . 96
4.5.6 How Available Consumable Resources are Counted . . . 97

4.5.7 Improving Scheduler Performance . . . 97
4.6 Using Queues in Scheduling . . . 98
4.7 Scheduling Restrictions and Caveats . . . 99
4.7.1 One Policy Per Scheduler . . . 99
4.7.2 Jobs that Cannot Run on Current Resources . . . 99
4.7.3 Resources Not Controlled by PBS . . . 99
4.7.4 No Pinning of Processes to Cores . . . 100
4.8 Errors and Logging . . . 100
4.8.1 Logfile for scheduler . . . 100
4.9 Scheduling Tools . . . 100
4.9.1 Anti-Express Queues . . . 102
4.9.2 Associating Vnodes with Queues . . . 103
4.9.3 Using Backfilling . . . 105
4.9.4 Examining Jobs Queue by Queue . . . 110
4.9.5 Checkpointing . . . 111
4.9.6 Organizing Job Chunks . . . 112
4.9.7 `cron` Jobs . . . 112
4.9.8 Using Custom and Default Resources . . . 113
4.9.9 Using Idle Workstation Cycle Harvesting . . . 115
4.9.10 Dedicated Time . . . 125
4.9.11 Dependencies . . . 126
4.9.12 Dynamic Resources . . . 126
4.9.13 Eligible Wait Time for Jobs . . . 126
4.9.14 Sorting Jobs by Entity Shares (Was Strict Priority) . . . 131
4.9.15 Estimating Job Start Time . . . 131
4.9.16 Calculating Job Execution Priority . . . 134
4.9.17 Calendaring Jobs . . . 137
4.9.18 Express Queues . . . 137
4.9.19 Using Fairshare . . . 138
4.9.20 FIFO Scheduling . . . 148
4.9.21 Using a Formula for Computing Job Execution Priority . . . 149
4.9.22 Gating Jobs at Server or Queue . . . 155
4.9.23 Managing Application Licenses . . . 155
4.9.24 Limits on Per-job Resource Usage . . . 156
4.9.25 Limits on Project, User, and Group Jobs . . . 156
4.9.26 Limits on Project, User, and Group Resource Usage . . . 156
4.9.27 Using Load Balancing . . . 156
4.9.28 Matching Jobs to Resources . . . 159
4.9.29 Node Grouping . . . 162
4.9.30 Overrides . . . 162
4.9.31 Peer Scheduling . . . 165
4.9.32 Placement Sets . . . 168
4.9.33 Using Preemption . . . 180
4.9.34 Using Primetime and Holidays . . . 191
4.9.35 Provisioning . . . 196
4.9.36 Queue Priority . . . 196
4.9.37 Reservations . . . 197
4.9.38 Round Robin Queue Selection . . . 204
4.9.39 Routing Jobs . . . 205
4.9.40 Scheduler Cycle Speedup . . . 209
4.9.41 Shared vs. Exclusive Use of Resources by Jobs . . . 210
4.9.42 Using Shrink-to-fit Jobs . . . 211
4.9.43 SMP Cluster Distribution . . . 218
4.9.44 Using Soft Walltime . . . 219

4.9.45 Sorting Jobs on a Key . . . 221
4.9.46 Sorting Jobs by Requested Priority . . . 223
4.9.47 Sorting Queues into Priority Order . . . 223
4.9.48 Starving Jobs . . . 223
4.9.49 Using Strict Ordering . . . 225
4.9.50 Sorting Vnodes on a Key . . . 226

4.2 Scheduling Each Partition Separately

You can leave your complex as a single default partition, or you can divide your complex into partitions, and run a separate scheduler for each partition. PBS automatically creates a default partition containing all queues and vnodes that are not explicitly labeled with a partition name. You can create named partitions, simply by labeling each queue and vnode with the desired partition. You can choose whether to assign each queue and vnode to a specific partition, or to have it remain as part of the default partition.

PBS has two kinds of schedulers:

- A default scheduler that handles the workload for the default partition (all queues and vnodes that have not been explicitly assigned to a named partition)

  The default scheduler runs its own scheduling policy.

  You cannot assign any non-default partitions to the default scheduler.

  The default scheduler runs only on the server host.
- A *multisched* that handles a named, non-default partition

  Each multisched runs its own scheduling policy, and can schedule jobs for one named partition.

  A multisched requires at least one queue and one vnode in its partition in order to be able to schedule jobs.

  Multischeds cannot share partitions.

  A multisched can run on any host.

A named partition is a collection of vnodes labeled with a partition name, along with one or more queues also labeled with the same partition name. A vnode can be in at most one partition. You can put some or all of your vnodes into partitions, where they will be scheduled by the multisched assigned to the partition. You can also leave vnodes out of named partitions; those vnodes will be scheduled by the default scheduler.

You can have as many named partitions and multischeds as you want. Each partition can have only one multisched. Each partition requires at least one execution queue.

Each multisched schedules only from the queue(s) in its partition, and only to the vnode(s) in its partition. Jobs do not span partitions.

You can define a unique policy for each scheduler.

4.2.1 Creating and Configuring a Multisched

4.2.1.1 Prerequisites for Creating a Multisched

You must be a PBS administrator or Manager to create a scheduler.

You must supply a name when you create a multisched. The maximum length for the name is 15 characters.

Before you start a multisched, you must create the sched_priv and sched_log directories for it.

- The default name for the sched_priv directory is sched_priv_<multisched name>, and the default location is on the server/scheduler host, directly under PBS_HOME, alongside the sched_priv of the default scheduler. You can set the name and location as desired, but we recommend keeping it in PBS_HOME. The sched_priv directory should have permissions 750, and should be accessible by the multisched. It should be owned by root. It cannot be shared with another multisched.
- Populate the sched_priv directory with the following:

  sched_config

  Required.

  holidays

  Required.

  resource_group

  Necessary for fairshare tree.

  dedicated_time

  Required.

  We provide default copies of these files in PBS_EXEC/etc.
- The default name and location for the multisched logging directory is sched_logs_<multisched name> (note the plural), on the server/scheduler host, directly under PBS_HOME, alongside the sched_logs of the default scheduler. You can set the name and location as desired, but we recommend keeping it in PBS_HOME. The sched_log directory should have permissions 755, and should be accessible by the multisched. It should be owned by root. It cannot be shared with another multisched.

4.2.1.2 Creating a Multisched

You use the qmgr command to create a scheduler:

qmgr -c "create sched <multisched name>"

For example:

```
qmgr -c "create sched multisched_1"
```

This creates a multisched with its attributes set to the defaults.

4.2.1.3 Configuring a Multisched

You must set the partition and sched_port multisched attributes:

qmgr -c "set sched <multisched name> partition = <partition name>"

qmgr -c "set sched <multisched name> sched_port = <port number>"

4.2.1.4 Enabling a Multisched

To enable a multisched, set its scheduling attribute to *True*:

```
qmgr -c "set sched <scheduler name> scheduling = 1"
```

4.2.2 Starting a Multisched

Do not start a multisched until:

- Its sched_priv directory is ready. See section 4.2.1.1, “Prerequisites for Creating a Multisched”, on page 55
- You have assigned it a partition. See section 4.2.1.3, “Configuring a Multisched”, on page 56

4.2.2.1 Starting a Multisched on Linux

Start a multisched by calling pbs_sched and specifying the name and port you already gave it:

pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

For example:

```
pbs_sched -I multisched_1 -S 15050
```

When you start a multisched, you must specify its name.

4.2.3 Configuring Your Partitions for Multischeds

To schedule using partitions, compose each partition and start its multisched:

- Put each vnode into at most one partition, or leave it in the default partition; partitions cannot share vnodes. If putting the vnode into a named partition, set the value of the partition vnode attribute to the name of its partition:

  qmgr -c set node <vnode name> partition=<partition name>

  For example:

  ```
  qmgr -c set node <vnode1> partition=<part1>
  ```

- Assign at least one execution queue to each named partition: set the value of the partition queue attribute to the name of its partition:

  qmgr -c set queue <queue name> partition=<partition name>

  For example:

  ```
  qmgr -c set queue <queue1> partition=<part1>
  ```

- Create the desired multisched. See section 4.2.1, “Creating and Configuring a Multisched”, on page 55.
- Assign a multisched to each named partition: set the value of the partition multisched attribute to the name of its partition:

  qmgr -c "set sched <multisched name> partition=<partition name>"

  For example:

  ```
  qmgr -c "set sched multisched_1 partition=part1"
  ```

- Enable the multisched:

  qmgr -c 'set sched <multisched name> scheduling=1'

  For example:

  ```
  qmgr -c 'set sched multisched_1 scheduling=1'
  ```

- Start the multisched, and specify the port number:

  pbs_sched -I <name of multisched> -S <same value as sched_port for this multisched>

  For example:

  ```
  pbs_sched -I multisched_1 -S 15050
  ```

4.2.4 Using the Default Scheduler with Multischeds

PBS automatically creates the default scheduler; its name is "default". The `sched_priv` directory of the default scheduler is always `$PBS_HOME/sched_priv`. The default scheduler writes its logs in `$PBS_HOME/sched_logs`. If you do nothing, the default scheduler uses the default scheduling policy defined in the default `sched_config` file. Default behavior is described in section 4.4.7, “Default Scheduling Policy”, on page 86. You can set the desired policy for the default scheduler. The default scheduler schedules jobs using queues and vnodes in the default partition.

4.2.4.1 Configuring the Default Scheduler

When you use the `qmgr` command to configure the default scheduler, specify its name:

qmgr -c "set sched default <attribute> = <value>"

For example:

```
qmgr -c "set sched default job_sort_formula_threshold = <value>"
```

4.2.5 Multisched Caveats and Restrictions

- You cannot delete the default scheduler.
- You cannot change the name of the default scheduler.
- You cannot set sched_host, sched_priv, or sched_port for the default scheduler.
- If you create a new queue or vnode without assigning it to a specific partition, it is scheduled by the default scheduler.
- You cannot assign a new multisched to a partition that is already assigned to a multisched, or vice versa. To make the change, offline the vnodes, wait for jobs to finish running, then un-assign and re-assign the multisched or partition.
- You cannot change a queue to a routing queue when the queue has its partition attribute set.
- You cannot associate a vnode and a queue and assign them separate partitions.
- If there is more than one scheduler, job run limits for the entire complex set at the server are not supported. Queue limits are enforced.
- All schedulers in the complex have as a default value for backfill_depth the value that is set at the server. For each scheduler, this is overridden by the setting at a scheduler's queue.
- All schedulers in the complex use the same value for job_sort_formula, so all schedulers use the same formula.
- If the complex has more than one scheduler, you cannot use complex-wide fairshare. Each scheduler manages its own fairshare tree.

4.2.6 Attributes Used with Multischeds

partition

Scheduler attribute. Partition for which this scheduler is to run jobs. Cannot be set on default scheduler.

Format: *String*

Default: "*None*"

partition

Vnode attribute. Name of partition to which this vnode is assigned. A vnode can be assigned to at most one partition.

Settable by Manager and Operator, viewable by all.

Format: *String*

partition

Queue attribute. Name of partition to which this queue is assigned.

Cannot be set for routing queue.

An execution queue cannot be changed to a routing queue while this attribute is set.

Settable by Manager, administrator, viewable by all.

Format: *String*

sched_log

Scheduler attribute. Directory where this scheduler writes its logs. Permissions should be *755*. Must be owned by root. Cannot be shared with another scheduler. For default scheduler, directory is always `PBS_HOME/sched_log`.

Settable by Manager, administrator, viewable by all.

Default: `$PBS_HOME/sched_logs_<scheduler name>`

sched_port

Scheduler attribute. Port on which this scheduler listens. Cannot be set on default scheduler.

Settable by Manager, administrator, viewable by all.

Default: None

sched_host

Scheduler attribute. Hostname on which scheduler runs.

Cannot be set on default scheduler; value for default scheduler is server hostname.

Settable by Manager or administrator, viewable by all.

scheduling

Scheduler attribute. Enables scheduling of jobs. If you set the server's scheduling attribute, that value is assigned to the default scheduler's scheduling attribute, and vice versa.

Settable by Manager or administrator, viewable by all.

Default value for default scheduler: *True*

Default value for multisched: *False*

scheduler_iteration

Scheduler attribute. Time between scheduling iterations. If you set the server's scheduler_iteration attribute, that value is assigned to the default scheduler's scheduler_iteration attribute, and vice versa.

Settable by Manager, administrator, viewable by all.

Default: *600*

sched_priv

Scheduler attribute. Directory where this scheduler keeps fairshare `usage`, `resource_group`, `holidays`, and `sched_config`. Must be owned by root. For default scheduler, directory is always `PBS_HOME/sched_priv`.

Settable by Manager or administrator, viewable by all.

Default: `$PBS_HOME/sched_priv_<scheduler name>`

state

Scheduler attribute. State of this scheduler.

Set by server. Readable by all.

Valid values: one of *down, idle, scheduling*

down: scheduler is not running

idle: scheduler is running and is waiting for a scheduling cycle to be triggered

scheduling: scheduler is running and is in a scheduling cycle

Format: *String*

Default value for default scheduler: *idle*

Default value for multisched: *down*

comment

Scheduler attribute. Can be set by PBS or administrator. For certain scheduler errors, PBS sets the scheduler's comment attribute to specific error messages. You can use the comment field to notify another administrator of something, but PBS does overwrite the value of comment under certain circumstances.

Format: *String*

4.2.6.1 Behavior of Attributes Shared by Server and Scheduler

If you set the server's scheduling or scheduler_iteration attributes, the changes are applied to the default scheduler, and its corresponding scheduling or scheduler_iteration attributes are given the new setting(s). The reverse is also true.

4.2.7 Multisched Errors and Logging

4.2.7.1 Multisched Error Messages Appearing in Scheduler Comment

A scheduler's comment attribute can be set to specific error messages.

- Setting the sched_log attribute to an invalid value produces a scheduler comment. If the log directory is not accessible by the scheduler:

  `Unable to change the sched_log directory`

  In addition, the scheduling attribute is set to *False*.
- Attempting to set the sched_priv attribute to an invalid value:

  `Unable to change the sched_priv directory`

  In addition, the scheduling attribute is set to *False*.
- Setting sched_priv to a directory that fails validation checking produces a scheduler comment. If the `sched_priv` directory is not accessible by the scheduler, or the scheduler files are not found in the directory:

  `PBS failed validation checks for sched_priv directory`

  In addition, the scheduling attribute is set to *False*.

4.2.7.2 Multisched Error Messages Appearing in Scheduler Logs

- Attempting to start a multisched before assigning it a partition:
  `Scheduler does not contain a partition`
- When the partition has been removed from a multisched, the multisched is shut down.
  `Scheduler does not contain a partition`
- When the scheduler cannot get its attribute information from the server:
  `Unable to retrieve the scheduler attributes from server`

4.2.7.3 Multisched Error Messages Appearing in Server Logs

- Attempting to set sched_port, sched_priv, or sched_host for the default scheduler:
  `Operation is not permitted on default scheduler`
- Attempting to set the partition for the default scheduler:
  `Operation is not permitted on default scheduler`
  In addition, the error code is set to 15223.
- Attempting to assign a `sched_priv` to a multisched while that directory is already assigned to another multisched:
  `Another scheduler has same value for its sched_priv directory`
  In addition, the error code is set to 15216.
- Attempting to assign a `sched_log` to a multisched while that directory is already assigned to another multisched:
  `Another scheduler has same value for its sched_log directory`
  In addition, the error code is set to 15215.
- If the server is not able to reach a scheduler one of the following messages appears:
  `Unable to reach scheduler associated with partition [<partition ID>]`
  `Unable to reach scheduler associated with job <job ID>`

4.2.7.4 Multisched Errors Returned by qmgr Command

- Attempting to associate a vnode with a queue that has not been assigned to the same partition:
  qmgr obj=<vnode name> svr=<server name>: Partition <partition name> is not part of queue for node
  qmgr: Error (15220) returned from server
- Attempting to assign a partition to a vnode when that vnode is associated with a queue and the queue is not assigned to the same partition:
  qmgr obj=<vnode name> svr=<server name>: Queue <queue name> is not part of partition for node
  qmgr: Error (15219) returned from server
- When a queue is associated with one or more vnodes, and a partition is assigned to the queue and vnodes, attempting to change the queue's partition:
  qmgr obj=<queue name> svr=<server name>: Invalid partition in queue
  qmgr: Error (15221) returned from server
- Attempting to assign a partition to a multisched while that partition is already assigned to another multisched:
  Partition <partition name> is already associated with scheduler <scheduler name>.
- Attempting to set the partition attribute for a routing queue:
  qmgr obj=<queue name> svr=<server name>: Cannot assign a partition to route queue
  qmgr: Error (15217) returned from server
- Attempting to change an execution queue to a routing queue while the partition attribute is set:
  qmgr obj=<queue name> svr=<server name>: Cannot queue_type=route if partition is set
  qmgr: Error (15218) returned from server

4.2.8 Multisched Deprecations

Using qmgr on the default scheduler, without specifying it name, is deprecated. The old syntax is supported for backward compatibility.

Example of old syntax:

```
qmgr -c "set sched job_sort_formula_threshold = <value>"
```

Same with new syntax:

```
qmgr -c "set sched default job_sort_formula_threshold = <value>"
```

4.3 Scheduling Policy Basics

4.3.1 How Scheduling Can Be Used

You can use the scheduling tools provided by PBS to implement your chosen scheduling policy, so that your jobs run in the way you want.

Your policy can do the following:

- Prioritize jobs according to your specification
- Run jobs according to their relative importance
- Award specific amounts of resources such as CPU time to projects, users, and groups according to rules that you set
- Make sure that resources are not misused
- Optimize how jobs are placed on vnodes, so that jobs run as efficiently as possible
- Use special time slots for particular tasks
- Optimize throughput or turnaround time for jobs

4.3.2 What Is Scheduling Policy?

Scheduling policy determines when each job is run and on which resources. In other words, a scheduling policy describes a goal, or intended behavior. For convenience, we describe a scheduling policy as being a combination of sub-goals, for example a combination of how resources should be allocated and how efficiency should be maximized.

You implement a scheduling policy using the tools PBS provides. A scheduling tool is a feature that allows you control over some aspect of scheduling. For example, the job sorting formula is a tool that allows you to define how you want job execution priority to be computed. Some scheduling tools are supplied by the PBS scheduler(s), and some are supplied by other elements of PBS, such as the hooks, server, queues or resources.

You can group the resources in your complex into partitions, and you can run a separate scheduling policy on each partition.

4.3.3 Basic PBS Scheduling Behavior

The basic behavior of PBS is that it always places jobs where it finds the resources requested by the job. PBS will not place a job where that job would use more resources than PBS thinks are available. For example, if you have two jobs, each requesting 1 CPU, and you have one vnode with 1 CPU, PBS will run only one job at a time on the vnode. You do not have to configure PBS for this basic behavior.

PBS determines what hardware resources are available and configures them for you. However, you do have to inform PBS which custom resources and non-hardware resources are available and where, how much, and whether they are consumable or not. In addition, in order to ensure that jobs are sent to the appropriate vnodes for execution, you also need to make sure that they request the correct resources. You can do this either by having users submit their jobs with the right resource requests, using hooks that set job resources, or by configuring default resources for jobs to inherit.

4.3.4 Sub-goals

Your scheduling policy is the combination that you choose of one or more sub-goals. For example, you might need to meet two particular sub-goals: you might need to prioritize jobs a certain way, and you might need to use resources efficiently. You can choose among various outcomes for each sub-goal. For example, you can choose to prioritize jobs according to size, owner, owner's usage, time of submission, etc.

In the following sections, we describe the tools PBS offers for meeting each of the following sub-goals.

- Job prioritization and preemption; see section 4.3.5, “Job Prioritization and Preemption”, on page 64.
- Resource allocation & limits; see section 4.3.6, “Resource Allocation to Users, Projects & Groups”, on page 70.
- Time slot allocation; see section 4.3.7, “Time Slot Allocation”, on page 72.
- Job placement optimizations; see section 4.3.8, “Job Placement Optimization”, on page 73.
- Resource efficiency optimizations; see section 4.3.9, “Resource Efficiency Optimizations”, on page 76.
- Overrides; see section 4.3.10, “Overrides”, on page 78.

4.3.5 Job Prioritization and Preemption

Job prioritization is any technique you use to come up with a ranking of each job's relative importance. You can specify separate priority schemes for both execution and preemption.

4.3.5.1 Where PBS Uses Job Priority

PBS calculates job priority for two separate tasks: job execution and job preemption. Job execution priority is used with other factors to determine when to run each job. Job preemption priority is used to determine which queued jobs are allowed to preempt which running jobs in order to use their resources and run. These two tasks are independent, and it is important to make sure that you do not make them work at cross-purposes. For example, you do not want to have a class of jobs having high execution priority and low preemption priority; these jobs would run first, and then be preempted first.

Preemption comes into play when a scheduler examines the top job and determines that it cannot run now. If preemption is enabled, a scheduler checks to see whether the top job has sufficient preemption priority to be able to preempt any running jobs, and then if it does, whether preempting jobs would yield enough resources to run the top job. If both are true, a scheduler preempts running jobs and runs the top job.

If you take no action to configure how jobs should be prioritized, they are considered in submission order, one queue at a time. If you don't prioritize queues, the queues are examined in an undefined order.

4.3.5.2 Overview of Prioritizing Jobs

PBS provides several tools for setting job execution priority. There are queue-based tools for organizing jobs, moving them around, and specifying the order in which groups of jobs should be examined. There are tools for sorting jobs into the order you want. There is a meta-tool (strict ordering) that allows you to specify that the top job must go next, regardless of whether the resources it requires are available now.

A scheduler can use multiple sorting tools, in succession. You can combine your chosen sorting tools with queue-based tools to give a wide variety of behaviors. Most of the queue-based tools can be used together. A scheduler can treat all jobs as if they are in a single queue, considering them all with respect to each other, or it can examine all queues that have the same priority as a group, or it can examine jobs queue by queue, comparing each job only to other jobs in the same queue.

You can change how execution priority is calculated, depending on which time slot is occurring. You can divide time up into primetime, non-primetime, and dedicated time.

When a scheduler calculates job execution priority, it uses a built-in system of job classes. PBS runs special classes of jobs before it considers queue membership. These classes are for reservation, express, preempted, and starving jobs. Please see section 4.9.16, “Calculating Job Execution Priority”, on page 134. After these jobs are run, a scheduler follows the rules you specify for queue behavior. Within each queue, jobs are sorted according to the sorting tools you choose.

4.3.5.3 Using Queue-based Tools to Prioritize Jobs

4.3.5.3.i Using Queue Order to Affect Order of Consideration

When a scheduler examines queued jobs, it can consider all of the jobs in its partition as a whole, it can round-robin through groups of queues where the queues are grouped by priority, or it can examine jobs in only one queue at a time. These three systems are incompatible. Queues are always sorted by priority.

The by_queue scheduler parameter controls whether a scheduler runs all the jobs it can from the highest-priority queue before moving to the next, or treats all jobs as if they are in a single queue. By default, this parameter is set to *True*. When examining jobs one queue at a time, a scheduler runs all of the jobs it can from the highest-priority queue first, then moves to the next highest-priority queue and runs all the jobs it can from that queue, and so on. See section 4.9.4, "Examining Jobs Queue by Queue", on page 110.

The round_robin scheduler parameter controls whether or not a scheduler round-robins through queues. When a scheduler round-robins through queues, it groups the queues by priority, and round-robins first through the highest-priority group, then the next highest-priority group, and so on, running all of the jobs that it can from that group. So within each group, if there are multiple queues, a scheduler runs the top job from one queue, then the top job from the next queue, and so on, then goes back to the first queue, runs its new top job, goes to the next queue, runs its new top job, and so on until it has run all of the jobs it can from that group. All queues in a group must have exactly the same priority. The order in which queues within a group are examined is undefined. If all queues have different priorities, a scheduler starts with the highest-priority queue, runs all its jobs, moves to the next, runs its jobs, and so on until it has run all jobs from each queue. This parameter overrides by_queue. See section 4.9.38, "Round Robin Queue Selection", on page 204.

If you want queues to be considered in a specific order, you must assign a different priority to each queue. Queues are always sorted by priority. See section 4.9.47, "Sorting Queues into Priority Order", on page 223. Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. If you want groups of queues to be considered together for round-robining, give all queues in each group one priority, and all queues in the next group a different priority. If the queues don't have priority assigned to them, the order in which they are considered is undefined. To set a queue's priority, use the `qmgr` command to assign a value to the priority queue attribute. See section 2.3.5.3, "Prioritizing Execution Queues", on page 24.

4.3.5.3.ii Using Express Queues in Job Priority Calculation

You can create express queues, and route jobs into them, if you want to give those jobs special priority.

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but this can be set using the preempt_queue_prio scheduler attribute. For information on configuring express queues, see section 2.3.5.3.i, "Express Queues", on page 24.

When calculating execution priority, a PBS scheduler uses a built-in job class called "*Express*" which contains all jobs that have a preemption level greater than that of the normal_jobs level. By default, those jobs are jobs in express queues. See section 4.9.16, "Calculating Job Execution Priority", on page 134.

You can create preemption levels that include jobs in express queues. Jobs in higher preemption levels are allowed to preempt jobs in lower levels. See section 4.9.33, "Using Preemption", on page 180.

4.3.5.3.iii Routing Jobs into Queues

You can configure PBS to automatically put each job in the most appropriate queue. There are several approaches to this. See section 4.9.39, "Routing Jobs", on page 205.

4.3.5.3.iv Using Queue Priority when Computing Job Priority

You can configure a scheduler so that job priority is partly determined by the priority of the queue in which the job resides. See section 4.9.36, "Queue Priority", on page 196.

4.3.5.4 Using Job Sorting Tools to Prioritize Jobs

A scheduler can use multiple job sorting tools in succession to determine job execution priority. A scheduler groups all jobs waiting to run into classes, and then applies the sorting tools you choose to all jobs in each class.

- You can create a formula that a scheduler uses to sort jobs. A scheduler applies this formula to all jobs in its partition, using it to calculate a priority for each job. For example, you can specify in the formula that jobs requesting more CPUs have higher priority. If the formula is defined, it overrides fairshare and sorting jobs on keys. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.
- You can use the fairshare algorithm to sort jobs. This algorithm allows you to set a resource usage goal for users or groups. Jobs are prioritized according to each entity's usage; jobs whose owners have used the smallest percentage of their allotment go first. For example, you can track how much CPU time is being used, and allot each group a percentage of the total. See section 4.9.19, “Using Fairshare”, on page 138.
- You can sort jobs according to the same usage allotments you set up for fairshare. In this case, jobs whose owners are given the highest allotment go first. See section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 131.
- You can sort jobs on one or more keys, for example, you can sort jobs first by the number of CPUs they request, then by the amount of memory they request. You can specify that either the high or the low end of the resulting sort has higher priority.

  You can create a custom resource, and use a hook to set a value for that resource for each job, and then sort on the resource.

  See section 4.9.45, “Sorting Jobs on a Key”, on page 221.
- You can run jobs in the order in which they were submitted. See section 4.9.20, “FIFO Scheduling”, on page 148.
- You can run jobs according to the priority requested for each job at submission time. This priority can be set via a hook. See section 4.9.46, “Sorting Jobs by Requested Priority”, on page 223 and the PBS Professional Hooks Guide.

4.3.5.5 Prioritizing Jobs by Wait Time

You can use the amount of time a job has been waiting to run in the priority calculation. There are two ways to measure wait time:

- Eligible waiting time: how long a job has been waiting to run due to a shortage of resources, rather than because its owner isn't allowed to run jobs now. See section 4.9.13, “Eligible Wait Time for Jobs”, on page 126
- Amount of time waiting in the queue

Both of these ways can be used when computing whether or not a job is starving. You can specify how long a job must be waiting to be considered starving. See section 4.9.48, “Starving Jobs”, on page 223.

You can use a job's eligible waiting time in the job sorting formula. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

When a job is considered to be starving, it is automatically assigned special execution priority, and placed in the Starving execution priority class; see section 4.9.16, “Calculating Job Execution Priority”, on page 134. You can configure preemption levels that include starving jobs; see section 4.9.33, “Using Preemption”, on page 180.

4.3.5.6 Calculating Preemption Priority

Execution priority and preemption priority are two separate systems of priority.

By default, if the top job cannot run now, and it has high preemption priority, a scheduler will use preemption to run the top job. A scheduler will preempt jobs with lower preemption priority so that it can use the resources to run the top job. The default definition of jobs with high preemption priority is jobs in express queues. You can configure many levels of preemption priority, specifying which levels can preempt which other levels. See section 4.9.33, "Using Preemption", on page 180.

4.3.5.7 Making Preempted Jobs Top Jobs

You can specify that a scheduler should make preempted jobs be top jobs. See section 4.9.3.7, "Configuring Backfilling", on page 107.

4.3.5.8 Preventing Jobs from Being Preempted

You may have jobs that should not be preempted, regardless of their priority. These can be jobs which cannot be effectively preempted, so that preempting them would waste resources. To prevent these jobs from being preempted, do one or both of the following:

- Set a value for the preempt_targets resource at all jobs that specify a value for a custom resource. For example, define a Boolean resource named Preemptable, and add "Resource_List.Preemptable=true" to preempt_targets for all jobs. Then set the value of Resource_List.Preemptable to *False* for the jobs you don't want preempted.

  For example, if we want JobA and JobB to be able to preempt Job1 and Job2, but not Job3:

  Define a Boolean resource named "Preemptable"

  For Job1 and Job2, set Resource_List.Preemptable to *True*:

  ```
  qsub ... -l Preemptable=True ...
  ```

  or

  ```
  qalter -l Preemptable=True Job1 Job2
  ```

  For Job3, set Resource_List.Preemptable to *False*:

  ```
  qalter -l Preemptable=False Job3
  ```

  For JobA and JobB, set Resource_List.preempt_targets to "*Preemptable=True*":

  ```
  qalter -l preempt_targets=Resource_List.Preemptable=True JobA JobB
  ```

- Route jobs you don't want preempted to one or more specific queues, and then use a hook to make sure that no jobs specify these queues in their preempt_targets.

4.3.5.9 Meta-priority: Running Jobs Exactly in Priority Order

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. If a job cannot run now because the resources required are unavailable, the default behavior is to skip the job and move to the next in order of priority.

You can tell PBS to use a different behavior called *strict ordering*. This means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

You can see that using strict ordering could lead to decreased throughput and idle resources. In order to prevent idle resources, you can tell PBS to run small filler jobs while it waits for the resources for the top job to become available. These small filler jobs do not change the start time of the top job. See section 4.9.49, "Using Strict Ordering", on page 225 and section 4.9.3, "Using Backfilling", on page 105.

4.3.5.10 Using Different Calculations for Different Time Periods

PBS allows you to divide time into two kinds, called *primetime* and *non-primetime*. All time is covered by one or the other of these two kinds of time. The times are arbitrary; you can set them up however you like. You can also choose not to define them, and instead to treat all time the same.

You can configure two separate, independent ways of calculating job priority for primetime and non-primetime. The same calculations are used during dedicated time; dedicated time is a time slot made up of primetime and/or non-primetime. Many scheduler parameters are prime options, meaning that they can be configured separately for primetime and non-primetime. For example, you can configure fairshare as your sorting tool during primetime, but sort jobs on a key during non-primetime.

If you use the formula, it is in force all of the time.

See section 4.9.34, “Using Primetime and Holidays”, on page 191.

4.3.5.11 When Priority Is Not Enough: Overrides

Sometimes, the tools available for setting job priority don't do everything you need. For example, it may be necessary to run a job right away, regardless of what else is running. Or you may need to put a job on hold. Or you might need to tweak the way the formula works for the next *N* jobs. See section 4.9.30, “Overrides”, on page 162.

4.3.5.12 Elements to Consider when Prioritizing Jobs

- Whether users, groups, or projects affect job priority: for techniques to use user, group, or project to affect job priority, see section 4.4.3, “Prioritizing Jobs by User, Project or Group”, on page 80.
- Reservation jobs: jobs in reservations cannot be preempted.
- Starving jobs: PBS has a built-in execution priority for starving jobs, but you can give starving jobs the highest execution priority by giving them the highest preemption priority and enabling preemption. See section 4.9.16, “Calculating Job Execution Priority”, on page 134 and section 4.9.33, “Using Preemption”, on page 180.
- Express jobs: PBS has a built-in execution priority for express jobs. You can set the preemption priority for express jobs; see section 4.9.33, “Using Preemption”, on page 180.
- Preempted jobs: PBS has a built-in execution priority for preempted jobs. See section 4.9.16, “Calculating Job Execution Priority”, on page 134.
- Large or small jobs: you may want to give large and/or small jobs special treatment. See section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 82.
- User's priority request for job: the job submitter can specify a priority for the job at submission. You can sort jobs according to each job's specified priority. See section 4.9.46, “Sorting Jobs by Requested Priority”, on page 223.
- Whether the top job must be the next to run, regardless of whether it can run now; see section 4.9.49, “Using Strict Ordering”, on page 225.

4.3.5.13 List of Job Sorting Tools

4.3.5.13.i Queue-based Tools for Organizing Jobs

- Queue-by-queue: PBS runs all the jobs it can from the first queue before moving to the next queue. Queue order is determined by queue priority. See section 4.9.4, “Examining Jobs Queue by Queue”, on page 110.
- Round-robin job selection: PBS can select jobs from queues with the same priority in a round-robin fashion. See section 4.9.38, “Round Robin Queue Selection”, on page 204.
- Queue priority: Queues are always ordered according to their priority; jobs in higher-priority queues are examined before those in lower-priority queues. See section 2.3.5.3, “Prioritizing Execution Queues”, on page 24.
- Sorting queues: PBS always sorts queues into priority order. See section 4.9.47, “Sorting Queues into Priority Order”, on page 223.
- Express queues: Jobs in express queues are assigned increased priority. See section 2.3.5.3.i, “Express Queues”, on page 24, and section 4.3.5.3.ii, “Using Express Queues in Job Priority Calculation”, on page 65.
- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 4.9.39, “Routing Jobs”, on page 205.

4.3.5.13.ii Job Sorting Tools

You can use multiple job sorting tools, one at a time in succession. You can use different sorting tools for primetime and non-primetime.

- Job sorting formula: You create a formula that PBS uses to calculate each job's priority. See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.
- Fairshare: PBS tracks past usage of specified resources, and starts jobs based on specified usage ratios. See section 4.9.19, “Using Fairshare”, on page 138.
- Sorting jobs on keys: PBS can sort jobs according to one or more keys, such as requested CPUs or memory; see section 4.9.45, “Sorting Jobs on a Key”, on page 221.
- Entity shares (strict priority): Jobs are prioritized according to the owner's fairshare allocation. See section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 131.
- FIFO: Jobs can be run in submission order. See section 4.9.20, “FIFO Scheduling”, on page 148.
- Job's requested priority: you can sort jobs on the priority requested for the job; see section 4.9.46, “Sorting Jobs by Requested Priority”, on page 223.

4.3.5.13.iii Other Job Prioritization Tools

- Strict ordering: you can specify that jobs must be run in priority order, so that a job that cannot run because resources are unavailable is not skipped. See section 4.9.49, "Using Strict Ordering", on page 225.
- Waiting time: PBS can assign increased priority to jobs that have been waiting to run. See section 4.9.13, "Eligible Wait Time for Jobs", on page 126, and section 4.9.48, "Starving Jobs", on page 223.
- Setting job execution priority: PBS can set job execution priority according to a set of rules. See section 4.9.16, "Calculating Job Execution Priority", on page 134.
- Preemption: PBS preempts lower-priority jobs in order to run higher-priority jobs. See section 4.9.33, "Using Preemption", on page 180.
- Starving jobs: Jobs that have been waiting for a specified amount of time can be given increased priority. See section 4.9.48, "Starving Jobs", on page 223.
- Preventing preemption: You can prevent certain jobs from being preempted. See section 4.3.5.8, "Preventing Jobs from Being Preempted", on page 67.
- Making preempted jobs top jobs: PBS can backfill around preempted jobs. See section 4.9.3.5, "Backfilling Around Preempted Jobs", on page 106.
- Behavior overrides: you can intervene manually in how jobs are run. See section 4.9.30, "Overrides", on page 162.

4.3.6 Resource Allocation to Users, Projects & Groups

If you need to ensure fairness, you may need to make sure that resources are allocated fairly. If different users, groups, or projects own or pay for different amounts of hardware or machine time, you may need to allocate resources according to these amounts or proportions.

You can allocate hardware-based resources such as CPUs or memory, and/or time-based resources such as walltime or CPU time, according to to the agreed amounts or proportions. You can also control who starts jobs.

4.3.6.1 Limiting Amount of Resources Used

4.3.6.1.i Allocation Using Resource Limits

You can use resource limits as a way to enforce agreed allocation amounts. This is probably the most straightforward way, and the easiest to explain to your users. PBS provides a system for limiting the total amount of each resource used by projects, users, and groups at the server and at each queue. For example, you can set a limit on the number of CPUs that any generic user can use at one time at QueueA, but set three different individual limits for each of three users that have special requirements, at the same queue. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

4.3.6.1.ii Allocation Using Fairshare

The PBS fairshare tool allows you to start jobs according to a formula based on resource usage by job owners. You can designate who the valid job owners are, which resources are being tracked, and how much of the resources each owner is allowed to be using. Fairshare uses a moving average of resource usage, so that a user who in the recent past has not used their share can use more now. For example, you can track usage of the cput resource, and give one group 40 percent of usage, one 50 percent, and one group, 10 percent. See section 4.9.19, "Using Fairshare", on page 138.

4.3.6.1.iii Allocation Using Routing

If you do not want to place usage limits directly on projects, users, or groups, you can instead route their jobs to specific queues, where those queues have their own resource usage limits.

To route jobs this way, force users to submit jobs to a routing queue, and set access control limits at each execution queue. See section 8.3, "Using Access Control Lists", on page 362. Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

Using this method, you place a limit for total resource usage at each queue, for each resource you care about. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

You can also route jobs to specific queues, where those queues can send jobs only to specific vnodes. See section 4.9.2, "Associating Vnodes with Queues", on page 103.

4.3.6.2 Limiting Jobs

4.3.6.2.i Limiting Number of Jobs per Project, User, or Group

You can set limits on the numbers of jobs that can be run by projects, users, and groups. You can set these limits for each project, user, and group, and you can set them at the server and at each queue. You can set a generic limit for all projects, users, or groups, and individual limits that override the generic limit. For example, you can set a limit that says that no user at its partition can run more than 8 jobs. Then you can set a more specific limit for QueueA, so that users at QueueA can run 4 jobs. Then you can set a limit for User1 and User2 at QueueA, so that they can run 6 jobs. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

4.3.6.2.ii Allocation Using Round-robin Queue Selection

PBS can select jobs from queues by examining groups of queues in round-robin fashion, where all queues in each group have the same priority. When using the round-robin method, a scheduler considers the first queue in a group, tries to run the top job from that queue, then considers the next queue, tries to run the top job from that queue, then considers the next queue, and so on, in a circular fashion. A scheduler runs all the jobs it can from the highest-priority group first, then moves to the group with the next highest priority.

If you want a simple way to control how jobs are started, you can use round-robin where each queue in a group belongs to a different user or entity. See section 4.9.38, "Round Robin Queue Selection", on page 204.

4.3.6.2.iii Limiting Resource Usage per Job

If you are having trouble with large jobs taking up too much of a resource, you can limit the amount of the resource being used by individual jobs. You can set these limits at each queue, and at the server. See section 5.15.2, "Placing Resource Limits on Jobs", on page 304.

4.3.6.3 Resource Allocation Tools

The following is a list of scheduling tools that you can use for allocating resources or limiting resources or jobs:

- Matching: PBS places jobs where the available resources match the job's resource requirements; see section 4.9.28, "Matching Jobs to Resources", on page 159.
- Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.9.37, "Reservations", on page 197.
- Fairshare: PBS tracks past usage of specified resources, and starts jobs based on specified usage ratios. See section 4.9.19, "Using Fairshare", on page 138.
- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.3.6, "Routing Queues", on page 25 and section 4.9.39, "Routing Jobs", on page 205.
- Limits on resource usage by projects, users, and groups: You can set limits on user and group resource usage. See section 4.9.26, "Limits on Project, User, and Group Resource Usage", on page 156.
- Round-robin job selection: PBS can select jobs from queues that have the same priority in a round-robin fashion. See section 4.9.38, "Round Robin Queue Selection", on page 204.
- Sorting queues: PBS always sorts queues into priority order. See section 4.9.47, "Sorting Queues into Priority Order", on page 223.
- Limits on number of jobs for projects, users, and groups: You can set limits on the numbers of jobs that can be run by projects, users, and groups. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.
- Limits on resources used by each job: You can set limits on the amount of each resource that any job can use. See section 4.9.24, "Limits on Per-job Resource Usage", on page 156.
- Using custom resources to limit resource usage: You use custom resources to manage usage. See section 4.9.8, "Using Custom and Default Resources", on page 113.
- Gating and admission requirements: You can specify admission requirements for jobs. See section 4.9.22, "Gating Jobs at Server or Queue", on page 155.
- Making jobs inherit default resources: You can use default resources to manage jobs. See section 4.9.8, "Using Custom and Default Resources", on page 113.

4.3.7 Time Slot Allocation

Time slot allocation is the process of creating time slots within which only specified jobs are allowed to run.

4.3.7.1 Why Allocate Time Slots

You may want to set up blocks of time during which only certain jobs are allowed to run. For example, you might need to ensure that specific high-priority jobs have their own time slot, so that they are guaranteed to be able to run and finish before their results are required.

You may want to divide jobs into those that run at night, when no one is around, and those that run during the day, because their owners need the results then.

You might want to run jobs on desktop clusters only at night, when the primary users of the desktops are away.

When you upgrade PBS, a chunk of dedicated time can come in very handy. You set up dedicated time for a time period that is long enough for you to perform the upgrade, and you make sure the time slot starts far enough out that no jobs will be running.

You may want to run different scheduling policies at different times or on different days.

4.3.7.2 How to Allocate Time Slots

Time slots are controlled by queues: primetime queues, non-primetime queues, dedicated time queues, and reservation queues. For this, you use your favorite routing method to move jobs into the desired queues. See section 4.9.39, "Routing Jobs", on page 205.

4.3.7.2.i Allocation Using Primetime and Holidays

You can specify how to divide up days or weeks, and designate each time period to be either primetime or non-primetime. You can use this division in the following ways:

- You can run a different policy during primetime from that during non-primetime
- You can run specific jobs during primetime, and others during non-primetime

See section 4.9.34, "Using Primetime and Holidays", on page 191.

4.3.7.2.ii Allocation Using Dedicated Time

Dedicated time is a time period where the only jobs that are allowed to run are the ones in dedicated time queues. The policy you use during dedicated time is controlled by the normal primetime and non-primetime policies; those times overlap dedicated time.

If you don't allow any jobs into a dedicated time queue, you can use it to perform maintenance, such as an upgrade.

See section 4.9.10, "Dedicated Time", on page 125.

4.3.7.2.iii Allocation Using Reservations

You and any other PBS user can create advance and standing reservations. These are time periods with a defined start and end, for a specific, defined set of resources. Reservations are used to make sure that specific jobs can run on time. See section 4.9.37, "Reservations", on page 197.

4.3.7.2.iv Allocation Using `cron` Jobs

You can use `cron` to run jobs at specific times. See section 4.9.7, "cron Jobs", on page 112.

4.3.7.3 Time Slot Allocation Tools

The following is a list of scheduling tools that you can use to create time slots:

- Primetime and holidays: You can specify days and times that are to be treated as prime execution time. See section 4.9.34, "Using Primetime and Holidays", on page 191.
- Dedicated time: You can set aside blocks of time reserved for certain system operations. See section 4.9.30.6, "Using Dedicated Time", on page 164.
- `cron` jobs: You can use `cron` to run jobs. See section 4.9.30.7, "Using cron Jobs", on page 164.
- Reservations: Users can create advance and standing reservations for specific resources for specific time periods. See section 4.9.37, "Reservations", on page 197.

4.3.8 Job Placement Optimization

PBS automatically places jobs where they can run, but you can refine how jobs are placed.

Optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of where jobs can be run.

PBS places jobs according to placement optimization settings in tools to specify how vnodes should be organized, how jobs should be distributed, and how resources should be used.

4.3.8.1 Why Optimize Placement

PBS automatically places jobs where they can run, matching jobs to resources, so why optimize placement?

- You can help PBS refine its understanding of hardware topology, so that PBS can place jobs where they will run most efficiently.
- If you have some vnodes that are faster than others, you can preferentially place jobs on those vnodes.
- You may need to place jobs according to machine ownership, so that for example only jobs owned by a specific group run on a particular machine.
- You can take advantage of unused workstation computing capacity.
- You can balance the workload between two or more PBS partitions or complexes, trading jobs around depending on the workload on each partition or complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

4.3.8.2 Matching Jobs to Resources

By default, PBS places jobs where the available resources match the job's resource requirements. See section 4.9.28, “Matching Jobs to Resources”, on page 159.

4.3.8.3 Organizing and Selecting Vnodes

By default, the order in which PBS examines vnodes is undefined. The default setting for vnode sorting is the following:

```
node_sort_key: "sort_priority HIGH all"
```

However, sort_priority means sort on each vnode's priority attribute, but by default, that attribute is unset.

PBS can organize vnodes into groups. By default, PBS does not organize vnodes into groups.

By default, when PBS chooses vnodes for a job, it runs down its list of vnodes, searching until it finds vnodes that can supply the job with the requested resources. You can improve this in two ways:

- PBS provides a way to organize your vnodes so that jobs can run on groups of vnodes, where the selected group of vnodes provides the job with good connectivity. This can improve memory access and interprocess communication timing. PBS then searches through these groups of vnodes, called *placement sets*, looking for the smallest group that satisfies the job's requirements. Each placement set is a group of vnodes that share a value for a resource. An illustrative example is a group of vnodes that are all connected to the same high speed switch, so that all of the vnodes have the same value for the switch resource. For detailed information on how placement sets work and how to configure them, see section 4.9.32, “Placement Sets”, on page 168.
- By default, the order in which PBS examines vnodes, whether in or outside of placement sets, is undefined. PBS can sort vnodes on one or more keys. Using this tool, you can specify which vnodes should be selected first. For information on sorting vnodes on keys, see section 4.9.50, “Sorting Vnodes on a Key”, on page 226.

You can sort vnodes in conjunction with placement sets.

4.3.8.4 Distributing Jobs

All of the following methods for distributing jobs can be used together.

4.3.8.4.i Filtering Jobs to Specific Vnodes

If you want to run certain kinds of jobs on specific vnodes, you can route those jobs to specific execution queues, and tie those queues to the vnodes you want. For example, if you want to route jobs requesting large amounts of memory to your large-memory machines, you can set up an execution queue called LMemQ, and associate that queue with the large-memory vnodes. You can route any kind of job to its own special execution queue. For example, you can route jobs owned by the group that owns a cluster to a special queue which is associated with the cluster. For details on routing jobs, see section 4.9.39, “Routing Jobs”, on page 205. For details on associating vnodes and queues, see section 4.9.2, “Associating Vnodes with Queues”, on page 103.

4.3.8.4.ii Running Jobs at Least-loaded Partition or Complex

You can set up cooperating PBS partitions and complexes that automatically run jobs from each other's queues. This allows you to dynamically balance the workload across multiple, separate PBS partitions and complexes. See section 4.9.31, “Peer Scheduling”, on page 165.

4.3.8.4.iii Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won't interfere with the user's operation. See section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 115.

4.3.8.4.iv Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.9.27, “Using Load Balancing”, on page 156.

4.3.8.4.v Placing Job Chunks on Desired Hosts

You can tell PBS to place each job on as few hosts as possible, to place each chunk of a job on a separate host, a separate vnode, or on any vnode. You can specify this behavior for the jobs at a queue and at the server.

You can do the following

- Set default placement behavior for the queue or server: jobs inherit placement if they do not request it; see section 5.9.3.5, “Specifying Default Job Placement”, on page 246
- Use a hook to set each job's placement request (Resource_List.place). See the PBS Professional Hooks Guide

For more on placing chunks, see section 4.9.6, “Organizing Job Chunks”, on page 112.

For information on how jobs request placement, see section 2.59.2.6, “Requesting Resources and Placing Jobs”, on page 216.

4.3.8.5 Shared or Exclusive Resources and Vnodes

PBS can give jobs their own vnodes, or fill vnodes with as many jobs as possible. A scheduler uses a set of rules to determine whether a job can share resources or a host with another job. These rules specify how the vnode sharing attribute should be combined with a job's placement directive. The vnode's sharing attribute supersedes the job's placement request.

You can set each vnode's sharing attribute so that the vnode or host is always shared, always exclusive, or so that it honors the job's placement request. See section 4.9.41, “Shared vs. Exclusive Use of Resources by Jobs”, on page 210.

4.3.8.6 Tools for Organizing Vnodes

- Placement sets: PBS creates sets of vnodes organized by the values of multiple resources. See section 4.9.32, "Placement Sets", on page 168.
- Sorting vnodes on keys: PBS can sort vnodes according to specified keys. See section 4.9.50, "Sorting Vnodes on a Key", on page 226.

4.3.8.7 Tools for Distributing Jobs

- Routing: You can set up a queue system so that jobs with certain characteristics are routed to specific queues. See section 2.3.6, "Routing Queues", on page 25 and section 4.9.39, "Routing Jobs", on page 205.
- Associating vnodes with queues: You can specify that jobs in a given queue can run only on specific vnodes, and vice versa. See section 4.9.2, "Associating Vnodes with Queues", on page 103.
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See section 4.9.9, "Using Idle Workstation Cycle Harvesting", on page 115.
- Peer scheduling: PBS partitions and complexes can exchange jobs. See section 4.9.31, "Peer Scheduling", on page 165.
- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.9.27, "Using Load Balancing", on page 156.
- SMP cluster distribution (**deprecated**): PBS can place jobs in a cluster as you specify. See section 4.9.43, "SMP Cluster Distribution", on page 218.

4.3.9 Resource Efficiency Optimizations

PBS automatically runs each job where the resources required for the job are available. You can refine the choices PBS makes.

Resource optimizations are the techniques you use to increase throughput, turnaround, or efficiency, by taking advantage of how resources are used.

Before reading this section, please make sure you understand how resources are used by reading section 4.9.28, "Matching Jobs to Resources", on page 159.

4.3.9.1 Why Optimize Use of Resources

You may want to take advantage of the following:

- If you are using strict ordering, you can prevent resources from standing idle while the top job waits for its resources to become available
- PBS can estimate the start times of jobs, so that users can stay informed
- PBS can provision vnodes with the environments that jobs require
- PBS can track resources that are outside of the control of PBS, such as scratch space
- You can take advantage of unused workstation computing capacity
- You can balance the workload between two or more PBS partitions or complexes, trading jobs around depending on the workload on each partition or complex.
- You can specify whether or not certain vnodes should be used for more than one job at a time.
- Users can specify that jobs that are dependent on the output of other jobs run only after the other jobs complete
- You can tell PBS to avoid placing jobs on highly-loaded vnodes

4.3.9.2 How to Optimize Resource Use

4.3.9.2.i Backfilling Around Top Jobs

PBS creates a list of jobs ordered by priority, and tries to run the jobs in order of priority. You can force all jobs to be run in exact order of their priority, using strict ordering. See section 4.9.49, “Using Strict Ordering”, on page 225. However, this can reduce resource utilization when the top job cannot run now and must wait for resources to become available, idling the entire partition or complex. You can offset this problem by using backfilling, where PBS tries to fit smaller jobs in around the top job that cannot run. The start time of the top job is not delayed. Job walltimes are required in order to use backfilling. You can specify the number of jobs around which to backfill. You can also disable this feature. See section 4.9.3, “Using Backfilling”, on page 105.

PBS can shrink the walltime of shrink-to-fit jobs into available time slots. These jobs can be used to backfill around top jobs and time boundaries such as dedicated time or reservations. See section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211.

If you do not use strict ordering, PBS won't necessarily run jobs in exact priority order. PBS will instead run jobs so that utilization is maximized, while trying to preserve priority order.

4.3.9.2.ii Using Dependencies

Job submitters can specify dependencies between jobs. For example, if you have a data analysis job that must run after data collection and cleanup jobs, you can specify that. See section 4.9.11, “Dependencies”, on page 126.

4.3.9.2.iii Estimating Start Time for Jobs

You can tell PBS to estimate start times and execution vnodes for either the number of jobs being backfilled around, or all jobs. Users can then see when their jobs are estimated to start, and the vnodes on which they are predicted to run. See section 4.9.15, “Estimating Job Start Time”, on page 131.

4.3.9.2.iv Provisioning Vnodes with Required Environments

PBS can provision vnodes with environments (applications or operating systems) that jobs require. This means that a job can request a particular environment that is not yet on a vnode, but is available to be instantiated there. See section 4.9.35, “Provisioning”, on page 196.

4.3.9.2.v Tracking Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for application licenses and scratch space. You can represent elements that are available to the entire partition or PBS complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.3.1.i, “Example of Configuring Dynamic Server-level Resource”, on page 267. For an example of configuring a dynamic host-level resource, see section 5.14.4.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 269.

For a complete description of how to create and use dynamic resources, see section 5.14, “Custom Resources”, on page 255.

4.3.9.3 Optimizing Resource Use by Job Placement

4.3.9.3.i Sending Jobs to Partition or Complex Having Lightest Workload

You can set up cooperating PBS partitions or complexes that automatically run jobs from each other's queues. This allows you to dynamically balance the workload across multiple, separate partitions or complexes. See section 4.9.31, “Peer Scheduling”, on page 165.

4.3.9.3.ii Using Idle Workstations

You can run jobs on workstations whenever they are not being used by their owners. PBS can monitor workstations for user activity or load, and run jobs when those jobs won't interfere with the user's operation. See section 4.9.9, "Using Idle Workstation Cycle Harvesting", on page 115.

4.3.9.3.iii Avoiding Highly-loaded Vnodes

You can tell PBS not to run jobs on vnodes that are above a specified load. This is in addition to the default behavior, where PBS does not run jobs that request more of a resource than it thinks each vnode can supply. See section 4.9.27, "Using Load Balancing", on page 156.

4.3.9.4 Resource Efficiency Optimization Tools

The following is a list of scheduling tools that you can use to optimize how resources are used:

- Backfilling around most important job(s): PBS can place small jobs in otherwise-unused blocks of resources. See section 4.9.3, "Using Backfilling", on page 105.
- Dependencies: Users can specify requirements that must be met by previous jobs in order for a given job to run. See section 4.9.11, "Dependencies", on page 126.
- Estimating start time of jobs: PBS can estimate when jobs will start, so that users can be informed. See section 4.9.15, "Estimating Job Start Time", on page 131.
- Provisioning vnodes with required environments: PBS can provision vnodes with the environments that jobs require. See section 4.9.35, "Provisioning", on page 196.
- Using dynamic resources: PBS can track resources such as scratch space and licenses. See section 4.9.12, "Dynamic Resources", on page 126.
- Idle workstation cycle harvesting: PBS can take advantage of unused workstation CPU time. See section 4.9.9, "Using Idle Workstation Cycle Harvesting", on page 115.
- Peer scheduling: PBS partitions and complexes can exchange jobs. See section 4.9.31, "Peer Scheduling", on page 165.
- Load balancing: PBS can place jobs so that machines have balanced loads. See section 4.9.27, "Using Load Balancing", on page 156.

4.3.10 Overrides

Overrides are the techniques you use to override the specified scheduling behavior of PBS.

4.3.10.1 Why and How to Override Scheduling

- If you need to run a job immediately, you can tell PBS to run a job now. You can optionally specify the vnodes and resources to run it. See section 4.9.30.1, “Run a Job Manually”, on page 162.
- If you need to prevent a job from running, you can tell PBS to place a hold on a job. See section 4.9.30.2, “Hold a Job Manually”, on page 163.
- If you need to change how the formula computes job priority, you can make on-the-fly changes to how the formula is computed. See section 4.9.30.5, “Change Formula On the Fly”, on page 164.
- If you need a block of time where you can control what's running, for example for upgrading PBS, you can create dedicated time. See section 4.9.30.6, “Using Dedicated Time”, on page 164.
- If you need to submit jobs at a certain time, you can use `cron` to run jobs. See section 4.9.30.7, “Using cron Jobs”, on page 164.
- If you need to change job resource requests, programs, environment, or attributes, you can use hooks to examine jobs and alter their characteristics. See the *PBS Professional Hooks Guide*.
- If you need to prevent a scheduler from calendaring jobs, you can set their topjob_ineligible attribute to *True*. See section 4.9.17, “Calendaring Jobs”, on page 137.

4.4 Choosing a Policy

4.4.1 Overview of Kinds of Policies

You can tune PBS to produce any of a wide selection in scheduling behaviors. You can choose from a wide variety of behaviors for each sub-goal, resulting in many possible scheduling policies. However, policies can be grouped into the following kinds:

- FIFO, where you essentially run jobs in the order in which they were submitted; see section 4.4.2, “FIFO: Submission Order”, on page 79
- According to user or group priority, where the job's priority is determined by the owner's priority; see section 4.4.3, “Prioritizing Jobs by User, Project or Group”, on page 80
- According to resource allocation rules, where jobs are run so that they use resources following a set of rules for how resources should be awarded to users or groups; see section 4.4.4, “Allocating Resources by User, Project or Group”, on page 80
- According to the size of the job, for example measured by CPU or memory request; see section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 82
- By setting up time slots for specific uses; see section 4.4.6, “Scheduling Jobs into Time Slots”, on page 84

4.4.2 FIFO: Submission Order

If you want jobs to run in the order in which they are submitted, use FIFO. You can use FIFO across the entire partition or complex, or within each queue.

If it's important that jobs run exactly in submission order, use FIFO with strict ordering. However, if you don't want resources to be idle while a top job is stuck, you can use FIFO with strict ordering and backfilling.

To run jobs in submission order, see section 4.9.20.1, “Configuring Basic FIFO Scheduling”, on page 148 .

To run jobs in submission order across the entire partition or complex, see section 4.9.20.2, “FIFO for Entire Partition Or Complex”, on page 148.

To run jobs in submission order, examining queues in order of queue priority, see section 4.9.20.3, “Queue by Queue FIFO”, on page 149.

To run jobs in submission order, with strict ordering, see section 4.9.20.4, “FIFO with Strict Ordering”, on page 149.

To run jobs in submission order, with strict ordering and backfilling, see section 4.9.20.5, “FIFO with Strict Ordering and Backfilling”, on page 149.

4.4.3 Prioritizing Jobs by User, Project or Group

If you need to run jobs from some users, groups, or projects before others, you can prioritize jobs using the following techniques:

- Routing each entity's jobs to its own execution queue, assigning the queue the desired priority, and examining jobs queue by queue. See the following:
  - For routing: section 2.3.6, “Routing Queues”, on page 25
  - For setting queue priority: section 2.3.5.3, “Prioritizing Execution Queues”, on page 24
  - For examining jobs queue by queue: section 4.9.4, “Examining Jobs Queue by Queue”, on page 110
- Routing each entity's jobs to its own execution queue, where the jobs inherit a custom resource that you use in the job sorting formula. See the following:
  - For routing: section 2.3.6, “Routing Queues”, on page 25
  - For inherited resources: section 14.3, “Allocating Resources to Jobs”, on page 505
  - For the job sorting formula: section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149
- Using a hook to allocate a custom resource to each job, where the hook sets the value according to the priority of the job's owner, group, or project, then using the resource in the job sorting formula. See the following:
  - For hooks: the PBS Professional Hooks Guide
  - For custom resources: section 5.14, “Custom Resources”, on page 255
  - For the job sorting formula: section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149
- Assigning a greater fairshare allocation in the fairshare tree to the users or groups whose jobs must run first, and running jobs according to entity shares. See the following:
  - For fairshare: section 4.9.19, “Using Fairshare”, on page 138
  - For entity shares: section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 131

4.4.4 Allocating Resources by User, Project or Group

When you want to divide up hardware usage among users, groups, or projects, you can make sure you allocate resources along those lines. You can do this in the following ways:

- Allocate portions of the entire partition or complex to each entity; see section 4.4.4.1, “Allocating Portions of Partition Or Complex”, on page 81
- Allocate portions of all machines or clusters to each entity, or use controlled allocation for some hardware, with a free-for-all elsewhere; see section 4.4.4.2, “Allocating Portions of Machines or Clusters”, on page 81
- Lock entities into using specific hardware; see section 4.4.4.3, “Locking Entities into Specific Hardware”, on page 82

4.4.4.1 Allocating Portions of Partition Or Complex

4.4.4.1.i Allocating Specific Amounts

To allocate specific amounts of resources across the entire partition or complex, you can use resource limits at the server. These limits set the maximum amount that can be used, ensuring that projects, users, or groups stay within their bounds. You can set a limit for each resource, and make it different for each project, user, and group. You can set a different limit for each project, user, and group, for each resource.

For example, you can set a limit of 48 CPUs in use at once by most groups, but give groupA a limit of 96 CPUs. You can give each individual user a limit of 8 CPUs, but give UserA a limit of 10 CPUs, and UserB a limit of 4 CPUs.

To set limits for usage across the entire partition or complex, set the limits at the server.

See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

4.4.4.1.ii Allocating Percentages

To allocate a percentage of the resources being used in the partition managed by a scheduler, you can use fairshare. Fairshare tracks a moving average of resource usage, so it takes past use into account. You choose which resources to track. You can tune the influence of past usage.

To use fairshare across the entire partition or complex, make sure that both by_queue and round_robin are *False*.

Fairshare is described in section 4.9.19, “Using Fairshare”, on page 138.

4.4.4.2 Allocating Portions of Machines or Clusters

You can allocate fixed amounts of a machine or groups of machines. You can do this for as many machines as you want. For example, on HostA, you can give GroupA 100 CPUs, GroupB 150 CPUs, and GroupC 50 CPUs, while at HostB, GroupA gets 10, GroupB gets 8, and GroupC gets 25.

To allocate fixed portions of a specific machine or group of machines, you use these tools in combination:

- Create an execution queue for this machine; see section 2.3.3, “Creating Queues”, on page 22.
- Route jobs belonging to the users or groups who share this machine into a queue. Each machine or cluster that requires controls gets its own queue. See section 4.9.39, “Routing Jobs”, on page 205.
- Associate the queue with the vnodes in question; see section 4.9.2, “Associating Vnodes with Queues”, on page 103.
- Set a limit at the queue for each resource that you care about, for each project, user, or group. These limits control use of the vnodes associated with the queue only. See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

You can prevent unauthorized usage by setting generic project, user, and group limits for the machine's queue to zero. However, you probably don't want users to submit their jobs to a queue where they are not allowed to run, only to have those jobs languish. You can avoid this by doing the following:

- Setting up a routing queue; see section 2.3.6, “Routing Queues”, on page 25.
- Making the routing queue be the default queue:

  ```
  Qmgr: set server default_queue = <routing queue name>
  ```

- Making the routing queue the only queue that accepts job submission: set from_route_only to *True* on execution queues tied to hardware. See section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 23.
- Using queue access control to limit which jobs are routed into the execution queue; see section 2.3.6.5, “Using Access Control to Route Jobs”, on page 29.

You can either set up allocations for every machine, or you can set up allocations for only some machines, leaving a free-for-all for the others. If you want access to be unrestricted for some machines, do not set limits at the server.

4.4.4.3 Locking Entities into Specific Hardware

You can send all jobs from some projects, users, or groups to designated hardware, essentially limiting them to a sandbox. To do this, do the following:

- Create an execution queue for the sandbox hardware; see section 2.3.3, “Creating Queues”, on page 22.
- Create at least one other execution queue; see section 2.3.3, “Creating Queues”, on page 22.
- Create a routing queue; see section 2.3.3, “Creating Queues”, on page 22.
- Make the routing queue be the default queue:

  ```
  Qmgr: set server default_queue = <routing queue name>
  ```
- Force all users to submit jobs to the routing queue: set from_route_only to *True* on all other queues. See section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 23.
- Use queue access control to route according to user or group: route jobs from the controlled users or groups into the sandbox queue only. See section 2.3.6.5, “Using Access Control to Route Jobs”, on page 29.
- Use a job submission hook to route according to project: route the jobs from the desired project(s) to the sandbox queue. See the PBS Professional Hooks Guide.
- Associate the sandbox queue with the sandbox vnodes. See section 4.9.2, “Associating Vnodes with Queues”, on page 103.

Note that you can either allow all projects, users, or groups into the sandbox queue, or allow only the controlled projects, users, or groups into the sandbox queue.

4.4.5 Scheduling Jobs According to Size Etc.

You may need to treat jobs differently depending on their size or other characteristics. For example, you might want to run jobs differently depending on the number of CPUs or amount of memory requested by the job, or whether the job requests GPUs.

- Give special priority to a group of jobs
- Run a group of jobs on designated hardware
- Run a group of jobs in designated time slots: reservations, dedicated time, and primetime or non-primetime

There are two main approaches to doing this. You can route jobs into queues, or you can use hooks to set values. Here is an outline:

- Route certain kinds of jobs into their own queues, in order to treat each kind differently. This works for priority, hardware, and time slots. See section 4.4.5.1, “Special Treatment via Routing”, on page 82
  - Route each kind to its own queue, using queue-based routing or a submission hook;
  - Use queue-based methods to set job priority or to run the jobs on certain hardware or in certain time slots
- Use hooks to set priority for jobs or to set a custom resource that will send jobs to certain hardware. This does not work for time slots. See section 4.4.5.2, “Special Treatment via Hooks”, on page 84.
  - Use a submission hook to set each job's Priority attribute, or set a value for a custom resource used in the job sorting formula
  - Use a submission hook to set a custom host-level resource value for each job, where the value matches the value at the desired hardware

4.4.5.1 Special Treatment via Routing

Use a routing queue or a hook to route jobs into a special queue, where the jobs are given special priority, or are run on special hardware, or are run in special time slots.

4.4.5.1.i Routing via Queues

- Create your destination queues. See section 2.3.3, “Creating Queues”, on page 22.
- Set limits at the destination queues, so that each queue receives the correct jobs. See section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 26.
- Create a routing queue, and set its destination queues. See section 2.3.6, “Routing Queues”, on page 25.
- Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

4.4.5.1.ii Using Hooks to Route Jobs

You can use a submission hook to move jobs into the queues you want. See section 4.9.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 207.

4.4.5.1.iii Giving Routed Jobs Special Priority

You can give routed jobs special priority in the following ways:

- Have the jobs inherit a custom resource from the special queue, and use this resource in the job sorting formula.
  - For how to have jobs inherit custom resources, see section 14.3, “Allocating Resources to Jobs”, on page 505.
  - For how to use the job sorting formula, see section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.
- Give the queue itself special priority, and use queue priority in the job sorting formula.
  - For how to assign priority to queues, see section 2.3.5.3, “Prioritizing Execution Queues”, on page 24
  - For how to use the job sorting formula, see section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

4.4.5.1.iv Running Jobs on Special Vnodes

Now that the special jobs are routed to a special queue, associate that queue with the special vnodes. See section 4.9.2, “Associating Vnodes with Queues”, on page 103.

4.4.5.1.v Running Jobs in Special Time Slots

If you want to run jobs during dedicated time, route the jobs into one or more dedicated time queues. In the same way, for primetime or non-primetime, route jobs into primetime or non-primetime queues. You can also route jobs into reservation queues for reservations that you have created for this purpose.

For using dedicated time, see section 4.9.10, “Dedicated Time”, on page 125

For using primetime and non-primetime, see section 4.9.34, “Using Primetime and Holidays”, on page 191

For using reservations, see section 4.9.37, “Reservations”, on page 197

4.4.5.2 Special Treatment via Hooks

4.4.5.2.i Setting Job Priority Via Hook

You can set a job's Priority attribute using a hook. Note that users can `qalter` the job's Priority attribute. Use a job submission hook to set the job priority, by doing one of the following:

- Set a custom numeric resource for the job, and use the resource in the job sorting formula
  - For how to use hooks, see the PBS Professional Hooks Guide
  - For how to use the job sorting formula, see section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.
- Set the job's Priority attribute, and sort jobs on a key, where the key is the job's Priority attribute.
  - For how to set job attributes, see the PBS Professional Hooks Guide
  - For how to sort jobs on a key, see section 4.9.45, "Sorting Jobs on a Key", on page 221

4.4.5.2.ii Routing Jobs to Hardware via Hooks

You can send jobs to particular hardware without using a particular queue, by using a hook. See section 4.9.39.4.i, "Using Hooks to Tag Jobs", on page 208.

4.4.6 Scheduling Jobs into Time Slots

You can schedule jobs in time slots in the following ways:

- Set aside time slots for specific entities; see section 4.4.6.1, "Setting Aside Time Slots for Entities", on page 84
- Lock entities into specific time slots; see section 4.4.6.2, "Locking Entities into Time Slots", on page 85

4.4.6.1 Setting Aside Time Slots for Entities

You can set aside time slots that are reserved exclusively for certain users or groups. You can use reservations, dedicated time, primetime, or non-primetime.

4.4.6.1.i Reservations

Reservations set aside one or more blocks of time on the requested resources. Users can create their own reservations, or you can create them and set their access control to allow only specified users to submit jobs to them. See section 4.9.37, "Reservations", on page 197.

4.4.6.1.ii Dedicated Time

During dedicated time, the only jobs allowed to run are those in dedicated queues. The drawback to dedicated time is that it applies to the entire partition or complex. If you want to set aside one or more dedicated time slots for a user or group, do the following:

- Create a dedicated queue. See section 2.3.5.2.i, "Dedicated Time Queues", on page 24.
- Define dedicated time. See section 4.9.10, "Dedicated Time", on page 125.
- Set access control on the dedicated queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.3.6.5, "Using Access Control to Route Jobs", on page 29.
- If you want to limit access on a dedicated queue to a specific project, set the generic limit for queued jobs for projects at that queue to zero, and then set the individual limit for the specific project higher.

4.4.6.1.iii Non-primetime

You can set up primetime and non-primetime so that one of them, for example, non-primetime, is used as a special time slot allocated to particular users or groups. The advantage of using non-primetime is that you can set up a separate scheduling policy for it, for example, using fairshare during non-primetime and sorting jobs on a key during primetime. Note that the formula, if defined, is in force all of the time. To use non-primetime, do the following:

- Create a non-primetime queue; see section 2.3.3, "Creating Queues", on page 22 and section 2.3.5.2.ii, "Primetime and Non-Primetime Queues", on page 24.
- Define primetime and non-primetime; see section 4.9.34, "Using Primetime and Holidays", on page 191.
- Set access control on the non-primetime queue so that only the particular users or groups you want can submit jobs to the queue. See section 2.3.6.5, "Using Access Control to Route Jobs", on page 29.
- Make sure that the scheduling policy you want is in force during non-primetime. See section 4.9.34.1, "How Primetime and Holidays Work", on page 192.

4.4.6.2 Locking Entities into Time Slots

You can make all jobs from some users or groups run during designated time slots. You can run them during a reservation, dedicated time, or non-primetime.

4.4.6.2.i Locking Entities into Reservations

To allow a user to submit jobs only into a reservation, do the following:

- Create a reservation for the resources and time(s) you want the controlled user(s) to use. When creating the reservation, set access control to allow the controlled user(s). See section 4.9.37, "Reservations", on page 197 and section 8.3.8.3, "Setting and Changing Reservation Access", on page 372.
- Set access control on all queues except the reservation's queue to deny the controlled user(s); see section 2.3.6.5, "Using Access Control to Route Jobs", on page 29.

4.4.6.2.ii Locking Entities into Dedicated Time

You can create a dedicated time queue, and send all jobs from controlled projects, users, or groups to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more projects, users, or groups into one or more dedicated time slots, do the following:

- Create a dedicated time queue; see section 2.3.3, "Creating Queues", on page 22 and section 2.3.5.2.i, "Dedicated Time Queues", on page 24.
- Create at least one other execution queue; see section 2.3.3, "Creating Queues", on page 22.
- Create a routing queue; see section 2.3.3, "Creating Queues", on page 22.
- Prevent controlled users from submitting to non-dedicated time execution queues: set from_route_only to *True* on the non-dedicated time execution queues. See section 2.3.5.1, "Where Execution Queues Get Their Jobs", on page 23.
- Use queue access control to allow jobs from the controlled users or groups into the dedicated time queue only. See section 2.3.6.5, "Using Access Control to Route Jobs", on page 29
- Use a job submission hook to route jobs from controlled projects into the dedicated time queue. See the PBS Professional Hooks Guide
- .Make the routing queue be the default queue:

```
Qmgr: set server default_queue = <routing queue name>
```

Note that you can either allow all users into the dedicated time queue, or allow only the controlled users into the dedicated time queue.

4.4.6.2.iii Locking Entities into Non-primetime

You can create a non-primetime queue, and send all jobs from controlled users, groups, or projects to that queue. You can route their jobs to it, and you can allow them to submit directly to it. To lock one or more users, groups, or projects into one or more non-primetime slots, do the following:

- Create a non-primetime queue; see section 2.3.3, “Creating Queues”, on page 22 and section 2.3.5.2.ii, “Primetime and Non-Primetime Queues”, on page 24.
- Create at least one other execution queue; see section 2.3.3, “Creating Queues”, on page 22.
- Create a routing queue; see section 2.3.3, “Creating Queues”, on page 22.
- Prevent controlled users from submitting to primetime execution queues: set from_route_only to *True* on the primetime execution queues. See section 2.3.5.1, “Where Execution Queues Get Their Jobs”, on page 23.
- Make the routing queue be the default queue:

  ```
  Qmgr: set server default_queue = <routing queue name>
  ```
- Use queue access control to allow jobs from the controlled users or groups into the non-primetime queue only. See section 2.3.6.5, “Using Access Control to Route Jobs”, on page 29.
- Use a job submission hook to route jobs from controlled projects into the non-primetime queue. See the PBS Professional Hooks Guide
- Define primetime and non-primetime; see section 4.9.34, “Using Primetime and Holidays”, on page 191.
- Make sure that the scheduling policy you want is in force during non-primetime. See section 4.9.34.1, “How Primetime and Holidays Work”, on page 192.

Note that you can either allow all users into the non-primetime queue, or allow only the controlled users into the non-primetime queue.

4.4.7 Default Scheduling Policy

The default scheduling policy is determined by the default settings for all of the attributes, parameters, etc. that determine a scheduler's behavior. For a list of all of these elements, see section 4.5.1, “Configuring a Scheduler”, on page 89.

The default behavior of a scheduler is the following:

- A scheduler matches jobs with available resources. This means that a scheduler places each job only where that job has enough resources to run. See section 4.9.28, “Matching Jobs to Resources”, on page 159.
- A scheduler will not over-allocate the resources that are listed in the scheduler's resources parameter. The defaults for these are ncpus, mem, arch, host, vnode, aoe. See section 4.9.28.1, “Scheduling on Consumable Resources”, on page 159.
- A scheduler sorts vnodes according to its node_sort_key parameter, whose default setting is the following:

  ```
  node_sort_key: "sort_priority HIGH all"
  ```

  This means that vnodes are sorted by the value of their priority attribute, with high-priority vnodes used first. A scheduler places jobs first on vnodes that are first in the sorted list.

  Note that all vnodes have the same default priority upon creation, so the default sorted order for vnodes is undefined.

See section 4.9.50, “Sorting Vnodes on a Key”, on page 226.

- Queues are sorted according to the value of their priority attribute, so that queues with a higher priority are considered before those with a lower priority. See section 2.3.5.3, “Prioritizing Execution Queues”, on page 24.
- Jobs are considered according to the priority of their queues. A scheduler runs all of the jobs that it can from the highest-priority queue before moving to the next queue, and so on. See section 4.9.4, “Examining Jobs Queue by Queue”, on page 110.
- Within each queue, jobs are considered in submission order.
- Starving jobs are given a special priority called *starving*. The default time required to become a starving job is 24 hours. See section 4.9.48, “Starving Jobs”, on page 223.
- Jobs in an express queue are placed in the *express_queue* preemption priority level. They are also placed in the *Express* execution priority class. The default priority for a queue to be an express queue is 150. See section 2.3.5.3.i, “Express Queues”, on page 24.
- Queued jobs are sorted according to their priority. Special jobs are all prioritized ahead of normal jobs, without regard to the queue in which they reside. The order for job priority for special jobs, highest first, is reservation jobs, jobs in express queues, preempted jobs, starving jobs. After this, a scheduler looks at normal jobs, queue by queue. All jobs in express queues, all preempted jobs, and all starving jobs are considered before a scheduler looks at the individual queues.

  See section 4.9.16, “Calculating Job Execution Priority”, on page 134.
- A scheduler will preempt lower-priority jobs in order to run higher-priority jobs (preemptive_sched is *True* by default). By default, it has two levels of job priority, *express_queue*, and *normal_jobs*, where *express_queue* jobs can preempt *normal_jobs*. This is set in the scheduler's preempt_prio attribute.

  When a scheduler chooses among jobs of the same priority for a job to preempt, it uses the only setting for preempt_sort, which is *min_time_since_start*, choosing jobs that have been running for the shortest time.

  When a scheduler chooses how to preempt a job, it uses the default setting for its preempt_order attribute, which is *SCR*, meaning that first it will attempt suspension, then checkpointing, then if necessary requeueing.

  See section 4.9.33, “Using Preemption”, on page 180.
- A scheduler will do its best to backfill smaller jobs around the job it has decided is the most important job. See section 4.9.3, “Using Backfilling”, on page 105.
- Primetime by default is 24/7. Any holiday is considered non-primetime. You can define primetime and holidays in the file `<sched_priv directory>/holidays`. These dates should be adjusted yearly to reflect your local holidays. See section 4.9.34, “Using Primetime and Holidays”, on page 191.
- A scheduler runs every 10 minutes unless a new job is submitted or a job finishes execution. See section 4.5.5, “The Scheduling Cycle”, on page 96.
- In TPP mode, a scheduler runs with the throughput_mode scheduler attribute set to *True* by default, so the scheduler runs asynchronously, and doesn't wait for each job to be accepted by MoM, which means it also doesn't wait for an execjob_begin hook to finish. Especially for short jobs, this can give better scheduling performance.

  When throughput_mode is *True*, jobs that have been changed can run in the same scheduling cycle in which they were changed, for the following changes:

  - Jobs that are qaltered
  - Jobs that are changed via server_dyn_res scripts
  - Jobs that are peered to a new queue

  See “Scheduler Attributes” on page 298 of the PBS Professional Reference Guide.

4.4.8 Examples of Workload and Policy

- If you need to have high-priority jobs run soon, and nothing distinguishes the high-priority jobs from the rest:
  - Create advance reservations for the high-priority jobs, and have users submit those jobs to the reservations; see section 4.9.37, “Reservations”, on page 197
- If you want to run jobs in submission order:
  - FIFO; see section 4.9.20, “FIFO Scheduling”, on page 148
- If you have low-priority jobs that should run only when other jobs don't need the resources:
  - Set up an anti-express queue; see section 4.9.1, “Anti-Express Queues”, on page 102
- If you have a mix of jobs, and want to run big jobs first:
  - Sort jobs on a key, using ncpus as the key, to run big jobs first; see section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 82
- If you have a mix of jobs, and want to give big jobs high priority, but avoid having idle resources:
  - Sort jobs on a key, using ncpus as the key, to run big jobs first; see section 4.4.5, “Scheduling Jobs According to Size Etc.”, on page 82
  - Use backfilling; see section 4.9.3, “Using Backfilling”, on page 105
- If you want to have all users start about the same number of jobs:
  - Use round robin, give each user their own queue, and give each queue the same priority; see section 4.9.38, “Round Robin Queue Selection”, on page 204
- If you want to always give each user access to a certain amount of a resource, but allow more if no one else is using it:
  - Use soft limits for the amount each user can use; see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287 and section 4.9.33, “Using Preemption”, on page 180
- If your partition or site has more than one funding source:
  - See section 4.4.4, “Allocating Resources by User, Project or Group”, on page 80
- If you have lots of users in a partition or complex:
  - Use resource limits; see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287, or
  - Use fairshare; see section 4.9.19, “Using Fairshare”, on page 138
- If you have jobs that must run at the end of the day:
  - Use dependencies for end-of-day accounting; see section 4.9.11, “Dependencies”, on page 126
- If you need to ensure that jobs run in certain hours on desktops:
  - Use cycle harvesting; see section 4.9.9, “Using Idle Workstation Cycle Harvesting”, on page 115, or
  - Use primetime & non-primetime for nighttime; see section 4.9.34, “Using Primetime and Holidays”, on page 191
- If you want to be sure a job will run:
  - Create an advance reservation; see section 4.9.37, “Reservations”, on page 197
- If you have more than one partition or complex, and you want to balance the workload across the partitions or complexes:
  - Use peer scheduling; see section 4.9.31, “Peer Scheduling”, on page 165
- If you have some jobs that should prefer to run on one set of vnodes, and other jobs that should prefer to run on another set of vnodes, but if the preferred vnodes are busy, a job can run on the non-preferred vnodes:
  - Use peer scheduling. Set up two partitions or complexes, give the pulling queues low priority, and use queue

priority in the job sorting formula. See section 4.9.31, "Peer Scheduling", on page 165, section 2.3.5.3, "Prioritizing Execution Queues", on page 24, and section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149. You can use a routing queue to initially send jobs to the correct partition or complex. See section 2.3.6, "Routing Queues", on page 25

- If you have two (or more) sets of vnodes, and jobs should run on one set or the other, but not both. Additionally, jobs should not have to request where they run. For example, one set of vnodes is new, and one is old:
  - Use a routing queue and two execution queues. Associate each execution queue with one set of vnodes. Put the execution queue for the preferred set of vnodes first in the routing list, but put a limit on the number of queued jobs in the execution queues, so that both queues will fill up. Otherwise the routing queue will preferentially fill the first in its routing list. See section 2.3.6, "Routing Queues", on page 25, and section 4.9.2, "Associating Vnodes with Queues", on page 103
- If you need to apportion a single vnode or cluster according to ownership:
  - See section 4.4.4, "Allocating Resources by User, Project or Group", on page 80
- If you have more than one high-priority queue, and at least one low-priority queue, and you want all jobs in high-priority queues to be considered as one group, and run in submission order:
  - Use the job sorting formula to sort jobs on queue priority:

    ```
    set server job_sort_formula = queue_priority
    ```
  - Give all queues whose jobs should be considered together the same priority
  - Set the by_queue scheduler attribute to *False*
- If you want to place jobs on the vnodes with the fewest CPUs first, saving bigger vnodes for larger jobs:
  - Sort vnodes so that those with fewer CPUs come first:

    ```
    node_sort_key: "ncpus LOW"
    ```

4.5 About Schedulers

Each scheduler, pbs_sched, implements its own scheduling policy.

4.5.1 Configuring a Scheduler

4.5.1.1 Where a Scheduler Gets Its Information

Each scheduler has its own sched_priv directory, where it keeps scheduler-specific files. For a multisched, this is $PBS_HOME/sched_priv_<scheduler name>; for the default scheduler, it is always $PBS_HOME/sched_priv/.

The behavior of a scheduler is controlled by the information provided by the following sources:

PBS_est

Hook that runs estimator process which calculates estimated start time and vnodes for jobs. See section 4.9.15, "Estimating Job Start Time", on page 131.

<sched_priv directory>/resource_group

Contains the description of the fairshare tree. Created by you. Can be edited. Read on startup and HUP of scheduler.

<sched_priv directory>/usage

Contains the usage database. Do not edit. Instead, use the pbsfs command while a scheduler is stopped; see "pbsfs" on page 31 of the PBS Professional Reference Guide.

<sched_priv directory>/sched_config

Contains scheduler configuration options, also called scheduler parameters, e.g. fairshare_decay_time, job_sort_key. Read on startup and HUP.

Can be edited. Each entry must be a single, unbroken line. Entries must be double-quoted if they contain whitespace.

See "Scheduler Parameters" on page 249 of the PBS Professional Reference Guide.

<sched_priv directory>/dedicated_time

Contains definitions of dedicated time. Can be edited. Read on startup and HUP.

<sched_priv directory>/holidays

Where you define primetime, non-primetime, and holidays. Can be edited. Read on startup and HUP.

Options to `pbs_sched` command

Control some scheduler behavior. Set on invocation. See "pbs_sched" on page 105 of the PBS Professional Reference Guide.

Scheduler attributes

Control some scheduler behavior. Can be set using `qmgr`. Read every scheduling cycle. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.

Server attributes

Several server attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the server attributes that affect scheduling, along with a brief description. Read every scheduling cycle.

Some limit attributes are marked as "old". These are incompatible with, and are replaced by, the new limit attributes described in section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

For a complete description of each attribute, see "Server Attributes" on page 281 of the PBS Professional Reference Guide.

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
backfill_depth	Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around.
default_queue	Specifies queue for jobs that don't request a queue
eligible_time_enable	Controls starving behavior.
est_start_time_freq	**Obsolete**. Not used. Interval at which PBS calculates estimated start times and vnodes for all jobs.
job_sort_formula	Formula for computing job priorities.
max_group_res	Old. The maximum amount of the specified resource that any single group may consume in this PBS complex.
max_group_res_soft	Old. The soft limit for the specified resource that any single group may consume in this complex.

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
max_group_run	Old. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.
max_group_run_soft	Old. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time.
max_queued	The maximum number of jobs allowed to be queued or running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_queued_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs queued or running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run	The maximum number of jobs allowed to be running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_res_soft.<resource name>	Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_run_soft	Soft limit on the number of jobs allowed to be running in the partition managed by a scheduler. Can be specified for users, groups, or all.
max_running	Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.
max_user_res	Old. The maximum amount within this complex that any single user may consume of the specified resource.
max_user_res_soft	Old. The soft limit on the amount of the specified resource that any single user may consume within a complex.
max_user_run	Old. The maximum number of jobs owned by a single user allowed to be running within the partition managed by a scheduler at one time.
max_user_run_soft	Old. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time.
node_fail_requeue	Controls whether running jobs are automatically requeued or are deleted when the primary execution host fails. Number of seconds to wait after losing contact with the primary execution host MoM before requeueing or deleting jobs. See "node_fail_requeue" on page 290 of the PBS Professional Reference Guide.
node_group_enable	Specifies whether node grouping is enabled.
node_group_key	Specifies the resource to use for node grouping.

Table 4-1: Server Attributes Involved in Scheduling

Attribute	Effect
resources_available	The list of available resources and their values defined on the server.
resources_max	The maximum amount of each resource that can be requested by any single job in this complex, if there is not a resources_max value defined for the queue at which the job is targeted.
scheduler_iteration **deprecated**	The time between scheduling iterations.
scheduling **deprecated**	Enables scheduling of jobs.
resources_assigned	The total of each type of resource allocated to jobs running and exiting in this complex, plus the total of each type of resource allocated to any started reservations.

Vnode attributes

Several vnode attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the vnode attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see "Vnode Attributes" on page 320 of the PBS Professional Reference Guide.

Table 4-2: Vnode Attributes Involved in Scheduling

Attribute	Effect
current_aoe	This attribute identifies the AOE currently instantiated on this vnode.
no_multinode_jobs	Controls whether jobs which request more than one chunk are allowed to execute on this vnode.
partition	The partition to which this vnode is assigned.
pcpus	The number of physical CPUs on the vnode.
priority	The priority of this vnode compared with other vnodes.
provision_enable	Controls whether this vnode can be provisioned.
queue **deprecated**	The queue with which this vnode is associated.
resources_assigned	The total amount of each resource allocated to running and exiting jobs and started reservations running on this vnode.
resources_available	The list of resources and the amounts available on this vnode
sharing	Specifies whether more than one job at a time can use the resources of the vnode or the vnode's host.
state	Shows or sets the state of the vnode.

Queue attributes

Several queue attributes control scheduler behavior. Can be set using `qmgr`. The following table lists the queue attributes that affect scheduling, along with a brief description. Read every scheduling cycle. For a complete description of each attribute, see "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

Table 4-3: Queue Attributes Involved in Scheduling

Attribute	Effect
backfill_depth	Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around.
enabled	Specifies whether this queue accepts new jobs.
from_route_only	Specifies whether this queue accepts jobs only from routing queues.
max_array_size	The maximum number of subjobs that are allowed in an array job.
max_group_res	Old. The maximum amount of the specified resource that any single group may consume in this queue.
max_group_res_soft	Old. The soft limit for the specified resource that any single group may consume in this queue.
max_group_run	Old. The maximum number of jobs owned by the users in one group allowed to be running within this queue at one time.
max_group_run_soft	Old. The maximum number of jobs owned by the users in one group allowed to be running in this queue at one time.
max_queuable	Old. The maximum number of jobs allowed to reside in the queue at any given time.
max_queued	The maximum number of jobs allowed to be queued in or running from the queue. Can be specified for users, groups, or all.
max_queued_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from the queue. Can be specified for users, groups, or all.
max_run	The maximum number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.
max_run_res.<resource name>	The maximum amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.
max_run_res_soft.<resource name>	Soft limit on the amount of the specified resource allowed to be allocated to jobs running from the queue. Can be specified for users, groups, or all.
max_run_soft	Soft limit on the number of jobs allowed to be running from the queue. Can be specified for users, groups, or all.
max_running	Old. The maximum number of jobs allowed to be selected for execution at any given time, from all possible jobs.

Table 4-3: Queue Attributes Involved in Scheduling

Attribute	Effect
max_user_res	Old. The maximum amount of the specified resource that the jobs of any single user may consume.
max_user_res_soft	Old. The soft limit on the amount of the specified resource that any single user may consume in this queue.
max_user_run	Old. The maximum number of jobs owned by a single user allowed to be running from the queue at one time.
max_user_run_soft	Old. The soft limit on the number of jobs owned by a single user that are allowed to be running from this queue at one time.
node_group_key	Specifies the resource to use for node grouping.
Priority	The priority of this queue compared to other queues of the same type in this PBS partition or complex.
resources_assigned	The total of each type of resource allocated to jobs running and exiting in this queue.
resources_available	The list of available resources and their values defined on the queue.
resources_max	The maximum amount of each resource that can be requested by any single job in this queue.
resources_min	The minimum amount of each resource that can be requested by a single job in this queue.
route_destinations	The list of destinations to which jobs may be routed.
route_held_jobs	Specifies whether jobs in the held state can be routed from this queue.
route_lifetime	The maximum time a job is allowed to reside in a routing queue. If a job cannot be routed in this amount of time, the job is aborted.
route_retry_time	Time delay between routing retries. Typically used when the network between servers is down.
route_waiting_jobs	Specifies whether jobs whose execution_time attribute value is in the future can be routed from this queue.
started	Specifies whether jobs in this queue can be scheduled for execution.
state_count	The number of jobs in each state currently residing in this queue.

List of jobs and server-level resources queried from server

Read every scheduling cycle.

Resources in Resource_List job attribute

Read every scheduling cycle.

List of host-level resources queried from MoMs

Read every scheduling cycle.

4.5.1.2 Reference Copies of Files

PBS is installed with a reference copy of the holidays file in which everything is commented out, in `PBS_EXEC/etc/pbs_holidays`.

4.5.2 Making a Scheduler Read its Configuration

If you change a scheduler's configuration file, the scheduler must re-read it for the changes to take effect. To get a scheduler to re-read its configuration information, without stopping the scheduler, you can HUP the scheduler:

```
kill -HUP <scheduler PID>
```

If you set a scheduler attribute using `qmgr`, the change takes effect immediately and you do not need to HUP the scheduler.

4.5.3 Scheduling on Resources

A scheduler honors all resources listed in the `resources:` line in `<sched_priv directory>/sched_config`. If this line is not present, a scheduler honors all resources, built-in and custom. It is more efficient to list just the resources that you want a scheduler to schedule on.

4.5.4 Starting, Stopping, and Restarting a Scheduler

4.5.4.1 When and How to Start a Scheduler

During normal operation, startup of the scheduler is handled automatically. The PBS daemons are started automatically at bootup by the PBS start/stop script. During failover, the secondary server automatically tries to use the primary scheduler, and if it cannot, it starts its own scheduler.

To start the default scheduler by hand:

```
PBS_EXEC/sbin/pbs_sched [options]
```

See "pbs_sched" on page 105 of the PBS Professional Reference Guide.

For how to start a multisched, see section 4.2.2, "Starting a Multisched", on page 57.

4.5.4.2 When and How to Stop a Scheduler

You must stop a scheduler for the following operations:

- (Recommended) Using the `pbsfs` command; see "pbsfs" on page 31 of the PBS Professional Reference Guide.
- Upgrading PBS Professional; see "Upgrading" on page 65 in the PBS Professional Installation & Upgrade Guide.

A scheduler traps signals during the scheduling cycle. You can kill a scheduler at the end of the cycle, or if necessary, immediately. A scheduler does not write the fairshare usage file when it is killed with -9, but it does write the file when it is killed without -9.

You must be root on the scheduler's host.

To stop a scheduler at the end of a cycle:

```
kill <scheduler PID>
```

To stop a scheduler immediately:

```
kill -9 <scheduler PID>
```

4.5.4.3 When and How to Restart a Scheduler

Under most circumstances, when you restart a scheduler, you do not need to specify any options to the pbs_sched command. See "pbs_sched" on page 105 of the PBS Professional Reference Guide. Start a scheduler this way:

```
PBS_EXEC/sbin/pbs_sched [options]
```

4.5.5 The Scheduling Cycle

A scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in each scheduler's scheduler_iteration server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in each scheduler's sched_cycle_length attribute. A scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes. A scheduler does not include the time it takes to query dynamic resources in its cycle measurement.

4.5.5.1 Triggers for Scheduling Cycle

A scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded
- A hook restarts the scheduling cycle

4.5.5.1.i Logging Scheduling Triggers

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 15.3.4.2, "Scheduler Commands", on page 550.

4.5.5.2 Actions During Scheduling Cycle

The following is a list of a scheduler's actions during a scheduling cycle. The list is not in any special order.

- A scheduler gets the state of the world:
  - A scheduler queries the server for the following:
    - Status of jobs in queues
    - All global server, queue, and host-level resources
    - Server, queue, vnode, and scheduler attribute settings
    - Reservations
  - A scheduler runs dynamic server resource queries for resources listed in the "server_dyn_res" line in `sched_config`
  - A scheduler runs dynamic host-level resource queries for resources listed in the "mom_resources" line (**deprecated** as of 18.2.1) in `sched_config`
- A scheduler logs a message at the beginning of each scheduling cycle saying whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is of this form:

  `"It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"`

  or

  `"It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"`
- Given scheduling policy, available jobs and resources, and scheduling cycle length, a scheduler examines as many jobs as it can, and runs as many jobs as it can.

4.5.6 How Available Consumable Resources are Counted

When a scheduler checks for available consumable resources, it uses the following calculation:

resouces_available.<resource name> - total resources assigned for this resource

total resources assigned is the total amount of resources_assigned.<resource name> for all other running and exiting jobs and, at the server and vnodes, for started reservations.

For example, if a scheduler is calculating available memory, and two other jobs are running, each with 2GB of memory assigned, and resources_available.mem is 8GB, the scheduler figures that it has 4GB to work with.

4.5.7 Improving Scheduler Performance

4.5.7.1 Improving Throughput of Jobs

You can tell a scheduler to run asynchronously, so it doesn't wait for each job to be accepted by MoM, which means it also doesn't wait for an execjob_begin hook to finish. For short jobs, this can give you better scheduling performance. To run a scheduler asynchronously, set the scheduler's throughput_mode attribute to *True* (this attribute is *True* by default).

When throughput_mode is *True*, jobs that have been changed can run in the same scheduling cycle in which they were changed, for the following changes:

- Jobs that are qaltered (for example, in `cron` jobs)
- Jobs that are changed via server_dyn_res scripts
- Jobs that are peered to a new queue

throughput_mode

Scheduler attribute. When set to *True*, this scheduler runs asynchronously and can start jobs faster. Only available when complex is in TPP mode.

Format: *Boolean*

Default: *True*

Example:

```
qmgr -c "set sched throughput_mode=<Boolean value>"
```

You can run a scheduler asynchronously only when the complex is using TPP mode. For details about TPP mode, see "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide. Trying to set the value to a non-Boolean value generates the following error message:

```
qmgr obj= svr=default: Illegal attribute or resource value
qmgr: Error (15014) returned from server
```

4.5.7.2 Limiting Number of Jobs Queued in Execution Queues

If you limit the number of jobs queued in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the queued_jobs_threshold queue or server attribute. See section 5.15.1.9, "How to Set Limits at Server and Queues", on page 296.

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See section 4.9.39, "Routing Jobs", on page 205.

4.5.7.3 Setting Number of Scheduler Threads

By default, each scheduler starts one thread on its host. You can modify the number of threads a scheduler starts, either by starting the scheduler with `pbs_sched -t <num threads>`, or by setting the PBS_SCHED_THREADS configuration parameter in `pbs.conf`, or the PBS_SCHED_THREADS environment variable. The `pbs_sched -t` option overrides the environment variable, which overrides the value in `pbs.conf`.

4.6 Using Queues in Scheduling

A queue is a PBS mechanism for holding jobs. PBS has queue-based tools for handling jobs; for example, you can set queue-based limits on resource usage by jobs. PBS uses queues for a variety of purposes. Before reading this section, please familiarize yourself with the mechanics of creating and configuring queues, by reading section 2.3, "Queues", on page 21.

Queues are used in the following ways:

- Holding submitted jobs
- Prioritizing jobs and ordering job selection:
  - PBS provides tools for selecting jobs according to the queue they are in; see section 4.3.5.3, “Using Queue-based Tools to Prioritize Jobs”, on page 65
  - Queue priority can be used in calculating job priority; see section 4.9.36, “Queue Priority”, on page 196
- Providing tools for managing time slots
  - Reservations: you can reserve specific resources for defined time slots. Queues are used for advance and standing reservations; see section 4.9.37, “Reservations”, on page 197, and "Reserving Resources", on page 135 of the PBS Professional User’s Guide
  - Dedicated time; see section 4.9.10, “Dedicated Time”, on page 125
  - Primetime and holidays; see section 4.9.34, “Using Primetime and Holidays”, on page 191
- Routing jobs: Many ways to route jobs are listed in section 4.9.39, “Routing Jobs”, on page 205
- Providing tools for managing resources
  - Managing resource usage by users; see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287
  - Managing resource usage by jobs; see section 5.15.2, “Placing Resource Limits on Jobs”, on page 304
  - Setting resource and job limits used for preemption: you can specify how much of a resource or how many jobs a user or group can use before their jobs are eligible to be preempted. See section 5.15.1.4, “Hard and Soft Limits”, on page 290 and section 4.9.33, “Using Preemption”, on page 180.
  - Assigning default resources to jobs; see section 5.9.4, “Allocating Default Resources to Jobs”, on page 247

4.7 Scheduling Restrictions and Caveats

4.7.1 One Policy Per Scheduler

Each scheduler runs a single scheduling policy.

4.7.2 Jobs that Cannot Run on Current Resources

A scheduler checks to see whether each job could possibly run now, counting resources as if there were no other jobs, and all current resources could be used by this job. A scheduler counts resources only from those vnodes that are on line. If a vnode is marked *offline*, its resources are not counted.

A scheduler determines whether a job cannot run on current resources only when backfilling is used. If backfilling is turned off, then a scheduler won't determine whether or not a job has requested more than can be supplied by current resources. It decides only that it can't run now. If the job cannot run now because vnodes are unavailable, there is no log message. If the job requests more than is available in the partition managed by a scheduler, there is a log message. In both cases, the job stays queued.

4.7.3 Resources Not Controlled by PBS

When a scheduler runs each cycle, it gets the state of its world, including dynamic resources outside of the control of PBS. If non-PBS processes are running on the vnodes PBS uses, it is possible that another process will use enough of a dynamic resource such as scratch space to prevent a PBS job that requested that resource from running.

4.7.4 No Pinning of Processes to Cores

PBS does not pin processes to cores. This can be accomplished in the job launch script using, for example, `taskset` or `dplace`.

4.8 Errors and Logging

4.8.1 Logfile for scheduler

You can set a scheduler's logging to record different kinds of events. See section 15.3.3.1.iii, "Specifying Scheduler Log Events", on page 548.

The server triggers scheduler cycles. The reason for triggering a scheduling cycle is logged by the server. See section 15.3.4.2, "Scheduler Commands", on page 550.

4.9 Scheduling Tools

In this section (all of section 4.9, "Scheduling Tools", on page 100, and its subsections), we describe each scheduling tool, including how to configure it.

The following table lists PBS scheduling tools, with links to descriptions:

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Anti-express queue	soft queue limits	See section 4.9.1, "Anti-Express Queues", on page 102
Associating vnodes with queues		See section 4.9.2, "Associating Vnodes with Queues", on page 103
Backfilling	fairshare or preemption w/backfilling+strict ordering	See section 4.9.3, "Using Backfilling", on page 105
Examining jobs queue-by-queue	round robin, queues as fair-share entities	See section 4.9.4, "Examining Jobs Queue by Queue", on page 110
Checkpointing		See section 4.9.5, "Checkpointing", on page 111
Organizing job chunks		See section 4.9.6, "Organizing Job Chunks", on page 112
`cron` jobs		See section 4.9.7, "cron Jobs", on page 112
Custom resources		See section 4.9.8, "Using Custom and Default Resources", on page 113
Cycle harvesting	reservations	See section 4.9.9, "Using Idle Workstation Cycle Harvesting", on page 115
Dedicated time		See section 4.9.10, "Dedicated Time", on page 125
Default resources		See section 4.9.8, "Using Custom and Default Resources", on page 113

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Dependencies		See section 4.9.11, "Dependencies", on page 126
Dynamic resources (server & host)		See section 4.9.12, "Dynamic Resources", on page 126
Eligible wait time for jobs		See section 4.9.13, "Eligible Wait Time for Jobs", on page 126
Entity shares (was strict priority)	formula, fairshare, FIFO	See section 4.9.14, "Sorting Jobs by Entity Shares (Was Strict Priority)", on page 131
Estimating job start time		See section 4.9.15, "Estimating Job Start Time", on page 131
Calculating job execution priority		See section 4.9.16, "Calculating Job Execution Priority", on page 134
Express queues		See section 4.9.18, "Express Queues", on page 137
Fairshare	starving, strict ordering, using the fairshare_perc option to job_sort_key	See section 4.9.19, "Using Fairshare", on page 138
FIFO		See section 4.9.20, "FIFO Scheduling", on page 148
Formula		See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149
Gating jobs at server or queue		See section 4.9.22, "Gating Jobs at Server or Queue", on page 155
Managing application licenses		See section 4.9.23, "Managing Application Licenses", on page 155
Limits on per-job resource usage		See section 4.9.24, "Limits on Per-job Resource Usage", on page 156
Limits on project, user, and group jobs		See section 4.9.25, "Limits on Project, User, and Group Jobs", on page 156
Limits on project, user, and group resource usage		See section 4.9.26, "Limits on Project, User, and Group Resource Usage", on page 156
Load balancing	node_sort_key using unused or assigned options,	See section 4.9.27, "Using Load Balancing", on page 156
Matching jobs to resources		See section 4.9.28, "Matching Jobs to Resources", on page 159
Node grouping		See section 4.9.29, "Node Grouping", on page 162
Overrides		See section 4.9.30, "Overrides", on page 162
Peer scheduling		See section 4.9.31, "Peer Scheduling", on page 165
Placement sets		See section 4.9.32, "Placement Sets", on page 168
Preemption	cgroups hook cannot be used with suspend/resume	See section 4.9.33, "Using Preemption", on page 180

Table 4-4: List of Scheduling Tools

Scheduling Tool	Incompatible Tools	Link
Preemption targets		See section 4.9.33.4, "Using Preemption Targets", on page 182
Primetime and holidays		See section 4.9.34, "Using Primetime and Holidays", on page 191
Provisioning		See section 4.9.35, "Provisioning", on page 196
Queue priority		See section 4.9.36, "Queue Priority", on page 196
Advance and standing reservations	cycle harvesting	See section 4.9.37, "Reservations", on page 197
Round robin queue examination	by_queue	See section 4.9.38, "Round Robin Queue Selection", on page 204
Routing jobs		See section 4.9.39, "Routing Jobs", on page 205
Shared or exclusive vnodes and hosts		See section 4.9.41, "Shared vs. Exclusive Use of Resources by Jobs", on page 210
Shrinking jobs to fit		See section 4.9.42, "Using Shrink-to-fit Jobs", on page 211
SMP cluster distribution	avoid_provision	See section 4.9.43, "SMP Cluster Distribution", on page 218
Soft walltime		See section 4.9.44, "Using Soft Walltime", on page 219.
Sorting jobs using job_sort_key		See section 4.9.45, "Sorting Jobs on a Key", on page 221
Sorting jobs on job's requested priority		See section 4.9.46, "Sorting Jobs by Requested Priority", on page 223
Sorting queues (**deprecated** in 13.0)		See section 4.9.47, "Sorting Queues into Priority Order", on page 223
Starving jobs	fairshare	See section 4.9.48, "Starving Jobs", on page 223
Strict ordering	Backfilling combined with fairshare	See section 4.9.49, "Using Strict Ordering", on page 225
Sorting vnodes on a key	smp_cluster_dist set to other than pack, or load balancing, with unused or assigned options to node_sort_key	See section 4.9.50, "Sorting Vnodes on a Key", on page 226

4.9.1 Anti-Express Queues

An anti-express queue is a preemptable low-priority queue, designed for jobs that should run only when no other jobs need the resources. These jobs are preempted if any other job needs the resources. An anti-express queue has the lowest priority of all queues in this queue's partition. Jobs in this queue have a soft limit of zero, so that any job running from this queue is over its queue soft limit.

See section 4.9.33, “Using Preemption”, on page 180.

4.9.1.1 Configuring Anti-express Queues via Priority

To configure an anti-express queue by using queue priority, do the following:

- Create an execution queue called *lowprio*:

```
Qmgr: create queue lowprio
Qmgr: set queue lowprio queue_type=e
Qmgr: set queue lowprio started=true
Qmgr: set queue lowprio enabled=true
```

- By default, all new queues have a priority of zero. Make sure all queues have a value set for priority, and that lowprio has the lowest priority:

```
Qmgr: set queue workq priority=10
```

- Set the soft limit on the number of jobs that can run from that queue to zero for all users:

```
Qmgr: set queue lowprio max_run_soft = "[u:PBS_GENERIC=0]"
```

- Make sure that jobs over their queue soft limits have lower preemption priority than normal jobs. Edit `<sched_priv directory>/sched_config`, and do the following:
  - Put "normal_jobs" before "queue_softlimits". For example:

```
preempt_prio: "express_queue, normal_jobs, queue_softlimits"
```

  - Use preemption:

```
preemptive_sched: True ALL
```

4.9.1.2 Configuring Anti-express Queues via Preemption Targets

To use preemption targets, include this queue in Resource_List.preempt_targets for all jobs. You can do this with a hook, with server and/or queue defaults, or by qaltering the jobs. Set each job's `Resource_List.preempt_targets=queue=<name of anti-express queue>`.

4.9.1.3 Anti-express Queue Caveats

If you use soft limits on the number of jobs that users can run at other queues, jobs that are over their soft limits at other queues will also have the lowest preemption priority.

4.9.2 Associating Vnodes with Queues

You can associate each vnode with one or more queues. When a vnode is associated with a queue, that means it accepts jobs from that queue only. You can associate one or more vnodes with multiple queues.

You do not need to associate vnodes with queues in order to have jobs run on the vnodes that have the right application, as long as the application is a resource that can be requested by jobs.

You can use custom host-level resources to associate one or more vnodes with more than one queue. A scheduler will use the resources for scheduling just as it does with any resource.

In order to map a vnode to more than one queue, you must define a new host-level string array custom resource. This string array holds a string that has the same value for the queue and vnode you wish to associate. The mechanism of association is that a job that lands in the queue inherits that value for the resource, and then the job can run only on vnodes having a matching value for the resource. You can associate more than one queue with a vnode by setting the resource to the same value at each queue.

In some cases, you can use the same resource to route jobs and to associate vnodes with queues. For the method described here, you use host-level resources to associate vnodes with queues. The rules for which resources can be used for routing are given in section 2.3.6.4.iii, “Resources Used for Routing and Admittance”, on page 27. How jobs inherit resources is described in section 5.9.4, “Allocating Default Resources to Jobs”, on page 247.

4.9.2.0.i Procedure to Associate Vnodes with Queues

To associate one or more vnodes with one or more queues, do the following:

1. Define the new host-level resource:

   ```
   qmgr -c 'create resource <new resource> type=string_array, flag=h'
   ```

2. Instruct the scheduler to honor the resource. Add the new resource to `$<sched_priv directory>/sched_config`:

   ```
   resources: "ncpus, mem, arch, host, vnode, <new resource>"
   ```

3. HUP the scheduler:

   ```
   kill -HUP <scheduler PID>
   ```

4. Set each queue's default_chunk for the new resource to the value you are using to associate it with vnodes:

   ```
   Qmgr: set queue <queue name> default_chunk.<new resource> = <value>
   ```

   For example, if one queue is "MathQ" and one queue is "SpareQ", and the new resource is "Qlist", and you want to associate a set of vnodes and queues based on ownership by the math department, you can make the queue resource value be "math":

   ```
   Qmgr: set queue MathQ default_chunk.Qlist = math
   Qmgr: set queue SpareQ default_chunk.Qlist = math
   ```

5. Set the value for the new resource at each vnode:

   ```
   Qmgr: set node <vnode name> resources_available.<new resource> = <associating value>
   ```

   For example, to have the vnode named "Vnode1" associated with the queues owned by the math department:

   ```
   Qmgr: set node Vnode1 resources_available.Qlist = math
   ```

4.9.2.0.ii Example of Associating Multiple Vnodes with Multiple Queues

Now, as an example, assume you have 2 queues: "PhysicsQ" and "ChemQ", and you have 3 vnodes: vn[1], vn[2], and vn[3]. You want Physics jobs to run on vn[1] and vn[2], and you want Chem jobs to run on vn[2] and vn[3]. Each department gets exclusive use of one vnode, but both must share a vnode.

To achieve the following mapping:

PhysicsQ -->vn[1], vn[2]

ChemQ --> vn[2], vn[3]

Which is the same as:

vn[1] <-- PhysicsQ

vn[2] <-- PhysicsQ, ChemQ

vn[3] <-- ChemQ

1. Define the new host-level resource:
   Qmgr: create resource Qlist type=string_array, flag=h
2. Instruct the scheduler to honor the resource. Add the new resource to `$<sched_priv directory>/sched_config`:

   `resources: "ncpus, mem, arch, host, vnode, Qlist"`
3. HUP the scheduler:

   kill -HUP <scheduler PID>
4. Add queue to vnode mappings:

   Qmgr: s n vn[1] resources_available.Qlist="PhysicsQ"
   Qmgr: s n vn[2] resources_available.Qlist= "PhysicsQ,ChemQ"
   Qmgr: s n vn[3] resources_available.Qlist="ChemQ"
5. Force jobs to request the correct Qlist values:

   Qmgr: s q PhysicsQ default_chunk.Qlist=PhysicsQ
   Qmgr: s q ChemQ default_chunk.Qlist=ChemQ

4.9.3 Using Backfilling

Backfilling means fitting smaller jobs around the higher-priority jobs that a scheduler is going to run next, in such a way that the higher-priority jobs are not delayed. When a scheduler is using backfilling, the scheduler considers highest-priority jobs *top jobs*. Backfilling changes the algorithm that a scheduler uses to run jobs:

- When backfilling is not being used, a scheduler looks at each job in priority order, tries to run the job now, and if it cannot, it moves on to the next-highest-priority job.
- When backfilling is being used, a scheduler tries to run the top job now, and if it cannot, it makes sure that no other job that it runs in this cycle will delay the top job. It also fits smaller jobs in around the top job.

Backfilling allows you to keep resources from becoming idle when the top job cannot run.

Backfilling applies all of the time; it is not a prime option.

4.9.3.1 Glossary

Top job

A top job has the highest execution priority according to scheduling policy, and a scheduler plans resources and start time for this job first. Top jobs exist only when a scheduler is using backfilling.

Filler job

Smaller job that fits around top jobs. Running a filler job does not change the start time or resources for a top job. This job runs next only when backfilling is being used (meaning that a top job cannot start next because insufficient resources are available for the top job, but whatever is available is enough for the filler job).

4.9.3.2 Backfilling Separately at the Server and Queues

You can configure the number of top jobs that PBS backfills around by setting the value of the backfill_depth server and queue attributes. For example, if you set backfill_depth to *3*, PBS backfills around the top 3 jobs. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

You can specify a different number of top jobs for each queue. You can also specify the number of top jobs for the server. Any queues that do not have their own backfill depth share in the server's backfill depth count. For example, you have three queues Q1, Q2, and Q3, and you set the backfill depth at Q1 to be 5 and the backfill depth at the server to be 3. In this example, the top 5 jobs in Q1 will run as soon as possible, and be backfilled around, but there are only 3 top job slots allocated to the jobs in Q2 and Q3.

If you do not set a value for the backfill depth at the server, it defaults to *1*.

4.9.3.3 How Backfilling Works

A scheduler makes a list of jobs to run in order of priority, for any queue that has an individual backfill depth, for the server if there are queues without a backfill depth set. These lists are composed according to execution priority described in section 4.9.16, “Calculating Job Execution Priority”, on page 134. These are top jobs.

If you use backfilling, a scheduler looks for smaller jobs that can fit into the usage gaps around the highest-priority jobs in each list. A scheduler looks in each prioritized list of jobs and chooses the highest-priority smaller jobs that fit. Filler jobs are run only if they will not delay the start time of top jobs.

A scheduler creates a fresh list of top jobs at every scheduling cycle, so if a new higher-priority job has been submitted, it will be considered.

You can use shrink-to-fit jobs to backfill into otherwise unusable time slots. PBS checks whether a shrink-to-fit job could shrink into the available slot, and if it can, runs it. See section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211.

Backfilling is useful in the following circumstances:

- When the strict_ordering scheduler parameter is turned on, filler jobs are fitted around higher-priority jobs. Without backfilling, no job runs if the top job cannot run. See section 4.9.49, “Using Strict Ordering”, on page 225
- When the help_starving_jobs scheduler parameter is turned on, filler jobs are fitted around starving jobs. See section 4.9.48, “Starving Jobs”, on page 223

4.9.3.4 Backfilling Around *N* Jobs

You can configure the number of top jobs that PBS backfills around by setting the value of the backfill_depth server attribute. For example, if you set backfill_depth to *3*, PBS backfills around the top 3 jobs. See “Server Attributes” on page 281 of the PBS Professional Reference Guide.

4.9.3.5 Backfilling Around Preempted Jobs

When you set the sched_preempt_enforce_resumption scheduler attribute to *True*, a scheduler adds preempted jobs to the set of jobs around which it backfills. A scheduler ignores backfill_depth when backfilling around jobs in the *Preempted* execution class. By default the sched_preempt_enforce_resumption scheduler attribute is *False*.

4.9.3.6 Backfilling Around Starving Jobs

When you take starving jobs into consideration, by setting the help_starving_jobs scheduler parameter to *True*, starving jobs can be added to the top jobs. They can continue to wait for resources once they are the top job, blocking other jobs from running. See section 4.9.48, “Starving Jobs”, on page 223.

4.9.3.7 Configuring Backfilling

To configure backfilling, do the following:

1. Choose how many jobs to backfill around. If you want to backfill around more than 1 job, set the backfill_depth server attribute to the desired number. The default is *1*. Set this parameter to less than *100*.
2. Choose whether you want any queues to share the list of top jobs at the server. Do not set backfill_depth at those queues. If you want any queues to share this list, set the server's backfill_depth attribute to the desired value. The default is *1*. Set this parameter to less than *100*.
3. For the queues where you want a separate backfill depth, choose how many jobs to backfill around at each queue. Set the backfill_depth queue attribute to the desired number.
4. Make sure that jobs request walltime by making them inherit a walltime resource if they don't explicitly request it. For options, see section 4.9.3.11.i, “Ensure Jobs Are Eligible for Backfilling”, on page 109.
5. Choose whether you want to backfill around preempted jobs. To do this, set the sched_preempt_enforce_resumption scheduler attribute to *True*.
6. Make sure that the strict_ordering scheduler parameter is set to *True* for all time if you use backfilling.
7. Choose whether you want to backfill around starving jobs. If you do, make sure that the help_starving_jobs scheduler parameter is set to *True*.

When most jobs become top jobs, they are counted toward the limit set in backfill_depth. Some top jobs are not counted toward backfill_depth. The following table shows how backfilling can be configured and which top jobs affect backfill_depth. Unless explicitly stated, top jobs are counted towards backfill_depth. A scheduler stops considering jobs as top jobs when it has reached backfill_depth, except for preempted jobs, which do not count toward that limit. When backfill is off, a scheduler does not have a notion of "top jobs". When help_starving_jobs is off, a scheduler has no notion of starving jobs.

Table 4-5: Configuring Backfilling

Parameter and Attribute Settings				When Classes Are Top Jobs			
backfill_depth	strict_ordering	sched_preempt_enforce_resumption	help_starving_jobs	Express	Preempted	Starving	Normal
>0	T	T	T	Top jobs	Top jobs, not counted in backfill_depth	Top jobs	Top jobs
>0	T	T	F	Top jobs	Top jobs, not counted in backfill_depth	Starving class does not exist	Top jobs
>0	T	F	T	Top jobs	Top jobs	Top jobs	Top jobs
>0	T	F	F	Top jobs	Top jobs	Starving class does not exist	Top jobs
>0	F	T	T	No	Top jobs, not counted in backfill_depth	Top jobs	No
>0	F	T	F	No	Top jobs, not counted in backfill_depth	Starving class does not exist	No
>0	F	F	T	No	No	Top jobs	No
>0	F	F	F	No	No	Starving class does not exist	No

4.9.3.8 Backfilling and Strict Ordering

When you use strict ordering, a scheduler runs jobs in exactly the order of their priority. If backfill_depth is set to zero and the top job cannot run, no job is able to run. Backfilling can prevent resources from standing idle while the top job waits for its resources to become available. See section 4.9.49, “Using Strict Ordering”, on page 225.

4.9.3.9 Backfilling and Scheduler Cycle Speed

You can choose a trade-off between scheduling cycle speed and the fineness of the granularity with which estimated start times are calculated. You do this by setting the opt_backfill_fuzzy scheduler attribute via `qmgr`. You can choose *off*, *low*, *medium*, or *high*. For no speedup, choose *off*. For maximum speedup, choose *high*.

```
Qmgr: set sched opt_backfill_fuzzy [off | low | medium | high]
```

See section 4.9.40, "Scheduler Cycle Speedup", on page 209.

4.9.3.10 Attributes and Parameters Affecting Backfilling

backfill_depth

Server and queue attribute. Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around. See "Server Attributes" on page 281 of the PBS Professional Reference Guide and "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

opt_backfill_fuzzy

Scheduler attribute. You can use this setting to trade between scheduling cycle speed and estimated start time granularity. See "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

sched_preempt_enforce_resumption

Scheduler attribute. When this attribute is *True* and backfill_depth is greater than zero, a scheduler treats preempted jobs like top jobs and backfills around them. This effectively increases the value of backfill_depth by the number of preempted jobs.

The configuration parameters backfill_prime and prime_exempt_anytime_queues do not relate to backfilling. They control the time boundaries of regular jobs with respect to primetime and non-primetime. See section 4.9.34, "Using Primetime and Holidays", on page 191.

4.9.3.11 Backfilling Recommendations and Caveats

4.9.3.11.i Ensure Jobs Are Eligible for Backfilling

When calculating backfilling, PBS treats a job that has no walltime specified as if its walltime is eternity. A scheduler will never use one of these jobs as a filler job. You can avoid this by ensuring that each job has a realistic walltime, by using the following methods:

- At `qsub` time via a hook
- By setting the queue's resources_default.walltime attribute
- By setting the server's resources_default.walltime attribute
- At `qsub` time via the server's default_qsub_arguments

4.9.3.11.ii Number of Jobs to Backfill Around

The more jobs being backfilled around, the longer the scheduling cycle takes.

4.9.3.11.iii Dynamic Resources and Backfilling

Using dynamic resources and backfilling may result in some jobs not being run because a dynamic resource is temporarily unavailable. This may happen when a job requesting a dynamic resource is selected as the top job. A scheduler must estimate when resources will become available, but it can only query for resources available at the time of the query, not resources already in use, so it will not be able to predict when resources in use become available. Therefore the scheduler won't be able to schedule the job. In addition, since dynamic resources are outside of the control of PBS, they may be consumed between the time a scheduler queries for the resource and the time it starts a job.

4.9.3.11.iv Avoid Using Strict Ordering, Backfilling, and Fairshare

It is inadvisable to use strict ordering and backfilling with fairshare.

The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that while a large job waits for a slot, jobs from the same entity or group will be chosen as the filler jobs, and the usage from these small jobs will lower the priority of the large job.

For example, if a user has a large job that is the most deserving but cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from ever being likely to run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.

4.9.3.11.v Using Preemption, Strict Ordering, and Backfilling

Using preemption with strict ordering and backfilling may reshuffle the top job(s) if high-priority jobs are preempted.

4.9.3.11.vi Warning About Backfilling and Provisioning

A scheduler will not run a job requesting an AOE on a vnode that has a top job scheduled on it in the future.

A scheduler will not use a job requesting an AOE as a top job.

4.9.3.11.vii Backfilling and Estimating Job Start Time

When a scheduler is backfilling around jobs, it estimates the start times and execution vnodes for the top jobs being backfilled around. See section 4.9.15, “Estimating Job Start Time”, on page 131.

4.9.3.11.viii Using Strict Ordering and Backfilling with Only One of Primetime or Non-primetime

If you use backfilling, it is used all of the time. However, you can use strict ordering during primetime, non-primetime, or all the time. When PBS is using strict ordering and backfilling, a scheduler saves a spot for each high-priority job around which it is backfilling. If you configure PBS to use strict ordering and backfilling for only one of primetime or non-primetime, and you have large jobs that must wait a long time before enough resources are available, the saved spots can be lost in the transition.

4.9.4 Examining Jobs Queue by Queue

When a scheduler examines waiting jobs, it can either consider all of the jobs in its partition as a whole, or it can consider jobs queue by queue. When considering jobs queue by queue, a scheduler runs all the jobs it can from the first queue before examining the jobs in the next queue, and so on. This behavior is controlled by the by_queue scheduler parameter.

When the by_queue scheduler parameter is set to *True*, jobs in the highest-priority queue are evaluated as a group, then jobs in the next-highest priority queue are evaluated. In this case, PBS runs all the jobs it can from each queue before moving to the next queue, with the following exception: if there are jobs in the *Reservation, Express*, *Preempted*, or *Starving* job execution classes, those are considered before any queue. These classes are described in section 4.9.16, “Calculating Job Execution Priority”, on page 134.

The by_queue parameter applies to all of the queues in a scheduler's partition. This means that either all jobs are scheduled as if they are in one large queue, or jobs are scheduled queue by queue.

All queues are always sorted by queue priority. To set queue priority, set each queue's priority attribute to the desired value. A queue with a higher value is examined before a queue with a lower value. If you do not assign priorities to queues, their ordering is undefined. See section 4.9.36, “Queue Priority”, on page 196.

The by_queue parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

See “by_queue” on page 250 of the PBS Professional Reference Guide.

4.9.4.1 Configuring PBS to Consider Jobs Queue by Queue

- Set the by_queue scheduler parameter to *True*
- Assign a priority to each queue
- Choose whether you want queue by queue during primetime, non-primetime, or both. If you want separate behavior for primetime and non-primetime, list by_queue twice. For example:

```
by_queue True prime
by_queue False non_prime
```

4.9.4.2 Parameters and Attributes Affecting Queue by Queue

- The by_queue scheduler parameter; see "by_queue" on page 250 of the PBS Professional Reference Guide.
- The priority queue attribute; see "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

4.9.4.3 Caveats and Advice for Queue by Queue

- The by_queue scheduler parameter is overridden by the round_robin scheduler parameter when round_robin is set to *True*.
- When by_queue is *True*, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.
- When by_queue is *True*, job execution priority may be affected. See section 4.9.16, "Calculating Job Execution Priority", on page 134.
- The by_queue parameter is not required when using express queues.
- You can have FIFO scheduling for all your jobs across a given scheduler's partition, if you are using a single execution queue or have by_queue set to *False*. However, you can have FIFO scheduling for the jobs within each queue if you set by_queue to *True* and specify a different priority for each queue. See section 4.9.20, "FIFO Scheduling", on page 148.

4.9.5 Checkpointing

You can use checkpointing as a scheduling tool, by including it as a preemption method, an aid in recovery, a way to capture progress from a shrink-to-fit job, and when using the `qhold` command.

For a complete description of how to use and configure checkpointing, see section 9.3, "Checkpoint and Restart", on page 418.

4.9.5.1 Checkpointing as a Preemption Method

When a job is preempted via checkpointing, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. When a scheduler elects to run the job again, the MoM runs the restart script to restart the job from where it was checkpointed. See section 4.9.33, "Using Preemption", on page 180.

4.9.5.2 Checkpointing as a Way to Capture Progress and Help Recover Work

When you use checkpointing to capture a job's progress before the job is terminated, for example when a shrink-to-fit job's wall time is exceeded, MoM runs the snapshot checkpoint script, and the job continues to run. See section 9.3, "Checkpoint and Restart", on page 418.

4.9.5.3 Checkpointing When Using the `qhold` Command

When the qhold command is used to hold a checkpointable job, MoM runs the checkpoint_abort script, and PBS kills, requeues, and holds the job. A job with a hold on it must have the hold released via the qrls command in order to be eligible to run. For a discussion of the use of checkpointing for the `qhold` command, see section 9.3.7.6, "Holding a Job", on page 430. See "qhold" on page 147 of the PBS Professional Reference Guide and "qrls" on page 180 in the PBS Professional Installation & Upgrade Guide.

4.9.6 Organizing Job Chunks

You can specify how job chunks should be organized onto hosts or vnodes. Jobs can request their placement arrangement, and you can set defaults at queues and at the server to be inherited by jobs that do not request a placement. You can tell PBS to do the following:

- Put all chunks from a job onto a single host using the `place=pack` statement.
- Put each chunk on a separate host using the `place=scatter` statement. The number of chunks must be fewer than or equal to the number of hosts.
- Put each chunk on a separate vnode using the `place=vscatter` statement. The number of chunks must be fewer than or equal to the number of vnodes.
- Put each chunk anywhere using the `place=free` statement.

To specify a placement default, set resources_default.place=<arrangement>, where arrangement is *pack*, *scatter*, *vscatter*, or *free*. For example, to have the default at QueueA be *pack*:

```
Qmgr: set queue QueueA resources_default.place=pack
```

You can specify that job chunks must be grouped in a certain way. For example, to require that chunks all end up on a shared router, use this:

```
place=group=router
```

For more about jobs requesting placement, see "Requesting Resources and Placing Jobs" on page 216 of the PBS Professional Reference Guide.

4.9.6.1 Caveats for Organizing Job Chunks

A placement specification for arrangement, sharing, and grouping is treated as one package by PBS. This means that if a job requests only one, any defaults set for the others are not inherited. For example, if you set a default of `place=pack:excl:group=router`, and a job requests only `place=pack`, the job does not inherit `excl` or `group=router`. See "Requesting Resources and Placing Jobs" on page 216 of the PBS Professional Reference Guide.

4.9.7 `cron` Jobs

You can use `cron` jobs to make time-dependent modifications to settings, where you are scheduling according to time slots. For example, you can change settings for primetime and non-primetime configurations, making the following changes:

- Set nodes *offline* or not *offline*
- Change the number of ncpus on workstations
- Change the priority of queues, for example to change preemption behavior
- Start or stop queues
- Set primetime & non-primetime options

4.9.7.1 Caveats for `cron` Jobs

- Make sure that your `cron` jobs behave correctly when PBS is not running.
- Be careful when changing available resources, such as when offlining vnodes. You might prevent jobs from running that would otherwise run. For details, see section 4.7.2, "Jobs that Cannot Run on Current Resources", on page 99.

  If PBS is down when your `cron` job runs, the change specified in the `cron` job won't happen. For example, if you use `cron` to offline a vnode and then bring it online later, it won't come online if PBS is down during the second operation.

4.9.8 Using Custom and Default Resources

The information in this section relies on understanding how jobs are allocated resources via inheriting defaults or via hooks. Before reading this section, please read section 14.3, "Allocating Resources to Jobs", on page 505.

For complete details of how to configure and use custom resources, please see section 5.14, "Custom Resources", on page 255.

You can use custom and default resources for several purposes:

- Routing jobs to the desired vnodes; see section 4.9.8.2, "Using Custom Resources to Route Jobs", on page 113
- Assigning execution priority to jobs; see section 4.9.8.3, "Using Custom Resources to Assign Job Execution Priority", on page 114
- Tracking and controlling the allocation of resources; see section 4.9.8.4, "Using Custom Resources to Track and Control Resource Allocation", on page 114
- Representing elements such as GPUs, FPGAs, and switches; see section 4.9.8.5, "Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.", on page 114
- Allowing users to request platform-specific resources, for example Cray-specific resources; see section 4.9.8.6, "Using Custom Resources to Allow Platform-specific Resource Requests", on page 114
- Allowing users to submit jobs that run on a Cray as they would if using the `aprun` command; see section 4.9.8.7, "Using Custom Resources to Allow Platform-specific Behavior", on page 115
- Shrinking job walltimes so that they can run in time slots that are less than the expected maximum. See section 4.9.42, "Using Shrink-to-fit Jobs", on page 211.

4.9.8.1 Techniques for Allocating Custom Resources to Jobs

In addition to using custom resources to represent physical elements such as GPUs, you can use custom resources as tags that you attach to jobs in order to help schedule the jobs. You can make these custom resources into tools that can be used only for managing jobs, by making them unalterable and unrequestable, and if desired, invisible to users.

For how to assign custom and default resources to jobs, see section 14.3, "Allocating Resources to Jobs", on page 505.

4.9.8.2 Using Custom Resources to Route Jobs

You can use several techniques to route jobs to the desired queues and/or vnodes. Depending on your partition's or site's configuration, you may find it helpful to use custom resources with one or more of these techniques.

- You can force users to submit jobs to the desired queues by setting resource limits at queues. You can use custom resources to represent arbitrary elements, for example, department. In this case you could limit which department uses each queue. You can set a default value for the department at the server, or create a hook that assigns a value for the department.

For how queue resource limits are applied to jobs, see section 2.3.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 26.

- Use default resources or a hook to assign custom resources to jobs when the jobs are submitted. Send the jobs to routing queues, then route them, using the resources, to other queues inside or outside the PBS partition or complex. Again, custom resources can represent arbitrary elements.

  For how routing queues work, see section 2.3.6, “Routing Queues”, on page 25

- Use peer scheduling to send jobs between PBS partitions or complexes. You can set resource limits on the furnishing queue in order to limit the kinds of jobs that are peer scheduled. You can assign custom resources to jobs to represent arbitrary elements, for example peer queueing only those jobs from a specific project. You can assign the custom resource by having the job inherit it or via a hook.

  For how to set up peer scheduling, see section 4.9.31, “Peer Scheduling”, on page 165

- You can route jobs from specific execution queues to the desired vnodes, by associating the vnodes with the queues. See section 4.9.2, “Associating Vnodes with Queues”, on page 103.

- You can create placement sets so that jobs are placed according to resource values. Placement sets are created where vnodes share a value for a resource; you can use custom resources to create the placement sets you want. See section 4.9.32, “Placement Sets”, on page 168.

4.9.8.3 Using Custom Resources to Assign Job Execution Priority

You can use custom resources as coefficients in the job sorting formula. You can assign custom resources to jobs using the techniques listed in section 14.3, “Allocating Resources to Jobs”, on page 505. The value of each custom resource can be based on a project, an application, etc.

For example, you can create a custom resource called "ProjPrio", and the jobs that request the "Bio" project can be given a value of 5 for ProjPrio, and the jobs that request the "Gravel" project can be given a value of 2 for ProjPrio. You can assign this value in a hook or by routing the jobs into special queues from which the jobs inherit the value for ProjPrio.

For information on using the job sorting formula, see section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

4.9.8.4 Using Custom Resources to Track and Control Resource Allocation

You can use resources to track and control usage of things like CPUs and memory. For example, you might want to limit the number of jobs using a particular vnode. See section 5.10, “Using Resources to Track and Control Allocation”, on page 252.

4.9.8.5 Using Custom Resources to Represent GPUs, FPGAs, Switches, Etc.

You can use custom resources to represent GPUs, FPGAs, high performance switches, etc. For examples, see section 5.14.7, “Using GPUs”, on page 283, and section 5.14.8, “Using FPGAs”, on page 286.

4.9.8.6 Using Custom Resources to Allow Platform-specific Resource Requests

PBS is integrated with Cray, and provides special custom resources to represent Cray resources. You can create other custom resources to represent other platform-specific elements. For an example, see section 11.9.6, “Allowing Users to Request Login Node Groups on Cray XC”, on page 480.

4.9.8.7 Using Custom Resources to Allow Platform-specific Behavior

You can create custom resources that allow Cray users to run jobs that behave the same way they would if the user had used the `aprun` command.

4.9.9 Using Idle Workstation Cycle Harvesting

You can configure workstations at your partition or site so that PBS can run jobs on them when their "owners" are away and they are idle. This is called *idle workstation cycle harvesting*. This can give your partition or site additional resources to run jobs during nights and weekends, or even during lunch.

You can configure PBS to use the following methods to decide when a workstation is not being used by its owner:

- Keyboard/mouse activity
- X-Window monitoring
- Load average (not recommended)

On some systems cycle harvesting is simple to implement, because the console, keyboard, and mouse device access times are periodically updated by the operating system. The PBS MoM process can track this information, and mark the vnode *busy* if any of the input devices is in use. On other systems, however, this data is not available: on some machines, PBS can monitor the X-Window system in order to obtain interactive idle time, and on others, PBS itself monitors keyboard and mouse activity.

Jobs on workstations that become *busy* are not migrated; they remain on the workstation until they complete execution, are rerun, or are deleted.

4.9.9.1 Platforms Supporting Cycle Harvesting

Due to different operating system support for tracking mouse and keyboard activity, the availability and method of support for cycle harvesting varies based on the computer platform in question. The following table lists the method and support for each platform.

Table 4-6: Cycle Harvesting Support Methods

System	Status	Method	Reference
Linux	supported	keyboard/mouse	section 4.9.9.3, “Cycle Harvesting Based on Keyboard/Mouse Activity”, on page 116
Windows	supported	keyboard/mouse	section 4.9.9.4, “Cycle Harvesting on Windows”, on page 117

4.9.9.2 The $kbd_idle MoM Configuration Parameter

Cycle harvesting based on keyboard/mouse activity and X-Windows monitoring is controlled by the $kbd_idle MoM configuration parameter in `PBS_HOME/mom_priv/config` on the workstation in question. This parameter has the following format:

$kbd_idle <idle_wait> <min_use> <poll_interval>

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use.

idle_wait

Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.

Must be set to value greater than 0 for cycle harvesting to be enabled.

Format: Integer

No default

min_use

Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.

Format: Integer

Default: *10*

poll_interval

Interval, in seconds, at which MoM checks for keyboard and mouse activity.

Format: Integer

Default: *1*

4.9.9.3 Cycle Harvesting Based on Keyboard/Mouse Activity

PBS can monitor a workstation for keyboard and mouse activity, and run batch jobs on the workstation when the keyboard and mouse are not being used. PBS sets the state of the vnode to either *free* or *busy*, depending on whether or not there is keyboard or mouse activity, and runs jobs only when the state of the vnode is *free*. PBS sets the state of the vnode to *free* when the vnode's mouse and keyboard have shown no activity for the specified amount of time. If PBS determines that the vnode is being used, it sets the state of the vnode to *busy* and suspends any running jobs, setting their state to *U (user busy)*.

This method is used for Linux operating systems.

4.9.9.3.i Configuring Cycle Harvesting Using Keyboard/Mouse Activity

To configure cycle harvesting using keyboard and mouse activity, do the following:

1. Set the $kbd_idle MoM configuration parameter by editing the $kbd_idle parameter in `PBS_HOME/mom_priv/config` on the workstation.
2. HUP the MoM on the workstation:

```
kill -HUP <pbs_mom PID>
```

4.9.9.3.ii Example of Cycle Harvesting Using Keyboard/Mouse Activity

The following is an example setting for the parameter:

```
$kbd_idle 1800 10 5
```

This setting for the parameter in MoM's `config` file specifies the following:

- PBS marks the workstation as *free* if the keyboard and mouse are idle for 30 minutes (1800 seconds)
- PBS marks the workstation as *busy* if the keyboard or mouse is used for 10 consecutive seconds
- The states of the keyboard and mouse are to be checked for activity every 5 seconds

Here, we walk through how this example would play out, to show the roles of the arguments to the $kbd_idle parameter:

Let's start with a workstation that has been in use for some time by its owner. The workstation is in state *busy*.

Now the owner goes to lunch. After 1800 seconds (30 minutes), PBS changes the workstation's state to *free* and starts a job on the workstation.

Some time later, someone walks by and moves the mouse or enters a command. Within the next 5 seconds (idle poll period), `pbs_mom` notes the activity. The job is suspended and placed in state *U,* and the workstation is marked *busy*.

If 10 seconds pass and there is no additional keyboard/mouse activity, the job is resumed and the workstation again is either *free* (if any CPUs are available) or *job-busy* (if all CPUs are in use.)

However, if keyboard/mouse activity continues during that 10 seconds, the workstation remains *busy* and the job remains suspended for at least the next 1800 seconds.

4.9.9.3.iii Caveats for Cycle Harvesting Using Keyboard/Mouse Activity

- There is no default for idle_wait; you must set it to a value greater than 0 in order to enable cycle harvesting using keyboard/mouse activity.

4.9.9.4 Cycle Harvesting on Windows

A process called `pbs_idled` monitors keyboard and mouse activity and keeps MoM informed of user activity. The user being monitored can be sitting at the machine, or using a remote desktop.

The `pbs_idled` process is managed in one of two ways. PBS can use a service called *PBS_INTERACTIVE* to monitor the user's session. If the PBS_INTERACTIVE service is registered, MoM starts the service, and the service starts and stops `pbs_idled`. The PBS_INTERACTIVE service runs under a local system account. PBS uses the PBS_INTERACTIVE service only where partition or site policy allows a local system account to be a service account. If this is not allowed (so the service is not registered), `pbs_idled` is started and stopped using the log on/log off script. Do not use both the PBS_INTERACTIVE service and a log on/log off script.

A `pbs_idled` process monitors the keyboard and mouse activity while a user is logged in. This process starts when the user logs on, and stops when the user logs off. Only a user with administrator privileges, or the user being monitored, can stop `pbs_idled.`

MoM uses two files to communicate with `pbs_idled`:

- MoM creates `PBS_HOME/spool/idle_poll_time` and writes the value of her $kbd_idle polling interval parameter to it. The `pbs_idled` process reads the value of the polling interval from `idle_poll_time`.
- MoM creates `PBS_HOME/spool/idle_touch`. The `pbs_idled` process updates the time stamp of the `idle_touch` file when a user is active, and MoM reads the time stamp.

4.9.9.4.i Configuring Cycle Harvesting on Windows

To configure cycle harvesting, do the following:

1. Make sure that you are a user with administrator privileges.
2. Set the $kbd_idle MoM configuration parameter by editing the $kbd_idle parameter in `PBS_HOME/mom_priv/config` on the workstation.
3. Configure how `pbs_idled` starts:
   a. If your policy allows a local system account to be a service account, register the PBS_INTERACTIVE service:

      ```
      pbs_interactive -R
      ```

   b. If your policy does not allow a local system account to be a service account:
      1. Configure the log on script as described in section 4.9.9.4.ii, “Configuring pbs_idled in Log On Script in Domain Environment”, on page 118.
      2. Configure the log off script as described in section 4.9.9.4.iii, “Configuring pbs_idled in Log Off Script in Domain Environment”, on page 119.
4. Restart the MoM.

4.9.9.4.ii Configuring `pbs_idled` in Log On Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open *Administrator Tools*.
3. In *Administrator Tools*, open *Active Directory Users and Computers*.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on *Properties*.
6. Go to the *Group Policy* tab under the *Properties* window.
7. Click on *New*.
8. Type "LOG-IN-OUT-SCRIPT" as the name of the policy.
9. Select the Group Policy Object you have just created; click *Edit*. The Group Policy Object editing window will open.
10. Open *Window Settings* in *User Configuration*.
11. Open *Scripts (Logon/Logoff)*.
12. Open *Logon*. A *Logon Properties* window will open.
13. Open Notepad in another window. In Notepad, you create the command that starts the `pbs_idled` process:

    ```
    pbs_idled start
    ```

14. Save that document as "*pbs_idled_logon.bat*".
15. In the *Logon Properties* window, click on *Show Files*. A logon script folder will open in a new window.
16. Copy `pbs_idled_logon.bat` into the logon script folder and close the logon script folder window.
17. In the *Logon Properties* window, click on *Add*, and then click on *Browse*. Select *pbs_idled_logon.bat* and then click on *Open*.
18. Click on *OK*, then *Apply*, then again *OK*.
19. Close the Group Policy Object editor and the *Properties* window.
20. Close the *Active Directory Users and Computers* window.
21. Close the *Administrator Tools* window.

4.9.9.4.iii Configuring pbs_idled in Log Off Script in Domain Environment

1. You must be a user with administrator privileges.
2. On the domain controller host, open *Administrator Tools*.
3. In *Administrator Tools*, open *Active Directory Users and Computers*.
4. Right-click on the Organizational Unit where you want to apply the group policy for logging on and logging off.
5. Click on *Properties*.
6. Go to the *Group Policy* tab under the *Properties* window.
7. Click on *New*.
8. Type "LOG-IN-OUT-SCRIPT" as the name of the policy.
9. Select the Group Policy Object you have just created; click *Edit*. The Group Policy Object editing window will open.
10. Open *Window Settings* in *User Configuration*.
11. Open *Scripts (Logon/Logoff)*.
12. Open *Logoff*. A *Logoff Properties* window will open.
13. Open Notepad in another window. In Notepad, you create the command that stops the pbs_idled process:

    ```
    pbs_idled stop
    ```

14. Save that document as "*pbs_idled_logoff.bat*".
15. In the *Logoff Properties* window, click on *Show Files*. A logoff script folder will open in a new window.
16. Copy pbs_idled_logoff.bat into the logoff script folder and close the logoff script folder window.
17. In the *Logoff Properties* window, click on *Add*, and then click on *Browse*. Select *pbs_idled_logoff.bat* and then click on *Open*.
18. Click on *OK*, then *Apply*, then again *OK*.
19. Close the Group Policy Object editor and the *Properties* window.
20. Close the *Active Directory Users and Computers* window.
21. Close the *Administrator Tools* window.

4.9.9.4.iv The PBS_INTERACTIVE Service

The PBS_INTERACTIVE service starts the pbs_idled process, as the current user, in the current active user's session. Each time a user logs on, the service starts a pbs_idled for that user, and when that user logs off, the service stops that user's pbs_idled process.

The service runs under a local system account. If your policy allows a local system account to be a service account, you can use PBS_INTERACTIVE. Otherwise you must configure pbs_idled in log on/log off scripts.

If you have configured the $kbd_idle MoM parameter, and you have registered the service, MoM starts the service. The service cannot be started manually.

If you will use PBS_INTERACTIVE, you must register the service. The installer cannot register the service.

- To register the PBS_INTERACTIVE service:

  ```
  pbs_interactive -R
  ```

Upon successful execution of this command, the following message is displayed:

```
"Service PBS_INTERACTIVE installed successfully"
```

- To unregister the PBS_INTERACTIVE service:

  pbs_interactive -U

  Upon successful execution of this command, the following message is displayed:

  ```
  "Service PBS_INTERACTIVE uninstalled successfully"
  ```

- To see the version number for PBS_INTERACTIVE service:

  pbs_interactive --version

4.9.9.4.v Errors and Logging

If the $kbd_idle MoM parameter is configured, MoM attempts to use cycle harvesting. MoM looks for the PBS_INTERACTIVE service in the Service Control Manager. If she finds the service, she starts it.

1. If she cannot find the service, MoM logs the following message at event class 0x0002:

   ```
   "Can not find PBS_INTERACTIVE service, Continuing Cycle Harvesting with Logon/Logoff Script"
   ```

2. MoM looks for PBS_HOME/spool/idle_touch. If she finds it, she uses cycle harvesting.
3. If she cannot find the file, MoM disables cycle harvesting and logs the following message at event class 0x0002:

   ```
   "Cycle Harvesting Failed, Please contact Admin"
   ```

MoM logs the following messages at event class 0x0001.

- If MoM fails to open the Service Control Manager:

  ```
  "OpenSCManager failed for PBS_INTERACTIVE"
  ```

- If MoM fails to open the PBS_INTERACTIVE service:

  ```
  "OpenService failed for PBS_INTERACTIVE"
  ```

- If MoM fails to start the PBS_INTERACTIVE service:

  ```
  "Could not start PBS_INTERACTIVE service"
  ```

- If MoM fails to get status information about the PBS_INTERACTIVE service:

  ```
  "Can not get information about PBS_INTERACTIVE service"
  ```

- If MoM fails to send a stop control message to the PBS_INTERACTIVE service:

  ```
  "Could not stop PBS_INTERACTIVE service"
  ```

- If the PBS_INTERACTIVE service does not respond in a timely fashion:

  ```
  "PBS_INTERACTIVE service did not respond in timely fashion"
  ```

- If MoM fails to create idle_touch and idle_poll_time in PBS_HOME/spool directory:

  ```
  "Can not create file < full path of idle file >"
  ```

- If MoM fails to write the idle polling interval into PBS_HOME/spool/idle_poll_time:

  ```
  "Can not write idle_poll time into < full path of idle_poll_time file > file"
  ```

4.9.9.4.vi Caveats for Cycle Harvesting on Windows

- Under Windows, if the pbs_idled process is killed, cycle harvesting will not work.
- Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in, and users are not employing Switch User.
- Do not use both the PBS_INTERACTIVE service and a log on/log off script.

4.9.9.5 Cycle Harvesting by Monitoring X-Windows

On Linux machines where the OS does not periodically update console, keyboard, and mouse device access times, PBS can monitor X-Window activity instead. PBS uses an X-Window monitoring process called `pbs_idled`. This process runs in the background and monitors X and reports to the `pbs_mom` whether or not the vnode is idle. `pbs_idled` is located in `$PBS_EXEC/sbin`.

To configure PBS for cycle harvesting by monitoring X-Windows, perform the following steps:

1. Create a directory for `pbs_idled`. This directory must have the same permissions as `/tmp` (i.e. mode *1777*). This will allow the `pbs_idled` program to create and update files as the user, which is necessary because the program runs as the user. For example:

   ```
   mkdir PBS_HOME/spool/idledir
   chmod 1777 PBS_HOME/spool/idledir
   ```

2. Turn on keyboard idle detection in the MoM `config` file:

   ```
   $kbd_idle <idle wait value>
   ```

3. Include `pbs_idled` as part of the X-Windows startup sequence.

   The best and most secure method of starting `pbs_idled` is via the system-wide `Xsession` file. This is the script which is run by `xdm` (the X login program) and sets up each user's X-Windows environment.

   You **must** place the startup line for `pbs_idled` before that of the window manager.

   You **must** make sure that `pbs_idled` runs in the background.

   On systems that use `Xsession` to start desktop sessions, insert a line invoking `pbs_idled` near the top of the file.

   For example, insert the following line in a Linux `Xsession` file:

   ```
   /usr/pbs/sbin/pbs_idled &
   ```

   If access to the system-wide `Xsession` file is not available, you can add `pbs_idled` to every user's personal `.xsession` or `.xinitrc` file, depending on the local OS requirements for starting X-windows programs upon login.

4.9.9.6 Cycle Harvesting Based on Load Average

As of version 2020.1, the load_balancing scheduler parameter is deprecated.

Cycle harvesting based on load average means that PBS monitors each workstation's load average, runs jobs where workstations have loads below a specified level, and suspends any batch jobs on workstations whose load has risen above the limit you set. When a workstation's owner uses the machine, the workstation's load rises.

When you configure cycle harvesting based on load average, you are performing the same configuration as for load balancing using load average. For a complete description of load balancing, see section 4.9.27, “Using Load Balancing”, on page 156.

4.9.9.6.i Attributes and Parameters Affecting Cycle Harvesting Based on Load Average

load_balancing

Deprecated (2020.1).

Scheduler parameter. When set to *True*, this scheduler places jobs only where the load average is below the specified limit.

Format: *Boolean*

Default: *False all*

$ideal_load <load>

MoM parameter. Defines the load below which the vnode is not considered to be *busy*. Used with the $max_load directive.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

$max_load <load> [suspend]

MoM parameter. Defines the load above which the vnode is considered to be *busy*. Used with the $ideal_load directive. No new jobs are started on a *busy* vnode.

The optional suspend directive tells PBS to suspend jobs running on the node if the load average exceeds the $max_load number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

We recommend setting this to a slightly higher value than your target load (which is typically the number of CPUs), for example *.25 + ncpus*.

Example:

```
$max_load 3.25
```

Format: *Float*

Default: number of CPUs

resv_enable

Vnode attribute. Controls whether the vnode can be used for advance and standing reservations. When set to *True*, this vnode can be used for reservations.

Format: *Boolean*

Default: *True*

no_multinode_jobs

Vnode attribute. Controls whether jobs which request more than one chunk are allowed to execute on this vnode. When set to *True*, jobs requesting more than one chunk are not allowed to execute on this vnode.

Format: *Boolean*

Default: *False*

4.9.9.6.ii How Cycle Harvesting Based on Load Average Works

Cycle harvesting based on load average means that PBS monitors the load average on each machine. When the load on a workstation is below what is specified in the $ideal_load MoM parameter, PBS sets the state of the workstation to *free*. A scheduler will run jobs on vnodes whose state is *free*. When the load on a workstation exceeds the setting for $max_load, PBS sets the state of the workstation to *busy*, and suspends jobs running on the workstation. PBS does not start jobs on a vnode whose state is *busy*. When the load drops below the setting for $ideal_load, PBS sets the state to *free*, and resumes the jobs that were running on the workstation.

PBS thinks that a 1-CPU job raises a vnode's load by 1. On machines being used for cycle harvesting, you set the values for $max_load and $ideal_load to reasonable limits. On other machines, you set these to values that will never be exceeded, so that load is effectively ignored.

On machines where these parameters are unset, the vnode's state is not set according to its load, so jobs are not suspended because a vnode is busy. However, if $max_load and $ideal_load are unset, they are treated as if they have the same value as resources_available.ncpus, and because there is usually a small background load, PBS will lose the use of a CPU's worth of load.

The load_balancing scheduler parameter (**deprecated** as of version 2020.1) controls a behavior wherein a scheduler won't place a job where the anticipated load would exceed $max_load. For example if a machine has a load of 1.25, is running a 1-CPU job, and has 2 CPUs, PBS won't place another 1-CPU job there.

4.9.9.6.iii Configuring Cycle Harvesting Based on Load Average

To set up cycle harvesting for idle workstations based on load average, perform the following steps:

1. If PBS is not already installed on the target execution workstations, do so now, selecting the execution-only install option. See the PBS Professional Installation & Upgrade Guide.
2. Edit the PBS_HOME/mom_priv/config configuration file on each target execution workstation, adding the $max_load and $ideal_load configuration parameters. Make sure they have values that will not interfere with proper operation. See section 4.9.9.6.v, “Caveats for Cycle Harvesting Based on Load Average”, on page 124.

   ```
   $max_load <load limit that allows jobs to run>
   $ideal_load <load at which to start jobs>
   ```
3. Edit the PBS_HOME/mom_priv/config configuration file on each machine where you are not using cycle harvesting, adding the $max_load and $ideal_load configuration parameters. Make sure they have values that will never be exceeded.

   ```
   $max_load <load limit that will never be exceeded>
   $ideal_load <load limit that will never be exceeded>
   ```
4. HUP the MoM:

   ```
   kill -HUP <pbs_mom PID>
   ```
5. Edit the <sched_priv directory>/sched_config configuration file to direct the scheduler to perform scheduling based on load_balancing.

   ```
   load_balancing: True   ALL
   ```
6. If you wish to oversubscribe the vnode's CPU(s), set its resources_available.ncpus to a higher number. Do this only on single-vnode machines. You must be cautious about matching ncpus and $max_load. See "Caveats for Cycle Harvesting Based on Load Average" on page 124 in the PBS Professional Administrator’s Guide.
7. HUP the scheduler:

   ```
   kill -HUP <pbs_sched PID>
   ```
8. Set the vnode's resv_enable attribute to *False*, to prevent the workstation from being used for reservations.

   ```
   Qmgr: set node <vnode name> resv_enable = False
   ```
9. Set the vnode's no_multinode_jobs attribute to *True*, to prevent the workstation from stalling multichunk jobs.

   ```
   Qmgr: set node <vnode name> no_multinode_jobs = True
   ```

4.9.9.6.iv Viewing Load Average Information

You can see the state of a vnode using the pbsnodes -a command.

4.9.9.6.v Caveats for Cycle Harvesting Based on Load Average

- Be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working.
- For information about keeping your partition or site running smoothly using $max_load and $ideal_load, see section 9.6.5, “Managing Load Levels on Vnodes”, on page 445
- If you set ncpus higher than the number of actual CPUs, and set $max_load higher to match, keep in mind that the workstation user could end up with an annoyingly slow workstation. This can happen when PBS runs jobs on the machine, but the combined load from the jobs and the user is insufficient for MoM to report being busy.

4.9.9.7 Cycle Harvesting and File Transfers

The cycle harvesting feature interacts with file transfers in one of two different ways, depending on the method of file transfer:

- If the user's job includes file transfer commands (such as `rcp` or `scp`) within the job script, and such a command is running when PBS decides to suspend the job on the vnode, then the file transfer is suspended as well.
- If the job has PBS file staging parameters (i.e. *stagein=*, *stageout=file1...*), and the load goes above $max_load, the file transfer is not suspended. This is because the file staging is not part of the job script execution, and is not subject to suspension. See "Detailed Description of Job Lifecycle", on page 37 of the PBS Professional User’s Guide.

4.9.9.8 Parallel Jobs With Cycle Harvesting

Cycle harvesting is not recommended for hosts that will run multi-host jobs. However, you may find that your partition or site benefits from using cycle harvesting on these machines. We provide advice on how to prevent cycle harvesting on these machines, and advice on how to accomplish it.

4.9.9.8.i General Advice: Parallel Jobs Not Recommended

Cycle harvesting is somewhat incompatible with multi-host jobs. If one of the hosts being used for a parallel job running on several hosts is being used for cycle harvesting, and the user types at the keyboard, job execution will be delayed for the entire job because the tasks running on that host will be suspended.

To prevent a machine which is being used for cycle harvesting from being assigned a multi-host job, set the vnode's no_multinode_jobs attribute to *True*. This attribute prevents a host from being used by jobs that span multiple hosts.

4.9.9.8.ii How to Use Cycle Harvesting with Multi-host Jobs

When a single-host job is running on a workstation configured for cycle harvesting, and that host becomes *busy*, the job is suspended. However, suspending a multi-host parallel job may have undesirable side effects because of inter-process communications. For a job which uses multiple hosts when one or more of the hosts becomes *busy*, the default action is to leave the job running.

However, you can specify that the job should be requeued and subsequently re-scheduled to run elsewhere when any of the hosts on which the job is running becomes *busy*. To enable this action, add the following parameter to MoM's configuration file:

```
$action multinodebusy 0 requeue
```

where multinodebusy is the action to modify; "*0*" (zero) is the action timeout value (it is ignored for this action); and *requeue* is the new action to perform. The only action that can be performed is requeueing.

Multi-host jobs which are not rerunnable (i.e. those submitted with the `qsub -rn` option) will be killed if the requeue argument is configured for the multinodebusy action and a vnode becomes busy.

4.9.9.9 Cycle Harvesting Caveats and Restrictions

4.9.9.9.i Cycle Harvesting and Multi-host Jobs

Cycle harvesting is not recommended for hosts that will run multi-host jobs. See section 4.9.9.8.i, "General Advice: Parallel Jobs Not Recommended", on page 124.

4.9.9.9.ii Cycle Harvesting and Reservations

Cycle harvesting is incompatible with jobs in reservations. Reservations should not be made on a machine used for cycle harvesting, because the user may appear during the reservation period and use the machine's keyboard. This will suspend the jobs in the reservation, defeating the purpose of making a reservation.

To prevent a vnode which is being used for cycle harvesting from being used for reservations, set the vnode's resv_enable attribute to *False*. This attribute controls whether the vnode can be used for reservations.

4.9.9.9.iii File Transfers with Cycle Harvesting

File transfers behave differently depending on job details. See section 4.9.9.7, "Cycle Harvesting and File Transfers", on page 124.

4.9.9.9.iv Cycle Harvesting on Windows

- Under Windows, if the `pbs_idled` process is killed, cycle harvesting will not work.
- Under Windows, cycle harvesting may not work correctly on machines where more than one user is logged in.

4.9.10 Dedicated Time

PBS provides a feature called *dedicated time* which allows you to define times during which the only jobs that can run are the ones in dedicated queues. You can use dedicated time for things like upgrades.

You can define multiple dedicated times. Any job in a dedicated time queue must have a walltime in order to run. Jobs without walltimes will never run. PBS won't let a reservation conflict with dedicated time. Hooks should not access or modify the dedicated time file.

For information on configuring dedicated time queues, see section 2.3.5.2.i, "Dedicated Time Queues", on page 24.

4.9.10.1 Dedicated Time File

You define dedicated time by adding one or more time slots in the file `<sched_priv directory>/dedicated_time`. A time slot is a start date and start time and an end date and end time. Format:

<start date> <start time> <end date> <end time>

expressed as

MM/DD/YYYY HH:MM MM/DD/YYYY HH:MM

Any line whose first non-whitespace character is a pound sign ("#") is a comment.

Example:

```
#Dedicated time for maintenance
04/15/2007 12:00 04/15/2007 15:30
```

A sample dedicated time file (`PBS_EXEC/etc/pbs_dedicated`) is included in the installation.

The dedicated time file is read on startup and HUP.

4.9.10.2 Steps in Defining Dedicated Time

You define dedicated time by performing the following steps:

1. Edit the file `<sched_priv directory>/dedicated_time` and add one or more time slots.
2. HUP or restart the scheduler:

   Linux:

   ```
   kill -HUP <pbs_sched PID>
   ```

4.9.10.3 Recommendations for Dedicated Time

If you need to set up dedicated time for something like system maintenance, you may want to avoid having the machines become idle for a significant period before dedicated time starts. You can allow jobs to shrink their walltimes to fit into those shorter-than-normal slots before dedicated time. See section 4.9.42, “Using Shrink-to-fit Jobs”, on page 211.

4.9.11 Dependencies

PBS allows job submitters to specify dependencies between jobs, for example specifying that job J2 can only run if job J1 finishes successfully. In addition, you can add dependencies to existing jobs via a hook, default arguments to `qsub`, or via the `qalter` command.

For a description of how job dependencies work, see "Using Job Dependencies", on page 107 of the PBS Professional User’s Guide.

For how to use hooks, see the PBS Professional Hooks Guide.

For how to add default `qsub` arguments, see “Server Attributes” on page 281 of the PBS Professional Reference Guide.

For how to use the `qalter` command, see “qalter” on page 127 of the PBS Professional Reference Guide.

4.9.12 Dynamic Resources

You can use dynamic PBS resources to represent elements that are outside of the control of PBS, typically for application licenses and scratch space. You can represent elements that are available to the entire PBS partition or complex as server-level resources, or elements that are available at a specific host or hosts as host-level resources. For an example of configuring a server-level dynamic resource, see section 5.14.3.1.i, “Example of Configuring Dynamic Server-level Resource”, on page 267. For an example of configuring a dynamic host-level resource, see section 5.14.4.1.i, “Example of Configuring Dynamic Host-level Resource”, on page 269.

For a complete description of how to create and use dynamic resources, see section 5.14, “Custom Resources”, on page 255.

4.9.13 Eligible Wait Time for Jobs

PBS provides a method for tracking how long a job that is eligible to run has been waiting to run. By "eligible to run", we mean that the job could run if the required resources were available. The time that a job waits while it is not running can be classified as "eligible" or "ineligible". Roughly speaking, a job accrues eligible wait time when it is blocked due to a resource shortage, and accrues ineligible wait time when it is blocked due to project, user, or group limits. A job can be accruing any of the following kinds of time. A job can only accrue one kind of wait time at a time, and cannot accrue wait time while it is running.

4.9.13.1 Types of Time Accrued

eligible_time

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by the other kinds of time. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f` by job owner, Manager and Operator. Settable by Operator or Manager.

ineligible_time

The amount of wall clock time a job has accrued because the job is blocked by limits on the job's project, owner, or group, or because the job is blocked because of its state.

run_time

The amount of wall clock time a job has spent running.

exiting

The amount of wall clock time a job has spent exiting.

initial_time

The amount of wall clock wait time a job has accrued before the type of wait time has been determined.

4.9.13.2 How Eligible Wait Time Works

A job accrues ineligible_time while it is blocked by project, user, or group limits, such as:

max_run

max_run_soft

max_run_res.<resource name>

max_run_res_soft.<resources>

A job also accrues ineligible_time while it is blocked due to a user hold or while it is waiting for its start time, such as when submitted via

`qsub -a <run-after> …`

A job accrues eligible_time when it is blocked by a lack of resources, or by anything not qualifying as ineligible_time or run_time. A job's eligible_time will only increase during the life of the job, so if the job is requeued, its eligible_time is preserved, not set to zero. The job's eligible_time is not recalculated when a job is qmoved or moved due to peer scheduling.

For information on project, user, and group limits, see section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

The kind of time a job is accruing is sampled periodically, with a granularity of seconds.

A job's eligible_time attribute can be viewed via `qstat -f`.

4.9.13.3 Configuring Eligible Wait Time

To enable using eligible time as the job's wait time, set the eligible_time_enable server attribute to *True*.

4.9.13.4 How Eligible Wait Time Is Used

- If eligible time is enabled, it is used as each job's starving time.
- You can choose to use each job's eligible wait time as the amount of time it is starving. See section 4.9.48, "Starving Jobs", on page 223.
- When a job is requeued, for example being checkpointed and aborted or preempted, its accumulated queue waiting time depends on how that time is calculated:
  - If you are using eligible time, the accumulated waiting time is preserved
  - If you are not using eligible time, the accumulated waiting time is lost

  See section 9.3, "Checkpoint and Restart", on page 418 and section 4.9.33, "Using Preemption", on page 180.

4.9.13.5 Altering Eligible Time

A Manager or Operator can set the value for a job's eligible_time attribute using the `qalter` command, for example:

```
qalter -Weligible_time=<time> <job ID>
```

4.9.13.6 Attributes Affecting Eligible Time

eligible_time_enable

Server attribute. Enables accumulation of eligible time for jobs. Controls whether a job's eligible_time attribute is used as its starving time. See section 4.9.48, "Starving Jobs", on page 223.

On an upgrade from versions of PBS prior to 9.1 or on a fresh install, eligible_time_enable is set to *False* by default.

When eligible_time_enable is set to *False*, PBS does not track eligible_time. Whether eligible_time continues to accrue for a job or not is undefined. The output of `qstat -f` does not include eligible_time for any job. Accounting logs do not show eligible_time for any job submitted before or after turning eligible_time_enable off. Log messages do not include accrual messages for any job submitted before or after turning eligible_time_enable off. If the scheduling formula includes eligible_time, eligible_time evaluates to *0* for all jobs.

When eligible_time_enable is changed from *False* to *True*, jobs accrue eligible_time or ineligible_time or run_time as appropriate. A job's eligible_time is used for starving calculation starting with the next scheduling cycle; changing the value of eligible_time_enable does not change the behavior of an active scheduling cycle.

accrue_type

Job attribute. Indicates what kind of time the job is accruing.

Table 4-7: The accrue_type Job Attribute

Type	Numeric Representation	Type
JOB_INITIAL	*0*	initial_time
JOB_INELIGIBLE	*1*	ineligible_time
JOB_ELIGIBLE	*2*	eligible_time
JOB_RUNNING	*3*	run_time
JOB_EXIT	*4*	exit_time

The job's accrue_type attribute is visible via `qstat` only by Manager, and is set only by the server.

eligible_time

Job attribute. The amount of wall clock wait time a job has accrued because the job is blocked waiting for resources, or any other reason not covered by ineligible_time. For a job currently accruing eligible_time, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f` by job owner, Manager and Operator. Settable by Operator or Manager.

4.9.13.7 Logging

The server prints a log message every time a job changes its accrue_type, with both the new accrue_type and the old accrue_type. These are logged at the 0x0400 event class.

Server logs for this feature display the following information:

- Time accrued between samples
- The type of time in the previous sample, which is one of initial time, run time, eligible time or ineligible time
- The next type of time to be accrued, which is one of run time, eligible time or ineligible time
- The eligible time accrued by the job, if any, until the current sample

Example:

```
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 0 secs of initial_time, new
    accrue_type=eligible_time, eligible_time=00:00:00
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1821 secs of eligible_time, new
    accrue_type=ineligible_time, eligible_time=01:20:22
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 2003 secs of ineligible_time, new
    accrue_type=eligible_time, eligible_time=01:20:22
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 61 secs of eligible_time, new
    accrue_type=run_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 100 secs of run_time, new
    accrue_type=ineligible_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 33 secs of ineligible_time, new
    accrue_type=eligible_time, eligible_time=01:21:23
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 122 secs of eligible_time, new
    accrue_type=run_time, eligible_time=01:23:25
08/07/2007 13:xx:yy;0040;Server@host1;Job;163.host1;job accrued 1210 secs of run_time, new
    accrue_type=exiting, eligible_time=01:23:25
```

The example shows the following changes in time accrual:

- initial to eligible
- eligible to ineligible
- ineligible to eligible
- eligible to running
- running to ineligible
- ineligible to eligible
- eligible to running
- running to exiting

The server also logs the change in accrual when the job's eligible_time attribute is altered using `qalter`. For example, if the job's previous eligible time was 123 seconds, and it has been altered to be 1 hour and 1 minute:

```
Accrue type is eligible_time, previous accrue type was eligible_time for 123 secs, due to qalter
    total eligible_time=01:01:00
```

4.9.13.8 Accounting

Each job's eligible_time attribute is included in the "E" and "R" records in the PBS accounting logs. See section 19.4, "Types of Accounting Log Records", on page 630.

Example:

```
08/07/2007 19:34:06;E;182.Host1;user=user1 group=user1 jobname=STDIN queue=workq ctime=1186494765
    qtime=1186494765 etime=1186494765 start=1186494767 exec_host=Host1/0
    exec_vnode=(Host1:ncpus=1) Resource_List.ncpus=1 Resource_List.nodect=1
    Resource_List.place=pack Resource_List.select=1:ncpus=1 session=4656 end=1186495446
    Exit_status=-12 resources_used.cpupercent=0 resources_used.cput=00:00:00
    resources_used.mem=3072kb resources_used.ncpus=1 resources_used.vmem=13356kb
    resources_used.walltime=00:11:21 eligible_time=00:10:00
```

4.9.13.9 Caveats for Eligible Time

- A job that is dependent on another job can accrue eligible time only after the job on which it depends has finished.
- The action of a hook may affect a job's eligible time. See "Effect of Hooks on Job Eligible Time" on page 73 in the PBS Professional Hooks Guide.
- A subjob that is not running because its array job has hit the max_run_subjobs limit accrues eligible time as long as no other limts have been hit.

4.9.14 Sorting Jobs by Entity Shares (Was Strict Priority)

You can sort jobs according to how much of the fairshare tree is allocated to the entity that owns the job. The fairshare percentages in the fairshare tree describe each entity's share. Using entity shares is sorting jobs on a key, using the fairshare_perc option to the job_sort_key scheduler parameter.

Using entity shares, the jobs from an entity with greater allocation in the fairshare tree run before the jobs with a smaller allocation.

4.9.14.1 Configuring Entity Shares

To configure entity shares, do the following:

- Define fairshare tree entity allocation in `<sched_priv directory>/resource_group`. See section 4.9.19, "Using Fairshare", on page 138. You can use a simple fairshare tree, where every entity's `parent_group` is `root`.
  - Give each entity shares according to desired priority, with higher-priority entities getting larger allocations.
  - Set the unknown_shares scheduler parameter to *1*. This causes any entity not in your list of approved entities to have a tiny allocation, and the lowest priority.

  For example:

```
usr1    60      root    5
usr2    61      root    15
usr3    62      root    15
usr4    63      root    10
usr5    64      root    25
usr6    65      root    30
```

- Set fairshare_perc as the option to job_sort_key, for example:

```
job_sort_key: "fairshare_perc HIGH all"
```

4.9.14.2 Viewing Entity Shares

When you are root, you can use the `pbsfs` command to view the fairshare tree allocations.

4.9.15 Estimating Job Start Time

PBS can use a built-in hook called *PBS_est* that runs the job start time estimator, to estimate when jobs will run, and which vnodes each job will use. PBS estimates job start times and vnodes for all jobs using an asynchronous process, not the PBS server, scheduler, or MoM daemons. This estimator process is started by the PBS_est hook, whose default interval is 120 seconds. By default, the PBS_est hook is disabled.

Jobs have an attribute called *estimated* for reporting estimated start time and estimated vnodes. This attribute reports the values of two read-only built-in resources, *start_time* and *exec_vnode*. Each job's estimated start time is reported in estimated.start_time, and its estimated vnodes are reported in estimated.exec_vnode.

PBS automatically sets the value of each job's estimated.start_time value to the estimated start time for each job.

4.9.15.1 Configuring Start Time Estimation

When a scheduler is backfilling around top jobs, it estimates the start times and exec_vnode for those jobs being backfilled around. By default, PBS_est is disabled. If you want PBS_est to estimate start times and exec_vnode for all jobs, enable it:

- Enable the built-in PBS_est hook:

```
qmgr -c "set pbshook PBS_est enabled = true"
```

The default frequency for PBS_est is 120 seconds. You can set the frequency:

```
qmgr -c "set pbshook PBS_est freq = <interval in seconds>"
```

You set the number of jobs to be backfilled around by setting the server and/or queue backfill_depth attribute to the desired number. See section 4.9.3, "Using Backfilling", on page 105.

Example 4-1: To estimate start times for the top 5 jobs every scheduling cycle, and for all jobs every 3000 seconds:

```
qmgr -c 'set server backfill_depth=5'
qmgr -c 'set pbshook PBS_est enabled = true'
qmgr -c 'set pbshook PBS_est freq = 3000'
```

At each interval, the PBS_est hook checks whether the estimator process is running. If the estimator process is running when the PBS_est hook hits an interval and performs this check, the PBS_est hook does not stop the estimator process or start a new one. It allows the estimator process to finish running. If the estimator process is not running when the PBS_est hook hits an interval, the PBS_est hook starts a new estimator process.

4.9.15.2 Controlling User Access to Start Times and Vnode List

4.9.15.2.i Making Start Time or Vnodes Invisible

You can make job estimated start times and vnodes invisible to unprivileged users by adding resource permission flags to the start_time or exec_vnode resources. To do this, use qmgr to add the resource, and include the i flag, in the same way you would for a custom resource being made invisible.

Example of making start_time and exec_vnode invisible to users:

```
qmgr -c 'set resource start_time flag=i'
qmgr -c 'set resource exec_vnode flag=i'
```

You can always make the start time and vnodes visible again to unprivileged users by removing the flags via qmgr.

See section 5.14.2.3.vi, "Resource Permission Flags", on page 260.

4.9.15.2.ii Allowing Users to See Only Their Own Job Start Times

If you want users to be able to see the start times for their own jobs, but not those of other users, set the server's query_other_jobs attribute to *False*, and do not set the i or r permission flags. Setting the server's query_other_jobs attribute to *False* prevents a user from seeing anything about other users' jobs.

4.9.15.3 Attributes and Parameters Affecting Job Start Time Estimation

backfill
: Server attribute

backfill_depth
: Server attribute

backfill_depth
: Queue attribute

enabled
: Hook attribute

estimated
: Job attribute

freq
: Hook attribute.

help_starving_jobs
: Scheduler parameter

strict_ordering
: Scheduler parameter

4.9.15.4 Viewing Estimated Start Times

You can view the estimated start times and vnodes of jobs using the `qstat` command. If you use the `-T` option to `qstat` when viewing job information, the *Est Start Time* field is displayed. Running jobs are shown above queued jobs.

See "qstat" on page 197 of the PBS Professional Reference Guide.

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

```
qstat -T
                                                                  Est
                                                  Req'd  Req'd    Start
Job ID   Username Queue Jobname SessID NDS TSK Memory Time  S Time
-------  -------- ----- -------- ----- --- --- ------ ----- - -----
5.host1   user1   workq  foojob  12345  1   1  128mb 00:10  R   --
9.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  11:30
10.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  Tu 15
7.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  Jul
8.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  2010
11.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  >5yrs
13.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q   --
```

4.9.15.5 Selecting Jobs By Estimated Start Time

You can use the `qselect` command to select jobs according to their start times by using the `-t` suboption to the `-t` option. This selects jobs according to the value of the estimated.start_time attribute. See "qselect" on page 186 of the PBS Professional Reference Guide.

4.9.15.6 Logging

Whenever a scheduler estimates the start time of a job, it logs the start time. A scheduler does not log the estimated exec_vnode of a job.

4.9.15.7 Caveats and Advice

- The estimated.start_time of a job array is the time calculated for the first queued subjob only.
- Cached estimated start times are only as fresh as the last time PBS calculated them.This should be taken into account when setting the values of the PBS_est hook's freq attribute and backfill_depth.
- The frequency of calculating start times is a trade-off between having more current start time information and using fewer computing cycles for non-job work. The background task of calculating start times can be computationally intensive. This should be taken into account when setting the value of the PBS_est hook's freq attribute. Depending on the size of your partition or site, it is probably a good idea not to set it to less than 10 minutes.
- The best value for the PBS_est hook's freq attribute is workload dependent, but we recommend setting it to two hours as a starting point.
- If your partition or site has short scheduling cycles of a few minutes, and can use backfilling (and at least one of strict ordering or starving jobs), you can have the start times for all jobs calculated at each scheduling cycle. To do this, set backfill_depth to a value greater than the number of jobs the partition or site will ever have, and do not set the PBS_est hook's freq attribute.
- We recommend setting backfill_depth to a value that is less than 100.
- The process of computing the estimated start time for jobs is not instantaneous.
- Note that setting backfill_depth changes your scheduling policy. See section 4.9.3, “Using Backfilling”, on page 105.

4.9.16 Calculating Job Execution Priority

When a scheduler examines jobs, either at the whole partition or complex or within a queue, it gives each job an execution priority, and then uses this job execution priority to select which job(s) to run. Job execution priority is mostly independent of job preemption priority. We discuss only job execution priority in this section.

Some of a scheduler's policy for determining job execution priority is built into PBS, but you can specify how execution priority is determined for most of the policy.

First, a scheduler divides queued jobs into classes. Then it sorts the jobs within each class.

4.9.16.1 Dividing Jobs Into Classes

PBS groups all jobs into classes, and handles one class at a time. There are special classes that supersede queue order, meaning that whether or not queues are being examined separately, the jobs in each of those classes are handled before a scheduler takes queues into account. Those jobs are not ordered according to which queue they reside in. For example, all starving jobs are handled as a group. PBS has one non-special class called *Normal* for all non-special jobs. This class typically contains most PBS jobs. Queue order is imposed on this class, meaning that queue priority affects job execution order if queues are being handled separately.

Job execution classes have a built-in order of precedence. All jobs in the highest class are considered before any jobs in the next class, and so on. Classes are listed in the following table, highest first:

Table 4-8: Job Execution Classes

Class	Description	Sort Applied Within Class
Reservation	Jobs submitted to an advance, standing, or job-specific reservation	Formula, job sort key, submission time
Express	All jobs with preemption priority higher than normal jobs. Preemption priority is defined in scheduler's preempt_prio attribute. Jobs are sorted into this class only when preemption is enabled. See section 4.9.33, "Using Preemption", on page 180.	First by preemption priority, then by preemption time, then starving time, then by formula, then fairshare, then job sort key, followed by job submission time
Preempted	All jobs that have been preempted. See section 4.9.33, "Using Preemption", on page 180.	First by preemption time, then starving time, then by formula, then fairshare, then job sort key, followed by job submission time
Starving (deprecate)	Starving jobs. Jobs are sorted into this class only when starving is enabled by setting help_starving_jobs to *True*. See section 4.9.48, "Starving Jobs", on page 223	Amount of time counted toward starving, then by formula, then fairshare, then job sort key, followed by job submission time
Normal	Jobs that do not belong in any of the special classes	Queue order, if it exists, then formula, then fairshare, then job sort key, followed by job submission time

4.9.16.2 Selecting Job Execution Class

A scheduler places each job in the highest-priority class into which the job can fit. So, for example, if a job is both in a reservation and is starving, the job is placed in the Reservation class.

4.9.16.3 Sorting Jobs Within Classes

Jobs within each class are sorted according to rules specific to each class. The sorting applied to each class is listed in Table 4-8, "Job Execution Classes," on page 135.

- The Reservation class is made up of all jobs in reservations.
  - The Reservation class is sorted within each reservation.
  - The first sort is according to the formula or job_sort_key, depending on which is defined.
  - The second sort key is submission time.
- The Express class is made up of all the jobs that have a higher priority than "normal_jobs" in the preempt_prio scheduler attribute.
  - The Express class is sorted first by applying the rules for preemption priority you set in a scheduler's preempt_prio attribute, making preemption priority the first sort key.
  - The second sort key is the time the job was preempted (if that happened), with the earliest-preempted job having the highest priority (in this sort).
  - The third sort key is the job's starving time, if any.
  - The fourth sort key is the formula, fairshare, or job_sort_key, depending on which is defined.
  - The fifth sort key is job submission time.

Jobs are sorted into this class only when preemption is enabled. See section 4.9.33, “Using Preemption”, on page 180. Please note that execution priority classes are distinct from preemption levels, and are used for different purposes.

For example, if preempt_prio is the following:

```
preempt_prio: "express_queue, starving_jobs, normal_jobs"
```

The Express class contains all jobs that have preemption priority that is greater than that of normal jobs. In this example, the Express class is prioritized with top priority for express queue jobs, followed by starving jobs.

Since preemption levels are applied so that a job is put into the highest preemption level possible, in this example, all starving jobs end up in the Express class.

- The Preempted class is made up of all preempted jobs.
  - The first sort key is the time the job was preempted, with the earliest-preempted job having the highest priority (in this sort).
  - The second sort key is the job's starving time, if any.
  - The third sort key is the formula, fairshare, or job_sort_key, depending on which is defined.
  - The fourth sort key is job submission time.

  When you set the sched_preempt_enforce_resumption scheduler attribute and the strict_ordering scheduler parameter to *True*, a scheduler tries harder to run preempted jobs. By default the attribute is *False*, and in each scheduling cycle, if a top job cannot run now, a scheduler moves on to the next top job and tries to run it. When the attribute and the parameter are *True*, a scheduler treats the job like a top job: it makes sure that no lower-priority job will delay this job, and it backfills around the job.
- The Starving class is made up of all jobs whose wait time qualifies them as starving.
  - The Starving class is sorted first according to the amount of time that counts toward starving for each job. You can use queue wait time or eligible time as starving time. Jobs are sorted into this class only when starving is enabled. See section 4.9.48, “Starving Jobs”, on page 223.
  - The second sort key is the time the job was preempted (if that happened), with the earliest-preempted job having the highest priority (in this sort).
  - The third sort key is the formula, fairshare, or job_sort_key, depending on which is defined.
  - The fourth sort key is job submission time.
- The Normal class is for any jobs that don't fall into any of the other classes. Most jobs are in this class.
  - If queue ordering exists (there are multiple queues, and queues have different priorities set, and round_robin or by_queue is *True*), jobs are sorted first by queue order.
  - If defined, the formula, fairshare, or job sort key is the second sort key.
  - The third sort key is job submission time.

4.9.16.3.i Precedence of Sort Method Used Within Class

If the formula is defined, it overrides fairshare and the job sort key. If fair share is defined, it overrides the job sort key. If none are defined, jobs are ordered by their arrival time in the queue.

For the job sorting formula, see section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

For fairshare, see section 4.9.19, “Using Fairshare”, on page 138.

For sorting jobs on a key, see section 4.9.45, “Sorting Jobs on a Key”, on page 221.

4.9.16.4 Execution Priority Caveats

- Limits are not taken into account when prioritizing jobs for execution. Limits are checked only after setting priority, when selecting a job to run. The only exception is in the Express class, where soft limits may be taken into account, because execution priority for Express class jobs is calculated using preemption priority. For details, see section 4.9.33, “Using Preemption”, on page 180.
- When you issue "`qrun <job ID>`", without the `-H` option, the selected job has execution priority between Reservation and Express.
- Jobs are sorted into the Express class only when preemption is enabled. Similarly, jobs are sorted into the Starving class only when starving is enabled.

4.9.17 Calendaring Jobs

In each scheduling cycle, PBS runs through its list of jobs in the order that you have defined. The backfill depth determines the number of top jobs; these are the highest-priority jobs in its current list. When strict priority and backfilling are in force, and PBS cannot run a top job right now, PBS holds a spot open for that job: PBS finds a future spot in the calendar that fits the job's needs, and doesn't schedule any other jobs that would interfere with the top job.

PBS rebuilds the calendar with each scheduling cycle.

4.9.17.1 Making Jobs Ineligible to be Top Jobs

By default, a job is eligible to be a top job, meaning that PBS holds resources for it if it cannot run right now (the topjob_ineligible job attribute defaults to *False*). If you set the value of a job's topjob_ineligible attribute to *True*, that job cannot become a top job, and PBS does not hold a spot open for that job if it cannot run the job right now. Having the highest priority is not the same as being a top job.

4.9.17.1.i Caveats for Making Jobs Ineligible to be Top Jobs

When sched_preempt_enforce_resumption is set to *True*, all preempted jobs become top jobs, regardless of their setting for topjob_ineligible.

4.9.18 Express Queues

An express queue is a queue whose priority is high enough to qualify as an express queue; the default for qualification is 150, but the cutoff can be set using the preempt_queue_prio scheduler attribute. For information on configuring express queues, see section 2.3.5.3.i, “Express Queues”, on page 24.

You can use express queues as tools to manage job execution and preemption priority.

- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in a scheduler, see section 4.9.16, “Calculating Job Execution Priority”, on page 134.
- You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.9.33, “Using Preemption”, on page 180.

The term "express" is also used in calculating execution priority to mean all jobs that have a preemption level greater than that of the normal_jobs level.

4.9.19 Using Fairshare

Fairshare provides a way to enforce a partition's or site's resource usage policy. It is a method for ordering the start times of jobs based on two things: how a site's resources are apportioned, and the resource usage history of partition or site members. Fairshare ensures that jobs are run in the order of how deserving they are. A scheduler performs the fairshare calculations each scheduling cycle. If fairshare is enabled, all jobs have fairshare applied to them and there is no exemption from fairshare.

You can employ basic fairshare behavior, or a policy of the desired complexity.

The fair_share parameter is a primetime option, meaning that you can configure it for either primetime or non-primetime, or you can specify it for all of the time. You cannot configure different behaviors for fairshare during primetime and non-primetime.

You can use fairshare information calculated by PBS in the job sorting formula. See section 4.9.19.6, "Computing Fairshare Values", on page 143 and section 4.9.21.7, "Using Fairshare in the Formula", on page 151.

4.9.19.1 One Fairshare System Per Scheduler

Each scheduler runs one fairshare system. Each fairshare system is independent of any others. If you are running only the default scheduler (no multischeds), it runs one fairshare system for the entire site. If you are using one or more multischeds, each of the multischeds runs its own fairshare system, and the default scheduler runs one fairshare system.

Each scheduler has its own `usage`, `resource_group`, etc., fairshare files in its `sched_priv` directory, and its own `sched_config` configuration file.

The `pbsfs` command operates on one scheduler's fairshare database at a time. You specify which scheduler's database to operate on using the `pbsfs -I <scheduler name>` option.

In the following sections on fairshare, we describe the behavior for any single scheduler.

4.9.19.2 Outline of How Fairshare Works

The owner of a PBS job can be defined for fairshare purposes to be a user, a group, the job's accounting string, etc. For example, you can define owners to be groups, and can explicitly set each group's relationship to all the other groups by using the tree structure. If you don't explicitly list an owner, it will fall into the "unknown" catchall. All owners in "unknown" get the same resource allotment. You can define one group to be part of a larger department.

You specify which resources to track and how you want usage to be calculated. So if you defined job owners to be groups, then only the usage of groups is considered. PBS tries to ensure that each owner gets the amount of resources that you have set for it.

4.9.19.3 Enabling Basic Fairshare

If the default fairshare behavior is enabled, PBS enforces basic fairshare rules where all users with queued jobs will get an equal share of CPU time. The root vertex of the tree will have one child, the `unknown` vertex. All users will be put under the unknown vertex, and appear as children of the `unknown` vertex.

Enable basic fairshare by doing the following:

- In the scheduler's `sched_config` file, set the scheduler configuration parameter fair_share to *True*
- Uncomment the `unknown_shares` setting so that it is set to `unknown_shares: 10`
- Specify how you want fairshare to work with primetime and non-primetime. If you want separate behavior for primetime and non-primetime, list the fair_share parameter twice, once for each time slot. The default is both. For example:

```
fair_share True prime
fair_share False non_prime
```

Note that a variant of basic fairshare has all users listed in the tree as children of root. Each user can be assigned a different number of shares.

4.9.19.4 Configuring the Fairshare Tree

Fairshare uses a tree structure, where each vertex in the tree represents some set of job owners and is assigned usage *shares*. Shares are used to apportion the partition's or site's resources. The default tree always has a root vertex and an *unknown* vertex. The default behavior of fairshare is to give all users the same amount of the resource being tracked. In order to apportion a partition's or site's resources according to a policy other than equal shares for each user, you create a fairshare tree to reflect that policy. To do this, you edit the `resource_group` file in the scheduler's `sched_priv` directory, which describes that scheduler's fairshare tree.

To configure non-default fairshare, set up a hierarchical tree structure made up of interior vertices and leaves. Interior vertices are *departments*, which can contain both departments and leaves. Leaves are for *fairshare entities*, defined by setting fairshare_entity to one of the following: *euser, egroup, egroup:euser, Account_Name*, or *queue*. Apportioning of resources for the partition or site is among these entities. These entities' usage of the designated resource is used in determining the start times of the jobs associated with them. All fairshare entities must be the same type. If you wish to have a user appear in more than one department, you can use `egroup:euser` to distinguish between that user's different resource allotments. Note that in the `resource_group` file and in the output of `pbsfs`, interior (non-leaf) vertices are referred to as "groups", and exterior (leaf) vertices are referred to as "users".

Table 4-9: Using Fairshare Entities

Keyword	Fairshare Entities	Purpose
euser	Username	Individual users are allotted shares of the resource being tracked. Each username may only appear once, regardless of group.
egroup	OS group name	Groups as a whole are allotted shares of the resource being tracked.
egroup:euser	Combinations of username and group name	Useful when a user is a member of more than one group, and needs to use a different allotment in each group.
Account_Name	Account IDs	Shares are allotted by account string (Account_Name job attribute).
queue	Queues	Shares are allotted between queues.

4.9.19.4.i Allotting Shares in the Tree

You assign shares to each vertex in the tree. The actual number of shares given to a vertex or assigned in the tree is not important. What is important is the ratio of shares among each set of sibling vertices. Competition for resources is between siblings only. The sibling with the most shares gets the most resources.

4.9.19.4.ii Shares Among Unknown Entities

The root vertex always has a child called `unknown`. Any entity not listed in the scheduler's `resource_group` file will be made a child of `unknown`, designating the entity as unknown. The shares used by unknown entities are controlled by two parameters in the scheduler's `sched_config` file: unknown_shares and fairshare_enforce_no_shares.

The parameter unknown_shares controls how many shares are assigned to the `unknown` vertex. The shipped `sched_config` file contains this line:

```
#unknown_shares 10
```

If you leave unknown_shares commented out, the `unknown` vertex will have 0 shares. If you simply remove the "#", the `unknown` vertex's shares default to 10. The children of the `unknown` vertex have equal amounts of the shares assigned to the `unknown` vertex.

The parameter fairshare_enforce_no_shares controls whether an entity without any shares can run jobs. If fairshare_enforce_no_shares is *True*, entities without shares cannot run jobs. If it is set to *False*, entities without any shares can run jobs, but only when no other entities' jobs are available to run.

4.9.19.4.iii Format for Describing the Tree

The file describing the fairshare tree contains four columns to describe the vertices in the tree. Here is the format for the columns:

<Vertex name> <vertex fairshare ID> <parent of vertex> <#shares>

The columns are for a vertex's name, its fairshare ID, the name of its parent vertex, and the number of shares assigned to this (not the parent) vertex. Vertex names and IDs must be unique. Vertex IDs are integers. The top row in `resource_group` contains information for the first vertex, rather than column labels.

Neither the root vertex nor the `unknown` vertex is described in the `resource_group` file. They are always added automatically. Parent vertices must be listed before their children.

For example, we have a tree with two top-level departments, Math and Phys. Under Math are the users Bob and Tom as well as the department Applied. Under Applied are the users Mary and Sally. Under Phys are the users John and Joe. Our `<sched_priv directory>/resource_group` looks like this:

```
Math            100     root        30
Phys            200     root        20
Applied         110     Math        25
Bob             101     Math        15
Tom             102     Math        10
Mary            111     Applied     1
Sally           112     Applied     2
John            201     Phys        2
Joe             202     Phys        2
```

If you wish to use egroup:euser as your entity, and Bob to be in two groups pbsgroup1 and pbsgroup2, and Tom to be in two groups pbsgroup2 and pbsgroup3:

```
Math            100     root        30
Phys            200     root        20
Applied         110     Math        20
pbsgroup1:Bob   101     Phys        20
pbsgroup2:Bob   102     Math        20
pbsgroup2:Tom   103     Math        10
pbsgroup3:Tom   104     Applied     10
```

When a user submits a job using `-Wgroup_list=<group>`, the job's egroup will be <group>. For example, user Bob is in pbsgroup1 and pbsgroup2. Bob uses "`qsub -Wgroup_list= pbsgroup1`" to submit a job that will be charged to pbsgroup1, and "`qsub -Wgroup_list=pbsgroup2`" to submit a job that will be charged to pbsgroup2.

The first and third fields are alphanumeric. The second and fourth fields are numeric. Fields can be separated by spaces and tabs.

4.9.19.4.iv Moving Entities within Fairshare Tree

To move an entity within the fairshare tree, change its parent:

1. Edit `<sched_priv directory>/resource_group`. Change the parent (column 3) to the desired parent
2. HUP or restart the scheduler

4.9.19.4.v Removing Entities from Fairshare Tree

You may want to remove an entity from the fairshare tree, either because they no longer run jobs, or because you don't want them to have their own place in the tree. When an entity that is not in the fairshare tree runs a job, their past and future usage, including that for jobs running while you remove the entity, shows up in the Unknown group. To remove an entity from the fairshare tree:

1. Edit the `resource_group` file to remove the entity line
2. HUP or restart the scheduler

If you do not want an entity's usage to show up in the Unknown group, use `pbsfs -e [-I <multisched name>]` to remove the usage. If you are working on a partition managed by a multisched, you must specify the name of the multisched:

1. Prevent jobs from being scheduled
2. Run `pbsfs -e [-I <multisched name>]`
3. Resume scheduling jobs

If you have removed a user from the PBS partition or complex and don't want their usage to show up any more:

1. Prevent jobs from being scheduled
2. Edit the `resource_group` file
3. Run `pbsfs -e [-I <multisched name>]`
4. Resume scheduling jobs

4.9.19.5 Resource Usage for Fairshare

4.9.19.5.i Tracking Resource Usage

You choose which resources to track and how to compute the usage by setting the fairshare_usage_res scheduler configuration parameter in the `sched_config` file to the fairshare resource formula you want. This parameter can contain the following:

- Built-in and custom job resources

  When you use a resource in the fairshare resource formula, if a value exists for resources_used.<resource name>, this value is used in the fairshare resource formula. Otherwise, the value is taken from Resource_List.<resource name>.

- Mathematical operators

  You can use standard Python operators and the operators in the Python math module.

The default for the tracked resource is cput (CPU time).

4.9.19.5.ii Adding Usage

An entity's usage always starts at 1. Resource usage tracking begins when a scheduler is started. Each scheduler cycle, the scheduler adds the usage increment between this cycle and the previous cycle to its sum for the entity.

A static resource does not change its usage from one cycle to the next. If you use a static resource such as ncpus, the amount being tracked will not change during the lifetime of the job; it will only be added once when the job starts.

Note that if a job ends between two scheduling cycles, its resource usage for the time between the previous scheduling cycle and the end of the job will not be recorded. A scheduler's default cycle interval is 10 minutes. The scheduling cycle can be adjusted via the `qmgr` command. Use

```
Qmgr: set sched [sched name>] scheduler_iteration=<new value>
```

If the fairshare resource formula in fairshare_usage_res evaluates to a negative number, PBS uses zero instead. So there is no way to accumulate negative usage.

4.9.19.5.iii Decaying Usage

Each entity's usage is *decayed*, or reduced periodically, at the interval set in the fairshare_decay_time parameter in the `sched_config` file. This interval defaults to 24 hours.

The amount by which usage is decayed is set in the fairshare_decay_factor scheduler parameter.

An entity with a lot of current or recent usage will have low priority for starting jobs, but if the entity cuts resource usage, its priority will go back up after a few decay cycles.

4.9.19.5.iv Setting Decay Interval and Factor

You set the interval at which usage is decayed by setting the fairshare_decay_time scheduler parameter to the desired time interval. The default value for this interval is 24 hours. For example, to set this interval to 14 hours and 23 minutes, put this line in the `sched_config` file:

```
fairshare_decay_time: 14:23:00
```

You set the decay factor by setting the fairshare_decay_factor scheduler parameter to the desired multiplier for usage. At each decay interval, the usage is multiplied by the decay factor. This attribute is a float whose value must be between 0 and 1. The value must be greater than 0 and less than 1. The default value for this multiplier is 0.5. For example, to set this multiplier to 70 percent, put this line in `sched_config:`

```
fairshare_decay_factor: .7
```

4.9.19.5.v Examples of Setting Fairshare Usage

To use CPU time as the resource to be tracked, put this line in `sched_config`:

```
fairshare_usage_res: cput
```

To use ncpus multiplied by walltime as the resource to be tracked, put this line in `sched_config`:

```
fairshare_usage_res: ncpus*walltime
```

An example of a more complex formula:

```
fairshare_usage_res: "ncpus*pow(walltime,.25)*fs_factor"
```

4.9.19.5.vi Fairshare Resource Advice

We recommend including a time-based resource in the fairshare formula so that usage will grow over time.

4.9.19.5.vii Viewing and Managing Fairshare Usage Data

The `pbsfs` command provides a command-line tool for viewing and managing some fairshare data. You can display the data as a tree, a table, or by entity. You can print all information about an entity, or set an entity's usage to a new value. You can force an immediate decay of all the usage values in the tree. You can compare two fairshare entities. You can also remove all entities from the `unknown` department. This makes the tree easier to read. The tree can become unwieldy because entities not listed in the `resource_group` file all land in the `unknown` group. See "pbsfs" on page 31 of the PBS Professional Reference Guide.

To change fairshare resource usage data, do the following:

1. Stop scheduling:
   Qmgr: set sched [<sched name>] scheduling = false
2. Wait until the current scheduling cycle finishes. Check the scheduler's log.
3. Trim the fairshare tree:
   pbsfs -e [-I <multisched name>]
4. Set each entity's usage to one (cannot be zero). For each leaf entity:
   pbsfs -s <entity name> 1 [-I <multisched name>]
5. Start scheduling:
   Qmgr: set sched [<sched name>] scheduling = true

The fairshare usage data is written to the file `usage` file at each scheduling cycle. The usage data is always up to date.

For more information on using the `pbsfs` command, see "pbsfs" on page 31 of the PBS Professional Reference Guide

4.9.19.6 Computing Fairshare Values

PBS provides fairshare_perc, fairshare_tree_usage, and fairshare_factor as terms to use in the job sorting formula. You can also use fairshare_perc as an argument to the job_sort_key scheduler parameter.

4.9.19.6.i Computing Target Usage for Each Vertex (fairshare_perc)

How much resource each entity should use is its target usage, computed in fairshare_perc. Target usage is the percentage of the shares in the tree allotted to the entity. Target usage does not take history into account.

A vertex's portion of all the shares in the tree is its fairshare_perc. This is computed for all of the vertices in the tree. Since the leaves of the tree represent the entities among which resources are to be shared, their fairshare_perc sums to 100 percent. Only the leaf nodes sum to 100%; if all of the nodes were summed, the result would be greater then 100%. Only the leaf nodes of the tree are fairshare entities.

A scheduler computes the fairshare_perc for the vertices this way:

First, it gives the root of the tree a fairshare_perc of 100 percent. It proceeds down the tree, finding the fairshare_perc first for immediate children of root, then their children, ending with leaves.

1. For each internal vertex A:

   sum the shares of its children;
2. For each child J of vertex A:

   divide J's shares by the sum to normalize the shares;

   multiply J's normalized shares by vertex A's fairshare_perc to find J's fairshare_perc.

The fairshare_perc value can be used in the job sorting formula and as an argument to the job_sort_key scheduler parameter.

4.9.19.6.ii Computing Effective Usage (fairshare_tree_usage)

An entity's effective usage is fairshare_tree_usage, and is a value between *0* and *1*.

PBS calculates the value for fairshare_tree_usage this way:

For root's children:

fairshare_tree_usage = percent total usage

For entities below root's children:

*fairshare_tree_usage = entity's percent total usage + ((parent's effective usage - entity's percent total usage) * entity's relative percent of shares within sibling group)*

where

entity's percent total usage = entity's usage / all usage in partition or complex

Summing effective usage for all leaves in the tree does not yield a useful number (such as 1).

4.9.19.6.iii Computing Relative Usage (fairshare_factor)

An entity's relative usage allows direct comparison between entities. Relative usage is fairshare_factor, and is a value between *0* and *1*. A value of *0.5* means that an entity is using exactly its target usage. A higher value indicates less resource usage by the entity, meaning that the entity is more deserving. Calculated this way:

2^-(fairshare_tree_usage / entity's fairshare_perc)

4.9.19.6.iv Example of Computing Fairshare Values

Example 4-2: The following fairshare tree shows shares and usage for two groups, each with two people:

Table 4-10: Example Fairshare Tree

Vertex	Shares	Actual Usage	% Total Usage	% Group Shares	Target Usage: fairshare_perc	Effective Usage: fairshare_tree_usage	Relative Usage: fairshare_factor
root		1200	100	1.0	1.0	1.0	1.0
group1	40	200	0.1667	0.4	0.4	0.167	0.75
Bob	50	100	0.0833	0.5	0.2	0.125	0.65
Cathy	50	100	0.0833	0.5	0.2	0.125	0.65
group2	60	1000	0.833	0.6	0.6	0.833	0.38
Suzy	60	0	0	0	0.36	0.5	0.38
Scott	40	1000	0.833	1	0.24	0.833	0.09

Comparing Suzy and Bob:

Bob:

Percent total usage: 100/1200 = 0.083

Parent's effective usage: 0.1667

Bob's percentage of shares in group: Bob's 50 shares / (Bob's 50 shares+ Cathy's 50 shares) = 0.5

Bob's effective usage: Bob's percent total usage: 0.0833 + (group1's effective usage: 0.1667 - Bob's percent total usage: 0.083) * 0.5 = 0.125

Relative usage formula: 2^-(0.125/0.2) = 0.648

Suzy:

Percent total usage: 0/1200 = 0

Parent's effective usage: 1000/1200 = 0.833

Suzy's percentage of shares in group: Suzy's 60 shares / (Suzy's 60 shares + Scott's 40 shares) = 0.6

Suzy's effective usage: Suzy's percent total usage: 0 + (group2's usage: 0.833 - Suzy's usage: 0) * 0.6 = 0.5

Relative usage formula: 2^-(0.5/0.36): 0.382

Even though Suzy had a higher fairshare_perc than Bob and less usage than Bob, her relative usage (fairshare_factor) is quite a bit lower than his, because of the huge amount Scott used.

Output of `pbsfs -g`:

Our `pbsfs` output uses the same fairshare data as the previous example.

```
# ./pbsfs -g scott
fairshare entity: scott
Resgroup                : 11
cresgroup               : 15
Shares                  : 40
Percentage              : 24.000000%
fairshare_tree_usage    : 0.832973
usage                   : 1000 (cput)
usage/perc              : 4167
Path from root:
TREEROOT  :     0       1201 / 1.000 = 1201
group2    :    11       1001 / 0.600 = 1668
scott     :    15       1000 / 0.240 = 4167
```

4.9.19.7 Choosing Which Job to Run

4.9.19.7.i Finding the Most Deserving Entity

The most deserving entity is found by starting at the root of the tree, comparing its immediate children, finding the most deserving, then looking among that vertex's children for the most deserving child. This continues until a leaf is found. In a set of siblings, the most deserving vertex will be the vertex with the lowest ratio of resource usage divided by fairshare_perc.

4.9.19.7.ii Sorting and Selecting Jobs to Run

The job to be run next is selected from the set of jobs belonging to the most deserving entity. The jobs belonging to the most deserving entity are sorted according to the methods a scheduler normally uses. This means that fairshare effectively becomes the primary sort key. If the most deserving job cannot run, then the next most is selected to run, and so forth. All of the most deserving entity's jobs are examined first, then those of the next most deserving entity, etcetera.

At each scheduling cycle, a scheduler attempts to run as many jobs as possible. It selects the most deserving job, runs it if it can, then recalculates to find the next most deserving job, runs it if it can, and so on.

When a scheduler starts a job, all of the job's requested usage is added to the sum for the owner of the job for one scheduling cycle. The following cycle, the job's usage is set to the actual usage that occurred between the first and second cycles. This prevents one entity from having all its jobs started and using up all of the resource in one scheduling cycle.

4.9.19.8 Files and Parameters Used in Fairshare

`sched_config`

File in the directory specified in the scheduler's sched_priv attribute

PBS uses the following parameters from `sched_config` to compute fairshare values:

Table 4-11: `sched_config` Parameters used in Fairshare

Parameter	Use
fair_share	[*True*/*False*] Enable or disable fairshare
fairshare_usage_res	Resource whose usage is to be tracked; default is cput
fairshare_decay_factor	Amount to decay usage at each decay interval
fairshare_decay_time	Decay interval; default is 24 hours
unknown_shares	Number of shares for unknown vertex; default 10, 0 if commented out
fairshare_entity	The kind of entity which is having fairshare applied to it. Leaves in the tree are this kind of entity. Default: euser.
fairshare_enforce_no_shares	If an entity has no shares, this controls whether it can run jobs. T: an entity with no shares cannot run jobs. F: an entity with no shares can only run jobs when no other jobs are available to run.
by_queue	If on, queues cannot be designated as fairshare entities, and fairshare will work queue by queue instead of on all jobs at once.

`resource_group`

File in the directory specified in the scheduler's sched_priv attribute

Contains the description of the fairshare tree.

`usage`

File in the directory specified in the scheduler's sched_priv attribute

Contains the usage database. Written by PBS. Do not edit. Written each scheduling cycle.

scheduler_iteration

Scheduler attribute. Specifies scheduler cycle frequency; default is 10 minutes.

`Qmgr: set sched <scheduler name> scheduler_iteration=<new value>`

resources_used.<resource name>

Job attribute. Contains resources used for tracking usage. Default is cput.

4.9.19.9 Ways to Use Fairshare

4.9.19.9.i Fairshare for Partition Or Complex or Within Queues

You can use fairshare to compare all jobs in the partition managed by a scheduler, or within each queue. Fairshare within a queue means that a scheduler examines the jobs in a queue, and compares them to each other, to determine which job to start next.

To use fairshare for the entire partition or complex, set the by_queue and round_robin scheduler configuration parameters to *False*.

To use fairshare within queues, set the by_queue scheduler parameter to *True*, and round_robin to *False*. If you want to examine queues in a particular order, prioritize the queues by setting each queue's priority attribute.

The scheduler configuration parameter by_queue in the `sched_config` file is set to *True* by default.

If by_queue is *True*, queues cannot be designated as fairshare entities.

4.9.19.9.ii Altering Fairshare According to Queue

You can introduce a fairshare factor that is different at each queue. To do this, create a custom floating point resource, and set each queue's resources_default.<resource name> to the desired value. Use this resource in the fairshare_usage_res computation. If you do not set this value at a queue, PBS uses zero for the value. To avoid having to set a value at multiple queues, you can set the servers's resources_default.<resource name> to the default value for all queues where the value is unset. The server value is used only where the queue value is unset; where the queue value is set, the queue value takes precedence.

For example, to reduce the priority for jobs in the "expensive" queue by assigning them twice the usage of the jobs in workq:

- Define the resource:

```
Qmgr: create resource fs_factor type = float, flag = i
```

- Set the resource values:

```
Qmgr: set server resources_default.fs_factor = 1
Qmgr: set queue workq resources_default.fs_factor = 0.3
Qmgr: set queue expensive resources_default.fs_factor = 0.6
```

- Edit sched_config:

```
fairshare_usage_res: "fs_factor*ncpus*walltime"
```

4.9.19.9.iii Using Fairshare in Job Execution Priority

Jobs are sorted as specified by the formula in job_sort_formula, if it exists, then by fairshare, if it is enabled, or if neither of those is used, by job_sort_key. The job sorting formula can use the following calculated values: fairshare_perc, the percentage of the fairshare tree for this job's entity, fairshare_tree_usage, an entity's effective usage, and fairshare_factor, an entity's comparative usage. See section 4.9.16, “Calculating Job Execution Priority”, on page 134.

4.9.19.9.iv Using Fairshare in Job Preemption Priority

You can use the *fairshare* preemption level in determining job preemption priority. This level applies to jobs whose owners are over their fairshare allotment. See section 4.9.33, “Using Preemption”, on page 180.

4.9.19.10 Fairshare Restrictions

- Entity shares (strict priority):

  If you enable entity shares (strict priority), you use the same fairshare tree that you would use for fairshare. Fairshare and entity shares (strict priority) are incompatible, so in order to use entity shares, you disable fairshare by setting fair_share to *False*. For how to configure entity shares, see section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 131.

- Requeued jobs:

  When a job is requeued, it normally retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job's owner has gone over their fairshare limit, etc.

- With strict_ordering or backfilling:

  We do not recommend using fairshare with strict_ordering, or with strict_ordering and backfilling. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group as the top job will be chosen as the filler job. The usage from the filler job will lower the priority of the most deserving, i.e. top, job. This could delay the execution of the top job.

However, if all of your leaf entities are children of root (the tree has only two levels), and all users tend to submit the same size jobs, results may be useful.

- With fairshare_perc option to job_sort_key:

  Do not use fairshare when using the fairshare_perc option to job_sort_key. You can still use the value of fairshare_perc in the job sorting formula.

- Static resources:

  Do not use static resources such as ncpus as the resource to track. A scheduler adds the incremental change in the tracked resource at each scheduling cycle, and a static resource will not change.

- With help_starving_jobs:

  Do not use fairshare with help_starving_jobs.

4.9.19.11 Fairshare Caveats and Advice

- The most deserving entity can change with every scheduling cycle, if each time a job is run, it changes usage sufficiently.
- Fairshare dynamically reorders the jobs with every scheduling cycle. Strict ordering is a rule that says we always run the next-most-deserving job. If there were no new jobs submitted, strict ordering could give you a snapshot of how the jobs would run for the next *n* days. Hence fairshare appears to break that. However, looked at from a dynamic standpoint, fairshare is another element in the strict order.
- The half_life parameter is **deprecated** and has been replaced by the fairshare_decay_time parameter.
- Beware of overflow: PBS stores fairshare allocations in a signed integer (32-bit on Linux x86_64 platforms), and fairshare usage in a long (64-bit on Linux x86_64 platforms)

4.9.20 FIFO Scheduling

With FIFO scheduling, PBS runs jobs in the order in which they are submitted. You can use FIFO order for all of the jobs in your partition or complex, or you can go queue by queue, so that the jobs within each queue are considered in FIFO order.

4.9.20.1 Configuring Basic FIFO Scheduling

To configure basic FIFO scheduling, whether across all a scheduler's partition or queue by queue, set the following scheduler parameters and queue/server attribute to these values:

```
round_robin:              False   ALL
job_sort_key:             (commented out)
fair_share                False   ALL
backfill_depth:           0
job_sort_formula: (unset)
```

4.9.20.2 FIFO for Entire Partition Or Complex

To configure FIFO across your entire partition or complex, follow the steps above and do one of the following:

- Use only one execution queue
- Set the by_queue scheduler parameter to *False*

4.9.20.3 Queue by Queue FIFO

To configure FIFO for each queue separately, first decide how you want queues to be selected. You can set the order in which PBS chooses queues from which to run jobs, or you can allow the queues to be selected in an undefined way. First configure this scheduler as in Section 4.9.20.1, "Configuring Basic FIFO Scheduling".

- To allow queues to be selected in an undefined way, set the by_queue scheduler parameter to *True*.
- To set the order in which queues are selected, do the following:
  - Specify a priority for each queue
  - Set the by_queue scheduler parameter to *True*

4.9.20.4 FIFO with Strict Ordering

If your jobs must run exactly in submission order, you can use strict ordering with FIFO scheduling. If you use strict ordering with FIFO scheduling, this means that when the job that is supposed to run next cannot run, no jobs can run. This can result in less throughput than you could otherwise achieve. To avoid that problem, you can use backfilling. See the following section.

To use strict ordering with FIFO scheduling, use the following scheduler parameter settings in `<sched_priv directory>/sched_config` and queue/server attribute settings:

```
strict_ordering:          True    ALL
round_robin:              False   ALL
job_sort_key:             (commented out)
fair_share                False   ALL
backfill_depth:           0
job_sort_formula: (unset)
```

4.9.20.5 FIFO with Strict Ordering and Backfilling

If you want to run your jobs in submission order, except for backfilling around top jobs that are stuck, use the following:

```
strict_ordering:          True    ALL
round_robin:              False   ALL
job_sort_key:             (commented out)
fair_share                False   ALL
backfill_depth:           <depth>
job_sort_formula: (unset)
```

4.9.21 Using a Formula for Computing Job Execution Priority

You can choose to use a formula by which to sort jobs at the finest-granularity level. The formula can only direct how jobs are sorted at the finest level of granularity. However, that is where most of the sorting work is done.

When a scheduler sorts jobs according to the formula, it computes a priority for each job. The priority computed for each job is the value produced by the formula. Jobs with a higher value get higher priority. See section 4.9.16.3, “Sorting Jobs Within Classes”, on page 135 for how the formula is used in setting job execution priority.

Only one formula is used to prioritize all jobs. At each scheduling cycle, the formula is applied to all jobs, regardless of when they were submitted. If you change the formula, the new formula is applied to all jobs.

For example, if you submit some jobs, change the formula, then submit more jobs, the new formula is used for all of the jobs, during the next scheduling cycle.

You can set a job priority threshold so that jobs with priority at or below the specified value do not run. See section 4.9.21.10, “Setting Minimum Job Priority Value for Job Execution”, on page 152.

You may find that the formula is most useful when you use it with custom resources inherited by or allocated to jobs. For example, you may want to route all jobs from a particular project to a queue where they inherit a specific value for a custom resource. Other jobs may end up at a different queue, where they inherit a different value, or they may inherit no value. You can then use this custom resource in the formula as a way to manage job priority. See section 14.3, “Allocating Resources to Jobs”, on page 505, and section 4.9.8, “Using Custom and Default Resources”, on page 113.

It may be helpful if these custom resources are invisible and unrequestable by users. See section 4.9.21.12, “Examples of Using Resource Permissions in Job Sorting Formula”, on page 153.

4.9.21.1 When the Formula is Applied

Once you set job_sort_formula via `qmgr`, it takes effect with the following scheduling cycle.

Variables are evaluated at the start of the scheduling cycle.

4.9.21.2 Configuring the Job Sorting Formula

- Define the formula:

  You specify the formula in the server's job_sort_formula attribute. To set the job_sort_formula attribute, use the `qmgr` command. When specifying the formula, be sure to follow the requirements for entering an attribute value via `qmgr`: strings containing whitespace, commas, or other special characters must be enclosed in single or double quotes. See “Caveats and Restrictions for Setting Attribute and Resource Values” on page 159 of the PBS Professional Reference Guide. Format:

  ```
  Qmgr: s s job_sort_formula = "<formula>"
  ```

- Optional: set a priority threshold. See section 4.9.21.10, “Setting Minimum Job Priority Value for Job Execution”, on page 152

4.9.21.3 Requirements for Creating Formula

The job sorting formula must be created at the server host.

Under Linux, root privilege is required in order to operate on the job_sort_formula server attribute.

4.9.21.4 Format of Formula

The formula must be valid Python, and must use Python syntax.The formula can be made up of any number of *expressions*, where expressions contain *terms* which are added, subtracted, multiplied, or divided. You can use parentheses, exponents, unary + and - operators, and the ternary operator (which must be Python). All operators use standard mathematical precedence. The formula can use standard Python mathematical operators and those in the Python math module.

The formula can be any length.

The range for the formula is defined by the IEEE floating point standard for a double.

4.9.21.5 Units in Formula

The variables you can use in the formula have different units. Make sure that some terms do not overpower others, by normalizing them where necessary. Resources like ncpus are integers, size resources like mem are in kb, so 1gb is 1048576kb, and time-based resources are in seconds (e.g. walltime). Therefore, if you want a formula that combines memory and ncpus, you'll have to account for the factor of 1024 difference in the units.

The following are the units for the supported built-in resources:

Table 4-12: Job Sorting Formula Units

Resource	Units	Example
Time resources	*Integer number of seconds*	300
Memory	*kb*	1gb => 1048576kb
ncpus	*Integer*	8

Example 4-3: If you use '1 * ncpus + 1 * mem', where mem=2mb, ncpus will have almost no effect on the formula result. However, if you use '1024 * ncpus + 1 * mem', the scaled mem won't overpower ncpus.

Example 4-4: You are using gb of mem:

```
Qmgr: s s job_sort_formula='1048576 * ncpus + 2 * mem'
```

Example 4-5: If you want to add days of walltime to queue priority, you might want to multiply the time by 0.0000115, equivalent to dividing by the number of seconds in a day:

```
Qmgr: s s job_sort_formula = '.0000115*walltime + queue_priority'
```

Note that a Python bug may make it necessary to multiply by 1.0 in order to prevent rounding to the nearest integer.

4.9.21.6 Resources in Formula

The formula can use resources in the job's Resource_List attribute, but no other resources. The resources in the job's Resource_List attribute are the numeric job-level resources, and may have been explicitly requested, inherited, or summed from consumable host-level resources. See section 5.9.2, "Resources Requested by Job", on page 245.

This means that all variables and coefficients in the formula must be resources that were either requested by the job or were inherited from defaults at the server or queue. These variables and coefficients can be custom numeric resources inherited by the job from the server or queue, or they are long integers or floats.

You may need to create custom resources at the server or queue level to be used for formula coefficients. See section 4.9.8, "Using Custom and Default Resources", on page 113.

4.9.21.7 Using Fairshare in the Formula

PBS provides the following fairshare values for use as keywords in the job sorting formula:

Table 4-13: Fairshare Terms in Formula

Keyword	Description
fairshare_perc (was fair_share_perc)	Percentage of fairshare tree allotted to this job's entity. See section 4.9.19.6.i, "Computing Target Usage for Each Vertex (fairshare_perc)", on page 143.
fairshare_tree_usage	Value between 0 and 1, reflecting an entity's effective usage. See *section 4.9.19.6.ii, "Computing Effective Usage (fairshare_tree_usage)", on page 143*.
fairshare_factor	Value between 0 and 1, which allows direct comparison between entities. A value of 0.5 means that an entity is using exactly its allotted usage. A higher value indicates less resource usage by the entity, meaning that the entity is more deserving. See *section 4.9.19.6.iii, "Computing Relative Usage (fairshare_factor)", on page 144*.

See section 4.9.19, "Using Fairshare", on page 138.

4.9.21.8 Terms in Formula

Table 4-14: Terms in Job Sorting Formula

Terms		Allowable Value
Constants		<number> or <number>.<number>
Attributes, keywords, parameters, etc.	queue_priority	Value of priority attribute for queue in which job resides
	job_priority	Value of the job's priority attribute
	fairshare_perc	Percentage of fairshare tree allotted to this job's entity
	fairshare_tree_usage	The effective usage by the entity
	fairshare_factor	Value allowing direct comparison between entities
	eligible_time	Amount of wait time job has accrued while waiting for resources
	accrue_type	Kind of time job is accruing. See section 4.9.13, "Eligible Wait Time for Jobs", on page 126.
Resources		ncpus
		mem
		walltime
		cput
Custom numeric job-wide resources		Uses the amount requested, not the amount used. Must be of type long, float, or size. See section 5.14.2.2, "Custom Resource Values", on page 258.

4.9.21.9 Modifying Coefficients For a Specific Job

Formula coefficients can be altered for each job by using the `qalter` command to change the value of that resource for that job. If a formula coefficient is a constant, it cannot be altered per-job.

4.9.21.10 Setting Minimum Job Priority Value for Job Execution

You can specify a minimum job priority value for jobs to run by setting the job_sort_formula_threshold scheduler attribute. If the value calculated for a job by the job sorting formula is at or below this value, the job cannot run during this scheduling cycle.

4.9.21.11 Examples of Using the Job Sorting Formula

Examples of formulas:

Example 4-6: 10 * ncpus + 0.01*walltime + A*mem

Here, "A" is a custom resource.

Example 4-7: ncpus + 0.0001*mem

Example 4-8: To change the formula on a job-by-job basis, alter the value of a resource in the job's Resource_List.<resource name>. So if the formula is A *queue_priority + B*job_priority + C*ncpus + D*walltime, where A-D are custom numeric resources. These resources can have a default value via resources_default.A ... resources_default.D. You can change the value of a job's resource through `qalter`.

Example 4-9: ncpus*mem

Example 4-10: Set via `qmgr`:

```
qmgr -c 'set server job_sort_formula= 5*ncpus+0.05*walltime'
```

Following this, the output from `qmgr -c 'print server'` will look like

```
set server job_sort_formula="5*ncpus+0.05*walltime"
```

Example 4-11:

```
Qmgr: s s job_sort_formula=ncpus
```

Example 4-12:

```
Qmgr: s s job_sort_formula='queue_priority + ncpus'
```

Example 4-13:

```
Qmgr: s s job_sort_formula='5*job_priority + 10*queue_priority'
```

Example 4-14: Sort jobs using the value of ncpus x walltime:

Formula expression: "ncpus * walltime"

Submit these jobs:

Job 1: ncpus=2 walltime=01:00:00 -> 2*60s = 120

Job 2: ncpus=1 walltime=03:00:00 -> 1*180s = 180

Job 3: ncpus=5 walltime=01:00:00 -> 5*60s = 300

The scheduler logs the following:

```
Job ;1.host1;Formula Evaluation = 120
Job ;2.host1;Formula Evaluation = 180
Job; 3.host1;Formula Evaluation = 300
```

The jobs are sorted in the following order:

Job 3

Job 2

Job 1

4.9.21.12 Examples of Using Resource Permissions in Job Sorting Formula

See section 5.14.2.3.vi, “Resource Permission Flags”, on page 260 for information on using resource permissions.

Example 4-15: You may want to create per-job coefficients in your job sorting formula which are set by system defaults and which cannot be viewed, requested or modified by the user. To do this, you create custom resources for the formula coefficients, and make them invisible to users. In this example, A, B, C and D are the coefficients. You then use them in your formula:

A *(Queue Priority) + B*(Job Class Priority) + C*(CPUs) + D*(Queue Wait Time)

Example 4-16: You may need to change the priority of a specific job, for example, have one job or a set of jobs run next. In this case, you can define a custom resource for a special job priority. If you do not want users to be able to change this priority, set the resource permission flag for the resource to *r*. If you do not want users to be able to see the priority, set its resource permission flag to *i*. For the job or jobs that you wish to give top priority, use `qalter` to set the special resource to a value much larger than any formula outcome.

Example 4-17: To use a special priority:

```
sched_priority = W_prio * wait_secs + P_prio * priority + ... +  special_priority
```

Here, special_priority is very large.

4.9.21.13 Caveats and Error Messages

- If the formula overflows or underflows the sorting behavior is undefined.
- If you set the formula to an invalid formula, `qmgr` will reject it, with one of the following error messages:

  ```
  "Invalid Formula Format"
  "Formula contains invalid keyword"
  "Formula contains a resource of an invalid type"
  ```
- If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:

  ```
  "1234.mars;Formula evaluation for job had an error.  Zero value will be used"
  ```
- The job sorting formula must be set via `qmgr` at the server host.
- When a job is moved to a new server or queue, it inherits new default resources from that server or queue. If it is moved to a new server, it is prioritized according to the formula on that server, if one exists.
- If the job is moved to another server through peer scheduling and the pulling server uses queue priority in its job sorting formula, the queue priority used in the formula will be that of the queue to which the job is moved.
- If you are using FIFO scheduling, the job_sort_formula server attribute must be unset.
- If you are using eligible time in the formula, and eligible_time_enable is *False*, each job's eligible time evaluates to zero in the formula.
- If a job is requeued, and you are using the formula, the job may lose its place, because various factors may affect the job's priority. For example, a higher-priority job may be submitted between the time the job is requeued and the time it would have run, or another job's priority may be increased due to changes in which jobs are running or waiting.
- If the formula is configured, it is in force during both primetime and non-primetime.
- If an error is encountered while evaluating the formula, the formula evaluates to zero for that job, and the following message is logged at event class 0x0100:

  ```
  "1234.mars;Formula evaluation for job had an error.  Zero value will be used"
  ```
- You may have to work around a Python bug by multiplying by 1.0, in order to prevent rounding to the nearest integer.

4.9.21.14 Logging

For each job, the evaluated formula answer is logged at the highest log level (0x0400):

```
"Formula  Evaluation = <answer>"
```

4.9.22 Gating Jobs at Server or Queue

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than a scheduler's partition or a queue can supply.

You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using the resources in the job sorting formula or to route jobs to particular vnodes.

You can either force users to submit their jobs to specific queues, or you can have users submit jobs to routing queues, and then route the jobs to the desired queues.

For information on using resources for gating, see section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 254.

For a description of which resources can be used for gating, see section 2.3.6.4.iii, “Resources Used for Routing and Admittance”, on page 27.

For how queue resource limits are applied to jobs, see section 2.3.6.4.i, “How Queue and Server Limits Are Applied, Except Running Time”, on page 26.

For how routing queues work, see section 2.3.6, “Routing Queues”, on page 25.

For how to route jobs to particular vnodes, see section 4.9.2, “Associating Vnodes with Queues”, on page 103.

For how to use resources in the job sorting formula, see section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

4.9.22.1 Gating Caveats

- For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. See section 5.9.3.6, “Using Gating Values As Defaults”, on page 247.
- For shrink-to-fit jobs, if a walltime limit is specified:
  - Both min_walltime and max_walltime must be greater than or equal to resources_min.walltime.
  - Both min_walltime and max_walltime must be less than or equal to resources_max.walltime.

4.9.23 Managing Application Licenses

PBS does not check application licenses out from the license server. PBS has no direct control over application licenses. However, you can have a scheduler use a dynamic resource to track application license use. This way, a scheduler knows how many application licenses are available, and how many have been checked out. For how to configure dynamic resources to represent application licenses, see section 5.14.6, “Supplying Application Licenses”, on page 274.

Unfortunately, some jobs or applications don't check out all of the application licenses they use until they have been running for some time. For example, job J1, which requests licenses, starts running, but doesn't check them out for a few minutes. Next, the scheduler considers job J2, which also requests licenses. The scheduler runs its query for the number of available licenses, and the query returns with a sufficient number of licenses to run J2, so the scheduler starts J2. Shortly afterward, J1 checks out licenses, leaving too few to run J2.

It might appear that you could track the number of application licenses being used with a static integer PBS resource, and force jobs requesting application licenses to request this resource as well, but there is a drawback: if a job that has requested this resource is suspended, its static resources are released, but its application licenses are not. In this case you could end up with a deceptively high number for available licenses.

You can limit the number of jobs that request application licenses, if you know how many jobs can run at one time:

- Create a custom server-level consumable integer resource to represent these jobs. See section 5.14.3, “Creating Server-level Custom Resources”, on page 267.
- Use `qmgr` to set resources_available.<job limit> at the server to the number of jobs that can run at one time.
- Force all jobs requesting the application to request one of these. See section 14.3, “Allocating Resources to Jobs”, on page 505.

4.9.24 Limits on Per-job Resource Usage

You can specify how much of each resource any job is allowed to request, at the server and queue level. The server and queues each have per-job limit attributes. The resources_min.<resource name> and resources_max.<resource name> server and queue attributes are limits on what each individual job may request.

You cannot set resources_min or resources_max limits on min_walltime or max_walltime.

See section 5.15.2, “Placing Resource Limits on Jobs”, on page 304, and section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 254.

4.9.25 Limits on Project, User, and Group Jobs

You can manage the number of jobs being run by users or groups, and the number of jobs being run in projects, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to *30*, and by any single user to *5*.

See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

4.9.26 Limits on Project, User, and Group Resource Usage

You can manage the total amount of each resource that is used by projects, users, or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.

See section 5.15.1, “Managing Resource Usage By Users, Groups, and Projects, at Server & Queues”, on page 287.

4.9.27 Using Load Balancing

As of version 2020.1, the load_balancing scheduler parameter is **deprecated**. We recommend sorting vnodes according to load average, described in section 4.9.50.3, “Sorting Vnodes According to Load Average”, on page 227.

PBS can use the load_balancing scheduler parameter to track the load on each execution host, running new jobs on the host according to the load on the host. You can specify that PBS does this for all machines in a scheduler's partition. This is somewhat different behavior from that used for managing the load on vnodes; when managing load levels on vnodes, a scheduler only pays attention to the state of the vnode, and does not calculate whether a job would put the vnode over its load limit. Managing load levels on vnodes does not require load balancing to be turned on. See section 9.6.5, “Managing Load Levels on Vnodes”, on page 445. This load balancing tool does not work across vnodes on a multi-vnoded host.

You use the load_balancing scheduler parameter to control whether PBS tracks the load on each host.

The load_balancing parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.9.27.1 How Load Average is Computed

When load balancing is on, a scheduler queries each MoM once each scheduling cycle for the MoM's load. MoM checks the load average on her host every 10 seconds. The load used by MoM is the raw one-minute averaged "loadave" returned by the operating system.

When a new load is added to a vnode, the load average increases slowly over time, so that more jobs than you want may be started at first. Eventually, the load average matches the actual load. If this is above the limit, PBS won't start any more jobs on that vnode. As jobs terminate, the load average slowly moves down, and it takes time before the vnode is chosen for new jobs.

Consult your OS documentation to determine load values that make sense.

MoM sets the load only on the parent vnode, so it is the same for all vnodes on a multi-vnode machine.

4.9.27.2 How PBS Uses Load Information

When choosing vnodes for a job, a scheduler considers the load on the vnode in addition to whether the vnode can supply the resources specified in the job's Resource_List attribute.

PBS estimates that a 1-CPU job will produce one CPU's worth of load. This means that if you have a 2-CPU machine whose load is zero, PBS will put two 1-CPU jobs, or one 2-CPU job, on that machine.

When using load balancing, if a vnode has gone above $max_load, PBS does not run new jobs on the vnode until the load drops below $ideal_load.

MoM sets the vnode's state according to its load. When a vnode's load goes above $max_load, MoM marks the vnode *busy*. When the load drops below $ideal_load, MoM marks the vnode *free*. When a vnode's state changes, for example from *free* to *busy*, MoM informs the server.

When using load balancing, PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked *busy*
- Vnodes whose resources, such as ncpus, are already fully allocated
- Vnodes that are above $max_load
- Vnodes where running the job would cause the load to go above $max_load

4.9.27.3 When to Use Load Balancing

When using load balancing (meaning the load_balancing scheduler parameter is *True*), the only changes to behavior are the following:

- A scheduler won't place a job on a vnode whose load is above $max_load
- A scheduler won't place a job on a vnode where that job would put the load above $max_load

Load balancing is useful when you want to oversubscribe CPUs, managing job placement by load instead. This can help when you want to run lots of jobs where each job will need only some CPU time, and the average load on the machine will be reasonable.

4.9.27.4 Suspending Jobs on Overloaded Vnodes

You can specify that MoM should suspend jobs when the load goes above $max_load, by adding the suspend argument to the $max_load parameter. See section , “$max_load <load> [suspend]”, on page 159. In this case, MoM suspends all jobs on the vnode until the load drops below $ideal_load, then resumes them. This option is useful only when the source of the load is not strictly PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between becoming overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, becoming overloaded, and so on.

4.9.27.5 Configuring Load Balancing

If you want to oversubscribe CPUs, set the value of ncpus on the vnode to the desired higher value.

We recommend setting the value of $max_load to a slightly higher value than the desired load, for example *.25 + ncpus*. Otherwise, a scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.

To configure load balancing, perform the following steps:

1. Turn on load balancing by setting the load_balancing scheduler parameter to *True*:

   ```
   load_balancing: True ALL
   ```

2. Choose whether you want load balancing during primetime, non-primetime, or all. If you want separate behavior for primetime and non-primetime, specify each separately. The default is both. Example of separate behavior:

   ```
   load_balancing True prime
   load_balancing False non_prime
   ```

3. Set the ideal and maximum desired load for each execution host, by specifying values for $ideal_load and $max_load in each execution host's MoM configuration file:

   ```
   $ideal_load <value at which to start new jobs>
   $max_load   <value at which to cease starting jobs>
   ```

4. Set each host's resources_available.ncpus to the maximum number of CPUs you wish to allocate on that host. Follow the recommendations in section 3.4.5, “Configuring Vnode Resources”, on page 47.

4.9.27.6 Load Balancing Caveats and Recommendations

- When setting ncpus and $max_load, consider the ratio between the two. PBS won't allocate more than the value of resources_available.ncpus, so you can use this value to keep the load average from getting too high.
- Make sure that load balancing does not interfere with communications. Please read section 9.6.5, “Managing Load Levels on Vnodes”, on page 445.
- Load balancing is incompatible with sorting vnodes on a key (node_sort_key) when sorting on a resource using the "*unused*" or "*assigned*" parameters. Load balancing will be disabled. See section 4.9.50, “Sorting Vnodes on a Key”, on page 226.
- You can use load balancing with SMP cluster distribution, but smp_cluster_dist will behave as if it is set to *pack*. See section 4.9.43, “SMP Cluster Distribution”, on page 218.
- We recommend setting the value of $max_load to a slightly higher value than the desired load, for example *.25 + ncpus*. Otherwise, a scheduler will not schedule jobs onto the last CPU, because it thinks a 1-CPU job will raise the load by 1, and the machine probably registers a load above zero.
- If you are using cycle harvesting via load balancing, make sure your load balancing settings do not interfere with cycle harvesting. Be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working. Please read section 4.9.9.6, “Cycle Harvesting Based on Load Average”, on page 121.
- Using load balancing with multi-vnoded machines is not supported. MoM sets the load average only on the parent vnode, so all vnodes on a multi-vnoded machine are given the same value regardless of their actual load.
- It is not recommended to specify that MoM should suspend jobs when the load goes above $max_load. See section 4.9.27.4, “Suspending Jobs on Overloaded Vnodes”, on page 157.
- If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

4.9.27.7 Parameters Affecting Load Balancing

$ideal_load <load>

MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load parameter.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

$max_load <load> [suspend]

MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load parameter.

If the optional suspend argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load, regardless of the source of the load (PBS and/or logged-in users).

Example:

```
$max_load 3.5
```

Format: *Float*

Default: *number of CPUs*

load_balancing <T|F> [time slot specification]

Scheduler parameter. When set to *True*, this scheduler takes into account the load average on vnodes as well as the resources listed in the `resources:` line in `sched_config`. See "load_balancing" on page 253 of the PBS Professional Reference Guide.

Format: *Boolean*

Default: *False all*

4.9.28 Matching Jobs to Resources

A scheduler places each job where the resources requested by the job are available. A scheduler handles built-in and custom resources the same way. For a complete description of PBS resources, see Chapter 5, "Using PBS Resources", on page 229.

4.9.28.1 Scheduling on Consumable Resources

A scheduler constrains the use of a resource to the value that is set for that resource in resources_available.<resource name>. For a consumable resource, a scheduler won't place more demand on the resource than is available. For example, if a vnode has resources_available.ncpus set to *4*, a scheduler will place jobs requesting up to a total of 4 CPUs on that vnode, but no more.

A scheduler computes how much of a resource is available by subtracting the total of resources_assigned.<resource name> for all running jobs and started reservations from resources_available.<resource name>.

4.9.28.2 Scheduling on Non-Consumable Resources

For non-consumable resources such as arch or host, a scheduler matches the value requested by a job with the value at one or more vnodes. Matching a job this way does not change whether or not other jobs can be matched as well; non-consumable resources are not used up by jobs, and therefore have no limits.

4.9.28.3 Scheduling on Dynamic Resources

At each scheduling cycle, a scheduler queries each dynamic resource. If a dynamic resource is not under the control of PBS, jobs requesting it may run in an unpredictable fashion.

4.9.28.4 Scheduling on the walltime Resource

A scheduler looks at each job in priority order, and tries to run the job. A scheduler checks whether there is an open time slot on the requested resources that is at least as long as the job's walltime. If there is, the scheduler runs the job.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's min_walltime and max_walltime. If the job can fit somewhere, PBS sets the job's walltime to a duration that fits the time slot, and runs the job. For more information about shrink-to-fit jobs, see section 4.9.42, "Using Shrink-to-fit Jobs", on page 211.

4.9.28.4.i Caveats for Scheduling on walltime

Do not set values for resources such as walltime at the server or a queue, because a scheduler will not allocate more than the specified value. This means that if you set resources_available.walltime at the server to *10:00:00*, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

4.9.28.5 Unrequestable or Invisible Resources

You can define custom resources that are invisible to and unrequestable by users, or simply unrequestable by users. A scheduler treats these resources the same as visible, requestable resources. See section 5.14.2.3.vi, "Resource Permission Flags", on page 260.

4.9.28.6 Enforcing Scheduling on Resources

A scheduler chooses which resources to schedule on according to the following rules:

- A scheduler always schedules jobs based on the availability of the following vnode-level resources:

  vnode

  host

  Any Boolean resource

- A scheduler will schedule jobs based on the availability of other resources only if those resources are listed in the "`resources:`" line in `<sched_priv directory>/sched_config`. Some resources are automatically added to this line. You can add resources to this line. The following resources are automatically added to this line:

  aoe

  arch

  eoe

  host

  mem

  ncpus

  vnode

4.9.28.7 Matching Unset Resources

When job resource requests are being matched with available resources, unset resources are treated as follows:

- A numerical resource that is unset on a host is treated as if it were *zero*
- An unset resource on the server or queue is treated as if it were infinite
- An unset string cannot be matched
- An unset Boolean resource is treated as if it were set to *False*.
- The resources ompthreads, mpiprocs, and nodes are ignored for unset resource matching.

The following table shows how a resource request will or won't match an unset resource at the host level.

Table 4-15: Matching Requests to Unset Host-level Resources

Resource Type	Unset Resource	Matching Request Value
Boolean	*False*	*False*
float	*0.0*	*0.0*
long	*0*	*0*
size	*0*	*0*
string	*""*	Never matches
string array	*""*	Never matches
time	*0, 0:0, 0:0.0, 0:0:0*	*0, 0:0, 0:0.0, 0:0:0*

4.9.28.7.i When Dynamic Resource Script Fails

If a server dynamic resource script fails, a scheduler uses the value of resources_available.<resource name>. If this was never set, it is treated as an unset resource, described above.

If a host-level dynamic resource script fails, a scheduler treats the resource as if its value is zero.

4.9.28.7.ii Backward Compatibility of Unset Resources

To preserve backward compatibility, you can set the server's resource_unset_infinite attribute with a list of host-level resources that will behave as if they are infinite when they are unset. See "resource_unset_infinite" on page 256 of the PBS Professional Reference Guide for information on resource_unset_infinite.

4.9.28.8 Resource Scheduling Caveats

- Do not set values for resources such as walltime at the server or a queue, because a scheduler will not allocate more than the specified value. This means that if you set resources_available.walltime at the server to *10:00:00*, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
- Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job's resource request will no longer match the same resources on the server, queues and vnodes.
- Beware of application license race conditions. If two jobs require the same application license, the first job may be started, but may not get around to using the license before the second job is started and uses the license. The first job must then wait until the license is available, taking up resources. A scheduler cannot avoid this problem.

4.9.29 Node Grouping

The term "*node grouping*" has been superseded by the term "*placement sets*". Vnodes were originally grouped according to the value of one resource, so for example all vnodes with a value of *linux* for arch were grouped together, and all vnodes with a value of *arch1* for arch were in a separate group. We use placement sets now because this means grouping vnodes according to the value of one *or more* resources. See section 4.9.32, "Placement Sets", on page 168.

4.9.29.1 Configuring Old-style Node Grouping

Configuring old-style node grouping means that you configure the simplest possible placement sets. In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will be in only one node group.

You enable node grouping by setting the server's node_group_enable attribute to *True*.

You can configure one set of vnode groups for the entire complex by setting the server's node_group_key attribute to a resource name.

You can configure node grouping separately for each queue by setting that queue's node_group_key attribute to a resource name.

4.9.30 Overrides

You can use various overrides to change how one or more jobs run.

4.9.30.1 Run a Job Manually

You can tell PBS to run a job now, and you can optionally specify where to run it. You run a job manually using the `qrun` command.

The `-H` option to the `qrun` command makes an important difference:

`qrun`

When preemption is enabled, a scheduler preempts other jobs in order to run this job. Running a job via `qrun` gives the job higher preemption priority than any other class of job, except for reservation jobs.

When preemption is not enabled, a scheduler runs the job only if enough resources are available.

`qrun -H`

PBS runs the job regardless of scheduling policy and available resources.

The `qrun` command alone overrides the following:

- Limits on resource usage by users, groups, and projects
- Limits on the number of jobs that can be run at a vnode
- Boundaries between primetime and non-primetime, specified in backfill_prime
- Whether the job is in a primetime queue: you can run a job in a primetime queue even when it's not primetime, or vice versa. Primetime boundaries are not honored.
- Dedicated time: you can run a job in a dedicated time queue, even if it's not in a dedicated time queue, and vice versa. However, dedicated time boundaries are still honored.
- Top jobs
- The threshold set in the job_sort_formula_threshold scheduler attribute
- The limit on the number of simultaneously running subjobs for an array job set in the max_run_subjobs job attribute

The qrun command alone does not override the following:

- Server and queue resource usage limits

4.9.30.1.i Using qrun Without -H Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via qrun, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job's max_walltime but not its min_walltime, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.
- If there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.9.30.1.ii Using qrun With -H Option on Shrink-to-fit Jobs

When a shrink-to-fit job is run via qrun -H, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via qrun -H, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for walltime, that value is used, instead of one set by PBS when the job runs. If no walltime is specified, the job runs without a walltime.

See "qrun" on page 182 of the PBS Professional Reference Guide, and section 4.9.33, "Using Preemption", on page 180.

4.9.30.1.iii qrun Caveats

- A job that has just been run via qrun has top priority only during the scheduling cycle where it was qrun. At the next scheduling cycle, that job is available for preemption just like any other job.
- Be careful when using qrun -H on jobs or vnodes involved in reservations.

4.9.30.2 Hold a Job Manually

You can use the qhold command to place a hold on a job. The effect of placing a hold depends on whether the job is running and whether you have checkpointing configured:

- If the job is queued, the job will not run.
- If the job is running and checkpoint-abort is configured, the job is checkpointed, requeued, and held.
- If the job is running and checkpoint-abort is not configured, the only change is that the job's Hold_Types attribute is set to User Hold. If the job is subsequently requeued, it will not run until the hold is released.

You can release the hold using the qrls command.

For information on checkpointing jobs, see section 9.3, "Checkpoint and Restart", on page 418.

See "qhold" on page 147 of the PBS Professional Reference Guide and "qrls" on page 180 of the PBS Professional Reference Guide.

4.9.30.3 Suspend a Job Manually

You can use the qsig -s suspend command to suspend a job so that it won't run. If you suspend a job, and then release it using the qsig -s resume command, the job remains in the suspended state until the required resources are available.

You can resume the job immediately by doing the following:

1. Resume the job:

   ```
   qsig -s resume <job ID>
   ```

2. Run the job manually:

   ```
   qrun <job ID>
   ```

See "qsig" on page 192 of the PBS Professional Reference Guide.

4.9.30.4 Set Special Resource Value Used in Formula

You can change the value of a resource used in the job sorting formula. For example, to give a particular job a higher priority by changing the value of a custom resource called "higher":

- Create a custom resource that is invisible to job submitters:
  Qmgr: create resource higher type=float, flag=i
- The formula expression includes "higher":
  Qmgr: s s job_sort_formula = "higher"
- Set the default for this resource at the server:
  Qmgr: set server resources_default.higher = 1
- These jobs are submitted:
  Job 1
  Job 2
  Job 3
- Change Job 2 so that its value for "higher" is 5:
  qalter -l higher = 5 job2
- The scheduler logs the following:
  Job;1.host1;Formula Evaluation = 1
  Job;2.host1;Formula Evaluation = 5
  Job;3.host1;Formula Evaluation = 1
- Jobs are sorted in this order:
  Job 2
  Job 1
  Job 3

4.9.30.5 Change Formula On the Fly

You can change the job sorting formula on the fly, so that the next scheduler iteration uses your new formula. This will change how job priorities are computed, and can rearrange the order in which jobs are run. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.

4.9.30.6 Using Dedicated Time

You can set up blocks of dedicated time, where the only jobs eligible to be started or running are the ones in dedicated time queues. You can use dedicated time for upgrades. See section 4.9.10, "Dedicated Time", on page 125, and section 2.3.5.2.i, "Dedicated Time Queues", on page 24.

4.9.30.7 Using `cron` Jobs

You can use `cron` jobs to change PBS settings according to the needs of your time slots. See section 4.9.7, "cron Jobs", on page 112.

4.9.30.8 Using Hooks

You can use hooks to examine jobs and alter their characteristics. See the *PBS Professional Hooks Guide*.

4.9.30.9 Preventing Jobs from Being Calendared

You can prevent a scheduler from calendaring a job by setting its topjob_ineligible attribute to *True*. See section 4.9.17, "Calendaring Jobs", on page 137.

4.9.31 Peer Scheduling

Peer scheduling allows separate PBS partitions or complexes to automatically run jobs from each other's queues. This means that you can dynamically balance the workload across multiple, separate PBS partitions or complexes. These cooperating PBS partitions or complexes are referred to as "*Peers*".

4.9.31.1 How Peer Scheduling Works

In peer scheduling, a PBS server pulls jobs from one or more peer servers and runs them locally. When Partition or Complex A pulls a job from Partition or Complex B, Partition or Complex A is the "pulling" complex and Partition or Complex B is the "furnishing" partition or complex. When the pulling scheduler determines that another partition's or complex's job can immediately run locally, it moves the job to the specified queue on the pulling server and immediately run the job. The job is run as if it had been submitted to the pulling partition or complex.

You can set up peer scheduling so that A pulls from B and C, and so that B also pulls from A and C.

A job is pulled **only** when it can run immediately.

The pulling partition or complex must have all of the resources required by the job, including custom resources.

When a job is pulled from one partition or complex to another, the pulling partition or complex applies its policy to the job. The job's execution priority is determined by the policy of the pulling partition or complex. You can set special priority for pulled jobs; see section 4.9.31.4.ii, "Setting Priority for Pulled Jobs", on page 167.

4.9.31.2 Prerequisites for Peer Scheduling

- You must create the pulling and furnishing queues before peer scheduling can be configured. See section 2.3.3, "Creating Queues", on page 22 on how to create queues.
- When configuring peer scheduling, it is *strongly* recommended to use the same version of PBS Professional at all peer locations.
- Make sure that custom resources are consistent across peer locations. Jobs requesting custom resources at one location will not be able to run at another unless the same resources are available.

4.9.31.3 Configuring Peer Scheduling

The following sections give details on how to configure peer scheduling. Here is a brief outline:

- Define a flat user namespace on all complexes
- Map pulling queues to furnishing queues
- If necessary, specify port
- Grant manager access to each pulling server
- If possible, make user-to-group mappings be consistent across complexes
- If any of the peering sites is using failover, configure peering to work with failover

4.9.31.3.i Defining a Flat User Namespace

Peer scheduling requires a flat user namespace in all complexes involved. This means that user "joe" on the remote peer system(s) must be the same as user "joe" on the local system. Your site must have the same mapping of user to UID across all hosts, and a one-to-one mapping of UIDs to usernames. It means that PBS does not need to check whether X@hostA is the same as X@hostB; it can just assume that this is true. Set flatuid to *True*:

```
Qmgr: set server flatuid = True
```

For more on flatuid, see section 8.3.12, “Flatuid and Access”, on page 376.

4.9.31.3.ii Mapping Pulling Queues to Furnishing Queues

You configure peer scheduling by mapping a furnishing peer's queue to a pulling peer's queue. You can map each pulling queue to more than one furnishing queue, or more than one pulling queue to each furnishing queue.

The pulling and furnishing queues must be *execution* queues, not route queues. However, the queues can be either ordinary queues used for normal work, or special queues set up just for peer scheduling.

You map pulling queues to furnishing queues by setting the peer_queue scheduler configuration option in `<sched_priv directory>/sched_config`. The format is:

```
peer_queue: "<pulling queue> <furnishing queue>@<furnishing server>.domain"
```

For example, Complex A's queue "workq" is to pull from two queues: Complex B's queue "workq" and Complex C's queue "slowq". Complex B's server is ServerB and Complex C's server is ServerC. You would add this to Complex A's `<sched_priv directory>/sched_config`:

```
peer_queue: "workq workq@ServerB.domain.com"
peer_queue: "workq slowq@ServerC.domain.com"
```

Or if you wish to direct Complex B's jobs to queue Q1 on Complex A, and Complex C's jobs to Q2 on Complex A:

```
peer_queue: "Q1 workq@ServerB.domain.com"
peer_queue: "Q2 fastq@ServerC.domain.com"
```

In one partition or complex, you can create up to 50 mappings between queues. This means that you can have up to 50 lines in `<sched_priv directory>/sched_config` beginning with "`peer_queue`".

4.9.31.3.iii Specifying Ports

The default port for the server to listen on is 15001, and a scheduler uses any privileged port (1023 and lower). If the furnishing server is not using the default port, you must specify the port when you specify the queue. For example, if ServerB is using port 16001, and you wish to pull jobs from workq at ServerB to workq at ServerA, add this to `<sched_priv directory>/sched_config` at ServerA:

```
peer_queue: "workq workq@ServerB.domain.com:16001"
```

A scheduler and server communicate via TCP.

4.9.31.3.iv Granting Manager Access to Pulling Servers

Each furnishing server must grant manager access to each pulling server. If you wish jobs to move in both directions, where Complex A will both pull from and furnish jobs to Complex B, ServerA and ServerB must grant manager access to each other.

On the furnishing complex:

```
Qmgr: set server managers += root@pullingServer.domain.com
```

4.9.31.3.v Making User-to-group Mappings Consistent Across Complexes

If possible, ensure that for each user in a peer complex, that user is in the same group in all participating complexes. So if user "joe" is in groupX on Complex A, user "joe" should be in groupX on Complex B. This means that a job's `egroup` attribute will be the same on both complexes, and any group limit enforcement can be properly applied.

There is a condition when using peer scheduling in which group hard limits may not be applied correctly. This can occur when a job's effective group, which is its egroup attribute, i.e. the job's owner's group, is different on the furnishing and pulling systems. When the job is moved over to the pulling complex, it can evade group limit enforcement if the group under which it will run on the pulling system has not reached its hard limit. The reverse is also true; if the group under which it will run on the pulling system has already reached its hard limit, the job won't be pulled to run, although it should.

This situation can also occur if the user explicitly specifies a group via `qsub -W group_list`.

It is recommended to advise users to *not* use the `qsub` options "`-u user_list`" or "`-W group_list=groups`" in conjunction with peer scheduling.

4.9.31.3.vi Configuring Peer Scheduling with Failover

If you are configuring peer scheduling so that Complex A will pull from Complex B where Complex B is configured for failover, you must configure Complex A to pull from both of Complex B's servers. For these instructions, see section 9.2.6.2, “Configuring Failover to Work With Peer Scheduling”, on page 415.

4.9.31.4 Peer Scheduling Advice

4.9.31.4.i Selective Peer Scheduling

You can choose the kinds of jobs that can be selected for peer scheduling to a different partition or complex. You can do the following:

- Set resource limits at the furnishing queue via the resources_min and resources_max queue attributes. See section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 26.
- Route jobs into the furnishing queue via a hook. See "Routing Jobs" on page 7 in the PBS Professional Hooks Guide.
- Route jobs into the furnishing queue via a routing queue. See section 2.3.6, “Routing Queues”, on page 25.

4.9.31.4.ii Setting Priority for Pulled Jobs

You can set a special priority for pulled jobs by creating a queue that is used only as a pulling queue, and setting the pulling queue's priority to the desired level. You can then use the queue's priority when setting job execution priority. See section 4.3.5.3.iv, “Using Queue Priority when Computing Job Priority”, on page 65.

For example, if you give the pulling queue the lowest priority, the pulling partition or complex will pull a job only when there are no higher-priority jobs that can run.

You can also have pulled jobs land in a special queue where they inherit a custom resource that is used in the job sorting formula.

4.9.31.5 How Peer Scheduling Affects Jobs

4.9.31.5.i How Peer Scheduling Affects Inherited Resources

If the job is moved partition or complex to another via peer scheduling, any default resources in the job's resource list inherited from the furnishing queue or server are removed. This includes any select specification and place directive that may have been generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing partition or complex. If the job is requeued and restarted at the pulling partition or complex, the job picks up new default resources from the pulling partition or complex, and is scheduled according to the newly-inherited resources from the pulling partition or complex.

4.9.31.5.ii How Peer Scheduling Affects Policy Applied to Job

After a job is pulled from one partition or complex to another, the scheduling policy of the pulling partition or complex is applied to the job.

For example, if you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the queue to which the job is moved.

When a job is pulled from one partition or complex to another, hooks are applied at the new partition or complex as if the job had been submitted locally. For example, if the pulling partition or complex has a *queuejob* hook, that hook runs when a job is pulled.

4.9.31.5.iii How Peer Scheduling Affects Job Eligible Time

The job's eligible_time is preserved when a job is moved due to peer scheduling.

4.9.31.5.iv Viewing Jobs That Have Been Moved to Another Server

If you are connected to ServerA and a job submitted to ServerA has been moved from ServerA to ServerB through peer scheduling, in order to display it via `qstat`, give the job ID as an argument to `qstat`. If you only give the `qstat` command, the job will not appear to exist. For example, the job `123.ServerA` is moved to ServerB. In this case, use

```
qstat 123
```

or

```
qstat 123.ServerA
```

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

4.9.31.5.v Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any movejob hooks at the furnishing server are run

4.9.31.6 Peer Scheduling Caveats

- Each partition or complex can peer with at most 50 other partitions or complexes.
- When using peer scheduling, group hard limits may not be applied correctly. This can occur when the job owner's group is different on the furnishing and pulling systems. For help in avoiding this problem, see section 4.9.31.3.v, "Making User-to-group Mappings Consistent Across Complexes", on page 166.
- When the pulling scheduler runs the job the first time, the job is run as if the job still had all of the resources it had at the furnishing partition or complex. If the job is requeued and restarted at the pulling partition or complex, the job picks up new default resources from the pulling partition or complex, and is scheduled according to the newly-inherited resources from the pulling partition or complex.
- Peer scheduling is not supported for job arrays.

4.9.32 Placement Sets

Placement sets are the sets of vnodes within which PBS will try to place a job. PBS tries to group vnodes into the most useful sets, according to how well connected the vnodes are, or the values of resources available at the vnodes. Placement sets are used to improve task placement (optimizing to provide a "good fit") by exposing information on system configuration and topology. A scheduler tries to put a job in the smallest appropriate placement set.

4.9.32.1 Definitions

Task placement

The process of choosing a set of vnodes to allocate to a job that will satisfy both the job's resource request (select and place specifications) and the configured scheduling policy.

Placement Set

A set of vnodes. Placement sets are defined by the values of vnode-level string array resources. A placement set is all of the vnodes that have the same value for a specified defining resource substring. For example, if the defining resource is a vnode-level string array named "*switch*", which can have values "S1", "S2", or "S3": the set of vnodes which have a substring matching "switch=S2" is a placement set.

Placement sets can be specified at the server or queue level.

Placement Set Series

A set of placement sets; a set of sets of vnodes.

A placement set series is all of the placement sets that are defined by specifying one string array resource. Each placement set in the series is the set of vnodes that share one value for the resource. There is one placement set for each value of the resource. If the resource takes on N values at the vnodes, then there are N sets in the series. For example, if the defining resource is a string array named "*switch*", which can have values "S1", "S2", or "S3", there are three sets in the series. The first is defined by the value "S1", where all the vnodes in that set have the value "S1" for the resource switch. The second set is defined by "S2", and the third by "S3".

Each of the resources named in node_group_key specifies a placement series. For example, if the server's node_group_key attribute contains "router,switch", then the server has two placement set series.

Placement Pool

All of the placement sets that are defined; the server can have a placement pool, and each queue can have its own placement pool. If a queue has no placement pool, a scheduler uses the server's placement pool.

A placement pool is the set of placement set series that are defined by one or more string array resources named in node_group_key.

For example, if the server's node_group_key attribute contains "router,switch", and router can take the values "R1" and "R2" and switch can take the values "S1", "S2", and "S3", then there are five placement sets, in two placement series, in the server's placement pool.

Static Fit

A job statically fits into a placement set if the job could fit into the placement set if the set were empty. It might not fit right now with the currently available resources.

Dynamic Fit

A job dynamically fits into a placement set if it will fit with the currently available resources (i.e. the job can fit right now).

4.9.32.2 Requirements for Placement Sets

- Placement sets are enabled by setting the server's node_group_enable attribute to *True*
- Server-level placement sets are defined by setting the server's node_group_key attribute to a list of vnode-level string array resources.
- Queue-level placement sets are defined by setting a queue's node_group_key attribute to a list of vnode-level string array resources.
- At least one vnode-level string array resource must exist on vnodes and be set to values that can be used to assign the vnodes to placement sets.

4.9.32.3 Description of Placement Sets

4.9.32.3.i What Defines a Placement Set, Series, or Pool

Placement sets are defined by the values of vnode-level string array resources. You define placement sets by specifying the names of these resources in the node_group_key attribute for the server and/or queues. Each value of each resource defines a different placement set. A placement set is all of the vnodes that have the same value for a specified defining resource. For example, if the defining resource is a vnode-level string array named "*switch*", which has the values "S1", "S2", and "S3", the set of vnodes where switch has the value "S2" is a placement set. If some vnodes have more than one substring, and one of those substrings is the same in each vnode, those vnodes make up a placement set. For example, if the resource is "*router*", and vnode V0 has resources_available.router set to "r1i0,r1", and vnode V1 has resources_available.router set to "r1i1,r1", V0 and V1 are in the placement set defined by resources_available.router = "r1". If the resource has *N* distinct values across the vnodes, including the value zero and being unset, there can be *N-1* or *N* placement sets defined by that resource. If the only_explicit_psets scheduler attribute is *False*, there are *N* placement sets. If the only_explicit_psets scheduler attribute is *True*, there are *N-1* placement sets; see section 4.9.32.3.v, "Placement Sets Defined by Unset Resources", on page 171.

Each placement set can have a different number of vnodes; the number of vnodes is determined only by how many vnodes share that resource value.

Each placement set series is defined by the values of a single resource across all the vnodes. For example, if there are three switches, S1, S2 and S3, and there are vnodes with resources_available.switch that take on one or more of these three values, then there will be three placement sets in the series.

Whenever you define any placement sets, you are defining a placement pool. Placement pools can be defined for the server and for each queue. You define a server-level placement pool by setting the server's node_group_key to a list of one or more vnode-level string array resources. You define a queue-level placement pool by similarly setting the queue's node_group_key.

4.9.32.3.ii Vnode Participation in Placement Sets, Series, and Pools

Each vnode can be in multiple placement sets, placement set series, and placement pools.

A vnode can be in multiple placement sets in the same placement set series. For example, if the resource is called "*router*", and a vnode's router resource is set to "R1, R2", then the vnode will be in the placement set defined by router = R1 and the set defined by router = R2.

A vnode is in a placement series whenever the resource that defines the series is defined on the vnode. For example, if placement sets are defined by the values of the router and the switch resources, and a vnode has value *R1* for router, and *S1* for switch, then the vnode is in both placement series, because it is in the set that shares the *R1* value for router, and the set that shares the *S1* value for switch. Each of those sets is one of a different series.

The server has its own placement pool if the server's node_group_key attribute is set to at least one vnode-level string array resource. Similarly, each queue can have its own placement pool. A vnode can be in any placement pool that specifies a resource that is defined on the vnode.

4.9.32.3.iii Multihost Placement Sets

Placement sets, series, and pools can span hosts. Placement sets can be made up of vnodes from anywhere, regardless of whether the vnode is from a multi-vnode host.

To set up a multihost placement set, choose a string array resource for the purpose, and list it in the desired node_group_key attribute. For example, create a string_array resource called "span":

```
Qmgr: create resource span type=string_array, flag=h
```

Add the resource "span" to node_group_key on the server or queue. Use `qmgr` to give it the same value on all the desired vnodes. You can write a script that sets the same value on each vnode that you want in your placement set.

4.9.32.3.iv Machines with Multiple Vnodes

Machines with multiple vnodes are represented as a generic set of vnodes. Placement sets are used to allocate resources on a single machine to improve performance and satisfy scheduling policy and other constraints. Jobs are placed on vnodes using placement set information.

4.9.32.3.v Placement Sets Defined by Unset Resources

The only_explicit_psets scheduler attribute controls whether unset resources define placement sets.

- If the only_explicit_psets scheduler attribute is *False*, vnodes where a defining resource is unset are grouped into their own placement set, for each defining resource. For example, if you have ten vnodes, on which there is a string resource COLOR, where two have COLOR set to "red", two are set to "blue", two are set to "green" and the rest are unset, there will be four placement sets defined by the resource COLOR. This is because the fourth placement set consists of the four vnodes where COLOR is unset. This placement set will also be the largest. Every resource listed in node_group_key can potentially define such a placement set.
- If the only_explicit_psets scheduler attribute is *True*, vnodes where a resource is unset are not grouped into placement sets.

4.9.32.3.vi Placement Sets and Node Grouping

Node grouping is the same as one placement set series, where the placement sets are defined by a single resource. Node grouping has been superseded by placement sets.

In order to have the same behavior as in the old node grouping, group on a single resource. If this resource is a string array, it should only have one value on each vnode. This way, each vnode will only be in one node group.

4.9.32.4 How Placement Sets Are Used

You use placement sets to group vnodes according to the value of one or more resources. Placement sets allow you to group vnodes into useful sets.

You can run multi-vnode jobs in one placement set. For example, it makes the most sense to run a multi-vnode job on vnodes that are all connected to the same high-speed switch.

PBS will attempt to place each job in the smallest possible set that is appropriate for the job.

4.9.32.4.i Order of Placement Pool Selection

A scheduler chooses one placement pool from which to select a placement set.

Queue placement pools override the server's placement pool. If a queue has a placement pool, jobs from that queue are placed using the queue's placement pool. If a queue has no placement pool (the queue's node_group_key is not defined), jobs are placed using the server's placement pool, if it exists.

A per-job placement set is defined by the `-l place` statement in the job's resource request. Since the job can only request one value for the resource, it can only request one specific placement set. A job's `place=group` resource request overrides the sets defined by the queue's or server's node_group_key.

A scheduler chooses the most specific placement pool available, following this order of precedence:

1. A per-job placement set (job's `place=group=` request)
2. A placement set from the placement pool for the job's queue
3. A placement set from the placement pool in a scheduler's partition

4.9.32.4.ii Order of Placement Set Consideration Within Pool

A scheduler looks in the selected placement pool and chooses the smallest possible placement set that is appropriate for the job. A scheduler examines the placement sets in the pool and orders them, from smallest to largest, according to the following rules:

1. Static total ncpus of all vnodes in set
2. Static total mem of all vnodes in set
3. Dynamic free ncpus of all vnodes in set
4. Dynamic free mem of all vnodes in set

If a job can fit statically within any of the placement sets in the placement pool, then a scheduler places a job in the first placement set in which it fits dynamically. This ordering ensures a scheduler will use the smallest possible placement set in which the job will dynamically fit. If there are multiple placement sets where the job fits statically, but some are being used, a scheduler uses the first placement set where the job can run now. If the job fits statically into at least one placement set, but these placement sets are all busy, a scheduler waits until a placement set can fit the job dynamically.

For example, we have the following placement sets, and a job that requests 8 CPUs:

Set1 ncpus = 4

Set2 ncpus = 12; this placement set is full

Set3 ncpus = 16; this placement set is not being used

The scheduler looks at Set1; Set1 is statically too small, and the scheduler moves to the next placement set. Set2 is statically large enough, but the job does not fit dynamically. The scheduler looks at Set3; Set3 is large enough, and the job fits dynamically. The scheduler runs the job in Set3.

If the job requests 24 CPUs, the scheduler attempts to run the job in the set consisting of all vnodes that are associated with a specific queue, if do_not_span_psets is *False*.

4.9.32.4.iii Determining Whether Job Can Run

Whether the job can run in the selected placement pool is determined by the value of the do_not_span_psets attribute.

- If this attribute is *False*, and a job cannot statically fit into any placement set in the selected placement pool, a scheduler ignores defined placement sets and uses all vnodes that satisfy job restrictions as its placement set, and runs the job without regard to placement sets. For example, if the job's queue has access to a restricted set of vnodes, the job runs within that set of vnodes.
- If the attribute is *True*, a scheduler does not run the job.

4.9.32.4.iv Order of Vnode Selection Within Set

A scheduler orders the vnodes within the selected placement set using the following rules:

- If node_sort_key is set, vnodes are sorted by node_sort_key. See section 4.9.50, "Sorting Vnodes on a Key", on page 226.
- If node_sort_key is not set, the order in which the vnodes are returned by `pbs_statnode()`. This is the default order the vnodes appear in the output of the `pbsnodes -a` command.

A scheduler places the job on the vnodes according to their ordering above.

4.9.32.5 Summary of Placement Set Requirements

The steps to configure placement sets are given in the next section. The requirements are summarized here for convenience:

- Definitions of the resources of interest
- Vnodes defining a value for each resource to be used for placement sets (e.g., rack)
  - If defined via vnode definition, you must HUP the MoMs involved
- The server's or queue's node_group_key attribute must be set to the resources to be used for placement sets. For example, if we have custom resources named "rack", "socket", "board", and "boardpair", which are to be used for placement sets:

  Qmgr: set server node_group_key = "rack,socket,board,boardpair"
  - No signals needed, takes effect immediately
- Placement sets must be enabled at the server by setting the server's node_group_enable attribute to *True*. For example:

  Qmgr: set server node_group_enable=True
  - No signals needed, takes effect immediately

Adding a resource to a scheduler's `resources`: line is required only if the resource is to be specifically requested by jobs. It is not required for `-lplace=group=<resource name>`.

4.9.32.6 How to Configure Placement Sets

The following steps show how to satisfy the requirements for placement sets:

1. If the vnodes that you will use in placement sets are not defined, define them. See section 3.3, “Creating Vnodes”, on page 38.
2. If the vnode-level string array resources that you will use to define placement sets do not exist, create them. See section 5.14.4, “Configuring Host-level Custom Resources”, on page 269.
3. If values for the vnode-level string array resources that you will use to define placement sets are not set at the vnodes you wish to use, set the values. See section 3.4, “Configuring Vnodes”, on page 41.
4. If you use vnode definition files to set values for vnode-level string array resources, HUP the MoMs involved.
5. To create queue placement pools, set the node_group_key attribute to the name(s) of one or more vnode-level string array resources. Do this for each queue for which you want a separate pool. For example:

   Qmgr: set queue workq node_group_key = <router,switch>
6. To create a server placement pool, set the node_group_key server attribute to the name(s) of one or more vnode-level string array resources. For example:

   Qmgr: set server node_group_key = <router,switch>

For example, to create a server-level placement pool for the resources host, L2 and L3:

```
Qmgr: set server node_group_key = "host,L2,L3"
```

7. Set the server's node_group_enable attribute to *True*

   ```
   Qmgr: set server node_group_enable = True
   ```

8. Set the do_not_span_psets scheduler attribute to *True* if you don't want jobs to span placement sets.

   ```
   Qmgr: set sched do_not_span_psets = True
   ```

9. Set the only_explicit_psets attribute to *True* if you don't want a scheduler to create placement sets from unset resources.

   ```
   Qmgr: set sched only_explicit_psets = True
   ```

10. For ease of reviewing placement set information, you can add the name of each resource used in each vnode's pnames attribute:

    ```
    Qmgr: active node <vnode name>,<vnode name>,...
    Qmgr: set node pnames += <resource name>
    ```

    or

    ```
    Qmgr: set node pnames = <resource list>
    ```

    For example:

    ```
    Qmgr: set node pnames = "board,boardpair,iruquadrant,iruhalf,iru,rack"
    ```

We recommend using the parent vnode for any placement set information that is invariant for a given host.

Resources used only for defining placement sets, and not for allocation to jobs, do not need to be listed in the `resources:` line in `<sched_priv directory>/sched_config`. So for example if you create a resource just for defining placement sets, and jobs will not be requesting this resource, you do not need to list it in the `resources:` line.

4.9.32.7 Examples of Creating Placement Sets

4.9.32.7.i Cluster with Four Switches

This cluster is arranged as shown with vnodes 1-4 on Switch1, vnodes 5-10 on Switch2, and vnodes 11-24 on Switch3. Switch1 and Switch2 are on Switch4.

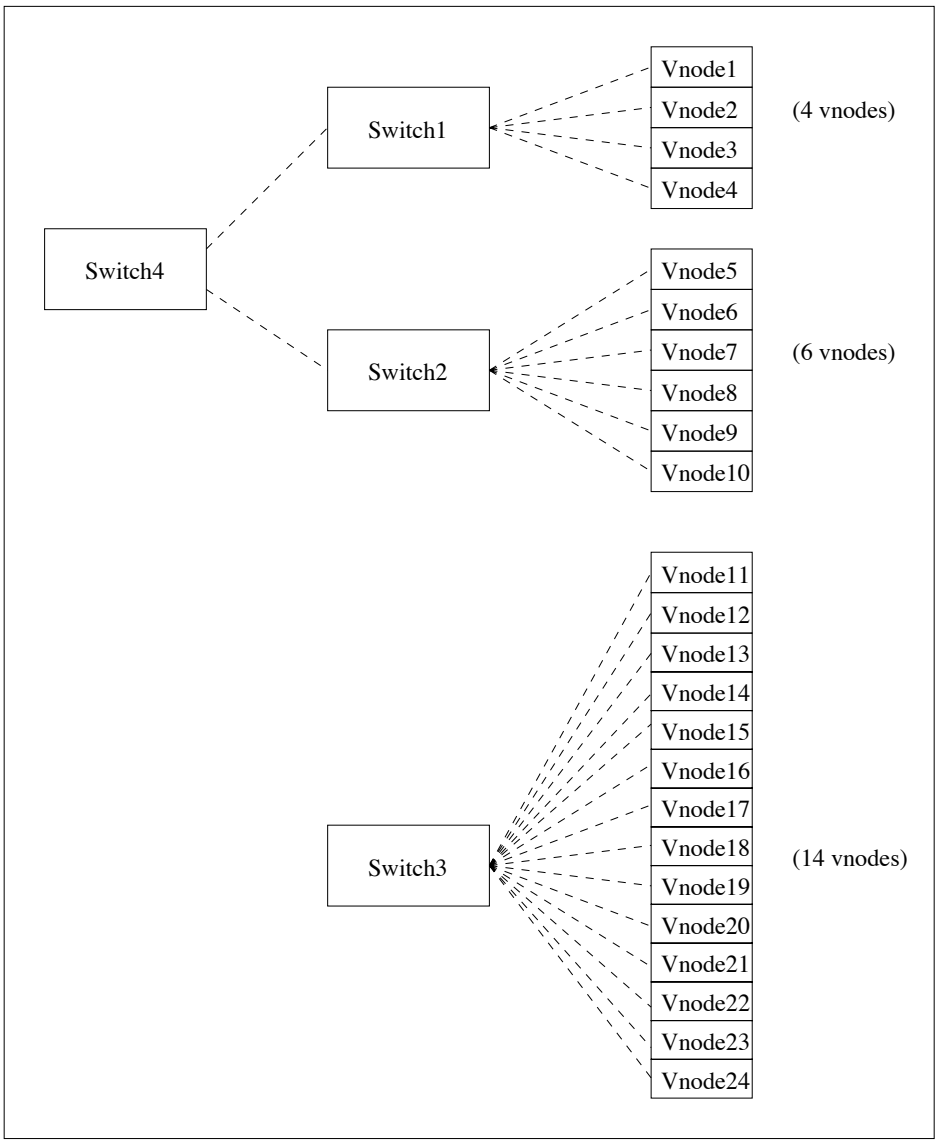


Figure 4-1:Cluster with Four Switches

To make the placement sets group the vnodes as they are grouped on the switches:

Create a custom resource called *switch*. The -h flag makes the resource requestable:

```
Qmgr: create resource switch  type=string_array, flag=h
```

On vnodes[1-4] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch1,switch4"
```

On vnodes[5-10] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch2,switch4"
```

On vnodes[11-24] set:

```
Qmgr: set node <vnode name> resources_available.switch="switch3"
```

On the server set:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

So you have 4 placement sets:

The placement set "switch1" has 4 vnodes

The placement set "switch2" has 6 vnodes

The placement set "switch3" has 14 vnodes

The placement set "switch4" has 10 vnodes

PBS will try to place a job in the smallest available placement set. Does the job fit into the smallest set (switch1)? If not, does it fit into the next smallest set (switch2)? This continues until it finds one where the job will fit.

4.9.32.7.ii Example of Configuring Placement Sets on a Multi-vnode Machine

For information on how to configure vnodes via Version 2 configuration files, see section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.

In this example, we define a new placement set using the new resource "NewRes". We create a file called `SetDefs` that contains the changes we want.

1. Create the new resource:
   ```
   Qmgr: create resource NewRes type=string_array, flag=h
   ```
2. Add NewRes to the server's node_group_key
   ```
   Qmgr: set server node_group_key+="NewRes"
   ```
3. Add NewRes to the value of the pnames attribute for the parent vnode. This makes the name of the resource you used easily available. Add a line like this to `SetDefs`:
   ```
   host3:  resources_available.pnames =...,NewRes
   ```
4. For each vnode, V, that's a member of a new placement set you're defining, add a line of the form:
   ```
   V: resources_available.NewRes = <value1[,...]>
   ```
   All the vnodes in the new set should have lines of that form, with the same resource value, in the new configuration file.

   Here the value of the resource is "P" and/or "Q".

   We'll put vnodes A, B and C into one placement set, and vnodes B, C and D into another.
   ```
   A:  resources_available.NewRes2 = P
   B:  resources_available.NewRes2 = P,Q
   C:  resources_available.NewRes2 = P,Q
   D:  resources_available.NewRes2 = Q
   ```
   For each new placement set you define, use a different value for the resource.
5. Add `SetDefs` and tell MoM to read it, to make a Version 2 vnode configuration file `NewConfig`:
   ```
   pbs_mom -s insert NewConfig SetDefs
   ```
6. Stop and restart the MoM. For Linux, see "Restarting and Reinitializing MoM" on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "Restarting MoMs" on page 173 in the PBS Professional Installation & Upgrade Guide.

4.9.32.7.iii Example of Placement Sets Using Colors

A placement pool is defined by two resources: `colorset1` and `colorset2,` by using "node_group_key=`colorset1,colorset2`".

If a vnode has the following values set:

```
resources_available.colorset1=blue, red
resources_available.colorset2=green
```

The placement pool contains at least three placement sets. These are:

{resources_available.colorset1=blue}

{resources_available.colorset1=red}

{resources_available.colorset2=green}

This means the vnode is in all three placement sets. The same result would be given by using one resource and setting it to all three values, e.g. `colorset=blue,red,green`.

Example: We have five vnodes `v1 - v5:`

```
v1 color=red host=mars
v2 color=red host=mars
v3 color=red host=venus
v4 color=blue host=mars
v5 color=blue host=mars
```

The placement sets are defined by

```
node_group_key=color
```

The resulting node groups would be: `{v1, v2, v3}, {v4, v5}`

4.9.32.7.iv Simple Switch Placement Set Example

Say you have a cluster with two high-performance switches each with half the vnodes connected to it. Now you want to set up placement sets so that jobs will be scheduled only onto the same switch.

First, create a new resource called "*switch*". See section 5.14.2, "Defining New Custom Resources", on page 257.

Next, we need to enable placement sets and specify the resource to use:

```
Qmgr: set server node_group_enable=True
Qmgr: set server node_group_key=switch
```

Now, set the value for switch on each vnode:

```
Qmgr: active node vnode1,vnode2,vnode3
Qmgr: set node resources_available.switch=A
Qmgr: active node vnode4,vnode5,vnode6
Qmgr: set node resources_available.switch=B
```

Now there are two placement sets:

switch=*A*: {vnode1, vnode2, vnode3}

switch=*B*: {vnode4, vnode5, vnode6}

4.9.32.8 Placement Sets and Reservations

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.9.32.4.ii, "Order of Placement Set Consideration Within Pool", on page 172.

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored. The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

4.9.32.9 Placement Sets and Load Balancing

If you configure both placement sets and load balancing, the net effect is that vnodes that are over their load limit will be removed from consideration.

4.9.32.10 Viewing Placement Set Information

You can find information about placement sets in the following places:

- The server's node_group_enable attribute shows whether placement sets are enabled
- The server's node_group_key attribute contains the names of resources used for that queue's placement pool
- Each queue's node_group_key attribute contains the names of resources used for that queue's placement pool
- Each vnode's pnames attribute can contain the names of resources used for placement sets, if properly set
- A scheduler's do_not_span_psets attribute shows whether jobs are allowed to span placement sets
- A scheduler's only_explicit_psets attribute shows placement sets are created using unset resources
- PBS-generated MoM configuration files contain names and values of resources

4.9.32.11 Placement Set Caveats and Advice

- If there is a vnode-level platform-specific resource set on the vnodes on a multi-vnode machine, then node_group_key should probably include this resource, because this will enable PBS to run jobs in more logical sets of vnodes.
- If the user specifies a job-specific placement set, for example `-lplace=group=switch`, but the job cannot statically fit into any switch placement set, then the job will still run, but not in a switch placement set.
- The pnames vnode attribute is for displaying to the administrator the resources used for placement sets. This attribute is not used by PBS.

4.9.32.11.i Non-backward-compatible Change in Node Grouping

Given the following example configuration:

vnode1: switch=A

vnode2: switch=A

vnode3: switch=B

vnode4: switch=B

vnode5: switch unset

```
Qmgr: s s node_group_key=switch
```

There is no change in the behavior of jobs submitted with `qsub -l ncpus=1`

version 7.1: The job can run on any node: node1, ..., node5

version 8.0: The job can run on any node: node1, ..., node5

Example of 8.0 and later behavior: jobs submitted with `qsub -lnodes=1`

version 7.1: The job can only run on nodes: node1, node2, node3, node4. It will never use node5

version 8.0: The job can run on any node: node1, ..., node5

Overall, the change for version 8.0 was to include every vnode in placement sets (when enabled). In particular, if a resource is used in node_group_key, PBS will treat every vnode as having a value for that resource, hence every vnode will appear in at least one placement set for every resource. For vnodes where a string resource is "unset", PBS will behave as if the value is "".

4.9.32.12 Attributes and Parameters Affecting Placement Sets

do_not_span_psets

Scheduler attribute. Specifies whether or not this scheduler requires the job to fit within one of the existing placement sets. When do_not_span_psets is set to *True*, a scheduler will require the job to fit within a single existing placement set. A scheduler checks all placement sets, whether or not they are currently in use. If the job fits in a currently-used placement set, the job must wait for the placement set to be available. If the job cannot fit within a single placement set, it will not run.

When this attribute is set to *False*, a scheduler first attempts to place the job in a single placement set. All existing placement sets are checked. If the job fits in an occupied placement set, the job waits for the placement set to be available. If there is no existing placement set, occupied or empty, into which the job could fit, the job runs regardless of placement sets, running on whichever vnodes can satisfy the job's resource request.

Format: *Boolean*

Default value: *False* (This matches behavior of PBS 10.4 and earlier)

Example: To require jobs to fit within one placement set:

```
Qmgr: set sched do_not_span_psets=True
```

node_group_enable

Server attribute. Specifies whether placement sets are enabled.

Format: *Boolean*

Default: *False*

node_group_key

Server and queues have this attribute. Specifies resources to use for placement set definition. Queue's attribute overrides server's attribute.

Format: *string_array*

Default: Unset

only_explicit_psets

Scheduler attribute. Specifies whether placement sets are created using unset resources. If *False*, for each defining resource, if there are vnodes where the value of the resource is unset, PBS creates a placement set for the series defined by that resource. If *True*, PBS does not create placement sets for resources that are unset.

Format: *Boolean*

Default: *False*

4.9.32.13 Errors and Logging

If do_not_span_psets is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  `"Not Running: can't fit in the largest placement set, and can't span psets"`
- The following message is printed to the scheduler's log:
  `"Can't fit in the largest placement set, and can't span placement sets"`

4.9.33 Using Preemption

PBS provides the ability to preempt currently running jobs in order to run higher-priority work. This is called *preemption* or *preemptive scheduling*. PBS has two different approaches to specifying preemption:

- You can define a set of preemption priorities for all jobs. Jobs that have high preemption priority preempt those with low preemption priority. Preemption priority is mostly independent of execution priority. See section 4.9.33.7, "Preemption Levels", on page 184.
- You can specify a set of preemption targets for each job. You can also set defaults for these targets at the server and queues. Preemption targets are jobs in specific queues or that have requested specific resources. See section 4.9.33.4, "Using Preemption Targets", on page 182.

Preemption is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.9.33.1 Glossary

Preempt

Stop one or more running jobs in order to start a higher-priority job

Preemption level

Job characteristic that determines preemption priority. Levels can be things like being in an express queue, starving, having an owner who is over a soft limit, being a normal job, or having an owner who is over a fair-share allotment

Preemption method

The method by which a job is preempted. This can be checkpointing, suspension, requeueing, or deletion

Preemption priority

How important this job is compared to other jobs, when considering whether to preempt

Preemption Target

A preemption target is a job in a specified queue or a job that has requested a specified resource. The queue and/or resource is specified in another job's Resource_List.preempt_targets.

4.9.33.2 Preemption Parameters and Attributes

The scheduler parameters that control preemption are defined in `<sched_priv directory>/sched_config`. A scheduler also has attributes that control preemption; they can be set via `qmgr`. Parameters and attributes that control preemption are listed here:

preemptive_sched

Parameter. Enables job preemption.

Format: *String*

Default: *True all*

preempt_order

Attribute. Defines the order of preemption methods which this scheduler will use on jobs. Can contain any of *S*, *C*, *R*, and *D*, in any order.

Format: *String*, as quoted list

Default: "*SCR*"

preempt_prio

Attribute. Specifies the ordering of priority of different preemption levels.

Format: *String*, as quoted list

Default: "*express_queue, normal_jobs*"

preempt_queue_prio

Attribute. Specifies the minimum queue priority required for a queue to be classified as an express queue.

Format: *Integer*

Default: *150*

preempt_sort

Attribute. Whether jobs most eligible for preemption will be sorted according to their start times. Allowable values: "*min_time_since_start*". The first job preempted will be that with most recent start time.

Format: *String*

Default: *min_time_since_start*

preempt_targets

Resource that a job can request or inherit from the server or a queue. The preempt_targets resource lists one or more queues and/or one or more resources. Jobs in those queues, and jobs that request those resources, are the jobs that can be preempted.

restrict_res_to_release_on_suspend

Server attribute. Comma-separated list of consumable resources to be released when jobs are suspended. If unset, all consumable resources are released on suspension. See section 5.9.6.2, "Job Suspension and Resource Usage", on page 250 and "Server Attributes" on page 281 of the PBS Professional Reference Guide.

Format: *string_array*

Default: *unset*

Python type: *list*

resources_released

Job attribute. Listed by vnode, consumable resources that were released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. See section 5.9.6.2, "Job Suspension and Resource Usage", on page 250 and "Job Attributes" on page 328 of the PBS Professional Reference Guide.

Format: String of the form: *(<vnode>:<resource name>=<value>:<resource name>=<value>:...)+(<vnode>:<resource name>=<value>:...)*

Python type: *str*

resource_released_list

Job attribute. Sum of each consumable resource requested by the job that was released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. See section 5.9.6.2, "Job Suspension and Resource Usage", on page 250 and "Job Attributes" on page 328 of the PBS Professional Reference Guide.

Format: String of the form: *resource_released_list.<resource name>=<value>,resource_released_list.<resource name>=<value>, ...*

sched_preempt_enforce_resumption

Scheduler attribute. Specifies whether this scheduler creates a special execution priority class for preempted jobs. If so, this scheduler runs these jobs just after any higher-priority jobs. See section 4.9.16, “Calculating Job Execution Priority”, on page 134.

Format: *Boolean*

Default: *False*

Python type: *pbs.pbs_resource*

4.9.33.3 How Preemption Works

If preemption is enabled, a scheduler uses preemption as part of its normal pattern of examining each job and figuring out whether or not it can run now. If a job with high preemption priority cannot run immediately, a scheduler looks for jobs with lower preemption priority. A scheduler finds jobs in the lowest preemption level that have been started the most recently. A scheduler preempts these jobs and uses their resources for the higher-priority job. A scheduler tracks resources used by lower-priority jobs, looking for enough resources to run the higher-priority job. If a scheduler cannot find enough work to preempt in order to run a given job, it will not preempt any work.

A job running in a reservation cannot be preempted.

A job's preemption priority is determined by its preemption level.

4.9.33.4 Using Preemption Targets

You can restrict the set of jobs that can be preempted by an entity, by setting that entity's preempt_targets resource to a list of jobs and/or queues that can be preempted. This resource is a string array which can contain a list of queues and/or job resources. You specify job resources as *Resource_List.<resource>=<value>*.

Syntax:

preempt_targets="Queue=<queue name>[,Queue=<queue name>],Resource_List.<resource name>=<value>[,Resource_List.<resource name>=<value>]"

or

preempt_targets=None

The preempt_targets resource has the following keywords:

Queue=<queue name>

Jobs in the specified queue are eligible to be preempted. "Queue" is case-insensitive.

None

The job, or the jobs at the queue or server whose preempt_targets resource is set to *NONE* cannot preempt other jobs. "None" is case-insensitive.

In order for a job to preempt another job, the job to be preempted must have lower preemption priority than the preempting job.

4.9.33.4.i Setting Job Preemption Targets

Preemption targets work as a restriction on which jobs can be preempted by a particular job. If a job has requested preempt_targets, a scheduler searches for lower-priority jobs among only the jobs specified in that job's preempt_targets. If a job has not requested preempt_targets, the scheduler searches among all jobs. For example, if a scheduler is trying to run JobA, and JobA requests `preempt_targets="queue=Queue1,Resource_List.arch=linux"`, JobA is eligible to preempt only those jobs in Queue1 and/or that request `arch=linux`. In addition, JobA can only preempt jobs with lower preemption priority than JobA.

You can prevent a job from preempting any other job in the complex by setting its preemption_targets to the keyword "None" (case-insensitive).

You can set preempt_targets for a job during submission:

```
-l preempt_targets=...
```

You can set preempt_targets via `qalter`:

```
qalter -l preempt_targets=...
```

4.9.33.4.ii Setting Queue Preemption Targets

You can set the default preemption target for jobs in a queue. For example, you can specify that the jobs in a particular queue can preempt the jobs in one or more listed queues:

qmgr -c 'set queue <queue name> resources_default.preempt_targets="QUEUE=<queue name>,QUEUE=<queue name>"'

For example:

```
qmgr -c 'set queue high_prio_queue resources_default.preempt_targets="QUEUE=queueA,QUEUE=queueB"'
```

You can prevent the jobs in a queue which don't explicitly request preempt_targets from preempting other jobs by setting the queue's default preempt_targets to "NONE":

qmgr -c "set queue <queue name> resources_default.preempt_targets=NONE"

For example:

```
qmgr -c "set queue lowest_prio_queue resources_default.preempt_targets=NONE"
```

4.9.33.4.iii Setting Default Server Preemption Targets

You can set the default preemption target for jobs at a server. For example, you can specify that the jobs at a server can preempt the jobs in one or more listed queues:

qmgr -c 'set server resources_default.preempt_targets="QUEUE=<queue name>,QUEUE=<queue name>"'

For example:

```
qmgr -c 'set server resources_default.preempt_targets="QUEUE=queueA,QUEUE=queueB"'
```

You can prevent the jobs which don't explicitly request preempt_targets from preempting other jobs by setting the server's default preempt_targets to "NONE":

qmgr -c "set server resources_default.preempt_targets=NONE"

For example:

```
qmgr -c "set server resources_default.preempt_targets=NONE"
```

4.9.33.5 Preemption and Job Execution Priority

PBS has an execution class we call *Preempted* for jobs that have been preempted. A scheduler restarts preempted jobs as soon as the preemptor finishes and any other higher-priority jobs finish. See section 4.9.16, “Calculating Job Execution Priority”, on page 134.

4.9.33.6 Triggers for Preemption

If preemption is enabled, preemption is used during the following:

- The normal scheduling cycle
- When you run a job via `qrun`

4.9.33.7 Preemption Levels

A preemption level is a class of jobs, where all the jobs in the class share a characteristic. PBS provides built-in preemption levels, and you can combine them or ignore them as you need, except for the *normal_jobs* class, which is required. The built-in preemption levels are listed in the table below.

Table 4-16: Built-in Preemption Levels

Preemption Level	Description
express_queue	Jobs in express queues. See section 4.9.33.7.ii, "The Express Queues Preemption Level", on page 186
starving_jobs	A job that is starving. See section 4.9.33.7.iv, "The Starving Job Preemption Level", on page 186
normal_jobs	The preemption level into which a job falls if it does not fit into any other specified level. See section 4.9.33.7.v, "The Normal Jobs Preemption Level", on page 187
fairshare	When the entity owning a job exceeds its fairshare limit. See section 4.9.33.7.iii, "The Fairshare Preemption Level", on page 186
queue_softlimits	Jobs which are over their queue soft limits. See section 4.9.33.7.i, "The Soft Limits Preemption Level", on page 185
server_softlimits	Jobs which are over their server soft limits. See section 4.9.33.7.i, "The Soft Limits Preemption Level", on page 185

You can specify the relative priority of each preemption level, by listing the levels in the desired order in the preempt_prio scheduler attribute. Placing a level earlier in the list, meaning to the left, gives it higher priority. For example, if your list is `"express_queue", "normal_jobs", "server_softlimits"`, you are giving the highest priority to jobs in express queues, and the lowest priority to jobs that are over their server soft limits. You can list levels in any order, but be careful not to work at cross-purposes with your execution priority. See section 4.9.16, "Calculating Job Execution Priority", on page 134.

The default value for preempt_prio is the following:

```
preempt_prio: "express_queue, normal_jobs"
```

If you do not list a preemption level in the preempt_prio scheduler attribute, the jobs in that level are treated like normal jobs. For example, if you do not list server_softlimits, then jobs that are over their server soft limits are treated like jobs in the normal_jobs level.

You can create new levels that are combinations of the built-in levels. For example, you can define a level which is *"express_queue + server_softlimits"*. This level contains jobs that are in express queues and are over their server soft limits. You would probably want to place this level just to the right of the express_queue level, meaning that these jobs could be preempted by jobs that are in express queues but are not over their server soft limits.

You can give two or more levels the same priority. To do this, put a plus sign ("+") between them, and do not list either level separately in preempt_prio. You are creating a new level that includes all the built-in levels that should have the same priority. For example, to list express queue jobs as highest in priority, then fairshare and starving jobs at the next highest priority, then normal jobs last, create a new level that contains the fairshare and starving_jobs levels:

```
preempt_prio: "express_queue, fairshare+starving_jobs, normal_jobs"
```

You can be specific about dividing up jobs: if you want jobs in the express queue to preempt jobs that are also in the express queue but are over their server soft limits, list each level separately:

```
preempt_prio: "express_queue, express_queue+server_softlimits, normal_jobs"
```

However, be careful not to create a runaway effect by placing levels that are over limits before those that are not, for example, `express_queue+server_softlimits` to the left of `express_queue`.

You must list normal_jobs in the preempt_prio scheduler attribute.

4.9.33.7.i The Soft Limits Preemption Level

You can set a limit, called a *hard limit*, on the number of jobs that can be run or the amount of a resource that can be consumed by a person, a group, or by everyone, and this limit can be applied at the server and at each queue. If you set such a limit, that is the greatest number of jobs that will be run, or the largest amount of the resource that will be consumed.

You can also set a *soft limit* on the number of jobs that can be run or the amount of a resource that can be consumed. This soft limit should be lower than the hard limit, and should mark the point where usage changes from being normal to being "extra, but acceptable". Usage in this "extra, but acceptable" range can be treated by PBS as being lower priority than the normal usage. PBS can preempt jobs that are over their soft limits. The difference between the soft limit and the hard limit provides a way for users or groups to use resources as long as no higher-priority work is waiting.

Example 4-18: Using group soft limits

One group of users, group A, has submitted enough jobs that the group is over their soft limit. A second group, group B, submits a job and are under their soft limit. If preemption is enabled, jobs from group A are preempted until the job from group B can run.

Example 4-19: Using soft limits on number of running jobs

Given the following:

- You have three users, UserA, UserB, and UserC
- Each has a soft limit of 3 running jobs
- UserA runs 3 jobs
- UserB runs 4 jobs
- UserC submits a job to an express queue

This means:

- UserB has 1 job over the soft limit, so UserB's jobs are eligible for preemption by UserC's job

Example 4-20: Using soft limits on amount of resource being used

Given the following:

- Queue soft limit for ncpus is 8
- UserA's jobs use 6 CPUs
- UserB's jobs use 10 CPUs

This means:

- UserB is over their soft limit for CPU usage
- UserB's jobs are eligible for preemption

To use soft limits in preemption levels, you must define soft limits. Soft limits are specified by setting server and queue limit attributes. The attributes that control soft limits are:

max_run_soft
: Sets the soft limit on the number of jobs that can be running

max_run_res_soft.<resource name>
: Sets the soft limit on the amount of a resource that can be consumed by running jobs

Soft limits are enforced only when they are used as a preemption level.

To use soft limits as preemption levels, add their keywords to the preempt_prio attribute:

- To create a preemption level for those over their soft limits at the server level, add "server_softlimits" to the preempt_prio attribute.
- To create a preemption level for those over their soft limits at the queue level, add "queue_softlimits" to the preempt_prio attribute.
- To create a preemption level for those over their soft limits at both the queue and server, add "server_softlimits+queue_softlimits" to the preempt_prio attribute.

The jobs of a user or group are over their soft limit only as long as the number of running jobs or the amount of resources used by running jobs is over the soft limit. If some of these jobs are preempted or finish running, and the soft limit is no longer exceeded, the jobs of that user or group are no longer over their soft limit, and no longer in that preemption level. For example, if the soft limit is 3 running jobs, and UserA runs 4 jobs, as soon as one job is preempted and only 3 of UserA's jobs are running, UserA's jobs are no longer over their soft limit.

For a complete description of the use of these attributes, see section 5.15.1.4, “Hard and Soft Limits”, on page 290.

4.9.33.7.ii The Express Queues Preemption Level

The express_queue preemption level applies to jobs residing in express queues. An express queue is an execution queue with priority at or above the value set in the preempt_queue_prio scheduler attribute. The default value for this parameter is *150*.

Express queues do not require the by_queue scheduler parameter to be *True*.

If you will use the express_queue preemption level, you probably want to configure at least one express queue, along with some method of moving jobs into it. See section 2.3, “Queues”, on page 21.

If you have more than one express queue, and they have different priorities, you are effectively creating separate sub-levels for express queues. Jobs in a higher-priority express queue have greater preemption priority than jobs in lower-priority express queues.

See “preempt_queue_prio” on page 254 of the PBS Professional Reference Guide.

4.9.33.7.iii The Fairshare Preemption Level

The fairshare preemption level applies to jobs owned by entities who are over their fairshare allotment. For example, if each of five users has 20 percent of the fairshare tree, and UserA is using 25 percent of the resources being tracked for fairshare, UserA's jobs become eligible for preemption at the fairshare preemption level.

To use the fairshare preemption level, you must enable fairshare. See section 4.9.19, “Using Fairshare”, on page 138.

4.9.33.7.iv The Starving Job Preemption Level

The starving_jobs preemption level applies to jobs that are starving. Starving jobs are jobs that have been waiting at least a specified amount of time to run.

To use the starving_jobs preemption level, you must enable starving:

- Set the <sched_priv directory>/sched_config help_starving_jobs parameter to *True*
- Set the amount of time that a job must wait before it is starving in the max_starve scheduler parameter
- Optionally, use eligible time for waiting time. See section 4.9.13, “Eligible Wait Time for Jobs”, on page 126.

See section 4.9.48, “Starving Jobs”, on page 223.

4.9.33.7.v The Normal Jobs Preemption Level

One special class, normal_jobs, is the default class for any job not otherwise specified. If a job does not fall into any of the specified levels, it is placed in normal_jobs.

Example 4-21: Starving jobs have the highest priority, then normal jobs, then jobs whose entities are over their fairshare limit:

```
preempt_prio: "starving_jobs, normal_jobs, fairshare"
```

Example 4-22: Starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs:

```
preempt_prio: "normal_jobs, starving_jobs+fairshare"
```

4.9.33.8 Selecting Preemption Level

PBS places each job in the most exact preemption level, or the highest preemption level that fits the job.

Example 4-23: We have a job that is starving and over its server soft limits. The job is placed in the "starving_jobs" level:

```
preempt_prio: "starving_jobs, normal_jobs, server_softlimits"
```

Example 4-24: We have a job that is starving and over its server soft limits. The job is placed in the "starving_jobs+server_softlimits" level:

```
preempt_prio: "starving_jobs, starving_jobs+server_softlimits, normal_jobs, server_softlimits"
```

4.9.33.9 Sorting Within Preemption Level

If there is more than one job within the preemption level chosen for preemption, PBS chooses jobs within that level according to their start time. By default, PBS preempts the job which started running most recently. .

For example, if we have two jobs where job A started running at 10:00 a.m. and job B started running at 10:30 a.m:

- The default behavior preempts job B

The allowable value for the preempt_sort attribute is "*min_time_since_start*".

The default value for the preempt_sort attribute is "*min_time_since_start*".

4.9.33.10 Preemption Methods

A scheduler can preempt a job in one of the following ways:

- Suspend the job
- Checkpoint the job
- Requeue the job
- Delete the job

A scheduler tries to preempt a job using the methods listed in the order you specify. This means that if you specify that the order is "checkpoint, suspend, requeue, delete", the scheduler first tries to checkpoint the job, and if it cannot, it tries to suspend the job, and if it cannot do that, it tries to requeue the job, and if it cannot requeue the job, it tries to delete it.

You can specify the order of these attempts in the preempt_order scheduler attribute.

The preempt_order attribute defines the order of preemption methods which a scheduler uses on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of *S, C*, *R,* and *D* (for suspend, checkpoint, requeue, and delete).

The contents is an ordering, for example "SCRD" optionally followed by a percentage of time remaining and another ordering.

The format is a quoted list("").

Example 4-25: PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, then requeue the job:

```
preempt_order: "SR"
```

Example 4-26: If the job has between 100-81% of requested time remaining, first try to suspend the job, then try checkpoint, then requeue. If the job has between 80-51% of requested time remaining, then attempt suspend then checkpoint; and between 50% and 0% time remaining just attempt to suspend the job:

```
preempt_order: "SCR 80 SC 50 S"
```

The default value for preempt_order is "*SCR*".

You cannot repeat a method within a percentage specification. Note that in the example above, the *S* method appears only once per percentage.

4.9.33.10.i Preemption Via Checkpoint

When a job is preempted via checkpointing, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. When a scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed.

To preempt via checkpointing, you must define both of the following:

- The checkpointing action in the MoM's checkpoint_abort $action parameter that is to take place when the job is preempted
- The restarting action in the MoM's restart $action parameter that is to take place when the job is restarted

To do this, you must supply checkpointing and restarting scripts or equivalents, and then configure the MoM's checkpoint_abort and restart $action parameters. Do not use the $action checkpoint MoM parameter; it is used when the job should keep running.

See section 9.3, "Checkpoint and Restart", on page 418.

4.9.33.10.ii Preemption Via Suspension

Jobs are normally suspended via the SIGSTOP signal and resumed via the SIGCONT signal. An alternate suspend or resume signal can be configured in MoM's $suspendsig configuration parameter. See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

4.9.33.10.iii Suspended Jobs and Resources

Suspended jobs will hold onto some memory and disk space. Suspended jobs may hold application licenses if the application releases them only when it exits. See section 5.9.6.2.iv, "Suspension/resumption Resource Caveats", on page 251.

4.9.33.10.iv Preemption Via Requeue

When a job is preempted and requeued, the job stops execution and is requeued. A requeued job's eligible time is preserved. The amount of time allowed to requeue a job is controlled by the job_requeue_timeout server attribute. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

A job that is not eligible to be requeued, meaning a job that was submitted with "`-r n`", will not be selected to be preempted via requeue.

4.9.33.10.v Preemption via Deletion

When a job is preempted via deletion, the job is deleted. It is not requeued. Deletion is not in the default preemption order.

4.9.33.11 Enabling Preemption

Preemptive scheduling is enabled by setting parameters in a scheduler's configuration file `<sched_priv directory>/sched_config`.

To enable preemption, you must do the following:

1. Specify the preemption levels to be used by setting preempt_prio to desired preemption levels (the default is `"express_queue, normal_jobs"`)

   The preempt_prio attribute must contain an entry for normal_jobs.

2. Optional: specify preemption order by setting preempt_order
3. If you will use the fairshare preemption level, configure fairshare. See section 4.9.19, “Using Fairshare”, on page 138.
4. If you will use the starving_jobs preemption level, configure starving. See section 4.9.33.7.iv, “The Starving Job Preemption Level”, on page 186.
5. If you will use the server_softlimits and/or queue_softlimits preemption levels, configure server and/or queue soft limits. See section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 185.
6. Enable preemption by setting preemptive_sched to *True* . It is *True* by default.
7. Choose whether to use preemption during primetime, non-primetime, or all of the time. The default is ALL. If you want separate behavior for primetime and non-primetime, specify each separately. For example:

   ```
   preemptive_sched: True prime
   preemptive_sched: False non_prime
   ```

4.9.33.12 Preemption Example

Below is an example of (part of) a scheduler's configuration file, showing an example configuration for preemptive scheduling.

```
# turn on preemptive scheduling
#
preemptive_sched:       TRUE ALL
#
# set the queue priority level for express queues
#
preempt_queue_prio:     150
#
# specify the priority of jobs as: express queue
# (highest) then starving jobs, then normal jobs,
# followed by jobs who are starving but the user/group
# is over a soft limit, followed by users/groups over
# their soft limit but not starving
#
preempt_prio: "express_queue, starving_jobs, normal_jobs, starving_jobs+server_softlimits,
    server_softlimits"
#
# specify when to use each preemption method.
# If the first method fails, try the next
# method. If a job has between 100-81% time
# remaining, try to suspend, then checkpoint
# then requeue. From 80-51% suspend and then
# checkpoint, but don't requeue.
# If between 50-0% time remaining, then just
# suspend it.
#
preempt_order: "SCR 80 SC 50 S"
```

4.9.33.13 Preemption Caveats and Recommendations

- Do not use preemption via deletion along with a runjob hook that can reject the job often. In this case jobs are deleted and the preempting job is rejected.
- When using any of the fairshare, soft limits, express queue, or starving jobs preemption levels, be sure to enable the corresponding PBS feature. For example, when using preemption with the fairshare preemption level, be sure to turn fairshare on. Otherwise, you will be using stale fairshare data to preempt jobs.
- It's important to be careful about the order of the preemption levels and the sizes of the limits at queue and server. For example, if you make users who are over their server soft limits have higher priority than users who are over their queue soft limits, and you set the soft limit higher at the server than at the queue, you can end up with users who have more jobs running preempting users who have fewer jobs running.

  In this example, a user with more jobs preempts a user with fewer jobs.
- If a subjob is not running because its array job has hit the limit in max_run_subjobs, the subjob is not eligible to start and PBS does not try to use preemption to start the subjob.

Given the following:

- preempt_prio attribute contains "server_softlimits, queue_softlimits"
- Server soft limit is 5
- Queue soft limit is 3
- User1 has 6 jobs running
- User2 has 4 jobs running

This means:

- User1 has higher priority, because User1 is over the server soft limit
- User1's jobs can preempt User2's jobs

To avoid this scenario, you could set the preempt_prio attribute to contain "server_softlimits, queue_softlimits, server_softlimits+queue_softlimits". In this case User1 would have lower priority, because User1 is over both soft limits.

- Preemption priority is mostly independent of execution priority. You can list preemption levels in any order in preempt_prio, but be careful not to work at cross-purposes with your execution priority. Be sure that you are not preempting jobs that have higher execution priority. See section 4.9.16, "Calculating Job Execution Priority", on page 134.
- Using preemption with strict ordering and backfilling may change which job is being backfilled around.
- When a job is suspended via checkpoint or requeue, it loses it queue wait time. This does not apply to preemption via suspension.
- If a high-priority job has been selected to preempt lower-priority jobs, but is rejected by a runjob hook, a scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.
- A job that has requested an AOE will not preempt another job, regardless of whether the job's requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOEs.
- If a job is checkpointed by a scheduler because it was preempted, a scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.
- When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. See section 9.6.3, "Setting Job Requeue Timeout", on page 445.
- When you issue "qrun <job ID>", without the -H option, the selected job has preemption priority between Reservation and Express, for that scheduling cycle. However, at the following scheduling cycle, the preemption priority of the selected job returns to whatever it would be without qrun.
- When sched_preempt_enforce_resumption is set to *True*, all suspended jobs become top jobs, regardless of their setting for topjob_ineligible.
- PBS will not use suspension or checkpointing to preempt a job that requests a value for eoe.
- Do not use suspend/resume and the cgroups hook on the same hosts. Doing so may result in jobs being rejected.

4.9.34 Using Primetime and Holidays

Often it is useful to run different scheduling policies for specific intervals during the day or work week. PBS provides a way to specify two types of interval, called *primetime* and *non-primetime*.

Between them, primetime and non-primetime cover all time. There is no time slot that is neither primetime nor non-primetime. This includes dedicated time. Primetime and/or non-primetime overlap dedicated time.

You can use non-primetime for such tasks as running jobs on desktop clusters at night.

4.9.34.1 How Primetime and Holidays Work

By default, primetime is 24/7. A scheduler looks in the <sched_priv directory>/holidays file for definitions of primetime, non-primetime, and holidays. You can edit this file to define your holidays and primetime.

Many PBS scheduling parameters can be specified separately for primetime, non-primetime, or all of the time. This means that you can use, for example, fairshare during primetime and no fairshare during non-primetime. These parameters have a time slot default of all, meaning that if enabled, they are in force all of the time.

A scheduler applies the parameters defined for primetime during the primetime time slots, and applies parameters defined for non-primetime during the non-primetime time slots. Any scheduler parameters defined for all time are run whether it is primetime or not.

Any holidays listed in the holidays file are treated as non-primetime. To have a holiday treated like a normal workday or weekend, do not list it in the holidays file.

There are default behaviors for primetime and non-primetime, but you can set up the behavior you want for each type. The names "primetime" and "non-primetime" are meant to be informative, but they are arbitrary. The default for primetime is 24/7, meaning that primetime is all of the time by default. Example holidays are provided, but commented out, in the holidays file.

You can define primetime and non-primetime queues. Jobs in these queues can run only during the designated time. Queues that are not defined specifically as primetime or non-primetime queues are called "anytime queues".

4.9.34.2 Configuring Primetime and Non-primetime

In order to use primetime and non-primetime, you must have a holidays file with the current year in it.

You can specify primetime and non-primetime time slots by specifying them in the <sched_priv directory>/holidays file.

The format of the primetime and non-primetime section of the holidays file is the following:

YEAR YYYY

<day> <prime> <nonprime>

<day> <prime> <nonprime>

In *YEAR YYYY, YYYY* is the current year.

Day can be *weekday, monday, tuesday, wednesday, thursday, friday, saturday,* or *sunday.*

Each day line must have all three fields.

Any line that begins with a "*" or a "#" is a comment.

Weekday names must be lowercase.

The ordering of elements in this file is important. The ordering of <day> lines in the holidays file controls how primetime is determined. A later line takes precedence over an earlier line.

For example:

```
weekday     0630    1730
friday      0715    1600
```

means the same as

```
monday      0630    1730
tuesday     0630    1730
wednesday   0630    1730
thursday    0630    1730
friday      0715    1600
```

However, if a specific day is followed by "weekday",

```
friday          0700    1600
weekday         0630    1730
```

the "weekday" line takes precedence, so Friday will have the same primetime as the other weekdays.

Times can be expressed as one of the following:

- *HHMM* with no colons(:)
- The word "`all`"
- The word "`none`"

4.9.34.3 Configuring Holidays

You can specify primetime and non-primetime time slots by specifying them in the `<sched_priv directory>/holidays` file.

You must specify the year, otherwise primetime is in force at all times, and PBS will not recognize any holidays. Specify the year here, where YYYY is the current year:

YEAR YYYY

Holidays are specified in lines of this form:

<day of year> <month day-of-month> <holiday name>

PBS uses the <day of year> field and ignores the <date> string.

Day of year is the julian day of the year between 1 and 365 (e.g. "1").

Month day-of-month is the calendar date, for example "Jan 1".

Holiday name is the name of the holiday, for example "New Year's Day".

4.9.34.4 Example of `holidays` File

```
YEAR    2020
*             Prime   Non-Prime
* Day         Start   Start
*
  weekday     0600    1730
  saturday    none    all
  sunday      none    all
*
* Day of    Calendar      Company Holiday
* Year       Date         Holiday
     1       Jan 1        New Year's Day
    20       Jan 20       Martin Luther King Day
    48       Feb 17       Presidents Day
   146       May 25       Memorial Day
   186       Jul 4        Independence Day
   251       Sep 7        Labor Day
   286       Oct 12       Columbus Day
   316       Nov 11       Veterans Day
   331       Nov 26       Thanksgiving
   360       Dec 25       Christmas Day
```

4.9.34.5 Reference Copy of `holidays` File

A reference copy of the holidays file contains example holidays that are commented out. It is provided in `PBS_EXEC/etc/pbs_holidays.` The file looks like this:

```
* UNCOMMENT AND CHANGE THIS TO THE CURRENT YEAR
*YEAR 1970
*
* Prime/Nonprime Table
*
* Prime Non-Prime
* Day Start Start
*
* UNCOMMENT AND SET THE REQUIRED PRIME/NON-PRIME START TIMES
* weekday 0600 1730
* saturday none all
* sunday none all
*
* Day of Calendar Company
* Year Date Holiday
*

* UNCOMMENT AND ADD CALENDAR HOLIDAYS TO BE CONSIDERED AS NON-PRIME DAYS
* 1 Jan 1 New Year's Day
* 359 Dec 25 Christmas Day
```

4.9.34.6 Defining Primetime and Non-primetime Queues

Jobs in a primetime queue can start only during primetime. Jobs in a non-primetime queue can start only during non-primetime. Jobs in an anytime queue can start at any time.

You define a primetime queue by naming it using the primetime prefix. The prefix is defined in the primetime_prefix scheduler parameter. The default is "*p_*". For example, you could name a primetime queue "*p_queueA*", using the default.

Similarly, you define a non-primetime queue by prefixing the name. The prefix is defined in the nonprimetime_prefix scheduler parameter, and defaults to "*np_*".

4.9.34.7 Controlling Whether Jobs Cross Primetime Boundaries

You can control whether jobs are allowed to start running in one time slot and finish in another, for example when job A starts during primetime and finishes a few minutes into non-primetime. When a job runs past the boundary, it delays the start of a job that is constrained to run only in the later time slot. For example, if job B can run only during non-prime-time, it may have to wait while job A uses up non-primetime before it can start. You can control this behavior for all queues, or you can exempt anytime queues, controlling only primetime and non-primetime queues. You can also specify how much time past the boundary a job is allowed to run.

To prevent a scheduler from starting any jobs which would run past a primetime/non-primetime boundary, set the backfill_prime scheduler parameter to *True.* You can specify this separately for primetime and non-primetime. If you specify it for one type of time slot, it prevents those jobs from crossing the next boundary. For example, if you set the following:

```
backfill_prime True prime
```

jobs in primetime slots are not allowed to cross into non-primetime slots.

If you set the following:

```
backfill_prime True non_prime
```

jobs in non-primetime slots are not allowed to cross into primetime slots.

To exempt jobs in anytime queues from the control of backfill_prime, set the prime_exempt_anytime_queues scheduler parameter to *True*. This means that jobs in an anytime queue are not prevented from running across a primetime/nonprimetime or non-primetime/primetime boundary.

To allow jobs to spill over a certain amount of time past primetime/non-primetime boundaries, but no more, specify this amount of time in the prime_spill scheduler parameter. You can specify separate behavior for primetime and non-primetime jobs. For example, to allow primetime jobs to spill by 20 minutes, but only allow non-primetime jobs to spill by 1minute:

```
prime_spill 00:20:00 prime
prime_spill 00:01:00 non_prime
```

The prime_spill scheduler parameter applies only when backfill_prime is *True*.

4.9.34.8 Logging

A scheduler logs a message at the beginning of each scheduling cycle indicating whether it is primetime or not, and when this period of primetime or non-primetime will end. The message is at log level 0x0100. The message is of this form:

```
"It is primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```

or

```
"It is non-primetime and it will end in NN seconds at MM/DD/YYYY HH:MM:SS"
```

4.9.34.9 Scheduling Parameters Affecting Primetime

backfill_prime

This scheduler will not run jobs which would overlap the boundary between primetime and non-primetime.

Format: *Boolean*

Default: *False all*

nonprimetime_prefix

Queue names which start with this prefix will be treated as non-primetime queues. Jobs within these queues will only run during non-primetime.

Format: *String*

Default: *np_*

primetime_prefix

Queue names starting with this prefix are treated as primetime queues. Jobs will only run in these queues during primetime.

Format: *String*

Default: *p_*

prime_exempt_anytime_queues

Determines whether anytime queues are controlled by backfill_prime.

If set to *True*, jobs in an anytime queue will not be prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an anytime queue may not cross this boundary, except for the amount specified by their prime_spill setting.

Format: *Boolean*

Default: *False*

prime_spill

Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for prime- and non-primetime. This option is only meaningful if backfill_prime is *True*.

Format: *Duration*

Default: *00:00:00*

4.9.34.10 Caveats for Primetime and Holidays

- In order to use primetime and non-primetime, you must have a `holidays` file with the current year in it. If there is no `holidays` file with a year in it, primetime is in force all of the time.
- You cannot combine `holidays` files.
- If you use the formula, it is in force all of the time.
- If there is no *YEAR* line in the holidays file, primetime is in force at all times. If there is more than one *YEAR* line, the last one is used.
- If the information for any day is missing or incorrect, primetime is in force for all of that day.

4.9.35 Provisioning

PBS provides automatic provisioning of an OS or application, on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

For a complete description of how provisioning works and how to configure it, see Chapter 7, "Provisioning", on page 327.

4.9.36 Queue Priority

Queues and queue priority play several different roles in scheduling, so this section contains pointers to other sections.

Each queue can have a different priority. A higher value for priority means the queue has greater priority. By default, queues are sorted from highest to lowest priority. Jobs in the highest priority queue will be considered for execution before jobs from the next highest priority queue. If queues don't have different priority, queue order is undefined.

Each queue's priority is specified in its priority attribute By default, the queue priority attribute is unset. There is no limit to the priority that you can assign to a queue, however it must fit within integer size. See "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

4.9.36.1 Configuring Queue Priority

You can specify the priority of each queue by setting a value for its priority attribute:

```
Qmgr: set queue <queue name> priority = <value>
```

4.9.36.2 Using Queue Priority

You can configure a scheduler so that job execution or preemption priority is partly or entirely determined by the priority of the queue in which the job resides. Queue priority can be used for the following purposes:

- Queue priority can be used as a term in the job sorting formula. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149
- Queue priority can be used to specify the order in which queues are examined when scheduling jobs. If you want jobs to be examined queue by queue, in order of queue priority, you must specify a different priority for each queue. A queue with a higher value is examined before a queue with a lower value. See section 4.3.5.3.i, "Using Queue Order to Affect Order of Consideration", on page 65
- You can set up execution priority levels that include jobs in express queues. For information on configuring job priorities in a scheduler, see section 4.9.16, "Calculating Job Execution Priority", on page 134.
- You can set up preemption levels that include jobs in express queues. For information on preemption, see section 4.9.33, "Using Preemption", on page 180.

A queue is an express queue if its priority is greater than or equal to the value that defines an express queue. For more about using express queues, see section 4.9.18, "Express Queues", on page 137.

4.9.36.3 Queue Priority Caveats

- If you use queue priority in the formula and the job is moved to another server through peer scheduling, the queue priority used in the formula will be that of the new queue to which the job is moved.

4.9.37 Reservations

PBS provides a way to reserve specific resources for a defined time period. If you want reservations in which to run jobs, you can make one-time reservations (called *advance reservations*), or you can make a series of reservations (called *standing reservations*), where each one is for the same resources, but for a different time period. Or, if you want to secure resources for a specific (perhaps troublesome) job, you can create a *job-specific reservation* for that job at submission time, while the job is queued, or later while the job is running.

If you want to sequester hosts for maintenance, you can create a *maintenance reservation*. Maintenance reservations block out time on specified machines, preventing jobs from being started where you need to perform maintenance tasks.

Advance, standing, and job-specific reservations are "job reservations", to distinguish them from maintenance reservations.

Reservations are useful for accomplishing the following job-related tasks:

- To get a time slot on a specific host
- To run a job in a specific time slot, meaning at or by a specific time
- To be sure a job will run
- To have a high-priority job run soon
- To make sure that a job doesn't lose access to resources when needs to be re-run

4.9.37.1 Definitions

Advance reservation

A reservation for a set of resources for a specified time. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

Degraded reservation

A job-specific or advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with the soonest occurrence are unavailable.

Job-specific reservation

A reservation created for a specific job, for the same resources that the job requested.

Job-specific ASAP reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS schedules the reservation to run as soon as possible, and PBS moves the job into the reservation. Created when you use `pbs_rsub -Wqmove=<job ID>` on a queued job.

Job-specific now reservation

Reservation created for a specific running job. PBS immediately creates a job-specific now reservation on the same resources as the job is using, and moves the job into the reservation. The reservation is created and starts running immediately when you use `pbs_rsub --job <job ID>` on a running job.

Job-specific start reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS starts the job according to scheduling policy. When the job starts, PBS creates and starts the reservation, and PBS moves the job into the reservation. Created when you use `qsub -Wcreate_resv_from_job=true` on a queued job.

Maintenance reservation

A reservation designed for performing maintenance on the specified hosts for the specified time. Created using `pbs_rsub --hosts <host list>`.

Occurrence of a standing reservation

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See "qsub" on page 213 of the PBS Professional Reference Guide.
- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the resv_nodes attribute reported by `pbs_rstat`.

Also called an *instance* of a standing reservation.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Standing reservation

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

4.9.37.2 Job Reservations

4.9.37.2.i Creating Advance and Standing Reservations

Any PBS user can create both advance and standing reservations for jobs using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The `pbs_rsub` command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<sequence number>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<sequence number>.<server name>

The user specifies the resources for a reservation using the same syntax as for a job.

See "Reserving Resources", on page 135 of the PBS Professional User's Guide, for detailed information on creation and use of reservations.

The time for which a reservation is requested is in the time zone at the submission host.

4.9.37.2.ii Job-Specific Reservations

A job-specific reservation is for the same resources that the job requested. Job-specific reservations are intended to preserve access to the job's resources in the case where a job may need to be modified and then re-run, so that the job does not need to wait to be re-scheduled.

Any PBS user can create a job-specific reservation.

A job-specific reservation ID has the format:

R<sequence number>.<server name>

Job-specific reservations cannot be used with job arrays.

4.9.37.2.iii Creating Job-specific Start Reservations

Job submitters can create a job-specific start reservation at submission time. The job is scheduled normally, and when it starts, PBS creates and starts a reservation on the same resources, and puts the job into the reservation. To create a job-specific reservation at submission time, set the job's create_resv_from_job attribute to *True*:

qsub ... -Wcreate_resv_from_job=1

To create a job-specific start reservation from a queued job, use `qalter` to set the create_resv_from_job attribute to *True*.

4.9.37.2.iv Creating Job-specific ASAP Reservations

Job submitters can create a job-specific ASAP from a queued job. PBS creates the reservation for the same resources the job requests, moves the job into the reservation, and schedules the reservation to start as soon as possible.

To create a job-specific ASAP reservation:

pbs_rsub -Wqmove=<job ID>

Note that job-specific ASAP reservations, once created, do not adjust themselves to a change in resource availability. An ASAP reservation can cause resources to go idle while waiting for the reservation to start. For example, if a job scheduled to finish before an ASAP reservation finishes early, and no jobs can be backfilled into the new open slot, resources will sit idle until the reservation runs. In addition, if a high-priority job comes in after an ASAP reservation has been created for a lower-priority job, the high-priority job must wait until after the reservation finishes.

To get the equivalent of flexible ASAP reservations that don't cause idle resources, use a job sort formula with a custom priority term, for example "cust_high_pri", and set this term to a high value, for example 10, for the desired job. Then you can alter the job: `qalter -l cust_high_pri=10 -Wcreate_resv_from_job=true`. Make sure that cust_high_pri has a large enough coefficient in the formula to change the job priority.

4.9.37.2.v Creating Job-specific Now Reservations

Job submitters can create a job-specific now from a running job. PBS creates the reservation for the same resources the job is using, starts the reservation, and moves the job into the reservation.

To create a job-specific now reservation:

pbs_rsub --job <job ID>

4.9.37.2.vi Job Reservations and Placement Sets

When PBS chooses a placement set for a reservation, it makes the same choices as it would for a regular job. It fits the reservation into the smallest possible placement set. See section 4.9.32.4.ii, “Order of Placement Set Consideration Within Pool”, on page 172.

When a reservation is created, it is created within a placement set, if possible. If no placement set will satisfy the reservation, placement sets are ignored, if the scheduler's do_not_span_psets attribute is *False*. If no placement set will satisfy the reservation, and the scheduler's do_not_span_psets attribute is *True*, the reservation is not created.

The vnodes allocated to a reservation are used as one single placement set for jobs in the reservation; they are not subdivided into smaller placement sets. A job within a reservation runs within the single placement set made up of the vnodes allocated to the reservation.

4.9.37.2.vii Requesting Resources for Job Reservations

Reservations request resources using the same mechanism that jobs use. If a resource is unrequestable, users cannot request it for a reservation. If a resource is invisible, users cannot view it or request it for a reservation.

4.9.37.2.viii Job Reservations and Provisioning

Users can create reservations that request AOEs. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE. See section 7.4.3, “Provisioning And Reservations”, on page 331.

The vnodes allocated to a reservation that requests an AOE are put in the *resv-exclusive* state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

For information on restrictions applying to reservations used with provisioning, see section 7.7.2.3, “Vnode Reservation Restrictions”, on page 344.

For how to avoid problems with provisioning and reservations, see section 7.10.1, “Using Provisioning Wisely”, on page 353.

4.9.37.2.ix Job Reservation Priority

A job running in a reservation cannot be preempted.

A job running in a reservation has the highest execution priority.

4.9.37.2.x Querying Reservations

To query a reservation, use the `pbs_rstat` command. See "Viewing the Status of a Reservation", on page 143 of the PBS Professional User's Guide. To delete a reservation, use the `pbs_rdel` command, not the `qmgr` command.

4.9.37.2.xi Controlling Access to Job Reservations

You can specify which projects, users, and groups can and cannot submit jobs to reservations. Use the `pbs_rsub -U/-G` command to set the reservation's acl_users and/or acl_groups attributes, and `pbs_ralter -U/-G` to change them. See section 8.3.8, "Reservation Access", on page 371.

4.9.37.2.xii Job Reservation Fault Tolerance

PBS automatically keeps track of the vnodes assigned to reservations, and tries to find replacement vnodes for those that become unavailable. See section 9.4, "Reservation Fault Tolerance", on page 432.

4.9.37.2.xiii Logging Standing Reservation Information

The start and end of each occurrence of a standing reservation is logged as if each occurrence were a single advance reservation.

Reservation-related messages are logged at level PBSEVENT_RESV, which is 0x0200 (512).

4.9.37.2.xiv Accounting

Resources requested for a reservation are recorded in the reservation's Resource_List attribute, and reported in the accounting log B and Y records for the reservation. See section 19.5.3, "Timeline for Reservation Accounting Messages", on page 647.

4.9.37.3 Maintenance Reservations

You can create maintenance reservations using `pbs_rsub --hosts <host list>`. Maintenance reservations are designed to make the specified hosts available for the specified amount of time, regardless of what else is happening:

- You can create a maintenance reservation that includes or is made up of vnodes that are down or offline.
- Maintenance reservations ignore the value of a vnode's resv_enable attribute.
- PBS immediately confirms any maintenance reservation.
- Maintenance reservations take precedence over other reservations; if you create a maintenance reservation that overlaps an advance or standing job reservation, the overlapping vnodes become unavailable to the job reservation, and the job reservation is in conflict with the maintenance reservation. PBS looks for replacement vnodes; see "Reservation Fault Tolerance" on page 432 in the PBS Professional Administrator's Guide.

PBS will not start any new jobs on vnodes overlapping or in a maintenance reservation. However, jobs that were already running on overlapping vnodes continue to run; you can let them run or requeue them.

You cannot specify place or select for a maintenance reservation; these are created by PBS:

- PBS creates the reservation's placement specification so that hosts are assigned exclusively to the reservation. The placement specification is always the following:

  -lplace=exclhost
- PBS sets the reservation's resv_nodes attribute value so that all CPUs on the reserved hosts are assigned to the maintenance reservation. The select specification is always the following:

  -lselect=host=<host1>:ncpus=<number of CPUs at host1>+host=<host2>:ncpus=<number of CPUs at host2>+...

Maintenance reservations are prefixed with *M*. A maintenance reservation ID has the format:

M<sequence number>.<server name>

You cannot create a recurring maintenance reservation.

Creating a maintenance reservation does not trigger a scheduling cycle.

You must have manager or operator privilege to create a maintenance reservation.

4.9.37.4 Modifying Reservations

You can use the pbs_ralter command to alter an existing reservation, whether it is an individual job-specific, advance, or maintenance reservation, or the next or current instance of a standing reservation. Syntax:

pbs_ralter [-D <duration>] [-E <end time>] [-G <auth group list>] [-I <block time>] [-l select=<select spec>] [-m <mail points>] [-M <mail list>] [-N <reservation name>] [-R <start time>] [-U <auth user list>] <reservation ID>

To force a change to the start time, end time, or duration of a reservation, you can use the `-Wforce` option:

pbs_ralter -Wforce [-D <duration>] [-E <end time>] [-R <start time>] <reservation ID>

Note that with the `-Wforce` option you can force PBS to oversubscribe resources, in which case you (the administrator) may need to manage them yourself.

You cannot change the start time of a reservation in which jobs are running.

When changing the select specification, the new specification must be a subset of the same chunks requested by the original reservation. If the reservation has started, you cannot release chunks where reservation jobs are running. To find unused chunks in a running reservation, you can compare the reservation's resv_nodes attribute to the exec_vnodes attribute of the jobs running in the reservation.

If the reservation has started and and is degraded, you must release all unavailable chunks in order to alter the reservation select specification.

If the reservation has not started, modifying the select specification may result in moving the reservation to different vnodes.

After the change is requested, the change is either confirmed or denied. On denial of the change, the reservation is not deleted and is left as is, and the following message appears in the server's log:

```
Unable to alter reservation <reservation ID>
```

When a reservation is confirmed, the following message appears in the server's log:

```
Reservation alter successful for <reservation ID>
```

To find out whether or not the change was allowed:

- Use the `pbs_rstat` command: see whether you altered reservation attribute(s)
- Use the interactive option: check for confirmation after the blocking time has run out

If the reservation has not started and it cannot be confirmed on the same vnodes, PBS searches for another set of vnodes. See section 9.4, “Reservation Fault Tolerance”, on page 432.

You must be the reservation owner or the PBS Administrator to run this command.

For details, see “pbs_ralter” on page 86 of the PBS Professional Reference Guide.

4.9.37.5 Attributes Affecting Reservations

We list the server, vnode, and job attributes affecting reservations here. See the full list of reservation attributes in "Reservation Attributes" on page 303 in the PBS Professional Administrator's Guide. See "Server Attributes" on page 281 of the PBS Professional Reference Guide and "Vnode Attributes" on page 320 of the PBS Professional Reference Guide.

Table 4-17: Attributes Affecting Reservations

Entity	Attribute	Effect
Server	acl_resv_host_enable	Controls whether or not the server uses the acl_resv_hosts access control lists.
Server	acl_resv_hosts	List of hosts from which reservations may and may not be created at this server.
Server	acl_resv_group_enable	Controls whether or not the server uses the acl_resv_groups access control lists.
Server	acl_resv_groups	List of groups who may and may not create reservations at this server.
Server	acl_resv_user_enable	Controls whether or not the server uses the acl_resv_users access control lists.
Server	acl_resv_users	List of users who may and may not create reservations at this server.
Server	resv_enable	Controls whether or not reservations can be created at this server.
Server	reserve_retry_time	Length of time to wait between when a reservation becomes degraded and when PBS tries to reconfirm the reservation, as well as interval between attempts to reconfirm a degraded reservation. Default: 600 (10 minutes)
Vnode	queue **deprecated**	Associates the vnode with an execution queue. If this attribute is set, this vnode cannot be used for reservations.
Vnode	resv_enable	Controls whether the vnode can be used for reservations. Default is *True*, but set to *False* for a vnode used for cycle harvesting.
Job	create_resv_from_job	Controls whether PBS creates a job-specific reservation for this job.

4.9.37.6 Reservation Advice and Caveats

- Do not delete a reservation's queue.
- Do not start a reservation's queue (do not set the reservation's started attribute to *True*). Jobs will run prematurely.
- Do not try to set attribute values for a reservation queue directly; instead, operate on the reservation.
- Reservations are incompatible with cycle harvesting. Do not allow reservations on machines used for cycle harvesting. The user may begin using the machine, which will suspend any PBS jobs, possibly preventing them from finishing before the reservation runs out. Set each cycle harvesting vnode's resv_enable attribute to *False*, to prevent the vnode from being used for reservations.
- You can write hooks that execute, modifying a reservation's attributes, when a reservation is created. See the PBS Professional Hooks Guide.
- Allow enough time in reservations. If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.
- Hosts or vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance reservations. Hosts or vnodes that are being used for cycle harvesting should not be used for reservations.
- Hosts with $max_load and $ideal_load configured should not be used for reservations. Set the resv_enable vnode attribute on these hosts to *False*.
- For troubleshooting problems with reservations, see "Reservation Caveats and Errors", on page 147 of the PBS Professional User's Guide.
- Be careful when using `qrun -H` on jobs or vnodes involved in reservations. Make sure that you don't oversubscribe reserved resources.
- In order to create reservations, the submission host must have its timezone set to a value that is understood by the PBS server. See section 21.9.5, "Unrecognized Timezone Variable", on page 675.
- Avoid making reservations for resources that are out of the control of PBS. Resources that are managed through a server_dyn_res script may not be available when jobs need them.
- If you create a maintenance reservation that overlaps an advance or standing job reservation, the maintenance reservation takes precedence, the overlapping vnodes become unavailable to the job reservation, and the job reservation becomes degraded. PBS looks for replacement vnodes; see section 9.4, "Reservation Fault Tolerance", on page 432. Any job reservation overlapping a maintenance reservation goes into the *RESV_IN_CONFLICT* substate (*12*).
- Note that job-specific ASAP reservations, once created, do not adjust themselves to a change in resource availability. An ASAP reservation can cause resources to go idle while waiting for the reservation to start. For example, if a job scheduled to finish before an ASAP reservation finishes early, and no jobs can be backfilled into the new open slot, resources will sit idle until the reservation runs.
- To get the equivalent of flexible ASAP reservations that don't cause idle resources, use a job sort formula with a custom priority term, for example "cust_high_pri", and set this term to a high value, for example 10, for the desired job. Then you can alter the job: `qalter -l cust_high_pri=10 -Wcreate_resv_from_job=true`.
- Beware of oversubscribing resources when using the `-Wforce` option to `pbs_ralter`.

4.9.38 Round Robin Queue Selection

PBS can select jobs from execution queues by examining the queues in round-robin fashion. The behavior is round-robin only when you have groups of queues where all queues in each group have the same priority.

The order in which queues are selected is determined by each queue's priority. You can set each queue's priority; see section 2.3.5.3, "Prioritizing Execution Queues", on page 24. If queue priorities are not set, they are undefined. If you do not prioritize the queues, their order is undefined.

When you have multiple queues with the same priority, a scheduler round-robins through all of the queues with the same priority as a group. So if you have Q1, Q2, and Q3 at a priority of 100, Q4 and Q5 at a priority of 50, and Q6 at a priority of 10, a scheduler will round-robin through Q1, Q2, and Q3 until all of those jobs are out of the way, then the scheduler will round-robin through Q4 and Q5 until there are no more jobs in them, and finally the scheduler will go through Q6.

When using the round-robin method with queues that have unique priorities, a scheduler runs all jobs from the first queue, then runs all the jobs in the next queue, and so on.

To specify that PBS should use the round-robin method to select jobs, set the value of the round_robin scheduler parameter to *True*.

The round_robin parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

You can use the round-robin method as a resource allocation tool. For example, if you need to run the same number of jobs from each group, you can put each group's jobs in a different queue, and then use round-robin to run jobs, one from each queue.

The round-robin method is also used in PBS for a feature that is not controlled by the round_robin scheduler attribute. The SMP cluster distribution parameter, smp_cluster_dist, can use a round-robin method to place jobs. See section 4.9.43, "SMP Cluster Distribution", on page 218.

See "round_robin" on page 256 of the PBS Professional Reference Guide.

4.9.38.1 Round-robin Caveats

- Each scheduling cycle starts with the highest-priority queue. Therefore, when using round-robin, this queue gets preferential treatment.
- When set to *True*, the round_robin parameter overrides the by_queue parameter.
- If round robin and strict ordering are *True*, and backfilling is not being used, and the top job cannot run, whether because of resources or rejection by MoM, no job runs. However, if round robin is *True* and strict ordering is *False*, and the top job in the current queue cannot run, the next top job is considered instead. For example, we have 3 queues, each with 3 jobs, and with the same priority:

  Q1: J1 J2 J3

  Q2: J4 J5 J6

  Q3: J7 J8 J9

  If round_robin and strict_ordering are *True*, and J1 cannot run, no job runs.

  If round_robin is *True* and strict_ordering is *False*, and J1 cannot run, job order is J4, J7, J2, J5, J8, J3, etc.
- With round_robin and strict_ordering set to *True*, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job with a start time at some later time to allow other jobs in the same queue to be run.

4.9.39 Routing Jobs

Before reading this section, please read about the mechanics of configuring and using routing queues, in section 2.3.6, "Routing Queues", on page 25.

In this section, we use the term "routing" to mean the general process of moving a job somewhere, whether it is from one queue to another, from one partition or complex to another, or from a queue to particular vnodes.

Routing jobs can involve collecting jobs so they don't stray into the wrong queues, moving those jobs to the correct queues, and filtering which jobs are allowed into queues.

You may need to collect jobs into a routing queue, before moving them to the correct destination queue. If you use a routing queue, you can force users to submit jobs to the routing queue only, you can grab jobs as they are submitted and put them in the routing queue, and you can set a routing queue as the default. The mechanisms to collect jobs are described below, and listed here:

- Setting default queue; see section 4.9.39.1.i, “Default Queue as Mechanism to Collect Jobs”, on page 206
- Grabbing jobs upon submission; see section 4.9.39.1.ii, “Grabbing Jobs Upon Submission”, on page 207
- Disallowing direct submission to execution queues; see section 4.9.39.1.iii, “Disallowing Direct Submission as Mechanism to Collect Jobs”, on page 207
- Disallowing submission using access controls; see section 4.9.39.3.ii, “Access Controls as Filtering Mechanism”, on page 208

There is also a one-step process, but depending on the number of jobs being submitted, it may be too slow. You can also simply examine them upon submission and send them where you want. The method is listed here:

- Examining jobs upon submission and routing them using a hook; see section 4.9.39.1.iv, “Examining Jobs Upon Submission”, on page 207.

You can use any of several mechanisms for moving jobs. Each is described in subsections below. The mechanisms for moving jobs are the following:

- Routing queues; see section 4.9.39.2.i, “Routing Queues as Mechanism to Move Jobs”, on page 207
- Hooks; see section 4.9.39.2.ii, “Hooks as Mechanism to Move Jobs”, on page 207
- Peer scheduling; see section 4.9.39.2.iii, “Peer Scheduling as Mechanism to Move Jobs”, on page 207
- The `qmove` command; see section 4.9.39.2.iv, “The qmove Command as Mechanism to Move Jobs”, on page 208

You can use filtering methods to control which jobs are allowed into destination queues. We describe filtering methods in subsections below. The filtering mechanisms are the following:

- Resource limits; jobs are filtered by resource request. See section 4.9.39.3.i, “Resource Limits as Filtering Mechanism”, on page 208
- Access control limits; jobs are filtered by owner. See section 4.9.39.3.ii, “Access Controls as Filtering Mechanism”, on page 208

You can use a combination of moving a job and "tagging" it, that is, including a special custom resource in the job's resource request, to route the job. If you set the resource using a hook, you can route the job either to a queue or to vnodes. If you make the job inherit the resource from a queue, you can route it only to vnodes. You can set resource limits for the special custom resource at the receiving queue, allowing in only jobs with the special resource. You can set the special custom resource at vnodes, so that the job must run there. Mechanisms for tagging jobs are listed here:

- Using a hook to assign a resource; see section 4.9.39.4.i, “Using Hooks to Tag Jobs”, on page 208
- Associating vnodes with queues; see section 4.9.2, “Associating Vnodes with Queues”, on page 103
- Changing the job's resource request using the `qalter` command; see section 4.9.39.4.ii, “Using the qalter Command to Tag Jobs”, on page 209

4.9.39.1 Mechanisms for Collecting Jobs

4.9.39.1.i Default Queue as Mechanism to Collect Jobs

To make it easy on your users, have their jobs land in your routing queue by default. You probably don't want frustrated users trying to submit jobs without specifying a queue, only to have the jobs be rejected if you have set access controls on, or only allowed routing to, the default queue. The server's default_queue attribute specifies the name of the default queue. To make things easy, make the default queue be the routing queue:

```
Qmgr: set server default_queue = <queue name>
```

4.9.39.1.ii Grabbing Jobs Upon Submission

You can allow users to submit jobs to any queue, and then scoop up the newly-submitted jobs and put them in the desired queue. To do this, you write a hook. See the *PBS Professional Hooks Guide*.

4.9.39.1.iii Disallowing Direct Submission as Mechanism to Collect Jobs

If you are using a routing queue, you can disallow job submission to all other queues. This forces users to submit jobs to the routing queue. You should probably make the routing queue be the default queue in this case, to avoid irritating users. Whether or not a queue allows direct job submission is controlled by its from_route_only attribute. To disallow job submission to a queue:

```
Qmgr: set queue <queue name> from_route_only = True
```

4.9.39.1.iv Examining Jobs Upon Submission

You can use a job submission hook to examine each job as it is submitted, and then route it to the desired queue. For example, you can route jobs directly according to resource request, project, owner, etc. See the *PBS Professional Hooks Guide*.

4.9.39.2 Mechanisms for Moving Jobs

4.9.39.2.i Routing Queues as Mechanism to Move Jobs

Routing queues are a mechanism supplied by PBS that automatically move jobs from a routing queue to another queue. You can direct which destination queues accept a job using these filters at each destination queue:

- Resource limits: you can set up execution queues designed for specific kinds of jobs, and then route each kind of job separately. For example, you can create two execution queues, and one routing queue, and route all jobs requesting large amounts of memory to one of the execution queues, and the rest of the jobs to the other queue. See section 2.3.6.4, “Using Resources to Route Jobs Between Queues”, on page 26.
- Access control limits: you can set up destination queues that are designed for specific groups of users. Each queue accepts jobs only from a designated set of users or groups. For example, if you have three departments, Math, Physics, and Chemistry, the queue belonging to Math accepts only users from the Math department. See section 2.3.6.5, “Using Access Control to Route Jobs”, on page 29.

When routing a job between complexes, the job's owner must be able to submit a job to the destination complex.

For how to configure and use routing queues, see section 2.3.6, “Routing Queues”, on page 25.

4.9.39.2.ii Hooks as Mechanism to Move Jobs

You can use a submission hook to move jobs into queues such as dedicated time queues, queues with special priority, or reservation queues. You write the hook so that it identifies the jobs that should go into a particular queue, and then moves them there. For example, your hook can move all jobs from ProjectA to a specific queue. This is a snippet, where you would replace <destination queue> with the queue name.

```
import pbs
e = pbs.event()
e.job.queue = pbs.server().queue("<destination queue>")
```

For complete information on hooks, see the PBS Professional Hooks Guide.

4.9.39.2.iii Peer Scheduling as Mechanism to Move Jobs

To send jobs from one partition or complex to another, you use peer scheduling. In peer scheduling, the partition or complex that supplies the jobs (the "furnishing" partition or complex) contains at least one special queue (the "furnishing queue"), whose jobs can be pulled over to another partition or complex, to be run at the other partition or complex. The partition or complex that pulls jobs contains a special queue (the "pulling queue"), where those pulled jobs land.

You can use any of the job routing methods, such as routing queues, tagging, or hooks, to control which jobs land in the furnishing queue.

You can use any of the job filtering methods, such as resource limits or access controls, to control which jobs land in the furnishing queue.

You can use job submission hooks on the jobs that land in the pulling queue.

See section 4.9.31, “Peer Scheduling”, on page 165.

4.9.39.2.iv The `qmove` Command as Mechanism to Move Jobs

You can use the `qmove` command, either manually or via a `cron` job, to move jobs into the desired queues. See “qmove” on page 172 of the PBS Professional Reference Guide.

4.9.39.3 Mechanisms for Filtering Jobs

4.9.39.3.i Resource Limits as Filtering Mechanism

You can filter whether each job is accepted at the server or a queue based on the job's resource request. For example, you can control which jobs are allowed to be submitted to the server, by limiting the amount of memory a job is allowed to request. You can do the same at execution queues. These limits apply regardless of the routing mechanism being used, and apply to jobs being submitted directly to the queue. See section 5.13, “Using Resources to Restrict Server or Queue Access”, on page 254.

4.9.39.3.ii Access Controls as Filtering Mechanism

You can filter jobs whether each job is accepted at the server or a queue based on the job's owner, or the job owner's group. At each queue and at the server, you can create a different list of the users who can submit jobs and the users who cannot submit jobs. You can do the same for groups.

For example, you can set up a routing queue and several execution queues, where each execution queue has access controls allowing only certain users and groups. When PBS routes the jobs from the routing queue, it will route them into the execution queues that accept owners of the jobs. See section 2.3.6.5, “Using Access Control to Route Jobs”, on page 29.

4.9.39.3.iii Hooks as Filtering Mechanism

You can filter which jobs are accepted at the server or queues according to any criterion, using a hook. For example, you can write a hook that disallows jobs that request certain combinations of resources. See the PBS Professional Hooks Guide.

4.9.39.4 Mechanisms for Tagging Jobs

4.9.39.4.i Using Hooks to Tag Jobs

You can use a hook to force certain jobs to run on particular hardware, by having the hook set the value of a host-level custom resource in a job's resource request. The hook sets this resource to match the value at the selected vnodes, so that the job must run on one or more of those vnodes. You can use the job's project to determine how the job is tagged. Note that the value at other vnodes should be different, otherwise the job could end up on vnodes you don't want.

- Define a host-level custom resource; see section 5.14.4, “Configuring Host-level Custom Resources”, on page 269.
- Set this resource to a special value on the special vnodes only. See section 5.7.2, “Setting Values for Global Static Resources”, on page 241.
- Create a hook that filters jobs by size, project, or other characteristic, and sets the value of the custom resource to the special value, in the job's resource request. See the PBS Professional Hooks Guide

If you must use a routing queue, and you need to route on host-level resources (resources in the job's select specification), you can use a hook to tag jobs so that they are routed correctly. The hook reads the job's host-level resource request, and sets the job's server-level resource request accordingly. This server-level resource is used for routing:

- Create a custom server-level resource that you use exclusively for routing; set it to appropriate values on the destination queues; see section 5.14.3, "Creating Server-level Custom Resources", on page 267
- Create a submit hook to extract the host-level resource value and use it to populate the custom resource that you use exclusively for routing; see the PBS Professional Hooks Guide

4.9.39.4.ii Using the `qalter` Command to Tag Jobs

You can change a job's resource request using the qalter command. This way you can override normal behavior. See "qalter" on page 127 of the PBS Professional Reference Guide.

4.9.40 Scheduler Cycle Speedup

4.9.40.1 Top Job Calculation Speedup

When you are using backfilling, you can choose whether and how much you want to speed up the scheduling cycle (within limits). You can get shorter scheduling cycle duration with coarser granularity in estimating start times for jobs. When you are using backfilling, a scheduler calculates estimated start times for jobs. You can choose not to make this trade-off (keeping fine granularity in start time estimation), or you can choose low, medium, or high speedup. See section 4.9.3, "Using Backfilling", on page 105.

4.9.40.1.i Configuring Top Job Calculation Speedup

You configure top job calculation speedup by using `qmgr` to set the opt_backfill_fuzzy scheduler attribute:

```
Qmgr: set sched opt_backfill_fuzzy [off | low | medium | high]
```

where each option has the following effect:

off
: This scheduler uses its normal, finest granularity. No speedup.

low
: This scheduler uses fairly fine granularity, not as fine as normal. Some speedup.

medium
: This scheduler uses medium granularity. Medium speedup.

high
: This scheduler uses the coarsest granularity. Greatest speedup.

The options *off*, *low*, *medium*, and *high* are not case-sensitive. You can use only one option at a time. Since this is an attribute and not a scheduler parameter, it is not a primetime option.

4.9.40.1.ii What Changing Calculation Speed Affects

Changing this attribute takes effect on the next scheduling cycle. If you change this attribute, top jobs are recalculated in the next scheduling cycle.

Once an ASAP reservation is made, it is fixed. If you change opt_backfill_fuzzy later, the reservation start time does not change, even if it becomes degraded. PBS finds new vnodes for degraded reservations, but does not change the start times.

4.9.40.1.iii Caveats and Restrictions for Top Job Calculation Speedup

This option is effective only when you are using backfilling.

4.9.41 Shared vs. Exclusive Use of Resources by Jobs

When PBS places a job, it can do so on hardware that is either already in use or has no jobs running on it. PBS can make the choice at the vnode level or at the host level. How this choice is made is controlled by a combination of the value of each vnode's sharing attribute and the placement requested by a job.

You can set each vnode's sharing attribute so that the vnode or host is always shared, is always exclusive, or so that it honors the job's placement request. If the vnode attribute is set to *force_shared* or *force_excl*, the value of a vnode's sharing attribute takes precedence over a job's placement request. If the vnode attribute is set to *default_*, the job request overrides the vnode attribute.

Each vnode can be allocated exclusively to one job (each job gets its own vnodes), or its resources can be shared among jobs (PBS puts as many jobs as possible on a vnode). If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

Hosts can also be allocated exclusively to one job, or shared among jobs. If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.

For a complete description of the sharing attribute, and a table showing the interaction between the value of the sharing attribute and the job's placement request, see “sharing” on page 324 of the PBS Professional Reference Guide.

4.9.41.1 Sharing on a Multi-vnode Machine

On a multi-vnode shared-memory machine, a scheduler will share memory from a chunk even if all the CPUs are used by other jobs. It will first try to put a chunk entirely on one vnode. If it can, it will run it there. If not, it will break the chunk up across any vnode it can get resources from, even for small amounts of unused memory.

To keep a job in a single vnode, use `-lplace=group=vnode`; if you want to restrict it to larger sets of vnodes, identify those sets using a custom string or string_array resource and use it in `-lplace=group=<resource>`. If you already have resources used in node_group_key you can usually use these.

4.9.41.2 Setting the sharing Vnode Attribute

To set the sharing attribute for a vnode, use either:

- An exechost_startup hook; see "Setting and Unsetting Vnode Resources and Attributes" on page 48 in the PBS Professional Hooks Guide
- A Version 2 configuration file; see section 3.4.4, “Configuring the Vnode Sharing Attribute”, on page 46

4.9.41.3 Viewing Sharing Information

You can use the `qmgr` or `pbsnodes` commands to view sharing information. See “qmgr” on page 149 of the PBS Professional Reference Guide and “pbsnodes” on page 35 of the PBS Professional Reference Guide.

4.9.41.4 Sharing Caveats

- On the Cray XC, on *cray_compute* vnodes, the sharing attribute is set to *force_exclhost* by default. Do not change this setting, because ALPS does not support sharing a compute vnode with more than one job.
- The term "sharing" is also used to describe the case where MoM manages a resource that is shared among her vnodes, for example an application license shared by the vnodes of a multi-vnode machine.
- The term "sharing" is also used to mean oversubscribing CPUs, where more than one job is run on one CPU; the jobs are "sharing" a CPU. See section 9.6.5, “Managing Load Levels on Vnodes”, on page 445
- If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.
- For vnodes with sharing=*default_shared*, jobs can share a vnode, so that unused memory on partially-allocated vnodes is allocated to a job. The exec_vnode attribute will show this allocation.

4.9.42 Using Shrink-to-fit Jobs

4.9.42.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the min_walltime and max_walltime resources. PBS chooses the actual walltime. Any job that requests min_walltime is a **shrink-to-fit** job.

4.9.42.1.i Requirements for a Shrink-to-fit Job

A job must have a value for min_walltime to be a shrink-to-fit job. Shrink-to-fit jobs are not required to request max_walltime, but it is an error to request max_walltime and not min_walltime.

Jobs that do not have values for min_walltime are not shrink-to-fit jobs, and their walltime can be specified by the user, inherited through defaults, or set in a hook.

4.9.42.1.ii Comparison Between Shrink-to-fit and Non-shrink-to-fit Jobs

Shrink-to-fit jobs are treated the same as non-shrink-to-fit jobs unless explicitly stated. For example, job priority is not affected by being shrink-to-fit. The only difference between a shrink-to-fit and a non-shrink-to-fit job is how the job's walltime is treated. PBS sets the walltime at the time the job is run; any walltime settings not computed by PBS are ignored.

4.9.42.2 Where to Use Shrink-to-fit Jobs

If you have jobs that can run for less than the expected time to completion and still make useful progress, you can use them as shrink-to-fit jobs in order to maximize utilization.

You can use shrink-to-fit jobs for the following:

- Jobs that are internally checkpointed. This includes jobs which are part of a larger effort, where a job does as much work as it can before it is killed, and the next job in that effort takes up where the previous job left off.
- Jobs using periodic PBS checkpointing
- Jobs whose real running time might be much less than the expected time
- When you have set up dedicated time for system maintenance, and you want to keep machines well-utilized right up until shutdown, submitters who want to risk having a job killed before it finishes can run speculative shrink-to-fit jobs. Similarly, speculative jobs can take advantage of the time just before a reservation starts
- Any job where the submitter does not mind running the job as a speculative attempt to finish some work

4.9.42.3 Running Time of a Shrink-to-fit Job

4.9.42.3.i Setting Running Time Range for Shrink-to-fit Jobs

It is only required that the job request min_walltime to be a shrink-to-fit job. If a job requests min_walltime but does not request max_walltime, you may want to use a hook or defaults to set a reasonable value for max_walltime. If you use defaults, you may want to route shrink-to-fit jobs to a special queue where they inherit a value for max_walltime if they haven't got one already. See section 4.9.39, “Routing Jobs”, on page 205.

Requesting max_walltime without requesting min_walltime is an error.

A job can end up with a value for min_walltime and max_walltime when the user specifies them, when it inherits them from server or queue defaults, or when they are set in a hook.

Job submitters can set the job's running time range by requesting min_walltime and max_walltime, for example:

qsub -l min_walltime=<min walltime>, max_walltime=<max walltime> <job script>

You can set min_walltime or max_walltime using a hook, whether or not the job requests it. You can set up defaults so that the job inherits these resources if they are not explicitly requested or set in a hook.

4.9.42.3.ii Inheriting Values for min_walltime and max_walltime

The min_walltime and max_walltime resources inherit values differently. A job can inherit a value for max_walltime from resources_max.walltime; the same is not true for min_walltime. This is because once a job is shrink-to-fit, PBS can use a walltime limit for max_walltime.

If a job is submitted without a value for min_walltime, the value for min_walltime for the job becomes the first of the following that exists:

- Server's default `qsub` arguments
- Queue's resources_default.min_walltime
- Server's resources_default.min_walltime

If a shrink-to-fit job is submitted without a value for max_walltime, the value for max_walltime for the job becomes the first of the following that exists:

- Server's default `qsub` arguments
- Queue's resources_default.max_walltime
- Server's resources_default.max_walltime
- Queue's resources_max.walltime
- Server's resources_max.walltime

4.9.42.3.iii Setting walltime for Shrink-to-fit Jobs

For a shrink-to-fit job, PBS sets the walltime resource based on the values of min_walltime and max_walltime, regardless of whether walltime is specified for the job. You cannot use a hook to set the job's walltime, and any queue or server defaults for walltime are ignored, except for the case where the job is run via `qrun -H`; see section 4.9.42.8.ii, “Using qrun With -H Option”, on page 214.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's min_walltime and max_walltime. If the job can fit somewhere, PBS sets the job's walltime to a duration that fits the time slot, and runs the job. The chosen value for walltime is visible in the job's Resource_List.walltime attribute. Any existing walltime value, regardless of where it comes from (user, queue default, hook, previous execution), is reset to the new calculated running time.

If a shrink-to-fit job is run more than once, PBS recalculates the job's running time to fit an available time slot that is between min_walltime and max_walltime, and resets the job's walltime, each time the job is run.

4.9.42.4 How PBS Places Shrink-to-fit Jobs

A PBS scheduler treats shrink-to-fit jobs the same way as it treats non-shrink-to-fit jobs when it schedules them to run. A scheduler looks at each job in order of priority, and tries to run it on available resources. If a shrink-to-fit job can be shrunk to fit in an available slot, a scheduler runs it in its turn. A scheduler chooses a time slot that is at least as long as the job's min_walltime value. A shrink-to-fit job may be placed in a time slot that is shorter than its max_walltime value, even if a longer time slot is available.

For a multi-vnode job, PBS chooses a walltime that works for all of the chunks required by the job, and places job chunks according to the placement specification.

4.9.42.5 Shrink-to-fit Jobs and Time Boundaries

The time boundaries that constrain job running time are the following:

- Reservations
- Dedicated time
- Primetime
- Start time for a top job

Time boundaries are not affected by shrink-to-fit jobs.

A shrink-to-fit job can shrink to avoid time boundaries, as long as the available time slot before the time boundary is greater than min_walltime.

If any job is already running, whether or not it is shrink-to-fit, and you introduce a new period of dedicated time that would impinge on the job's running time, PBS does not kill or otherwise take any action to prevent the job from hitting the new boundary.

4.9.42.5.i Shrink-to-fit Jobs and Prime Time

If you have enabled prime time by setting backfill_prime to *True*, shrink-to-fit jobs will honor the boundary between primetime and non-primetime. If prime_spill is *True*, shrink-to-fit jobs are scheduled so that they cross the prime-non-prime boundary by up to prime_spill duration only. If prime_exempt_anytime_queues is set to *True*, a job submitted in an anytime queue is not affected by primetime boundaries.

4.9.42.6 Shrink-to-fit Jobs and Resource Limits

4.9.42.6.i Shrink-to-fit Jobs and Gating at Server or Queue

Shrink-to-fit jobs must honor any resource limits at the server or queues. If a walltime limit is specified:

- Both min_walltime and max_walltime must be greater than or equal to resources_min.walltime.
- Both min_walltime and max_walltime must be less than or equal to resources_max.walltime.

If resource limits are not met, a job submission or modification request will fail with the following error:

```
"Job exceeds queue and/or server resource limits"
```

4.9.42.6.ii Gating Restrictions

You cannot set resources_min or resources_max for min_walltime or max_walltime. If you try, you will see the following error message, for example for min_walltime:

```
"Resource limits can not be set for min_walltime"
```

4.9.42.7 Shrink-to-fit Jobs and Preemption

When preempting other jobs, shrink-to-fit jobs do not shrink. Their walltime is set to their max_walltime.

4.9.42.8 Using qrun on Shrink-to-fit Jobs

If you use qrun on a shrink-to-fit job, its behavior depends on whether you use the -H option to qrun.

4.9.42.8.i Using qrun Without -H Option

When a shrink-to-fit job is run via qrun, it can shrink into available space to run. However, if preemption is enabled and there is a preemptable job that must be preempted in order to run the shrink-to-fit job, the preemptable job is preempted and the shrink-to-fit job shrinks and runs.

When a shrink-to-fit job is run via qrun, and there is a hard deadline, e.g. reservation or dedicated time, that conflicts with the shrink-to-fit job's max_walltime but not its min_walltime, the following happens:

- If preemption is enabled and there is a preemptable job before the hard deadline that must be preempted in order to run the shrink-to-fit job, preemption behavior means that the shrink-to-fit job does not shrink to fit; instead, it conflicts with the deadline and does not run.
- If preemption is enabled and there is no preemptable job before the hard deadline, the shrink-to-fit job shrinks into the available time and runs.

4.9.42.8.ii Using qrun With -H Option

When a shrink-to-fit job is run via qrun -H, the shrink-to-fit job runs, regardless of reservations, dedicated time, other jobs, etc. When run via qrun -H, shrink-to-fit jobs do not shrink. If the shrink-to-fit job has a requested or inherited value for walltime, that value is used, instead of one set by PBS when the job runs. If no walltime is specified, the job runs without a walltime.

4.9.42.9 Modifying Shrink-to-fit and Non-shrink-to-fit Jobs

4.9.42.9.i Modifying min_walltime and max_walltime

You can change min_walltime and/or max_walltime for a shrink-to-fit job using modifyjob or queuejob hooks, or by using the qalter command. Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.9.42.9.ii Making Non-shrink-to-fit Jobs into Shrink-to-fit Jobs

You can convert a normal non-shrink-to-fit job into a shrink-to-fit job using the following methods:

- Use a hook that does the following:
  - Sets max_walltime to the job's walltime
  - Sets min_walltime to a useful value
- Use resources_default at the server or a queue. For a queue, you might want to set that queue's from_route_only attribute to True.
- Route to a queue that has resources_default.min_walltime set.
- Use the qalter command to set values for min_walltime and max_walltime.

Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

4.9.42.9.iii Making Shrink-to-fit Jobs into Non-shrink-to-fit Jobs

To make a shrink-to-fit job into a normal, non-shrink-to-fit job, use either a hook or the `qalter` command to do the following:

- Set the job's walltime to the value for max_walltime (beware of allowing the job to run into existing reservations etc.)
- Unset min_walltime
- Unset max_walltime

4.9.42.9.iv Hooks for Running Time Limits

If you want to set a new running time limit for shrink-to-fit jobs, you can use a hook. However, this hook must set the value of max_walltime, rather than walltime, since hook settings for walltime for a shrink-to-fit job are ignored.

4.9.42.10 Viewing Running Time for a Shrink-to-fit Job

4.9.42.10.i Viewing min_walltime and max_walltime

You can use `qstat -f` to view the values of the min_walltime and max_walltime. For example:

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<job-id>
% qstat -f <job-id>
...
resource_list.min_walltime=01:00:15
resource_list.max_walltime=03:30:00
```

You can use tracejob to display max_walltime and min_walltime as part of the job's resource list. For example:

```
12/16/2011 14:28:55  A    user=pbsadmin group=Users project=_pbs_project_default
…
    Resource_List.max_walltime=10:00:00
    Resource_List.min_walltime=00:00:10
```

4.9.42.10.ii Viewing walltime for a Shrink-to-fit Job

PBS sets a job's walltime only when the job runs. While the job is running, you can see its walltime via `qstat -f`. While the job is not running, you cannot see its real walltime; it may have a value set for walltime, but this value is ignored.

You can see the walltime value for a finished shrink-to-fit job if you are preserving job history. See section 14.15, "Managing Job History", on page 530.

You can see the walltime value for a finished shrink-to-fit job in the scheduler log.

4.9.42.11 Lifecycle of a Shrink-to-fit Job

4.9.42.11.i Execution of Shrink-to-fit Jobs

Shrink-to-fit jobs are started just like non-shrink-to-fit jobs.

4.9.42.11.ii Termination of Shrink-to-fit Jobs

When a shrink-to-fit job exceeds the walltime PBS has set for it, it is killed by PBS exactly as a non-shrink-to-fit job is killed when it exceeds its walltime.

4.9.42.12 The min_walltime and max_walltime Resources

max_walltime

Maximum walltime allowed for a shrink-to-fit job. Job's actual walltime is between max_walltime and min_walltime. PBS sets walltime for a shrink-to-fit job. If this resource is specified, min_walltime must also be specified. Must be greater than or equal to min_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. If not specified, PBS uses an eternal time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

min_walltime

Minimum walltime allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's walltime. Job's actual walltime is between max_walltime and min_walltime. Must be less than or equal to max_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

4.9.42.13 Accounting and Logging for Shrink-to-fit Jobs

4.9.42.13.i Accounting Log Entries for min_walltime and max_walltime

The accounting log will contain values for min_walltime and max_walltime, as part of the job's Resource_List attribute. This attribute is recorded in the S, E, and R records in the accounting log. For example, if the following job is submitted:

```
qsub -l min_walltime="00:01:00",max_walltime="05:00:00" -l select=2:ncpus=1 job.sh
```

This is the resulting accounting record:

```
…S…….. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18 resources_assigned.ncpus=2


…R…….. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18


…E……. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18…….
```

4.9.42.13.ii Logging

- When a scheduler finds a primetime/dedicated time conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level PBSEVENT_DEBUG2:

  ```
  "Considering shrinking job to duration=<duration>, due to prime/dedicated time conflict"
  ```

  Sample message from the scheduler log:

  ```
  "03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3;Considering shrinking job to duration=1:06:05,
      due to a prime/dedicated time conflict"
  ```

  This message doesn't indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job's maximum running time conflicted with primetime and the job can still be run by shrinking its running time.

- When a scheduler finds a reservation/top job conflict with a shrink-to-fit job, and the job can be shrunk, the following message is logged in the scheduler logs, with log level PBSEVENT_DEBUG2:

  ```
  "Considering shrinking job to duration=<duration>", due to reservation/top job conflict"
  ```

Sample log message from the scheduler log:

```
"03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3;   Considering shrinking job to
   duration=1:06:05, due to  reservation/top job conflict"
```

This message doesn't indicate or guarantee that the job will eventually be shrunk and run. This message shows that the job's maximum running time conflicted with a reservation or top job and the job can still be run by shrinking its running time.

- When a scheduler runs the shrink-to-fit job, the following message is logged in the scheduler logs with log level PBSEVENT_DEBUG2:

  ```
  "Job will run for duration=<duration>"
  ```

  Sample scheduler log message:

  ```
  "03/26/2012 11:53:55;0040;pbs_sched;Job;98.host3;Job will  run for duration=1:06:05"
  ```

4.9.42.14 Caveats and Restrictions for Shrink-to-fit Jobs

- It is erroneous to specify max_walltime for a job without specifying min_walltime. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:

  ```
  'Can not have "max_walltime" without "min_walltime"'
  ```

- It is erroneous to specify a min_walltime that is greater than max_walltime. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:

  ```
  '"min_walltime" can not be greater than "max_walltime"'
  ```

- Job arrays cannot be shrink-to-fit. You cannot have a shrink-to-fit job array. It is erroneous to specify a min_walltime or max_walltime for a job array. If a queuejob or modifyjob hook attempts this, the following error appears in the server logs. If attempted via `qsub` or `qalter`, the following error appears in the server log and is printed as well:

  ```
  '"min_walltime" and "max_walltime" are not valid resources for a job array'
  ```

- Reservations cannot be shrink-to-fit. You cannot have a shrink-to-fit reservation. It is erroneous to set min_walltime or max_walltime for a reservation. If attempted via `pbs_rsub`, the following error is printed:

  ```
  '"min_walltime" and "max_walltime" are not valid resources for reservation.'
  ```

- It is erroneous to set resources_max or resources_min for min_walltime and max_walltime. If attempted, the following error message is displayed, whichever is appropriate:

  ```
  "Resource limits can not be set for min_walltime"
  "Resource limits can not be set for max_walltime"
  ```

4.9.43 SMP Cluster Distribution

This tool is **deprecated**. PBS provides a method for distributing single-chunk jobs to a cluster of single-vnode machines according to a simple set of rules. The method is called *SMP cluster distribution*. It takes into account the resources specified on the `resources:` line in `<sched_priv directory>/sched_config`. The SMP cluster distribution method allows you to choose one of three job distribution systems:

Table 4-18: SMP Cluster Distribution Options

Option	Meaning
pack	Pack all jobs onto one vnode, until that vnode is full, then move to the next vnode
round_robin	Place one job on each vnode in turn, before cycling back to the first vnode
lowest_load	Place the job on the host with the lowest load average

4.9.43.1 How to Use SMP Cluster Distribution

To use SMP cluster distribution, do the following:

- Set the smp_cluster_dist scheduler parameter to the desired value. For example, to enable SMP cluster distribution using the round robin algorithm during primetime, and the pack algorithm during non-primetime, set the following in the scheduler's configuration file:

  ```
  smp_cluster_dist: round_robin prime
  smp_cluster_dist: pack non_prime
  ```

- Set resources_available.<resource name> to the desired limit on each vnode. You do not need to set any of the resources that are automatically set by PBS. For a list of these, see section 5.7.1.1, "How Vnode Available Resource Values are Set", on page 239.
- Specify the resources to use during scheduling, in `<sched_priv directory>/sched_config`:

  ```
  resources: "ncpus, mem, arch, host, ..."
  ```

The smp_cluster_dist parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.9.43.2 How To Disable SMP Cluster Distribution

To ensure that SMP cluster distribution does not interfere with your scheduling policy, leave the smp_cluster_dist parameter set to its default value:

```
smp_cluster_dist   pack   all
```

4.9.43.3 SMP Cluster Distribution Caveats and Advice

- This feature was intended for early implementations of complexes, and probably is not useful for you.
- If you use this feature, you are committed to using it for the entire partition or complex; you cannot designate some machines where it will be used and others where it will not be used.
- If smp_cluster_dist with either *round_robin* or *lowest_load* is used with node_sort_key set to *unused* or *assigned*, smp_cluster_dist is set to *pack*.
- The avoid_provision provisioning policy is incompatible with the smp_cluster_dist scheduler configuration parameter. If a job requests an AOE, the avoid_provision policy overrides the behavior of smp_cluster_dist.
- This feature is applied only to single-chunk jobs that specify an arrangement of pack. Multi-chunk jobs are ignored.
- This feature is useful only for single-vnode machines. On a multi-vnoded machine, this feature distributes jobs across vnodes, but those jobs can end up all stuck on a single host.
- The choice of smp_cluster_dist with round_robin can be replaced by sorting vnodes according to unused CPUs, which does a better job:

  ```
  node_sort_key: "ncpus HIGH unused"
  ```

4.9.44 Using Soft Walltime

A scheduler requires walltime to do backfilling. Job submitters want to avoid having their jobs killed if they run over their walltimes, so they may overestimate job walltimes. You can give a scheduler tighter time slots by giving jobs soft walltimes. Jobs are not killed if they go over their soft walltimes. If a job has both a walltime and a soft walltime, a scheduler uses the soft walltime.

When a job exceeds its soft walltime, PBS estimates a new soft walltime, and records the estimate in the job's estimated.soft_walltime attribute. The estimated.soft_walltime job attribute is readable by all, but writable only by PBS.

4.9.44.1 Assigning Soft Walltime to Jobs

You can set a soft walltime for a job by having it request the soft_walltime resource. You can set it in a server hook, or by using `qalter` or resources_default.

The soft_walltime resource can be requested for a job only by PBS Managers. The soft_walltime resource cannot be set at job submission time, except by a queuejob hook, because job submission uses user permissions. Soft walltime cannot be set in MoM hooks.

You can create a custom resource and allow users to request it, and then set the value of soft_walltime to that resource. See an example in section 4.9.44.4, “Allowing Job Submitters to Set Soft Walltime”, on page 220.

4.9.44.2 How Soft and Hard Walltimes Are Used

When a job is queued:

- If the job is a top job, its soft_walltime is used in determining where the job fits into the calendar
- If the job is a filler job, its soft_walltime is used in determining whether the job conflicts with top jobs
- If the job is a filler job, its hard walltime is used in determining whether the job conflicts with confirmed reservations
- If dedicated time is used, soft_walltime is used in determining whether the job will finish before dedicated time starts
- If backfill_prime is set, soft_walltime is used in determining whether the job will finish before the next prime boundary + prime_spill

When a job is running:

- If resources_used.walltime <= soft_walltime, the job continues to run
- If resources_used.walltime > soft_walltime, the job has exceeded its soft_walltime. The job is not killed; the job's soft_walltime is extended:
  - Every time the job exceeds its soft_walltime, it is extended by 100% of its original soft_walltime
  - If both a soft_walltime and a hard walltime are set, the soft_walltime is never extended past the job's hard walltime
  - If a job exceeds its soft_walltime and crosses over into dedicated_time, PBS does not kill the job
  - The value of Resource_List.soft_walltime does not change. A scheduler sets the estimated.soft_walltime job attribute to the new soft_walltime estimate
- If a job is a preemption candidate, and preempt_order is based on the percentage the job has completed (e.g., `preempt_order SCR 20 S`), the initial soft_walltime request is used to determine the percentage of completion
  - If the job runs past its initial soft_walltime request, preempt_order behaves as if the job is 100% complete. It remains at 100% complete for the remainder of the job regardless of how many times the soft_walltime is extended. For example, if a job has soft_walltime=1:00:00, at 59m, the job is at 99% complete. At 1:00:00, the soft_walltime is extended to 2:00:00. At 1:30:00 the job remains at 100% complete since it has reached its original soft_walltime request

When confirming reservations:

- Only a job's hard walltime is used in determining when jobs end
- A job's soft_walltime is not used when confirming reservations

4.9.44.3 Examples of Using Soft Walltime

Example 4-27: Job J has a soft_walltime=1:00:00 but no hard walltime

J exceeds its soft_walltime. J is extended by its original soft_walltime to 2:00:00.

If J exceeds its soft_walltime again, J is extended again by its original soft_walltime to 3:00:00

Example 4-28: Job K has a soft_walltime=1:00:00 and a hard walltime=1:30:00

K exceeds its soft_walltime. Because 2:00:00 is past its hard walltime, K is extended to its limit of 1:30:00 instead.

4.9.44.4 Allowing Job Submitters to Set Soft Walltime

Example of hook to allow users to directly set soft_walltime:

```
import pbs
e = pbs.event()
j = e.job

j.Resource_List["soft_walltime"] = pbs.duration(j.Resource_List["set_soft_walltime"])
```

Job submitters request the new resource:

```
% qsub -l set_soft_walltime=1:00:00 -l select=1:ncpus=1
```

4.9.44.5 Caveats and Restrictions for Soft Walltime

- The soft_walltime resource is not sent to the MoM when the job is started.
- A shrink-to-fit job requesting soft_walltime is rejected, because a job's min_walltime is the minimum amount of time a job needs to get any real work done. A job's hard walltime can be set to its min_walltime. A job's soft_walltime has to be shorter than its hard walltime. This means that the soft_walltime would have to be shorter than the job's minimum amount of time to get any real work done. The two features do not make sense together.

4.9.45 Sorting Jobs on a Key

PBS allows you to sort jobs on a key that you specify. This can be used when setting both execution and preemption priority. Sorting jobs comes into play after jobs have been divided into classes, because each class may contain more than one job. You can sort on one or more of several different keys, and for each key, you can sort either from low to high or from high to low.

You configure sorting jobs on a key by setting values for the job_sort_key scheduler parameter. When preemption is enabled, jobs are automatically sorted by preemption priority. Table 4-8, “Job Execution Classes,” on page 135 shows where this step takes place.

You can create an invisible, unrequestable custom resource, and use a hook to set the value of this resource for each job. The hook modifies the job's resource request to include the new resource, and sets the value to whatever the hook computes. Then you can sort jobs according to the value of this resource.

The job_sort_key parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

4.9.45.1 job_sort_key Syntax

job_sort_key: "<sort key> HIGH | LOW <primetime option>"

You can use the following keys for sorting jobs:

Table 4-19: Keys for Sorting Jobs

Sort Key	Allowed Order	Description
<PBS resource>	HIGH | LOW	Sorts jobs according to how much of the specified resource they request.
fairshare_perc	HIGH | LOW	Sorts according to fairshare percentage allotted to entity that owns job. This percentage is defined in the `resource_group` file. If user A has more priority than user B, all of user A's jobs are always run first. Past history is not used.
job_priority	*HIGH | LOW*	Sorts jobs by the value of each job's priority attribute.
sort_priority	*HIGH | LOW*	**Deprecated**. Replaced by job_priority option.

You can sort on up to 20 keys.

The argument to the job_sort_key parameter is a quoted string. The default for job_sort_key is that it is not in force.

See “job_sort_key” on page 252 of the PBS Professional Reference Guide.

4.9.45.2 Configuring Sorting Jobs on a Key

You can specify more than one sort key, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for job_sort_key, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the job_sort_key scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- Specify the primetime behavior

A scheduler's configuration file is read on startup and HUP.

4.9.45.3 Examples of Sorting Jobs on Key

Example 4-29: Sort jobs so that those with long walltime come first:

```
job_sort_key: "walltime HIGH"
```

Example 4-30: For example, if you want big jobs to run first, where "big" means more CPUs, and if the CPUs are the same, more memory, sort on the number of CPUs requested, then the amount of memory requested:

```
job_sort_key: "ncpus HIGH" all
job_sort_key: "mem HIGH" all
```

Example 4-31: Sort jobs so that those with lower memory come first:

```
job_sort_key: "mem LOW" prime
```

Example 4-32: Sort jobs according to the value of an invisible custom resource called *JobOrder*:

```
job_sort_key: "JobOrder LOW" all
```

4.9.45.4 Caveats and Advice for Sorting Jobs on Key

- Do not use fairshare_perc as the sort key when using fairshare, meaning the fair_share scheduler parameter is enabled. If you do this, a scheduler will attempt to sort a set of jobs where each job has the same sort key value. This will not sort the jobs.
- Use the fairshare_perc option only when ordering jobs by entity shares. See section 4.9.14, “Sorting Jobs by Entity Shares (Was Strict Priority)”, on page 131.
- To run big jobs first, use ncpus as the primary sort key for job_sort_key:
  ```
  job_sort_key: "ncpus HIGH"
  ```
- The job_sort_key parameter is overridden by the job sorting formula and by fairshare. It is invalid to set both job_sort_formula and job_sort_key at the same time. If they are both set, job_sort_key is ignored and the following error message is logged:
  ```
  "Job sorting formula and job_sort_key are incompatible. The job sorting formula will be used."
  ```
- A scheduler's configuration file contains an example line for job_sort_key. This line is commented out, but shows an example of job_sort_key with "*cput*" as the sorting key.
- The preempt_priority argument to the job_sort_key parameter is **deprecated**. Jobs are now automatically sorted by preemption priority when preemption is enabled.

4.9.46 Sorting Jobs by Requested Priority

You can sort jobs according to the priority that was requested for the job. This value is found in the job's Priority attribute. You can use this value in the following ways:

- The term job_priority represents the value of the job's priority attribute in the job sorting formula. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.
- The job_sort_key scheduler parameter can take the term job_priority as an argument. The term *job_priority* represents the value of the job's Priority attribute. See section 4.9.45, "Sorting Jobs on a Key", on page 221.

You can use a hook to set or change the value of a job's Priority attribute. See the PBS Professional Hooks Guide.

4.9.47 Sorting Queues into Priority Order

PBS always sorts all the execution queues in your partition or complex according to their priority, and uses that ordering when examining queues individually. Queues are ordered with the highest-priority queue first.

If you want queues to be considered in a specific order, you must assign a different priority to each queue. Give the queue you want considered first the highest priority, then the next queue the next highest priority, and so on. To set a queue's priority, use the `qmgr` command to assign a value to the priority queue attribute.

```
Qmgr: set queue <queue name> priority = <value>
```

Sorting queues into priority order is useful for the following:

- Examining queues one at a time. See section 4.9.4, "Examining Jobs Queue by Queue", on page 110.
- Selecting jobs from queues in a round-robin fashion. See section 4.9.38, "Round Robin Queue Selection", on page 204.

4.9.47.1 Caveats and Advice when Sorting Queues

- If you do not set queue priorities, queue ordering is undefined.
- The sort_queues parameter is **obsolete** (version 20).

4.9.48 Starving Jobs

PBS can keep track of the amount of time a job has been waiting to run, and then mark the job as *starving* if this time has passed a specified limit. You can use this starving status in calculating both execution and preemption priority.

4.9.48.1 Enabling Starving

You enable tracking whether jobs are starving by setting the help_starving_jobs scheduler parameter to *True*.

You specify the amount of time required for a job to be considered starving in the max_starve scheduler parameter. The default for this parameter is 24 hours.

The help_starving_jobs parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See "help_starving_jobs" on page 251 of the PBS Professional Reference Guide.

4.9.48.2 Time Used for Starving

PBS can use one of the following kinds of time to determine whether a job is starving:

- The job's eligible wait time, described in section 4.9.13, "Eligible Wait Time for Jobs", on page 126
- The amount of time the job has been queued

You specify which to use in the server's eligible_time_enable attribute. When eligible_time_enable is set to *True*, each job's eligible_time value is used as its wait time for starving. If eligible_time_enable is set to *False*, the amount of time the job has been queued is used as its wait time for starving. The default for eligible_time_enable is *False*.

If the server's eligible_time_enable attribute is set to *False*, the following rules apply:

- The amount of time the job has been queued is used as its wait time for starving.
- Jobs lose their queue wait time whenever they are requeued, as with the `qrerun` command. This includes when they are checkpointed or requeued (but not suspended) during preemption.
- Suspended jobs do not lose their queue wait time. However, when they become suspended, the amount of time since they were submitted is counted towards their queue wait time. For example, if a job was submitted, then remained queued for 1 hour, then ran for 26 hours, then was suspended, if max_starve is 24 hours, then the job will become starving.

If the server's eligible_time_enable attribute is set to *True*, the following rules apply:

- The job's eligible_time value is used as its wait time for starving.
- Jobs do not lose their eligible_time when they are requeued.
- Jobs do not lose their eligible_time when they are suspended.

4.9.48.3 Starving and Job Priority

Starving is one of the job classes used by PBS to calculate job execution priority. If you enable starving jobs, PBS will classify starving jobs in the Starving class, which gives them greater than ordinary priority. See section 4.9.16, "Calculating Job Execution Priority", on page 134. Each job's eligible wait time can also be used in the job sorting formula used to calculate job execution priority. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.

Starving is one of the job classes that you can use when specifying how preemption should work. You can choose how much preemption priority is given to starving jobs when you set preemption levels. See section 4.9.33, "Using Preemption", on page 180.

4.9.48.4 Parameters and Attributes Affecting Starving

The following table lists the parameters and attributes that affect starving:

Table 4-20: Parameters and Attributes Affecting Starving

Parameter or Attribute	Location	Effect
help_starving_jobs	`<sched_priv directory>/sched_config`	Controls whether long-waiting jobs are considered starving. When set to *True*, jobs can be starving. Default: *True all*
max_starve	`<sched_priv directory>/sched_config`	Amount of wait time for job to be considered starving. Default: 24 hours.
eligible_time_enable	Server attribute	Controls whether a job's wait time is taken from its eligible_time or from its queued time. When set to *True*, a job's eligible_time is used as its wait time. Default: *False*.
eligible_time	Job attribute	The amount of time a job has been blocked from running due to lack of resources.

4.9.48.5 Starving and Queued or Running Jobs

A job can only accumulate starving time while it waits to run, not while it runs. When a job is running, it keeps the starving status it had when it was started. While a job is running, if it wasn't starving before, it can't become starving. However, it keeps its starving status if it became starving while queued.

4.9.48.6 Starving and Subjobs

Subjobs that are queued can become starving. Starving status is applied to individual subjobs in the same way it is applied to jobs. The queued subjobs of a job array can become starving while others are running. If a job array has starving subjobs, then the job array is starving.

4.9.48.7 Starving and Backfilling

Because a starving job can become a top job, but can continue to be unable to run due to a lack of resources, you may find it useful to use backfilling around starving jobs. See section 4.9.3, “Using Backfilling”, on page 105.

4.9.48.8 Starving Caveats

Do not enable starving with fairshare, meaning do not set both the fair_share and help_starving_jobs scheduler parameters to *True*.

4.9.49 Using Strict Ordering

By default, when scheduling jobs, PBS orders jobs according to execution priority, then considers each job, highest-priority first, and runs the next job that can run now. Using strict ordering means that you tell PBS that it must not skip a job when choosing which job to run. If the top job cannot run, no job runs.

Strict ordering does not change how execution priority is calculated.

4.9.49.1 Configuring Strict Ordering

To configure strict ordering, set the strict_ordering scheduler parameter to *True*.

The strict_ordering parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time. See “strict_ordering” on page 257 of the PBS Professional Reference Guide.

4.9.49.2 How Strict Ordering Works

When strict_ordering is *True*, a scheduler runs jobs in exactly the order of their priority.

Strict ordering does not affect how job priority is calculated, but it does change which execution priority classes a scheduler uses; see section 4.9.16, “Calculating Job Execution Priority”, on page 134.

4.9.49.3 Combining Strict Ordering and Backfilling

Strict ordering alone may cause some resources to stand idle while the top job waits for resources to become available. If you want to prevent this, you can use backfilling with strict ordering. Using backfilling, if the top job cannot run, filler jobs can be squeezed in around the job that cannot run. See section 4.9.3, “Using Backfilling”, on page 105.

4.9.49.4 Strict Ordering and Calendaring

If you mark a job's topjob_ineligible attribute *True*, PBS does not put that job in the calendar if it cannot run right now. See section 4.9.17, “Calendaring Jobs”, on page 137.

4.9.49.5 Strict Ordering Caveats

- It is inadvisable to use strict ordering and backfilling with fairshare. The results may be non-intuitive. Fairshare will cause relative job priorities to change with each scheduling cycle. It is possible that a job from the same entity or group as the desired large job will be chosen as the filler job. The usage from these filler jobs will lower the priority of the top job.

  For example, if a user has a large job that is the top job, and that job cannot run, smaller jobs owned by that user will chew up the user's usage, and prevent the large job from being likely to ever run. Also, if the small jobs are owned by a user in one area of the fairshare tree, no large jobs owned by anyone else in that section of the fairshare tree are likely to be able to run.
- Using dynamic resources with strict ordering and backfilling may result in unpredictable scheduling. See section 4.9.3.11, “Backfilling Recommendations and Caveats”, on page 109.
- Using preemption with strict ordering and backfilling may change which job is the top job.
- With both round robin and strict ordering, a job continually rejected by a runjob hook may prevent other jobs from being run. A well-written hook would put the job on hold or requeue the job at some later time to allow other jobs in the same queue to be run.

4.9.50 Sorting Vnodes on a Key

PBS can sort vnodes according to a key that you specify. This can be used when deciding which vnodes to use for jobs. Sorting vnodes comes into play after a placement set has been selected, or when a job will run on vnodes associated with a queue, or when placement sets are not used, because in those cases there may be more vnodes available than are needed. You can sort vnodes on one or more different keys, and for each key, you can sort from high to low, or the reverse.

You can sort on the last_used_time and Priority vnode attributes, and vnode resources.

The default way to sort vnodes is according to the value of the vnode priority attribute, from higher to lower.

When you sort vnodes according to the assigned or unused amount of a resource, the vnode list is re-sorted after every job is run. This is because each job may change the usage for that resource.

You configure sorting vnodes on a key by setting values for the node_sort_key scheduler parameter.

The node_sort_key parameter is a primetime option, meaning that you can configure it separately for primetime and non-primetime, or you can specify it for all of the time.

When vnodes are not sorted on a key, their order is undefined.

4.9.50.1 node_sort_key Syntax

node_sort_key: "sort_priority HIGH | LOW" <prime option>

node_sort_key: "<resource name> HIGH | LOW' <prime option>

node_sort_key: "<resource name> HIGH | LOW total | assigned | unused" <prime option>

where

total

Use the resources_available value

assigned

Use the resources_assigned value

unused

Use the value given by resources_available - resources_assigned

Specifying a resource such as mem or ncpus sorts vnodes by the resource specified. Note that a scheduler rounds all resources of type size, including mem , up to the nearest kb.

Specifying the sort_priority keyword sorts vnodes on the vnode priority attribute.

The default third argument for a resource is *total*. If the third argument, *total* | *assigned* | *unused*, is not specified with a resource, *total* is used. This provides backwards compatibility with previous releases.

The values used for sorting must be numerical.

4.9.50.2 Configuring Sorting Vnodes on a Key

You can specify up to 20 sort keys, where you want a primary sort key, a secondary sort key, etc.

If you specify more than one entry for node_sort_key, the first entry is the primary sort key, the second entry is the secondary sort key, which is used to sort equal-valued entries from the first sort, and so on.

Each entry is specified one to a line.

To sort jobs on a key, set the node_sort_key scheduler parameter:

- Set the desired key
- Specify whether high or low results should come first
- For sorting on a resource, optionally specify total, assigned, or unused
- Specify the primetime behavior

A scheduler's configuration file is read on startup and HUP.

The argument to the node_sort_key parameter is a quoted string. The default for node_sort_key is the following:

```
node_sort_key: "sort_priority HIGH" all
```

See "node_sort_key" on page 253 of the PBS Professional Reference Guide.

4.9.50.3 Sorting Vnodes According to Load Average

To place jobs on the vnodes with the lowest load:

- Create a custom host-level resource to reflect current load, named for example "r5m":

  ```
  qmgr -c "create resource aveload type=long,flag=h"
  ```

- Write an exechost_periodic hook to set the resource to the value of the load average; see "Log loads on vnodes" on page 284 in the PBS Professional Installation & Upgrade Guide for an example of an exechost_periodic hook that reads the load on the host.
- Use the aveload resource as your node_sort_key:

  ```
  node_sort_key: "r5m LOW" all
  ```

4.9.50.4 Examples of Sorting Vnodes

Example 4-33: This sorts vnodes by the highest number of unused CPUs:

```
node_sort_key: "ncpus HIGH unused" all
```

Example 4-34: This sorts vnodes by the highest amount of memory assigned to vnodes, but only during primetime:

```
node_sort_key: "mem HIGH assigned" prime
```

Example 4-35: This sorts vnodes according to speed. You want to run jobs on the fastest host available. You have 3 machines, where HostA is fast, HostB is medium speed, and HostC is slow.

Set node priorities so that faster machines have higher priority:

```
Qmgr: set node HostA priority = 200
Qmgr: set node HostB priority = 150
Qmgr: set node HostC priority = 100
```

Specify that vnodes are sorted according to priority, with highest priority first:

```
node_sort_key: "sort_priority HIGH" ALL
```

Example 4-36: The old "nodepack" behavior can be achieved by this:

```
node_sort_key: "ncpus low unused"
```

Example 4-37: In this example of the interactions between placement sets and node_sort_key, we have 8 vnodes numbered 1-8. The vnode priorities are the same as their numbers. However, in this example, when unsorted, the vnodes are selected in the order 4, 1, 3, 2, 8, 7, 5, 6. This is to illustrate the change in behavior due to node_sort_key.

We use:

```
node_sort_key: "sort_priority LOW"
```

Using node_sort_key, the vnodes are sorted in order, 1 to 8. We have three placement sets:

A: 1, 2, 3, 4 when sorted by node_sort_key; 4, 1, 3, 2 when no node_sort_key is used

B: 5, 6, 7, 8 when sorted by node_sort_key; 8, 7, 5, 6 when no node_sort_key is used

C: 1-8 when sorted, 4, 1, 3, 2, 8, 7, 5, 6 when not sorted.

A 6-vnode job will not fit in either A or B, but will fit in C. Without the use of node_sort_key, it would get vnodes 4, 1, 3, 2, 8, 7. With node_sort_key, it would get vnodes 1 - 6, still in placement set C.

4.9.50.5 Caveats for Sorting Vnodes

- Sorting on a resource with node_sort_key and using "unused" or "assigned" cannot be used with load_balancing. If both are used, load balancing will be disabled.
- Sorting on a resource and using "unused" or "assigned" cannot be used with smp_cluster_dist when it is set to anything but "pack". If both are used, smp_cluster_dist will be set to "pack".
- A scheduler rounds all resources of type size, including mem, up to the nearest kb. This can affect how vnodes are sorted when you are sorting on mem.

5 Using PBS Resources

This chapter covers PBS resources, including providing resources for user jobs, setting up resources such as application licenses and scratch space, how to make objects inherit resources, and how to use, define, and view resources.

For a list of built-in and custom Cray XC resources automatically created by PBS, see "List of Built-in Resources" on page 259 of the PBS Professional Reference Guide.

5.1 Chapter Contents

5.1 Chapter Contents . . . 229
5.2 Introduction to PBS Resources . . . 230
5.3 Glossary . . . 230
5.4 Categories of Resources . . . 233
5.4.1 Built-in vs. Custom Resources . . . 233
5.4.2 Server vs. Queue vs. Vnode Resources . . . 233
5.4.3 Consumable vs. Non-consumable Resources . . . 234
5.4.4 Static vs. Dynamic Resources . . . 234
5.4.5 Global vs. Local Resources . . . 235
5.4.6 Requested vs. Default Resources . . . 235
5.4.7 Shared vs. Non-shared Vnode Resources . . . 235
5.4.8 Platform-specific vs. Generally Available Resources . . . 236
5.4.9 Job-wide vs. Chunk Resources . . . 236
5.5 Resource Types . . . 237
5.6 Resource Formats . . . 237
5.6.1 Resource Names . . . 238
5.7 Setting Values for Resources . . . 239
5.7.1 How Resource Values are Set . . . 239
5.7.2 Setting Values for Global Static Resources . . . 241
5.7.3 Setting Values for Local Static Resources . . . 241
5.7.4 Setting Values for String Arrays . . . 241
5.7.5 When Resource Changes Take Effect . . . 242
5.7.6 Caveats for Setting Resource Values . . . 242
5.8 Overview of Ways Resources Are Used . . . 243
5.8.1 How the Scheduler Uses Resources . . . 243
5.8.2 Advice on Using string and string_array Resources . . . 243
5.9 Resources Allocated to Jobs and Reservations . . . 244
5.9.1 Allocating Chunks . . . 244
5.9.2 Resources Requested by Job . . . 245
5.9.3 Specifying Job Default Resources . . . 245
5.9.4 Allocating Default Resources to Jobs . . . 247
5.9.5 Dynamic Resource Allocation Caveats . . . 250
5.9.6 Period When Resource is Used by Job . . . 250
5.10 Using Resources to Track and Control Allocation . . . 252
5.11 Using Resources for Topology and Job Placement . . . 253
5.11.1 Restrictions on Using Resources for Job Placement . . . 253
5.12 Using Resources to Prioritize Jobs . . . 254
5.13 Using Resources to Restrict Server or Queue Access . . . 254

5.13.1 Admittance Limits for walltime, min_walltime, and max_walltime 254
5.13.2 Restrictions on Resources Used for Admittance 255
5.14 Custom Resources 255
5.14.1 How to Use Custom Resources 255
5.14.2 Defining New Custom Resources 257
5.14.3 Creating Server-level Custom Resources 267
5.14.4 Configuring Host-level Custom Resources 269
5.14.5 Using Scratch Space 273
5.14.6 Supplying Application Licenses 274
5.14.7 Using GPUs 283
5.14.8 Using FPGAs 286
5.14.9 Defining Host-level Resource for Applications 286
5.14.10 Custom Resource Caveats 286
5.15 Managing Resource Usage 287
5.15.1 Managing Resource Usage By Users, Groups, and Projects, at Server & Queues 287
5.15.2 Placing Resource Limits on Jobs 304
5.15.3 Limiting the Number of Jobs in Queues 309
5.16 Where Resource Information Is Kept 309
5.16.1 Files 309
5.16.2 MoM Configuration Parameters 310
5.16.3 Attributes 311
5.17 Viewing Resource Information 311
5.17.1 Resource Information in Accounting Logs 312
5.17.2 Resource Information in Daemon Logs 312
5.17.3 Finding Current Value 313
5.17.4 Restrictions on Viewing Resources 313
5.18 Resource Recommendations and Caveats 313

5.2 Introduction to PBS Resources

PBS resources represent things such as CPUs, memory, application licenses, switches, scratch space, and time. They can also represent whether or not something is true, for example, whether a machine is dedicated to a particular project. PBS provides a set of built-in resources, and allows you to define additional custom resources. For some systems, PBS creates specific custom resources. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS can enforce limits on resource usage by jobs. The administrator can specify which resources are available at the server, each queue, and each vnode.

5.3 Glossary

Reservation

A reservation for a specific set of resources for a specified start time and duration in the future. See section 4.9.37, “Reservations”, on page 197.

Borrowing vnode

A shared vnode resource is available for use by jobs at more than one vnode, but is managed at just one vnode. A *borrowing vnode* is a vnode where a shared vnode resource is available, but not managed.

Built-in resource

A resource that is defined in PBS Professional as shipped. Examples of built-in resources are ncpus, which tracks the number of CPUs, and mem, which tracks memory. See section 5.4.1, "Built-in vs. Custom Resources", on page 233.

Chunk

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

Consumable resource

A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See section 5.4.3, "Consumable vs. Non-consumable Resources", on page 234.

CPU

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.
2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called "*ncpus*". The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

Custom resource

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See section 5.4.1, "Built-in vs. Custom Resources", on page 233 and section 5.14, "Custom Resources", on page 255.

Floating license

A unit of license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).

Generic group limit

A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic user limit

A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Global resource

A global resource is defined in a resources_available attribute, at the server, a queue, or a host. Global resources can be operated on via the `qmgr` command and are visible via the `qstat` and `pbsnodes` commands. See section 5.4.5, "Global vs. Local Resources", on page 235.

Group limit

Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

Indirect resource

A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Individual group limit

Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual user limit

Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Limit

A maximum that can be applied in various situations:

- The maximum number of jobs that can be queued
- The maximum number of jobs that can be running
- The maximum number of jobs that can be queued and running
- The maximum amount of a resource that can be allocated to queued jobs
- The maximum amount of a resource that can be consumed at any time by running jobs
- The maximum amount of a resource that can be allocated to queued and running jobs

Local resource

A local resource is defined in a Version 1 MoM configuration file. Local resources cannot be operated on via the `qmgr` command and are not visible via the `qstat` and `pbsnodes` commands. Local resources can be used by the scheduler. See section 5.4.5, “Global vs. Local Resources”, on page 235.

Managing vnode

The vnode where a shared vnode resource is defined, and which manages the resource.

Memory-only vnode

Represents a node board that has only memory resources (no CPUs).

Non-consumable resource

A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and walltime. See section 5.4.3, “Consumable vs. Non-consumable Resources”, on page 234.

Overall limit

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

Resource

A *resource* can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, licenses, or time, or it can be an arbitrary item defined for another purpose. PBS provides built-in resources, and allows custom-defined resources.

Shared resource

A vnode resource defined and managed at one vnode, but available for use at other vnodes.

User limit

Refers to configurable limits on resources and jobs. A user's limit, whether generic or individual.

5.4 Categories of Resources

A PBS resource has several defining characteristics describing where and how it is used, how it was defined, etc. Each characteristic puts it in one of a set of categories. A resource inhabits several categories at once; for example, a resource can be a custom global static consumable server resource. We describe the sets of categories below.

5.4.1 Built-in vs. Custom Resources

Built-in resources are the resources that are already defined for you in PBS. PBS supplies built-in resources including number of CPUs, CPU time, and memory. For a list of built-in resources, see "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide. Custom resources are those that you define, or that PBS creates for some systems. For example, if you wanted a resource to represent scratch space, you could define a resource called *Scratch*, and specify a script which queries for the amount of available scratch space. See section 5.14, "Custom Resources", on page 255.

5.4.2 Server vs. Queue vs. Vnode Resources

PBS resources can be available at the server, queues, both the server and queues, or at vnodes. Any of these resources can be static or dynamic, built-in or custom, and consumable or non-consumable. Vnode resources can additionally be global or local.

5.4.2.1 Server Resources

A server resource, also called a server-level resource, is a resource that is available at the server. A server resource is available to be consumed or matched at the server if you set the server's resources_available.<resource name> attribute to the available or matching value. For example, you can define a custom resource called *FloatingLicenses* and set the server's resources_available.FloatingLicenses attribute to the number of available floating licenses.

A server resource is a job-wide resource. This means that a job can request this resource for the entire job, but not for individual chunks.

An example of a job-wide resource is shared scratch space, or any custom resource that is defined at the server and queue level.

5.4.2.2 Queue Resources

A queue resource, also called a queue-level resource, is available to be consumed or matched by jobs in the queue if you set the queue's resources_available.<resource name> attribute to the available or matching value.

A queue resource is a job-wide resource. A job can request a queue resource for the entire job, but not for individual chunks.

An example of a job-wide resource is floating licenses, or any custom resource that is defined at both server and queue level.

5.4.2.3 Resources Defined at Both Server and Queue

Custom resources can be defined to be available either at vnodes or at both the server and queues. Consumable custom resources that are defined at the server and queue level have their consumption monitored at the server and queue level. In our example, if a job requests one FloatingLicenses, then the value of the resources_assigned.FloatingLicenses attribute is incremented by one at both the server and the queue in which the job resides.

5.4.2.4 Vnode Resources

A vnode resource, also called a vnode-level or host-level resource, is available only at vnodes. A vnode resource is a chunk-level resource, meaning that it can be requested for a job only inside of a chunk.

5.4.3 Consumable vs. Non-consumable Resources

A *consumable* resource is one that is reduced by being used. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the -n or -f flags.

A *non-consumable* resource is not reduced through use, meaning that allocation to one job does not affect allocation to other jobs. The scheduler matches jobs to non-consumable resources. Examples of non-consumable resources are walltime, file, cput, pcput, pmem, pvmem, nice, or Boolean resources.

The following table shows which resource types are consumable:

Table 5-1: Consumable and Non-consumable Resources

Resource Type	Consumable vs. Non-consumable
Boolean	Non-consumable
duration	Non-consumable
float	Consumable
long	Consumable
Size	Consumable
string	Non-consumable
string_array	Non-consumable

5.4.4 Static vs. Dynamic Resources

Static resources are managed by PBS and have values that are fixed until you change them or until you change the hardware and MoM reports a new value for memory or number of CPUs.

Dynamic resources are not under the control of PBS, meaning that they can change independently of PBS. Dynamic resources are reported via a script; PBS runs a query to discover the available amount. Server dynamic resources use a script that runs at the server host. Host-level (MoM) dynamic resources use a script that runs at the execution host.

Static and dynamic resources can be available at the server or host level.

The default timeout for a server dynamic resource script is 30 seconds. You can specify a timeout for server dynamic resources in each scheduler's server_dyn_res_alarm attribute. If the script does not finish before the timeout, the scheduler uses a value of zero for the dynamic server resource. If you set the timeout to zero, the scheduler does not place a time limit on the script.

5.4.4.1 Dynamic Resource Caveats

- Dynamic resource values are displayed in `qstat`, but the value displayed is the last value retrieved, not the current value. Dynamic resources have no resources_available.<resource name> representation anywhere in PBS.
- Dynamic resources can take longer to discover because PBS runs a script to determine each one.

5.4.5 Global vs. Local Resources

5.4.5.1 Global Static Resources

Global static resources are defined in resources_available attributes at the server, queue, or vnode, and are available at the server, queue, or vnode level. Global static resources can be operated on via the `qmgr` command and viewed via the `qstat` command. Values for built-in global static resources are set via the `qmgr` command. The walltime and aoe resources are examples of global static resources. For custom global static resources, see section 5.14.2.9, "Example of Defining Each Type of Custom Resource", on page 265.

5.4.5.2 Global Dynamic Resources

Global dynamic resources can be used at the server, queue, or vnode level. Global host-level dynamic resources can be viewed via the `qstat` command. Server dynamic resource values have no resources_available.<resource name> representation anywhere in PBS. See section 5.14.3.1, "Dynamic Server-level Resources", on page 267.

The value displayed via `qstat` for a dynamic resource is the most recently retrieved, not the current value.

5.4.5.3 Local Static Resources

It is not recommended to use local static resources. Local static resources are defined in the MoM Version 1 configuration file. These resources cannot be operated on via the `qmgr` command or viewed via the `qstat` command. They have no entry in the vnode's resources_available.<resource name> resources. They can be used by the scheduler.

5.4.5.4 Local Dynamic Resources

Dynamic local resources are defined in the MoM Version 1 configuration file. These are scripts that run on the execution host where they are defined and return a value. These resources can be used by the scheduler. Host dynamic resource values have no resources_available.<resource name> representation anywhere in PBS. See section 5.14.4.1, "Dynamic Host-level Resources", on page 269.

The value displayed via `qstat` for a dynamic resource is the most recently retrieved, not the current value.

5.4.6 Requested vs. Default Resources

A job's requested resources are the resources explicitly requested by the job. Default resources are resources that you specify that each job should have if not requested. For example, you can specify that any job that does not request walltime gets 12 hours of walltime. For jobs that do request walltime, the default of 12 hours is not applied.

For information on default resources, see section 5.9.3, "Specifying Job Default Resources", on page 245 and section 5.9.4, "Allocating Default Resources to Jobs", on page 247.

5.4.7 Shared vs. Non-shared Vnode Resources

5.4.7.1 Non-shared Vnode Resources

Most vnode resources are not shared. When a resource is defined at one vnode for use by jobs only at that vnode, the resource is not shared. For example, when resources_available.ncpus is set to *4* on a single-vnode machine, and no other vnodes have resources_available.ncpus defined as a pointer to this resource, this resource is not shared.

5.4.7.2 Shared Vnode Resources

When more than one vnode needs access to the same actual resource, that resource can be shared among those vnodes. The resource is defined at one vnode, and the other vnodes that supply the resource contain a pointer to that vnode. Any of the vnodes can supply that resource to a job, but only up to the amount where the total being used by jobs is less than or equal to the total available at the vnode where the resource is defined. For example, if you had a 4-vnode machine which had 8GB of memory, and wanted any single vnode to be able to supply up to 8GB to jobs, you would make the memory a shared resource. See section 5.14.4.3, "Shared Host-level Resources", on page 271.

5.4.8 Platform-specific vs. Generally Available Resources

Most PBS built-in resources are available on, and apply to, all supported platforms. However, PBS provides some resources specifically designed for a given platform. These platform-specific resources are not applicable to any other platform, and cannot be used on platforms other than the one(s) for which they are designed. For example, PBS creates custom resources that represent Cray XC elements, such as the Cray XC nid and the Cray XC label.

5.4.9 Job-wide vs. Chunk Resources

5.4.9.1 Job-wide Resources

A job-wide resource applies to the entire job, and is available at the server or queue, but not at the host level. Job-wide resources are requested outside of a select statement, using this form:

-l <resource name>=<value>

For example, to request one hour of walltime for a job:

-l walltime=1:00:00

Examples of job-wide resources are walltime, scratch space, and licenses.

5.4.9.2 Chunk Resources

A chunk resource applies to the part of the job running on that chunk, and is available at the host level. Chunk resources are requested inside a select statement. A single chunk is requested using this form:

-l select=<resource name>=<value>:<resource name>=<value>

For example, one chunk might have 2 CPUs and 4GB of memory:

```
-l select=ncpus=2:mem=4gb
```

To request multiples of a chunk, prefix the chunk specification by the number of chunks:

-l select=[number of chunks]<chunk specification>

For example, to request six of the previous chunk:

```
-l select=6:ncpus=2:mem=4gb
```

To request different chunks, concatenate the chunks using the plus sign ("+"):

-l select=[number of chunks]<chunk specification>+[number of chunks]<chunk specification>

For example, to request two kinds of chunks, one with 2 CPUs per chunk, and one with 8 CPUs per chunk, both kinds with 4GB of memory:

```
-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4GB
```

5.5 Resource Types

PBS supplies the following types of resources:

Boolean

Duration

Float

Long

Size

String

String Array

5.6 Resource Formats

Custom resources follow the same rules as built-in resources: custom resource names must be PBS NAMEs, allowable values for float and long resources are the same as for built-in resources, and custom Boolean, time, size, string or string array resources must have the same format as built-in resources.

Boolean

Name of Boolean resource is a string.

Values:

TRUE, True, true, T, t, Y, y, 1

FALSE, False, false, F, f, N, n, 0

Duration

A period of time, expressed either as

An integer whose units are seconds

or

[[hours:]minutes:]seconds[.milliseconds]

in the form:

[[HH:]MM:]SS[.milliseconds]

Milliseconds are rounded to the nearest second.

Float

Floating point. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

Long

Long integer. Allowable values: 0-9 [[0-9] ...], and + and -

<queue name>@<server name>

Size

Number of bytes or words. The size of a word is 64 bits.

Format: *<integer>[<suffix>]*

where *suffix* can be one of the following:

Table 5-2: Size in Bytes

Suffix	Meaning	Size
b or w	Bytes or words	1
kb or kw	Kilobytes or kilowords	2 to the 10th, or 1024
mb or mw	Megabytes or megawords	2 to the 20th, or 1,048,576
gb or gw	Gigabytes or gigawords	2 to the 30th, or 1,073,741,824
tb or tw	Terabytes or terawords	2 to the 40th, or 1024 gigabytes
pb or pw	Petabytes or petawords	2 to the 50th, or 1,048,576 gigabytes

Default: *bytes*

Note that a scheduler rounds all resources of type size up to the nearest kb.

String

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~] ...]

String resource values are case-sensitive. No limit on length.

String Array

Comma-separated list of strings.

Strings in string_array may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

5.6.1 Resource Names

Resource names are case-insensitive PBS NAMEs. Resource names can be 64 characters in length.

This is a generic term, used to describe various PBS entities. For example, attribute names are PBS NAMEs.

Must start with an alphabetic character, and may contain only the following: alpha-numeric, underscore ("_"), or dash ("-").

Do not use PBS keywords as PBS NAMEs.

5.7 Setting Values for Resources

5.7.1 How Resource Values are Set

PBS automatically collects information about some resources such as ncpus and mem and sets their initial values accordingly. If you explicitly set the value for a resource, that value is carried forth across server restarts.

Since the value for each dynamic resource is set by PBS to the value returned by a script or command, it makes sense set values for static resources only.

Resources that are not explicitly set can inherit their values from defaults. Jobs can inherit default resources; see section 5.9.4, “Allocating Default Resources to Jobs”, on page 247.

You set values for custom and built-in resources using the same methods. You can set resource values using the following methods:

- Using qmgr:

  To set the available amount of a non-string_array resource, use the qmgr command, either from the command line or within qmgr:

  ```
  qmgr -c "set <object> ressources_available.<resource name> = <value>"
  Qmgr: set <object> resources_available.<resource name> = <value>
  ```

  To set or change the available amount of a string_array resource, use the qmgr command, either from the command line or within qmgr:

  ```
  qmgr -c "set <object> resources_available.<resource name> = <value>"
  qmgr -c 'set <object> resources_available.<resource name> = "<value,value>"'
  qmgr -c 'set <object> resources_available.<resource name> += <value>'
  qmgr -c 'set <object> resources_available.<resource name> -= <value>'
  Qmgr: set <object> resources_available.<resource name> = <value>
  Qmgr: set <object> resources_available.<resource name> = '<value,value>'
  Qmgr: set <object> resources_available.<resource name> += <value>
  Qmgr: set <object> resources_available.<resource name> -= <value>
  ```

  To unset the value of an attribute:

  ```
  qmgr -c "unset <object> resources_available.<resource name>"
  Qmgr: unset <object> resources_available.<resource name>
  ```

  where <*object*> is one of *server*, *queue*, *hook*, *node*, or *sched*.

  For example, to set resources_max.walltime at the server to be 24 hours:

  ```
  Qmgr: set server resources_max.walltime = 24:00:00
  ```

  See “qmgr” on page 149.
- Using a Version 2 configuration file; see section 3.4.3, “Version 2 Vnode Configuration Files”, on page 42.
- Setting the value in a hook; see "Using Attributes and Resources in Hooks" on page 44 in the PBS Professional Hooks Guide.

5.7.1.1 How Vnode Available Resource Values are Set

PBS stores values for the global resources available at a vnode in that vnode's resources_available.<resource name> attribute.

5.7.1.1.i Vnode Resources Set by PBS

PBS automatically sets the value for certain resources available at each vnode, meaning that PBS sets the value for the vnode's resources_available.<resource name> attribute. For example, PBS automatically sets the value of resources_available.ncpus at each vnode. The following table lists the vnode resources that are set automatically by PBS.

Table 5-3: Resources Set by PBS

Resource Name	Initial Value	Notes
arch	*Value reported by OS*	Settable. If you unset the value, it remains unset until MoM is restarted.
host	*Short form of hostname in Mom vnode attribute*	Settable. If you unset the value, it remains unset until MoM is restarted.
mem	*Amount reported by OS*	Settable. If you unset the value, it remains unset until MoM is restarted.
ncpus	*Number of CPUs reported by OS*	Settable. If you unset this value, the MoM will reset it to the value reported by the OS.
PBScrayhost	*On CLE 3.0 and higher, set to value of mpp_host for this system*	Do not set.
PBScraylabel_<label name>	*Concatenation of PBScraylabel_ and label name. Set to True on all of node's vnodes .*	Do not set.
PBScraynid	*Value of node_id for this compute node*	Do not set.
PBScrayorder	Value starts at 1 and increments by 1 for each node in inventory	Do not set
router	*Name of router, from topology file*	Applies to vnodes on certain HPE systems only
vnode	*Name of the vnode*	Vnode name must be specified via the `qmgr create node` command.

5.7.1.1.ii Setting Vnode Resources Manually

You can set values for available vnode resources:

- You can set values for a vnode's resources_available in a hook. See "Setting and Unsetting Vnode Resources and Attributes" on page 48 in the PBS Professional Hooks Guide. If you set a vnode resource in a hook, MoM will no longer update the resource.
- You can set values for a vnode's resources_available attribute in a Version 2 configuration file; see section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.
- You can set most values for a vnode's resources_available attribute using `qmgr`, but not for resources_available.host; see "Operating on Attributes and Resources" on page 158 of the PBS Professional Reference Guide.

5.7.1.2 Setting Server and Queue Resource Values

You can set resources, such as default and available resources, for queues and for the server, using qmgr:

```
Qmgr: set queue <queue name> resources_default<resource name> = <value>
Qmgr: set queue <queue name> resources_available.<resource name> = <value>
Qmgr: set server resources_available.<resource name> = <value>
```

5.7.1.3 Setting Job Resources

5.7.1.3.i Setting Requested Resource Values

Job resources, stored in the Resource_List job attribute, can be set initially at submission in the job request. You can augment or change these values. You can set values for a job's Resource_List attribute using hooks. See "Setting Job Resources in Hooks" on page 49 in the PBS Professional Hooks Guide.

5.7.1.3.ii Setting Used Resource Values

The resources used by a job are set in the job's resources_used attribute

You can set values for a job's resources_used attribute using hooks. These values will appear in the accounting log and in qstat -f output. See "Setting Job Resources in Hooks" on page 49 in the PBS Professional Hooks Guide.

5.7.1.3.iii Setting Estimated Values

If the PBS_est built-in hook is enabled, PBS automatically sets the value of the estimated.start_time job resource to the estimated start time for each job. Otherwise, PBS sets the value only for top jobs.

5.7.2 Setting Values for Global Static Resources

To set the value for a global vnode, queue, or server resource, use the qmgr command to set the value of the appropriate resources_available.<resource name> attribute.

Example 5-1: Set the value of floatlicenses at the server to *10*:

```
Qmgr: set server resources_available.floatlicenses = 10
```

Example 5-2: Set the value of RunsMyApp to *True* at the vnode named *vnode1*:

```
Qmgr: set node vnode1 resources_available.RunsMyApp = True
```

5.7.2.1 Restrictions on Setting Values for Global Static Resources

When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.4.5, “Configuring Vnode Resources”, on page 47.

5.7.3 Setting Values for Local Static Resources

It is not recommended to use local static resources, because these resources cannot be requested, and cannot be viewed using qstat or managed using qmgr. To set the value of a parent vnode resource, edit PBS_HOME/mom_priv/config and change the value section of the resource's line.

5.7.4 Setting Values for String Arrays

A string array that is defined on vnodes can be set to a different set of strings on each vnode.

Example of defining and setting a string array:

- Define a new resource:
  Qmgr: create resource foo_arr type=string_array, flag=h
- Setting via qmgr:
  Qmgr: set node n4 resources_available.foo_arr="f1, f3, f5"
- Vnode n4 has 3 values of foo_arr: f1, f3, and f5. We add f7:
  Qmgr: set node n4 resources_available.foo_arr+=f7
- Vnode n4 now has 4 values of foo_arr: f1, f3, f5 and f7.
- We remove f1:
  Qmgr: set node n4 resources_available.foo_arr-=f1
- Vnode n4 now has 3 values of foo_arr: f3, f5, and f7.
- Submission:
  qsub -l select=1:ncpus=1:foo_arr=f3

5.7.5 When Resource Changes Take Effect

If you change the value of a resource via the qmgr command, the change takes effect immediately.

If you change the value of a resource in a configuration file, the change takes effect the next time the configuration file is read.

5.7.6 Caveats for Setting Resource Values

- It is not recommended to set the value for resources_available.ncpus. The exception is when you want to oversubscribe CPUs. See section 9.6.5.1.iii, "How To Share CPUs", on page 446.
- Do not attempt to set values for resources_available.<resource name> for dynamic resources.
- Do not set values for any resources, except those such as shared scratch space or floating licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set resources_available.walltime at the server to *10:00:00*, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.

5.7.6.1 Caveats for Setting Resource Values at Multi-vnode Machines

- When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.4.5, "Configuring Vnode Resources", on page 47.
- It is not recommended to change the value of ncpus at vnodes on a multi-vnoded machine.
- On the parent vnode, all values for resources_available.<resource name> should be *zero* (*0*), unless the resource is being shared among other vnodes via indirection.

5.8 Overview of Ways Resources Are Used

Resources are used in several ways in PBS. The following table lists the ways resources are used, and gives links to the section describing each one:

Table 5-4: How Resources Are Used

Use	Description
Allocation to and use by jobs	See section 5.9, "Resources Allocated to Jobs and Reservations", on page 244
Limiting job resource usage	See section 5.15.2, "Placing Resource Limits on Jobs", on page 304
Restricting access to server and queues	See section 5.13, "Using Resources to Restrict Server or Queue Access", on page 254
Routing jobs	See section 2.3.6.4, "Using Resources to Route Jobs Between Queues", on page 26
Describing topology and placing jobs	See section 5.11, "Using Resources for Topology and Job Placement", on page 253
Setting job execution priority	See section 5.12, "Using Resources to Prioritize Jobs", on page 254
Reserving resources ahead of time	See section 4.9.37, "Reservations", on page 197.
Tracking and controlling allocation	See section 5.10, "Using Resources to Track and Control Allocation", on page 252
Determining job preemption priority	See section 4.9.33, "Using Preemption", on page 180

5.8.1 How the Scheduler Uses Resources

How the scheduler uses resources is described in section 4.9.28, "Matching Jobs to Resources", on page 159.

5.8.2 Advice on Using string and string_array Resources

Resource names are case-insensitive.

5.8.2.1 Using string Resources

Format of string resources:

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~] ...]

String resource values are case-sensitive. No limit on length.

Non-consumable.

We do not recommend using non-printing characters.

When using `qsub -l <string resource>=<string value>`, you must escape string values for both `qsub` and the shell. Example:

```
qsub -lteststring='\"abc def\"'
```

The final quote should be single, not double.

5.8.2.2 Using string_array Resources

Format of string_array resources:

Comma-separated list of strings.

Strings in string_array may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

Non-consumable. Resource request will succeed if request matches one of the values. Resource request can contain only one string.

The value of resources_default.<string array resource> can only be one string.

5.9 Resources Allocated to Jobs and Reservations

Resources allocated to jobs provide the job with amounts of CPUs and memory to be consumed by the job's processes, as well as qualities such as architecture and host. The resources allocated to a job are those that the job requests and those that are assigned to it through resource defaults that you define, or by hooks you write.

Jobs use resources at the job-wide and chunk level. Job-wide resources such as walltime or vmem are applied to and requested by the job as a whole. Chunk-level resources, such as ncpus, are applied and requested in individual chunks.

Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide and "Requesting Resources", on page 51 of the PBS Professional User's Guide.

Jobs inherit resource defaults for resources not explicitly requested. See section 5.9.4, "Allocating Default Resources to Jobs", on page 247.

Chunk-level resources are made available at the host (vnode) level by defining them via resources_available.<resource name> at the vnode, and are requested using `-l select=<resource name>=<value>`.

Job-wide resources are made available by defining them via resources_available.<resource name> at the queue or server. These resources are requested using `-l <resource name> =<value>`.

The scheduler matches requested resources with available resources, according to rules defined by the administrator.

When a job is requesting a string array resource, it can request only one of the values set in the string array resource. The job will only be placed on a vnode where the job's requested string matches one of the values of the string array resource. For example, if the resource named *Colors* is set to "*red, blue, green*" on vnode V1, and "*red, blue*" on V2:

- A job can request only one of "*red*", "*blue*", or "*green*"
- A job requesting `Colors=green` will only be placed on V1

5.9.1 Allocating Chunks

Chunks cannot be split across hosts. Chunks can be made up of vchunks. If a chunk is broken up over multiple vnodes, all participating vnodes must belong to the same execution host. Each vnode supplies a vchunk. These participating vnodes are supplying the vchunks that make up the chunk. A chunk defines a logical set of resources, for example, those needed for an MPI task. The resources must come from a single host, but if the requested resources exceed that of any one vnode, the physical resources can be taken from multiple vnodes on the same host.

5.9.2 Resources Requested by Job

The job's Resource_List attribute lists the following resources requested by the job:

- Job-wide resources explicitly requested by the job, inherited from defaults, or assigned by hooks
- The following built-in chunk-level resources explicitly requested by the job, inherited from defaults, or assigned by hooks:

  mpiprocs

  ncpus

  mem

  vmem
- Custom vnode-level (chunk-level) resources that are global and have the n, q, or f flags set, explicitly requested by the job, inherited from defaults, or assigned by hooks

5.9.3 Specifying Job Default Resources

You can specify which resources are automatically added to job resource requests. When a job does not request a specific resource, the default value for that resource is automatically added to the job's resource request.

You can also use hooks to add resources to a job's resource request, but we describe that elsewhere in the PBS Professional Hooks Guide.

The amount of each resource a job is allowed to use is the amount in its resource request. See section 5.15.2, "Placing Resource Limits on Jobs", on page 304. Therefore you may wish to add default limits on resource usage. This is done by adding default resources to the job's resource request. For example, if a job does not request walltime, but you do not want jobs not specifying walltime to run for more than 12 hours, you can specify a default of 12 hours for walltime. Jobs that do specify walltime do not inherit this default; they keep their requested amount.

You can use default resources to manage jobs. For example, if you want to keep track of and limit the number of jobs using something such as a disk arm, you can have each job using the disk arm automatically request one counting resource. Then you can place a limit on the amount of this resource that can be in use at one time. This technique is described in section 5.10, "Using Resources to Track and Control Allocation", on page 252.

Default resources can be defined for the server and for each queue. Default resources defined at the server are applied to all jobs. Default resources at a queue are applied only to the jobs that are in that queue.

Default resources on the server and queue can be job-wide, which is the same as adding `-l <resource name>` to the job's resource request, or they can be chunk resources, which is the same as adding `:<resource name>=<value>` to a chunk.

Job-wide resources are specified via resources_default on the server or queue, and chunk resources are specified via default_chunk on the server or queue. You can also specify default resources to be added to any `qsub` arguments. In addition, you can specify default placement of jobs.

5.9.3.1 Specifying Job-wide Default Resources at Server

To specify a server-level job-wide default resource, use the `qmgr` command to set the server's resources_default attribute:

Qmgr: set server resources_default.<resource name>=<value>

For example, to set the default architecture on the server:

```
Qmgr: set server resources_default.arch=linux
```

5.9.3.2 Setting Server and Queue Default Job Chunk Resource Values

If a job doesn't request a specific resource, PBS can assign a default value you specify. PBS stores default values for job chunk resources in the default_chunk.<resource name> attribute for the server and each queue.

PBS automatically sets the value for default_chunk.ncpus to *1* at the server and queues.

5.9.3.2.i Specifying Chunk Default Resources at Server

To specify a server-level chunk default resource, use the `qmgr` command to set the server's default_chunk attribute:

Qmgr: set server default_chunk.<resource name>=<value>

For example, if you want all chunks that don't specify ncpus or mem to inherit the values you specify:

```
Qmgr: set server  default_chunk.ncpus=1
Qmgr: set server  default_chunk.mem=1gb
```

5.9.3.2.ii Specifying Chunk Default Resources at Queue

To specify a queue-level chunk default resource, use the `qmgr` command to set the queue's default _chunk attribute:

Qmgr: set queue <queue name> default_chunk.<resource name>=<value>

For example, if you want all chunks that don't specify ncpus or mem to inherit the values you specify:

```
Qmgr: set queue small default_chunk.ncpus=1
Qmgr: set queue small default_chunk.mem=512mb
```

5.9.3.3 Specifying Job-wide Default Resources at Queue

To specify a default for a job-wide resource at a queue, use the `qmgr` command to set the queue's resources_default attribute:

Qmgr: set queue <queue name> resources_default.<resource name> = <value>

5.9.3.4 Specifying Default `qsub` Arguments

You can set defaults for any `qsub` arguments not explicitly requested by each job. You do this at the server by using the `qmgr` command to set the server's default_qsub_arguments attribute:

Qmgr: set server default_qsub_arguments=<string containing arguments>

For example, to set the default for the Rerunable job attribute in each job's resource request, and the name of the job:

```
Qmgr: set server default_qsub_arguments= "-r y -N MyJob"
```

Or to set a default Boolean in each job's resource request so that jobs don't run on *Red* unless they explicitly ask to do so:

```
Qmgr: set server default_qsub_arguments="-l Red=False"
```

5.9.3.5 Specifying Default Job Placement

You can specify job placement defaults at both the server and queue level. You use the `qmgr` command to set the resources_default.place attribute at the server or queue:

Qmgr: set queue <queue name> resources_default.place=<value>

For example, to set the default job placement for a queue:

```
Qmgr: set queue Q1 resources_default.place=free
```

When setting default placement involving a colon, enclose the value in double quotes:

```
Qmgr: set server resources_default.place="<value>"
```

For example, to set default placement at the server to *pack:shared*, do the following:

```
Qmgr: set server resources_default.place= "pack:shared"
```

See "Specifying Job Placement", on page 64 of the PBS Professional User's Guide for detailed information about how -l `place` is used.

5.9.3.6 Using Gating Values As Defaults

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of resources_max.<resource name> is used. If this is set only at the server, the job inherits the value set at the server.

5.9.3.7 Default Resource Caveats

- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue resources_default.<resource name> attribute.
- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue resources_default or default_chunk as an input argument.
- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as *excl*, with `-lplace=excl`, and the user specifies an arrangement, such as *pack*, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for select; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

5.9.4 Allocating Default Resources to Jobs

Jobs inherit default resources, job-wide or per-chunk, with the following order of precedence.

Table 5-5: Order In Which Default Resources Are Assigned to Jobs

Order of assignment	Default value	Affects Chunks?	Job-wide?
1	Default `qsub` arguments	If specified	If specified
2	Queue's default_chunk	Yes	No
3	Server's default_chunk	Yes	No
4	Queue's resources_default	No	Yes
5	Server's resources_default	No	Yes
6	Queue's resources_max	No	Yes
7	Server's resources_max	No	Yes

See section 5.9.3, "Specifying Job Default Resources", on page 245 for how to set these defaults.

For each chunk in the job's selection statement, first default `qsub` arguments are applied, then queue chunk defaults are applied, then server chunk defaults are applied. If the chunk does not contain a resource defined in the defaults, the default is added. The chunk defaults are specified in the default_chunk.<resource name> server or queue attribute.

For example, if the queue in which the job is enqueued has the following defaults defined,

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

then a job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

will have this specification after the default_chunk elements are applied:

```
select=2:ncpus=4:mem=2gb+1:ncpus=1:mem=9gb
```

In the above, `mem=2gb` and `ncpus=1` are inherited from default_chunk.

The job-wide resource request is checked against queue resource defaults, then against server resource defaults, then against the queue's resources_max.<resource name>, then against the server's resources_max.<resource name>. If a default or maximum resource is defined which is not specified in the resource request, it is added to the resource request.

5.9.4.1 Default Resource Allocation for min_walltime and max_walltime

The min_walltime and max_walltime resources inherit values differently. A job can inherit a value for max_walltime from resources_max.walltime; the same is not true for min_walltime. This is because once a job is shrink-to-fit, PBS can use a walltime limit for max_walltime. See section 4.9.42.3.ii, “Inheriting Values for min_walltime and max_walltime”, on page 212.

5.9.4.2 Default Resource Allocation Caveats

- Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default_qsub_arguments. Be aware that creating custom resources with permission flags and then using these in the default_qsub_arguments server attribute can cause jobs to be rejected. See section 5.14.2.3.vi, “Resource Permission Flags”, on page 260.
- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as *excl*, with `-lplace=excl`, and the user specifies an arrangement, such as *pack*, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for select; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

5.9.4.3 Moving Jobs Between Queues or Servers Changes Defaults

If the job is moved from the current queue to a new queue or server, any default resources in the job's Resource_List inherited from the current queue or server are removed. The job then inherits any new default resources. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Jobs may be moved between servers when peer scheduling is in operation. Given the following set of queue and server default values:

- Server

  resources_default.ncpus=1
- Queue QA

resources_default.ncpus=2

default_chunk.mem=2GB

- Queue QB

  default_chunk.mem=1GB

  no default for ncpus

The following illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

Example 5-3: Submission:

```
qsub -l ncpus=1 -lmem=4gb
```

- In QA:

  select=1:ncpus=1:mem=4gb

  - No defaults need be applied

- In QB:

  select=1:ncpus=1:mem=4gb

  - No defaults need be applied

Example 5-4: Submission:

```
qsub -l ncpus=1
```

- In QA:

  select=1:ncpus=1:mem=2gb

  - Picks up 2GB from queue default chunk and 1 ncpus from qsub

- In QB:

  select=1:ncpus=1:mem=1gb

  - Picks up 1GB from queue default_chunk and 1 ncpus from qsub

Example 5-5: Submission:

```
qsub -lmem=4gb
```

- In QA:

  select=1:ncpus=2:mem=4gb

  - Picks up 2 ncpus from queue level job-wide resource default and 4GB mem from qsub

- In QB:

  select=1:ncpus=1:mem=4gb

  - Picks up 1 ncpus from server level job-wide default and 4GB mem from qsub

Example 5-6: Submission:

```
qsub -lnodes=4
```

- In QA:

  select=4:ncpus=1:mem=2gb

  - Picks up a queue level default memory chunk of 2GB. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

- In QB:

  select=4:ncpus=1:mem=1gb

(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

Example 5-7: Submission:

```
qsub -l mem=16gb -lnodes=4
```

- In QA:

  ```
  select=4:ncpus=1:mem=4gb
  ```

  (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

- In QB:

  ```
  select=4:ncpus=1:mem=4gb
  ```

  (In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

5.9.5 Dynamic Resource Allocation Caveats

When a job requests a dynamic resource, PBS checks to see how much of the resource is available, but cannot know how much will be used by another job while this job executes. This can lead to a resource shortage. For example, there is 20GB of scratch on a disk, no jobs are running, and a job requests 15GB. This job writes to 5GB during the first part of its execution, then another job requests 10GB. The second job is started by PBS, because there is 15GB available. Now there is a shortage of scratch space.

You can avoid this problem by configuring a static consumable resource to represent scratch space. Set it to the amount of available scratch space. See section 5.14.5.3, “Static Server-level Scratch Space”, on page 274 and section 5.14.5.4, “Static Host-level Scratch Space”, on page 274.

5.9.6 Period When Resource is Used by Job

5.9.6.1 Exiting Job Keeps Resource

A job that is exiting is still consuming resources assigned to it. Those resources are available for other jobs only when the job is finished.

5.9.6.2 Job Suspension and Resource Usage

5.9.6.2.i Resource Usage on Suspension

When suspended, a job is not executing and is not charged for walltime.

5.9.6.2.ii Releasing Resources on Suspension

You can specify which consumable resources should be released by PBS when a job is suspended, using the restrict_res_to_release_on_suspend server attribute. In this attribute you list all of the resources that should be released when a job is suspended. If you leave this attribute unset, PBS releases all of a job's consumable resources when the job is suspended. This does not include the licenses used by the application, if any. You can modify the list to add and remove resources using "+=" and "-=" operators.

Server attribute where you specify resources to be released:

restrict_res_to_release_on_suspend

Comma-separated list of consumable resources to be released when jobs are suspended. If unset, all consumable resources are released on suspension.

Format: *Comma-separated list*

Python type: *list*

Default value: unset, meaning all consumable resources are released on suspension

You can see which resources have been released for a given job by looking at these job attributes:

resources_released

Listed by vnode, consumable resources that were released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server.

Format: String of the form: *(<vnode>:<resource name>=<value>:<resource name>=<value>:...)+(<vnode>:<resource name>=<value>:...)*

Python type: *str*

resource_released_list

Sum of each consumable resource requested by the job that was released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server. You will also see this amount released at the queue and/or server.

Format: String of the form: *resource_released_list.<resource name>=<value>,resource_released_list.<resource name>=<value>, ...*

Python type: *pbs.pbs_resource*

Jobs are suspended when they are preempted and via `qsig -s suspend`.

A job is resumed only when sufficient resources are available. When a person resumes a job via `qsig -s resume`, the job is not run until resources are available.

5.9.6.2.iii Releasing Resources on Suspension on Cray XC

On Cray XC, when a job is suspended, PBS releases ncpus, but all other resources remain assigned.

5.9.6.2.iv Suspension/resumption Resource Caveats

Dynamic resources can cause problems with suspension and resumption of jobs.

When a job is suspended, its resources are freed, but the scratch space written to by the job is not available.

A job that uses scratch space may not suspend and resume correctly. This is because if the job writes to scratch, and is then suspended, when PBS queries for available scratch to resume the job, the script may return a value too small for the job's request. PBS cannot determine whether the job itself is the user of the scratch space; PBS can only determine how much is still unused. If a single suspended job has left less scratch space available than it requests, that job cannot be resumed.

The above is true for any dynamic resource, such as application licenses.

When suspended, a job is not executing and is not charged for walltime.

5.9.6.3 Shrink-to-fit Jobs Get walltime When Executed

PBS computes the walltime value for each shrink-to-fit job when the scheduler runs the job, not before. See section 4.9.42.3.iii, “Setting walltime for Shrink-to-fit Jobs”, on page 212.

5.10 Using Resources to Track and Control Allocation

You can use resources to track and control usage of things like hardware and application licenses. For example, you might want to limit the number of jobs using floating licenses or a particular vnode. There is more than one way to accomplish this.

Example 5-8: You can set a complex-wide limit on the number of jobs using a type of complex-wide floating application license. This example uses a single queue for the entire complex. This method requires job submitters to request one of a floatlicensecount resource in order to be able to use the license. To set a complex-wide limit, take the following steps:

1. Create a custom static integer license resource that will be tracked at the server and queue:

   a. Use `qmgr` to create the resource:

   ```
   Qmgr: create resource floatlicensecount   type=long, flag=q
   ```

   b. Add the resource to the `resources:` line in `<sched_priv directory>/sched_config`:

   ```
   resources: "[...], floatlicensecount"
   ```

2. HUP the scheduler:

   ```
   kill -HUP <scheduler PID>
   ```

3. Set the available resource at the server using `qmgr`. If you have enough floating licenses for 4 jobs:

   ```
   Qmgr: set server resources_available.floatlicensecount = 4
   ```

4. Inform job submitters that jobs using they must request one job-wide floatlicensecount resource via the following:

   ```
   qsub -l floatlicensecount=1
   ```

   The scheduler will schedule up to 4 jobs at a time using the licenses. You do not need to set the resource at any queue.

Example 5-9: Here, your job submitters don't need to request a counting resource. Jobs are routed based on the size of the request for memory, and the counting resource is inherited from a default. In this example, we are limiting the number of jobs from each group that can use a particular vnode that has a lot of memory. This vnode is called *MemNode*.

Jobs that request 8GB or more of memory are routed into queue *BigMem*, and inherit a default counting resource called *memcount*. All other jobs are routed into queue *SmallMem*. The routing queue is called *RouteQueue*.

1. Create a custom static integer memcount resource that will be tracked at the server and queue:

   a. Use `qmgr` to create the resource:

   ```
   Qmgr: create resource memcount   type=long, flag=q
   ```

   b. Add the resource to the resources: line in `<sched_priv directory>/sched_config`:

   ```
   resources: "[...], memcount"
   ```

2. HUP the scheduler:

   ```
   kill -HUP <scheduler PID>
   ```

3. Set limits at *BigMem* and *SmallMem* so that they accept the correct jobs:

   ```
   Qmgr: set queue BigMem resources_min.mem = 8gb
   Qmgr: set queue SmallMem resources_max.mem = 8gb
   ```

4. Set the order of the destinations in the routing queue so that *BigMem* is tested first, so that jobs requesting exactly 8GB go into *BigMem*:

   ```
   Qmgr: set queue RouteQueue route_destinations = "BigMem, SmallMem"
   ```

5. Set the available resource at *BigMem* using `qmgr`. If you want a maximum of 6 jobs from *BigMem* to use *MemNode*:

   ```
   Qmgr: set queue BigMem resources_available.memcount = 6
   ```

6. Set the default value for the counting resource at *BigMem*, so that jobs inherit the value:

   ```
   Qmgr: set queue BigMem resources_default.memcount = 1
   ```

7. Associate the vnode with large memory with the *BigMem* queue. See section 4.9.2, “Associating Vnodes with Queues”, on page 103.

   The scheduler will only schedule up to 6 jobs from *BigMem* at a time on the vnode with large memory.

5.11 Using Resources for Topology and Job Placement

Using the topology information in the server's node_group_key attribute, PBS examines the values of resources at vnodes, and uses those values to create placement sets. Jobs are assigned to placement sets according to their resource requests. Users can specify particular placement sets by requesting the resources that define that particular placement set. For example, if the switch named *A25* connects the desired set of vnodes, a user can request the following:

```
-l switch=A25
```

See section 4.9.32, “Placement Sets”, on page 168.

5.11.1 Restrictions on Using Resources for Job Placement

Only vnode-level resources can be used to direct jobs to particular vnodes.

5.12 Using Resources to Prioritize Jobs

You can define the formula the scheduler uses to compute job execution priorities. Elements in this formula can be inherited default custom resources. These resources must be job-wide numeric resources, or consumable host-level resources. See section 5.9.3, "Specifying Job Default Resources", on page 245 and section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.

You can make jobs inherit numeric resources according to non-numeric qualities, such as the job owner's group or whether the job requests a Boolean or string resource. You can do this by either of the following methods:

- Use a hook to identify the jobs you want and alter their resource requests to include the custom resources for the formula. See the PBS Professional Hooks Guide
- Use a routing queue and minimum and maximum resource limits to route jobs to queues where they inherit the default custom resources for the formula. See section 2.3.6.4, "Using Resources to Route Jobs Between Queues", on page 26

For details on how job execution priority is calculated, see section 4.9.16, "Calculating Job Execution Priority", on page 134.

For a complete description of how PBS prioritizes jobs, see section 4.3.5, "Job Prioritization and Preemption", on page 64.

5.13 Using Resources to Restrict Server or Queue Access

You can set resource limits at the server and queues so that jobs must conform to the limits in order to be admitted. This way, you can reject jobs that request more of a resource than the complex or a queue can supply. You can also force jobs into specific queues where they will inherit the desired values for unrequested or custom resources. You can then use these resources to manage jobs, for example by using them in the job sorting formula or to route jobs to particular vnodes.

You set a maximum for each resource at the server using the resources_max.<resource name> server attribute; there is no resources_min.<resource name> at the server.

You can set a minimum and a maximum for each resource at each queue using the resources_min.<resource name> and resources_max.<resource name> queue attributes.

Job resource requests are compared to resource limits the same way, whether at the server or a queue. For a complete description of how jobs are tested against limits, see section 2.3.6.4.i, "How Queue and Server Limits Are Applied, Except Running Time", on page 26.

Job resource requests are compared first to queue admittance limits. If there is no queue admittance limit for a particular resource, the job's resource request is compared to the server's admittance limit.

5.13.1 Admittance Limits for walltime, min_walltime, and max_walltime

Because min_walltime and max_walltime are themselves limits, they behave differently from other time-based resources. When a shrink-to-fit job (a job with a value for min_walltime) is compared to server or queue limits, the following must be true in order for the job to be accepted:

- Both min_walltime and max_walltime must be greater than or equal to resources_min.walltime.
- Both min_walltime and max_walltime must be less than or equal to resources_max.walltime.

You cannot set resources_min or resources_max for min_walltime or max_walltime.

5.13.2 Restrictions on Resources Used for Admittance

For a list of resources that are compared to admittance limits, see section 2.3.6.4.iii, "Resources Used for Routing and Admittance", on page 27. For information on using strings, string arrays, and Booleans for admittance controls, see section 2.3.6.4.iv, "Using String, String Array, and Boolean Values for Routing and Admittance", on page 27.

5.14 Custom Resources

You can define, that is, create, new resources within PBS. This section describes how to define and use custom resources.

Once new resources are defined, jobs may request these new resources and the scheduler can schedule on the new resources.

Using this feature, it is possible to schedule resources where the number or amount available is outside of PBS's control.

Custom resources can be made invisible to users or unalterable by users via resource permission flags. See section 5.14.2.3.vi, "Resource Permission Flags", on page 260. A user will not be able to print or list custom resources which have been made either invisible or unalterable.

PBS provides certain custom resources that are designed to reflect resources or properties found on specific systems. Do not create custom resources with the names that PBS uses for these resources. See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide.

5.14.1 How to Use Custom Resources

Custom resources can be static or dynamic, server-level or host-level, and local or global. They can also be shared or not.

5.14.1.1 Choosing the Resource Category

Use dynamic resources for quantities that PBS does not control, such as externally-managed application licenses or scratch space. PBS runs a script or program that queries an external source for the amount of the resource available and returns the value via `stdout`. Use static resources for things PBS does control. PBS tracks these resources internally.

Use server-level resources for things that are not tied to specific hosts, that is, they can be available to any of a set of hosts. An example of this is a floating application license. Use host-level resources for things that are tied to specific hosts, like the scratch space on a machine or node-locked application licenses.

5.14.1.1.i Examples of Configuring a Custom Resource

The following table gives examples of configuring each kind of custom resource:

Table 5-6: Examples of Configuring Custom Resources

Use for Resource	Link to Example
License: Floating, externally-managed	See section 5.14.6.3.i, "Example of Floating, Externally-managed License", on page 275
License: Floating, externally-managed with features	See section 5.14.6.3.ii, "Example of Floating, Externally-managed License with Features", on page 276
License: Floating, PBS-managed	See section 5.14.6.3.iii, "Example of Floating License Managed by PBS", on page 277

Table 5-6: Examples of Configuring Custom Resources

Use for Resource	Link to Example
License: Node-locked, per-host	See section 5.14.6.4.iv, "Example of Per-host Node-locked Licensing", on page 279
License: Node-locked, per-CPU	See section 5.14.6.4.vi, "Example of Per-CPU Node-locked Licensing", on page 281
License: Node-locked, per-use	See section 5.14.6.4.v, "Example of Per-use Node-locked Licensing", on page 280
FPGAs	See section 5.14.8, "Using FPGAs", on page 286
GPUs	See section 5.14.7, "Using GPUs", on page 283
Scratch space: shared	See section 5.14.5.1, "Dynamic Server-level (Shared) Scratch Space", on page 273 and section 5.14.5.3, "Static Server-level Scratch Space", on page 274
Scratch space: local to a host	See section 5.14.5.2, "Dynamic Host-level Scratch Space", on page 274 and section 5.14.5.4, "Static Host-level Scratch Space", on page 274
Generic dynamic server-level	See section 5.14.3.1.i, "Example of Configuring Dynamic Server-level Resource", on page 267
Generic static server-level	See section 5.14.3.2.i, "Example of Configuring Static Server-level Resource", on page 268
Generic dynamic host-level	See section 5.14.4.1.i, "Example of Configuring Dynamic Host-level Resource", on page 269
Generic static host-level	See section 5.14.4.2.i, "Example of Configuring Static Host-level Resource", on page 270
Generic shared static host-level	See section 5.14.4.3.v, "Configuring Shared Static Resources", on page 271

5.14.1.2 Dynamic Custom Resources

A dynamic resource is one which is not under the control of PBS, meaning it can change independently of PBS. In order to use a dynamic resource, PBS must run a query to discover the available amount of that resource. Dynamic custom resources can be defined at the server or vnodes.

5.14.1.2.i Dynamic Server-level Custom Resources

A dynamic server-level custom resource is used to track a resource that is available at the server. You use a dynamic server-level resource to track something that is not under the control of PBS, and changes outside of PBS, for example, floating application licenses. At each scheduler cycle, the scheduler runs a script at the server host to determine the available amount of that resource. Server-level custom resources are used as job-wide resources.

5.14.1.2.ii Dynamic Host-level Custom Resources

A dynamic host-level custom resource is used to track a resource that is available at the execution host or hosts. You use a dynamic host-level resource for a resource that is not under the control of PBS, and changes outside of PBS, for example, scratch space. At each scheduler cycle, the scheduler queries the MoM for the available amount of the resource. The MoM runs a script which returns the current value of the resource. Host-level dynamic resources are used inside chunks.

5.14.1.3 Static Custom Resources

A static resource is one which is under the control of PBS. Any changes to the value are performed by PBS or by the administrator. Static custom resources are defined ahead of time, at the server, queues or vnodes. Static custom resources can be local or global.

5.14.1.3.i Global Static Custom Resources

Global static custom resource values at vnode, queue and server are set via `qmgr`, by setting resources_available.<custom resource name> = <value>. These resources are available at the server, queues, or vnodes.

5.14.1.3.ii Local Static Custom Resources

Local static custom resources are defined in `PBS_HOME/mom_priv/config`, and are available only at the host where they are defined. Note that these resources cannot be set via `qmgr` or viewed via `qstat`. It is not recommended to use local static custom resources.

5.14.1.4 Shared Vnode Resources

A shared vnode resource is managed at one vnode, but available to be used by jobs at others. This allows flexible allocation of the resource. See section 5.14.4.3, “Shared Host-level Resources”, on page 271 for information on resources shared across vnodes.

5.14.1.5 Using Custom Resources for Application Licenses

The following table lists application licenses and what kind of custom resource to define for them. See section 5.14.6, “Supplying Application Licenses”, on page 274 for specific instructions on configuring each type of license and examples of configuring custom resources for application licenses.

Table 5-7: Custom Resources for Application Licenses

Floating or Node-locked	Unit Being Licensed	How License is Managed	Level	Resource Type
Floating (site-wide)	Token	External license manager	Server	Dynamic
Floating (site-wide)	Token	PBS	Server	Static
Node-locked	Host	PBS	Host	Static
Node-locked	CPU	PBS	Host	Static
Node-locked	Instance of Application	PBS	Host	Static

5.14.1.6 Using Custom Resources for Scratch Space

You can configure a custom resource to report how much scratch space is available on machines. Jobs requiring scratch space can then be scheduled onto machines which have enough. This requires dynamic host-level resources. See section 5.14.5, “Using Scratch Space”, on page 273 and section 5.14.4.1, “Dynamic Host-level Resources”, on page 269.

5.14.2 Defining New Custom Resources

You can define new custom resources as follows:

- To define any custom resource, you can use `qmgr`.
- To define custom host-level non-consumable resources at vnodes, you can use hooks; see "Adding Custom Non-consumable Host-level Resources" on page 64 in the PBS Professional Hooks Guide.

5.14.2.1 Defining and Setting Static and Dynamic Custom Resources

The following table lists the differences in defining and setting static and dynamic custom resources at the server, queue and host level.

Table 5-8: Defining and Setting New Custom Resources

Resource Type	Server-level	Queue-level	Host-level
static	Set via `qmgr`	Set via `qmgr`	Set via `qmgr` or hook
dynamic	Add to `server_dyn_res` line in `<sched_priv directory>/sched_config`	Cannot be used.	Add to MoM config file `PBS_HOME/mom_priv/config` and `mom_resources` line (**deprecated** as of 18.2.1) in `<sched_priv directory>/sched_config`

5.14.2.2 Custom Resource Values

The rules for custom resource values are the same as for built-in resource values. See "Resource Formats" on page 359 of the PBS Professional Reference Guide.

If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want. If the string resource value contains commas, the string must be enclosed in an additional set of quotes so that the command (e.g. `qsub`, `qalter`) will parse it correctly. If the string resource value contains quotes, plus signs, equal signs, colons or parentheses, the string resource value must be enclosed in yet another set of additional quotes.

5.14.2.3 Resource Flags

5.14.2.3.i Resource Accumulation Flags

When you define a custom resource, you can specify whether it is server-level or host-level, and whether it is consumable or not. This is done by setting resource accumulation flags via `qmgr`. A consumable resource is tracked, or accumulated, in the server, queue or vnode resources_assigned attribute. The resource accumulation flags determine where the value of resources_assigned.<resource name> is incremented.

5.14.2.3.ii Allowable Values for Resource Accumulation Flags

The value of <*resource flags*>, which is the resource accumulation flag for a resource can be one of the following:

Table 5-9: Resource Accumulation Flags

Flag	Meaning
(no flags)	Indicates a queue-level or server-level resource that is not consumable.
fh	The amount is consumable at the host level for only the first vnode allocated to the job (vnode with first task.) Must be consumable or time-based. Cannot be used with Boolean or string resources. . This flag specifies that the resource is accumulated at the first vnode, meaning that the value of resources_assigned.<resource> is incremented only at the first vnode when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.
h	Indicates a host-level resource. Used alone, means that the resource is not consumable. Required for any resource that will be used inside a select statement. This flag selects hardware. This flag indicates that the resource must be requested inside of a select statement. Example: for a Boolean resource named "green": `Qmgr: create resource green type=boolean, flag=h`
nh	The amount is consumable at the host level, for all vnodes assigned to the job. Must be consumable or time-based. Cannot be used with Boolean or string resources. This flag specifies that the resource is accumulated at the vnode level, meaning that the value of resources_assigned.<resource> is incremented at relevant vnodes when a job is allocated this resource or when a reservation requesting this resource on this vnode starts. This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.
q	The amount is consumable at the queue and server level. When a job is assigned one unit of a resource with this flag, the resources_assigned.<resource> attribute at the server and any queue is incremented by one. Must be consumable or time-based. This flag specifies that the resource is accumulated at the queue and server level, meaning that the value of resources_assigned.<resource> is incremented at each queue and at the server when a job is allocated this resource. When a reservation starts, allocated resources are added to the server's resources_assigned attribute. This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

5.14.2.3.iii When to Use Accumulation Flags

The following table shows when to use accumulation flags.

Table 5-10: When to Use Accumulation Flags

Resource Category	Server	Queue	Host
Static, consumable	flag = q	flag = q	flag = nh or fh
Static, not consumable	flag = (none of h, n, q or f)	flag = (none of h, n, q or f)	flag = h
Dynamic	`server_dyn_res` line in `sched_config`, flag = (none of h, n, q or f)	(cannot be used)	MoM config and `mom_resources` line (**deprecated** as of 18.2.1) in `sched_config`, flag = h

5.14.2.3.iv Example of Resource Accumulation Flags

When defining a static consumable host-level resource, such as a node-locked application license, you would use the "n" and "h" flags.

When defining a dynamic resource such as a floating license, you would use no flags.

5.14.2.3.v Resource Accumulation Flag Restrictions and Caveats

- Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be under-used. These resources are tracked automatically by the scheduler.

5.14.2.3.vi Resource Permission Flags

When you define a custom resource, you can specify whether unprivileged users have permission to view or request the resource, and whether users can `qalter` a request for that resource. This is done by setting a resource permission flag via `qmgr`.

5.14.2.3.vii Allowable Values for Resource Permission Flags

The permission flag for a resource can be one of the following:

Table 5-11: Resource Permission Flags

Flag	Meaning
(no flag)	Users can view and request the resource, and `qalter` a resource request for this resource.
i	"Invisible". Users cannot view or request the resource. Users cannot `qalter` a resource request for this resource.
r	"Read only". Users can view the resource, but cannot request it or `qalter` a resource request for this resource.

5.14.2.3.viii Effect of Resource Permission Flags

- PBS Operators and Managers can view and request a resource, and `qalter` a resource request for that resource, regardless of the i and r flags.
- Users, operators and managers cannot submit a job which requests a restricted resource. Any job requesting a restricted resource will be rejected. If a manager needs to run a job which has a restricted resource with a different value from the default value, the manager must submit the job without requesting the resource, then `qalter` the resource value.
- While users cannot request these resources, their jobs can inherit default resources from resources_default.<resource name> and default_chunk.<resource name>.

  If a user tries to request a resource or modify a resource request which has a resource permission flag, they will get an error message from the command and the request will be rejected. For example, if they try to `qalter` a job's resource request, they will see an error message similar to the following:

  ```
  "qalter: Cannot set attribute, read only or insufficient permission  Resource_List.hps 173.mars"
  ```

5.14.2.3.ix Resource Permission Flag Restrictions and Caveats

- You can specify only one of the i or r flags per resource. If both are specified, the resource is treated as if only the i flag were specified, and an error message is logged at the default log level and printed to standard error.
- Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag whether that resource was requested by the user or came from default_qsub_arguments.
- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  `pbsnodes`

  Users cannot view restricted host-level custom resources.

  `pbs_rstat`

  Users cannot view restricted reservation resources.

  `pbs_rsub`

  Users cannot request restricted custom resources for reservations.

  `qalter`

  Users cannot alter a restricted resource.

  `qmgr`

  Users cannot print or list a restricted resource.

  `qselect`

  Users cannot specify restricted resources via -l Resource_List.

  `qsub`

  Users cannot request a restricted resource.

  `qstat`

  Users cannot view a restricted resource.

5.14.2.3.x Allowing Execution Hooks to Read Custom Job Resources Faster

You can make it faster for execution hooks to read custom job resources. Execution hooks cannot read custom job resources via the event, only via the server. However, you can cache a copy of a custom job resource at the MoMs for faster local reading by execution hooks, by setting the *m* flag for the resource. The job resources that can be cached are found in the following job attributes:

exec_vnode
Resource_List
resources_used

To create a resource with the *m* flag set, include the flag. For example, to create two host-level consumable resources r1 and r2 of type long that will be cached at MoMs:

```
qmgr -c "create resource r1,r2 type=long,flag=mnh"
```

To unset this flag for r1:

```
qmgr -c "set resource r1 flag=nh"
```

You can combine this flag with any other resource flag. Job resources created in an exechost_startup hook have the m flag set automatically.

5.14.2.3.xi Caveats for Caching Custom Job Resources

Large numbers of job resources that are cached at MoMs can slow things down. If you don't need execution hooks to be able to read a custom job resource often, don't cache the resource at the MoMs.

5.14.2.3.xii Setting Types and Flags for Custom Resources via qmgr

To set the type for a resource:

set resource <resource name> type = <type>

For example:

```
qmgr -c "set resource foo type=string_array"
```

To set the flags for a resource:

set resource <resource name> flag=<flag(s)>

For example:

```
qmgr -c "set resource foo flag=nh"
```

To set the type and flags for a resource:

set resource <resource name> type=<type>, flag=<flag(s)>

For example:

```
qmgr -c "set resource foo type=long,flag=nhi"
```

You can set multiple resources by separating the names with commas. For example:

```
qmgr -c "set resource r1, r2 type=long"
```

You cannot set the *nh*, *fh*, or *q* flag for a resource of type string, string_array, or Boolean.

You cannot set both the *n* and the *f* flags on one resource.

You cannot have the *n* or *f* flags without the *h* flag.

You cannot set both the *i* and *r* flags on one resource.

You cannot unset the type for a resource.

You cannot set the type for a resource that is requested by a current or history job or reservation, or set on a server, queue, or vnode.

You cannot set the flag(s) to *h*, *nh*, *fh*, or *q* for a resource that is currently requested by a current or history job or reservation.

You cannot unset the flag(s) for a resource that is currently requested by a current or history job or a reservation, or set on any server, queue, or vnode.

You cannot alter a built-in resource.

You can unset custom resource flags, but not their type.

5.14.2.3.xiii

5.14.2.4 Defining Custom Resources via qmgr

You can use `qmgr` to create and delete custom resources, and to set their type and flags.

You must have PBS Manager privilege to operate on resources via `qmgr`.

5.14.2.4.i Creating Custom Resources via qmgr

When you define or change a custom resource via `qmgr`, the changes take place immediately, and you do not have to restart the server, but you do have to restart scheduler(s).

To create a resource:

qmgr -c 'create resource <resource name>[,<resource name>] [type = <type>], [flag = <flags>]'

For example:

```
Qmgr: create resource foo type=long,flag=q
```

To create multiple resources of the same type and flag, separate each resource name with a comma:

```
qmgr -c "create resource r1,r2 type=long,flag=nh"
```

You can abbreviate "resource" to "r":

```
qmgr -c "create r foo type=long,flag=nh"
```

You cannot create a resource with the same name as an existing resource.

After you have defined your new custom resource, tell the scheduler how to use it. See section 5.14.2.6, “Allowing Jobs to Use a Resource”, on page 265.

5.14.2.4.ii Caveats for Defining Host-level Custom Resources via qmgr

If you plan on using a hook to set a job's resources_used value for a custom host-level resource, or you want to have a custom resource summed in a job's resources_used attribute and shown in the accounting log, you must create that resource using a hook.

5.14.2.4.iii Deleting Custom Resources

If you want to be able to delete a custom resource, make sure that the resource is not requested by any current or history jobs or current reservations. You can let those jobs finish, `qalter` them, or delete them. Delete and re-create any reservations that request the resource.

Before you delete a custom resource, you must remove all references to that resource, including where it is used in hooks or the scheduling formula. When you delete a resource that is set on the server, a queue, or a vnode, PBS unsets the resource for you.

You cannot delete a custom resource that is listed in the restrict_res_to_release_on_suspend server attribute. You must first remove the resource from the list:

```
Qmgr: set server restrict_res_to_release_on_suspend -= <resource name>
```

You cannot delete a built-in resource.

To remove a custom resource:

1. Remove all references to the resource
   - Remove it from the formula
   - Remove it from hooks
   - Let jobs requesting it finish, requeue and then `qalter` them while they are queued, or delete them
   - Delete and re-create any reservations that request the resource
2. Edit the `resources:` line in `<sched_priv directory>/sched_config` to remove the unwanted resource name:
   - If the resource is a server dynamic resource, remove the resource name from the `server_dyn_res`: line
   - If the resource is a MoM dynamic resource, remove the resource from the `mom_resources:` line (**deprecated** as of 18.2.1)
3. For each MoM whose Version 2 configuration file contains references to the resource, use the `pbs_mom -s insert` command to update the Version 2 configuration file. See section 3.4.3, “Version 2 Vnode Configuration Files”, on page 42.
4. If the resource is a local dynamic resource, defined in the MoM Version 1 configuration file:

   For each host where the unwanted resource is defined, edit `PBS_HOME/mom_priv/config` and remove the resource entry line.
5. HUP each MoM; for Linux, see “Restarting and Reinitializing MoM” on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see “Restarting MoMs” on page 173 in the PBS Professional Installation & Upgrade Guide.
6. Delete the resource using `qmgr:`

   qmgr -c 'delete resource <resource name>'

   For example:

   ```
   qmgr -c "delete resource foo"
   ```

5.14.2.5 Defining Custom Resources via Hooks

You can use hooks to add new custom host-level non-consumable resources, and set their values. See "Adding Custom Non-consumable Host-level Resources" on page 64 in the PBS Professional Hooks Guide.

You must make the resource usable by the scheduler: see section 5.14.2.6, “Allowing Jobs to Use a Resource”, on page 265.

To delete a custom resource created in a hook, use `qmgr`. See section 5.14.2.4.iii, “Deleting Custom Resources”, on page 263.

5.14.2.6 Allowing Jobs to Use a Resource

After you define your resource, you need to make it usable by jobs:

1. Put the resource in the "`resources:`" line in `<sched_priv directory>/sched_config.` If the resource is a host-level boolean, you do not need to add it here.
2. If the resource is static, set the value via `qmgr`.
3. If the resource is dynamic, add it to the correct line in the scheduler's configuration file:
   - If it's a host -level dynamic resource, it must be added to the `mom_resources` line (**deprecated** as of 18.2.1)
   - If it's a server-level resource, it must be added to the `server_dyn_res` line
4. HUP the scheduler(s)

5.14.2.7 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

5.14.2.8 Dynamic Resource Scripts/Programs

You create the script or program that PBS uses to query the external source. The external source can be a license manager or a command, as when you use the `df` command to find the amount of available disk space. If the script is for a server-level dynamic resource, it is placed on the server. If it is for a host-level resource, it is placed on the host(s) where it will be used.

The default timeout for a server dynamic resource script is 30 seconds. You can specify a timeout for server dynamic resources in each scheduler's server_dyn_res_alarm attribute. If the script does not finish before the timeout, the scheduler uses a value of zero for the dynamic server resource. If you set the timeout to zero, the scheduler does not place a time limit on the script.

5.14.2.8.i Requirements for Scripts/Programs

- The script must be available to the scheduler, which runs the script
- If you have set up peer scheduling, make sure that the script is available to any scheduler that must run it
- The script must return its output via `stdout`, and the output must be in a single line ending with a newline
- The script must be owned by root
- The directory containing the script must be owned by root
- The directory containing the script must not give write permission to group or others.

5.14.2.9 Example of Defining Each Type of Custom Resource

In this example, we add five custom resources: a static and a dynamic host-level resource, a static and a dynamic server-level resource, and a static queue-level resource.

1. The resource must be defined, with appropriate flags set:

```
Qmgr: create resource staticserverresource   type=long, flag=q
Qmgr: create resource statichostresource     type=long, flag=nh
Qmgr: create resource dynamicserverresource  type=long
Qmgr: create resource dynamichostresource    type=long, flag=h
Qmgr: create resource staticqueueresource    type=long, flag=q
```

2. The resource must be added to the scheduler's list of resources:

Add resource to "resources:" line in <sched_priv directory>/sched_config:

```
resources:  "[...], staticserverresource, statichostresource, dynamicserverresource,
    dynamichostresource, staticqueueresource"
```

Host-level Boolean resources do not need to be added to the "resources:" line.

3. HUP the scheduler:

```
kill -HUP <scheduler PID>
```

4. If the resource is static, use qmgr to set it at the host, queue or server level:

```
Qmgr: set node Host1 resources_available.statichostresource=1
Qmgr: set queue Queue1 resources_available.staticqueueresource=1
Qmgr: set server resources_available.staticserverresource=1
```

See "qmgr" on page 149 of the PBS Professional Reference Guide.

5. If the resource is dynamic:

   a. If it's a host-level resource, add it to the "mom_resources" line (**deprecated** as of 18.2.1) in <sched_priv directory>/sched_config:

```
mom_resources: "dynamichostresource"
```

   b. Add it to the MoM config file PBS_HOME/mom_priv/config:

Linux or Windows:

```
dynamichostresource !path-to-command
```

Windows, spaces in path:

```
dynamichostresource !"path-to-command"
```

   c. If it's a server-level resource, add it to the "server_dyn_res" line in <sched_priv directory>/sched_config:

Linux:

```
server_dyn_res:  "dynamicserverresource !path-to-
command"
```

Windows, no spaces in path:

```
server_dyn_res: 'dynamicserverresource !path-to-
command'
```

or:

```
server_dyn_res: "dynamicserverresource !path-to-
command"
```

Windows, spaces in path:

```
server_dyn_res: 'dynamicserverresource !"path-to-
command including spaces"'
```

5.14.3 Creating Server-level Custom Resources

5.14.3.1 Dynamic Server-level Resources

The availability of a dynamic server-level resource is determined by running a script or program specified in the `server_dyn_res` line of `<sched_priv directory>/sched_config`. The value for resources_available.<resource name> is updated at each scheduling cycle with the value returned by the script. This script is run at the host where the scheduler runs, once per scheduling cycle. The script must return the value via `stdout` in a single line ending with a newline.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic server-level resource query is a shell escape:

server_dyn_res: "<resource name> !<path to command>"

In this query,

<resource name> is the name of the resource.

<path to command> is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

The scheduler runs the query and waits for it to finish or time out. The default timeout for a server dynamic resource script is 30 seconds. You can specify a timeout for server dynamic resources in each scheduler's server_dyn_res_alarm attribute.

Dynamic server-level resources are usually used for site-wide externally-managed floating application licenses.

Server dynamic resource values are never visible in `qstat`, and have no resources_available.<resource name> representation anywhere in PBS. If a job has requested a server dynamic resource, then the requested value shows up in the output of `qstat`.

5.14.3.1.i Example of Configuring Dynamic Server-level Resource

For a site-wide externally-managed floating application license you will need two resources: one to represent the licenses themselves, and one to mark the vnodes on which the application can be run. The first is a server-level dynamic resource and the second is a host-level Boolean, set on the vnodes to send jobs requiring that license to those vnodes.

These are the steps for configuring a dynamic server-level resource for a site-wide externally-managed floating license. If this license could be used on all vnodes, the Boolean resource would not be necessary.

1. Define the resources, for example floatlicense and CanRun:
   ```
   Qmgr: create resource floatlicense type=long
   Qmgr: create resource CanRun type=boolean, flag=h
   ```
2. Write a script, for example `serverdyn.pl`, that returns the available amount of the resource via stdout, and place it on the server host. For example, it could be placed in `/usr/local/bin/serverdyn.pl`
3. Configure the scheduler to use the script by adding the resource and the path to the script in the `server_dyn_res` line of `<sched_priv directory>/sched_config`:

   ```
   server_dyn_res: "floatlicense !/usr/local/bin/serverdyn.pl"
   ```
4. Optional: give the scheduler a time limit for the script by setting its server_dyn_res_alarm attribute:

   ```
   Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
   ```
5. Add the new dynamic resource to the `resources:` line in `<sched_priv directory>/sched_config`:

   ```
   resources: "ncpus, mem, arch, [...], floatlicense"
   ```
6. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.
7. Set the Boolean resource on the vnodes where the floating licenses can be run. Here we designate *vnode1* and *vnode2* as the vnodes that can run the application:

   ```
   Qmgr: active node vnode1,vnode2
   Qmgr: set node resources_available.CanRun=True
   ```

To request this resource, the job's resource request would include:

```
-l floatlicense=<number of licenses or tokens required>
-l select=1:ncpus=N:CanRun=1
```

5.14.3.2 Static Server-level Resources

Static server-level resources are used for resources like floating licenses that PBS will manage. PBS keeps track of the number of available licenses instead of querying an external license manager.

5.14.3.2.i Example of Configuring Static Server-level Resource

These are the steps for configuring a static server-level resource:

1. Define the resource, for example sitelicense:
   ```
   Qmgr: create resource sitelicense type=long, flag=q
   ```
2. Use the `qmgr` command to set the value of the resource on the server:

   ```
   Qmgr: set server resources_available.sitelicense=<number of licenses>
   ```
3. Add the new resource to the `resources:` line in `<sched_priv directory>/sched_config`.

   ```
   resources: "ncpus, mem, arch, [...], sitelicense"
   ```
4. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

5.14.4 Configuring Host-level Custom Resources

Host-level custom resources can be static and consumable, static and not consumable, or dynamic. Dynamic host-level resources are used for things like scratch space.

5.14.4.1 Dynamic Host-level Resources

For dynamic host-level custom resources, the scheduler sends a resource query to each MoM to get the current availability for the resource, and uses that value for scheduling. If the MoM returns a value, this value replaces the resources_available value reported by the server. If the MoM returns no value, the value from the server is kept. If neither specifies a value, the scheduler sets the resource value to 0.

The available amount of the resource is determined by running a script or program which returns the amount via `stdout`. This script or program is specified in the `mom_resources` line (**deprecated** as of 18.2.1) in `<sched_priv directory>/sched_config`.

The script is run once per scheduling cycle. For a multi-vnode machine, the script is run for the parent vnode. The resource is shared among the MoM's vnodes.

The scheduler tracks how much of each numeric dynamic server-level custom resource has been assigned to jobs, and will not overcommit these resources.

The format of a dynamic host-level resource query is a shell escape:

<resource name> !<path to command>

In this query,

<resource name> is the name of the resource.

<path to command> is typically the full path to the script or program that performs the query in order to determine the status and/or availability of the new resource you have added.

MoM starts the query and waits for output. The default amount of time that MoM waits is 10 seconds; this period can be set via the `-a alarm_timeout` command line option to `pbs_mom`. For Linux, see "Restarting and Reinitializing MoM" on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "Restarting MoMs" on page 173 in the PBS Professional Installation & Upgrade Guide. If the timeout is exceeded and the shell escape process has not finished, a log message, "resource read alarm" is written to the MoM's log file. The process is given another alarm period to finish and if it does not, another log message is written. The user's job may not run.

An example of a dynamic host-level resource is scratch space on the execution host.

Host dynamic resource values are never visible in `qstat`, and have no resources_available.<resource name> representation anywhere in PBS.

5.14.4.1.i Example of Configuring Dynamic Host-level Resource

In this example, we configure a custom resource to track host-level scratch space. The resource is called *dynscratch*. These are the steps for configuring a dynamic host-level resource:

1. Define the resource, for example dynscratch:
   ```
   Qmgr: create resource dynscratch type=size, flag=h
   ```
2. Write a script, for example `hostdyn.pl`, that returns the available amount of the resource via `stdout`. The script must return the value in a single line, ending with a newline. Place the script on each host where it will be used. For example, it could be placed in `/usr/local/bin/hostdyn.pl`.
3. Configure each MoM to use the script by adding the resource and the path to the script in `PBS_HOME/mom_priv/config`:

   Linux:

   ```
   dynscratch !/usr/local/bin/hostdyn.pl
   ```

Windows:

```
dynscratch !"C:\Program Files\PBS\hostdyn.pl"
```

4. Reinitialize the MoMs. For Linux, see "Restarting and Reinitializing MoM" on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "Restarting MoMs" on page 173 in the PBS Professional Installation & Upgrade Guide.
5. You may optionally specify any limits on that resource via `qmgr`, such as the maximum amount available, or the maximum that a single user can request. For example:

   ```
   Qmgr: set server resources_max.scratchspace=1gb
   ```
6. Add the new resource to the `resources:` line in `<sched_priv directory>/sched_config`:

   ```
   resources: "ncpus, mem, arch, [...], dynscratch"
   ```
7. Add the new resource to the `mom_resources:` line (**deprecated** as of 18.2.1) in `<sched_priv directory>/sched_config`. Create the line if necessary:

   ```
   mom_resources: "dynscratch"
   ```
8. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request this resource, the resource request would include

```
-l select=1:ncpus=N:dynscratch=10MB
```

5.14.4.2 Static Host-level Resources

Use static host-level resources for things that are managed by PBS and available at the host level, such as GPUs.

5.14.4.2.i Example of Configuring Static Host-level Resource

In this example, we configure a consumable host-level resource to track GPUs. These are the steps for configuring a static host-level resource:

1. Define the resource, for example ngpus:

   ```
   Qmgr: create resource ngpus type=long, flag=nh
   ```
2. Use the `qmgr` command to set the value of the resource on the host:

   ```
   Qmgr: set node Host1 ngpus=<number of GPUs>
   ```
3. Add the new resource to the `resources` line in `<sched_priv directory>/sched_config`.

   ```
   resources: "ncpus, mem, arch, [...], ngpus"
   ```
4. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.
5. If the GPU host is a multi-vnode machine, you may want to define which GPUs belong in which vnodes. In this case, do the following:
   a. Create a vnode definition file. See section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.
   b. Restart the MoM. For Linux, see "Restarting and Reinitializing MoM" on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "Restarting MoMs" on page 173 in the PBS Professional Installation & Upgrade Guide.

See section 5.14.6.4.iv, "Example of Per-host Node-locked Licensing", on page 279, section 5.14.6.4.v, "Example of Per-use Node-locked Licensing", on page 280, and section 5.14.6.4.vi, "Example of Per-CPU Node-locked Licensing", on page 281. These sections give examples of configuring each kind of node-locked license.

5.14.4.3 Shared Host-level Resources

Two or more vnodes can share the use of a resource. The resource is managed at one vnode, but available for use at other vnodes. The MoM manages the sharing of the resource, allocating only the available amount to jobs. For example, if you want jobs at two separate vnodes to be able to use the same 4GB of memory, you can make the memory be a shared resource. This way, if a job at one vnode uses all 4GB, no other jobs can use it, but if one job at one vnode uses 2GB, other jobs at either vnode can use up to 2GB.

5.14.4.3.i Shared Resource Glossary

Borrowing vnode

The vnode where a shared vnode resource is available, but not managed.

Indirect resource

A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Managing vnode

The vnode where a shared vnode resource is defined, and which manages the resource.

Shared resource

A vnode resource defined at managed at one vnode, but available for use at others.

5.14.4.3.ii Configuring Shared Host-level Resources

The resource to be shared is defined as usual at one vnode. This is the managing vnode for that resource. For example, to make memory be managed at Vnode1:

```
Qmgr: set node Vnode1 mem = 4gb
```

At vnodes which will use the same resource, the resource is defined to be indirect. For example, to make memory be shared and borrowed at *Vnode2*:

```
Qmgr: set node Vnode2 mem = @Vnode1
```

5.14.4.3.iii Shared Dynamic Host-level Resources

Vnode-level dynamic resources, meaning those listed in the `mom_resources:` line (**deprecated** as of 18.2.1) in `<sched_priv directory>/sched_config`, are shared resources.

5.14.4.3.iv Shared Static Host-level Resources

You can define a static host-level resource to be shared between vnodes. The resource is not shared if you set it to a value at each vnode.

5.14.4.3.v Configuring Shared Static Resources

1. If the resource to be shared is a custom resource, you must define the resource before setting its value:

   ```
   Qmgr: create resource <resource name> type=<resource type> [flag = <flags>]
   ```

2. Set the resource on the managing vnode:

   To set a static value via `qmgr`:

   ```
   Qmgr: s n managing_vnode resources_available.<resource name> =<value>
   ```

   To set a static value, in a Version 2 configuration file:

   ```
   managing_vnode:<resource name>=<value>
   ```

3. Next, set the resource on the borrowing vnode:

   To set a shared resource on a borrowing vnode via `qmgr`:

   ```
   Qmgr: s n borrowing_vnode resources_available.<resource name>=@managing_vnode
   ```

To set a shared resource in a Version 2 configuration file:

```
borrowing_vnode:<resource name>=@managing_vnode
```

4. HUP the MoMs involved; for Linux, see "Restarting and Reinitializing MoM" on page 167 in the PBS Professional Installation & Upgrade Guide, and for Windows, see "Restarting MoMs" on page 173 in the PBS Professional Installation & Upgrade Guide.

Example 5-10: To make a static host-level license `dyna-license` on hostA be managed by the parent vnode at hostA and indirect at vnodes hostA0 and hostA1:

```
Qmgr: set node hostA resources_available.dyna-license=4
Qmgr: set node hostA0 resources_available.dyna-license=@hostA
Qmgr: set node hostA1 resources_available.dyna-license=@hostA
```

5.14.4.3.vi Restrictions on Shared Host-level Resources

- If your vnodes represent physical units such as blades, sharing resources like ncpus across vnodes may not make sense.
- If you want to make a resource shared across vnodes, remember that you do not want to schedule jobs on the parent vnode. To avoid this, the following resources should not be explicitly set on the parent vnode:

  ncpus

  mem

  vmem

5.14.4.3.vii Defining Shared and Non-shared Resources for Multi-vnode Machines

On a multi-vnode machine, you can manage the resources at each vnode. For dynamic host-level resources, the resource is shared across all the vnodes on the machine, and MoM manages the sharing. For static host-level resources, you can either define the resource as shared or not. Shared resources are usually set on the parent vnode and then made indirect at any child vnodes on which you want the resource available. For resources that are not shared, you can set the value at each vnode.

Example 5-11: To set the resource `string_res` to *round* on the parent vnode of host03 and make it indirect at host03[0] and host03[1]:

```
Qmgr: set node host03 resources_available.string_res=round
Qmgr: s n host03[0] resources_available.string_res=@host03
Qmgr: s n host03[1] resources_available.string_res=@host03
pbsnodes -va
host03
    ...
    string_res=round
    ...
host03[0]
    ...
    string_res=@host03
    ...
host03[1]
    ...
    string_res=@host03
    ...
```

If you had set the resource string_res individually on host03[0] and host03[1]:

```
Qmgr: s n host03[0] resources_available.string_res=round
Qmgr: s n host03[1] resources_available.string_res=square
pbsnodes -va
host03
    ...
        <--------string_res not set on parent vnode
    ...
host03[0]
    ...
    string_res=round
    ...
host03[1]
    ...
    string_res=square
    ...
```

5.14.4.3.viii Shared Resource Restrictions for Multi-vnode Machines

- On the parent vnode, all values for resources_available.<resource name> should be *zero* (*0*), unless the resource is being shared among other vnodes via indirection.

5.14.5 Using Scratch Space

5.14.5.1 Dynamic Server-level (Shared) Scratch Space

If you have scratch space set up so that it's available to all execution hosts, you can use a server-level custom dynamic resource to track it. The following are the steps for configuring a dynamic server-level resource called globalscratch to track globally available scratch space:

1. Define the resource:
   ```
   Qmgr: create resource globalscratch type=long
   ```
2. Write a script, for example `serverdynscratch.pl`, that returns the available amount of the resource via `stdout`, and place it on the server host. For example, it could be placed in `/usr/local/bin/serverdynscratch.pl`
3. Configure the scheduler to use the script by adding the resource and the path to the script in the `server_dyn_res` line of `<sched_priv directory>/sched_config`:
   ```
   server_dyn_res: "globalscratch !/usr/local/bin/serverdynscratch.pl"
   ```
4. Optional: give the scheduler a time limit for the script by setting its server_dyn_res_alarm attribute:
   ```
   Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
   ```
5. Add the new dynamic resource to the `resources:` line in `<sched_priv directory>/sched_config`:
   ```
   resources: "ncpus, mem, arch, [...], globalscratch"
   ```
6. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request this resource, the job's resource request would include:

```
-l globalscratch=<space required>
```

5.14.5.2 Dynamic Host-level Scratch Space

Say you have jobs that require a large amount of scratch disk space during their execution. To ensure that sufficient space is available during job startup, create a custom dynamic resource so that jobs can request scratch space. To create this resource, take the steps outlined in section 5.14.4.1.i, "Example of Configuring Dynamic Host-level Resource", on page 269.

5.14.5.3 Static Server-level Scratch Space

If you want to prevent jobs from stepping on each others' scratch space, you can define additional vnodes that are used only to allocate scratch devices, with one vnode per scratch device. Set the sharing attribute on each scratch vnode to *force_excl*, so that only one job can request each scratch device. To set the sharing attribute, follow the rules in section 3.4.4, "Configuring the Vnode Sharing Attribute", on page 46. For example, the scratch devices are `/scratch1`, `/scratch2`, `/scratch3`, etc. On each scratch device, set resources as follows:

```
resources_available.ncpus = 0
resources_available.mem = 0
resources_available.scratch = 1
sharing = force_excl
```

Jobs then request one additional chunk to represent the scratch device, for example:

```
-l 16:ncpus=1+1:scratch=1
```

If a job needs to request a specific scratch device, for example `/scratch2`, that can be done by additionally asking for the scratch resource:

```
:scratch=1
```

5.14.5.4 Static Host-level Scratch Space

If the scratch areas are not mounted on all execution hosts, you can specify which scratch areas are shared among which subsets of vnodes using indirect resources. See section 5.14.4.3, "Shared Host-level Resources", on page 271.

5.14.5.5 Caveats for Scratch Space and Jobs

When more than one job uses scratch space, or when a job is suspended, scratch space usage may not be handled correctly. See section 5.9.5, "Dynamic Resource Allocation Caveats", on page 250 and section 5.9.6, "Period When Resource is Used by Job", on page 250.

5.14.6 Supplying Application Licenses

5.14.6.1 Types of Licenses

Application licenses may be managed by PBS or by an external license manager. Application licenses may be floating or node-locked, and they may be *per-host*, where any number of instances can be running on that host, *per-CPU*, where one license allows one CPU to be used for that application, or *per-run*, where one license allows one instance of the application to be running. Each kind of license needs a different form of custom resource.

5.14.6.1.i Externally-managed Licenses

Whenever an application license is managed by an external license manager, you must create a custom dynamic resource for it. This is because PBS has no control over whether these licenses are checked out, and must query the external license manager for the availability of those licenses. PBS does this by executing the script or program that you specify in the dynamic resource. This script returns the amount via `stdout`, in a single line ending with a newline.

5.14.6.1.ii Preventing Oversubscription of Externally-managed Licenses

Some applications delay the actual license checkout until some time after the application begins execution. Licenses could be oversubscribed when the scheduler queries for available licenses, and gets a result including licenses that essentially belong to a job that is already running but has not yet checked them out. To prevent this, you can create a consumable custom static integer resource, assign it the total number of licenses, and make each job that requests licenses request this resource as well. You can use a hook to accomplish this. Alternatively, if you know the maximum number of jobs that can run using these licenses, you can create a consumable custom static integer resource to track the number of jobs using licenses, and make each job request this resource.

If licenses are also checked out by applications outside of the control of PBS, this technique will not work.

5.14.6.1.iii PBS-managed Licenses

When an application license is managed by PBS, you can create a custom static resource for it. You set the total number of licenses using `qmgr`, and PBS will internally keep track of the number of licenses available.

Use static host-level resources for node-locked application licenses managed by PBS, where PBS is in full control of the licenses. These resources are *static* because PBS tracks them internally, and *host-level* because they are tracked at the host.

5.14.6.2 License Units and Features

Different licenses use different license units to track whether an application is allowed to run. Some licenses track the number of CPUs an application is allowed to run on. Some licenses use tokens, requiring that a certain number of tokens be available in order to run. Some licenses require a certain number of features to run the application.

When using units, after you have defined the license resource called license_name to the server, be sure to set resources_available.license_name to the correct number of units.

Before starting you should have answers to the following questions:

- How many units of a feature does the application require?
- How many features are required to execute the application?
- How do I query the license manager to obtain the available licenses of particular features?

With these questions answered you can begin configuring PBS Professional to query the license manager servers for the availability of application licenses. Think of a license manager feature as a resource. Therefore, you should associate a resource with each feature.

5.14.6.3 Server-level (Floating) Licenses

5.14.6.3.i Example of Floating, Externally-managed License

Here is an example of setting up floating licenses that are managed by an external license server.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have one licensed application which uses floating licenses from an external license manager. Furthermore we want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses, the hosts on which the licenses should be used, and a description of the type of license used by the application.

Application	Licenses	Hosts	DESCRIPTION
AppF	4	3-6	Uses licenses from an externally managed pool

For the floating licenses, we will use three resources. One is a dynamic server resource for the licenses themselves. One is a global server-level integer to prevent oversubscription. The last is a Boolean resource used to indicate that the floating license can be used on a given host.

Server Configuration

1. Define the new resources. Specify the resource names, type, and flag(s):
   Qmgr: create resource <resource name> type=<type>,flag=<flags>

Host Configuration

2. Set the Boolean resource on the hosts where the floating licenses can be used.

   Qmgr: active node host3,host4,host5,host6
   Qmgr: set node resources_available.runsAppF = True

Scheduler Configuration

3. Edit the scheduler configuration file:

   cd $<sched_priv directory>/
   [edit] sched_config

4. Append the new resource names to the **resources:** line:

   resources: "ncpus, mem, arch, host, [...], AppF, AppFcount, runsAppF"

5. Edit the **server_dyn_res:** line:

   server_dyn_res: "AppF !/local/flex_AppF"

6. Optional: give the scheduler a time limit for the script by setting its server_dyn_res_alarm attribute:

   Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>

7. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

You can write a hook that examines the number of AppF licenses requested by each job, and assigns that many AppFcount to the job, or you can ask your users to request AppFcount.

To request a floating license for AppF and a host on which AppF can run:

qsub -l AppF=1 -l AppFcount=1
-l select=runsAppF=True

The example below shows what the host configuration would look like. What is shown is actually truncated output from the **pbsnodes -a** command. Similar information could be printed via the **qmgr -c "print node @default"** command as well.

```
host1
host2
host3
     resources_available.runsAppF = True
host4
     resources_available.runsAppF = True
host5
     resources_available.runsAppF = True
host6
     resources_available.runsAppF = True
```

5.14.6.3.ii Example of Floating, Externally-managed License with Features

This is an example of a floating license, managed by an external license manager, where the application requires a certain number of features to run. Floating licenses are treated as server-level dynamic resources. The license server is queried by an administrator-created script. This script returns the value via **stdout** in a single line ending with a newline.

The license script runs on the server host once per scheduling cycle and queries the number of available licenses/tokens for each configured application.

When submitting a job, the user's script, in addition to requesting CPUs, memory, etc., also requests licenses.

When the scheduler looks at all the enqueued jobs, it evaluates the license request alongside the request for physical resources, and if all the resource requirements can be met the job is run. If the job's token requirements cannot be met, then it remains queued.

PBS doesn't actually check out the licenses; the application being run inside the job's session does that. Note that a small number of applications request varying amounts of tokens during a job run.

Our example needs four features to run an application, so we need four custom resources.

1. Write four scripts, one to query the license server for each of your four features. Complexity of the script is entirely site-specific due to the nature of how applications are licensed.
2. Define four non-consumable server-level features. These features are defined with no flags:

```
Qmgr: create resource feature1   type=long
Qmgr: create resource feature3   type=long
Qmgr: create resource feature6   type=long
Qmgr: create resource feature8   type=long
```

3. Add the feature resources to the `resources:` line in `<sched_priv directory>/sched_config`:

```
resources: "ncpus, mem, arch, [...], feature1, feature3, feature6, feature8"
```

4. Add each feature's script path to the `server_dyn_res:` line in `PBS_HOME/server_priv/config`:

```
server_dyn_res: "feature1 !/path/to/script [args]"
server_dyn_res: "feature3 !/path/to/script [args]"
server_dyn_res: "feature6 !/path/to/script [args]"
server_dyn_res: "feature8 !/path/to/script [args]"
```

5. Optional: give the scheduler a time limit for the scripts by setting its server_dyn_res_alarm attribute:

```
Qmgr: set sched <scheduler name> server_dyn_res_alarm=<new value>
```

6. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

5.14.6.3.iii Example of Floating License Managed by PBS

Here is an example of configuring custom resources for a floating license that PBS manages. For this you need a server-level static resource to keep track of the number of available licenses. If the application can run only on certain hosts, then you will need a host-level Boolean resource to direct jobs running the application to the correct hosts.

In this example, we have six hosts numbered 1-6, and the application can run on hosts 3, 4, 5 and 6. The resource that will track the licenses is called *AppM*. The Boolean resource is called *RunsAppM*.

Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>,flag=<flags>
```

   Example:

```
Qmgr: create resource AppM type=long, flag=q
Qmgr: create resource runsAppM type=boolean, flag=h
```

2. Set a value for AppM at the server. Here, we're allowing 8 copies of the application to run at once:

```
Qmgr: set server resources_available.AppM=8
```

Host Configuration

3. Set the value of runsAppM on the hosts. Each qmgr directive is typed on a single line:

```
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.runsAppM = True
```

Scheduler Configuration

4. Edit the scheduler configuration file:

```
cd $<sched_priv directory>/
[edit] sched_config
```

5. Append the new resource name to the resources: line. Note that it is not necessary to add a host-level Boolean resource to this line.

```
resources: "ncpus, mem, arch, host, [...], AppM, runsAppM"
```

6. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request both the application and a host that can run AppM:

```
qsub -l AppM=1
-l select=1:runsAppM=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the pbsnodes -a command. Similar information could be printed via the qmgr -c "print node @default" command as well. Since unset Boolean resources are the equivalent of *False*, you do not need to explicitly set them to *False* on the other hosts. Unset Boolean resources will not be printed.

```
host1
host2
host3
     resources_available.runsAppM = True
host4
     resources_available.runsAppM = True
host5
     resources_available.runsAppM = True
host5
     resources_available.runsAppM = True
```

5.14.6.4 Host-level (Node-locked) Licenses

5.14.6.4.i Per-host Node-locked Licenses

If you are configuring a custom resource for a per-host node-locked license, where the number of jobs using the license does not matter, use a host-level Boolean resource on the appropriate host. This resource is set to *True*. When users request the license, they can use the following requests:

For a two-CPU job on a single vnode:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

Users can also use "license=True", but this way they do not have to change their scripts.

5.14.6.4.ii Per-CPU Node-locked Licenses

If you are configuring a custom resource for a per-CPU node-locked license, use a host-level consumable resource on the appropriate vnode. This resource is set to the maximum number of CPUs you want used on that vnode. Then when users request the license, they will use the following request:

For a two-CPU, two-license job:

```
-l select=1:ncpus=2:license=2
```

5.14.6.4.iii Per-use Node-locked License

If you are configuring a custom resource for a per-use node-locked license, use a host-level consumable resource on the appropriate host. This resource is set to the maximum number of instances of the application allowed on that host. Then when users request the license, they will use:

For a two-CPU job on a single host:

```
-l select=1:ncpus=2:license=1
```

For a multi-vnode job where each chunk needs two CPUs:

```
-l select=2:ncpus=2:license=1
-l place=scatter
```

5.14.6.4.iv Example of Per-host Node-locked Licensing

Here is an example of setting up node-locked licenses where one license is required per host, regardless of the number of jobs on that host.

For this example, we have a 6-host complex, with one CPU per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-host node-locked licenses. We want to limit use of the application only to specific hosts. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

Application	Licenses	Hosts	DESCRIPTION
AppA	1	1-4	uses a local node-locked application license

For the per-host node-locked license, we will use a Boolean host-level resource called resources_available.runsAppA. This will be set to *True* on any hosts that should have the license, and will default to *False* on all others. The resource is not consumable so that more than one job can request the license at a time.

Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

   ```
   create resource <resource name> type=<type>,flag=<flag>
   ```

   Example:

   ```
   Qmgr: create resource runsAppA type=boolean, flag=h
   Qmgr: create resource AppA type=long, flag=h
   ```

Host Configuration

2. Set the value of runsAppA on the hosts. Each `qmgr` directive is typed on a single line:

   ```
   Qmgr: active node host1,host2,host3,host4
   Qmgr: set node resources_available.runsAppA = True
   ```

Scheduler Configuration

3. Edit the scheduler configuration file.

```
cd $<sched_priv directory>/
[edit] sched_config
```

4. Append the new resource name to the "resources:" line. Note that it is not necessary to add the host-level Boolean resource to this line.

```
resources: "ncpus, mem, arch, [...], AppA, runsAppA"
```

5. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request a host with a per-host node-locked license for AppA:

```
qsub -l select=1:runsAppA=1 <jobscript>
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the pbsnodes -a command. Similar information could be printed via the qmgr -c "print node @default" command as well. Since unset Boolean resources are the equivalent of *False*, you do not need to explicitly set them to *False* on the other hosts. Unset Boolean resources will not be printed.

```
host1
     resources_available.runsAppA = True
host2
     resources_available.runsAppA = True
host3
     resources_available.runsAppA = True
host4
     resources_available.runsAppA = True
host5
host6
```

5.14.6.4.v Example of Per-use Node-locked Licensing

Here is an example of setting up per-use node-locked licenses. Here, while a job is using one of the licenses, it is not available to any other job.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-use node-locked licenses. We want to limit use of the application only to specific hosts. The licensed hosts can run two instances each of the application. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

Application	Licenses	Hosts	DESCRIPTION
AppB	2	1-2	Uses a local node-locked application license

For the node-locked license, we will use one static host-level resource called resources_available.AppB. This will be set to *2* on any hosts that should have the license, and to *0* on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 2 licenses, only two jobs can use those licenses on that host at a time.

Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

```
Qmgr: create resource <resource name> type=<type>,flag=<flags>
```

Example:

```
Qmgr: create resource AppB type=long, flag=nh
```

Host Configuration

2. Set the value of AppB on the hosts to the maximum number of instances allowed. Each qmgr directive is typed on a single line:

```
Qmgr: active node host1,host2
Qmgr: set node resources_available.AppB = 2
Qmgr: active node host3,host4,host5,host6
Qmgr: set node resources_available.AppB = 0
```

Scheduler Configuration

3. Edit the scheduler configuration file.

```
cd $<sched_priv directory>/
[edit] sched_config
```

4. Append the new resource name to the resources: line:

```
resources: "ncpus, mem, arch, host, [...], AppB"
```

5. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request a host with a node-locked license for AppB, where you'll run one instance of AppB on two CPUs:

```
qsub -l select=1:ncpus=2:AppB=1
```

The example below shows what the host configuration would look like. What is shown is actually truncated output from the pbsnodes -a command. Similar information could be printed via the qmgr -c "print node @default" command as well.

```
host1
     resources_available.AppB = 2
host2
     resources_available.AppB = 2
host3
     resources_available.AppB = 0
host4
     resources_available.AppB = 0
host5
     resources_available.AppB = 0
host6
     resources_available.AppB = 0
```

5.14.6.4.vi Example of Per-CPU Node-locked Licensing

Here is an example of setting up per-CPU node-locked licenses. Each license is for one CPU, so a job that runs this application and needs two CPUs must request two licenses. While that job is using those two licenses, they are unavailable to other jobs.

For this example, we have a 6-host complex, with 4 CPUs per host. The hosts are numbered 1 through 6. On this complex we have a licensed application that uses per-CPU node-locked licenses. We want to limit use of the application to specific hosts only. The table below shows the application, the number of licenses for it, the hosts on which the licenses should be used, and a description of the type of license used by the application.

Application	Licenses	Hosts	DESCRIPTION
AppC	4	3-4	uses a local node-locked application license

For the node-locked license, we will use one static host-level resource called resources_available.AppC. We will provide a license for each CPU on hosts 3 and 4, so this will be set to *4* on any hosts that should have the license, and to *0* on all others. The "nh" flag combination means that it is host-level and it is consumable, so that if a host has 4 licenses, only four CPUs can be used for that application at a time.

Server Configuration

1. Define the new resource. Specify the resource names, type, and flag(s):

   Qmgr: create resource <resource name> type=<type>,flag=<flags>

   Example:

   Qmgr: create resource AppC type=long, flag=nh

Host Configuration

2. Set the value of AppC on the hosts. Each qmgr directive is typed on a single line:

   Qmgr: active node host3,host4
   Qmgr: set node resources_available.AppC = 4
   Qmgr: active node host1,host2,host5,host6
   Qmgr: set node resources_available.AppC = 0

Scheduler Configuration

3. Edit the scheduler configuration file:

   cd $<sched_priv directory>/
   [edit] sched_config

4. Append the new resource name to the **resources:** line:

   resources: "ncpus, mem, arch, host, [...], AppC"

5. Restart the scheduler. See "Restarting and Reinitializing Scheduler or Multisched" on page 166 in the PBS Professional Installation & Upgrade Guide.

To request a host with a node-locked license for AppC, where you'll run a job using two CPUs:

qsub -l select=1:ncpus=2:AppC=2

The example below shows what the host configuration would look like. What is shown is actually truncated output from the `pbsnodes -a` command. Similar information could be printed via the `qmgr -c "print node @default"` command as well.

```
host1
     resources_available.AppC = 0
host2
     resources_available.AppC = 0
host3
     resources_available.AppC = 4
host4
     resources_available.AppC = 4
host5
     resources_available.AppC = 0
host6
     resources_available.AppC = 0
```

5.14.7 Using GPUs

You can configure PBS to manage GPU resources. You only need to use one method. You can use any of the following methods, but we recommend using the cgroups hook:

- Managing GPUs via Cgroups does the configuration for you, takes advantage of topology, and provides isolation. You can use this method to restrict each job to the GPU(s) assigned to it. We recommend using this method: you get device isolation, job submission is much easier, etc.
- Basic GPU Scheduling works well if you have single-GPU vnodes. The basic method will meet the needs of most job submitters; it allows a job to request the number of GPUs it needs, as long as the job requests exclusive use of each node containing the GPUs.
- Advanced GPU Scheduling allows jobs to request specific GPUs. The advanced method provides some flexibility for multi-job or multi-GPU vnodes, but does not isolate GPUs. PBS Professional allocates GPUs for jobs, but does not perform the actual binding. The application or the CUDA library binds the application to one or more GPUs. You cannot combine this with the cgroups hook.

5.14.7.1 Basic GPU Scheduling

Basic scheduling consists of prioritizing jobs based on partition or site policies, controlling access to nodes with GPUs, ensuring that GPUs are not over-subscribed, and tracking use of GPUs in accounting logs.

Configuring PBS to perform basic scheduling of GPUs is relatively simple, and only requires defining and configuring a single custom resource to represent the number of GPUs on each node.

This method allows jobs to request unspecified GPUs. Jobs should request exclusive use of the node to prevent other jobs being scheduled on their GPUs.

5.14.7.1.i Configuring PBS for Basic GPU Scheduling

You configure a single custom consumable resource to represent all GPU devices on an execution host. Create a host-level global consumable custom resource to represent GPUs. We recommend that the custom GPU resource is named *ngpus*. Set the value for this resource at each vnode to the number of GPUs on the vnode.

The ngpus resource is used exactly the way you use the ncpus resource.

5.14.7.1.ii Example of Configuring PBS for Basic GPU Scheduling

In this example, there are two execution hosts, HostA and HostB, and each execution host has 4 GPU devices.

1. Create the ngpus resource:
   Qmgr: create resource ngpus type=long, flag=nh
2. Stop the server and scheduler. On the server's host, type:

   systemctl stop pbs

   or

   /etc/init.d/pbs stop
3. Edit <sched_priv directory>/sched_config to add ngpus to the list of scheduling resources:

   resources: "ncpus, mem, arch, host, vnode, ngpus"
4. Start the server and scheduler. On the server's host, type:

   systemctl start pbs

   or

   /etc/init.d/pbs start
5. Add the number of GPU devices available to each execution host in the cluster via qmgr:

   Qmgr: set node HostA resources_available.ngpus=4
   Qmgr: set node HostB resources_available.ngpus=4

5.14.7.2 Advanced GPU Scheduling

Advanced scheduling allows a job to separately allocate (request and/or identify) each individual GPU on a node.

In this case, both PBS and the applications themselves must support individually allocating the GPUs on a node. Advanced scheduling requires defining a child vnode for each GPU.

This capability is useful for sharing a single multi-GPU node among multiple jobs, where each job requires exclusive use of its GPUs.

5.14.7.2.i Configuring PBS for Advanced GPU Scheduling

You configure each GPU device in its own vnode, and each GPU vnode has a resource to contain the device number of its GPU.

Create and set two custom resources:

- Create a host-level global consumable resource to represent the GPUs on a vnode. We recommend that this resource is called *ngpus*.

  Set ngpus on each node to the number of GPUs on that node.
- Create a host-level global non-consumable resource containing the GPU device number, which serves to tie the individual GPU to the vnode. We recommend that this resource is called *gpu_id*.

  Set gpu_id for each GPU to the device number of that GPU.

5.14.7.2.ii Example of Configuring PBS for Advanced GPU Scheduling

In this example, there is one execution host, HostA, that has two child vnodes, HostA[0] and HostA[1], as well as the parent vnode. HostA has 4 CPUs, 2 GPUs, and 16 GB of memory.

1. Create the new custom resources:
   ```
   Qmgr: create resource ngpus type=long, flag=nh
   Qmgr: create resource gpu_id type=string, flag=h
   ```
2. Stop the server and scheduler. On the server's host, type:

   ```
   systemctl stop pbs
   ```

   or

   ```
   /etc/init.d/pbs stop
   ```
3. Edit <sched_priv directory>/sched_config to add ngpus and gpu_id to the list of scheduling resources:

   ```
   resources: "ncpus, mem, arch, host, vnode, ngpus, gpu_id"
   ```
4. Start the server and scheduler. On the server's host, type:

   ```
   systemctl start pbs
   ```

   or

   ```
   /etc/init.d/pbs start
   ```
5. Create a vnode configuration file for each execution host where GPUs are present. See section 3.4.3, “Version 2 Vnode Configuration Files”, on page 42. The script for HostA is named *hostA_vnodes*, and is shown here:

   ```
   $configversion 2
   hostA: resources_available.ncpus = 0
   hostA: resources_available.mem = 0
   hostA[0]: resources_available.ncpus = 2
   hostA[0] : resources_available.mem = 8gb
   hostA[0] : resources_available.ngpus = 1
   hostA[0] : resources_available.gpu_id = gpu0
   hostA[0] : sharing = default_excl
   hostA[1] : resources_available.ncpus = 2
   hostA[1] : resources_available.mem = 8gb
   hostA[1] : resources_available.ngpus = 1
   hostA[1] : resources_available.gpu_id = gpu1
   hostA[1]: sharing = default_excl
   ```
6. Create a Version 2 configuration file for each host with GPUs. For example:

   ```
   PBS_EXEC/sbin/pbs_mom -s insert HostA_vnodes HostA_vnodes
   ```
7. Signal each MoM to re-read its configuration files:

   ```
   kill -HUP <pbs_mom PID>
   ```

5.14.7.3 Managing GPUs Via Cgroups Hook

We describe how to manage your GPUs via cgroups in section 16.5.5.1, “Managing GPUs via Cgroups”, on page 600.

5.14.8 Using FPGAs

You can configure a custom resource that allows PBS to track the usage of FPGAs. The FPGAs are detected outside of PBS at boot time. There are two basic methods for automatic configuration of the FPGA resource:

- Create a global static host-level resource called *nfpgas*. Create a boot-up script in `init.d` that detects the presence of the FPGAs, and sets the value of the nfpgas resource.
- Create a global dynamic host-level resource called *nfpgas*. This resource calls a script to detect the presence of FPGAs

We recommend the static resource, because FPGAs are static, and there is a performance penalty for a dynamic resource.

5.14.9 Defining Host-level Resource for Applications

You may need to tag your vnodes with the software that can run on them. You cannot use the built-in software resource for this; it is a server-level resource and cannot be set per host. You can define a custom resource named, for example, "node_software". It should be a string_array, since a host may be able to run more than one application. You can use `qmgr` to create your resource:

```
Qmgr: create resource node_software type=string_array, flag=h
```

You can use your new custom resource to route jobs: see section 4.9.39, “Routing Jobs”, on page 205.

5.14.10 Custom Resource Caveats

- Because some custom resources are external to PBS, they are not completely under the control of PBS. Therefore it is possible for PBS to query and find a resource available, schedule a job to run and use that resource, only to have an outside entity take that resource before the job is able to use it. For example, say you had an external resource of "scratch space" and your local query script simply checked to see how much disk space was free. It would be possible for a job to be started on a host with the requested space, but for another application to use the free space before the job did.
- If a resource is not put in the scheduler's `resources:` line, when jobs request the resource, that request will be ignored. If the resource is ignored, it cannot be used to accept or reject jobs at submission time. For example, if you create a string resource String1 on the server, and set it to *foo*, a job requesting "`-l String1=bar`" will be accepted. The only exception is host-level Boolean resources, which are considered when scheduling, whether or not they are in the scheduler's `resources:` line.
- Do not create resources with the same names or prefixes that PBS uses when you create custom resources for specific systems.
- Using dynamic host-level resources can slow the scheduler down, because the scheduler must wait for each resource-query script to run.

5.15 Managing Resource Usage

You can manage resource usage from different directions:

- You can manage resource usage by users, groups, and projects, and the number of jobs, at the server and queue level. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.
  - You can manage the total amount of each resource that is used by projects, users or groups, at the server or queue level. For example, you can manage how much memory is being used by jobs in queue QueueA.
  - You can manage the number of jobs being run by projects, users or groups, at the server or queue level. For example, you can limit the number of jobs enqueued in queue QueueA by any one group to *30*, and by any single user to *5*.
- You can specify how much of each resource any job is allowed to use, at the server and queue level. See section 5.15.2, "Placing Resource Limits on Jobs", on page 304 and section 5.13, "Using Resources to Restrict Server or Queue Access", on page 254.
- You can set default limits for usage for each resource, at the server or queue level, so that jobs that do not request a given resource inherit that default, and are limited to the inherited amount. For example, you can specify that any job entering queue QueueA not specifying mem is limited to using 4MB of memory. See section 5.9.3, "Specifying Job Default Resources", on page 245.
- You can set limits on the number of jobs that can be in the queued state at the server and/or queue level. You can apply these limits to users, groups, projects, or everyone. This allows users to submit as many jobs as they want, while allowing the scheduler to consider only the jobs in the execution queues, thereby speeding up the scheduling cycle. See section 5.15.3, "Limiting the Number of Jobs in Queues", on page 309.

5.15.1 Managing Resource Usage By Users, Groups, and Projects, at Server & Queues

You can set separate limits for resource usage by individual users, individual groups, individual projects, generic users, generic groups, generic projects, and the total used overall, for queued jobs, running jobs, and queued and running jobs. You can limit the amount of resources used, and the number of queued jobs, the number of running jobs, and the number of queued and running jobs. These limits can be defined separately for each queue and for the server. You define the limits by setting server and queue limit attributes. For information about projects, see section 14.4, "Grouping Jobs By Project", on page 507.

There are **two incompatible sets of server and queue limit attributes** used in limiting resource usage. The first set existed in PBS Professional before Version 10.1, and we call them the **old limit attributes**. The old limit attributes are discussed in section 5.15.1.15, "Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1", on page 302. The set introduced in Version 10.1 is called simply the limit attributes, and they are discussed here.

You can use either the limit attributes or the old limit attributes for the server and queues, but not both. See section 5.15.1.13.v, "Do Not Mix Old And New Limits", on page 301.

The server and queues each have per-job limit attributes which operate independently of the limits discussed in this section. The resources_min.<resource name> and resources_max.<resource name> server and queue attributes are limits on what each individual job may use. See section 5.13, "Using Resources to Restrict Server or Queue Access", on page 254 and section 5.15.2, "Placing Resource Limits on Jobs", on page 304.

5.15.1.1 Examples of Managing Resource Usage at Server and Queues

You can limit resource usage and job count for specific projects, users and groups:

- UserA can use no more than 6 CPUs, and UserB can use no more than 4 CPUs, at one time anywhere in the PBS complex.
- The crashtest group can use no more than 16 CPUs at one time anywhere in the PBS complex.
- UserC accidentally submitted 200,000 jobs last week. UserC can now have no more than 25 jobs enqueued at one time.
- All jobs request the server-level custom resource nodehours, which is used for allocation. UserA cannot use more than 40 nodehours in the PBS complex. Once UserA reaches the nodehours limit, then all queued jobs owned by UserA are not eligible for execution.
- You wish to allow UserD to use 12 CPUs but limit all other users to 4 CPUs.
- Jobs belonging to Project A can use no more than 8 CPUs at Queue1.

You can limit the number of jobs a particular project, user or group runs in a particular queue:

- UserE can use no more than 2 CPUs at one time at Queue1, and 6 CPUs at one time at Queue2.
- You wish to limit UserF to 10 running jobs in queue Queue3, but allow all other users unlimited jobs running in the same queue.
- UserG is a member of Group1. You have a complex-wide limit of 5 running jobs for UserG. You have a limit at Queue1 of 10 running jobs for Group1. This way, up to 10 of the running jobs in Queue1 can belong to Group1, and 5 of these can belong to UserG.
- UserH is a member of Group1. You have a complex-wide limit of 5 running jobs for UserH. You have a limit at Queue1 of 10 running jobs for any group in Queue1. This way, no group in Queue1 can run more than 10 jobs total at one time, and 5 of these can belong to UserH.
- UserJ is a member of Group1. You have a complex-wide limit of 10 running jobs for UserJ. You also have a limit at Queue1 of 5 running jobs for Group1. This means that there may be up to 5 running jobs owned by users belonging to Group1 in Queue1, and up to 5 of these can be owned by UserJ. UserJ can also have another 5 running jobs owned by Group1 in any other queue, or owned by a different group in Queue1.
- No more than 12 jobs belonging to Project A can run at Queue1, and all other projects are limited to 8 jobs at Queue1.

You can ensure fairness in the use of resources:

- You have multiple departments which have shared the purchase of a large machine. Each department would like to ensure fairness in the use of the machine, by setting limits on individual users and groups.
- You have multiple departments, each of which purchases its own machines. Each department would like to limit the use of its machines so that all departmental users have specific limits. In addition, each department would like to allow non-departmental users to use its machines when they are under-utilized, while giving its own users priority on its machines. A non-departmental user can run jobs on a departmental machine, as long as no departmental users' jobs are waiting to run.

5.15.1.2 Glossary

Limit

The maximum amount of a resource that can be consumed at any time by running jobs or allocated to queued jobs, or the maximum number of jobs that can be running, or the maximum number of jobs that can be queued.

Overall limit

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

Generic user limit

Applies separately to users at the server or a queue. The limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Generic group limit

Applies separately to groups at the server or a queue. The limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic project limit

Applies separately to projects at the server or a queue. The limit for projects which have no individual limit specified. A limit for generic projects is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Individual user limit

Applies separately to users at the server or a queue. Limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Individual group limit

Applies separately to groups at the server or a queue. Limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual project limit

Applies separately to projects at the server or a queue. Limit for a project which has its own individual limit specified. An individual project limit overrides the generic project limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

User limit

A limit placed on one or more users, whether generic or individual.

Group limit

This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

Project

In PBS, a project is a way to group jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's project attribute specifies the job's project.

Project limit

This is a limit applied to the total used by a project, whether the limit is a generic project limit or an individual project limit.

Queued jobs

In a queue, queued jobs are the jobs that are waiting in that queue.

5.15.1.3 Difference Between PBS_ALL and PBS_GENERIC

Note the very important **difference** between the *overall limit* and a *generic* limit. We will describe how this works for users, but this applies to other entities as well. You set PBS_ALL for an overall limit on the total usage of that resource by all entities, whereas you set PBS_GENERIC for a limit for any single generic user.

Example 5-12: Difference between overall limit and generic user limit

Given the following:

- The overall server limit for running jobs is *100*
- The server limit for generic users is *10*
- The individual limit for User1 is 12 jobs

This means:

- Generic users (any single user except User1) can run no more than 10 jobs at this server
- User1 can run 12 jobs at this server
- At this server, no more than 100 jobs can be running at any time

5.15.1.4 Hard and Soft Limits

Hard limits are limits which cannot be exceeded. Soft limits are limits which mark the point where a project, user or group is using "extra, but acceptable" amounts of a resource. When this happens, the jobs belonging to that project, user or group are eligible for preemption. See section 4.9.33, “Using Preemption”, on page 180. Soft limits are discussed in section 4.9.33.7.i, “The Soft Limits Preemption Level”, on page 185.

5.15.1.5 Scope of Limits at Server and Queues

Each of the limits described above can be set separately at the server and at each queue. Each limit's scope is the PBS object where it is set. The individual and generic project, user and group limits that are set within one scope interact with each other only within that scope. For example, a limit set at one queue has no effect at another queue.

The scope of limits set at the server encompasses queues, so that the minimum, more restrictive limit of the two is applied. For precedence within a server or queue, see section 5.15.1.7, “Precedence of Limits at Server and Queues”, on page 293.

5.15.1.6 Ways To Limit Resource Usage at Server and Queues

You can create a complete set of limits at the server, and you can create another complete set of limits at each queue. You can set hard and soft limits. See section 4.9.33.7.i, "The Soft Limits Preemption Level", on page 185. You can limit resource usage at the server and the queue level for the following:

- Running jobs
  - Number of running jobs
  - Number of running jobs (soft limit)
  - Amount of each resource allocated for running jobs
  - Amount of each resource allocated for running jobs (soft limit)
- Queued jobs (this means jobs that are waiting to run from that queue)
  - Number of queued jobs
  - Amount of each resource allocated for queued jobs
- Queued and running jobs (this means both jobs that are waiting to run and jobs that are running from that queue)
  - Number of queued and running jobs
  - Amount of each resource allocated for queued and running jobs

These limits can be applied to the following:

- The total usage at the server
- The total usage at each queue
- Amount used by a single user
  - Generic users
  - Individual users
- Amount used by a single group
  - Generic groups
  - Individual groups
- Amount used by a single project
  - Generic projects
  - Individual projects

5.15.1.6.i Limits at Queues

You can limit the number of jobs that are queued at a queue, and running at a queue, and that are both queued and running at a queue.

You can limit the resources allocated to jobs that are queued at a queue, and running at a queue, and that are both queued and running at a queue.

Jobs queued at a queue are counted the same whether they were submitted to that queue via the `qsub` command or its equivalent API, moved to that queue via the `qmove` command or its equivalent API, or routed to that queue from another queue.

When PBS requeues a job, it does not take limits into account.

Routing queues do not run jobs, so you cannot set a limit for the number of running jobs, or the amount of resources being used by running jobs, at a routing queue.

5.15.1.6.ii Generic and Individual Limits

You can set a generic limit for groups, so that each group must obey the same limit. You can likewise set a generic limit for users and projects. Each generic limit can be set separately at the server and at each queue. For example, if you have two queues, the generic limit for the number of jobs a user can run be 4 at QueueA and 6 at QueueB.

You can set a different individual limit for each user, and you can set individual limits for groups and for projects. Each user, group, and project can have a different individual limit at the server and at each queue.

You can use a combination of generic and individual project, user or group limits, at the server and at each queue. Within the scope of the server or a queue, all projects, users or groups except the ones with the individual limits must obey the generic limit, and the individual limits override the generic limits.

Example 5-13: Generic and individual user limits on running jobs at QueueA and QueueB

At QueueA:

- At QueueA, the generic user limit is *5*
- At QueueA, Bob's individual limit is *8*
- Tom has no individual limit set at QueueA; the generic limit applies

At QueueB:

- At QueueB, the generic user limit is *2*
- At QueueB, Tom's individual limit is *1*
- Bob has no individual limit at QueueB; the generic limit applies

This means:

- Bob can run 8 jobs at QueueA
- Bob can run 2 jobs at QueueB
- Tom can run 5 jobs at QueueA
- Tom can run 1 job at QueueB

5.15.1.6.iii Overall Limits

The overall limit places a cap on the total amount of the resource that can be used within the scope in question (server or queue), regardless of whether project, user, or group limits have been reached. A project, user, or group at the server or a queue cannot use any more of a resource for which the overall limit has been reached, even if that project, user, or group limit has not been reached.

Example 5-14: Overall limit at server

Given the following:

- Overall server limit on running jobs is *100*
- Bob's user limit is *10* running jobs
- 98 jobs are already running
- Bob is running zero jobs

This means:

- Bob can start only 2 jobs

5.15.1.7 Precedence of Limits at Server and Queues

5.15.1.7.i Interactions Between Limits Within One Scope

Within the scope of a PBS object (server or queue), there is an order of precedence for limits when more than one applies to a job. The order of precedence for the limits at a queue is the same as the order at the server. The following table shows how limits interact within one scope:

Table 5-12: Limit Interaction Within One Scope

	Individual User	Generic User	Individual Group	Generic Group	Individual Project	Generic Project
Individual User	Individual user	Individual user	More restrictive	More restrictive	More restrictive	More restrictive
Generic User	Individual user	Generic user	More restrictive	More restrictive	More restrictive	More restrictive
Individual Group	More restrictive	More restrictive	Individual group	Individual group	More restrictive	More restrictive
Generic Group	More restrictive	More restrictive	Individual group	Generic group	More restrictive	More restrictive
Individual Project	More restrictive	More restrictive	More restrictive	More restrictive	Individual project	Individual project
Generic Project	More restrictive	More restrictive	More restrictive	More restrictive	Individual project	Generic project

An individual user limit overrides a generic user limit.

Example 5-15: Individual user limit overrides generic user limit

Given the following:

- Bob has a limit of 10 running jobs
- The generic limit is 5

This means:

- Bob can run 10 jobs

An individual group limit overrides a generic group limit in the same manner as for users.

If the limits for a user and the user's group are different, the more restrictive limit applies.

Example 5-16: More restrictive user or group limit applies

Given the following:

- Tom's user limit for running jobs is 8
- Tom's group limit is 7

This means:

- Tom can run only 7 jobs in that group

If a user belongs to more than one group, that user can run jobs up to the lesser of his user limit or the sum of the group limits.

Example 5-17: User can run jobs in more than one group

Given the following:

- Tom's user limit is *10* running jobs
- GroupA has a limit of *2* and GroupB has a limit of *4*
- Tom belongs to GroupA and GroupB

This means:

- Tom can run 6 jobs, 2 in GroupA and 4 in GroupB

An individual project limit overrides a generic project limit, similar to the way user and group limits work.

Project limits are applied independently of user and group limits.

Example 5-18: Project limits are applied without regard to user and group limits

Given the following:

- Project A has a limit of 2 jobs
- Bob has an individual limit of 4 jobs
- Bob's group has a limit of 6 jobs
- Bob is running 2 jobs, both in Project A

This means:

- Bob cannot run any more jobs in Project A

5.15.1.7.ii Interactions Between Queue and Server Limits

If the limits for a queue and the server are different, the more restrictive limit applies.

Example 5-19: More restrictive queue or server limit applies

Given the following:

- Server limit on running jobs for generic users is *10*
- Queue limit for running jobs from QueueA for generic users is *15*
- Queue limit for running jobs from QueueB for generic users is *5*

This means:

- Generic users at QueueA can run 10 jobs
- Generic users at QueueB can run 5 jobs

Example 5-20: More restrictive queue or server limit applies

Given the following:

- Bob's user limit on running jobs, set on the server, is *7*
- Bob's user limit on running jobs, set on QueueA, is *6*

This means:

- Bob can run 6 jobs from QueueA

5.15.1.8 Resource Usage Limit Attributes for Server and Queues

Each of the following attributes can be set at the server and each queue:

max_run
: The maximum number of jobs that can be running.

max_run_soft
: The soft limit on the maximum number of jobs that can be running.

max_run_res.<resource name>

The maximum amount of the specified resource that can be allocated to running jobs.

max_run_res_soft.<resource name>

The soft limit on the amount of the specified resource that can be allocated to running jobs.

max_queued

The maximum number of jobs that can be queued and running. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

max_queued_res.<resource name>

The maximum amount of the specified resource that can be allocated to queued and running jobs. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

queued_jobs_threshold

The maximum number of jobs that can be queued. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

queued_jobs_threshold_res.<resource name>

The maximum amount of the specified resource that can be allocated to queued jobs. At the server level, this includes all jobs in the complex. Queueing a job includes the `qsub` and `qmove` commands and the equivalent APIs.

Each attribute above can be used to specify all of the following:

- An overall limit (at the queue or server)
- A limit for generic users
- Individual limits for specific users
- A limit for generic projects
- Individual limits for specific projects
- A limit for generic groups
- Individual limits for specific groups

For example, you can specify the limits for the number of running jobs:

- In the complex:
  - The overall server limit (all usage in the entire complex) is *10,000*
  - The limit for generic users is *5*
  - The limit for Bob is *10*
  - The limit for generic groups is *50*
  - The limit for group GroupA is *75*
  - The limit for generic projects is *25*
  - The limit for Project A is *35*
- At QueueA:
  - The overall queue limit (all usage in QueueA) is *200*
  - The limit for generic users is *2*
  - The limit for Bob is *1*
  - The limit for generic groups is *3*
  - The limit for group GroupA is *7*
  - The limit for generic projects is *10*
  - The limit for Project A is *15*
- At QueueB:
  - The overall queue limit (all usage in QueueB) is *500*
  - The limit for generic users is *6*
  - The limit for Bob is *8*
  - The limit for generic groups is *15*
  - The limit for group GroupA is *11*
  - The limit for generic projects is *20*
  - The limit for Project A is *30*

5.15.1.9 How to Set Limits at Server and Queues

You can set, add, and remove limits by using the `qmgr` command to set limit attributes.

5.15.1.9.i Syntax

Format for setting a limit attribute:

set server <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],..."

set <queue> <queue name> <limit attribute> = "[limit-spec=<limit>], [limit-spec=<limit>],..."

Format for adding a limit to an attribute:

set server <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],..."

set <queue> <queue name> <limit attribute> += "[limit-spec=<limit>], [limit-spec=<limit>],..."

Format for removing a limit from an attribute; note that the value for <*limit*> need not be specified when removing a limit:

set server <limit attribute> -= "[limit-spec], [limit-spec],..."

set <queue> <queue name> <limit attribute> -= "[limit-spec], [limit-spec],..."

Alternate format for removing a limit from an attribute; note that the value of <*limit*> used when removing a limit must match the value of the limit:

set server <limit attribute> -= "[limit-spec=<limit>], [limit-spec=<limit>],..."

set <queue> <queue name> <limit attribute> -= "[limit-spec=<limit>], [limit-spec=<limit>],..."

where *limit-spec* specifies a user limit, a group limit, or an overall limit:

Table 5-13: Specifying Limits

Limit	limit-spec
Overall limit	*o:PBS_ALL*
Generic users	*u:PBS_GENERIC*
An individual user	*u:<username>*
Generic groups	*g:PBS_GENERIC*
An individual group	*g:<group name>*
Generic projects	p:PBS_GENERIC
An individual project	p:<project name>

The *limit-spec* can contain spaces anywhere except after the colon (":").

If there are comma-separated *limit-specs*, the entire string must be enclosed in double quotes.

A username, group name, or project name containing spaces must be enclosed in quotes.

If a username, group name, or project name is quoted using double quotes, and the entire string requires quotes, the outer enclosing quotes must be single quotes. Similarly, if the inner quotes are single quotes, the outer quotes must be double quotes.

PBS_ALL is a keyword which indicates that this limit applies to the usage total.

PBS_GENERIC is a keyword which indicates that this limit applies to generic users or groups.

When removing a limit, the limit value does not need to be specified.

PBS_ALL and *PBS_GENERIC* are case-sensitive.

5.15.1.9.ii Examples of Setting Server and Queue Limits

Example 5-21: To set the max_queued limit on QueueA to *5* for total usage, and to limit user bill to *3*:

```
Qmgr: s q QueueA max_queued = "[o:PBS_ALL=5], [u:bill =3]"
```

Example 5-22: On QueueA, set the maximum number of CPUs and the maximum amount of memory that user bill can request in his queued jobs:

```
Qmgr: s q QueueA max_queued_res.ncpus ="[u:bill=5]", max_queued_res.mem =
   "[u:bill=100mb]"
```

Example 5-23: To set a limit for a username with a space in it, and to set a limit for generic groups:

```
Qmgr: s q QueueA max_queued = '[u:"\PROG\Named User" = 1], [g:PBS_GENERIC=4]'
```

Example 5-24: To set a generic server limit for projects, and an individual server limit for Project A:

```
Qmgr: set server max_queued = '[p:PBS_GENERIC=6], [p:ProjectA=8]'
```

5.15.1.9.iii Examples of Adding Server and Queue Limits

Example 5-25: To add an overall limit for the maximum number of jobs that can be queued at QueueA to *10*:

```
Qmgr: s q QueueA max_queued += [o:PBS_ALL=10]
```

Example 5-26: To add an individual user limit, an individual group limit, and a generic group limit on queued jobs at QueueA:

```
Qmgr: s q QueueA max_queued += "[u:user1= 5], [g:GroupMath=5],[g:PBS_GENERIC=2]"
```

Example 5-27: To add a limit at QueueA on the number of CPUs allocated to queued jobs for an individual user, and a limit at QueueA on the amount of memory allocated to queued jobs for an individual user:

```
Qmgr: s q QueueA max_queued_res.ncpus += [u:tom=5], max_queued_res.mem += [u:tom=100mb]
```

Example 5-28: To add an individual server limit for Project B:

```
Qmgr: set server max_queued += [p:ProjectB=4]
```

5.15.1.9.iv Examples of Removing Server and Queue Limits

It is not necessary to specify the value of the limit when removing a limit, but you can specify the value of the limit.

Example 5-29: To remove the generic user limit at QueueA for queued jobs, use either of the following:

```
Qmgr: set queue QueueA max_queued -= [u:PBS_GENERIC]
Qmgr: set queue QueueA max_queued -= [u:PBS_GENERIC=2]
```

Example 5-30: To remove the limit on queued jobs at QueueA for *Named User*, use either of the following:

```
Qmgr: set queue QueueA max_queued -= [u:"\PROG\Named User"]
Qmgr: set queue QueueA max_queued -= [u:"\PROG\Named User"=1]
```

Example 5-31: To remove the limit at QueueA on the amount of memory allocated to an individual user, use either of the following:

```
Qmgr: set queue QueueA max_queued_res.mem -= [u:tom]
Qmgr: set queue QueueA max_queued_res.mem -= [u:tom=100mb]
```

To remove the limit on the number of CPUs allocated to queued jobs for user bill, use either of the following:

```
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill]
Qmgr: set queue QueueA max_queued_res.ncpus -= [u:bill=5]
```

Example 5-32: To remove a generic user limit and an individual user limit, use either of the following:

```
Qmgr: set queue QueueA max_queued - -= "[u:user1], [u:PBS_GENERIC]"
Qmgr: set queue QueueA max_queued -= "[u:user1=2], [u:PBS_GENERIC=4]"
```

Example 5-33: To remove the individual server limit for Project B, use either of the following:

```
Qmgr: set server max_queued -=[p:ProjectB]
Qmgr: set server max_queued -=[p:ProjectB=4]
```

5.15.1.10 Who Can Set Limits at Server and Queues

As with other server and queue attributes, only PBS Managers and Operators can set limit attributes.

5.15.1.11 Viewing Server and Queue Limit Attributes

5.15.1.11.i Printing Server and Queue Limit Attributes

You can use the qmgr command to print the commands used to set the limit attributes at the server or queue.

Example 5-34: To print all the limit attributes for queue QueueA:

```
Qmgr: p q QueueA max_queued, max_queued_res
#
# Create queues and set their attributes.
#
# Create and define queue QueueA
#
create queue QueueA
set queue QueueA max_queued =  "[o:PBS_ALL=10]"
set queue QueueA max_queued += "[u:PBS_GENERIC=2]"
set queue QueueA max_queued += "[u:bill=3]"
set queue QueueA max_queued += "[u:tom=15]"
set queue QueueA max_queued += "[u:user1=3]"
set queue QueueA max_queued += '[u:"\PROG\Named User"=1]'
set queue QueueA max_queued += "[g:PBS_GENERIC=2] "
set queue QueueA max_queued += "[g:GroupMath=5]"
set queue QueueA max_queued_res.ncpus = "[u:bill=5]"
set queue QueueA max_queued_res.ncpus += "[u:tom=5]"
set queue QueueA max_queued_res.mem = "[u:bill=100mb]"
set queue QueueA max_queued_res.mem += "[u:tom=100mb]"
```

5.15.1.11.ii Listing Server and Queue Limit Attributes

You can use the qmgr command to list the limit attributes for the queue or server.

Example 5-35: To list the max_queued and max_queued_res attributes for QueueA:

```
Qmgr: l q QueueA max_queued, max_queued_res
Queue: QueueA
    max_queued = [o:PBS_ALL=10]
    max_queued = [g:PBS_GENERIC=2]
    max_queued = [g:GroupMath=5]
    max_queued = [u:PBS_GENERIC=2]
    max_queued = [u:bill=3]
    max_queued = [u:tom=15]
    max_queued = [u:user1=3]
    max_queued = [u:"\PROG\Named User"=1]
    max_queued_res.ncpus = [u:bill=5]
    max_queued_res.ncpus = [u:tom=5]
    max_queued_res.mem = [u:bill=5]
    max_queued_res.mem = [u:bill=100mb]
    max_queued_res.mem = [u:tom=100mb]
```

5.15.1.11.iii Using the qstat Command to View Queue Limit Attributes

You can use the qstat command to see the limit attribute settings for the queue or server.

Example 5-36: To see the settings for the max_queued and max_queued_res limit attributes for QueueA using the qstat command:

```
qstat -Qf QueueA
Queue: QueueA
       ...
    max_queued = [o:PBS_ALL=10]
    max_queued = [g:PBS_GENERIC=2]
    max_queued = [g:GroupMath=5]
    max_queued = [u:PBS_GENERIC=2]
    max_queued = [u:bill=3]
    max_queued = [u:tom=3]
    max_queued = [u:cs=3]
    max_queued = [u:"\PROG\Named User"=1]
    max_queued_res.ncpus = [u:bill=5]
    max_queued_res.ncpus = [u:tom=5]
    max_queued_res.mem = [u:bill=5]
    max_queued_res.mem =[u:bill=100mb]
    max_queued_res.mem =[u:tom=100mb]
```

5.15.1.12 How Server and Queue Limits Work

Affected jobs are jobs submitted by the user or group, or jobs belonging to a project, whose limit has been reached. The following table shows what happens when a given limit is reached:

Table 5-14: Actions Performed When Limits Are Reached

Limit	Action
Running jobs	No more affected jobs are run at this server or queue until the number of affected running jobs drops below the limit.
Queued jobs	The queue does not accept any more affected jobs until the number of affected queued jobs drops below the limit. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue. If a job is requeued, the limit is ignored.
Resources for running jobs	The queue does not run any more affected jobs until the limit would not be exceeded if the next affected job were to start.
Resources for queued jobs	The queue does not accept any more affected jobs until the limit would not be exceeded if the next affected job were to start. Affected jobs submitted directly to the queue are rejected. Affected jobs in a routing queue whose destination is this queue remain in the routing queue.

5.15.1.13 Caveats and Advice for Server and Queue Limits

5.15.1.13.i Avoiding Overflow

On PBS server platforms for which the native size of a long is less than 64 bits, you should refrain from defining a limit on a resource of type long whose cumulative sum over all queued jobs would exceed the storage capacity of the resource variable. For example, if each submitted job were to request 100 hours of the cput resource, overflow would occur on a 32-bit platform when 5965 jobs (which is $(2^{31} - 1)/360000$ seconds) were queued.

5.15.1.13.ii Ensuring That Limits Are Effective

In order for limits to be effective, each job must specify each limited resource. This can be accomplished using defaults; see section 5.9.3, "Specifying Job Default Resources", on page 245. You can also use hooks; see the PBS Professional Hooks Guide.

5.15.1.13.iii Array Jobs

An array job with N subjobs is considered to consume N times the amount of resources requested when it was submitted. For example, if there is a server limit of 100 queued jobs, no user would be allowed to submit an array job with more than 100 subjobs.

5.15.1.13.iv Avoiding Job Rejection

Jobs are rejected when users, groups, or projects who have reached their limit submit a job in the following circumstances:

- The job is submitted to the execution queue where the limit has been reached
- The job is submitted to the complex, and the server limit has been reached

If you wish to avoid having jobs be rejected, you can set up a routing queue as the default queue. Set the server's default_queue attribute to the name of the routing queue. See section 2.3.6, "Routing Queues", on page 25.

5.15.1.13.v Do Not Mix Old And New Limits

The new limit attributes are incompatible with the old limit attributes. See section 5.15.1.15, "Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1", on page 302. You cannot mix the use of old and new resource usage limit attributes. This means that:

- If any old limit attribute is set, and you try to set a new limit attribute, you will get error 15141.
- If any new limit attribute is set, and you try to set an old limit attribute, you will get error 15141.

You must unset all of one kind in order to set any of the other kind.

5.15.1.13.vi Do Not Limit Running Time

Beware creating limits such as max_run_res.walltime or max_run_res.max_walltime. The results probably will not be useful. You will be limiting the amount of walltime that can be requested by running jobs for a user, group, or project. For example, if you set a walltime limit of 10 hours for group A, then group A cannot run one job requesting 5 hours and another job requesting 6 hours.

5.15.1.14 Errors and Logging for Server and Queue Limits

5.15.1.14.i Error When Setting Limit Attributes

Attempting to set a new limit attribute while an old limit attribute is set:

```
"use new/old qmgr syntax, not both"
"Attribute name <new> not allowed.  Older name <old> already set'
```

Attempting to set an old limit attribute while a new limit attribute is set:

```
"use new/old qmgr syntax, not both"
"Attribute name <old> not allowed:  Newer name <new> already set''
```

5.15.1.14.ii Logging Events

Whenever a limit attribute is set or modified, the server logs the event, listing which attribute was modified and who modified it.

Whenever a limit is reached, and would be exceeded by a job, the scheduler logs the event, listing the limit attribute and the reason.

5.15.1.14.iii Queued Limit Error Messages

When a limit for queued jobs or resources allocated to queued jobs is reached, the command involved presents a message. This command can be `qsub`, `qmove` or `qalter`.

5.15.1.14.iv Run Limit Error Messages

See "Run Limit Error Messages" on page 387 of the PBS Professional Reference Guide for a list of run limit error messages.

5.15.1.15 Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1

The old server and queue limit attributes discussed here existed in PBS Professional before Version 10.1. The old limit attributes continue to function as they did in PBS Professional 10.0. These attributes are incompatible with the limit attributes introduced in Version 10.1. See section 5.15.1.13.v, "Do Not Mix Old And New Limits", on page 301 and section 5.15.1.14.i, "Error When Setting Limit Attributes", on page 301.

The following table shows how the old limit attributes are used:

Table 5-15: Resource Usage Limits Existing Before Version 10.1

Limit	Overall Limit	Generic Users	Generic Groups	Indi-vidual Users	Indi-vidual Group
Maximum number of running jobs	max_running	max_user_run	max_group_run	N/A	N/A
Maximum number of running jobs (soft limit)	N/A	max_user_run_soft	max_group_run_soft	N/A	N/A
Maximum amount of specified resource allocated to running jobs	N/A	max_user_res	max_group_res	N/A	N/A
Maximum amount of specified resource allocated to running jobs (soft limit)	N/A	max_user_res_soft	max_group_res_soft	N/A	N/A
Maximum number of queued jobs	max_queuable	N/A	N/A	N/A	N/A
Maximum amount of specified resource allocated to queued jobs	N/A	N/A	N/A	N/A	N/A

5.15.1.15.i Precedence of Old Limits

If an old limit is defined at both the server and queue, the more restrictive limit applies.

5.15.1.15.ii Old Server Limits

For details of these limits, see "Server Attributes" on page 281 of the PBS Professional Reference Guide.

max_running
: The maximum number of jobs allowed to be selected for execution at any given time.

max_group_res,
max_group_res_soft
: The maximum amount of the specified resource that all members of the same Linux group may consume simultaneously.

max_group_run,
max_group_run_soft
: The maximum number of jobs owned by a Linux group that are allowed to be running from this server at one time.

max_user_res,
max_user_res_soft
: The maximum amount of the specified resource that any single user may consume.

max_user_run,
max_user_run_soft
: The maximum number of jobs owned by a single user that are allowed to be running at one time.

5.15.1.15.iii Old Queue Limits

For details of these limits, see "Queue Attributes" on page 311 of the PBS Professional Reference Guide.

max_group_res,
max_group_res_soft
: The maximum amount of the specified resource that all members of the same Linux group may consume simultaneously, in the specified queue.

max_group_run,
max_group_run_soft
: The maximum number of jobs owned by a Linux group that are allowed to be running from this queue at one time

max_queuable
: The maximum number of jobs allowed to reside in the queue at any given time. Once this limit is reached, no new jobs will be accepted into the queue.

max_user_res,
max_user_res_soft
: The maximum amount of the specified resource that any single user may consume in submitting to this queue.

max_user_run,
max_user_run_soft
: The maximum number of jobs owned by a single user that are allowed to be running at one time from this queue.

5.15.2 Placing Resource Limits on Jobs

Jobs are assigned limits on the amount of resources they can use. Each limit is set at the amount requested or allocated by default. These limits apply to how much the job can use on each vnode (per-chunk limit) and to how much the whole job can use (job-wide limit). Limits are derived from both requested resources and applied default resources. For information on default resources, see section 5.9.3, "Specifying Job Default Resources", on page 245.

Each chunk's per-chunk limits determine how much of any resource can be used in that chunk. Per-chunk resource usage limits are the amount of per-chunk resources requested, both from explicit requests and from defaults.

The consumable resources requested for chunks in the select specification are summed, and this sum makes a job-wide limit. Job resource limits from sums of all chunks override those from job-wide defaults and resource requests.

Job resource limits set a limit for per-job resource usage. Various limit checks are applied to jobs. If a job's job resource limit exceeds queue or server restrictions, it will not be put in the queue or accepted by the server. If, while running, a job exceeds its limit for a consumable or time-based resource, it will be terminated.

5.15.2.1 How Limits Are Derived

Job resource limits are derived in this order from the following:

1. Explicitly requested job-wide resources (e.g. `-l resource=value`)
2. The following built-in chunk-level resources in the job's select specification (e.g. `-l select =`...)

   accelerator_memory

   mem

   mpiprocs

   naccelerators

   ncpus

   nodect

   vmem
3. The server's default_qsub_arguments attribute
4. The queue's resources_default.<resource name>
5. The server's resources_default.<resource name>
6. The queue's resources_max.<resource name>
7. The server's resources_max.<resource name>

The server's default_chunk.<resource name> does **not** affect job-wide limits.

You can use a hook to set a per-chunk limit, using any hook that operates on jobs, such as a job submission hook, a modify job hook, etc.

5.15.2.2 Configuring Per-job Limits at Server and Queue

You can set per-job limits on the amount of each resource that any one job can use. You can set these limits at the server and at each queue. For example, you can specify the following limits:

- Jobs at the server can use no more than 48 hours of CPU time
- Jobs at QueueA can use no more than 12 hours of CPU time
- Jobs at QueueA must request more than 2 hours of CPU time

To set these limits, specify values for the server's resources_max.<resource name> attribute and each queue's resources_max.<resource name> and resources_min.<resource name> attributes. The server does not have a resources_min.<resource name> attribute. To set the maximum at the server, the format is:

Qmgr: set server resources_max.<resource name> = value

To set the maximum and minimum at the queue, the format is:

Qmgr: set queue <queue name> resources_max.<resource name> = value

Qmgr: set queue <queue name> resources_min.<resource name> = value

For example, to set the 48 hour CPU time limit:

```
Qmgr: set server resources_max.cput = 48:00:00
```

5.15.2.2.i Running Time Limits at Server and Queues

For non-shrink-to-fit jobs, you can set limits on walltime at the server or queue. To set a walltime limit for non-shrink-to-fit jobs at the server or a queue, use resources_max.walltime and resources min.walltime.

For shrink-to-fit jobs, running time limits are applied to max_walltime and min_walltime, not walltime. To set a running time limit for shrink-to-fit jobs, you cannot use resources_max or resources_min for max_walltime or min_walltime. Instead, use resources_max.walltime and resources_min.walltime. See section 4.9.42.6, “Shrink-to-fit Jobs and Resource Limits”, on page 213.

5.15.2.3 Configuring Per-job Resource Limit Enforcement at Vnodes

For a job, enforcement of resource limits is per-MoM, not per-vnode. So if a job requests 3 chunks, each of which has 1MB of memory, and all chunks are placed on one host, the limit for that job for memory for that MoM is 3MB. Therefore one chunk can be using 2 MB and the other two using 0.5MB and the job can continue to run.

Job resource limits can be enforced for single-vnode jobs, or for multi-vnode jobs that are using a PBS-aware MPI. See the following table for an overview. Memory limits are handled differently depending on the operating system. See "Job Memory Limit Enforcement on Linux” on page 306. The ncpus limit can be adjusted in several ways. See "Job ncpus Limit Enforcement” on page 307 for a discussion. The following table summarizes how resource limits are enforced at vnodes:

Table 5-16: Resource Limit Enforcement at Vnodes

Limit	What Determines When Limit Is Enforced	Scope of Limit	Enforcement Method
file size	automatically	per-process	`setrlimit()`
vmem	If job requests or inherits vmem	job-wide	MoM poll
pvmem	If job requests or inherits pvmem	per-process	`setrlimit()`
pmem	If job requests or inherits pmem	per-process	`setrlimit()`
pcput	If job requests or inherits pcput	per-process	`setrlimit()`
cput	If job requests or inherits cput	job-wide	MoM poll
walltime	If job requests or inherits walltime	job-wide	MoM poll
mem	if $enforce mem in MoM's config	job-wide	MoM poll
ncpus	if $enforce cpuaverage, $enforce cpuburst, or both, in MoM's config. See "Job ncpus Limit Enforcement” on page 307.	job-wide	MoM poll

5.15.2.4 Job Memory Limit Enforcement

You may wish to prevent jobs from swapping memory. To prevent this, you can set limits on the amount of memory a job can use. Then the job must request an amount of memory equal to or smaller than the amount of physical memory available.

PBS measures and enforces memory limits in two ways:

- On each host, by setting OS-level limits, using the limit system calls
- By periodically summing the usage recorded in the `/proc` entries.

Enforcement of mem is dependent on the following:

- Adding $enforce mem to the MoM's `config` file
- The job requesting or inheriting a default value for mem

You can configure default `qsub` parameters in the default_qsub_arguments server attribute, or set memory defaults at the server or queue. See section 5.9.3, "Specifying Job Default Resources", on page 245.

5.15.2.4.i Job Memory Limit Enforcement on Linux

By default, memory limits are not enforced. To enforce mem resource usage, put `$enforce mem` into MoM's `config` file, and set defaults for mem so that each job inherits a value if it does not request it.

The mem resource can be enforced at both the job level and the vnode level. The job-wide limit is the smaller of a job-wide resource request and the sum of that for all chunks. The vnode-level limit is the sum for all chunks on that host.

Job-wide limits are enforced by MoM polling the working set size of all processes in the job's session. Jobs that exceed their specified amount of physical memory are killed. A job may exceed its limit for the period between two polling cycles. See section 3.1.2, "Configuring MoM Polling Cycle", on page 34.

Per-process limits are enforced by the operating system kernel. PBS calls the kernel call `setrlimit()` to set the limit for the top process (the shell), and any process started by the shell inherits those limits. PBS does not know whether the kernel kills a process for exceeding the limit.

If a user submits a job with a job limit, but not per-process limits (`qsub -l cput=10:00`) then PBS sets the per-process limit to the same value. If a user submits a job with both job and per-process limits, then the per-process limit is set to the lesser of the two values.

Example: a job is submitted with `qsub -lcput=10:00`

- There are two CPU-intensive processes which use 5:01 each. The job will be killed by PBS for exceeding the cput limit. 5:01 + 5:01 is greater than 10:00.
- There is one CPU-intensive process which uses 10:01. It is very likely that the kernel will detect it first.
- There is one process that uses 0:02 and another that uses 10:00. PBS may or may not catch it before the kernel does depending on exactly when the polling takes place.

If a job is submitted with a pmem limit, or without pmem but with a mem limit, PBS uses the `setrlimit(2)` call to set the limit. For most operating systems, `setrlimit()` is called with `RLIMIT_RSS` which limits the Resident Set (working set size). This is not a hard limit, but advice to the kernel. This process becomes a prime candidate to have memory pages reclaimed.

If vmem is specified and no single process exceeds that limit, but the total usage by all the processes in the job does, then PBS enforces the vmem limit, but not the pvmem limit, and logs a message. PBS uses MoM polling to enforce vmem.

The limit for pmem is enforced if the job specifies, or inherits a default value for, pmem. When pmem is enforced, the limit is set to the smaller of mem and pmem. Enforcement is done by the kernel, and applies to any single process in the job.

The limit for pvmem is enforced if the job specifies, or inherits a default value for, pvmem. When pvmem is enforced, the limit is set to the smaller of vmem and pvmem. Enforcement is done by the kernel, and applies to any single process in the job.

The following table shows which OS resource limits can be used by each operating system.

Table 5-17: RLIMIT Usage in PBS Professional

OS	file	mem/pmem	vmem/pvmem	cput/pcput
Linux	RLIMIT_FSIZE	RLIMIT_RSS	RLIMIT_AS	RLIMIT_CPU

Note that `RLIMIT_RSS`, `RLIMIT_UMEM`, and `RLIMIT_VMEM` are not standardized (i.e. do not appear in the Open Group Base Specifications Issue 6).

5.15.2.4.ii Memory Enforcement on cpusets

There should be no need to do so: either the vnode containing the memory in question has been allocated exclusively (in which case no other job will also be allocated this vnode, hence this memory) or the vnode is shareable (in which case using *mem_exclusive* would prevent two CPU sets from sharing the memory). Essentially, PBS enforces the equivalent of *mem_exclusive* by itself.

5.15.2.5 Job ncpus Limit Enforcement

Enforcement of the ncpus limit (number of CPUs used) is available on all platforms. The ncpus limit can be enforced using average CPU usage, burst CPU usage, or both. By default, enforcement of the ncpus limit is off. See "$enforce <limit>" on page 241 of the PBS Professional Reference Guide.

5.15.2.5.i Average CPU Usage Enforcement

Each MoM enforces cpuaverage independently, per MoM, not per vnode. To enforce average CPU usage, put `$enforce cpuaverage` in MoM's `config` file. You can set the values of three variables to control how the average is enforced. These are shown in the following table.

Table 5-18: Variables Used in Average CPU Usage

Variable	Type	Description	Default
cpuaverage	Boolean	If present (=*True*), MoM enforces ncpus when the average CPU usage over the job's lifetime usage is greater than the specified limit.	*False*
average_trialperiod	integer	Modifies cpuaverage. Minimum job walltime before enforcement begins. Seconds.	*120*
average_percent_over	integer	Modifies cpuaverage. Percentage by which the job may exceed ncpus limit.	*50*
average_cpufactor	float	Modifies cpuaverage. ncpus limit is multiplied by this factor to produce actual limit.	*1.025*

Enforcement of cpuaverage is based on the polled sum of CPU time for all processes in the job. The limit is checked each poll period. Enforcement begins after the job has had average_trialperiod seconds of walltime. Then, the job is killed if the following is true:

(cput / walltime) > (ncpus * average_cpufactor + average_percent_over / 100)

5.15.2.5.ii CPU Burst Usage Enforcement

To enforce burst CPU usage, put `$enforce cpuburst` in MoM's `config` file. You can set the values of four variables to control how the burst usage is enforced. These are shown in the following table.

Table 5-19: Variables Used in CPU Burst

Variable	Type	Description	Default
cpuburst	Boolean	If present (=*True*), MoM enforces ncpus when CPU burst usage exceeds specified limit.	*False*
delta_percent_over	integer	Modifies cpuburst. Percentage over limit to be allowed.	*50*
delta_cpufactor	float	Modifies cpuburst. ncpus limit is multiplied by this factor to produce actual limit.	*1.5*
delta_weightup	float	Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing.	*0.4*
delta_weightdown	float	Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing.	*0.1*

MoM calculates an integer value called cpupercent each polling cycle. This is a moving weighted average of CPU usage for the cycle, given as the average percentage usage of one CPU. For example, a value of 50 means that during a certain period, the job used 50 percent of one CPU. A value of 300 means that during the period, the job used an average of three CPUs.

new_percent = *change_in_cpu_time**100 / *change_in_walltime*

weight = delta_weight[up|down] * walltime/*max_poll_period*

new_cpupercent = (*new_percent* * *weight*) + (*old_cpupercent* * (1-*weight*))

delta_weight_up is used if *new_percent* is higher than the old cpupercent value. delta_weight_down is used if *new_percent* is lower than the old cpupercent value. delta_weight_[up|down] controls the speed with which cpupercent changes. If delta_weight_[up|down] is 0.0, the value for cpupercent does not change over time. If it is 1.0, cpupercent will take the value of *new_percent* for the poll period. In this case cpupercent changes quickly.

However, cpupercent is controlled so that it stays at the greater of the average over the entire run or ncpus*100.

max_poll_period is the maximum time between samples, set in MoM's `config` file by $max_check_poll, with a default of 120 seconds.

The job is killed if the following is true:

new_cpupercent > ((ncpus * 100 * delta_cpufactor) + delta_percent_over)

The following entries in MoM's `config` file turn on enforcement of both average and burst with the default values:

```
$enforce cpuaverage
$enforce cpuburst
$enforce delta_percent_over 50
$enforce delta_cpufactor 1.05
$enforce delta_weightup 0.4
$enforce delta_weightdown 0.1
$enforce average_percent_over 50
$enforce average_cpufactor 1.025
$enforce average_trialperiod 120
```

The cpuburst and cpuaverage information show up in MoM's log file, whether or not they have been configured in mom_priv/config. This is so a site can test different parameters for cpuburst/cpuaverage before enabling enforcement. You can see the effect of any change to the parameters on your job mix before "going live".

Note that if the job creates a child process whose usage is not tracked by MoM during its lifetime, CPU usage can appear to jump dramatically when the child process exits. This is because the CPU time for the child process is assigned to its parent when the child process exits. MoM may see a big jump in cpupercent, and kill the job.

5.15.2.5.iii Job Memory Limit Restrictions

Enforcement of mem resource usage is available on all Linux platforms, but not Windows.

5.15.2.6 Changing Job Limits

The qalter command is used to change job limits, with these restrictions:

- A non-privileged user may only lower the limits for job resources
- A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as pcput and pmem, because these are set when the process starts, and enforced by the kernel.
- When you lengthen the walltime of a running job, make sure that the new walltime will not interfere with any existing reservations etc.

See "qalter" on page 127 of the PBS Professional Reference Guide.

5.15.3 Limiting the Number of Jobs in Queues

If you limit the number of jobs in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the queued_jobs_threshold queue or server attribute. See section 5.15.1.9, "How to Set Limits at Server and Queues", on page 296.

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See section 4.9.39, "Routing Jobs", on page 205.

5.16 Where Resource Information Is Kept

Definitions and values for PBS resources are kept in the following files, attributes, and parameters. Attributes specifying resource limits are not listed here. They are listed in section 5.15.1.8, "Resource Usage Limit Attributes for Server and Queues", on page 294 and section 5.15.1.15, "Old Limit Attributes: Server and Queue Resource Usage Limit Attributes Existing Before Version 10.1", on page 302.

5.16.1 Files

<sched_priv directory>/sched_config

resources: line

In order for scheduler to be able to schedule using a resource, the resource must be listed in the **resources:** line. Format:

resources: "<resource name>, [<resource name>, ...]"

Example:

```
resources: "ncpus, mem, arch, [...], LocalScratch, FloatLicense, SharedScratch"
```

The only exception is host-level Boolean resources, which do not need to appear in the `resources:` line.

`server_dyn_res:` line

Each dynamic server resource must be listed in its own `server_dyn_res:` line. Format:

server_dyn_res: "<resource name> !<path to script/command>"

Example:

```
server_dyn_res: "SharedScratch !/usr/local/bin/serverdynscratch.pl"
```

`mom_resources:` line (**deprecated** as of 18.2.1)

Dynamic host resources must be listed in the `mom_resources:` line. Format:

mom_resources: "<resource name>"

Example:

```
mom_resources: "LocalScratch"
```

PBS_HOME/mom_priv/config

Contains MoM configuration parameters and any local resources. Format:

<resource name> !<path to script/command>

Example:

```
LocalScratch !/usr/local/bin/localscratch.pl
```

See "MoM Parameters" on page 239 of the PBS Professional Reference Guide.

Version 2 Configuration Files

Contain vnode information. See section 3.4.3, "Version 2 Vnode Configuration Files", on page 42.

5.16.2 MoM Configuration Parameters

$cputmult <factor>

This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MoM's system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

```
$cputmult 1.5
```

If MoM's system is slower, set factor to a value between *1.0* and *0.0*. For example:

```
$cputmult 0.75
```

$wallmult <factor>

Each job's walltime usage is multiplied by this factor. For example:

```
$wallmult 1.5
```

5.16.3 Attributes

Resources are tracked in the following attributes:

Table 5-20: Attributes Where Resources Are Tracked

Resource Being Tracked	Attribute Name			
	Server and Queue	Vnode	Job	Reservation
Amount of each resource available for use at the object (server, queue, vnode)	resources_available.<resource name>	resources_available.<resource name>		
Amount of each resource allocated to jobs running and exiting at the object (server, queue, vnode)	resources_assigned.<resource name>	resources_assigned.<resource name>		
Amount of each resource used by the job			resources_used.<resource name>	
Amount of each job-wide resource that is assigned to any job that does not explicitly request the resource	resources_default.<resource name>			
Amount of each host-level resource that is assigned to each chunk of any job where that does not explicitly request the resource	default_chunk.<resource name>			
List of resources requested by the object (job or reservation)			Resource_List.<resource name>	Resource_List.<resource name>
List of chunks for the job. Each chunk shows the name of the vnode from which it is taken along with the host-level, consumable resources allocated from that vnode.			exec_vnode	
List of vnodes and resources allocated to them to satisfy the chunks requested for this reservation or occurrence				resv_nodes

5.17 Viewing Resource Information

You can see attribute values of resources for the server, queues, and vnodes using the `qmgr` or `pbsnodes` commands. The value in the server, queue, or vnode resources_assigned attribute is the amount explicitly requested by running and exiting jobs and, at the server and vnodes, started reservations.

You can see job attribute values using the `qstat` command. The value in the job's Resource_List attribute is the amount explicitly requested by the job. See "Resources Requested by Job" on page 245 in the PBS Professional Administrator's Guide.

The following table summarizes how to find resource information:

Table 5-21: How to View Resource Information

Location	Item to View	Command
server	default_chunk, default_qsub_arguments, resources_available, resources_assigned, resources_default	`qmgr`, `qstat`, `pbsnodes`
scheduler	`sched_config file`	Favorite editor or viewer
queues	default_chunk, resources_available, resources_assigned, resources_default	`qmgr`, `qstat`
MoM and vnodes	resources_available, sharing, pcpus, resources_assigned	`qmgr`, `pbsnodes`
	`mom_config file`	Favorite editor or viewer
job	Resource_List	`qstat`
reservation	Resource_List	pbs_rstat -f
accounting	`resources_assigned` entry in accounting log	Favorite editor or viewer

Every consumable resource, for example mem, can appear in four PBS attributes. These attributes are used in the following elements of PBS:

Table 5-22: Values Associated with Consumable Resources

Attribute	Vnode	Queue	Server	Accounting Log	Job	Scheduler
resources_available	Yes	Yes	Yes			Yes
resources_assigned	Yes	Yes	Yes	Yes		
resources_used				Yes	Yes	Yes
Resource_List					Yes	Yes

5.17.1 Resource Information in Accounting Logs

For a complete description of the resource information in the PBS accounting logs, see Chapter 19, "Accounting", on page 627.

5.17.2 Resource Information in Daemon Logs

At the end of each job, the server logs the values in the job's resources_used attribute, at event class 0x0010.

Upon startup, MoM logs the number of CPUs reported by the OS, at event class 0x0002.

At the end of each job, the MoM logs cput and mem used by each job, and cput used by each job task, at event class 0x0100.

5.17.3 Finding Current Value

You can find the current value of a resource by subtracting the amount being used from the amount that is defined.

Use the `qstat -Bf` command, and grep for resources_available.<resource name> and resources_used.<resource name>. To find the current amount not being used, subtract resources_used.<resource name> from resources_available.<resource name>.

5.17.4 Restrictions on Viewing Resources

- Dynamic resources shown in `qstat` do not display the current value, they display the most recent retrieval. Dynamic resources have no resources_available.<resource name> representation anywhere in PBS.
- Local static host-level resources cannot be viewed via `qstat` or managed via `qmgr`.

5.18 Resource Recommendations and Caveats

- It is not recommended to set the value for resources_available.ncpus. The exception is when you want to oversubscribe CPUs. See section 9.6.5.1.iii, "How To Share CPUs", on page 446.
- It is not recommended to change the value of ncpus at vnodes on a multi-vnoded machine.
- If you want to limit how many jobs are run, or how much of each resource is used, use the new limits. See section 5.15, "Managing Resource Usage", on page 287.
- It is not recommended to create local host-level resources by defining them in the MoM configuration file.
- Do not attempt to set values for resources_available.<resource name> for dynamic resources.
- Externally-managed application licenses may not be available when PBS thinks they are. PBS doesn't actually check out externally-managed licenses; the application being run inside the job's session does that. Between the time that the scheduler queries for licenses, and the time the application checks them out, another application may take the licenses. In addition, some applications request varying amounts of tokens during a job run.
- Jobs may be placed on different vnodes from those where they would have run in earlier versions of PBS. This is because a job's resource request will no longer match the same resources on the server, queues and vnodes.
- While users cannot request custom resources that are created with the `r` flag, jobs can inherit these as defaults from the server or queue resources_default.<resource name> attribute.
- A `qsub` or `pbs_rsub` hook does not have resources inherited from the server or queue resources_default or default_chunk as an input argument.
- Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag, whether that resource was requested by the user or came from default_qsub_arguments. Be aware that creating custom resources with permission flags and then using these in the default_qsub_arguments server attribute can cause jobs to be rejected. See section 5.14.2.3.vi, "Resource Permission Flags", on page 260.
- Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be underused. These resources are tracked automatically by scheduler.
- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  `pbsnodes`

  Users cannot view restricted host-level custom resources.

  `pbs_rstat`

  Users cannot view restricted reservation resources.

`pbs_rsub`

Users cannot request restricted custom resources for reservations.

`qalter`

Users cannot alter a restricted resource.

`qmgr`

Users cannot print or list a restricted resource.

`qselect`

Users cannot specify restricted resources via -l Resource_List.

`qsub`

Users cannot request a restricted resource.

`qstat`

Users cannot view a restricted resource.

- Do not set values for any resources, except those such as shared scratch space or floating application licenses, at the server or a queue, because the scheduler will not allocate more than the specified value. For example, if you set resources_available.walltime at the server to *10:00:00*, and one job requests 5 hours and one job requests 6 hours, only one job will be allowed to run at a time, regardless of other idle resources.
- If a job is submitted without a request for a particular resource, and no defaults for that resource are set at the server or queue, and either the server or queue has resources_max.<resource name> set, the job inherits that maximum value. If the queue has resources_max.<resource name> set, the job inherits the queue value, and if not, the job inherits the server value.
- When setting global static vnode resources on multi-vnode machines, follow the rules in section 3.4.5, "Configuring Vnode Resources", on page 47.
- Do not create custom resources with the same names or prefixes that PBS uses when you create custom resources for specific systems.
- Do not set resources_available.place for a vnode.
- Using dynamic host-level resources can slow the scheduler down, because the scheduler must wait for each resource-query script to run.
- On the parent vnode, all values for resources_available.<resource name> should be *zero* (*0*), unless the resource is being shared among child vnodes via indirection.
- Default `qsub` arguments and server and queue defaults are applied to jobs at a coarse level. Each job is examined to see whether it requests a select and a place. This means that if you specify a default placement, such as *excl*, with `-lplace=excl`, and the user specifies an arrangement, such as *pack*, with `-lplace=pack`, the result is that the job ends up with `-lplace=pack`, NOT `-lplace=pack:excl`. The same is true for select; if you specify a default of `-lselect=2:ncpus=1`, and the user specifies `-lselect=mem=2GB`, the job ends up with `-lselect=mem=2GB`.

6 Managing Power Usage

6.1 Monitoring and Controlling Job Power Usage

6.1.1 Power Provisioning: Monitoring and Controlling Job Power Usage

PBS Professional can control and monitor job power usage. PBS can assign a power profile for each job at submission time to control the job's power draw, and jobs can request power profiles.

PBS collects energy consumption information and records it in each job's resources_used.energy value. PBS provides information about job energy usage in the output of the `qstat` command, and records energy usage in the accounting logs.

For each job or power profile change, PBS provisions each vnode with the required power profile. The eoe resource represents one or more power profiles on each vnode. Each vnode's current_eoe attribute shows its current power profile.

Jobs can request a power profile, by requesting a value for the eoe resource, or PBS can set each job's eoe request. When a job requests a power profile, it is sent to vnodes that have this profile available, and when the job runs, the vnodes where the job runs are set to the power profile requested by the job.

The default power setting for a node is no power capping. If a job runs on a node, the node uses the requested power profile, but when the job finishes, the node goes back to the default setting.

6.1.1.1 Monitoring Power Use by Jobs

To see the power used by a job, you can use `qstat` to examine the job's resources_used.energy.

6.1.2 Platforms Supporting Power Provisioning

Power provisioning is supported on Cray XC machines and HPE 8600 machines with HPE Performance Cluster Manager (HPCM). It is possible but unsupported to use the feature on other platforms that supply a power management interface.

6.1.3 Power Provisioning on Cray XC

On Cray XC systems, you use the PBS_power hook to provision vnodes with the power requested by jobs, and to monitor job power usage. This is the same hook used for powering nodes up and down and limiting ramp rate. See section 6.2.1, “Managing Node Power on Cray XC with PBS”, on page 320 and section 6.2.2, “Limiting Ramp Rate for Node Power on Cray XC”, on page 320.

6.1.3.1 Overview of Power Provisioning on Cray

- You find out what power settings each vnode can support.
- You choose what power settings will make up each vnode power profile.
- You set the values for resources_available.eoe on vnodes.
- You enable the power provisioning hook.
- You enable power provisioning at specific vnodes.
- Each job requests or is assigned a power profile.
- PBS provisions each vnode with the requested power settings.

6.1.3.2 Selecting Power Profiles

You get the supported power capabilities of each compute node by running the `capmc` tool with the argument "`get_power_cap_capabilities`". Each node reports ranges for its pstate, pgov, pcap_node, and pcap_accelerator power settings.

You choose the power profiles you want, by selecting values for the pstate, pgov, pcap_node, and pcap_accelerator attributes for each power profile. Note that some settings for pcap_node may cause pstate to be ignored.

6.1.3.3 Setting Vnode Power Resources and Attributes

You specify the desired power profiles for each vnode by setting the vnode's resources_available.eoe to the list of profiles available at that vnode. For example:

```
Qmgr: set node compute_node1 resources_available.eoe="low,med"
Qmgr: set node compute_node2 resources_available.eoe="med,high"
```

Once a job is submitted and the hook sets the job's attributes, the process of setting power profile attributes on vnodes is handled automatically by PBS when PBS provisions each vnode with the requested values for the power attributes. PBS tells the Cray what settings to use on each vnode.

6.1.3.4 Setting Job Power Resource Requests

Each job can request or be assigned one of the power profiles you have defined.

You write a queuejob hook that sets each job's power attributes to the correct values for the requested profile. The hook maps each profile to a value for each of the power attributes. For example, if a compute node supports the following settings:

pstate range = 100-300

pgov range = 50-150

pcap_node range = 250-500

pcap_accelerator range = 500-1000

and you want to have profiles named "low", "med", and "high", the following table shows sample power profile settings:

Table 6-1: Sample Power Profile Settings

Power Profile	pstate	pgov	pcap_node	pcap_accelerator
low	100	50	250	500
med	200	100	350	700
high	300	150	500	1000

6.1.3.4.i Writing Power Profile Hook for Cray

Create a queuejob hook that sets each job's attributes to reflect its requested profile. Set each job's desired power profile by setting any of the following job attributes in the hook:

- Set the pcap_node job attribute to the value corresponding to the Cray `capmc set_power_cap --node` setting
- Set the pstate job attribute to the corresponding Cray ALPS p-state setting. Note that pcap_node takes precedence, and some settings for pcap_node can result in pstate being ignored.
- Set the pcap_accelerator job attribute to the value corresponding to the Cray `capmc set_power_cap --accel` setting
- Set the pgov job attribute for CPU throttling to the corresponding setting for p-governor

Example 6-1: We will use three profiles called "low", "med", and "high", and we will set pcap_node and pstate for each job requesting a profile:

```
import pbs

e = pbs.event()
j = e.job

profile = j.Resource_List['eoe']
if profile is None:
    res = j.Resource_List['select']
    if res is not None:
        for s in str(res).split('+')[0].split(':'):
            if s[:4] == 'eoe=':
                profile = s.partition('=')[2]
                break

pbs.logmsg(pbs.LOG_DEBUG, "got profile '%s'" % str(profile))
if profile == "low":
    j.Resource_List["pstate"] = "1900000"
    j.Resource_List["pcap_node"] = 100
    pbs.logmsg(pbs.LOG_DEBUG, "set low")
elif profile == "med":
    j.Resource_List["pstate"] = "220000"
    j.Resource_List["pcap_node"] = 200
    pbs.logmsg(pbs.LOG_DEBUG, "set med")
elif profile == "high":
    j.Resource_List["pstate"] = "240000"
    pbs.logmsg(pbs.LOG_DEBUG, "set high")
else:
    pbs.logmsg(pbs.LOG_DEBUG, "unhandled profile '%s'" % str(profile))

e.accept()
```

6.1.3.5 Enabling Power Provisioning on Cray

Enable power provisioning:

1. On each vnode where you want power provisioning enabled, set the power_provisioning vnode attribute to *True*:
   Qmgr: set node <node name> power_provisioning=true
2. Enable the PBS_power hook:

   Qmgr: set pbshook PBS_power enabled=true

6.1.3.6 Reporting Energy Usage on Cray

On Cray, PBS collects energy consumption information using `capmc` or `RUR`.

If `RUR` is configured and the energy reported by `RUR` is different from what is reported by `capmc`, PBS reports the larger value for the job.

Using `RUR` is optional. If you want PBS to collect energy information from `RUR`, enable `RUR` using the PBS output plugin. You must modify the `RUR` configuration file to use the PBS output plugin. This plugin is located in `/opt/pbs/lib/cray/pbs_output.py`

See the Cray instructions for this procedure in *CLE XC™ System Administration Guide S-2393-5204xc*:

`http://docs.cray.com/books/S-2393-5204xc//S-2393-5204xc.pdf`

See especially the following:

12.6.2 RUR Configuration File

12.7 RUR Plugins

12.7.2 Output Plugins

6.1.3.7 Caveats for Power Provisioning on Cray

Note that pcap_node takes precedence, and some settings for pcap_node can result in pstate being ignored.

6.1.4 Power Provisioning on HPE

6.1.4.1 Overview of Power Provisioning on HPE

On HPE, PBS uses the power API for HPE Performance Cluster Manager (HPCM). PBS handles querying for available power profiles and setting the eoe resource to the available power profiles. You enable power provisioning at the server and vnodes.

6.1.4.2 Setting Power Profiles on HPE

On each vnode, PBS queries the HPE Performance Cluster Manager (HPCM) for available power profiles, and sets the vnode's resources_available.eoe to the power profiles available on that vnode.

6.1.4.3 Enabling Power Provisioning on HPE

To enable power provisioning:

1. On each vnode where you want power provisioning enabled, set the power_provisioning vnode attribute to *True*:
   Qmgr: set node <node name> power_provisioning=true
2. Enable the PBS_power hook:
   Qmgr: set pbshook PBS_power enabled=true
3. HUP each MoM. If the vnode does not report resources_available.eoe, HUP the MoM again.

6.1.5 Terminology for Power Provisioning

Activate a power profile

To set a power profile on a node, for example, to set a node's power profile to match the specifications for "low".

Deactivate a power profile

To reset the power profile of the node to its default setting, which is no power capping.

6.1.6 Caveats and Restrictions for Using Power Profiles

- Make sure that your power provisioning queuejob hook takes into account your desired order of precedence for an explicit request for pcap_node and/or pstate versus a request for a profile. You may want users to be able to override power profile settings for pcap_node and/or pstate, or you may want profiles to override explicit requests.
- You cannot use power profiles on any hosts where the PBS server and/or scheduler are running.
- You cannot suspend jobs on vnodes that are using power profiles, meaning that you cannot use preemption via suspension on vnodes that are using power profiles.
- If you disable the PBS_power hook while a job is running, the vnodes where the job runs do not have their profile deactivated when the job finishes, and the job's resources_used.energy value is not set at the end of the job.
- A prologue script will not run when the PBS_power hook is enabled. Any prologue script must be converted to an execjob_prologue hook.
- If a job does not request a value for eoe, there is no activation of a power profile on a node, but the job's resources_used.energy is still calculated.
- If a job requests values for both aoe and eoe, PBS addresses the aoe request first.
- There is no PBS interface to RUR that can be used by administrators or job submitters.
- If pbs.conf is not in /etc on a host, add PBS_CONF_FILE to the PBS_HOME/pbs_environment file for that host, and set it to the path to pbs.conf on the host. For example, if /var/pbs.conf is the location of the pbs.conf file, add the following line to PBS_HOME/pbs_environment:
  PBS_CONF_FILE=/var/pbs.conf
- PBS does not automatically set resources_available.eoe on machines that host the PBS server/scheduler.
- PBS can provide precise power consumption accounting only where jobs are allocated exclusively. Vnodes must have the sharing attribute set so that jobs get exclusive use of the vnode, or jobs must request exclusive use of vnodes.

6.2 Managing Node Power on Cray XC

You can manage node power and limit the ramp rate on Cray XC* systems running CLE 6.0 and later. You can use the *PBS_power* hook to manage node power and to limit ramp rate on Cray XC systems. This is the same hook used on Cray XC for provisioning vnodes with the power requested by jobs and monitoring job power usage. See section 6.1.3, Power Provisioning on Cray XC , on page 315. PBS_power is a built-in periodic server hook. It takes a list of nodes to power up and manages power ramp rate, and communicates with vendor power APIs through a generic PMI interface. This hook powers up or down no more than max_concurrent_nodes nodes at once. When this hook is enabled, PBS sets the power_provisioning server attribute to *True*.

6.2.1 Managing Node Power on Cray XC with PBS

This version of PBS Professional allows you to automatically power down nodes that are not being used, and then power them up again only when they are needed for a job or reservation.

PBS periodically checks to see which nodes should be powered down, and if a node has been idle for the minimum specified amount of time, PBS powers down the node. You can specify the amount of node idle time that triggers powering down a node. PBS also periodically checks to see which nodes should be powered up so that a job or reservation can use them.

You can specify which nodes are eligible for power management by PBS, and the maximum number of nodes to power down or up at one time.

In addition, PBS minimizes the number of power cycles experienced by nodes, because power cycling a node puts a strain on the hardware. To do this, PBS avoids powering up nodes that have just been powered down.

6.2.1.1 Using the PBS_power hook for Managing Node Power on Cray

PBS manages node power through the *PBS_power* hook. The hook takes a list of nodes and communicates with vendor power APIs through a generic PMI interface to manage node power.

You configure node power management via server and vnode attributes. To enable this feature, enable the PBS_power hook.

You specify which nodes are eligible for power management by PBS via the power_provisioning vnode attribute.

You set the minimum amount of idle time for a node to be powered down in the node_idle_limit hook configuration parameter. Set the limit on the number of nodes that can be powered up or down at one time in the max_concurrent_nodes hook configuration parameter. Specify the minimum amount of time a node is powered down in the min_node_down_delay hook configuration parameter.

PBS records the time when a node is powered off in the last_state_change_time vnode attribute, and tracks the node's idle time by updating the last_used_time vnode attribute at the end of any job or reservation involving the node.

When a job or reservation is scheduled on a vnode, PBS sets the the job's estimated.start_time attribute. The hook calculates how much time it needs to power up all the nodes required. Using the Cray rules for powering a node up, PBS gives a node 600 seconds to power up. The PBS_power hook powers up nodes in batches of max_concurrent_nodes, with 600 seconds between each batch.

6.2.2 Limiting Ramp Rate for Node Power on Cray XC

PBS allows you to limit the ramp rate for the Cray system, by communicating with the `capmc` utility. The PBS_power power management hook controls the number of nodes whose C-state can be changed.

When ramping nodes up or down, the hook steps through sleep states.

6.2.2.1 Using the PBS_power for Managing Ramp Rate on Cray

You can manage the ramp rate behavior of the power management hook by setting parameters in the PBS_power hook configuration file. The hook configuration file contains parameters the hook uses as guides for its behavior, and parameters that the hook uses when it communicates with `capmc`. This file must conform to JSON syntax.

We show a sample file in section 6.2.2.2, “Sample Power Management Configuration File”, on page 321.

You can also export and look at the installed PBS power management hook configuration file:

```
qmgr -c "export pbshook PBS_power application/x-config default" > PBS_power.json
```

You can edit this file and change its parameters, then read it back in:

```
qmgr -c "import pbshook PBS_power application/x-config default PBS_power.json"
```

6.2.2.2 Sample Power Management Configuration File

Here is a sample configuration file, showing the ramp rate parameters:

```
{
"power_ramp_rate_enable": "True",
"power_on_off_enable": "False",
"node_idle_limit": "1000",
"min_node_down_delay": "600",
"max_jobs_analyze_limit": "80",
"max_concurrent_nodes": "5"
}
```

6.2.3 Power Management Hook Configuration Parameters

Table 6-2: Hook Configuration Parameters

Configuration File Parameter	Default Value	Description
power_ramp_rate_enable	*False*	Enables ramp rate limiting for the system. Nodes can be ramped up to sleep state C1 and ramped down to sleep state C6.
power_on_off_enable	*False*	Enables powering nodes on and off for nodes where the poweroff_eligible vnode attribute is *True*.
node_idle_limit	*1800*	Length of time in seconds for a node to be left idle before it is considered for powering down or ramping down.
min_node_down_delay	*1800*	Length of time in seconds for a node that is powered off to remain so until it is considered for powering up or ramping up.
max_jobs_analyze_limit	*100*	The maximum number of jobs that are analyzed for power-on or ramp-up. The jobs considered here are those which have a current estimated.start_time and exec_vnode. For these attributes to be current, set the strict_ordering scheduler parameter to *True*, and make sure jobs are submitted with a specification for walltime.
max_concurrent_nodes	*5*	Limit on number of nodes that can be powered on or off or ramped up or down at a time.

6.3 Power Management Attributes, Resources, Etc.

energy

Resource. Consumable. PBS records the job's energy usage in the job's resources_used.energy.

Format: *float*

Units: *kWh*

eoe

Resource. Stands for "Energy Operational Environment". Non-consumable. When set on a vnode in resources_available.eoe, contains the list of available power profiles. When set for a job in Resource_List.eoe, can contain at most one power profile. (A job can request only one power profile.) Automatically added to `resources:` line in `sched_config`.

Default value for resources_available.eoe: *unset*

Format: *string_array*

current_eoe

Vnode attribute. Shows the current value of eoe on the vnode.

Visible to all. Settable by manager. We do not recommend setting this attribute manually.

Format: *string*

Default: *unset*

last_state_change_time

Vnode attribute. Records the most recent time that this node changed state.

Set by PBS. Readable by Manger and Operator.

Format: integer seconds since epoch

Default: no default

last_used_time

Vnode attribute. Records the most recent time that this node finished being used for a job or reservation.

Set at creation or reboot time. Updated when node is released early from a running job. Reset when node is ramped up.

Set by PBS. Readable by Manger and Operator.

Format: integer seconds since epoch

Default: no default

max_concurrent_nodes

Hook configuration parameter. Specifies the maximum number of nodes that can be powered up or down at one time. Enabled when the the PBS_power hook is enabled. Used by the PBS_power hook.

set by Manager and Operator. Readable by all.

Format: positive integer

Default: *5*

min_node_down_delay

Hook configuration parameter. Specifies the minimum time a node is powered down before it can be powered back up. Enabled when the the PBS_power hook is enabled.

set by Manager and Operator. Readable by all.

Format: integer seconds

Default: *1800 seconds*

node_idle_limit

Hook configuration parameter. Specifies the minimum idle time for a node to be considered for powering down. Enabled when the the PBS_power hook is enabled.

set by Manager and Operator. Readable by all.

Format: integer seconds

Default: *1800 seconds*

pstate

Job attribute. Cray ALPS reservation setting for CPU frequency corresponding to p-state. See BASIL 1.4 documentation.

Settable by and visible to all PBS users.

Units: *hertz*

Format: *string*

Default: *unset*

Example: *pstate = 2200000*

pgov

Job attribute. Cray ALPS reservation setting for CPU throttling corresponding to p-governor. See BASIL 1.4 documentation. We do not recommend using this attribute.

Visible to all. Settable by all.

Format: *string*

Default: *unset*

pcap_node

Job attribute. Power cap for a node. Corresponds to Cray `capmc set_power_cap --node` setting. See `capmc` documentation.

Visible to and settable by all.

Units: *watts*

Format: *int*

Default: *unset*

pcap_accelerator

Job attribute. Power cap for an accelerator. Corresponds to Cray `capmc set_power_cap --accel` setting. See `capmc` documentation.

Visible to and settable by all.

Units: *watts*

Format: *int*

Default: *unset*

poweroff_eligible

Vnode attribute. Specifies whether this node is eligible to have its power managed by PBS.

set by Manager. Readable by all.

Format: Boolean

Default: False (not eligible)

power_provisioning

Server attribute. Reflects use of power profiles and managing node power via PBS. Set by PBS to *True* when the PBS_power hook is enabled.

Set by PBS. Read-only.

Format: Boolean

Default: *unset*, which behaves like *False* (not enabled)

power_provisioning

Vnode attribute. Specifies whether this node is eligible to have its power managed by PBS, including whether it can use power profiles.

set by Manager. Readable by all.

Format: Boolean

Default: *False* (not eligible)

sleep

Vnode state. Indicates that this vnode was ramped down or powered off via PBS power management. This tells the scheduler that it can schedule jobs on this vnode; in that case PBS powers the vnode back up.

6.4 Caveats and Restrictions for Power Management

- If a reservation is created with a start time coming up soon, where the reservation requires nodes that are currently powered off, the reservation may start in degraded mode until all of the nodes can be powered up.
- Do not set resources_available.eoe on vnodes. This is handled by PBS.

7 Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

7.1 Introduction

You can configure vnodes so that PBS will automatically install the OS or application that jobs need in order to run on those vnodes. For example, you can configure a vnode that is usually running RHEL to run SLES instead whenever the Physics group runs a job requiring SLES. If a job requires an application that is not usually installed, PBS can install the application in order for the job to run.

You can use provisioning for booting multi-boot systems into the desired OS, downloading an OS to and rebooting a diskless system, downloading an OS to and rebooting from disk, instantiating a virtual machine, etc. You can also use provisioning to run a configuration script or install an application.

7.2 Definitions

AOE

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

Master Provisioning Script, Master Script

The script that makes up the provisioning hook

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioning Hook

The hook which performs the provisioning, either by calling other scripts or running commands

Provisioning Tool

A tool that performs the actual provisioning, e.g. HPE Performance Cluster Manager (HPCM)

Provisioned Vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

7.3 How Provisioning Can Be Used

- Each application requires specific version of OS

The site runs multiple applications, and each application may be certified to run on a specific OS. In this situation, a job that will run an application requiring a specific OS requests the OS, and PBS provisions the required OS.

- The site needs differently configured images of the same OS to be loaded at different times

  The site has multiple projects, and each project requires the OS to be configured in a different way on a group of hosts. In this situation, PBS provisions groups of hosts with the correct OS image, for the time period needed by each project. The OS image is configured and supplied by the site administrator.

- The entire site needs different OSes at different times of day

  The entire site runs one OS during certain hours, and a different OS at other times.

- A user reserves multiple vnodes running the same version of an OS

  A user may need a specific version of an OS for a period of time. For example, a user needs 5 nodes running a specific version of RHEL from 5pm Friday until 5am Monday.

- The administrator wants to limit the number of hosts that are being provisioned at any one time, for any of the following reasons:
  - The network can become overwhelmed transferring OS images to execution nodes
  - The hosts can draw excessive power if many are powering up at the same time
  - Some sites notify the administrator whenever an execution node goes down, and when several vnodes are provisioned, the administrator is paged repeatedly

7.4 How Provisioning Works

7.4.1 Overview of Provisioning

PBS allows you to create a provisioning hook, which is a hook that is triggered by a provisioning event. When this hook is triggered, it manages the required provisioning on the vnodes to be provisioned. The hook calls a provisioning mechanism such as HPE Performance Cluster Manager to accomplish the provisioning.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

PBS allows you to configure each vnode with a list of available AOEs. This list is specified in the vnode's resources_available.aoe resource. Each vnode's current_aoe attribute shows that vnode's current AOE. The scheduler queries each vnode's aoe resource and current_aoe attribute in order to determine which vnodes to provision for each job.

When users submit jobs, they can request a specific AOE for each job. When the scheduler runs each job, it either finds the vnodes that satisfy the job's requirements, or provisions the required vnodes.

Users can create reservations that request AOEs. Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE.

7.4.1.1 Rebooting When Provisioning

When provisioning a vnode with some AOEs, the vnode must be rebooted as part of the provisioning process. Some OS installations, for example, require rebooting. In this case, the provisioning script must cause the vnode to be rebooted.

When the installation does not require a reboot, the provisioning script does not need to cause the vnode to be rebooted. For example, provisioning with some applications does not require a reboot.

7.4.2 How Vnodes Are Selected for Provisioning

Each job can request at most one AOE. When scheduling the job, PBS looks for vnodes with the requested AOE, as with any other resource. If there are not enough vnodes with the requested AOE, PBS tries to provision vnodes in order to satisfy the job's requirements.

7.4.2.1 Provisioning Policy

PBS allows a choice of provisioning policies. You set the scheduler's provision_policy configuration parameter to be either "*avoid_provision*" or "*aggressive_provision*". The default provisioning policy is "*aggressive_provision*".

avoid_provision

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, it does the following:

- PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.
- Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

7.4.2.2 Examples of Vnode Selection

The following examples show how provisioning policy can affect which vnodes are selected for provisioning.

Example 7-1: 3 vnodes

In `sched_config`:

```
node_sort_key: "ncpus HIGH"
```

We have 3 nodes as described in the following table:

Table 7-1: Example Configuration

Vnode/Host	Number of CPUs	Current AOE	State
host1	1	aoe1	*free*
host2	2	unset	*free*
host3	3	aoe2	*free*

No jobs are running on any of the vnodes.

Case 1: aggressive provisioning

```
provision_policy: "aggressive_provision"
```

Job submitted with -lselect=ncpus=1:aoe=aoe1

In this case, host3 is used to run the job and host3 is provisioned.

Case 2: avoiding provisioning

```
provision_policy: "avoid_provision"
```

Job submitted with -lselect=ncpus=1:aoe=aoe1

In this case, host1 is used to run the job and host1 is not provisioned.

Example 7-2: 5 vnodes

The following table shows the example configuration:

Table 7-2: Example Configuration

Vnode/Host	AOE Available	Current AOE	State
N1	aoe1, aoe2	aoe1	*busy*
N2	aoe1, aoe2	aoe2	*free*
N3	aoe1, aoe2	NULL	*free*
N4	aoe1, aoe2	aoe1	*free*
N5	aoe1, aoe2	aoe1	*free*

The vnodes are sorted in the order N1, N2, N3, N4, N5.

A job is submitted with:

```
qsub -lselect=3:ncpus=1:aoe=aoe1 -lplace=scatter
```

The job needs three vnodes with aoe1. Assume that all other requests except that for the AOE can be satisfied by any vnode.

Case 1: aggressive provisioning

The scheduler selects N2, N3 and N4. It has not considered the AOE instantiated on these vnodes. It then provisions N2 and N3 since N2 has a different AOE instantiated on it and N3 is not provisioned yet. N4 is not provisioned, because it has the requested AOE already instantiated.

Case 2: avoiding provisioning

First, the scheduler selects N4 and N5. It does not choose N2 since it has a different AOE instantiated, and it does not choose N3 since it does not have any AOE instantiated. But N4 and N5 together do not satisfy the job's requirement of 3 vnodes.

Second, the scheduler seeks vnodes that if provisioned can satisfy the job's request. N2 and N3 can each satisfy the job's request, so it chooses N2, because it comes first in sorted order.

The job runs on N4, N5 and N2. N2 is provisioned.

7.4.2.3 Rules for Vnode Selection for Provisioning

A vnode is not selected for provisioning for the following reasons:

- It does not have the requested AOE available in its list
- It does not have provisioning enabled on it
- It has other running or suspended jobs
- It already has the requested AOE

7.4.2.4 Triggering Provisioning

When a job requires a vnode, and the vnode's current_aoe attribute is unset, or is set to a different AOE from the one requested, the vnode is provisioned.

7.4.3 Provisioning And Reservations

7.4.3.1 Creating Reservations that Request AOEs

A reservation can request at most one AOE.

When a user creates a reservation that requests an AOE, the scheduler searches for vnodes that can satisfy the reservation. When searching, the scheduler follows the rule specified in the provision_policy scheduling parameter in `<sched_priv directory>/sched_config`. See the `pbs_sched(8B)` manual page.

The vnodes allocated to a reservation that requests an AOE are put in the resv-exclusive state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

7.4.3.2 Submitting Jobs to a Reservation

If a job that requests an AOE is submitted to a reservation, the reservation must request the same AOE.

7.4.3.3 Running a Job in a Reservation Having a Requested AOE

A job can run in a reservation that has requested an AOE, as long as the job fits the following criteria:

- It requests the same AOE as the reservation

If the job has requested no AOE, or an AOE different from that of the reservation, the job is rejected.

7.4.4 How Provisioning Affects Jobs

7.4.4.1 Preemption and Provisioning

A job that has requested an AOE will not preempt another job, regardless of whether the job's requested AOE matches an instantiated AOE. Running jobs are not preempted by jobs requesting AOEs.

7.4.4.2 Backfilling and Provisioning

If the job being backfilled around or the job doing the backfilling share a vnode, a job that has requested an AOE will not play any part in backfilling:

- It will not be backfilled around by smaller jobs.
- It will not be used as the job that backfills around another job.

7.4.4.3 Walltime and Provisioning

A job's walltime clock is started after provisioning is over.

7.4.4.4 Using `qrun`

When a job requesting an AOE is run via `qrun -H`, the following happens:

- If the requested AOE is available on the specified vnodes, those vnodes are provisioned with the requested AOE
- If the requested AOE is not available on the specified vnodes, the job is held

7.4.5 Vnode States and Provisioning

7.4.5.1 States Associated With Provisioning

The following vnode states are associated with provisioning:

provisioning

A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.

wait-provision

There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the server's max_concurrent_provision attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the *wait-provisioning* state. No jobs are run on vnodes in the *wait-provisioning* state.

resv-exclusive

The vnodes allocated to a reservation that requests an AOE are put in the resv-exclusive state when the reservation runs. These vnodes are not shared with other reservations or with jobs outside the reservation.

7.4.5.2 Provisioning Process

The following table describes how provisioning and vnode state transitions interact:

Table 7-3: Vnode State Transitions and Provisioning

Event	Starting Vnode State	Ending Vnode State
Vnode is selected for provisioning	*free*	*provisioning*
Provisioning on vnode finishes	*provisioning*	*free*
1. Job running on this vnode leaving some resources available 2. No job running on this vnode	*free*	*free*
Job running on this vnode, using all resources	*free*	*job-busy*
Vnode is selected for provisioning, but other vnodes being provisioned have already reached maximum allowed number of concurrently provisioning vnodes	*free*	*wait-provisioning*
This vnode is waiting to be provisioned for a multi-vnode job, and provisioning fails for another of the job's vnodes	*wait-provisioning*	*free*
Provisioning fails for this vnode	*provisioning*	*offline*
This vnode is waiting to be provisioned, and another vnode finishes provisioning, bringing the total number of provisioning vnodes below the limit specified in max_concurrent_provision	*wait-provisioning*	*provisioning*

7.4.5.3 Vnode State When Provisioning Fails

If provisioning fails on a vnode, that vnode is put into the *offline* state.

If provisioning for a multi-vnode job fails on one vnode, any vnodes in the *wait-provisioning* state are put into the *free* state.

7.4.5.4 Using the `qmgr` Command on Vnodes In Process of Provisioning

The following changes cannot be made to a provisioning vnode (a vnode in the *provisioning* state):

- Changing value of current_aoe vnode attribute
- Modifying resource resources_available.aoe
- Changing the state of the vnode. The `qmgr` command returns an error if this is attempted.
- Deleting the vnode from the server. The `qmgr` command returns an error if this is attempted.

The following can be modified while a vnode is provisioning:

- The server's max_concurrent_provision attribute
- A provisioning vnode's provision_enable attribute

The following cannot be set on the server host:

- current_aoe vnode attribute
- provision_enable vnode attribute
- The resources_available.aoe resource

7.4.6 Attributes, Resources, and Parameters Affecting Provisioning

7.4.6.1 Host-level Resources

aoe

The built-in aoe resource is a list of AOEs available on a vnode. Case-sensitive. You specify the list of AOEs that can be requested on a vnode by setting the value of resources_available.aoe to that list. Each job can request at most one AOE.

Automatically added to the "`resources`" line in `<sched_priv directory>/sched_config`.

Cannot be modified while a vnode is provisioning.

Non-consumable. Cannot be set on the server host. Can be set only by a Manager.

Format: string_array.

Default: unset.

Python attribute value type: str

7.4.6.2 Vnode Attributes

current_aoe

The current_aoe vnode attribute shows which AOE is currently instantiated on a vnode. Case-sensitive.

At startup, each vnode's current_aoe attribute is unset. You must set the attribute to the currently instantiated AOE if you want the scheduler to be able to choose vnodes efficiently.

The value of this attribute is set automatically after a vnode is provisioned.

This attribute cannot be modified while a vnode is provisioning.

Cannot be set on the server host. Settable by Manager only; visible to all.

Format: String.

Default: Unset.

provision_enable

This attribute controls whether the vnode can be provisioned. If set to *True*, the vnode can be provisioned.

Cannot be set on the server host.

Settable by Manager only; visible to all.

Format: Boolean

Default: Unset

7.4.6.3 Server Attributes

max_concurrent_provision

The maximum number of vnodes allowed to be in the process of being provisioned. Settable by Manager only; readable by all. When unset, default value is used. Cannot be set to zero; previous value is retained.

Format: Integer

Default: *5*

Python attribute value type: *int*

7.4.6.4 Hook Attributes

All attributes of the provisioning hook affect provisioning. See "Hook Attributes" on page 350 of the PBS Professional Reference Guide.

7.4.6.5 Scheduler Configuration Parameters

provision_policy

Specifies the provisioning policy to be used. Valid values: *avoid_provision*, *aggressive_provision*.

avoid_provision

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If it cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, it does the following:

PBS uses node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision
PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses node_sort_key to select the vnodes on which to run the job, choosing from vnodes that are free, provisionable, and offer the requested AOE, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Default: "*aggressive_provision*".

7.5 Configuring Provisioning

7.5.1 Overview of Configuring Provisioning

The administrator configures provisioning attributes, provides a provisioning tool, and writes a provisioning hook. The administrator configures each vnode to be provisioned with a list of AOE resources, where each resource is an AOE that is available to be run on that vnode. These resources are tags that tell the scheduler what can be run on that vnode. The administrator should also inform the scheduler about the current environment on each vnode, by setting the vnode's current_aoe attribute. It is also necessary to enable provisioning on each vnode to be provisioned and to set provisioning policy at the server and scheduler.

7.5.1.1 Steps in Configuring Provisioning

These are the steps that the administrator must take:

1. Provide a provisioning tool such as HPE Performance Cluster Manager (HPCM). See section 7.5.2, “Provide a Provisioning Tool”, on page 335.
2. Prepare each OS, application, or script that is to be used in provisioning. See section 7.5.3, “Prepare Images”, on page 336.
3. Configure each vnode to be provisioned with the appropriate resources. See section 7.5.4, “Define aoe Resources”, on page 336.
4. Optional: publish each vnode's current AOE. See section 7.5.5, “Inform Scheduler of Current AOE”, on page 336.
5. Write the provisioning hook's script. See section 7.5.6, “Write the Provisioning Script”, on page 337.
6. Create the empty provisioning hook, import the script, and configure the hook. See section 7.5.7, “Create and Configure the Provisioning Hook”, on page 338.
7. Configure provisioning policy. See section 7.5.8, “Configure Provisioning Policy”, on page 339.
8. Enable provisioning on vnodes. See section 7.5.9, “Enable Provisioning on Vnodes”, on page 340.
9. Enable the provisioning hook. See section 7.5.10, “Enable Provisioning Hook”, on page 340.

7.5.2 Provide a Provisioning Tool

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. This provisioning tool can either be written into the provisioning hook script, or be a separate script that is called by the provisioning hook script. You can write the provisioning tool yourself, or you can use something like the HPE Performance Cluster Manager (HPCM) cluster management tool. Your provisioning tool may be able to employ network-accessible power control units.

7.5.3 Prepare Images

You must prepare each image, application, or script you will use. Make sure that each is available to the target vnode. For example, if you use a diskless node server, put your images on the diskless node server.

The values for the ncpus and mem resources must be the same for all OS images that may be instantiated on a given vnode.

7.5.4 Define aoe Resources

The aoe resource is of type string_array, and is used to hold the names of the AOEs available at each vnode. This resource is not consumable. This resource is unset by default, and by default is added to the `resources` line in `<sched_priv directory>/sched_config`. See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide. The aoe resource is visible to all, but settable by the PBS Manager and Operator only.

The scheduler must be able to find out which AOEs can be run on which vnodes. To tag each vnode with the AOEs that can run on it, set that vnode's resources_available.aoe attribute to the list of available AOEs. For example, if vnode V1 is to run RHEL and SLES, and the hook script will recognize *rhel* and *sles*, set the vnode's resources_available.aoe attribute to show this:

```
Qmgr: set node V1 resources_available.aoe = "rhel, sles"
```

It is recommended that you make a list of all of the AOEs that may be used in provisioning in your PBS complex. The list is to facilitate script writing and resource configuration. Each entry in this list should contain at least the following information:

- Full description of the AOE
- Resource name of the AOE
- Vnodes that are to run the AOE
- Location where script should look for the AOE

For example, the list might look like the following table:

Table 7-4: Example AOE List

Description	Resource Name	Vnodes	Location
SuSE SLES 12 64-bit	*sles12*	mars, jupiter, neptune, pluto	`imageserver.example.com:/images/sles12-image`
SuSE SLES 15 64-bit	*sles15*	mars, jupiter, pluto	`imageserver.example.com:/images/sles15-image`
Red Hat Enterprise Linux 8 64-bit	*rhel7*	luna, aitne, io	`imageserver.example.com:/images/rhel8-image`
Windows Server 2016 64-bit	*winsrv16*	luna, aitne, io	`\\WinServer\ C:\images\winsrv16`

7.5.5 Inform Scheduler of Current AOE

Each vnode has an attribute called current_aoe which is used to tell the scheduler what the vnode's current AOE is. This attribute is unset by default. The attribute is of type string. It is visible to all, but settable by the PBS Manager only.

You can set this attribute on each vnode that will be used in provisioning. Set it to the value of the AOE that is currently instantiated on the vnode. So for example, using the table in section 7.5.4, "Define aoe Resources", on page 336, if vnode pluto is running 64-bit SuSE SLES 15, set current_aoe to *sles15*:

```
Qmgr: set node pluto current_aoe = sles15
```

When PBS provisions a vnode with a new AOE, the PBS server sets the value of current_aoe to the new AOE.

If PBS cannot provision a vnode with the desired AOE, it marks the vnode *offline* and unsets the value of current_aoe.

7.5.6 Write the Provisioning Script

You create the provisioning hook using a provisioning script which must manage all provisioning, either directly, or indirectly by calling other scripts. The script in the hook is the master provisioning script.

The script that does the provisioning must have the logic needed to provision the specified vnode with the specified AOE.

There are two types of provisioning. One is when the vnode is rebooted after installing/uninstalling the OS/application or running the script. The other is when the vnode is not rebooted after installing/uninstalling the OS/application or running the script.

The master provisioning script must meet the following requirements:

- Written in Python
- Arguments to the script are the vnode name and the AOE name
- If the vnode must be rebooted for provisioning, the provisioning script must cause the target vnode to be rebooted
- Must indicate success using the correct return value:
  - Return pbs.event.accept(0) if provisioning is successful and the vnode is rebooted
  - Return pbs.event.accept(1) if provisioning is successful and the vnode is not rebooted
- Must indicate failure to PBS by using pbs.event.reject()
- If the master provisioning script calls other scripts, it must wait for them to finish before returning success or failure to PBS

7.5.6.1 Arguments to Master Script

The arguments to the master script are the following:

- Name of vnode to be provisioned

  Supplied to the hook via the PBS provision event object, as pbs.event.vnode
- Name of AOE to be instantiated on the target vnode

  Supplied to the hook via the PBS provision event object, as pbs.event.aoe

These values can be passed to scripts that are called by the master script.

7.5.6.2 Return Values

The master script must indicate to PBS whether it succeeded or failed in a way that PBS can understand.

7.5.6.2.i Success

By default, pbs.event.accept() returns zero. The script must return different values for successful provisioning, depending on whether the vnode is rebooted:

- If provisioning is successful and the vnode is rebooted, the script must return *0* (zero) to PBS via pbs.event.accept(0).
- If provisioning is successful and the vnode is not rebooted, the script must return *1* (one) to PBS via pbs.event.accept(1).

7.5.6.2.ii Failure

If provisioning fails, the script must use pbs.event.reject() to indicate failure. By default, pbs.event.reject() returns *255*. To return another failure code, use the following:

```
pbs.event.reject(error message, error code)
```

where error code is any number between *2* and *255*. Returning an error code in pbs.event.reject() is optional.

7.5.6.3 Master Script Calls Subscript

Often, the master script (the hook script) calls another script, depending on the provisioning required. The subscript does the actual provisioning of the target vnode with the requested AOE. In this case, the master script must wait for the subscript to return and indicate success or failure. The master script then propagates the result to PBS.

Example of a fragment of a master script calling a subscript:

```
return_value = os.system("/var/vendor/vendor_prov.sh " <arguments to vendor_prov.sh>)
```

7.5.7 Create and Configure the Provisioning Hook

The provisioning hook causes any provisioning to happen. The provisioning hook is a Python script which either does the provisioning directly or calls other scripts or tools. Typically the provisioning hook calls other scripts, which do the actual work of provisioning. For complete information on writing hooks, see the PBS Professional Hooks Guide.

You can have at most one provisioning hook. Do not attempt to create more than one provisioning hook.

In the steps that follow, we use as examples a provisioning hook named "Provision_Hook", and an ASCII script named "`master_provision.py`".

7.5.7.1 Create the Hook

To create the provisioning hook:

```
Qmgr: create hook <hook name>
```

For example, to create a provisioning hook called Provision_Hook:

```
Qmgr: create hook Provision_Hook
```

7.5.7.2 Import the Hook Script

If the hook script is called "`master_provision.py`", and it is ASCII, and it is located in `/root/data/`, importing the hook script looks like this:

```
Qmgr: import hook Provision_Hook application/x-python default
   /root/data/master_provision.py
```

See "Importing Hooks" on page 34 in the PBS Professional Hooks Guide for more about importing hooks.

7.5.7.3 Configure the Hook Script

7.5.7.3.i Set Event Type

The event type for the provisioning hook is called "*provision*". To set the event type:

```
Qmgr: set hook Provision_Hook event = provision
```

Do not try to assign more than one event type to the provisioning hook.

7.5.7.3.ii Set Alarm Time

The default alarm time for hooks is *30 seconds*. This may be too short for a provisioning hook. You should set the alarm time to a value that is slightly more than the longest time required for provisioning. Test provisioning each AOE, and find the longest time required, then add a small amount of extra time. To set the alarm time:

```
Qmgr: set hook Provision_Hook alarm = <number of seconds required>
```

7.5.8 Configure Provisioning Policy

7.5.8.1 Set Maximum Number of Concurrently Provisioning Vnodes

The value of the server's max_concurrent_provision attribute specifies the largest number of vnodes that can be in the process of provisioning at any time. The default value of this attribute is *5*. Set the value of this attribute to the largest number of vnodes you wish to have concurrently provisioning. See section 7.4.6.3, “Server Attributes”, on page 334 for more information on the attribute.

7.5.8.1.i Considerations

You may wish to limit the number of hosts that can be in the process of provisioning at the same time:

- So that the network isn't overwhelmed transferring OS images to execution nodes
- So the hosts won't draw excessive power when powering up at the same time

Many sites have tools that notify them when an execution node goes down. You may want to avoid being paged every time an execution node is provisioned with a new AOE.

7.5.8.2 Set Scheduling Policy

When a job is scheduled to be run, and the job requests an AOE, PBS can either try to fit the job on vnodes that already have that AOE instantiated, or it can choose the vnodes regardless of AOE. Choosing regardless of AOE is the default behavior; the assumption is that the chances of finding free vnodes that match all the requirements including that of the requested AOE are not very high.

Provisioning policy is controlled by the provision_policy scheduling parameter in `<sched_priv directory>/sched_config`. This parameter is a string which can take one of two values: *avoid_provision* or *aggressive_provision*. If you want PBS to try first to use vnodes whose AOEs already match the requested AOE, set provision_policy to *avoid_provision*. If you want PBS to choose vnodes regardless of instantiated AOE, set it to *aggressive_provision*.

For details about the provision_policy parameter, see section 7.4.2.1, “Provisioning Policy”, on page 329.

For jobs that do not request an AOE, node_sort_key is used to choose vnodes.

7.5.9 Enable Provisioning on Vnodes

PBS will provision only those vnodes that have provisioning enabled. Provisioning on each vnode is controlled by its provision_enable attribute. This attribute is Boolean, with a default value of *False*. You enable provisioning on a vnode by setting its provision_enable attribute to *True*.

This attribute cannot be set to *True* on the server host.

See section 7.4.6.2, "Vnode Attributes", on page 334 for details about the provision_enable vnode attribute.

7.5.10 Enable Provisioning Hook

The last step in configuring provisioning is enabling the provisioning hook. The provisioning hook is enabled when its enabled attribute is set to *True*. To set the enabled attribute to *True* for the provisioning hook named Provision_Hook:

```
Qmgr: set hook Provision_Hook enabled = True
```

7.6 Viewing Provisioning Information

7.6.1 Viewing Provisioning Hook Contents

To see the contents of the provisioning hook, export them:

qmgr -c "export hook <hook name> application/x-python default" > <output-path>/<output-filename>

For example, if the provisioning hook is named Provision_Hook, and you wish to export the contents to /usr/user1/hook_contents:

```
qmgr -c "export hook Provision_Hook application/x-python default" > /usr/user1/hook_contents
```

7.6.2 Viewing Provisioning Hook Attributes

To view the provisioning hook's attributes, use the list hook option to the qmgr command:

```
qmgr -c "list hook <hook name>"
```

7.6.3 Printing Provisioning Hook Creation Commands

To print the provisioning hook's creation commands, use the print hook option to the qmgr command:

```
qmgr -c "p hook"
#
# Create hooks and set their properties.
#
#
# Create and define hook my_prov_hook
#
create hook my_prov_hook
set hook my_prov_hook type = site
set hook my_prov_hook enabled = True
set hook my_prov_hook event = provision
set hook my_prov_hook user = pbsadmin
set hook my_prov_hook alarm = 30
set hook my_prov_hook order = 1
import hook my_prov_hook application/x-python base64 -
c2xzbGwK
```

7.6.4 Viewing Attributes and Resources Affecting Provisioning

7.6.4.1 Server Attributes

To see the server attributes affecting provisioning, print the server's information using the qmgr command:

```
qmgr -c "print server"
```

You will see output similar to the following:

```
# qmgr
Max open servers: 49
Qmgr: p s
#
# Create queues and set their attributes.
#
#
# Create and define queue workq
#
create queue workq
set queue workq queue_type = Execution
set queue workq enabled = True
set queue workq started = True
#
# Set server attributes.
#
set server scheduling = True
set server default_queue = workq
set server log_events = 511
set server mail_from = adm
set server resv_enable = True
set server node_fail_requeue = 310
set server pbs_license_min = 0
set server pbs_license_max = 2147483647
set server pbs_license_linger_time = 31536000
set server license_count = "Avail_Global:0 Avail_Local:256 Used:0 High_Use:0"
set server max_concurrent_provision = 5
```

7.6.4.2 Viewing Vnode Attributes and Resources

To see vnode attributes and resources affecting provisioning, use the -a option to the pbsnodes command:

```
pbsnodes -a
host1
     Mom = host1
     ntype = PBS
     state = free
     pcpus = 2
     resources_available.aoe = osimage1, osimage2
     resources_available.arch = linux
     resources_available.host = host1
     resources_available.mem = 2056160kb
     resources_available.ncpus = 2
     resources_available.vnode = host1
     resources_assigned.mem = 0kb
     resources_assigned.ncpus = 0
     resources_assigned.vmem = 0kb
     resv_enable = True
     sharing = default_shared
     provision_enable = True
     current_aoe = osimage2
```

7.7 Requirements and Restrictions

7.7.1 Site Requirements

7.7.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

7.7.1.2 Provisioning Tool Required

For each vnode you wish to provision, there must be a provisioning tool that can be used on that vnode. Examples of provisioning tools are the following:

- The HPE Performance Cluster Manager (HPCM) cluster management tool
- Dual boot system
- Network-accessible power control units

7.7.1.3 Single Provisioning Hook Allowed

The PBS server allows only one provisioning hook. If you have an existing provisioning hook and you import a provisioning script, that script will become the contents of the hook, whether or not the hook already has a script. The new script will overwrite the existing provisioning hook script.

7.7.1.4 Provisioning Hook Cannot Have Multiple Event Types

The provisioning hook cannot have more than one event type.

7.7.1.5 AOE Names Consistent Across Complex

Make AOE names consistent across the complex. The same AOE should have the same name everywhere.

7.7.2 Usage Requirements

7.7.2.1 Restriction on Concurrent AOEs on Vnode

Only one AOE can be instantiated at a time on a vnode.

Only one kind of aoe resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

7.7.2.2 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

7.7.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOEs are requested by the two occurrences.

7.7.2.4 Hook Script and AOE Must Be Compatible

The requested AOE must be available to the vnode to be provisioned. The following must be *True*:

- The AOE must be in the list of available AOEs for the vnode
- Each AOE listed on a vnode must be recognized by the provisioning hook script.
- The vnode must have provisioning enabled

7.7.2.5 Provisioning Hook Must Be Ready

- The provisioning hook must obey the following rules:
  - It must exist
  - It must have a Python script imported
  - It must be enabled
  - It must be designed to invoke an external script or command for AOEs that are to be used

7.7.2.6 Server Host Cannot Be Provisioned

The server host cannot be provisioned: a MoM can run on the server host, but that MoM's vnode cannot be provisioned. The provision_enable vnode attribute, resources_available.aoe, and current_aoe cannot be set on the server host.

7.7.2.7 PBS Attributes Not Available to Provisioning Hook

The provisioning hook cannot operate on PBS attributes except for the following:

- The name of the vnode to be provisioned: pbs.event.vnode
- The AOE to be instantiated: pbs.event.aoe

7.7.2.8 avoid_provision Incompatible with smp_cluster_dist

The avoid_provision provisioning policy is incompatible with the smp_cluster_dist scheduling scheduler configuration parameter. If a job requests an AOE, the avoid_provision policy overrides the behavior of smp_cluster_dist.

7.8 Defaults and Backward Compatibility

By default, PBS does not provide provisioning. You must configure PBS to provide provisioning.

7.9 Example Scripts

7.9.1 Sample Master Provisioning Hook Script With Explanation

We show a sample provisioning hook script, and an explanation of what the script does. For readability, the sample script is a master script calling two subscripts.

This provisioning hook allows two kinds of provisioning request:

- For the application AOE named "App1", via the script `app_prov.sh`

  The `app_prov.sh` script does not reboot the vnode
- For other provisioning, via the vendor-provided provisioning shell script `vendorprov.sh`

  The `vendorprov.sh` script reboots the vnode

7.9.1.1 Sample Master Provisioning Hook Script

```
import pbs                                          (1)
import os                                           (2)

e = pbs.event()                                     (3)
vnode = e.vnode                                     (4)
aoe = e.aoe                                         (5)

if (aoe == "App1"):                                 (6)
    appret = os.system("/var/user/app_prov.sh
   " + vnode + " " + aoe )                          (7)
    if appret != 1:                                  (8)
        e.reject("Provisioning without reboot
          failed", 210)                             (9)
    else:
        e.accept(1)                                  (10)

ret = os.system("/var/vendor/vendorprov.sh
    " + vnode + " " + aoe )                         (11)

if ret != 0:                                        (12)
    e.reject("Provisioning with reboot
          failed", 211)                             (13)
else:
    e.accept(0)                                      (14)
```

7.9.1.2 Explanation of Sample Provisioning Hook Script

- Lines 1 and 2 import the pbs and os modules.
- Line 3 puts the PBS provisioning event into the local variable named "e".
- Lines 4 and 5 store the target vnode name and the name of the AOE to be instantiated on the target vnode in local variables.
- Line 6 checks whether provisioning of the application AOE named "App1" is requested.
- Line 7 is where the actual code to do non-rebooting provisioning could go. In this example, we call a subscript, passing the name of the target vnode and the requested AOE, and storing the return value in "appret".

  The non-rebooting provisioning subscript should return *1* on success.
- Line 8 checks whether non-rebooting provisioning via `app_prov.sh` succeeded.
- Line 9 returns the error code *210* and an error message to PBS if `app_prov.sh` failed.
- Line 10 returns *1* via pbs.event.accept(1) if non-rebooting provisioning succeeded.
- Line 11 calls the vendor-supplied script that is responsible for doing rebooting provisioning whenever "App1" is not the AOE.

  The name of the target vnode and the requested AOE are passed to this script.

The vendor-supplied script should expect these two arguments. The return value from this script is stored in the variable named "ret".

- Line 12 checks whether rebooting provisioning via the vendor-supplied script `vendorprov.sh` was successful.
- Line 13: If the return value is anything but *zero* (success), the provisioning hook script passes the error code *211* back to PBS, along with an error message.
- Line 14 returns success to PBS via pbs.event.accept(0) and the master script exits.

7.9.2 Sample Master Provisioning Hook Script Calling Performance Cluster Manager

The following is a master provisioning hook script that calls HPE Performance Cluster Manager (HPCM):

```
# -*- coding: utf-8 -*-
import pbs
import os

e = pbs.event()
vnode = e.vnode
aoe = e.aoe

if (aoe=="App1"):
    ret = os.system("/root/osprov/application.sh " + vnode + " " + aoe)
    if ret != 0:
        e.reject("Non-reboot provisioning failed",ret)
    else:
        e.accept(1)

ret = os.system("/root/osprov/sgi_provision.sh " + vnode + " " + aoe)
if ret != 0:
    e.reject("Reboot provisioning failed",ret)
else:
    e.accept(0)
```

7.9.3 Sample Script Set

This is a set of example Linux scripts designed to work together. They are the following:

`provision_hook.py`

This is the script for the provisioning hook. It calls the master provisioning script.

`provision_master.py:`

This is the master provisioning script. It is responsible for rebooting the machine being provisioned. It calls `update_grub.sh` to update the current AOE.

`update_grub.sh`

This shell script updates the linux `grub.conf` file and sets the value for current_aoe after the reboot.

The `update_grub.sh` script must be modified according to the `grub` configuration of the system in question before being run.

7.9.3.1 Provisioning Hook Script

provision_hook.py:

```
    import pbs
    import os

    e = pbs.event()
    vnode = e.vnode
    aoe = e.aoe
    #print "vnode:" + vnode
    #print "AOE:" + aoe

    if (aoe=="App1"):
        print "Provisioning an application"
        e.accept(1)

    ret = os.system("python /root/provision_master.py " + vnode + " " + aoe + " " + "lin")
    #print "Python top level script returned " + str(ret)
    if ret != 0:
        e.reject("Provisioning failed",ret)
    else:
        e.accept(0)
```

7.9.3.2 Master Provisioning Script

provision_master.py:

```
#!/usr/bin/python

#--------------------
# success : 0
# failure : 1
#--------------------
# win_or_lin == 1 : windows
# win_or_lin == 0 : linux
#--------------------
# 1 is TRUE
# 0 is FALSE
#--------------------

import sys
import os

vnode = sys.argv[1]
aoe = sys.argv[2]
win_or_lin = sys.argv[3]

print vnode, aoe

if not aoe.find('win'):
    print "aoe is win"
    isvnodewin = 1
else:
    print "aoe is *nix"
    isvnodewin = 0

print "win_or_lin = [", win_or_lin, "]"

if (win_or_lin == "win"):
    print "entering window server"
    if isvnodewin:
#------------ WINDOWS -> WINDOWS
        ret = os.system("pbs-sleep 05")
#------------ WINDOWS -> WINDOWS

    else:
#------------ WINDOWS -> LINUX
        ret = os.system("pbs-sleep 05")
#------------ WINDOWS -> LINUX
```

```
        ret = os.system("pbs-sleep 45")
        print "Pinging machine until it is up..."
        timeout = 120
        ticks = 0

        while 1:
            ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
            if not ret:
                print "that machine is now up"
                exit(0)

            ticks = ticks + 1
            print "ticks = ", ticks
            if ticks > timeout:
                print "exit ticks = ", ticks
                print "that machine didn't come up after 2 mins,FAIL"
                exit(1)

    else:
        print "entering linux server"
        if isvnodewin:
    #------------ LINUX -> WINDOWS
            ret = os.system("sleep 05")
    #------------ LINUX -> WINDOWS

        else:
    #------------ LINUX -> LINUX

            ret = os.system("scp -o StrictHostKeyChecking=no /root/update_grub.sh " + vnode + ":/root
        > /dev/null 2>&1")
            if ret != 0:
                print "scp failed to copy"
                exit(1)

            ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + "
        \"/root/update_grub.sh " + vnode + " " + aoe + " 1 " + " \" > /dev/null 2>&1")
            if ret != 0:
                print "failed to run script"
                exit(1)

            ret = os.system("/usr/bin/ssh -o StrictHostKeyChecking=no " + vnode + " \"reboot\"" + " >
        /dev/null 2>&1")
            if ret != 0:
                print "failed to reboot that machine"
                exit(1)

    #------------ LINUX -> LINUX
```

```
ret = os.system("sleep 45")
print "Pinging machine until it is up..."
timeout = 120
ticks = 0

while 1:
    ret = os.system("ping -c 1 -i 5 " + vnode + " -w 10 > /dev/null 2>&1")
    if not ret:
        print "that machine is now up"
        exit(0)

    print "ticks = ", ticks
    ticks = ticks + 1
    if ticks > timeout:
        print "That machine didn't come up after 2 mins. FAIL"
        exit(1)
```

7.9.3.3 Grub Update Shell Script

update_grub.sh:

```
#! /bin/sh

if [ $# -lt 2 ]; then
    echo "syntax: $0 <machine ip> <aoe name>"
    exit 1
fi

machine=$1
aoe_name=$2

menufile="/boot/grub/grub.conf"
if [ ! -f "$menufile" ]; then
    echo "grub.conf file not found. $machine using grub bootloader?"
    exit 1
fi

link=`ls -l $menufile | cut -c1`
if [ "$link" = "l" ]; then
    menufile=`ls -l $menufile | awk -F"-> " '{print $2}'`
    echo "Found link file, original file is $menufile"
fi

titles=`cat $menufile | grep title | awk -F"title" '{print $2}' | sed 's/^[ \t]//g'`
lines=`echo -e "$titles" | wc -l`

found_aoe_index=-1
count=0
while [ $count -lt $lines ]
do
    lineno=`expr $count + 1`
    title=`echo -e "$titles" | head -n $lineno | tail -n 1`
    if [ "$aoe_name" = "$title" ]; then
        found_aoe_index=$count
    fi
    count=`expr $count + 1`
done

if [ $found_aoe_index = -1 ]; then
    echo "Requested AOE $aoe_name is not found on machine $machine"
    exit 2
fi

new_def_line="default=$found_aoe_index"
```

```
def_line=`cat $menufile | grep "^default="`

echo "new_def_line=$new_def_line"
echo "def_line=$def_line"
echo "menufile=$menufile"

cp $menufile /boot/grub/grub.conf.backup
cat $menufile | sed "s/^$def_line/$new_def_line/g" > grub.out
if [ -s grub.out ]; then
    mv grub.out $menufile
else
    exit 1
fi

service pbs stop

exit 0
```

7.10 Advice and Caveats

7.10.1 Using Provisioning Wisely

It is recommended that when using provisioning, you set PBS up so as to prevent things such as the following:

- User jobs not running because vnodes used in a reservation have been provisioned, and provisioning for the reservation job will take too long
- Excessive amounts of time being taken up by provisioning from one AOE to another and back again

In order to avoid problems like the above, you can do the following to keep specific AOE requests together:

- For each AOE, associate a set of vnodes with a queue. Use a hook to move jobs into the right queues.
- Create a reservation requesting each AOE, then use a hook to move jobs requesting AOEs into the correct reservation.

7.10.1.1 Preventing Provisioning

You may need to prevent specific users or groups from using provisioning. You can use a job submission, job modification, or reservation creation hook to prevent provisioning. For more about hooks, see the PBS Professional Hooks Guide. The following is an example of a hook script to prevent USER1 from provisioning:

```
import pbs
import re

#--- deny user access to provisioning

e = pbs.event()
j = e.job   #--- Use e.resv to restrict provisioning in reservation
who = e.requestor

unallow_ulist = ["USER1"]

if who not in unallow_ulist
    e.accept(0)

#User request AOE in select?
if j.Resource_List["select"] != None:
    s = repr(j.Resource_List["select"])
    if re.search("aoe=", s) != None:
        pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to
        provision" % (who))
        e.reject("User not allowed to provision")

#User request AOE?
if j.Resource_List["aoe"] != None:
    pbs.logmsg(pbs.LOG_DEBUG, "User %s not allowed to
    provision" % (who))
    e.reject("User not allowed to provision")

e.accept(0)
```

7.10.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

7.11 Errors and Logging

7.11.1 Errors

7.11.1.1 Errors Resulting in Marking Vnodes Offline

A vnode is marked *offline* if:

- Provisioning fails for the vnode
- The AOE reported by the vnode does not match the requested AOE after the provisioning script finishes

A vnode is not marked *offline* if provisioning fails to start due to internal errors in the script.

7.11.1.2 Errors Resulting in Requeueing Job

Before provisioning a vnode with a requested OS, the server checks to see whether MoM's hook files are synced. If not, the server creates a timed task to check again. If the server fails to create the timed task, it requeues the job, and logs following error messages at log level 0x0001:

```
"Resource temporarily unavailable (11) in prov_startjob, Unable to set task for prov_startjob;
   requeueing the job"
"Cannot allocate memory (12) in prov_startjob, Unable to set task for prov_startjob; requeuing the
   job"
"Cannot allocate memory (12) in check_and_run_jobs, Unable to set task for prov_startjob;
   requeueing the job"
```

7.11.2 Logging

7.11.2.1 Accounting Logs

For each job and reservation, an accounting log entry is made whenever provisioning starts and provisioning ends. Each such log entry contains a list of the vnodes that were provisioned, the AOE that was provisioned on these vnodes, and the start and end time of provisioning.

The accounting log entry for the start of provisioning is identified by the header "*P*", and the entry for the end of provisioning is identified by the header "*p*".

Example:

Printed when job starts provisioning:

```
"01/15/2009 12:34:15;P;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
   prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=START start_time=1231928746"
```

Printed when job stops provisioning:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
   prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=SUCCESS
   end_time=1231928812"
```

Printed when provisioning for job failed:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
   prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=FAILURE
   end_time=1231928812"
```

7.11.2.2 Server Logs

7.11.2.2.i Messages Printed at Log Level 0x0080

"vnode <vnode name>: Vnode offlined since it failed provisioning"

"vnode <vnode name>: Vnode offlined since server went down during provisioning"

"Provisioning for Job <job id> succeeded, running job"

"Job failed to start provisioning"

"Provisioning for Job <job id> failed, job held"

"Provisioning for Job <job id> failed, job queued"

7.11.2.2.ii Messages Printed at Log Level 0x0100

"Provisioning of Vnode <vnode name> successful"

"Provisioning of <vnode name> with <AOE name> for <job ID> failed, provisioning exit status=<number>"

"Provisioning of <vnode name> with <aoe name> for <job id> timed out"

"Provisioning vnode <vnode> with AOE <AOE> started successfully"

"provisioning error: AOE mis-match"

"provisioning error: vnode offline"

7.11.2.2.iii Messages Printed at Log Level 0x0002

"Provisioning hook not found"

7.11.2.2.iv Messages Printed at Log Level 0x0001

"Provisioning script recompilation failed"

"Resource temporarily unavailable (11) in prov_startjob, Unable to set task for prov_startjob; requeueing the job"

"Cannot allocate memory (12) in prov_startjob, Unable to set task for prov_startjob; requeueing the job"

"Cannot allocate memory (12) in check_and_run_jobs, Unable to set task for prov_startjob; requeueing the job"

7.11.2.3 Scheduler Logs

7.11.2.3.i Messages Printed at Log Level 0x0400

Printed when vnode cannot be selected for provisioning because requested AOE is not available on vnode:

"Cannot provision, requested AOE <aoe-name> not available on vnode"

Printed when vnode cannot be selected for provisioning because vnode has running or suspended jobs, or the reservation or job would conflict with an existing reservation:

"Provision conflict with existing job/reservation"

Printed when vnode cannot be selected for provisioning because provision_enable is unset or set *False* on vnode:

"Cannot provision, provisioning disabled on vnode"

Printed when job cannot run because server is not configured for provisioning:

"Cannot provision, provisioning disabled on server"

Printed when multiple vnodes are running on the host:

"Cannot provision, host has multiple vnodes"

Printed when vnodes are sorted according to avoid_provision policy:

"Re-sorted the nodes on aoe <aoe name>, since aoe was requested"

7.11.2.3.ii Messages Printed at Log Level 0x0100

Printed when a vnode is selected for provisioning by a job:

"Vnode <vnode name> selected for provisioning with <AOE name>"

7.11.3 Error Messages

Printed when vnode is provisioning and current_aoe is set or unset or resources_available.aoe is modified via qmgr:

"Cannot modify attribute while vnode is provisioning"

Printed when qmgr is used to change state of vnode which is currently provisioning:

"Cannot change state of provisioning vnode"

Printed when vnode is deleted via 'qmgr > delete node <name>' while it is currently provisioning:

"Cannot delete vnode if vnode is provisioning"

Printed when provision_enable, current_aoe or resources_available.aoe are set on host running PBS server, scheduler, and communication daemons:

"Cannot set provisioning attribute on host running PBS server and scheduler"

Printed when current_aoe is set to an AOE name that is not listed in resources_available.aoe of the vnode:

"Current AOE does not match with resources_available.aoe"

Printed when an event of a hook is set to '*provision*' and there exists another hook that has event '*provision*':

"Another hook already has event 'provision', only one 'provision' hook allowed"

Printed when qsub has *-laoe* and *-lselect=aoe*:

"-lresource= cannot be used with "select" or "place", resource is: aoe"

Job comment printed when job fails to start provisioning:

"job held, provisioning failed to start"

Printed when job is submitted or altered so that it does not meet the requirements that all chunks must request same AOE, and this AOE must match that of any reservation to which the job is submitted:

"Invalid provisioning request in chunk(s)"

8 Security

This chapter describes the security features of PBS. These instructions are for the PBS administrator and Manager.

8.1 Configurable Features

This section gives an overview of the configurable security mechanisms provided by PBS, and gives links to information on how to configure each mechanism.

The following table lists configurable PBS security mechanisms and their configuration procedures.

Table 8-1: Security Mechanisms and their Configuration Procedures

Security Mechanism	Configuration Procedure
Authentication with daemons and users	"Authentication for Daemons & Users" on page 378
Encrypting communication	"Encrypting PBS Communication" on page 387
Access control for server, queues, reservations	"Using Access Control Lists" on page 362
Event logging for server, scheduler, MoMs	"Event Logging" on page 546
File copy mechanism	"Setting File Transfer Mechanism" on page 559
Levels of privilege (user roles)	"User Roles and Required Privilege" on page 359
Restricting access to execution hosts via $restrict_user	"Restricting Execution Host Access" on page 391

8.2 User Roles and Required Privilege

8.2.1 Root Privilege

Root privilege is required to perform some operations in PBS involving writing to the server's private, protected data. Root privilege is required in order to do the following:

- Create hooks
- Alter MoM and scheduler configuration files
- Set scheduler priority formula
- Run certain commands, including the following:
  - `pbs_probe`
  - `pbs_mom`
  - `pbs_sched`
  - `pbs_server`
  - `pbsfs`
- Use the `tracejob` command to view accounting log information

There are some operations that root privilege alone does not allow. These operations require Manager privilege but not root privilege. Manager privilege, but not root privilege, is required in order to do the following:

- Set attributes
- Create or delete vnodes using the `qmgr` command

8.2.2 User Roles

PBS allows certain privileges based on what role a person has, and whether that person has root privilege. PBS recognizes only three roles, and all those using PBS must be assigned one of these roles. These roles are *Manager*, *Operator*, and *user*. Roles are assigned by PBS Managers only. No roles can be added, and roles cannot be modified; the function of roles is hardcoded in the server.

In addition to these roles, PBS requires a PBS Administrator to perform some downloading, installation, upgrading, configuration, and management functions. PBS does not recognize *PBS Administrator* as a PBS role; this term is used in PBS documentation to mean the person who performs these tasks.

PBS roles and PBS Administrators are described in the following sections:

8.2.2.1 User

8.2.2.1.i Definition of User

Users are those who submit jobs to PBS.

Users have the lowest level of privilege. Users are referred to in the PBS documentation as "users". By default, users may operate only on their own jobs. They can do the following:

- Submit jobs
- Alter, delete, and hold their own jobs
- Status their own jobs, and those of others if permission has been given via the query_other_jobs server attribute. The query_other_jobs server attribute controls whether unprivileged users are allowed to select or query the status of jobs owned by other users.
- List and print some but not all server, queue, vnode, scheduler, and reservation attributes

8.2.2.1.ii Defining List of Users

PBS allows you to define a list of users allowed or denied access to the PBS server, however this is done using the PBS access control list mechanism. Access control is described in section 8.3, “Using Access Control Lists”, on page 362.

8.2.2.2 Operator

8.2.2.2.i Definition of Operator

A PBS Operator is a person who has an account that has been granted Operator privilege.

Operators have more privilege than users, and less privilege than Managers.

Operators can manage the non-security-related attributes of PBS such as setting and unsetting non-security attributes of vnodes, queues, and the server. Operators can also set queue ACLs.

Operators can do the following:

- All operations that users can perform
- Set non-security-related server, queue, and vnode attributes (Operators are not permitted to set server ACLs)
- Alter some job attributes
- Set or alter most resources on the server, queues, and vnodes
- Rerun, requeue, delete, and hold all jobs
- Run any command to act on a job

8.2.2.2.ii Defining List of Operators

To define the list of Operators at a PBS complex, set the server's operators attribute to a list of usernames, where each username should be an Operator. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

It is important to grant Operator privilege to appropriate persons only, since Operators can control how user jobs run.

8.2.2.3 Manager

8.2.2.3.i Definition of Manager

A Manager is a person who has an account that has been granted PBS Manager privilege.

Managers have more privilege than Operators. Managers can manage the security aspects of PBS such as server ACLs and assignment of User Roles.

Managers can do the following:

- All operations that Operators can perform
- Create or delete queues or vnodes
- Set all server, queue, and vnode attributes, including server ACLs

8.2.2.3.ii Defining List of Managers

To define the list of Managers at a PBS complex, set the server's managers attribute to a list of usernames, where each username should be a Manager. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

If the server's managers attribute is not set or is unset, root on the server host is given Manager privilege.

It is important to grant Manager privilege to appropriate persons only, since Managers control much of PBS.

8.2.2.4 PBS Administrator

8.2.2.4.i Definition of PBS Administrator

Linux: person with Manager privilege and root access.

Windows: person with Manager privilege who is a member of the local Administrators group.

A person who administers PBS, performing functions such as downloading, installing, upgrading, configuring, or managing PBS. PBS Administrators perform all the functions requiring root privilege, as described in section 8.2.1, "Root Privilege", on page 359.

PBS Administrator is distinguished from "site administrator", although often these are the same person.

8.3 Using Access Control Lists

8.3.1 Access Definitions

In this section we describe the meaning of access for each entity and object where the access of the entity to the object has an access control mechanism.

8.3.1.1 Access to a PBS Object

Below are the definitions of what access to each of the following PBS objects means:

Access to the server

Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the server and queues.

Access to a queue

Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

Access to a reservation

Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.

8.3.1.2 Access by a PBS Entity

Access can be granted at the server, queues, and reservations for each of the following entities:

User access

The specified user is allowed access.

Group access

A user in the specified group is allowed access

Host access

A user is allowed access from the specified host

8.3.2 Requirement for Access

In order to have access to a PBS object such as the server or a queue, a user must pass all enabled access control tests: the user must be allowed access, the user's group must be allowed access, and the host where the user is working must be allowed access.

In some cases, Manager or Operator privilege overrides access controls. For some kinds of access, there are no controls. See section 8.3.10, “Operations Controlled by ACLs”, on page 374.

8.3.3 Managing Access via Lists

PBS uses access control lists (ACLs) to manage access to the server, queues, and reservations. There is a separate set of ACLs for the server, each queue, and each reservation. The server enforces the access control policy for User Roles supported by PBS. The policy is hardcoded within the server. ACLs can specify which entities are allowed access and which entities are denied access.

Each server and queue ACL can be individually enabled or disabled by a Manager. If an ACL is enabled, access is allowed or denied based on the contents of the ACL. If the ACL is disabled, access is allowed to all. The contents of each server or queue ACL can be set or altered by a Manager.

Reservation ACLs are enabled only by the reservation creator or the PBS Administrator. The server's resv_enable attribute controls whether reservations can be created. When this attribute is set to *False*, reservations cannot be created.

No default ACLs are shipped.

8.3.4 ACLs

An ACL, or Access Control List, is a list of zero or more entities (users, groups, or hosts from which users or groups may be attempting to gain access) allowed or denied access to parts of PBS such as the server, queues, or reservations. A server ACL applies to access to the server, and therefore all of PBS. A queue's ACL applies only to that particular queue. A reservation's ACL applies only to that particular reservation. The server, each queue, and each reservation has its own set of ACLs.

8.3.4.1 Format of ACLs

Entity access is controlled according to the list of entities allowed or denied access as specified in the object's *acl_<entity>* attribute. The object's access control attribute contains a list of entity names, where each entity name is marked with a plus sign ("+") if the entity is allowed access, and with a minus sign ("-") if the entity is denied access. For example, to allow User1@host1.example.com, and deny User2@host1.example.com:

```
+User1@host1.example.com, -User2@host1.example.com
```

8.3.4.2 Default ACL Behavior

If an entity name is included without either a plus or a minus sign, it is treated as if it has a plus sign, and allowed access.

If an entity name is not in the list, the default behavior is to deny access to the entity. Therefore, if the list is empty but enabled because the object's acl_<entity>_enable attribute is set to *True* (see section 8.3.5, “Enabling Access Control”, on page 365), all entities are denied access.

8.3.4.3 Modifying ACL Behavior

You can specify how an ACL treats an unmatched entity by including special flags in the ACL itself. These are the plus and minus signs.

To allow access for all unmatched entities (the reverse of the default behavior), put a plus sign ("+") anywhere by itself in the list. For example:

```
+User1@host1.example.com, +, -User2@host1.example.com
```

To deny access for all unmatched entities (the default behavior), put a minus sign ("-") anywhere by itself in the list. For example:

```
+User1@host1.example.com, -, -User2@host1.example.com
```

If there are entries for both a plus and a minus sign, the last entry in the list (closest to the rightmost side of the list) will control the behavior of the ACL.

8.3.4.4 Contents of User ACLs

User ACLs contain a username and hostname combination.The subject's username and hostname combination is compared to the entries in the user ACL. Usernames take this form:

User1@host.domain.com

User1@host.subdomain.domain.com

User names can be wildcarded. See section 8.3.4.7, “Wildcards In ACLs”, on page 364.

8.3.4.5 Contents of Group ACLs

Group ACLs contain names based on the user's groups, as defined by the operating system where the server is executing. All of the user's groups are included. The subject's group names on the server are compared to the entries in the Group ACL. Group names cannot be wildcarded.

8.3.4.6 Contents of Host ACLs

Host ACLs contain fully-qualified hostnames. The subject's host name is compared to the entries in the host ACL. To find the fully-qualified name of a host, use the `pbs_hostn` command. See “pbs_hostn” on page 63 of the PBS Professional Reference Guide.

Hostnames can be wildcarded. See the following section.

8.3.4.7 Wildcards In ACLs

Usernames and hostnames can be wildcarded. The hostname portion of the username is wildcarded exactly the same way a hostname is wildcarded. The non-hostname portion of a username cannot be wildcarded.

The only character that can be used to wildcard entity names is the asterisk ("*"). Wildcarding must follow these rules:

- The asterisk must be to the right of the at sign ("@")
- There can be at most one asterisk per entity name
- The asterisk must be the leftmost label after the at sign

The following table shows how hostnames are wildcarded:

Table 8-2: How Hostnames Are Wildcarded

Wildcard Use	Meaning
`*.test.example.com`	Any host in the test subdomain in example.com
`*.example.com`	Any host in example.com
`*.com`	Any host in .com
`*`	Any host

The following examples show how wildcarding works in host ACLs:

Example 8-1: To limit host access to host myhost.test.example.com only:

```
myhost.test.example.com
```

Example 8-2: To limit host access to any host in the test.example.com subdomain:

```
*.test.example.com
```

Example 8-3: To limit host access to any host in example.com:

```
*.example.com
```

Example 8-4: To allow host access for all hosts:

```
*
```

The following examples show how wildcarding works in user ACLs:

Example 8-5: To limit user access to UserA requesting from host myhost.test.example.com only:

```
UserA@myhost.test.example.com
```

Example 8-6: To limit user access to UserA on any host in the test.example.com subdomain:

```
UserA@*.test.example.com
```

Example 8-7: To limit user access to UserA on any host in example.com:

```
UserA@*.example.com
```

Example 8-8: To limit user access to UserA from anywhere:

```
UserA@*
```

or

```
UserA
```

Listing a username without specifying the host or domain is the equivalent of listing the username followed by "@*". This means that

```
User1
```

is the same as

```
User1@*
```

8.3.4.8 Restrictions on ACL Contents

All access control lists are traversed from left to right, and the first match found is used. It is important to make sure that entries appear in the correct order.

To single out a few, specify those few first, to the left of the other entries.

Example 8-9: To allow all users in your domain except User1 access, the list should look like this:

```
-User1@example.com, +*@example.com
```

Example 8-10: To deny access to all users in your domain except User1, the list should look like this:

```
+User1@example.com, -*@example.com
```

8.3.5 Enabling Access Control

Each server and queue ACL is controlled by a Boolean switch whose default value is *False*, meaning that access control is turned off. When access control is turned off, all entities have access to the server and to each queue. When access control is turned on, access is allowed only to those entities specifically granted access.

To use access control, first set the contents of the ACL, then enable it by setting its switch to *True*.

Reservation ACLs are enabled when the reservation creator sets their contents. Reservation ACLs do not have switches. Reservations use queues, which are regular queues whose ACL values have been copied from the reservation. These queues are not intended to be operated on directly. See section 8.3.8, “Reservation Access”, on page 371.

8.3.5.1 Table of ACLs and Switches

The following table lists the ACLs and their switches, with defaults, for the server, queues, and reservations.

Table 8-3: ACLs and Their Switches

		User (Default Value)	Group (Default Value)	Host (Default Value)
Server	Switch	acl_user_enable (*False*)	None	acl_host_enable (*False*)
	List	acl_users (all users allowed)	None	acl_hosts (all hosts allowed)
Queue	Switch	acl_user_enable (*False*)	acl_group_enable (*False*)	acl_host_enable (*False*)
	List	acl_users (all users allowed)	acl_groups (all groups allowed)	acl_hosts (all hosts allowed)
Reservation	Switch	None	None	None
	List	Authorized_Users (creator only)	Authorized_Groups (creator's group only)	Authorized_Hosts (all hosts allowed)
Reservation queue	Switch	acl_user_enable (*True*)	acl_group_enable (*False*)	acl_host_enable (*False*)
	List	acl_users (creator only)	acl_groups (all groups allowed)	acl_hosts (all hosts allowed)

8.3.6 Creating and Modifying ACLs

Server and queue ACLs follow the same rules for creation and modification. Reservation queue ACLs behave the same way regular queue ACLs do. Reservation ACLs can only be created and modified by the reservation creator and the administrator. See section 8.3.8, “Reservation Access”, on page 371.

8.3.6.1 Rules for Creating and Modifying Server and Queue ACLs

- Server and queue ACLs are created and modified using the `qmgr` command.
- An ACL is a list of entries. When you operate on the list, the first match found, searching from left to right, is used. If there is more than one match for the entity you wish to control, ensure that the first match gives the behavior you want.
- When you create or add to an ACL, you can use the + or - operators to specify whether or not an entity is allowed access. Omitting the operator is equivalent to adding a + operator.
- When you re-create an existing ACL, this is equivalent to unsetting the old ACL and creating a new one.
- When you add to an ACL, the new entry is appended to the end of the ACL, on the right-hand side.
- When you remove an entity from an ACL, you cannot use + or - operators to specify which entity to remove, even if there are multiple entries for an entity and each entry has a different operator preceding it, for example `"-bob, +bob"`.
- When you remove an entity, only the first match found is removed.

8.3.6.2 Examples of Creating and Modifying Server and Queue ACLs

The following examples show the server's user ACL being set. Queue ACLs work the same way as server ACLs, and the equivalent `qmgr` command can be used for queues. So, where we use the following for the server:

```
Qmgr: set server acl_users ...
```

the same effect can be achieved at the queue using this:

```
Qmgr: set queue <queue name> acl_users ...
```

If the queue name is Q1, the qmgr command looks like this:

```
Qmgr: set queue Q1 acl_users ...
```

Example 8-11: To create a server or queue ACL:

```
Qmgr: set <object> <ACL> = <entity list>
```

Example:

```
Qmgr: set server acl_users ="-User1@*.example.com,+User2@*.example.com"
```

ACL looks like this:

```
-User1@*.example.com, +User2@*.example.com
```

Example 8-12: To add to a server or queue ACL:

```
Qmgr: set <object> <ACL> += <entity list>
```

Example:

```
Qmgr: set server acl_users += -User3@*.example.com
```

ACL looks like this:

```
-User1@*.example.com, +User2@*.example.com, -User3@example.com
```

Example 8-13: To remove an entry from an ACL:

```
Qmgr: set <object> <ACL> -= <entity>
```

Example:

```
Qmgr: set server acl_users -= User2@*.example.com
```

ACL looks like this:

```
-User1@*.example.com, -User3@*.example.com
```

Example 8-14: To remove two entries for the same entity from an ACL:

```
Qmgr: set <object> <ACL> -= <entity1, entity1>
```

Example: If ACL contains +A, +B, -C, -A, +D, +A

```
Qmgr: set server acl_users -= "A, A"
```

ACL looks like this:

```
+B, -C, +D, +A
```

Example 8-15: To remove multiple entities from an ACL:

```
Qmgr: set <object> <ACL> -= <entity list>
```

Example: If ACL contains +B, -C, +D, +A

```
Qmgr: set server acl_users -= "B, D"
```

ACL looks like this:

```
-C, +A
```

8.3.6.3 Who Can Create, Modify, Enable, or Disable ACLs

The following table summarizes who can create, modify, enable, or disable ACLs and their associated switches:

Table 8-4: Who Can Create, Modify, Enable, Disable ACLs

ACLs and Switches		Manager	Operator	User
Server ACLs and Switches	Create	Yes	No	No
	Modify	Yes	No	No
	Enable	Yes	No	No
	Disable	Yes	No	No
Queue ACLs and Switches	Create	Yes	Yes	No
	Modify	Yes	Yes	No
	Enable	Yes	Yes	No
	Disable	Yes	Yes	No
Reservation ACLs	Create	Only if reservation creator	Only if reservation creator	When creating reservation
	Modify	Yes, if administrator	No	Yes
	Enable	Creator and administrator	No	When creating reservation
	Disable	No	No	No

Table 8-4: Who Can Create, Modify, Enable, Disable ACLs

ACLs and Switches		Manager	Operator	User
Reservation Queue ACLs and Switches	Create	Yes	Yes	Indirectly when creating reservation
	Modify	Yes	Yes	No
	Enable	Yes	Yes	Indirectly when creating reservation
	Disable	Yes	Yes	Group and host ACLs can be indirectly disabled by user during reservation creation. User ACL cannot be disabled by user.

8.3.6.4 Who Can Operate on Server ACLs

PBS Managers only can create or modify server ACLs and the Boolean switches that enable them.

8.3.6.5 Who Can Operate on Queue ACLs

PBS Managers and Operators, but not users, can create and modify queue ACLs and their Boolean switches.

8.3.6.6 Who Can Operate on Reservation ACLs

When creating a reservation, the reservation creator cannot disable the user ACL, but can choose to enable or disable the group and host ACLs implicitly via the command line, and can specify the contents of all three ACLs. Reservation ACLs can be modified via pbs_ralter or disabled.

8.3.6.7 Who Can Operate on Reservation Queue ACLs

Unprivileged users cannot directly create, modify, enable, or disable reservation queue ACLs or the associated switches. The reservation creator can indirectly create and enable the reservation queue's ACLs during reservation creation. If a user wants to modify a reservation queue's ACLs, they can do so indirectly by deleting the reservation and creating a new one with the desired ACLs.

PBS Managers and Operators can modify, enable, or disable a reservation queue's ACLs.

A reservation queue's user ACL is always enabled unless explicitly disabled after creation by a Manager or Operator.

8.3.7 Server and Queue ACLs

Access control for an entity such as a user, group, or host is enabled by setting the attribute enabling that entity's ACL to *True*. When this attribute is *True*, entity access is controlled according to the list of entities allowed or denied access as specified in the ACL for that entity. The default value for each ACL's switch attribute is *False*, meaning that entity access is not controlled.

8.3.7.1 Server ACLs

The server has a host ACL and a user ACL.

Server access is controlled by these attributes:

- User access: acl_user_enable and acl_users
- Host access: acl_host_enable and acl_hosts

8.3.7.2 Queue ACLs

Each queue has three ACLs: a host ACL, a user ACL, and a group ACL.

Queue access is controlled by these attributes:

- User access: acl_user_enable and acl_users
- Group access (queue only): acl_group_enable and acl_groups
- Host access: acl_host_enable and acl_hosts

8.3.7.3 Access to Server for MoMs

You can specify whether all MoMs should have the same privilege when contacting the server as hosts listed in the acl_hosts server attribute using the acl_host_moms_enable server attribute. If you set this to *True*, all MoMs are allowed privileged access to the server, and you don't need to explicitly add their hosts to the ACL. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

8.3.7.4 Examples of Setting Server and Queue Access

To restrict access to the server or queue, first set the contents of the ACL, then enable the ACL by setting its switch to *True*.

Example 8-16: To allow server access for all users in your domain except User1, and to allow server access for User2 in another domain:

Set the server's acl_users attribute:

```
Qmgr: set server acl_users = "-User1@example.com, +*@example.com,
   +User2@otherdomain.com"
```

Enable user access control by setting the server's acl_user_enable attribute to *True*:

```
Qmgr: set server acl_user_enable = True
```

Example 8-17: To require that users of a queue be in Group1 only:

Set the queue's acl_groups attribute:

```
Qmgr: set queue Queue1 acl_groups = +Group1
```

Enable group access control by setting the queue's acl_group_enable attribute to *True*:

```
Qmgr: set queue Queue1 acl_group_enable = True
```

Example 8-18: To allow access to Queue1 from Host1 only:

Set the queue's acl_hosts attribute:

```
Qmgr: set q Queue1 acl_hosts = +Host1@example.com
```

Enable host access control by setting the queue's acl_host_enable attribute to *True*:

```
Qmgr: set q Queue1 acl_host_enable = True
```

8.3.8 Reservation Access

Advance, job-specific, and standing reservations are intended to be created by job submitters, although managers and operators can create them as well. Maintenance reservations can be created only by managers and operators. The administrator controls whether reservations can be created via the server's resv_enable attribute. When this attribute is set to *True*, reservations can be created.

Reservation ACLs allow or deny access based on group names, usernames, and hostnames. Each reservation has its own access control attributes that can be used to specify which users and groups have access to the reservation, and the hosts from which these users and groups are allowed access. The creator of the reservation sets the lists of users, groups and hosts that have access to the reservation (the reservation ACLs). This is done while creating the reservation, using options to the `pbs_rsub` command.

When you create a reservation ACL, it is automatically enabled; you do not have to explicitly enable it. The reservation's list of authorized users is always enabled during reservation creation. The reservation's lists of authorized groups and authorized hosts are only enabled if explicitly set by the reservation creator. PBS checks for membership in authorized lists only when that ACL is enabled. So for example, if you create a reservation and do not specify a list of authorized groups, no groups are added to the reservation's ACL, but you can submit jobs to the reservation because PBS does not check for group membership.

While you will see that each reservation has its own queue, do not attempt to manipulate reservation queue attributes directly. You operate on the reservation attributes, and PBS manages the queue's attributes, making them mirror those of the reservation. Set or modify reservation attributes using pbs_rsub and pbs_ralter.

8.3.8.1 Meaning of Reservation Access

Access to a reservation via the reservation's ACLs is required for the following actions:

- Submitting a job into the reservation
- Moving a job into the reservation

A job owner can perform the following actions on their own jobs, regardless of ACLs:

- Delete their job
- Hold their job
- Move their job out of the reservation

For example, if an Operator qmoves User1's job into a reservation to which User1 is denied access, User1 can still perform operations on the job such as deleting or holding the job, and User1 can `qmove` the job out of the reservation.

8.3.8.2 Reservation Access Attributes

Reservation access is controlled by the following reservation attributes:

- User access: Authorized_Users
  - Default: the reservation creator only is allowed access
  - This ACL is always enabled
- Group access: Authorized_Groups
  - Default: no groups are allowed access
  - This ACL is enabled only when you specify a list of groups
- Host access: Authorized_Hosts
  - Default: all hosts are allowed access
  - This ACL is enabled only when you specify a list of hosts

8.3.8.3 Setting and Changing Reservation Access

The reservation creator uses options to the pbs_rsub command to set reservation access attributes:

-U <authorized user list>

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's Authorized_Users attribute to *auth user list*.

This list becomes the acl_users attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the Authorized_Users reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<username>[@<hostname>][,[+|-]<username>[@<hostname>]...]

Default: Job owner only

-G <authorized group list>

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's Authorized_Groups attribute to *auth group list*.

This list becomes the acl_groups list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the Authorized_Groups reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<group name>[,[+|-]<group name> ...]

Default: No groups are authorized to submit jobs

-H <authorized host list>

Comma-separated list of hosts from which jobs can and cannot be submitted to this reservation. This list becomes the acl_hosts list for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. If the reservation creator specifies this list, the creator's host is not automatically added to the list.

See the Authorized_Hosts reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Format: *[+|-]<hostname>[,[+|-]<hostname> ...]*

Default: All hosts are authorized to submit jobs

Use the pbs_ralter command to modify existing advance, job-specific, or standing reservations:

-U <authorized user list>

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's Authorized_Users attribute to *auth user list*.

This list becomes the acl_users attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the Authorized_Users reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<username>[@<hostname>][,[+|-]<username>[@<hostname>]...]

-G <authorized group list>

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's Authorized_Groups attribute to *auth group list*.

This list becomes the acl_groups list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the Authorized_Groups reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<group name>[,[+|-]<group name> ...]

Default: no default

8.3.8.3.i Examples of Setting and Changing Reservation Access

Example 8-19: To disallow access for User1 and allow access for all other users at your domain:

Set reservation's Authorized_Users attribute using the -U option to `pbs_rsub`:

```
pbs_rsub ... -U "-User1@example.com, +*@example.com"
```

Example 8-20: To allow access for Group1 and Group2 only:

Set reservation's Authorized_Groups attribute using the -G option to `pbs_rsub`:

```
pbs_rsub ... -G "+Group1, +Group2"
```

Note that any users in Group1 and Group2 to whom you wish to grant access must be explicitly granted access in the Authorized_Users list.

Example 8-21: To allow access from Host1 and Host2 only:

Set reservation's Authorized_Hosts attribute using the -H option to `pbs_rsub`:

```
pbs_rsub ... -H "+Host1.example.com, +Host2.example.com, -*.example.com"
```

Example 8-22: To allow User2 and User3 access to the the reservation:

Use `pbs_ralter -U` to add User2 and User3 to the the reservation's Authorized_Users attribute:

```
pbs_ralter ... -U "+User2@example.com,+User3@example.com"
```

Example 8-23: To disallow Group3 access to the the reservation:

Use `pbs_ralter -G` to remove Group3 from the the reservation's Authorized_Groups attribute:

```
pbs_ralter ... -G "-Group3@example.com"
```

8.3.8.4 Reservation Queues

While you will see that each reservation has its own queue, do not attempt to manipulate reservation queue attributes directly. You operate on the reservation attributes, and PBS manages the queue's attributes, making them mirror those of the reservation. Set or modify reservation attributes using pbs_rsub and pbs_ralter.

You can move jobs into or out of reservation queues.

8.3.8.4.i Reservation Queue ACLs

If the group or host reservation ACL is specified by the reservation creator, the associated Boolean switch for the reservation queue ACL is set to *True*.

Authorized_Users is always set to the creator and copied to the queue's acl_users attribute, and acl_user_enable is always set to *True*.

If Authorized_Groups is specified by the creator, it is copied to the queue's acl_groups attribute and acl_group_enable is set to *True*. If the reservation creator does not specify a value for Authorized_Groups, nothing is copied to the queue's acl_groups, and acl_group_enable remains at its default value of *False*.

If Authorized_Hosts is specified by the creator, it is copied to the queue's acl_hosts attribute and acl_host_enable is set to *True*. If the reservation creator does not specify a value for Authorized_Hosts, nothing is copied to the queue's acl_hosts, and acl_host_enable remains at its default value of *False*.

The following table shows the relationships between reservation ACLs and reservation queue ACLs:

Table 8-5: Relationship Between Reservation ACLs and Reservation Queue ACLs

Entity	Reservation ACL	Reservation Queue ACL	Reservation Queue ACL Switch
Users	Authorized_Users	acl_users	acl_user_enable
Groups	Authorized_Groups	acl_groups	acl_group_enable
Hosts	Authorized_Hosts	acl_hosts	acl_host_enable

8.3.9 Scope of Access Control

Queue-level ACLs provide different security functionality from that provided by server-level ACLs. Access to PBS commands is controlled by server-level ACLs. For example, access to the `qstat` and `qselect` operations are controlled only at the server level. For unprivileged users, access to a specific queue is controlled through that queue's ACLs.

The users allowed access to a queue or reservation are a subset of the users allowed access to the server. Therefore, if you wish to allow a user access to a queue, that user must also be allowed access to the server. The hosts from which a user may run commands at a queue are a subset of the hosts from which a user may run commands at the server. See "Server Attributes" on page 281 of the PBS Professional Reference Guide, "Queue Attributes" on page 311 of the PBS Professional Reference Guide, and "Reservation Attributes" on page 303 of the PBS Professional Reference Guide.

8.3.10 Operations Controlled by ACLs

ACLs control some operations in PBS, but not others. Manager and Operator privileges override some ACL restrictions.

8.3.10.1 Server Operations Controlled by ACLs

8.3.10.1.i Server Host ACL

If it is enabled, the server host ACL is checked for and controls all server operations, and is honored regardless of privilege. Any request coming from a disallowed host is denied.

8.3.10.1.ii Server User ACL

If it is enabled, the server's user ACL is checked for and controls all server operations, but is overridden by Manager or Operator privilege. This means that the server's user ACL applies only to users, not to Managers or Operators. Even if explicitly denied access in the server's user ACL, a PBS Manager or Operator is allowed access to the server. Note that queue access is controlled separately by queue ACLs; even if Managers or Operators are explicitly denied access in the server's user ACL, if a queue's ACLs are not enabled, Managers and Operators have access to the queue. The same is true for reservations.

8.3.10.2 Queue Operations Controlled by ACLs

If enabled, queue ACLs are applied only when an entity is attempting to enqueue a job. Enqueueing a job can happen in any of three ways:

- Moving a job into the queue
- Submitting a job to the queue
- Routing a job into the queue

Queue ACLs are not applied for non-enqueueing operations, for example:

- Moving a job out of the queue
- Holding a job
- Deleting a job
- Signaling a job
- Getting job status

8.3.10.2.i Queue Host ACL

If a queue's host ACL is enabled, it is checked when an entity attempts to enqueue a job. The host ACL is always honored, regardless of privilege.

8.3.10.2.ii Queue User and Group ACLs

If a queue's user or group ACL is enabled, it is applied when an entity attempts to enqueue a job. Manager and Operator privileges override queue user and group ACLs when an entity attempts to move a job into a queue. This means that a PBS Manager or Operator who is explicitly denied access by the user or group ACL for queue Q1 can still use the qmove command to move a job into Q1, as long as other ACLs allow the operation (the server's user and host ACLs must both allow this).

A queue user or group ACL is applied in the following way:

Table 8-6: How Queue User and Group ACLs Are Applied

Operation	Applied to Users	Applied to Managers/Operators
Moving a job into the queue	Yes	No
Submitting a job to the queue	Yes	Yes
Having a job routed into the queue	Yes	Yes

8.3.10.3 Reservation Operations Controlled by ACLs

Access to a reservation's queue is controlled through its queue's ACLs. A reservation's queue behaves exactly the same way as a regular queue.

8.3.10.4 Table of Operations Controlled by ACLs and Overrides

The following table lists which operations are and are not controlled by server and queue ACLs, and which controls are overridden.

Table 8-7: Operations Controlled by ACLs, and ACL Overrides

Operation	Server ACLs						Queue ACLs								
	Host			User			Host			User			Group		
	Applied	Manager Override	Operator Override	Applied	Manager Override	Operator Override	Applied	Manager Override	Operator Override	Applied	Manager Override	Operator Override	Applied	Manager Override	Operator Override
Move job into queue	Y	N	N	Y	Y	Y	Y	N	N	Y	Y	Y	Y	Y	Y
Move job out of queue	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Submit job to queue	Y	N	N	Y	Y	Y	Y	N	N	Y	N	N	Y	N	N
Have job routed into queue	Y	N	N	Y	Y	Y	Y	N	N	Y	N	N	Y	N	N
Delete job	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Hold job	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Release job	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Signal job	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Status job	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Status server	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-
Status queue	Y	N	N	Y	Y	Y	N	-	-	N	-	-	N	-	-

8.3.11 Avoiding Problems

8.3.11.1 Using Group Lists

When a user specifies a group list, each and every group in which that user might execute a job must have a group name and an entry in the groups database, for example, `/etc/group`.

8.3.12 Flatuid and Access

The server's flatuid attribute affects both when users can operate on jobs and whether users without accounts on the server host can submit jobs.

8.3.12.1 How flatuid Controls When Users Can Operate On Jobs

This section describes how the server's flatuid attribute affects the circumstances under which users can operate on jobs.

This attribute specifies whether, for each user, the username at the submission host must be the same as the one at the server host. The username at the server host must always be the same as the username at the execution host. When flatuid is set to *True*, the server assumes that UserA@host1 is the same as UserA@host2. Therefore, if flatuid is *True*, UserA@host2 can operate on UserA@host1's job.

The value of flatuid also affects whether `.rhosts` and `hosts.equiv` are checked. If flatuid is *True*, `.rhosts` and `hosts.equiv` are not queried, and for any users at host2, only UserA is treated as UserA@host1. If flatuid is *False*, `.rhosts` and `hosts.equiv` are queried.

That is, when flatuid is *True*, even if UserB@host2 is in UserA@host1's `.rhosts`, UserB@host2 cannot operate on UserA's job(s). If flatuid is *False*, and UserB@host2 is in UserA@host1's `.rhosts`, UserB@host2 is allowed to operate on UserA's job(s).

Example:

UserA@host1 has a job

UserB@host2 is in UserA@host1's .`rhosts`

a. flatuid = *True*: UserB@host2 cannot operate on UserA's job

b. flatuid = *False*: UserB@host2 can operate on UserA's job

The following table shows how access is affected by both the value of the server's flatuid attribute and whether UserB@host2 is in UserA@host1's `.rhosts`:

Table 8-8: Effect of flatuid Value on Access

	flatuid = *True*		flatuid = *False*	
UserB@host2 in UserA@host1's `.rhosts`	**Yes**	**No**	**Yes**	**No**
Is UserA@host1 treated as UserA@host2?	Yes	Yes	No	No
`Is .rhosts` queried?	No	No	Yes	Yes
Can UserB operate on UserA's jobs?	No	No	Yes	No

8.3.12.2 How flatuid Affects Users Without Server Accounts

This section describes how the server's flatuid attribute affects users who have no account on the server host.

8.3.12.2.i Linux and flatuid

- If flatuid is set to *False*, users who have no account at the server host cannot submit jobs to PBS.
- If flatuid is set to *True*, these users can submit jobs. However, the job will only run if it is sent to execution hosts where the user does have an account. If the job is sent to execution hosts where the user does not have an account, the job will not run, and the MoM will log an error message.

8.3.12.2.ii Windows and flatuid

Regardless of the value of flatuid , users who have no account at the server host cannot submit jobs to PBS. Users must have an account at the server, and it must have the same password.

8.4 Authentication for Daemons & Users

PBS uses a client-server model for authentication. Note that communication between MoM and comm or PBS server and comm is initiated by MoM or PBS server, not comm. The following table shows the authentication method used for each communication pair:

Table 8-9: Authentication Method Selection

Sender	Recipient	Authentication Method Specified At...
PBS server	Comm	Server (in this case, PBS server)
MoM	Comm	Client (in this case, MoM)
Comm A	Comm B	Client (in this case, comm A)
PBS commands, e.g. `qsub`, `qstat`	PBS server	Client (in this case, the command)

By default, PBS on Linux uses reserved ports for authentication of daemons and users. On Windows, PBS uses `pwd`. You can use other methods such as MUNGE. We use MUNGE for mixed-mode operation.

Authentication is independent of encryption. For encryption, see section 8.5, “Encrypting PBS Communication”, on page 387.

For server-to-scheduler communication, PBS always uses reserved ports for authentication; this is not configurable.

8.4.1 Specifying Allowed Authentication Methods

PBS can use more than one authentication method at the same time. You specify which authentication methods are to be allowed by listing them in the PBS_SUPPORTED_AUTH_METHODS parameter in `pbs.conf` on all PBS hosts. If you leave this field blank, it defaults to "resvport" (reserved ports). If you specify any value, for example "munge", that is the only allowed method. So if you want both reserved ports and MUNGE, use "munge,resvport" (without quotes). This value is used only by the authenticating server, and is ignored by the client.

8.4.1.1 Supported Authentication Methods

You can use any of the following authentication methods/libraries:

MUNGE

resvport (reserved port)

pwd (password, used on Windows)

If you do not configure a method, PBS uses resvport.

8.4.2 Specifying Authentication Method Used by Authentication Client

To specify the default method to be used by an authentication client at a given host, set the PBS_AUTH_METHOD parameter in `pbs.conf` on that host to the desired library/method, for example, "munge". This parameter is case-insensitive.

Make sure that the authentication method you choose for the authentication client is listed in the PBS_SUPPORTED_AUTH_METHODS parameter in `pbs.conf` on the server host. This parameter is also case-insensitive.

The pbs.conf parameter is used only by the authentication client.

To override this value, set the authentication method in the PBS_AUTH_METHOD environment variable. Job submitters can set this in their profiles.

Job submitters can use multiple authentication methods. For example, a job submitter in a mixed Linux-Windows complex can submit a Linux job using MUNGE, then set their authentication method to "pwd" and submit a Windows job.

8.4.3 Authentication via Reserved Ports

PBS commands and daemons can call the pbs_iff command to authenticate a user or daemon. The pbs_iff command runs as a privileged user, binds to a reserved port, and sends a request from the client to the server.

8.4.4 Authentication via MUNGE

You can use the MUNGE authentication daemon to create and validate credentials within a PBS complex. Using MUNGE, the hosts in the PBS complex form a security realm and share a cryptographic key. PBS processes can use MUNGE to authenticate the UID and GID of other processes on hosts in a PBS complex. The client machines in the complex can create and validate credentials without using root privilege, reserved ports, or methods requiring a specific platform.

PBS Professional uses the MUNGE authentication service to authenticate the UID and GID of PBS processes, and to create and validate credentials. Once MUNGE is integrated, communication for PBS commands and daemons is validated via MUNGE. All PBS daemons are authenticated via MUNGE when they try to connect to pbs_comm.

The MUNGE key is in /etc/munge/munge.key.

8.4.4.1 Steps to Integrate MUNGE with PBS

1. Download and install a supported version of MUNGE on all machines in the PBS complex This includes server, execution, and submission hosts. You can get MUNGE either via your Linux distribution package repositories or from the MUNGE project directly; see https://dun.github.io/munge/.
2. Start MUNGE.
3. Integrate MUNGE with PBS Professional on the PBS server, client, and execution hosts, using either of the following methods:
   a. Edit the PBS configuration file (/etc/pbs.conf) and add this line:

      ```
      PBS_AUTH_METHOD=MUNGE
      ```

      The value specified in the PBS_AUTH_METHOD parameter is case-insensitive.
   b. Export the PBS_AUTH_METHOD environment variable:

      ```
      $ export PBS_AUTH_METHOD=MUNGE
      ```
4. Restart the PBS daemons.

8.4.5 Configuring SSSD

8.4.5.1 Configuring SSSD on RHEL 7 and CentOS 7

We show an example of configuring SSSD on RHEL 7 and CentOS 7, using the following steps:

1. Install the required packages:
   a. Install required packages for sssd:

      yum install realmd oddjob oddjob-mkhomedir sssd adcli openldap-clients policycoreutils-python samba-common samba-common-tools krb5-workstation

   b. Check whether libpam is already installed on the system. If not, install libpam.
   c. The pam library name may be libpam.so.<version>. If so, you may need to create a soft link:

      ln -s /usr/lib64/libpam.so.0.83.1 /usr/lib64/libpam.so

2. Find out whether we are in a domain:

   realm list

3. Discover the Active Directory domain for your Windows hosts:

   realm discover <domain controller hostname>.<domain to join>.com

   ```
   <domain to join>.com
   type: kerberos
   realm-name: <domain to join>.COM
   domain-name: <domain to join>.com
   configured: no
   server-software: active-directory
   client-software: sssd
   required-package: oddjob
   required-package: oddjob-mkhomedir
   required-package: sssd
   required-package: adcli
   required-package: samba-common-tools
   ```

4. Add the Linux host to Active Directory:

   realm join --user=Administrator@<domain to join>.com <domain controller hostname>.<domain to join>.com

5. If no errors are encountered, users should be able to see the domain information:

   realm list

   ```
   type: kerberos
   realm-name: <domain to join>.COM
   domain-name: <domain to join>.com
   configured: kerberos-member
   server-software: active-directory
   client-software: sssd
   required-package: oddjob
   required-package: oddjob-mkhomedir
   required-package: sssd
   required-package: adcli
   ```

```
required-package: samba-common-tools
login-formats: %U@<domain to join>.com
login-policy: allow-realm-logins
```

6. Verify that the Kerberos configuration file /etc/krb5.conf and sssd configuration file /etc/sssd/sssd.conf have the correct domain name specified where required.
7. Set the appropriate permissions for sssd.conf:

```
chown root:root /etc/sssd/sssd.conf
chmod 0600 /etc/sssd/sssd.conf
restorecon /etc/sssd/sssd.conf
authconfig --enablesssd --enablesssdauth --enablemkhomedir --update
systemctl start sssd
```

8. In the file /etc/sssd/sssd.conf, set use_fully_qualified_names to *False*:

```
use_fully_qualified_names = False
```

9. Restart the sssd service:

```
systemctl restart sssd
```

8.4.5.2 Configuring SSSD on RHEL8

Example 8-24: Configuring SSSD on RHEL8:

1. Follow the steps in section 8.4.5.1, “Configuring SSSD on RHEL 7 and CentOS 7”, on page 380
2. Make sure that the following lines are in the /etc/pam.d/passwd file; add them if necessary:

```
auth     include  system-auth
account  include  system-auth
```

3. Add the following line to /etc/sssd/sssd.conf:

```
access_provider = permit
```

8.4.5.3 Configuring SSSD on Ubuntu 16

Example 8-25: Configuring SSSD on Ubuntu 16:

1. Follow the steps in section 8.4.5.1, “Configuring SSSD on RHEL 7 and CentOS 7”, on page 380
2. Make sure that the following lines are in the /etc/pam.d/passwd file; add them if necessary:

```
auth     include  common-auth
account  include  common-auth
```

3. Add the following line to /etc/sssd/sssd.conf:

```
access_provider = permit
```

8.4.5.4 Configuring SSSD on Ubuntu 18

Configuring SSSD on Ubuntu 16:

1. Update and install:

```
sudo apt -y update
sudo apt-get install -y packagekit
sudo apt install sssd-ad sssd-tools realmd adcli
```

2. In /etc/hosts, add an entry for the host where AD is configured

3. Discover and join the domain:

```
sudo realm -v discover ad1.example.com
sudo realm join --user <administrator>@<domain to join>.COM <domain controller hostname>.<domain
    to join>.com
```

4. List the newly joined domain:

```
realm list
sudo pam-auth-update --enable mkhomedir
```

5. In the /etc/sssd/sssd.conf file, set the following:

```
use_fully_qualified_names = False
```

6. Restart sssd:

```
service sssd restart
```

7. Check whether sssd works:

```
id <username>
su - <username>
```

For more information, see https://ubuntu.com/server/docs/service-sssd.

8.4.5.5 Configuring SSSD on SUSE 15

Example 8-26: Configuring SSSD on SUSE 15 in order to connect to a Windows server host:

We use these settings for our example:

- Windows Domain = WINAUTHTEST.COM
- Windows Server Name = advmsetup
- Windows Server IP Address = 10.79.102.6
- AD Administrator user = loginuser
- Test User on AD = servacc

1. Install required packages and their dependencies:

```
zypper ref
zypper in krb5-client samba-client sssd sssd-ad
```

2. Edit /etc/krb5.conf so that it contains the following:

```
includedir /etc/krb5.conf.d
[libdefaults]
# "dns_canonicalize_hostname" and "rdns" are better set to false for improved security.
# If set to true, the canonicalization mechanism performed by Kerberos client may
# allow service impersonification, the consequence is similar to conducting TLS certificate
# verification without checking host name.
# If left unspecified, the two parameters will have default value true, which is less secure.
dns_canonicalize_hostname = false
rdns = false
default_realm = WINAUTHTEST.COM
dns_lookup_realm = false

[realms]
WINAUTHTEST.COM = {
    kdc = advmsetup.winauthtest.com
    master_kdc = advmsetup.winauthtest.com
    admin_server = advmsetup.winauthtest.com
}

[logging]
kdc = FILE:/var/log/krb5/krb5kdc.log
admin_server = FILE:/var/log/krb5/kadmind.log
default = SYSLOG:NOTICE:DAEMON

[domain_realm]
.winauthtest.com = WINAUTHTEST.COM
winauthtest.com = WINAUTHTEST.COM
```

3. Edit /etc/samba/smb.conf. Add the following lines to the global section:

```
[global]
workgroup = WINAUTHTEST
passdb backend = tdbsam
```

```
printing = cups
printcap name = cups
printcap cache time = 750
cups options = raw
map to guest = Bad User
logon path = \\%L\profiles\.msprofile
logon home = \\%L\%U\.9xprofile
logon drive = P:
usershare allow guests = Yes
realm = WINAUTHTEST.COM
security = ADS
template shell = /bin/bash
winbind refresh tickets = yes
winbind use default domain = yes
kerberos method = secrets and keytab
client signing = yes
client use spnego = yes
```

4. Edit /etc/hosts. Add the Windows server IP address and hostname:

```
10.79.102.6 advmsetup.winauthtest.com advmsetup winauthtest.com
10.79.102.22 sles15server2 sles15server2.winauthtest.com
```

5. Add the SLES 15 server to the AD domain:

   a. Run the `kinit` command as administrator:

```
pbsadmin@sles15server:~> kinit loginuser
Password for loginuser@WINAUTHTEST.COM:
Warning: Your password will expire in 3 days on Sunday 14 June 2020 02:16:13 PM UTC
```

   b. Join the AD domain:

```
pbsadmin@sles15server:~> sudo net ads join -U loginuser -S advmsetup.winauthtest.com
Enter loginuser's password:
Using short domain name -- WINAUTHTEST
Joined 'SLES15SERVER' to dns domain 'winauthtest.com'
No DNS domain configured for sles15server. Unable to perform DNS Update.
DNS update failed: NT_STATUS_INVALID_PARAMETER
```

   (Ignore the warning you get here for DNS.)

6. Verify that the user is authenticated to the SLES server using `ldapsearch`. This gives the AD attributes:

```
pbsadmin@sles15server:~> /usr/bin/ldapsearch -H ldap://advmsetup.winauthtest.com/ -Y GSSAPI -N -b
    "dc=winauthtest,dc=com" "(&(objectClass=user)(sAMAccountName=servacc))"
SASL/GSSAPI authentication started
SASL username: loginuser@WINAUTHTEST.COM
SASL SSF: 56
SASL data security layer installed.
# extended LDIF
#
# LDAPv3
# base <dc=winauthtest,dc=com> with scope subtree
```

```
# filter: (&(objectClass=user)(sAMAccountName=servacc))
# requesting: ALL
#

# servacc, Users, winauthtest.com
dn: CN=servacc,CN=Users,DC=winauthtest,DC=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: servacc
givenName: servacc
distinguishedName: CN=servacc,CN=Users,DC=winauthtest,DC=com
instanceType: 4
whenCreated: 20200403032934.0Z
whenChanged: 20200608034248.0Z
displayName: servacc
uSNCreated: 69750
uSNChanged: 335995
name: servacc
objectGUID:: micBdtED4UKRiEl4r2bFCg==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 132361895928844770
lastLogoff: 0
lastLogon: 132363412329813043
pwdLastSet: 132325361902386414
primaryGroupID: 513
objectSid:: AQUAAAAAAAUVAAAAcGLL19eyqDlLBPfLvhsAAA==
accountExpires: 9223372036854775807
logonCount: 6695
sAMAccountName: servacc
sAMAccountType: 805306368
userPrincipalName: servacc@winauthtest.com
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=winauthtest,DC=com
dSCorePropagationData: 20200403033300.0Z
dSCorePropagationData: 16010101000001.0Z
lastLogonTimestamp: 132360613688616817

# search reference
ref: ldap://ForestDnsZones.winauthtest.com/DC=ForestDnsZones,DC=winauthtest,DC
=com

# search reference
```

```
ref: ldap://DomainDnsZones.winauthtest.com/DC=DomainDnsZones,DC=winauthtest,DC
=com

# search reference
ref: ldap://winauthtest.com/CN=Configuration,DC=winauthtest,DC=com

# search result
search: 4
result: 0 Success

# numResponses: 5
# numEntries: 1
# numReferences: 3
```

7. Edit /etc/sssd/sssd.conf. You add the domain in the *domain* section. Add the following lines in the *sssd*, *nss*, and *domain* sections:

```
[sssd]
config_file_version = 2
services = nss, pam
# SSSD will not start if you do not configure any domains.
# Add new domain configurations as [domain/<NAME>] sections, and
# then add the list of domains (in the order you want them to be
# queried) to the "domains" attribute below and uncomment it.
; domains = LDAP
domains = winauthtest.com

[nss]

filter_users = root
filter_groups = root
[pam]

[domain/winauthtest.com]
debug_level = 6
id_provider = ad
auth_provider = ad
ad_domain = winauthtest.com
ad_server = advmsetup.winauthtest.com
ad_hostname = advmsetup.winauthtest.com
ldap_id_mapping = True
override_homedir = /home/%u
ldap_schema = ad
default_shell = /bin/bash
use_fully_qualified_names = False
```

8. Edit /etc/nsswitch.conf. NSS is used for name resolution. Since we are adding another service to resolve names, that service needs to be added to /etc/nsswitch.conf. Add 'sss' to the *passwd* and *group* fields:

```
sles15server:/home/pbsadmin # vi /etc/nsswitch.conf

passwd: compat sss
group: compat   sss
shadow: compat
```

9. Edit /etc/nscd.conf.

   Prevent ncsd from caching so that it doesn't interfere with the sssd password and group caching. If both nscd and sss are caching passwords and groups, the daemons could conflict and AD users would not be resolved. Find the enable-cache option for the password and group and set it to '*no*':

   ```
   enable-cache passwd no
   enable-cache group no
   ```

10. Restart the nscd service:

    ```
    sudo systemctl restart nscd
    ```

11. Start the sssd service:

    ```
    sudo systemctl start sssd
    ```

    You can verify whether sssd is started:

    ```
    ps -eaf | grep sssd ]
    ```

12. Enable authentication through SSSD to AD. Add the sss pam module:

    ```
    pam-config --add --sss
    pam-config --add --mkhomedir
    ```

13. Test user resolution and authentication.

    a. Run the id command to check user name resolution:

       ```
       id servacc
       ```

    b. Switch user to see whether the home directory is created:

       ```
       su - servacc
       ```

For more information on configuring sssd, see https://www.suse.com/support/kb/doc/?id=000019039.

8.5 Encrypting PBS Communication

PBS can encrypt communication sent via commands and between daemons, providing end-to-end encryption. To encrypt your PBS communication, provide the encryption mechanism, and set the PBS_ENCRYPT_METHOD parameter in pbs.conf on all PBS hosts to the method that clients will use. For end-to-end encryption, set it on all PBS hosts.

You may want to use encryption especially for cloud hosts.

TLS encryption is required on Windows.

8.5.1 Supported Encryption Methods

PBS supports TLS for encryption.

8.5.2 Using Transport Layer Security (TLS) for Client-Server Communication

You can use transport layer security (TLS) encryption for a PBS complex that has both Windows and Linux execution hosts, or when you want an extra layer of security. TLS encryption will provide greater security for your client-server connections when one PBS daemon sends a request to another daemon.

Encryption is independent of authentication. For authentication information, see section 8.4, "Authentication for Daemons & Users", on page 378.

8.5.2.1 Overview of Configuring PBS for TLS Encryption

We walk you through the steps to configure PBS for TLS encryption, and we provide example steps here. To summarize:

1. Get or create a CA certificate (the public certificate)
2. Get or create a self-signed TLS certificate
3. Copy the TLS certificate into the appropriate location
4. Generate a private key
5. Edit `pbs.conf` and set TLS as your encryption method
6. Restart PBS

For additional information, see https://www.openssl.org/docs/man1.1.1/man1/openssl-ca-html.

8.5.2.2 Example of Configuring PBS for TLS Encryption

The following steps show an example of configuring PBS for TLS encryption.

1. Log in as root or administrator.

   Perform all of the following steps as root on Linux or Administrator on Windows.
2. Create a configuration file for a certificate using X509v3 extensions.

For this step, make sure you choose options and configuration parameters that meet your requirements. See the OpenSSL documentation for help. In our example, the file is named "my.conf" and the current working directory is /root/certs. Contents of my.conf:

```
[ cacert ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:TRUE
keyUsage = critical, digitalSignature, cRLSign, keyCertSign, keyEncipherment
extendedKeyUsage = clientAuth, serverAuth, emailProtection
nsCertType = server, client, email
nsComment = "CA Certificate Generated By OpenSSL for PBSPro"

[ usrcert ]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
basicConstraints = critical, CA:FALSE
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, serverAuth, emailProtection
nsCertType = server, client, email
nsComment = "User Certificate Generated By OpenSSL for PBSPro"
```

3. Generate your root certificate authority:

```
# openssl genrsa -out rootca.key.pem 4096
# openssl req -new -key rootca.key.pem -out rootca.csr.pem -subj "/O=PBSPro/OU=PBSPro/CN=RootCA/"
# openssl x509 -req -signkey ./rootca.key.pem -extfile ./my.conf -extensions cacert -days 12775
    -in rootca.csr.pem -out rootca.cert.pem
```

4. Generate your intermediate certificate authority:

```
# openssl genrsa -out intca.key.pem 4096
# openssl req -new -key intca.key.pem -out intca.csr.pem -subj "/O=PBSPro/OU=PBSPro/CN=IntCA/"
# openssl x509 -req -CAkey ./rootca.key.pem -CA ./rootca.cert.pem -CAcreateserial -CAserial
    ./serials.txt -extfile ./my.conf -extensions cacert -days 9125 -in intca.csr.pem -out
    intca.cert.pem
```

5. Generate your CA certificate:

```
# cat rootca.cert.pem intca.cert.pem > ca.cert.pem
```

6. Generate certificates for PBS server and communication daemons:

```
# openssl genrsa -out pbspro.key.pem 2048
# openssl req -new -key pbspro.key.pem -out pbspro.csr.pem -subj "/O=PBSPro/OU=PBSPro/CN=PBSProS-
    ervices/"
# openssl x509 -req -CAkey ./intca.key.pem -CA ./intca.cert.pem -CAcreateserial -CAserial ./seri-
    als.txt -extfile ./my.conf -extensions usrcert -days 1825 -in pbspro.csr.pem -out
    pbspro.cert.pem
# openssl verify -CAfile ca.cert.pem pbspro.cert.pem
```

7. Edit PBS configuration files:

On each PBS host (server, scheduler, MoM, comm, client), edit pbs.conf, and set the PBS_ENCRYPT_METHOD parameter to "tls" (don't include the quotes). The PBS_ENCRYPT_METHOD parameter is case-insensitive.

8. Make it so we can use the value of PBS_HOME in pbs.conf:

```
# source /etc/pbs.conf
```

9. Create certificate directory:

   The PBS server and comms use a certificate key pair stored in a certificate directory.

   On each host running a PBS server or comm, create PBS_HOME/certs:

```
# mkdir ${PBS_HOME}/certs
```

10. Copy files into certificate directory:

    a. Copy your pbspro.cert.pem file to ${PBS_HOME}/certs/cert.pem:

```
# cp /root/certs/pbspro.cert.pem ${PBS_HOME}/certs/cert.pem
```

    b. Copy your pbspro.key.pem file to ${PBS_HOME}/certs/key.pem:

```
# cp /root/certs/pbspro.key.pem ${PBS_HOME}/certs/key.pem
```

11. Set permissions and ownership for certificate directory:

    a. Make sure that permissions for the certificate directory and its contents are *0600:*

```
# chmod -R 0600 ${PBS_HOME}/certs
```

    b. Make sure the owner is root on Linux or Administrator on Windows:

```
# chown -R root: ${PBS_HOME}/certs
```

12. Install CA certificate file:

    On each host running a PBS server or comm, install the file in /etc/pbs_ca.pem (Linux), or <PBS installation directory>/pbs_ca.pem (Windows). For example, if PBS is installed on Windows in C:\Program Files (x86)\PBS, you'd put the file there.

    Copy your /root/certs/ca.cert.pem file to /etc/pbs_ca.pem:

```
# cp /root/certs/ca.cert.pem /etc/pbs_ca.pem
```

13. Set permissions and ownership for CA certificate file:

    a. Make sure the permissions for the file are *0644:*

```
# chmod -R 0644 /etc/pbs_ca.pem
```

    b. Make sure the owner is root on Linux or Administrator on Windows:

```
# chown -R root: /etc/pbs_ca.pem
```

14. Restart PBS daemons:

- On every Linux host in the complex:

```
# <path to start/stop script>/pbs restart
```

  or

```
# systemctl start pbs
```

- On every Windows execution host in the complex:

```
net stop pbs_mom
net start pbs_mom
```

8.6 Restricting Execution Host Access

You can configure each PBS execution host so that the only users who have access to the machine are those who are running jobs on the machine. You can specify this by adding the $restrict_user parameter to the MoM configuration file PBS_HOME/mom_priv/config. This parameter is a Boolean, which if set to *True*, prevents any user not running a job from running any process on the machine for more than 10 seconds. The interval between when PBS applies restrictions depends upon MoM's other activities, but can be no more than 10 seconds.

You can specify which users are exempt from this restriction by adding the $restrict_user_exceptions parameter to the same file. See the description of the parameter in the next section.

You can allow system processes to run by specifying the maximum numeric user ID allowed access to the machine when not running a job. You do this by adding the $restrict_user_maxsysid parameter to the MoM configuration file. PBS automatically tries to allow system processes to run: if $restrict_user is enabled and $restrict_user_maxsysid is unset, PBS looks in /etc/login.defs for SYSTEM_UID_MAX for the value to use. If there is no maximum ID set there, it looks for SYSTEM_MIN_UID, and uses that value minus 1. Otherwise PBS uses the default value of *999*. See section 15.4.7, “Restricting User Access to Execution Hosts”, on page 556 and “$restrict_user <True | False>” on page 245 of the PBS Professional Reference Guide.

Access to pbs_mom is controlled through a list of hosts specified in the $clienthost parameter in the pbs_mom's configuration file. By default, only "localhost", the name returned by gethostname(2), and the host named by PBS_SERVER from /etc/pbs.conf are allowed. See “MoM Parameters” on page 239 of the PBS Professional Reference Guide for more information on the configuration file.

8.6.1 MoM Access Configuration Parameters

These are the configuration parameters in PBS_HOME/mom_priv/config that can be set to restrict and specify access to each execution host. Each execution host has its own configuration file.

$clienthost

List of hosts which are allowed to connect to MoM as long as they are using a privileged port. For example, this allows the hosts "fred" and "wilma" to connect to MoM:

```
$clienthost fred
$clienthost wilma
```

The following hostnames are added to $clienthost automatically: the server, the localhost, and if configured, the secondary server. The server sends each MoM a list of the hosts in the nodes file, and these are added internally to *$clienthost*. None of these hostnames need to be listed in the configuration file.

Two hostnames are always allowed to connect to pbs_mom, "localhost" and the name returned to MoM by the system call gethostname(). These hostnames do not need to be added to the MoM configuration file.

The hosts listed as "clienthosts" make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

$restrict_user <value>

Controls whether users not submitting jobs have access to this machine. When *True*, only those users running jobs are allowed access.

Format: Boolean

Default: off

$restrict_user_exceptions <user_list>

List of users who are exempt from access restrictions applied by $restrict_user. Maximum number of names in list is 10.

Format: Comma-separated list of usernames; space allowed after comma

$restrict_user_maxsysid <value>

Allows system processes to run when $restrict_user is enabled. Any user with a numeric user ID less than or equal to value is exempt from restrictions applied by $restrict_user.

Format: Integer

Default: 999

8.6.2 Examples of Restricting Access

To restrict user access to those running jobs, add:

```
$restrict_user True
```

To specify the users who are allowed access whether or not they are running jobs, add:

```
$restrict_user_exceptions <user list>
```

For example:

```
$restrict_user_exceptions User1, User2
```

To allow system processes to run, specify the maximum numeric user ID by adding:

```
$restrict_user_maxsysid <user ID>
```

For example:

```
$restrict_user_maxsysid 999
```

8.7 Access to Schedulers

Access to `pbs_sched` is not limited other than it must be from a privileged port.

8.8 Changing the PBS Service Account Password

Normally, the password for the PBS service account on Windows should not be changed. But if it is necessary to change it, perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account's password on one machine in a command prompt from an admin-type of account by typing:

   Domain environments:

   `net user <name of PBS service account> * /domain`

   Non-domain environment:

   `net user <name of PBS service account> *`

2. Provide the Service Control Manager (SCM) with the new password given above. Do this either using the GUI-based Services application which is one of the Administrative Tools, or by unregistering and re-registering the PBS services with the password. See "pbs_account" on page 53 of the PBS Professional Reference Guide.

   To unregister:

   `pbs_account --unreg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"`

   To re-register:

   `pbs_account --reg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"`

When re-registering, you can give an additional -p password argument to the pbs_account command, to specify the password on the command line.

8.9 Paths and Environment Variables

A significant effort has been made to ensure the various PBS components themselves cannot be a target of opportunity in an attack on the system. The two major parts of this effort are the security of files used by PBS and the security of the environment. Any file used by PBS, especially files that specify configuration or other programs to be run, must be secure. The files must be owned by root and in general cannot be writable by anyone other than root.

A corrupted environment is another source of attack on a system. To prevent this type of attack, each component resets its environment when it starts. If it does not already exist, the environment file is created during the install process. As built by the install process, it will contain a very basic path and, if found in root's environment, the following variables:

- TZ
- LANG
- LC_ALL
- LC_COLLATE
- LC_CTYPE
- LC_MONETARY
- LC_NUMERIC
- LC_TIME

The environment file may be edited to include the other variables required on your system.

The entries in the PBS_ENVIRONMENT file can take two possible forms:

```
variable_name=value
variable_name
```

In the latter case, the value for the variable is obtained before the environment is reset.

8.9.1 Path Caveats

Note that PATH must be included. This value of PATH will be passed on to batch jobs. To maintain security, it is important that PATH be restricted to known, safe directories. Do NOT include "." in PATH. Another variable which can be dangerous and should not be set is IFS.

8.10 File and Directory Permissions

Each parent directory above PBS_HOME must be owned by root and writable by root only. All files and directories used by PBS should be writable by root only. Permissions should allow read access for all files and directories except those that are private to the daemons. The following should not be writable by any but root:

```
PBS_HOME/mom_priv
<sched_priv directory>
PBS_HOME/server_priv
```

The PBS_HOME directory must be readable and writable from server hosts by root (Administrator) on Linux.

On Windows, PBS_HOME must have Full Control permissions for the local "Administrators" group on the local host.

PBS checks permissions for certain files and directories. The following error message is printed for certain files and directories (e.g. /etc/pbs.conf, /var/spool/PBS/mom_priv/config, etc.) if their permissions present a security risk:

```
<command>: Not owner (1) in chk_file_sec, Security violation "<directory>" resolves to
    "<directory>"
<command>: Unable to configure temporary directory.
```

8.11 Root-owned Jobs

The server will reject any job which would execute under the UID of zero unless the owner of the job, typically root, is listed in the server attribute acl_roots.

In order to submit a job from a root account on the local host, be sure to set acl_roots. For instance, if user foo has root privilege, you need to set:

```
Qmgr: set server acl_roots += foo
```

in order to submit jobs and not get a "bad UID for job execution" message.

Windows Administrators are not considered to have root access, so a Windows Administrator can run a job without being listed in acl_roots.

8.11.1 Caveats for Root-owned Jobs

Allowing root jobs means that they can run on a configured host under the same account which could also be a privileged account on that host.

8.12 Passwords

PBS has different password requirements dictated by the Linux and Windows operating systems. Jobs submitted on Linux systems do not require passwords. Jobs on Windows MoM systems require passwords.

See the PBS Professional 2021.1 release notes for a list of supported architectures.

8.12.1 Windows User Passwords

Windows execution host systems require a password for PBS to run a process as the user, so users on these systems must supply a password. Users cache their passwords via the pbs_login command. Job submitters run the pbs_login command once per submission host, initially and for each password change.

8.12.2 Changing the PBS Service Account Password

Normally, the PBS service account password should not be changed. But if it is necessary to change it perhaps due to a security breach, then do so using the following steps:

1. Change the PBS service account's password on a machine in a command prompt from an admin-type of account by typing:

   net user <name of PBS service account> * /domain

2. Provide the Service Control Manager (SCM) with the new password specified above. This can be done via the GUI-based Services application found as one of the Administrative Tools, or by unregistering and re-registering the PBS MoM with the new password.

   pbs_account --unreg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"

   pbs_account --reg "\Program Files (x86)\PBS\exec\sbin\pbs_mom.exe"

   The register form (last line above) can take an additional argument `-p password` so that you can specify the password on the command line directly.

3. Run the `pbs_login` command:

   pbs_login -m <PBS service account password>

4. Restart MoM:

   net stop pbs_mom

   net start pbs_mom

8.12.2.1 Caveats for Changing Service Account Password

Using `pbs_account --unreg` and `pbs_account--reg` stops and restarts MoM, which can kill jobs.

8.13 Windows Firewall

Under Windows, the Windows Firewall may have been turned on by default. If so, it will block incoming network connections to all services including PBS. Therefore after installing PBS Professional, to allow `pbs_mom` to accept incoming connections:

Access *Settings->Control Panel->Security Center->Windows Firewall*, and verify that the Windows Firewall has been set to "*ON*" to block incoming network connections.

From this panel, you can either turn Windows Firewall "*off*", or click on the *Exceptions* tab and add the following to the list:

`[INSTALL PATH]\exec\sbin\pbs_mom.exe`

8.14 Logging Security Events

Each PBS daemon logs security-related events, at event class 32 (0x0020) or at event class 128 (0x0080). For information about daemon logfiles, see section 15.3, “Event Logging”, on page 546.

8.14.1 Events Logged at Event Class 32 (0x0020)

The following security-related events are logged at decimal event class 32 (0x0020):

- When an execution host has access restrictions in place via the $restrict_user configuration parameter, and MoM detects that a user who is not exempt from access restriction is running a process on the execution host, MoM kills that user's processes and writes a log message:

  ```
  01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
  killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
  ```

  See section 8.6, "Restricting Execution Host Access", on page 391.

- If for some reason the access permissions on the PBS file tree are changed from their default settings, a daemon may detect this as a security violation, refuse to execute, and write an error message in the corresponding log file. The following are examples of each daemon's log entry:

  ```
  Server@<host>: Permission denied (13) in chk_file_sec, Security violation
      "/var/spool/pbs/server_priv/jobs/" resolves to "/var/spool/pbs"
  pbs_mom: Permission denied (13) in chk_file_sec, Security violation
      "/var/spool/pbs/mom_priv/jobs/" resolves to "/var/spool/pbs"
  pbs_sched: Permission denied (13) in chk_file_sec, Security violation "/var/spool/pbs/sched_priv"
      resolves to "/var/spool/pbs"
  ```

A Manager can run `pbs_probe` (on Linux) or `pbs_mkdirs` (on Windows) to check and optionally correct any directory permission or ownership problems.

- When a user without a password entry (an account) on the server attempts to submit a job, the server logs this event. The following is an example log entry:

  ```
  8/21/2009 15:28:30;0080;Server@capella;Req;req_reject;Reject reply code=15023, aux=0, type=1,
      from User1@host1.example.com
  ```

- If a daemon detects that a file or directory in the PBS hierarchy is a symbolic link pointing to a non-secure location, this is written to the daemon's log. The resulting log message is the same as for a permission violation:

  ```
  Server@<host>: Permission denied (13) in chk_file_sec, Security violation
      "/var/spool/pbs/server_priv/jobs/" resolves to "/var/spool/pbs"
  pbs_mom: Permission denied (13) in chk_file_sec, Security violation
      "/var/spool/pbs/mom_priv/jobs/" resolves to "/var/spool/pbs"
  pbs_sched: Permission denied (13) in chk_file_sec, Security violation "/var/spool/pbs/sched_priv"
      resolves to "/var/spool/pbs"
  ```

- If an $action script is to be executed for a job belonging to a user who does not have an account on an execution host, the execution host's MoM logs this event. The following is an example log entry:

  ```
  08/21/2009 16:06:49;0028;pbs_mom;Job;2.host1;No Password Entry for User User1
  ```

- When a job triggers an action script for which the environment cannot be set up, perhaps due to a system error, the MoM attempting to run the action script logs the event. The log message contains the following:

  ```
  :<job ID>:failed to setup dependent environment!
  ```

- When the scheduler attempts to run a job on an execution host where the job's owner does not have an account, the MoM on the execution host logs this event. The following is an example log entry:

  ```
  08/21/2009 15:51:14;0028;pbs_mom;Job;1.host1;No Password Entry for User User1
  ```

- When the scheduler attempts to run a job on an execution host where the job's owner does not have a home directory, and when the job's sandbox attribute is not set to *PRIVATE*, the execution host's MoM logs this event. The log message contains the following:

  ```
  Access from host not allowed, or unknown host: <numeric IP address>
  ```

See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).

8.14.1.1 Events Logged at Event Class 128 (0x0080)

The following security-related event is logged at event class 128 (0x0080):

- If an attempt is made to connect to a host in the PBS complex from an unknown host, the PBS daemon logs the information at both levels 32 and 128 (0x0020 and 0080).
- If a user or Operator tries to set an attribute that can be set by Managers only, or attempts to create or delete vnodes:

  The `qmgr` command returns this error message:

  ```
  qmgr obj=<object> svr=default: Unauthorized Request
  qmgr: Error (15007) returned from server
  ```

  The server logs the following message:

  ```
  Req;req_reject;Reject reply code=15007, aux=0, type=9, from <username>
  ```

- When a user is denied access to the server because of the contents of the acl_users server attribute, the server logs the following:

  ```
  Req;req_reject;Reject reply code=15007, aux=0, type=21, from username@host.domain.com
  ```

8.14.1.2 Events Logged at Event Class 1

- When an attempt is made to contact MoM from a non-privileged port for a request requiring a privileged port, MoM logs the following:

  ```
  pbs_mom;Svr;pbs_mom;Unknown error: 0 (0) in rm_request, bad attempt to connect message refused
      from port 61558 addr 127.0.0.1
  ```

8.14.1.3 Events Not Logged

The following events are not logged:

- When an attempt is made to connect to a host in the PBS complex from a disallowed host
- When an ACL check denies an entity access to a PBS object
- A user tries to query other users' jobs when the server's query_other_jobs attribute is set to *False*
- When an Operator or Manager overrides the server's user ACL

8.15 Securing Containers

- Use the security enhancement named "pbs_container"; see section 18.4.4, "Configure Security Enhancement for Docker", on page 625.
- Make sure that when you are configuring the container hook, if you whitelist any container arguments in the container_args_allowed hook configuration parameter, do not whitelist "--group-add". This would allow job submitters to add themselves to any groups inside the container. Instead, set the enable_group_add_arg hook parameter to *True* so the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs. See section 18.4.2, "Configure PBS Container Hook", on page 621.

9 Making Your Site More Robust

This chapter describes how to configure PBS to make your site more robust.

9.1 Robustness

PBS provides the following mechanisms that support site robustness and flexibility:

Failover

The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.

Checkpoint and Restart

Allows jobs to be checkpointed and restarted. Uses OS-provided or third-party checkpoint/restart facility.

Reservation Fault Tolerance

PBS attempts to ensure that reservations run by finding usable vnodes when reservation vnodes become unavailable.

Vnode Fault Tolerance for Job Start and Run

PBS lets you allocate extra vnodes at job startup or for the life of the job, to compensate for vnode failure and allow the job to successfully start or run on the required number of vnodes.

Preventing Communication and Timing Problems

PBS allows setting parameters to prevent problems in communication, timing, and load on vnodes.

Preventing File System Problems

PBS gives you tools to prevent file system problems.

OOM Killer Protection

PBS is installed so that daemons are protected from an OOM killer.

9.2 Failover

9.2.1 Glossary

Primary Server

The PBS Professional server daemon which is running during normal operation.

Secondary Server

The PBS Professional server daemon which takes over when the primary server fails.

Primary Scheduler

The PBS Professional scheduler daemon which is running during normal operation.

Secondary Scheduler

The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

Active

A server daemon is active when it is managing user requests and communicating with the scheduler and MoMs.

Idle

A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

9.2.2 How Failover Works

During normal operation, the primary server is active and the secondary server is idle. If the primary server fails for any reason, the secondary server becomes active and takes over server functions for the complex. No work is lost during the transition between servers. PBS functions the same during failover as it does during normal operation. The PBS data service is considered to be part of the PBS server; if it fails, this triggers failover.

9.2.2.1 Primary and Secondary Schedulers

Each server is paired with and uses its own scheduler. If the secondary server becomes active, it starts its own scheduler.

9.2.2.2 Primary and Secondary Data Services

Each server is paired with and uses its own data service. If the secondary server becomes active, it starts its own data service.

9.2.2.3 Normal Post-configuration Behavior

After you have configured PBS for failover, and started both servers, the secondary server periodically attempts to connect to the primary server until it succeeds and registers itself with the primary server. The secondary server must be registered in order to take over upon failure of the primary server.

9.2.2.4 Behavior During Failover

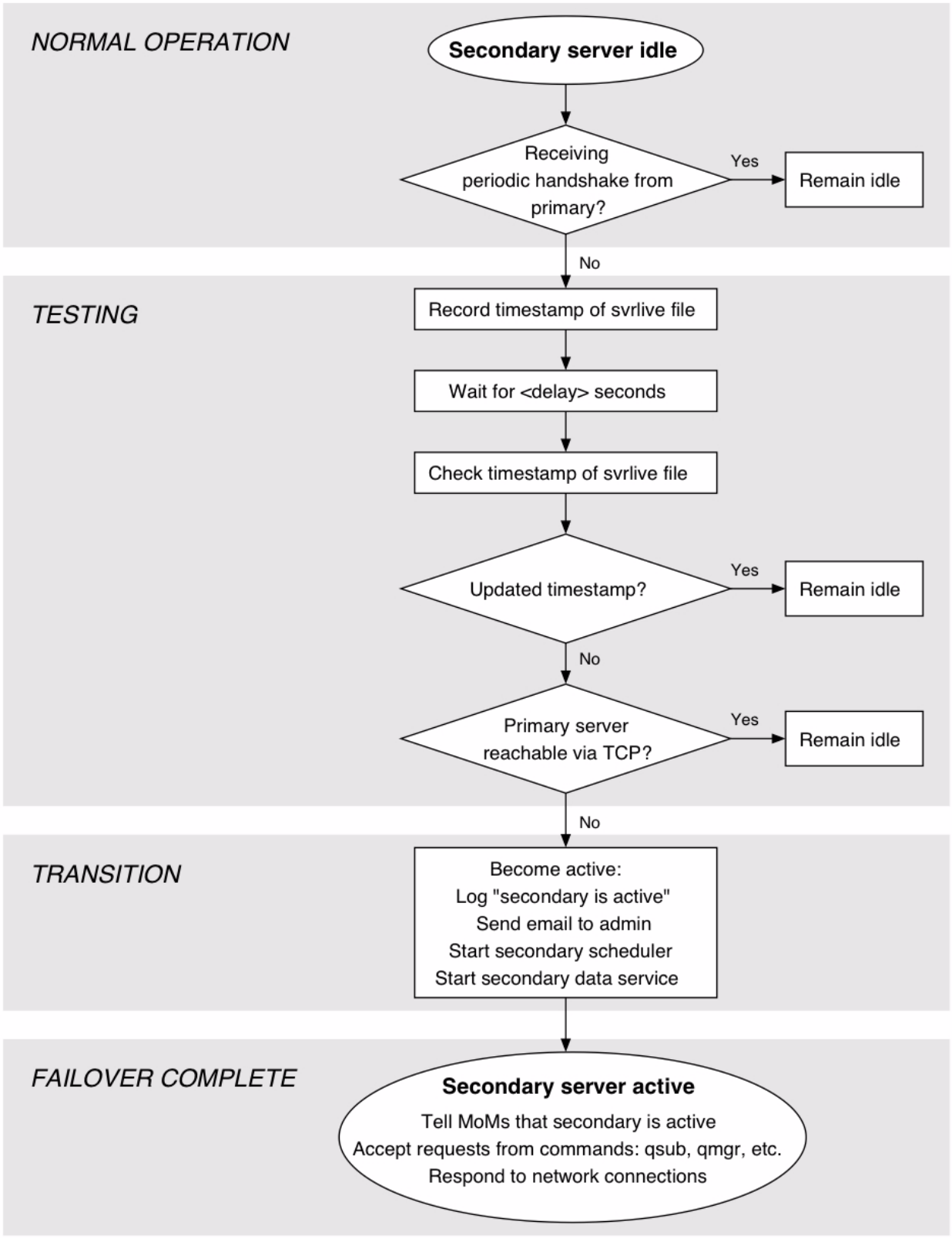


Figure 9-1:Behavior During Failover

When both server daemons are running, the primary server sends periodic handshake messages to the secondary. The primary server also periodically updates the timestamp of the PBS_HOME/server_priv/svrlive file. If the secondary server stops receiving handshake messages from the primary server, the following happens:

- The secondary server waits for a specified delay period before taking over. This delay is specified using the pbs_server -F option. The default period is 30 seconds.
  - The secondary server reads the timestamp of the PBS_HOME/server_priv/svrlive file and stores it in memory
  - The secondary waits for the specified delay, then checks the time stamp again, and compares it to the timestamp it stored in memory
  - If the timestamp has changed, the secondary server remains idle
  - If the timestamp has not changed, the secondary attempts to open a new TCP connection to the primary
  - If the secondary server cannot open a TCP connection to the primary, the secondary becomes active
- The secondary server logs a message saying that failover has occurred.
- An email is sent to and from the account defined in the server's mail_from attribute, saying that failover has occurred.
- The secondary server starts the secondary scheduler on the secondary server host.
- The secondary server starts the secondary data service on the secondary server host.
- The secondary server notifies all of the MoMs that it is the active server.
- The secondary server begins responding to network connections and accepting requests from client commands such as qstat and qsub.

9.2.2.5 Delay During Failover Transition

The default delay between when the primary becomes unavailable and the secondary takes over is about 5 minutes. You can change this using pbs_server -F <seconds>. If you use pbs_server -F -1, the secondary makes only one attempt to contact the primary, then takes over. We include these instructions in the configuration steps.

9.2.2.6 Behavior When Primary Resumes Control

When the primary server starts back up, it takes control from the secondary server, becoming the active server. The secondary server becomes idle and resumes listening for the regular handshake messages from the primary server.

The primary server may have been stopped for any of several reasons. The restart method will vary accordingly. If the host was stopped, the PBS server is restarted automatically when the host is started. If the host is still up but the server was stopped, restart the server. See "Starting Servers With Failover" on page 163 in the PBS Professional Installation & Upgrade Guide.

The primary server uses only its own scheduler and data service. When the primary server resumes control, it starts a scheduler and data service, and stops the secondary scheduler and data service. No data is lost in the transition.

When the primary has taken control, the secondary logs a message saying so:

```
received takeover message from primary, going inactive
```

9.2.2.7 Server Name and Job IDs During Failover

The server name and job IDs do not change when the secondary server is active. For example, the primary server is on a host named *PrimaryHost.example.com*, and the secondary server is on a host named *SecondaryHost.example.com*. When the primary server is active, the server name is *PrimaryHost*, jobs are given job IDs of the form *NNNN.PrimaryHost*, and the value of the server_host server attribute is *PrimaryHost.example.com*. When the secondary server is active, the server name is still *PrimaryHost*, jobs are still given job IDs of the form *NNNN.PrimaryHost*, but the value of server_host is *SecondaryHost.example.com*.

The table below summarizes the server name, value of server_host and the IDs given to jobs, when either the primary or secondary server is active.

Table 9-1: Server Name, Job ID and Value of server_host Depending on Which Server is Active

	Active Server	
	Primary	**Secondary**
Hostname	*PrimaryHost.example.com*	*SecondaryHost.example.com*
Server Name	*PrimaryHost*	*PrimaryHost*
Value of server_host	*PrimaryHost.example.com*	*SecondaryHost.example.com*
Job Name	*NNNN.PrimaryHost*	*NNNN.PrimaryHost*

9.2.2.8 Information Used by Primary and Secondary Servers

The primary and secondary servers share a single source for attribute information, so anything set via the `qmgr` command need only be set once. `PBS_HOME` is in a shared location. License information is shared and needs to be set at only one server.

Each server, execution and client host uses its own `pbs.conf` file, so these must be set for each host in the complex.

9.2.2.9 Impact on Users

Users may not notice when a failover occurs. When a user uses a PBS command such as `qstat`, the command tries to connect to the primary server first. If that fails, the command tries the secondary server. There may be up to a two-minute delay in server commands while failover is taking place.

If the secondary server responds to the command, the command creates a local file so that this process is not repeated for every PBS command.

The file is named:

`/tmp/.pbsrc.UID`

where *UID* is the user ID.

When this file exists, commands try the secondary server first, eliminating the delay in attempting to connect to the down server. If a command cannot connect to the secondary server, and can connect to the primary server, the command removes the file.

The file is removed when the primary server takes over.

9.2.2.10 Determining Which Server Is Active

The server attribute server_host contains the name of the host on which the active server is running. Use the `qstat -Bf` command to see the value of server_host.

9.2.2.11 Delay Between Primary Failure and Secondary Becoming Active

The default delay time from detection of possible primary server failure until the secondary server takes over is 30 seconds. A secondary server on a very reliable network can use a shorter delay. A secondary server on an unreliable network may need to use a longer delay. The delay is specified via the -F option to the `pbs_server` command.

9.2.2.12 Communication

If PBS is configured for failover, each server host runs a `pbs_comm`. Note that communication traffic is handled independently of failover behavior. During normal operation, the comm on the primary server host handles communication traffic, but if that comm becomes unavailable, the comm on the secondary automatically takes over the communication traffic. You do not need to perform any configuration to get this behavior; the communication daemons are automatically configured for you. See "Failover and Communication Daemons" on page 52 in the PBS Professional Installation & Upgrade Guide.

9.2.2.12.i Communication with MoMs

- If a MoM will see different server addresses, add a $clienthost entry to MoM's configuration file for each possible server address.
- The secondary server is automatically added to the list of hosts allowed to connect to MoMs, in the $clienthost MoM configuration parameter.

9.2.3 Windows Locations

PBS is installed on Windows systems in `\Program Files (x86)\PBS\`.

9.2.4 Prerequisites for Failover

9.2.4.1 Checklist of Prerequisites for Failover

The following table contains a checklist of the prerequisites for failover. Each entry has a link to more detailed information about the entry.

Table 9-2: Prerequisites for Failover

Prerequisite	Explanation
Identical server hosts	See section 9.2.4.2, "Server Host Requirements", on page 405
MoMs on server hosts don't share a `mom_priv` directory	See section 9.2.4.3, "Requirements for MoMs on Server Hosts", on page 405
All hosts must be able to communicate over the network	See section 9.2.4.4, "Ensuring Communication Between Hosts", on page 406
All hosts must be able resolve hostnames of other hosts in complex	See section 9.2.4.5, "Hostname Resolution", on page 406
Filesystem must be shared, on a separate host from either server host, and provide features required for failover; no root squash on shared filesystem	See section 9.2.4.6, "Shared Filesystem", on page 406
Administrator must have access to filesystem from both server hosts	See section 9.2.4.7, "Permission Requirements", on page 407
Same version of PBS for all components	See section 9.2.4.8, "Same PBS Versions Everywhere", on page 407
Primary server's scheduler must be able to run when primary server runs	See section 9.2.4.9, "Requirement for Scheduler", on page 407

Table 9-2: Prerequisites for Failover

Prerequisite	Explanation
Data service user account must be the same on both primary and secondary server hosts	See section 9.2.4.10, "Same Data Service Account on Both Server Hosts", on page 407
Data service host must be default	See section 9.2.4.11, "Data Service Host Configuration Requirement", on page 407
User names must be consistent across primary & secondary servers hosts	See section 9.2.4.12, "Consistent User Names", on page 407
The mail_from server attribute specifies an email address that is monitored. Not required, but recommended.	See section 9.2.4.13, "Monitor Server Mail", on page 408

9.2.4.2 Server Host Requirements

The primary and secondary servers must run on two separate host machines. Both host machines must have the same architecture. They must be binary compatible, including word length, byte order, and padding within structures. There must be exactly one primary and one secondary server.

On an HPE 8600, use two different service nodes to run the primary and secondary servers.

9.2.4.3 Requirements for MoMs on Server Hosts

You can run a MoM on both the primary and secondary server hosts, but this is **not** recommended.

If a MoM is to run on both server hosts, the two MoMs must not share the same PBS_HOME/mom_priv directory. In addition, it is strongly recommended that the following be true:

- The mom_priv directory structure be replicated on a local, non-shared, filesystem. On Windows, MoM already has a local directory on each server host. On Linux, you must create these.

  Replicate the mom_priv and mom_logs directory structures on the primary server host if they don't exist there already. You must put these in the same location. Do the following on the primary server host:

  ```
  scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
  scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
  ```

  Replicate the mom_priv and mom_logs directory structures on the secondary server host if they don't exist there already. You must put these in the same location. Do the following on the secondary server host:

  ```
  scp -r <existing PBS_HOME/mom_priv> <local PBS_HOME/mom_priv>
  scp -r <existing PBS_HOME/mom_logs> <local PBS_HOME/mom_logs>
  ```

- Each MoM use its own, local, mom_priv directory structure

  The PBS_MOM_HOME entry in pbs.conf specifies the location that contains the mom_priv and mom_logs directories. If PBS_MOM_HOME is specified in pbs.conf, pbs_mom uses that location instead of PBS_HOME.

  To prevent the MoMs from automatically using the same directory, do one of the following:

  - Recommended: Specify the separate, local PBS_MOM_HOME entry in each server host's pbs.conf file (each pbs_mom will use the location for mom_priv specified in its PBS_MOM_HOME). Give the location of the local PBS_HOME/mom_priv that you replicated on each host. You can perform this step now, or later, when editing pbs.conf on each server host, in section 9.2.5.3, "Host Configuration for Failover on Linux", on page 411, or section 9.2.5.4, "Host Configuration for Failover on Windows", on page 415.
  - Use the -d option when starting at least one pbs_mom to specify that they use the local, non-default locations for mom_priv

9.2.4.4 Ensuring Communication Between Hosts

Both the primary and secondary server hosts must be able to communicate over the network with each other and all execution hosts.

Beware of dependencies on remote file systems: The $PBS_CONF_FILE environment variable must point to `pbs.conf`. PBS depends on the paths in `pbs.conf` being available when its start/stop script is executed. PBS will hang if a remote file access hangs, and normal privileges don't necessarily carry over for access to remote file systems. For example, a FAT filesystem mounted via NFS won't support permissions.

9.2.4.5 Hostname Resolution

Hostname resolution must work between each host in the PBS complex. **Make sure that all hosts in the complex** (the primary and secondary server hosts, the file server host, and all execution and client hosts) **are set up so that they can resolve the names of all other hosts in the complex**. If you are not sure whether hostname resolution is working, run the `pbs_hostn` command at each host, testing the hostnames of the other hosts. The `pbs_hostn` command will return the canonical hostname of the specified host.

9.2.4.6 Shared Filesystem

The filesystem you use for the machines managed by PBS should be highly reliable. We recommend, in this order, the following filesystems:

- HA DAS
- DAS, such as `xfs` or `gfs`
- HA NFS
- NFS

`PBS_HOME` is the top directory used by the PBS server. The primary and secondary servers share the same `PBS_HOME` directory. The `PBS_HOME` directory must conform to the following:

- The `PBS_HOME` directory must be available under the same name to both the primary and secondary server hosts.
- The `PBS_HOME` directory must be on a file system which meets the following requirements:
  - It should reside on a different machine from either of the server hosts.
  - It must be shared by the primary and secondary server hosts.
  - It must be reliable. The file system must be always available to both the primary and secondary servers. A failure of the file system will stop PBS from working.
  - The file system protocol must provide file locking support.
  - The file locking daemons must be running.
  - For Linux, the filesystem must support POSIX (Open Group) file semantics.
  - It must support concurrent read and write access from two hosts.
  - It must support multiple export/mounting.
  - No root squash on the shared filesystem.

If your filesystem does not conform to the specifications above, follow the steps in the next sections.

9.2.4.6.i Using NFS Filesystems

When using NFS for `PBS_EXEC`, NFS must be configured to allow root access and to allow `setuid-root` programs to execute from it.

If possible, mount NFS file systems synchronously (without caching) to avoid reliability problems.

NFS filesystems should be hard mounted.

9.2.4.6.ii Setting Up the Shared Filesystem

You can use NFS or another filesystem protocol to set up the shared filesystem on which PBS_HOME resides. Examples are Lustre, IBM GPFS, and Red Hat GFS. Make sure your protocol supports:

- Multiple export/mounting
- Simultaneous read/write from two hosts
- File locking support

To set up your file system:

1. Choose a machine for the file server host. This machine must not be either of the server hosts.
2. Make sure the file system is mounted by both the primary and secondary server hosts. For NFS, make sure the file system is hard mounted by both hosts.
3. Make sure the file system can provide file locking. For NFS, the lock daemon, lockd, must be running.
4. Make sure that PBS_HOME is available under the same name to both the primary and secondary server hosts.

9.2.4.7 Permission Requirements

The PBS_HOME directory must meet the security requirements of PBS. Each parent directory above PBS_HOME must be owned by root and writable by root only.

The PBS_HOME directory must be readable and writable from both server hosts by the PBS Administrator.

9.2.4.8 Same PBS Versions Everywhere

Both server hosts, all the execution hosts, and all the client hosts must run the same version of PBS Professional.

9.2.4.9 Requirement for Scheduler

The primary scheduler must be able to run whenever the primary server is running, and the secondary scheduler must be able to run when the secondary server is running. If a server becomes active but cannot use its own scheduler, PBS will not be able to schedule jobs.

9.2.4.10 Same Data Service Account on Both Server Hosts

The PBS Data service management account must be the same on both server hosts. The UID of the PBS data service management account must be identical on both the primary and secondary server hosts. We recommend that the PBS data service management account is called *pbsdata*.

If you change either data service management account, both must be changed at the same time and both servers must be restarted. The name of the Data service account must be the same as the data service management account.

9.2.4.11 Data Service Host Configuration Requirement

The DATA_SERVICE_HOST parameter must not be set in pbs.conf. If this parameter is set, failover cannot take place.

9.2.4.12 Consistent User Names

User names must be consistent across the primary and secondary server hosts. If usernames are not consistent, jobs are killed.

9.2.4.13 Monitor Server Mail

Use the `qmgr` command to set the mail_from server attribute to an address that is monitored regularly:

```
Qmgr: s server mail_from=<address>
```

See section 2.2.1, "Configuring Server Mail Address", on page 19.

9.2.5 Configuring Failover

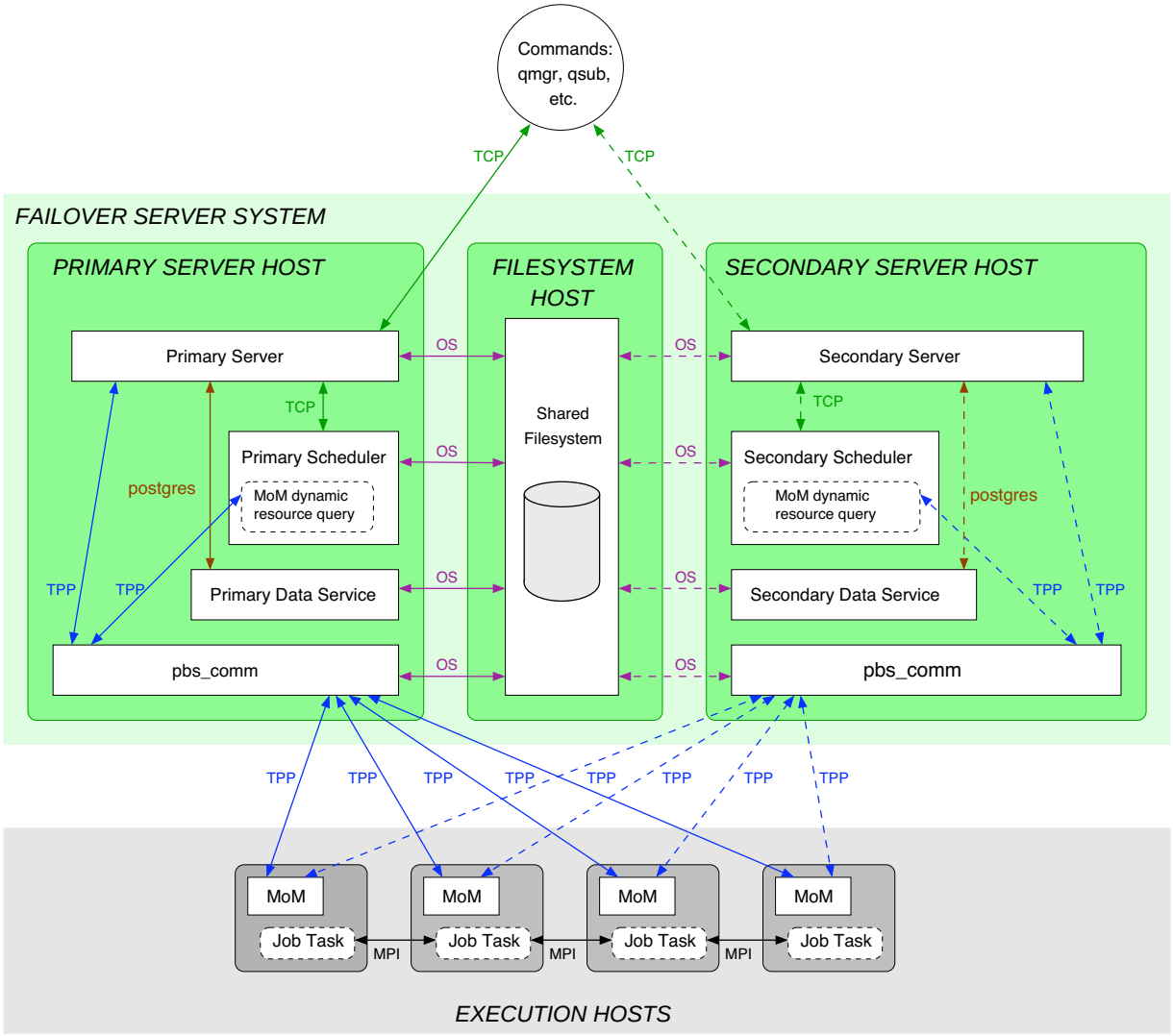


Figure 9-2:Failover Configuration

9.2.5.1 Overview of Configuring Failover

If PBS is not already installed, install it according to the PBS Professional Installation & Upgrade Guide.

Please make sure that you have satisfied all of the prerequisites under section 9.2.4, "Prerequisites for Failover", on page 404.

Make a copy of your PBS configuration. Follow the instructions in "Back Everything Up to Transfer Location" on page 97 in the PBS Professional Installation & Upgrade Guide.

The following table contains a guide to the steps in configuring PBS for failover. The table contains a link to the description of each step.

Table 9-3: Overview of Configuring Failover

Step	Linux	Windows
Configure `/etc/pbs.conf` on each host in the complex	See section 9.2.5.2, "Configuring the pbs.conf File for Failover", on page 409	
Configure the primary server	See section 9.2.5.3.i, "Configuring Failover For the Primary Server on Linux", on page 411	
Configure the secondary server	See section 9.2.5.3.ii, "Configuring Failover For the Secondary Server on Linux", on page 413	
Recommended: configure STONITH script	See section 9.2.5.3.iii, "Configuring STONITH Script for Use by Secondary Server", on page 413	
Configure execution and client hosts	See section 9.2.5.3.iv, "Configuring Failover For Execution and Client Hosts on Linux", on page 414	See section 9.2.5.4.i, "Configuring Failover for Execution and Client Hosts on Windows", on page 415
Configure failover with peer scheduling	See section 9.2.6.2, "Configuring Failover to Work With Peer Scheduling", on page 415	
Configure failover with routing queues	See section 9.2.6.1, "Configuring Failover to Work with Routing Queues", on page 415	
Configure failover with access control	See section 9.2.6.3, "Configuring Failover to Work With Access Controls", on page 416	

9.2.5.2 Configuring the `pbs.conf` File for Failover

The $PBS_CONF_FILE environment variable contains the path to the `pbs.conf` file. Each host in the complex must have a properly configured `/etc/pbs.conf` file. This file specifies the hostnames of the primary and secondary servers, the location of `PBS_HOME` and `PBS_MOM_HOME`, and whether to start a server, a scheduler, or a MoM on this host.

The name used for the server in the PBS_SERVER variable in the `pbs.conf` file must not be longer than 255 characters. If the short name for the server resolves to the correct host, you can use this in `pbs.conf` as the value of PBS_SERVER. However, if the fully-qualified domain name is required in order to resolve to the correct host, then the this must be the value of the PBS_SERVER variable.

Table 9-4: Parameters in `pbs.conf` for Failover

Parameters	Value	Meaning
PBS_EXEC	Path	Location of PBS `bin` and `sbin` directories
PBS_HOME	Path	Location of PBS working directories in shared filesystem; use specific path on that host
PBS_MOM_HOME	Path	Location of `mom_priv` on each host; overrides `PBS_HOME` for `mom_priv`
PBS_PRIMARY	FQDN of hostname	Hostname of primary server host. If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.
PBS_SECONDARY	FQDN of hostname	Hostname of secondary server host. If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
PBS_SERVER	Hostname	Name of primary server host. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER.
PBS_START_COMM	*0* or *1*	Specifies whether a comm is to run on this host
PBS_START_MOM	*0* or *1*	Specifies whether a MoM is to run on this host
PBS_START_SCHED	*0* or *1*	Specifies whether scheduler is to run on this host
PBS_START_SERVER	*0* or *1*	Specifies whether server is to run on this host

9.2.5.2.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

9.2.5.3 Host Configuration for Failover on Linux

- Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 404.
- PBS should already be installed in the default location on the primary and secondary server hosts and on the execution hosts. The client commands should already be installed on the client hosts.
- Make root a Manager on both server hosts:

  qmgr -c "set server managers =root@<primary server host>"

  qmgr -c "set server managers +=root@<secondary server host>"
- If the primary server and scheduler are running, shut them down. See “qterm” on page 233 of the PBS Professional Reference Guide.

9.2.5.3.i Configuring Failover For the Primary Server on Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 404.
2. Stop PBS on both the primary and secondary server hosts:

   On the primary server host:

   systemctl stop pbs

   or

   <path to init.d>/init.d/pbs stop

   On the secondary server host:

   systemctl stop pbs

   or

   <path to init.d>/init.d/pbs stop
3. On the primary server host, edit the `/etc/pbs.conf` file so that it DOES NOT include failover settings. It should look like this:

   ```
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>
   PBS_START_SCHED=1
   ```

   We recommend not running a MoM on any server host. The following setting in `pbs.conf` will prevent a MoM from running:

   ```
   PBS_START_MOM=0
   ```

   If you will run a MoM on the server hosts, specify this:

   ```
   PBS_START_MOM=1
   ```

   If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 405:

   ```
   PBS_MOM_HOME=<location of local, replicated mom_priv>
   ```
4. On the primary server host, start the primary PBS server and scheduler daemons:

   systemctl start pbs

or

<path to init.d>/init.d/pbs start

5. Stop the PBS server on the primary server host:

systemctl stop pbs

or

<path to init.d>/init.d/pbs stop

6. On the primary server host, edit the /etc/pbs.conf file to include the failover settings for PBS_PRIMARY and PBS_SECONDARY. It should look like this:

```
PBS_PRIMARY=<primary_host>
PBS_SECONDARY=<secondary_host>
PBS_SERVER=<short name for primary host>
PBS_HOME=<shared location of PBS_HOME>
```

The primary scheduler will start automatically:

```
PBS_START_SCHED=1
```

We recommend not running a MoM on any server host. The following setting in pbs.conf will prevent a MoM from running:

```
PBS_START_MOM=0
```

If you will run a MoM on the server hosts, specify this:

```
PBS_START_MOM=1
```

If you will run a MoM on both server hosts, specify PBS_MOM_HOME on this host. The location you specify is the directory that you replicated in section 9.2.4.3, "Requirements for MoMs on Server Hosts", on page 405:

```
PBS_MOM_HOME=<location of local, replicated mom_priv>
```

If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.

If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.

7. Run a comm on the primary server host. Set the following in pbs.conf on the primary server host:

```
PBS_START_COMM = 1
```

8. On the primary server host, start the primary PBS server, scheduler, and comm daemons:

systemctl start pbs

or

<path to init.d>/init.d/pbs start

9.2.5.3.ii Configuring Failover For the Secondary Server on Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 404.
2. On the secondary server host, edit the `/etc/pbs.conf` file to include the following settings:

   ```
   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<shared location of PBS_HOME>
   ```

   The secondary server will start its own scheduler if it needs to; a scheduler should not automatically start on the secondary server host. Include the following so that a scheduler does not automatically start on this host:

   ```
   PBS_START_SCHED=0
   ```

   We recommend not running a MoM on any server host. The following setting in `pbs.conf` will prevent a MoM from running:

   ```
   PBS_START_MOM=0
   ```

   If you will run a MoM on the server hosts, specify this:

   ```
   PBS_START_MOM=1
   ```

   If you will run a MoM on both server hosts, specify `PBS_MOM_HOME` on this host. The location you specify is the directory that you replicated in section 9.2.4.3, “Requirements for MoMs on Server Hosts”, on page 405:

   ```
   PBS_MOM_HOME=<location of local, replicated mom_priv>
   ```

   If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.

   If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
3. On the secondary server host, to change the delay time between failure of the primary server and activation of the secondary server from its default of 30 seconds, use the -F <delay> option on the secondary server's command line in the PBS start script on the secondary server host. Edit the `init.d/pbs` script so that the server is invoked with the -F <delay> option:

   ```
   pbs_server -F <delay>
   ```

   See “pbs_server” on page 108 of the PBS Professional Reference Guide.
4. Run a comm on the secondary server host. Set the following in `pbs.conf` on the secondary server host:

   ```
   PBS_START_COMM = 1
   ```
5. On the secondary server host, start the secondary PBS server and comm daemons:

   ```
   systemctl start pbs
   ```

   or

   ```
   <path to init.d>/init.d/pbs start
   ```

9.2.5.3.iii Configuring STONITH Script for Use by Secondary Server

We strongly recommend that before the secondary server becomes active, it prevents a race condition between the primary and secondary data services by calling a script which shuts down the primary server host. This script is called STONITH, for "shoot the other node in the head". If the script returns failure, the secondary server waits for 10 seconds, then calls the script again. The secondary server does not become active until the script returns success.

Requirements for STONITH:

- You must write the STONITH script, and put it in $PBS_HOME/server_priv/stonith.
- Permissions for the script should be *0755*.
- The STONITH script takes one argument, which is the hostname of the primary server. This hostname is the same as what is listed for PBS_PRIMARY in pbs.conf.
- The STONITH script returns zero for success, and non-zero for failure.

Note that you must supply the command used to power down the primary server host.

Example 9-1: Sample STONITH Script

```
#!/bin/bash
# This script powers down the primary server host.
# This script runs only on the secondary server host.
PBS_PRIMARY=$1
SECONDARY=`hostname`
POWERDOWN_CMD="<command to power down the primary server host>"

echo  "INFO: Secondary starting Stonith script.  Secondary server host is ${SECONDARY}."
echo  "INFO: This Stonith script will power down the primary server host."
echo  "INFO: Primary server host is ${PBS_PRIMARY}."

# Power down the primary server host
# You can also include a timeout, and check the value of the result.
# Example: timeout_result=$( { timeout 10 ${POWERDOWN_CMD} ${PBS_PRIMARY} ; } 2>&1 )
${POWERDOWN_CMD} ${PBS_PRIMARY}

if [ $? -eq 0 ] ; then
    echo "INFO: Stonith script succeeded in powering down primary server host ${PBS_PRIMARY}."
        exit 0
else
    echo "ERROR: Stonith script failed to power down primary server host ${PBS_PRIMARY}."
    exit 1
fi
```

9.2.5.3.iv Configuring Failover For Execution and Client Hosts on Linux

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, “Prerequisites for Failover”, on page 404.
2. On each execution or client host, configure the /etc/pbs.conf file to include the following parameters:

   ```
   PBS_PRIMARY=<primary_host>
   PBS_SECONDARY=<secondary_host>
   PBS_SERVER=<short name for primary host>
   PBS_HOME=<location of PBS_HOME>
   ```

   The pbs.conf files on execution hosts are already configured to start the MoM daemon only. Similarly, the pbs.conf files on client hosts are already configured to start no daemons.
3. On each execution host, restart the MoM:

   ```
   systemctl start pbs
   ```

or

```
<path to init.d>/init.d/pbs start
```

9.2.5.4 Host Configuration for Failover on Windows

9.2.5.4.i Configuring Failover for Execution and Client Hosts on Windows

1. Make sure that you have satisfied all of the prerequisites under section 9.2.4, "Prerequisites for Failover", on page 404.
2. On each execution or client host, specify the location of PBS_HOME for the primary server:

   ```
   pbs-config-add "PBS_HOME=\\<shared filesystem host>\pbs_home"
   ```

3. On each execution or client host, specify the primary and secondary server names in the pbs.conf file by running the following commands:

   ```
   pbs-config-add "PBS_SERVER=<short name of primary server host>"
   pbs-config-add "PBS_PRIMARY=<FQDN of primary server host>"
   pbs-config-add "PBS_SECONDARY=<FQDN of secondary server host>"
   ```

4. If this is an execution host, restart the MoM:

   ```
   net start pbs_mom
   ```

9.2.6 Configuring Failover with Other PBS Features

9.2.6.1 Configuring Failover to Work with Routing Queues

You must configure failover to work with routing queues which have destinations in another complex. No additional configuration is required for routing queues which have destinations in the same complex.

For a routing queue in one complex which points to a queue *Q1* in another PBS complex that is set up for failover, it is a good idea to specify both *Q1@primary.example.com* and *Q1@secondary.example.com* as destinations.

For example, if a routing queue has a destination queue at another complex's primary server:

```
Qmgr: set queue r66 route_destinations=workq@primary.example.com
```

you need to add the same queue at the other complex's secondary server:

```
Qmgr: set queue r66 route_destinations+=workq@secondary.example.com
```

See section 2.3.6, "Routing Queues", on page 25.

9.2.6.2 Configuring Failover to Work With Peer Scheduling

For peer queueing where the furnishing complex is set up for failover:

- You must list the furnishing queue at both primary and secondary servers. If the furnishing queue is *Q1*, the peer_queue line in the pulling complex's sched_config file must list *Q1@primary.example.com* and *Q1@secondary.example.com*

For peer queueing where the pulling complex is set up for failover:

- You must add *<manager>@primary.example.com* and *<manager>@secondary.example.com* to the list of managers at the furnishing server.

See section 4.9.31, "Peer Scheduling", on page 165.

9.2.6.3 Configuring Failover to Work With Access Controls

If you are using access control on the server (the acl_host_enable server attribute is set to *True* and the acl_hosts server attribute is specified), add the secondary server to the host list in acl_hosts:

```
Qmgr: s server acl_hosts+=<secondary server host>
```

See section 8.3.4, "ACLs", on page 363.

9.2.7 Using PBS with Failover Configured

9.2.7.1 Stopping Servers

To stop both servers when the primary server is active, and the secondary server is running and idle, do the following:

```
qterm -f
```

To stop the primary server and leave the secondary server idle:

```
qterm -i
```

To stop the secondary server only:

```
qterm -F
```

9.2.7.2 Starting Servers

After configuring the servers, you can start them in any order.

If you want to start the primary server when the secondary server is the active server, you do not need to stop the secondary. When the primary server starts, it informs the secondary that the secondary can become idle.

However, if there is a network outage while the primary starts and the secondary cannot contact it, the secondary will assume the primary is still down, and remain active, resulting in two active servers. In this case, stop the secondary server, and restart it when the network is working:

```
qterm -F
pbs_server
```

To restart the secondary server while it is the active server:

```
pbs_server -F -1
```

The secondary server makes one attempt to contact the primary server, and becomes active immediately if it cannot.

See "pbs_server" on page 108 of the PBS Professional Reference Guide and "qterm" on page 233 of the PBS Professional Reference Guide.

9.2.8 Recommendations and Caveats

- **Do not** start or stop the data service using anything except the `pbs_dataservice` command. Start or stop the data service using only the `pbs_dataservice` command.
- If you do not wish for the secondary server to take over, use the -i option to the `qterm` command when stopping the primary server.
- When the primary server is active, and the secondary server is running and idle, the `pbs start/stop` script stops the active server, but leaves the idle server running. This means that the idle server becomes the active server.
- `PBS_HOME` should not be on either server host
- Neither PBS server should be the NFS fileserver
- Each scheduler and data service must be able to run when its server is started, otherwise no jobs will be scheduled; each server can use only its own scheduler and data service.
- Just because servers are redundant, that doesn't mean that your complex is. Look for single points of failure.
- If the "*take over*" delay time specified with the `pbs_server -F` option is too long, there may be a period, up to that amount of time, when clients cannot connect to either server.
- If the "*take over*" delay time specified with the `pbs_server -F` option is too short and there are transient network failures, then the secondary server may attempt to take over while the primary server is still active.
- While the primary server is active and the secondary server is inactive, the secondary server will not respond to any network connection attempts. Therefore, you cannot status the secondary server to determine whether it is running.
- If the secondary server is running, and the primary server cannot contact the secondary server when the primary server is restarted, the primary assumes the secondary is not running and takes over. This can result in two servers running at once.

9.2.9 Troubleshooting Failover

9.2.9.1 PBS Does Not Start

- If you see the following error:

  ```
  "Failover is configured. Temporarily disable failover before running pbs_ds_password"
  ```

  This means that PBS was started for the first time with failover configured. PBS cannot be started for the first time with failover configured. Remove definitions for PBS_PRIMARY and PBS_SECONDARY from `pbs.conf` on the primary server host, start PBS, stop PBS, replace the definitions, and start PBS again.

9.2.9.2 Primary and Secondary Servers Both Running

If both servers are running, this may be because the primary server was stopped and then restarted, and while the primary was stopped, the secondary began to take over. While the secondary server was coming up, it was not able to receive the message from the primary server indicating that it should go idle, or it couldn't register with the primary.

To avoid this problem, use the -i option to the `qterm` command, which tells the secondary server to remain idle.

9.2.9.3 Primary or Secondary Server Fails to Start

It does not matter in which order the primary and secondary servers are started.

If the primary or secondary server fails to start with the error:

```
another server running
```

then check for the following conditions:

1. There may be lock files left in PBS_HOME/server_priv that need to be removed.

   The primary and secondary servers use different lock files:

   - primary: server.lock
   - secondary: server.lock.secondary

2. On Linux, the RPC lockd daemon may not be running. You can manually start this daemon by running as root:

   <path to daemon>/rpc.lockd

   Check that all daemons required by your NFS are running.

9.2.9.4 Primary Server Periodically Restarting

If the primary server keeps restarting, an unknown secondary server may be contacting it. This can happen when PBS_PRIMARY and PBS_SECONDARY are missing from pbs.conf, but a secondary server has been started.

9.2.9.5 Cannot Connect to Host

If you see an error message about not being able to connect to server, check the permissions of pbs_iff on the secondary server. The setuid bit may be wrong (permissions should be -rsxr-xr-x), or it may be on a shared filesystem that disallows setuid programs from running.

9.3 Checkpoint and Restart

PBS Professional allows you to configure MoM to checkpoint jobs using your scripts and checkpoint tools. In addition, users may manage their own checkpointing from within their application.

9.3.1 Glossary

Application Checkpoint

The application performs its own checkpointing when it receives the appropriate signal etc.

Checkpoint and Abort, checkpoint_abort

The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job uses the restart file when it resumes execution.

Restart

A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

Restart File

The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

Restart Script

The script that MoM runs to restart a job. This script is common to all jobs, and so must use the information in a job's restart file to restart the job.

Snapshot Checkpoint

The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes based on this restart file if the system experiences a problem during the job's subsequent execution.

9.3.2 How Checkpointing Works

When a job is checkpointed, MoM executes a checkpoint script. The checkpoint script saves all of the information necessary to checkpoint the job. If the checkpoint is for a snapshot, the job continues to run. If the job is checkpointed and aborted, PBS kills and requeues the job after checkpointing it.

When a job is restarted, MoM executes a restart script. The restart script uses the saved information to restore the job. The restart script also reads the $PBS_NODEFILE. The manner of restarting the job depends on how it was checkpointed:

- If the job was checkpointed during shutdown, the job becomes eligible to run when PBS is restarted, and will start from where it was checkpointed.
- If the job was checkpointed by the scheduler because it was preempted, the scheduler briefly applies a hold, but releases the hold immediately after checkpointing the job, and runs the restart script when the job is scheduled to run.
- If the job was checkpointed and held via the `qhold` command, the hold must be released via the `qrls` command for the job to be eligible to run. Then when the scheduler next runs the job, the restart script is executed, and the job runs from where it was checkpointed.

You can configure PBS to requeue jobs that were snapshot checkpointed while they ran, if the epilogue exits with a special value. These jobs are then restarted from the restart file. However, if you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

You can provide checkpointing for jobs using any combination of scripts that you write and third-party checkpointing tools such as Meiosys Checkpoint and BLCR (Berkeley Lab Checkpoint/Restart). You can configure PBS to trigger the scripts or tools, so that the scripts and/or tools create a job's restart file.

You can configure one behavior for snapshots, and another behavior for checkpoint and abort.

Some applications provide their own checkpointing, which is triggered, for example, when the application receives a signal or detects a change in a file.

9.3.2.1 Types of Checkpointing

9.3.2.1.i Checkpoint and Abort

Checkpoint and abort is used when a job is checkpointed before being killed. When the job is checkpointed, the following takes place:

- MoM runs the checkpoint_abort script; the checkpoint script or tool writes a restart file specific to that job
- The checkpoint_abort script terminates the job
- PBS requeues the job
- If the job was held via the `qhold` command, PBS applies a hold to the job (puts it in the *Held* state)

The job resumes execution based on the information in the restart file.

Checkpoint and abort is applied when:

- The `qhold` command is used on a job
- The server is shut down via `qterm -t immediate` or `qterm -t delay`
- The scheduler preempts a job using the checkpoint method

9.3.2.1.ii Snapshot Checkpoint

Snapshot checkpointing is used for checkpointing a job at regular intervals. The job continues to run. When the job is checkpointed, the following takes place:

- MoM runs the snapshot checkpoint script; the checkpoint script or tool writes a restart file specific to that job
- The job continues to execute

The job resumes execution based on this restart file if the system crashes or if the epilogue returns -2. See section 9.3.7.3, "Requeueing via Epilogue", on page 429.

The interval can be specified by the user via `qsub -c <checkpoint spec>`. You can specify a default interval, in the checkpoint_min queue attribute, or in the Checkpoint job attribute. See "qsub" on page 213 of the PBS Professional Reference Guide and "Job Attributes" on page 328 of the PBS Professional Reference Guide.

9.3.2.1.iii Application Checkpoint

Application checkpointing is when an application checkpoints itself. PBS can be used to trigger application checkpointing, but does not manage the checkpoint files or process. Application checkpointing can be triggered when the application receives a signal or detects a change in a file.

9.3.2.2 Events That Trigger Checkpointing

The following table lists the events that can trigger checkpointing, and the kind of checkpointing that is used.

Table 9-5: Events Triggering Checkpointing

Event	Type of Checkpointing Used	Description
The `qhold` command is used on a job	checkpoint_abort	See section 9.3.7.6, "Holding a Job", on page 430
Server shut down via `qterm -t immediate` or `qterm -t delay`	checkpoint_abort	See section 9.3.7.2, "Checkpointing During Shutdown", on page 429
Scheduler preempts a job using the checkpoint method	checkpoint_abort	See section 9.3.7.5, "Preemption Using Checkpoint", on page 430
Periodic checkpointing of a job, as specified by `qsub -c <checkpoint spec>`, or the queue's checkpoint_min attribute	Snapshot	See section 9.3.7.1, "Periodic Job Checkpointing", on page 429
Periodic checkpoint of an application, where checkpoint script triggers application checkpoint	Snapshot and application checkpoint	See section 9.3.7.7, "Periodic Application Checkpoint", on page 431
User sends application checkpoint signal, or user creates checkpoint trigger file	Application checkpoint	See section 9.3.7.8, "Manual Application Checkpoint", on page 431

9.3.2.3 Effect of Checkpointing on Jobs

When a job is checkpointed and aborted (requeued), its accumulated queue waiting time depends on how that time is calculated:

- If you are using eligible time, the accumulated waiting time is preserved
- If you are not using eligible time, the accumulated waiting time is lost

The job exit code for being checkpointed and aborted is *-12*, named *JOB_EXEC_CHKP*.

When a job is restarted, it runs on the same machine as it did when it was checkpointed.

9.3.2.4 Effect of Checkpointing on Job Resources

When a job is checkpointed and aborted, all of its resources are freed.

A snapshot checkpoint does not affect a job's resources.

9.3.2.5 Restarting a Job

When a job is restarted, MoM runs the restart script specified in the $action restart MoM parameter. This script looks in the checkpoint directory (see section 9.3.6.5, "Specifying Checkpoint Path", on page 428) for the restart file for that job. It uses the information in that file to restart the job.

For a job that was checkpointed and aborted because it was held, the job has had a hold placed on it so that it will not be eligible for execution until the hold is released. In order for a checkpointed and held job to be eligible for execution, the hold must be removed using the `qrls` command. The job's owner can remove a User hold, but other holds must be removed by a Manager or Operator. See "qrls" on page 180 of the PBS Professional Reference Guide.

If the job was preempted via checkpointing, the scheduler releases the hold on the job immediately after checkpointing the job. This will show up in the scheduler's log file, but the job will not appear to be held because the hold duration is very short.

A job that was checkpointed and requeued during shutdown is not held. This job is eligible for execution as soon as the necessary daemons are back up. See section 9.3.7.4, "Checkpointed Jobs and Server Restart", on page 430.

A job that was snapshot checkpointed and later requeued because the epilogue returned a special exit status is requeued in the *Q* state, and is eligible to be restarted when the scheduler selects it for execution.

When a checkpointed and aborted job is restarted, MoM resumes tracking the job. She tracks either the original PID of the job, or the PID of the restart script, depending on the setting of the $restart_transmogrify MoM parameter. See section 9.3.4.3, "Setting $restart_transmogrify MoM Parameter", on page 424.

9.3.3 Prerequisites for Checkpointing Jobs

The following are the prerequisites for checkpointing jobs:

- The MoM must be configured for checkpointing
  - Specified checkpoint directories must correspond to available directories (see section 9.3.6.5, "Specifying Checkpoint Path", on page 428)
  - Checkpoint and restart MoM configuration parameters must be specified (see section 9.3.4.2, "Specifying Checkpoint and Restart Parameters", on page 422)
- A checkpointing script or tool must be available for each type of checkpointing to be used

9.3.3.1 Restrictions on Checkpointing

- Checkpointing is not supported for job arrays.
- PBS does not directly support OS-level checkpointing.
- You can configure only one snapshot script, so if more than one kind of snapshot checkpointing is required, the script must distinguish which kind of snapshot to perform.
- You can configure only one checkpoint_abort script, so if more than one kind of checkpoint_abort is required, the script must also distinguish which kind of checkpoint_abort to perform.
- You can configure only one restart script. The restart script is run once for each of the job's tasks, so if some restarts are for application checkpointing, the script must handle those restarts correctly (application restarts may require only one iteration.)
- A restarted job must run on the same machine where it was running when it was checkpointed.
- Checkpointing cannot be used for interactive jobs. See section 9.3.8.2, “Sockets and Checkpointing”, on page 431.

9.3.4 Configuring Checkpointing

9.3.4.1 Overview of Configuring Checkpointing

You configure checkpointing by editing the MoM configuration file, `PBS_HOME/mom_priv/config`. You edit MoM configuration parameters to do the following:

- Specify script paths
  - Specify path to checkpoint_abort script, if needed
  - Specify path to snapshot script, if needed
  - Specify path to restart script
- Set $restart_transmogrify MoM parameter to fit your restart script
- Make the checkpoint path match that specified in the restart script

9.3.4.1.i Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

9.3.4.2 Specifying Checkpoint and Restart Parameters

To configure checkpointing, you specify a path to a script that MoM executes when checkpointing is called for. You can specify a separate path/script for each of checkpoint_abort, snapshot, and restart using the following MoM configuration parameters:

$action checkpoint timeout !path/script script-args

Specifies snapshot behavior.

$action checkpoint_abort timeout !path/script script-args

Specifies checkpoint_abort behavior.

$action restart timeout !path/script script-args

Specifies restart behavior.

where

$action

Specifies that MoM perform the indicated action.

checkpoint

MoM executes the script specified in path/script once for each of the job's tasks when a snapshot is called for.

checkpoint_abort

MoM executes the script specified in path/script once for each of the job's tasks when a checkpoint_abort is called for.

restart

MoM executes the script specified in path/script once for each of the job's tasks when a restart is called for.

timeout

The number of seconds allowed for the script or tool to execute. The value of the $restart_transmogrify MoM parameter determines whether this limit is applied. Values for $restart_transmogrify, and resulting behavior:

False

If the script/tool does not finish running during this time, it is killed and handled as if it had returned failure.

True

No timeout limit is applied.

path/script

The path to the script, including the name of the script. The path can be absolute or relative. If the path is relative, it is relative to `PBS_HOME/mom_priv`.

Examples of absolute paths and script names:

```
/usr/bin/checkpoint/snapshot
/usr/bin/checkpoint/checkpt-abort
/usr/bin/checkpoint/restart
```

script-args

These are the arguments to the script, if any.

PBS automatically expands some arguments to checkpoint and restart scripts. The following table lists the arguments that are expanded by PBS:

Table 9-6: Checkpoint Script Arguments Expanded by PBS

Argument	Description
%globid	Global ID (no longer used)
%jobid	Job ID
%sid	Session ID
%taskid	Task ID
%path	File or directory name to contain restart files

9.3.4.2.i Examples of Checkpoint and Restart Parameters

The following are examples of snapshot, checkpoint_abort, and restart MoM parameters:

```
$action checkpoint 60 !/usr/bin/checkpoint/snapshot %jobid %sid %taskid %path
$action checkpoint_abort 60 !/usr/bin/checkpoint/checkpt-abort %jobid %sid %taskid %path
$action restart 30 !/usr/bin/checkpoint/restart %jobid %sid %taskid %path
```

9.3.4.3 Setting $restart_transmogrify MoM Parameter

The $restart_transmogrify MoM parameter controls how MoM runs the restart script, and whether she expects to resume tracking the job's original PID or a new PID. When she runs a restart script, MoM forks a child process, which `exec()`s the start script. If $restart_transmogrify is *True*, the start script becomes the top task of the job. If $restart_transmogrify is *False*, the start script does not become the top task of the job.

If your restart script preserves the job's original PID, set $restart_transmogrify to *False*. This way, the script does not become the top task of the job, and MoM continues to track the job's original PID.

If your restart script results in a new PID for the job, set $restart_transmogrify to *True*. This way, the restart script becomes the top task of the job, and MoM tracks the PID of the new top process, which is the script.

9.3.5 Parameters and Attributes Affecting Checkpointing

9.3.5.1 MoM Configuration Parameters Affecting Checkpointing

$action checkpoint <timeout> !<script-path> <args>

Checkpoints the job, allowing the job to continue running.

$action checkpoint_abort <timeout> !<script-path> <args>

Checkpoints, kills, and requeues the job.

$action restart <timeout> !<script-path> <args>

Restarts checkpointed job.

The <timeout> is the time allowed for checkpoint or restart script to run.

$checkpoint_path <path>

MoM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to `PBS_HOME/mom_priv`. Overrides default. Overridden by path specified in the `pbs_mom -C` option and by PBS_CHECKPOINT_PATH environment variable.

$restart_background <*True*|*False*>

Specifies whether MoM runs the restart script in the background (MoM doesn't wait) or foreground (MoM waits). When set to *True*, MoM runs the restart script in the background.

Automatically set by MoM; Controlled by value of $restart_transmogrify. When $restart_transmogrify is *True*, $restart_background is set to *False*. When $restart_transmogrify is *False*, $restart_background is set to *True*.

Format: *Boolean*

Default: *False*

$restart_transmogrify <*True*|*False*>

Specifies which PID MoM tracks for a job that has been checkpointed and restarted.

When this parameter is set to *True*, MoM tracks the PID of the restart script. When this parameter is set to *False*, MoM tracks the PID of the original job.

The value of $restart_transmogrify controls the value of $restart_background.

Format: *Boolean*

Default: *False*

9.3.5.2 Options to pbs_mom Affecting Checkpointing

-C checkpoint_directory

Specifies the path to the directory where MoM creates job-specific subdirectories used to hold each job's restart files. MoM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods. Any directory specified with the -C option must be owned, readable, writable, and executable by root only (*rwx,---,---*, or *0700*), to protect the security of the restart files. See the -d option to pbs_mom.

Format: *String*

Default: PBS_HOME/checkpoint

9.3.5.3 Job Attribute Affecting Checkpointing

Checkpoint

Determines when the job will be checkpointed. Can take on one of the following values:

c

Checkpoint at intervals, measured in CPU time, set on the job's execution queue. If there is no interval set on the queue, the job is not checkpointed.

c=<minutes of CPU time>

Checkpoint at intervals of the specified number of minutes of job CPU time. This value must be greater than zero. If the interval specified is less than that set on the job's execution queue, the queue's interval is used.

Format: *Integer*

w

Checkpoint at intervals, measured in walltime, set on the job's execution queue. If there is no interval set at the queue, the job is not checkpointed.

w=<minutes of walltime>

Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less that that set on the execution queue in which the job resides, the queue's interval is used.

Format: *Integer*

n

No checkpointing.

s

Checkpoint only when the server is shut down.

u

Unset. Defaults to behavior when interval argument is set to *s*.

Default: *u*.

Format: *String*

9.3.5.4 Queue Attribute Affecting Checkpointing

checkpoint_min

Specifies the minimum number of minutes of CPU time or walltime allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in checkpoint_min is used for both CPU minutes and walltime minutes. See the Checkpoint job attribute.

Format: *Integer*

Default: None

Python attribute value type: pbs.duration

9.3.5.5 Environment Variable Affecting Checkpointing

PBS_CHECKPOINT_PATH

MoM passes this path to the checkpoint and restart scripts. Overridden by -c option to pbs_mom; overrides $checkpoint_path MoM parameter and default. See section 9.3.6.5, "Specifying Checkpoint Path", on page 428.

PBS_NODEFILE

PBS uses the $PBS_NODEFILE to restart the job. Make sure it is available.

9.3.5.6 The Epilogue

PBS will requeue a job which was snapshot checkpointed, if the epilogue returns the value *2*. See section 9.3.7.3, "Requeueing via Epilogue", on page 429.

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

9.3.6 Checkpoint and Restart Scripts

The restart script is run by the same MoM that ran the checkpoint script. The checkpoint and restart scripts are run for each task of the job. When MoM executes a checkpoint or restart script, she forks a child process, which exec()s the script. The restart script looks for the restart file in the job-specific subdirectory created by MoM, under the specified path. See section 9.3.6.5, "Specifying Checkpoint Path", on page 428.

9.3.6.1 Environment Variables for Scripts

PBS sets the following variables in the checkpoint and restart scripts' environments before running the scripts:

Table 9-7: Checkpoint/Restart Script Environment Variables

Environment Variable	Value of Variable
GID	Job owner's group ID
HOME	Job owner's PBS home directory
LOGNAME	Job owner's login name
PBS_JOBCOOKIE	128-bit random number used as token to authenticate job processes
PBS_JOBID	The job's ID
PBS_JOBNAME	The job's name
PBS_MOMPORT	Port number on which MoM listens for resource manager requests
PBS_NODEFILE	Path and filename of this job's node file
PBS_NODENUM	Index into the node file; index of this vnode; starts at 0
PBS_QUEUE	Name of the job's execution queue
PBS_SID	Session ID of task for which script is being called
PBS_TASKNUM	Index into task table for this job; index of task for which script is being called
SHELL	Job owner's login shell

Table 9-7: Checkpoint/Restart Script Environment Variables

Environment Variable	Value of Variable
UID	Job owner's execution ID
USER	Job owner's username
USERPROFILE	(Windows only) Job owner's Windows home directory
USERNAME	(Windows only) Job owner's Windows username

9.3.6.2 The Checkpoint Script

The checkpoint script writes a restart file that is specific to the job being checkpointed. The checkpoint script must save all of the information needed to restart the job. This is the information that will be used by the restart script to restart the job. PBS runs the script for each running job task, on each vnode where a task is running.

9.3.6.2.i Requirements for Checkpoint Script

- The first line of the script must specify the shell to be used, for example:
  `#!/bin/sh`
- The script should return the following error codes:
  - *Zero* for success
  - *Non-zero* for failure
- The script should block until the checkpoint process is finished.
- The restart file and its directory should be owned by root, and writable by root only, with permission 0755.
- Under Linux, the checkpoint script should be owned by root, and writable by root only, with permission 0755.
- Under Windows, the checkpoint script must have at least Full Control permission for the local Administrators group.
- The checkpoint script must write the restart file(s) in the location expected by the restart script. You don't have to use the %path parameter passed by MoM.
- If the script is for checkpoint-abort, the script must ensure that all processes are killed, whether directly or indirectly, for example by touching a file. All job processes must exit.

9.3.6.3 The Restart Script

The restart script does only one of the following:

- Reinstates the job's original PID, so that MoM tracks the original PID
- Becomes the new top process of the job, so that MoM tracks the PID of the script

If $restart_transmogrify is set to *True*, the restart script becomes the new top task for the job, and MoM begins tracking its process ID, where she was tracking the job's original process ID. If $restart_transmogrify is set to *False*, MoM continues to track the original job PID.

The restart script can use `pbs_attach()` to attach job processes to the original job PID, or to the script's PID. See "pbs_attach" on page 55 of the PBS Professional Reference Guide.

9.3.6.3.i Caveats for Restart Script

The `pbs_attach()` command is not supported under Windows.

9.3.6.3.ii Requirements for Restart Script

The restart script must handle everything required to restart the job from the information saved by the checkpoint script.

The restart script must block until the restart process is finished.

Under Linux, the restart script should be owned by root, and writable by root only, with permission 0755.

Under Windows, the restart script must have at least Full Control permission for the local Administrators group.

9.3.6.3.iii Return Values for Restart Script

The restart script must inform PBS of success or failure. It must return one of the following:

- *Zero* for success
- *Non-zero* for failure

9.3.6.4 Scripts for Application Checkpointing

If a user's application can be checkpointed periodically according to walltime or CPU time, you can use the PBS snapshot checkpoint facility to trigger snapshot checkpointing by the application.

If a user's application can be checkpointed, you can use the PBS checkpoint_abort facility before shutting down PBS to avoid losing intermediate results.

Some applications produce a restart file when they are sent a specific signal, or when a specific file is affected. A checkpoint script for this purpose sends the application the correct signal, or makes the correct change to the file.

Some applications only need the checkpoint and restart scripts to be run once each. In this case, the checkpoint and restart scripts should handle this requirement.

9.3.6.5 Specifying Checkpoint Path

When a job is checkpointed, information about the job is saved into a file. The location for this file can be any directory accessible to MoM.

The path to the checkpoint directory is composed of two parts. The first part is common to all jobs; this part can specified. The second part is a job-specific subdirectory, created by MoM for each job, under the common directory. The job's restart file is written in this job-specific subdirectory.

The default common directory, `PBS_HOME/checkpoint`, is provided for convenience.

You can specify the filename and the path for the common directory using any of the following methods. If the first is specified, PBS uses it. If not, and the second is specified, PBS uses the second, and so on.

- The -C path option to the `pbs_mom` command
- The PBS_CHECKPOINT_PATH environment variable
- The $checkpoint_path MoM configuration option in `PBS_HOME/mom_priv/config`
- The default value of `PBS_HOME/checkpoint`

The job-specific subdirectory is named with the following format:

<job ID>.CK

For example, if you specify `/usr/bin/checkpoint` for the common directory, and the job's ID is *1234.host1*, the job's restart file is written under `/usr/bin/checkpoint/1234.host1.CK`.

The restart file and its directory should be owned by root, and writable by root only.

9.3.6.5.i Checkpoint Path Caveats

If the checkpoint file is in `PBS_HOME/checkpoint/<job ID>.CK/`, and MoM thinks that a checkpoint failed (the checkpoint script returned non-zero), she will remove the checkpoint file. If the checkpoint script puts the checkpoint file in another location, MoM does not remove the checkpoint file.

9.3.7 Using Checkpointing

9.3.7.1 Periodic Job Checkpointing

If a job's Checkpoint attribute is set to *c*, *c=<minutes>*, *w*, or *w=<minutes>*, the job is periodically checkpointed. The checkpoint interval is specified either in the job's Checkpoint attribute or in the queue's checkpoint_min attribute. See "Job Attributes" on page 328 of the PBS Professional Reference Guide. The job's Checkpoint attribute is set using the -c <interval> option to the `qsub` command. See "qsub" on page 213 of the PBS Professional Reference Guide.

When this attribute is set, at every *<interval>* the job is checkpointed and a restart file is written, but the job keeps running.

9.3.7.2 Checkpointing During Shutdown

The effect on jobs of shutting down PBS depends on the method used to shut PBS down. When a job is checkpointed during shutdown, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. PBS does not hold the job, so the job is eligible to be run again as soon as the server starts up.

If you use the `qterm` command, there are three different suboptions to the -t option to control whether jobs are checkpointed, requeued, or allowed to continue running.

If you use the PBS start/stop script, the script affects only the host where the script is run. Any jobs running completely or partly on that host are killed and requeued, but not checkpointed. Any jobs not running on that host are left running.

The effect of each shutdown method is described here:

Table 9-8: Effect of Shutdown on Jobs

Shutdown Method	Effect on Checkpointable Jobs	Effect on Non-checkpointable Jobs
`qterm -t quick`	Continue to run	Continue to run
`qterm -t delay`	Checkpointed, killed, requeued, held	Requeued if rerunnable; continue to run if not rerunnable
`qterm -t immediate`	Checkpointed, killed, requeued, held	Requeued if rerunnable; deleted if not rerunnable

`systemctl stop pbs`
or
`init.d/pbs stop`
	Any jobs running completely or partly on host where stop script is run are killed and requeued
	Jobs not running on host where stop script is run are left running
		Any jobs running completely or partly on host where stop script is run are killed and requeued
		Jobs not running on host where stop script is run are left running

Any running subjobs of a job array keep running when the server is shut down.

9.3.7.3 Requeueing via Epilogue

You can configure MoM to requeue a failed job that was snapshot checkpointed during its execution. For example, if a job terminates, but had a hardware failure during execution, PBS can requeue the job, and MoM will run the start script, which can restart the job from its restart file.

When the job is requeued via the epilogue mechanism, it is in the *Q* state.

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

9.3.7.3.i Requirements for Requeueing via Epilogue

The following requirements must be met in order for a job to be requeued via the epilogue mechanism:

- The epilogue must return a value of *2*
- The job must have been checkpointed under the control of PBS
- The MoM must be configured with a restart script in the $action restart MoM configuration parameter
- The MoM must be configured to snapshot checkpoint jobs in the $action checkpoint MoM configuration parameter
- The jobs must request checkpointing via their Checkpoint attribute. See section 9.3.7.1, "Periodic Job Checkpointing", on page 429
- The epilogue script in `PBS_HOME/mom_priv/epilogue` must return the following:
  - *Zero* (*0*) for successful termination (requeue is not required)
  - *Two* (*2*) for failure (requeue is required)

9.3.7.4 Checkpointed Jobs and Server Restart

When the server is restarted using the `pbs_server -t warm` command, `systemd`, or the `init.d/pbs start` script, jobs that were checkpointed and aborted upon shutdown are waiting in their queues, and are eligible to be run according to the scheduler's algorithm.

When the server is restarted using the `pbs_server -t hot` command, jobs that were checkpointed and aborted upon shutdown are immediately rerun, before the scheduler selects which jobs to run.

9.3.7.5 Preemption Using Checkpoint

When a job is preempted via checkpointing, MoM runs the checkpoint_abort script, and PBS kills and requeues the job. When the scheduler elects to run the job again, the scheduler runs the restart script to restart the job from where it was checkpointed. For a description of using preemption, see section 4.9.33, "Using Preemption", on page 180.

9.3.7.6 Holding a Job

When anyone uses the `qhold` command to hold a checkpointable job, MoM runs the checkpoint_abort script, which kills all job processes, and PBS requeues, and holds the job.

A job with a hold on it must have the hold released via the `qrls` command in order to be eligible to run.

The following is the sequence of events when a job is held:

- MoM runs the checkpoint_abort script
- The job's execution is halted
- The resources assigned to the job are released
- The job is placed in the *Held* state in the execution queue
- The job's Hold_Types attribute is set appropriately

A held job is waiting in its queue. The following is the sequence of events when a held job is restarted:

- The hold is released by means of the `qrls` command; the job is now in the *Queued* state
- The job continues to wait in its queue until the scheduler schedules it for execution
- The scheduler selects the job for execution
- The job is sent to its original MoM for execution
- The MoM runs the restart script

9.3.7.6.i Restrictions on Holding a Job

A job in the process of provisioning cannot be held.

The `qhold` command can be used on jobs and job arrays, but not on subjobs or ranges of subjobs.

If the job cannot be checkpointed and aborted, `qhold` simply sets the job's Hold_Types attribute. The job continues to execute.

The checkpoint-abort script must terminate all job processes, or the `qhold` command will appear to hang.

9.3.7.7 Periodic Application Checkpoint

The snapshot checkpoint script can trigger checkpoint by a job's application, if the application is written to support checkpointing itself. Note that an application may be designed to be checkpointed at specific stages in its execution, rather than at specific points in time. If an application can be usefully checkpointed at specific points in time, then snapshot checkpointing may be useful. See section 9.3.7.1, “Periodic Job Checkpointing”, on page 429.

9.3.7.8 Manual Application Checkpoint

When an application is checkpointed manually, the user triggers checkpointing by the application by sending the application a specific signal, or by creating a file.

9.3.8 Advice and Caveats

9.3.8.1 PBS_NODEFILE Required

Make sure that the $PBS_NODEFILE is available during restart.

9.3.8.2 Sockets and Checkpointing

Multi-vnode jobs may cause network sockets to be opened between submission and execution hosts, and open sockets may cause a checkpointing script or tool to fail. The following use sockets:

- An interactive job, i.e. a job submitted using `qsub -I`, opens unprivileged sockets. `qsub` binds a socket to a port, then waits to accept a connection from MoM on that socket. Data from standard in is written to the socket and data from the socket is written to standard out.
- The `pbs_demux` process collects `stdio` streams from all tasks
- The `pbsdsh` program spawns tasks. The `-o` option to this command prevents it from waiting for spawned tasks to finish, so that no socket is left open to the MoM to receive task manager events. When the `-o` option is used, the shell must use some other method to wait for the tasks to finish. See “pbsdsh” on page 29 of the PBS Professional Reference Guide.

9.3.9 Accounting

If a job is checkpointed and requeued, the exit status passed to the epilogue and recorded in the accounting record is the following:

-12, meaning that the job was checkpointed and aborted

A checkpoint ("*C*") record is written in the accounting log when the job is checkpointed and requeued, as when the `qhold` command is used, or the job is checkpointed and aborted.

9.4 Reservation Fault Tolerance

If the vnodes associated with an advance reservation, the soonest occurrence of a standing reservation, or a job-specific reservation become unavailable, PBS marks the reservation as *degraded* (state *10*). If the vnodes are instead taken over by a maintenance reservation, PBS marks the reservation as *in conflict* (state *12*).

PBS attempts to reconfirm degraded or in-conflict reservations by finding replacements for vnodes that have become unavailable.

When a reservation is degraded, PBS may still be able to use the unavailable original vnodes, if they become available in time. When a reservation is in conflict, the vnodes that were taken over by the maintenance reservation are removed from the reservation; they are no longer in the reservation's resv_nodes attribute, and PBS looks for other vnodes.

States of available vnodes:

free

busy

job-exclusive

job-sharing

job-busy

States of unavailable vnodes:

down

maintenance

offline

provisioning

stale

state-unknown, down

unresolvable

wait-provisioning

9.4.1 States for Degraded and In-conflict Reservations

A degraded reservation's state becomes *RESV_DEGRADED*, abbreviated DG, and its substate becomes *RESV_DEGRADED*.

If vnodes associated with an occurrence later than the soonest occurrence of a standing reservation become unavailable, the reservation stays in state *RESV_CONFIRMED*, but its substate becomes *RESV_DEGRADED*.

During the time that a degraded advance or job-specific reservation, or the soonest occurrence of a degraded standing reservation is running, its state is *RESV_RUNNING*, and its substate is *RESV_DEGRADED*.

An in-conflict reservation's state becomes *RESV_IN_CONFLICT*, abbreviated IC, and its substate becomes *RESV_IN_CONFLICT*.

For a table of degraded and in-conflict reservation states and substates, see "Degraded Reservation Substates" on page 368 of the PBS Professional Reference Guide. For a table of numeric values for reservation states and substates, see "Reservation States" on page 367 of the PBS Professional Reference Guide.

9.4.2 Finding Replacement Vnodes for Degraded and In-conflict Reservations

PBS attempts to reconfirm reservations by finding replacements for vnodes that have become unavailable. If a reservation is not running, PBS will use any available vnodes. If it is running, PBS retains usable vnodes from the original reservation.

PBS attempts to reconfirm a reservation only during periods when this makes sense:

- If a reservation is not actively running, PBS waits the time specified in reserve_retry_time, then starts periodically trying to reconfirm the reservation, including making an attempt to reconfirm the reservation just before the start time of each occurrence.
- If an in-conflict reservation is actively running, PBS does not attempt to reconfirm it.
- If a degraded reservation is actively running, and the reservation is not in conflict, and the reservation has no running jobs on the unavailable vnodes, PBS periodically attempts to reconfirm it every reserve_retry_time seconds.
- If an actively running reservation is degraded because a vnode becomes unavailable, and the reservation has running jobs:
  - If the unavailable vnode has any jobs running on it, PBS waits until those jobs are finished to periodically attempt to reconfirm the reservation.
  - If the unavailable vnode has no jobs running on it, PBS does not wait until the jobs are finished to periodically attempt to reconfirm the reservation. PBS periodically attempts to reconfirm it every reserve_retry_time seconds.

A degraded or in-conflict reservation has a read-only reservation attribute called reserve_retry, whose value is the next time at which the reservation is due to be reconfirmed.

9.4.2.1 Attributes Affecting Reservation Reconfirmation

reserve_retry_time

Server attribute. The time period between attempts to reconfirm the reservation.

Settable by Manager; readable by all

Format: Integer (seconds)

Values: Must be greater than *zero*

Default: *600* (10 minutes)

Python attribute value type: int

9.4.3 Allocating New Vnodes

Once new vnodes are allocated for a reservation:

- The reservation has been confirmed
- If the reservation is not running, the state and substate of the reservation are *RESV_CONFIRMED*
- If the reservation is running, the state of the reservation is *RESV_RUNNING* and the substate is *RESV_CONFIRMED*
- The reservation's resv_nodes attribute lists the new vnodes

9.4.4 Restarting the Server

When the server is restarted, reservations are assumed confirmed until associated vnodes are recognized as unavailable. If any reservations become degraded or in conflict after a server restart, PBS sets the time when the reservation becomes degraded to the time of the restart. If a vnode is set *offline* before the restart, it is considered unavailable after the restart, so all its associated reservations become degraded.

9.5 Vnode Fault Tolerance for Job Start and Run

PBS lets you allocate extra vnodes to a job so that the job can successfully start and run even if some vnodes fail. You can allocate the extra vnodes only for startup, or for the life of the job. Later, for jobs where the extra vnodes are needed only for reliable startup, you can trim the allocated vnodes back to just what the job will use to run, releasing the unneeded vnodes for other jobs.

You allocate extra vnodes in a queuejob hook using the pbs.select.increment_chunks() method, and you release vnodes in an execjob_launch or execjob_prologue hook using the pbs.event().job.release_nodes() method.

We provide an example hook in `$PBS_EXEC/unsupported/ReliableJobStartup.py`.

9.5.1 Overview of Padding and Trimming Vnode Requests

Here is an overview of the steps for improving job startup and run reliability. We describe each of them in detail in the next subsections, and we give an example at the end of this section.

- Use a queuejob hook to do the following:
  - Save the job's initial vnode request
  - Set the job's tolerate_node_failures attribute to the desired value
  - Pad the job's vnode request
- Configure primary MoMs to wait for sister MoMs to acknowledge joining job
- Configure primary MoMs to wait for hooks to complete
- Use an execjob_launch or execjob_prologue hook to trim the vnodes not used by the job from the job's vnode request

9.5.2 Saving Job Initial Vnode Request

To save the job's initial resource request so that you know how much to trim later, use a queuejob hook to save it into a built-in resource such as the site resource (currently, it cannot be a custom resource). Here is a code snippet:

```
import pbs
e=pbs.event()
j = e.job
e.job.Resource_List["site"] = str(e.job.Resource_List["select"])
```

9.5.3 Configuring Primary MoMs to Wait for Sister MoMs

When the primary MoM gets a job whose tolerate_node_failures attribute is set to *all* or *job_start*, the primary MoM can wait to start the job for up to a configured number of seconds if the sister MoMs do not immediately acknowledge joining the job. This gives the sister MoMs more time to join the job. You configure the number of seconds for the primary MoM to wait for sister MoMs via the sister_join_job_alarm configuration parameter in MoM's config file:

```
$sister_join_job_alarm <number of seconds to wait>
```

The default value for this parameter is the sum of the values of the alarm attributes of any enabled execjob_begin hooks. If there are no enabled execjob_begin hooks, the default value is 30 seconds. For example, if there are two enabled execjob_begin hooks, one with alarm = 30 and one with alarm = 20, the default value of MoM's sister_join_job_alarm is 50 seconds.

After all the sister MoMs have joined the job, or MoM has waited for the value of the sister_join_job_alarm parameter, she starts the job.

9.5.4 Configuring MoMs to Wait for Hooks

When the primary MoM gets a job whose tolerate_node_failures attribute is set to *all* or *job_start*, the primary MoM can wait to start the job (running the job script or executable) for up to a configured number of seconds. During this time, execjob_prologue hooks can finish and the primary MoM can check for communication problems with sister MoMs. You configure the number of seconds for the primary MoM to wait for hooks via the job_launch_delay configuration parameter in MoM's config file:

```
$job_launch_delay <number of seconds to wait>
```

The default value for this parameter is the sum of the values of the alarm attributes of any enabled execjob_prologue hooks. If there are no enabled execjob_prologue hooks, the default value is 30 seconds. For example, if there are two enabled execjob_prologue hooks, one with alarm = 30 and one with alarm = 60, the default value of MoM's job_launch_delay is 90 seconds.

After all the execjob_prologue hooks have finished, or MoM has waited for the value of the job_launch_delay parameter, she starts the job.

9.5.4.1 Caveats for Configuring MoMs to Wait for Hooks

This configuration option is not supported under Windows.

9.5.5 Padding Vnode Request

To add extra vnodes to a job's vnode request, specify for the job whether you want more vnodes for startup, for the life of the job, or not at all, and specify how you want to pad the job's vnode request.

9.5.5.1 Specifying Whether and When to Pad Vnode Request

To specify whether and when the job gets extra vnodes, set the job's tolerate_node_failures attribute to one of *none*, *job_start*, or *all*.

Table 9-9: Behavior for tolerate_node_failures

Value of tolerate_node_failures	Behavior
none or unset	No extra vnodes are allocated to the job. Default behavior.
job_start	Extra vnodes are allocated only long enough to start the job successfully. Tolerate vnode failures that occur only during job start, just before executing the job's top level shell or executable or any execjob_launch hooks. Failures tolerated are those such as an assigned sister MoM failing to join the job and communication errors between MoMs.
all	Extra vnodes are allocated for the life of the job. Tolerate all node failures resulting from communication problems, such as polling problems, between the primary MoM and the sister MoMs assigned to the job Tolerate failures due to rejections from execjob_begin or execjob_prologue hooks run at sister MoMs.

9.5.5.1.i Setting the tolerate_node_failures Job Attribute

You or the job submitter can set the job's tolerate_node_failures attribute via `qsub`, `qalter`, or in a Python hook, for example a queuejob hook. If set via `qalter` while the job is already running, the attribute is consulted the next time the job is rerun.

You can set a value for tolerate_node_failures for all jobs via the server's default_qsub_arguments attribute.

Examples of setting this attribute:

- Via `qsub`:
  ```
  qsub -W tolerate_node_failures="all" <job script>
  ```
- Via `qalter`:
  ```
  qalter -W tolerate_node_failures="job_start" <job ID>
  ```
- Via a hook. The following code snippet shows how to set this attribute:
  ```
  # cat qjob.py
  import pbs
  e=pbs.event()
  e.job.tolerate_node_failures = "all"
  ```

9.5.5.2 Specifying How Chunks Are Padded

To specify how you want each chunk padded, use the pbs.select.increment_chunks(<increment specification>) method. This method increments the job's chunks according to the rules you give in the *increment specification*. See "Method to Increment select Object Chunks" on page 148 in the PBS Professional Hooks Guide.

9.5.5.2.i Example of Padding Chunks

The following code snippet illustrates padding a job's vnode request by one extra vnode per chunk:

```
import pbs
e=pbs.event()
j = e.job
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

9.5.5.3 Caveats for Padding Vnode Requests

The tolerate_node_failures job attribute is not supported on Cray systems. It is ignored on Cray systems.

9.5.6 Trimming Vnode Request

When you trim a job's vnode request, you can trim the larger padded amount back to the job's initial vnode request. To trim a job's vnode request, use the pbs.event().job.release_nodes(keep_select) method. This method automatically selects vnodes that satisfy the new request and are healthy, keeps them in the job's vnode request, and releases all others. The method automatically trims out any vnodes in the pbs.event().vnode_list_fail[] list.

You can call pbs.event().job.release_nodes(keep_select = <desired vnodes>) in an execjob_launch or execjob_prologue hook. Note that despite the method being named "release_nodes", it **keeps** the specified vnodes and **releases** all other vnodes. You can specify the job's original vnode request as the vnodes to keep.

The pbs.event().job.release_nodes() method returns a PBS job object which has the updated values for the job's exec_vnode and Resource_List attributes.

See "Job Object Method to Release Vnodes" on page 128 in the PBS Professional Hooks Guide.

9.5.6.1 Example of Trimming Job Vnode Request

Here we use an execjob_prologue hook to trim a job's vnode request:

```
pj = e.job.release_nodes(keep_select="ncpus=2:mem=2gb+ncpus=2:mem=2gb+ncpus=1:mem=1gb")
if pj != None:
   pbs.logmsg(pbs.LOG_DEBUG, "pj.exec_vnode=%s" % (pj.exec_vnode,))
else:               # returned None job object, so we can put a hold on the job and requeue it,
    rejecting the hook event
   e.job.Hold_Types = pbs.hold_types("s")
   e.job.rerun()
   e.reject("unsuccessful at LAUNCH")
```

9.5.6.2 Offlining Vnodes that Have Gone Bad During Start or Run

See "Using List of Failed Vnodes to Offline Vnodes that Have Gone Bad During Start or Run" on page 66 in the PBS Professional Hooks Guide.

9.5.7 Checking Vnodes and Marking Them as Failed

For each execjob_prologue and execjob_launch event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. PBS records this list in the pbs.event().vnode_list_fail[] object. See "The Failed Vnode List Event Member" on page 115 in the PBS Professional Hooks Guide.

Any sister vnodes that are able to join the job are considered healthy.

The successful outcome of a join job request may be the result of a check made by a remote execjob_begin hook. After successfully joining the job, the vnode may further check its status via a remote execjob_prologue hook. A rejection by the remote execjob_prologue hook causes the primary MoM to treat the sister vnode as a problem vnode, and the sister vnode is marked as unhealthy.

If there's an execjob_prologue hook in place, the primary MoM tracks vnode hosts that have acknowledged their execution of the execjob_prologue hook. Then after some job_launch_delay amount of time for job startup, the primary MoM starts reporting as failed vnodes those which have not given their positive acknowledgement during execjob_prologue hook execution.

If after some time, a vnode's host comes back with an acknowledgement of successful execjob_prologue hook execution, the primary MoM adds that host back to the healthy list.

You may want to offline any bad vnodes; see "Offlining Bad Vnodes" on page 66 in the PBS Professional Hooks Guide.

9.5.8 Example of Reliable Job Startup and Run

In order to have a job start reliably, we need these:

- A queuejob hook that does the following:
  - Makes the job tolerate vnode failures by setting the tolerate_node_failures job attribute to *job_start*
  - Adds extra chunks to the job's select specification using the pbs.event().job.select.increment_chunks() method
  - Saves the job's original vnode request into a built-in string resource (for example, "site")
- An execjob_launch hook that calls pbs.event().job.release_nodes() to trim the job's vnode request back to the original.

9.5.8.1 Example Queuejob Hook for Setup and Padding

We will use a queuejob hook called *qjob.py* in our example. In the queuejob hook:

- Make the job tolerant of failures:

```
import pbs
e=pbs.event()
j = e.job
j.tolerate_node_failures = job_start
```

- Save the job's initial vnode request in the built-in resource named *site*:

```
e.job.Resource_List["site"] = str(e.job.Resource_List["select"])
```

- Add extra chunks to the vnode request:

```
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

Instantiate the queuejob hook:

```
# qmgr -c "c h qjob event=queuejob"
# qmgr -c "i h qjob application/x-python default qjob.py"
```

9.5.8.2 Example Hook for Trimming

We will use an execjob_launch hook named *launch.py* to trim the job's padded vnode request back to the original vnode request. This hook runs before the job executes.

```
import pbs
e=pbs.event()
if 'PBS_NODEFILE' not in e.env:
    e.accept()
j = e.job
pj = j.release_nodes(keep_select=e.job.Resource_List["site"])
if pj is None:       # not successful pruning the vnodes
    j.rerun()        # rerun (requeue) the job
    e.reject("something went wrong pruning the job back to its original select request")
```

Instantiate the execjob_launch hook:

```
# qmgr -c "c h launch event=execjob_launch"
# qmgr -c "i h launch application/x-python default launch.py"
```

9.5.8.3 Example Job

Here is our example job:

```
% cat jobr.scr
#PBS -l select="ncpus=3:mem=1gb+ncpus=2:mem=2gb+ncpus=1:mem=3gb"
#PBS -l place=scatter:excl

echo $PBS_NODEFILE
cat $PBS_NODEFILE
echo END
echo "HOSTNAME tests"
echo "pbsdsh -n 0 hostname"
pbsdsh -n 0 hostname
echo "pbsdsh -n 1 hostname"
pbsdsh -n 1 hostname
echo "pbsdsh -n 2 hostname"
pbsdsh -n 2 hostname
echo "PBS_NODEFILE tests"
for host in `cat $PBS_NODEFILE`
do
    echo "HOST=$host"
    echo "pbs_tmrsh $host hostname"
    pbs_tmrsh $host hostname
    echo "ssh $host pbs_attach -j $PBS_JOBID hostname"
    ssh $host pbs_attach -j $PBS_JOBID hostname
done
```

9.5.8.4 Example of Job Vnode Assignment Padding and Trimming

When our job first starts, it is assigned 5 vnodes, because its select specification was modified by adding 2 vnodes:

```
% qstat -f 20
Job Id: 20.mars.example.com
...
exec_host = mars/0*3+jupiter/0*2+saturn/0*2+mercury/0+neptune/0
exec_vnode =
    (mars:ncpus=3:mem=1048576kb)+(jupiter:ncpus=2:mem=2097152kb)+(saturn:ncpus=2:mem=2097152kb)+
    (mercury:ncpus=1:mem=3145728kb)+(neptune:ncpus=1:mem=3145728kb)
Resource_List.mem = 11gb
Resource_List.ncpus = 9
Resource_List.nodect = 5
Resource_List.place = scatter:excl
Resource_List.select = ncpus=3:mem=1gb+2:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb
Resource_List.site = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb

tolerate_node_failures = job_start
```

Now jupiter and neptune go down, and just before the job runs its program, the execjob_launch hook executes and prunes the job's vnode assignment back to the original select request. Now the job has this vnode assignment:

```
% qstat -f 20
Job Id: 20.mars.example.com
...
exec_host = mars/0*3+saturn/0*2+mercury/0*2
exec_vnode =
    (mars:ncpus=3:mem=1048576kb)+(saturn:ncpus=2:mem=2097152kb)+(mercury:ncpus=1:mem=3145728kb)
Resource_List.mem = 6gb
Resource_List.ncpus = 6
Resource_List.nodect = 3
Resource_List.place = scatter:excl
Resource_List.select = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb

Resource_List.site = 1:ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+1:ncpus1:mem=3gb
```

A snapshot of the job's output shows the pruned list of vnodes:

```
/var/spool/PBS/aux/20.mars.example.com <-- updated contents of $PBS_NODEFILE
mars.example.com
saturn.example.com
mercury.example.com
END

HOSTNAME tests

pbsdsh -n 0 hostname
mars.example.com
pbsdsh -n 1 hostname
saturn.example.com
pbsdsh -n 2 hostname
mercury.example.com

PBS_NODEFILE tests
HOST=mars.example.com
pbs_tmrsh mars.example.com hostname
mars.example.com
ssh mars.example.com pbs_attach -j 20.mars.example.com hostname
mars.example.com
HOST=saturn.example.com
pbs_tmrsh saturn.example.com hostname
saturn.example.com
ssh saturn.example.com pbs_attach -j 20.mars.example.com hostname
saturn.example.com
HOST=mercury.example.com
pbs_tmrsh mercury.example.com hostname
mercury.example.com
ssh mercury.example.com pbs_attach -j 20.mars.example.com hostname
mercury.example.com
```

9.6 Preventing Communication and Timing Problems

9.6.1 Introduction

PBS communicates with remote execution hosts in order to track their availability and manage the jobs running on them. PBS is dependent upon your network for this communication. If there are network outages, or if the execution node becomes too busy for MoM to be able to respond to the server's queries, PBS will not be able to function properly. You can configure PBS to be better able to withstand these types of communication issues.

The following attributes and parameters control how PBS handles communication timing:

Table 9-10: Attributes and Parameters For Communication and Timing

Attribute or Parameter	Description	Cross Reference
Server Attributes		
job_requeue_timeout	Controls how long the process of requeueing a job is allowed to take	See section 9.6.3, "Setting Job Requeue Timeout", on page 445
node_fail_requeue	Controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job's primary execution host	See section 9.6.2, "Node Fail Requeue: Jobs on Failed Vnodes", on page 443
rpp_max_pkt_check	Maximum number of TPP messages processed by the main server thread per iteration. Default: 64	See "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide
rpp_retry	Server attribute. In a fault-tolerant setup (multiple `pbs_comms`), when the first `pbs_comm` fails partway through a message, this is number of times TPP tries to use any other remaining `pbs_comms` to send the message. Integer Valid values: *Greater than or equal to zero* Default: *10* Python type: *int*	See "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide
rpp_highwater	Server attribute. This is the maximum number of messages per stream (meaning the maximum number of messages between each pair of endpoints). Integer Valid values: *Greater than or equal to one* Default: *1024* Python type: *int*	See "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide
MoM Configuration Parameters		
$max_load	Vnode is considered to be *busy* if it is above this load.	See section 9.6.5, "Managing Load Levels on Vnodes", on page 445
$ideal_load	Vnode is considered to be not *busy* if it is below this load.	See section 9.6.5, "Managing Load Levels on Vnodes", on page 445
$prologalarm	Maximum number of seconds the prologue and epilogue may run before timing out	See section 9.6.6, "Prologue & Epilogue Running Time", on page 447
Queue Attributes		
route_retry_time	Interval between retries at routing a job	See section 9.6.7, "Time Between Routing Retries", on page 448

See "Robust Communication with TPP" on page 52 in the PBS Professional Installation & Upgrade Guide.

9.6.2 Node Fail Requeue: Jobs on Failed Vnodes

The node_fail_requeue server attribute controls how long the server waits before requeueing or deleting a job when it loses contact with the MoM on the job's primary execution host.

9.6.2.1 How Node Fail Requeue Works

You can specify how long the server waits after it loses contact with primary execution before deleting or requeueing her jobs. This behavior is controlled by the server's node_fail_requeue attribute.

This attribute's value is the delay between the time the server determines that the primary execution host MoM cannot be contacted and the time it requeues the job, and does not include the time it takes to determine that the host is out of contact.

If this attribute is set to a value other than zero, and the server loses contact with an execution host, all jobs for which this is the primary execution host are requeued or deleted at the same time.

If node_fail_requeue is unset, and the host where primary execution is running fails, the server assumes that the job is still running until one of the following happens:

- The primary execution host MoM comes back up and tells the server to requeue the job
- The job is manually rerun

9.6.2.2 Effect Of Requeueing On Jobs

When a job is thus requeued, it retains its original place in its execution queue with its former priority. The job is usually the next job to be considered during scheduling, unless the relative priorities of the jobs in the queue have changed. This can happen when the job sorting formula assigns higher priority to another job, another higher-priority job is submitted after the requeued job started, this job's owner has gone over their fairshare limit, etc.

Any resources that were being used by a job are freed when the job is requeued.

9.6.2.3 The node_fail_requeue Server Attribute

Format: *Integer*

9.6.2.3.i Allowable Values

The node_fail_requeue attribute can take these values:

Greater than zero

The server waits for the specified number of seconds after losing contact with a primary execution host MoM, then attempts to contact the primary execution host MoM, and if it cannot, requeues any jobs that can be rerun and deletes any jobs that cannot be rerun.

Zero

Jobs are not requeued; they are left in the *Running* state until the execution host MoM is recovered, whether or not the server has contact with their primary execution host MoM.

Less than zero

The attribute is treated as if it were set to *1*, and jobs are deleted or requeued after the server has been out of contact with the primary execution host MoM for 1 second.

Unset

Behaves as if set to the default value of *310*.

9.6.2.3.ii Default Value

The default value for this attribute is *310*, meaning that when the server loses contact with an execution host, it waits for 310 seconds after losing contact with the primary execution host MoM before requeueing or deleting jobs.

9.6.2.4 Where node_fail_requeue Applies

The server's node_fail_requeue attribute applies only in the case where the server loses contact with the primary execution host MoM.

When the primary execution host MoM loses contact with a sister MoM, the job is immediately deleted or requeued.

9.6.2.5 Jobs Eligible to be Requeued

Jobs are eligible to be requeued if they meet either of the following criteria:

- The job's Rerunable attribute is set to *y*
- The job did not begin execution, for example:
  - a multi-host job did not start on one or more vnodes
  - provisioning failed for the job

Jobs are ineligible to be requeued if their Rerunable attribute is set to *n* and they have started execution.

See "Job Attributes" on page 328 of the PBS Professional Reference Guide and "Server Attributes" on page 281 of the PBS Professional Reference Guide.

9.6.2.6 Using node_fail_requeue

The number of seconds selected should be long enough to exceed any transient non-vnode failures, but short enough to requeue the job in a timely fashion. Transient non-vnode failures can prevent MoM from reporting back to the server before the server marks the vnode *down*. These include:

- Network outages
- Vnode is too busy to respond, perhaps due to heavy swapping

Using this feature requires that you take the following into account:

- If the host where the primary execution host MoM is running fails, and node_fail_requeue is unset, the server assumes that the job is still running until one of the following happens:
  - The primary execution host MoM comes back up and tells the server to requeue the job
  - The job is manually rerun

  If your site has hosts that fail and are not monitored, failed jobs may go unnoticed for a long time.
- If your network has temporary failures, and node_fail_requeue is set to a duration shorter than the outage, jobs will be unnecessarily requeued. This can be especially annoying when the job has been running for days.

9.6.2.7 Advice and Caveats

- If your site experiences frequent network failures or your execution hosts are often too busy to respond to the server, it is recommended that you either set node_fail_requeue to a value greater than the time MoM is unavailable, or set it to *zero*. This way jobs won't be requeued just because the network had a temporary outage or the vnode was too busy. Choose a value greater than both the longest likely network outage time and the time MoM is unavailable. For example, one site has set the value to 10 minutes, and another has set it to 15 minutes (900 seconds) to avoid problems due to swapping.
- The value shown in the server log for the time between losing communication and requeueing a job is sometimes one or two seconds less than the specified value.
- If the server is restarted when node_fail_requeue is set to a given value, node_fail_requeue retains that value. If the server is started when node_fail_requeue is unset, node_fail_requeue reverts to its default value.

9.6.3 Setting Job Requeue Timeout

When jobs are preempted via requeueing, the requeue can fail if the job being preempted takes longer than the allowed timeout. The time for requeueing includes post-processing such as staging files out, deleting files, and changing the job's state from *R* to *Q*. See section 4.9.33, “Using Preemption”, on page 180. The time allowed for a job to be requeued is controlled by the job_requeue_timeout server attribute.

You can use `qmgr` to set the job_requeue_timeout server attribute to a value that works for the jobs at your site. This attribute is of type Duration, with a minimum allowed value of 1 second and a maximum allowed value of 3 hours. The default timeout is 45 seconds. See “Server Attributes” on page 281 of the PBS Professional Reference Guide.

9.6.4 Setting MoM Reconnection Timeout

When the primary execution host MoM detects that a sister mom has lost connectivity (e.g. MoM went down or the network is having problems) it waits for a specified amount of time for the sister to reconnect before it gives up and kills the job. You can configure the time the primary execution host MoM waits by setting MoM's $max_poll_downtime parameter in `PBS_HOME/mom_priv/config`. The default value is five minutes.

9.6.5 Managing Load Levels on Vnodes

An overloaded execution host may end up too busy for MoM to respond to the server's queries, and causing the server to mark the MoM as *down*.

PBS can track the state of each execution host, running new jobs on the host according to whether the host is marked *busy* or not.

This behavior is somewhat different from load balancing, described in section 4.9.27, “Using Load Balancing”, on page 156. In load balancing, the scheduler estimates how much load a job would produce, and will not place a job where doing so would put the load above the limit. When managing load levels on vnodes as described here, the scheduler uses the state of the vnode to determine whether to place a job on that vnode.

The state of the vnode is set by MoM, according to its load. You can set two load levels using the $max_load and $ideal_load MoM configuration parameters. When the load goes above $max_load, the vnode is marked as *busy*. When the load drops below $ideal_load, the vnode is marked *free*.

PBS does not run new jobs on vnodes under the following conditions:

- Vnodes that are marked *busy*
- Vnodes whose resources, such as ncpus, are already fully allocated
- Vnodes where the load is above $max_load, when load balancing is turned on. See section 4.9.27, “Using Load Balancing”, on page 156.
- Vnodes where running the job would cause the load to go above $max_load, when load balancing is turned on. See section 4.9.27, “Using Load Balancing”, on page 156.

The load used by MoM is the following:

- On Linux, it is the raw one-minute averaged "loadave" returned by the operating system
- On Windows, it is based on the processor queue length

The $max_load and $ideal_load MoM configuration parameters are also used for cycle harvesting (see section 4.9.9.6, “Cycle Harvesting Based on Load Average”, on page 121) and load balancing (see section 4.9.27, “Using Load Balancing”, on page 156.)

MoM checks the load average on her host every 10 seconds.

When a vnode's state changes, for example from *free* to *busy*, MoM informs the server.

9.6.5.1 Techniques for Managing Load

Whether or not you set $max_load, PBS will not run jobs requesting a total of more than the available number of CPUs, which is set in resources_available.ncpus. So for example if resources_available.ncpus is set to *4*, and a job running on the vnode has requested 2 CPUs, PBS will not run jobs requesting a total of more than 2 CPUs.

9.6.5.1.i Types of Workload

How you manage load depends on your workload. Some jobs do not lend themselves to sharing CPUs, but some jobs can share CPUs without being hindered. Most MPI jobs would be hindered if some processes had to wait because others were slowed by sharing a CPU. If you need a job to have reproducible timing, it cannot share a CPU. Certain single-vnode jobs that alternate between CPU usage and I/O can share a CPU without being slowed significantly, thereby increasing throughput.

9.6.5.1.ii How Not To Share CPUs

For vnodes primarily running jobs that would be slowed or invalidated by sharing a CPU, have PBS assign jobs according to the number of available CPUs, so that there is no sharing of CPUs. Set resources_available.ncpus to the number of available CPUs. Do not set $max_load or $ideal_load.

9.6.5.1.iii How To Share CPUs

For vnodes running only jobs that can share CPUs, you can have PBS manage jobs according to the load on the vnodes, not the number of CPUs. This is called oversubscribing the CPUs. Set resources_available.ncpus to a value greater than the actual number of CPUs, such as two or three times the actual number. Set $max_load to a reasonable value so that PBS will run new jobs until $max_load is reached. Set $ideal_load to the minimum load that you want on the vnode.

9.6.5.1.iv Suspending Jobs on Overloaded Vnodes

You can specify that MoM should suspend jobs when the load goes above $max_load, by adding the suspend argument to the $max_load parameter. See section , “$max_load <load> [suspend]”, on page 447. In this case, MoM suspends all jobs on the vnode until the load drops below $ideal_load, then resumes them. This option is useful only when the source of the load includes work other than PBS jobs. This option is not recommended when the load is due solely to PBS jobs, because it can lead to the vnode cycling back and forth between being overloaded, being marked busy, suspending all jobs, being marked free, then starting all jobs, being overloaded, and so on.

9.6.5.2 Caveats and Recommendations

- It is recommended that the value for $ideal_load be lower than the value for $max_load. The value for $ideal_load should be low enough that new jobs are not run before existing jobs are done using the vnode's spare load.
- If you set only one of $max_load and $ideal_load, for example you set $max_load, but not $ideal_load, PBS sets the other to the same value.
- Do not allow reservations on hosts where $max_load and $ideal_load are configured. Set the resv_enable vnode attribute on these hosts to *False*.
- If you are using cycle harvesting via load balancing, be careful with the settings for $ideal_load and $max_load. You want to make sure that when the workstation owner is using the machine, the load on the machine triggers MoM to report being busy, and that PBS does not start any new jobs while the user is working. See section 4.9.9.6, "Cycle Harvesting Based on Load Average", on page 121.

9.6.5.2.i Allowing Non-job Processes on Execution Host

If you wish to run non-PBS processes on a host, you can prevent PBS from using more than you want on that host. Set the $ideal_load and $max_load MoM configuration parameters to values that are low enough to allow other processes to use some of the host.

9.6.5.3 Load Configuration Parameters

$ideal_load <load>

MoM parameter. Defines the load below which the vnode is not considered to be busy. Used with the $max_load parameter.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

$max_load <load> [suspend]

MoM parameter. Defines the load above which the vnode is considered to be busy. Used with the $ideal_load parameter.

If the optional suspend argument is specified, PBS suspends jobs running on the vnode when the load average exceeds $max_load, regardless of the source of the load (PBS and/or logged-in users).

Example:

```
$max_load 3.5
```

Format: *Float*

Default: number of CPUs

9.6.6 Prologue & Epilogue Running Time

Each time the scheduler runs a job, it waits for the prologue to finish before it runs another job. In order to prevent a hung prologue from halting job execution, prologues and epilogues are only allowed to run for a specified amount of time before PBS kills them. The running time is specified in the $prologalarm MoM configuration parameter. The default value for this parameter is *30 seconds*.

9.6.6.1 Prologue Timeout Configuration Parameter

$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

```
$prologalarm 30
```

Format: *Integer*

Default: *30*

9.6.7 Time Between Routing Retries

If the network is flaky, PBS may not be able to route a job from a routing queue to the destination queue. If all destination queues for a routing queue are at capacity, a job in a routing queue remains where it is. The time between routing retries is controlled by the route_retry_time queue attribute.

If the network experiences long outages, you may wish to set the time between retries to a sufficiently long time that PBS is not wasting cycles attempting to route jobs.

If jobs in a routing queue are not being routed because the destination queues are full, and most jobs are long-running jobs, you may wish to set the time between retries so that attempts are infrequent. It is recommended that the time between retries be no longer than the longest time acceptable to have an open slot in an execution queue.

9.6.7.1 Routing Retry Attribute

route_retry_time

Time delay between routing retries. Typically used when the network between servers is down. Used only with routing queues.

Format: *Integer seconds*

Default: *30 seconds*

Python type: pbs.duration

9.7 Preventing File System Problems

9.7.1 Avoid Filling Location of Temp Files for PBS Components

When the location used by PBS components to store temporary files becomes full, various failures may result, including jobs not initializing properly. To help avoid this, you can set the root directory for these files to a location less likely to fill up. See section 15.8, “Temporary File Location for PBS Components”, on page 567.

In addition, we recommend periodic cleaning of this location.

9.7.2 Avoid Filling Filesystem with Log Files

You must avoid having log files fill up the available space. You may have to rotate and archive log files frequently to ensure that adequate space remains available. See “Adequate Space for Logfiles” on page 8 in the PBS Professional Installation & Upgrade Guide.

9.8 OOM Killer Protection

PBS automatically protects against OOM killers. If the system hosting a PBS daemon or data service runs low on memory, the system may use an out-of-memory killer (OOM killer) to terminate processes. The PBS daemons and data service are protected from being terminated by an OOM killer.

10 Using MPI with PBS

10.1 Integration with MPI

PBS Professional is integrated with several implementations of MPI. When PBS is integrated with an MPI, PBS can track resource usage, control jobs, clean up job processes, perform accounting for all of the tasks run under the MPI, and create TMPDIR on each of the job's hosts.

When PBS is not integrated with an MPI, PBS can track resource usage, clean up processes, and perform accounting only for processes running on the primary host. This means that accounting and tracking of CPU time and memory aren't accurate, and job processes on sister hosts cannot be signaled.

10.2 Prerequisites

Before you integrate an MPI with PBS, the MPI must be working by itself. For example, you must make sure that all required environment variables are set correctly for the MPI to function.

10.3 Types of Integration

PBS provides support for integration for many MPIs. You can integrate MPIs with PBS using the following methods:

- Intel MPI 4.0.3 on Linux uses `pbs_tmrsh` when it sees certain environment variables set. No other steps are required. See section 10.5, “Integrating Intel MPI 4.0.3 On Linux Using Environment Variables”, on page 453.
- Wrapping the MPI with a PBS-supplied script which uses the TM (task manager) interface to manage job processes. PBS supplies a master script to wrap any of several MPIs. See section 10.10, “Integration by Wrapping”, on page 455
- PBS supplies wrapper scripts for some MPIs, for wrapping those MPIs by hand. See section 10.13, “Integration By Hand”, on page 460
- For non-integrated MPIs, job scripts can integrate the MPIs on the fly using the `pbs_tmrsh` command. Note that a PBS job script that uses `mpirun` with `pbs_tmrsh` cannot be used outside of PBS. See section 10.9, “Integration on the Fly using the pbs_tmrsh Command”, on page 454 and "Integrating an MPI on the Fly", on page 83 of the PBS Professional User’s Guide.
- Some MPIs can be compiled to use the TM interface. See section 10.8, “Integration Using the TM Interface”, on page 454
- Some MPIs require users to call `pbs_attach` See section 10.13.5, “Integrating HPE MPI”, on page 465.
- Altair support can help integrate your MPI with PBS so that the MPI always calls `pbs_attach` when it calls `ssh`. If you would like to use this method, contact Altair support at www.pbsworks.com.

To integrate an MPI with PBS, you use just one of the methods above. The method you choose depends on the MPI. The following table lists the supported MPIs, how to integrate them, and gives links to the steps involved and any special notes about that MPI:

Table 10-1: List of Supported MPIs

MPI Name	Versions	Method	Integration Steps	MPI-specific Notes
HP MPI	1.08.03 2.0.0	Use `pbs_mpihp`	"Steps to Integrate HP MPI or Platform MPI"	"Integrating HP MPI and Platform MPI"
Intel MPI	4.0.3 on Linux	Set environment variables	"Integrating Intel MPI 4.0.3 On Linux Using Environment Variables"	None
Intel MPI	4.0.3 on Windows	Use wrapper script	"Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script"	None
Intel MPI	2.0.022 3 4	Use `pbsrun_wrap` (**wrapper is deprecated**)	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
LAM MPI	6.5.9 **Support is deprecated.**	Use `pbs_mpilam` and `pbs_lamboot`	"Wrapping LAM MPI 6.5.9"	"Integrating LAM MPI and Open MPI"
LAM MPI	7.0.6 7.1.1 **Support for LAM is deprecated.**	Compile with TM	"Integration Using the TM Interface"	"Integrating LAM MPI and Open MPI"
MPICH-P4	1.2.5 1.2.6 1.2.7 **Support for all is deprecated.**	Use `pbs_mpirun`	"Steps to Integrate MPICH-P4"	"Integrating MPICH-P4"
MPICH-GM (MPICH 1.2.6.14b)	**Support for wrapper is deprecated.**	Use `pbsrun_wrap`	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
MPICH-MX	**Support for wrapper is deprecated.**	Use `pbsrun_wrap`	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
MPICH2	1.0.3 1.0.5 1.0.7 on Linux	Use `pbsrun_wrap`	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
MPICH2	1.4.1p1 on Windows	Use wrapper script	"Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script"	None

Table 10-1: List of Supported MPIs

MPI Name	Versions	Method	Integration Steps	MPI-specific Notes
MVAPICH	1.2 **Support for wrapper is deprecated.**	Use `pbsrun_wrap`	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
MVAPICH2	1.8	Use `pbsrun_wrap`	"Wrapping an MPI Using the pbsrun_wrap Script"	"Integration by Wrapping"
Open MPI	1.4.x	Compile with TM	"Integration Using the TM Interface"	"Integrating LAM MPI and Open MPI"
Platform MPI	8.0	Use `pbs_mpihp`	"Steps to Integrate HP MPI or Platform MPI"	"Integrating HP MPI and Platform MPI"
HPE MPI	Any	Optional: Use `mpiexec`, or users put `pbs_attach` in `mpirun` command line	"Steps to Integrate HPE MPI"	"Integrating HPE MPI"

10.4 Transparency to the User

Many MPIs can be integrated with PBS in a way that is transparent to the job submitter. This means that a job submitter can use the same MPI command line inside and outside of PBS. All of the MPIs listed above can be made to be transparent.

10.5 Integrating Intel MPI 4.0.3 On Linux Using Environment Variables

You can allow Intel MPI 4.0.3 to automatically detect when it runs inside a PBS job and use `pbs_tmrsh` to integrate with PBS. When it has detected that it is running in a PBS job, it uses the hosts allocated to the job.

On hosts running Intel MPI 4.0.3 that have `PBS_EXEC/bin` in the default PATH, set the following environment variables in `PBS_HOME/pbs_environment`:

```
I_MPI_HYDRA_BOOTSTRAP=rsh
I_MPI_HYDRA_BOOTSTRAP_EXEC=pbs_tmrsh
```

On hosts running Intel MPI 4.0.3 that do not have PBS_EXEC/bin in their default PATH, use the full path to `pbs_tmrsh`. For example:

```
I_MPI_HYDRA_BOOTSTRAP_EXEC=/opt/pbs/bin/pbs_tmrsh
```

The default process manager for Intel MPI 4.0.3 is Hydra.

10.5.1 Restrictions for Intel MPI 4.0.3

The unwrapped version of Intel MPI 4.0.3 `mpirun` on Linux does not support MPD.

10.6 Integrating Intel MPI 4.0.3 on Windows Using Wrapper Script

This version of PBS provides a wrapper script for Intel MPI 4.0.3 on Windows. The wrapper script is named `pbs_intelmpi_mpirun.bat`, and it is located in `$PBS_EXEC\bin`. This script uses `pbs_attach` to attach MPI tasks to a PBS job. You do not need to take any steps to integrate Intel MPI on Windows; job submitters must call the wrapper script inside their job scripts.

10.7 Integrating MPICH2 1.4.1p1 on Windows Using Wrapper Script

This version of PBS provides a wrapper script for MPICH2 1.4.1p1 on Windows. The wrapper script is named `pbs_mpich2_mpirun.bat`, and it is located in `$PBS_EXEC\bin`. This script uses `pbs_attach` to attach MPI tasks to a PBS job. You do not need to take any steps to integrate Intel MPI on Windows; job submitters must call the wrapper script inside their job scripts.

10.8 Integration Using the TM Interface

PBS provides an API to the PBS task manager, or TM, interface. You can configure an MPI to use the PBS TM interface directly.

When a job process is started on a sister host using the TM interface, the sister host's MoM starts the process and the primary host's MoM has access to job process information.

An MPI that we know can be compiled with the TM interface is Open MPI.

10.9 Integration on the Fly using the `pbs_tmrsh` Command

If using a non-integrated MPI, job submitters can integrate an MPI on the fly by using the `pbs_tmrsh` command. This command emulates `rsh`, but uses the TM interface to talk directly to `pbs_mom` on sister hosts. The `pbs_tmrsh` command informs the primary and sister MoMs about job processes on sister hosts. PBS can track resource usage for all job processes.

Job submitters use this command by setting the appropriate environment variable to `pbs_tmrsh`. For example, to integrate MPICH, set P4_RSHCOMMAND to `pbs_tmrsh`. For details, see "Integrating an MPI on the Fly", on page 83 of the PBS Professional User's Guide.

The following figure illustrates how a the pbs_tmrsh command can be used to integrate an MPI on the fly:

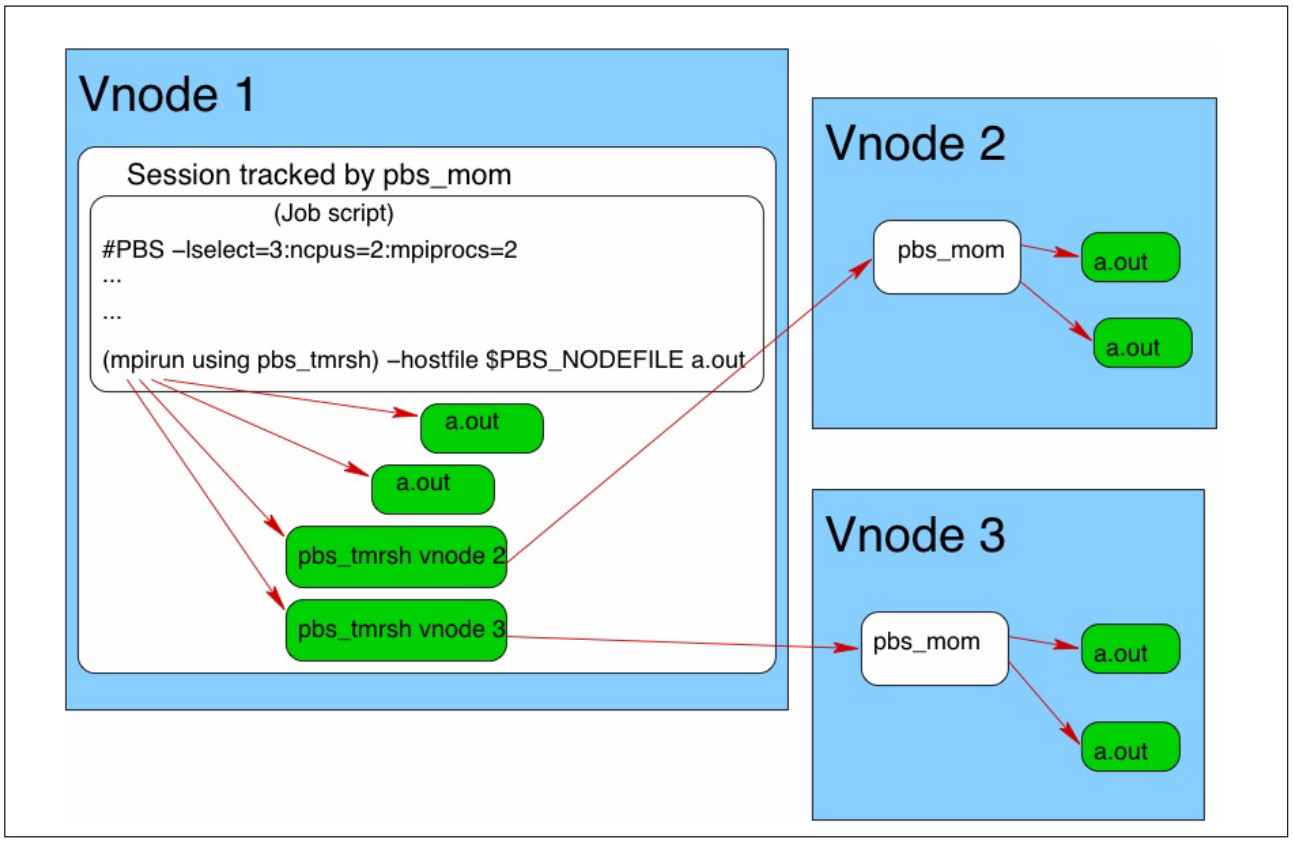


Figure 10-1: PBS knows about processes on vnodes 2 and 3, because pbs_tmrsh talks directly to pbs_mom, and pbs_mom starts the processes on vnodes 2 and 3

10.9.1 Caveats for the pbs_tmrsh Command

- This command cannot be used outside of a PBS job; if used outside a PBS job, this command will fail.
- The pbs_tmrsh command does not perform exactly like rsh. For example, you cannot pipe output from pbs_tmrsh; this will fail.

10.10 Integration by Wrapping

Wrapping an MPI means replacing its mpirun or mpiexec with a script which calls the original executable and, indirectly, pbs_attach. Job processes are started by rsh or ssh, but the pbs_attach command informs the primary and sister MoMs about the processes, so that PBS has control of the job processes. See "pbs_attach" on page 55 of the PBS Professional Reference Guide.

PBS provides a master script called pbsrun_wrap that you use to wrap many MPIs. PBS supplies special wrapper scripts so that you can wrap other MPIs by hand.

The following figure shows how a wrapped `mpirun` call works:

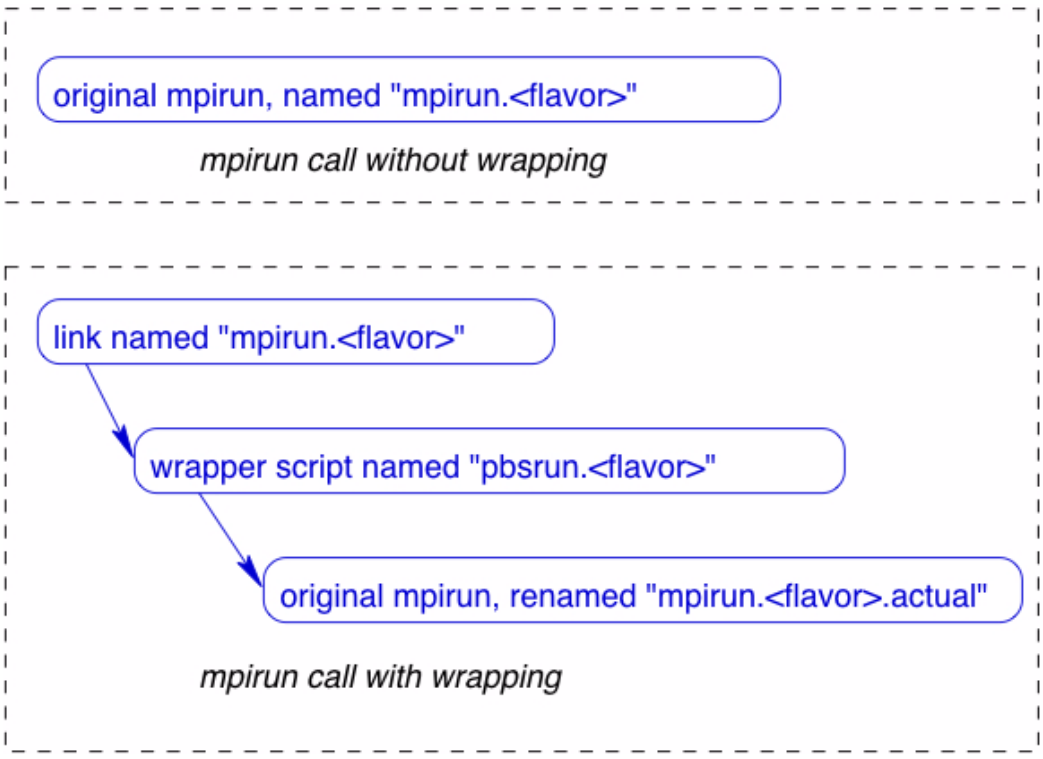


Figure 10-2:The job script calls the link that has the name of the original `mpirun`

10.10.1 Wrap the Correct Instance

Figure 10-3: The job script calls the link that has the name of the original `mpirun`

When you wrap an MPI, make sure that you are wrapping the first instance of the name found in the user's search path. This is the one returned by the 'which' command on Linux.

For example, on our example system `my_mpi` is installed as follows:

```
rwxrwxrwx 1 root system 31 Apr 18 19:21 /usr/bin/my_mpi -> /usr/my_mpi_dir/bin/my_mpi
```

And '`which`' returns the following:

```
bash-2.05b# which my_mpi
/usr/bin/my_mpi
```

Here, you must wrap the link, not the binary.

10.11 Wrapping an MPI Using the pbsrun_wrap Script

The master script is the pbsrun_wrap command, which takes two arguments: the mpirun to be wrapped, and a PBS-supplied wrapper. The pbsrun_wrap command neatly wraps the original mpirun so that everything is transparent for the job submitter. See "pbsrun_wrap" on page 51 of the PBS Professional Reference Guide, and "pbsrun" on page 40 of the PBS Professional Reference Guide.

The pbsrun_wrap command does the following:

- Renames the original, named *mpirun.<flavor>*, to *mpirun.<flavor>.actual*
- Instantiates the wrapper as *pbsrun.<flavor>*
- Creates a link named *mpirun.<flavor>* that calls pbsrun.<flavor>
- Creates a link so that pbsrun.<flavor> calls mpirun.<flavor>.actual

10.11.1 Passing Arguments

Any mpirun version/flavor that can be wrapped has an initialization script ending in "*.init*", found in $PBS_EXEC/lib/MPI:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init
```

When executed inside a PBS job, the pbsrun.<flavor> script calls a version-specific initialization script which sets variables to control how the pbsrun.<flavor> script uses options passed to it. For example, pbsrun.<flavor> calls $PBS_EXEC/lib/MPI/pbsrun.<flavor>.init to manage the arguments passed to it. You can modify the .init scripts to specify which arguments should be retained, ignored, or transformed.

When the mpirun wrapper script is run inside a PBS job, then it translates any mpirun call of the form:

mpirun [options] <executable> [args]

into

mpirun [options] pbs_attach [special_option_to_pbs_attach] <executable> [args]

where *[special options]* refers to any option needed by pbs_attach to do its job (e.g. -j $PBS_JOBID).

See "Options" on page 41 of the PBS Professional Reference Guide for a description of how to customize the initialization scripts.

10.11.2 Restricting MPI Use to PBS Jobs

You can specify that a wrapped MPI can be used only inside of PBS, by using the -s option to the pbsrun_wrap command. This sets the strict_pbs option in the initialization script (e.g. pbsrun.ch_gm.init, etc...) to *1* from the default of *0*. This means that the mpirun being wrapped by pbsrun will be executed only when it is called inside a PBS environment. Otherwise, the user gets the following error:

```
Not running under PBS
exiting since strict_pbs is enabled; execute only in PBS
```

By default, when the wrapper script is executed outside of PBS, a warning is issued about "not running under PBS", but it proceeds as if the actual program had been called in standalone fashion.

10.11.3 Format of pbsrun_wrap Command

The pbsrun_wrap command has this format:

pbsrun_wrap [-s] <path_to_actual_mpirun> pbsrun.<keyword>

Make sure that you wrap the correct instance of the mpirun. If a user's job script would call a link, wrap the link. See section 10.10.1, “Wrap the Correct Instance”, on page 456.

10.11.4 Actions During Wrapping

The pbsrun_wrap script instantiates the pbsrun wrapper script as pbsrun.<mpirun version/flavor> in the same directory where pbsrun is located, and sets up the link to the actual mpirun call via the symbolic link:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```

For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

- Save original mpirun.ch_gm script:
  ```
  mv /opt/mpich-gm/bin/mpirun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm.actual
  ```
- Instantiate pbsrun wrapper script as pbsrun.ch_gm:
  ```
  cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm
  ```
- Link mpirun.ch_gm to actually call pbsrun.ch_gm:
  ```
  ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm
  ```
- Create a link so that pbsrun.ch_gm calls mpirun.ch_gm.actual:
  ```
  ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link
  ```

10.11.5 Requirements

The mpirun being wrapped must be installed and working on all the vnodes in the PBS cluster.

10.11.6 Caveats and Restrictions

- For MPIs that are wrapped using pbsrun_wrap, the maximum number of ranks that can be launched in a job is the number of entries in the $PBS_NODEFILE.
- MVAPICH2 must use the "mpd" process manager if it is to be integrated with PBS. During the configuration step when you build MVAPICH2, set the "process manager" setting to *mpd*, as follows:
  ```
  --with-pm=mpd
  ```
  Other process managers such as "hydra" and "gforker" may not work correctly with PBS.
- If you wrap a version of Intel MPI mpirun less than 4.0.3, Hydra is not supported.
- Wrapping Intel MPI is **deprecated**.

10.11.7 Links to Wrapper Script Information

The following table lists the links to the description of each wrapper script used by `pbsrun_wrap`:

Table 10-2: Links to Wrapper Descriptions

MPI Wrapper	Link to Description
MPICH2	See "MPICH2 mpirun: pbsrun.mpich2" on page 46 of the PBS Professional Reference Guide.
MPICH-GM with MPD **Wrapper is deprecated.**	"MPICH-GM mpirun (mpirun.mpd) with MPD: pbsrun.gm_mpd" on page 44 of the PBS Professional Reference Guide.
MPICH-GM with `rsh`/`ssh` **Wrapper is deprecated.**	"MPICH-GM mpirun (mpirun.ch_gm) with rsh/ssh: pbsrun.ch_gm" on page 43 of the PBS Professional Reference Guide.
MPICH-MX with MPD **Wrapper is deprecated.**	"MPICH-MX mpirun (mpirun.mpd) with MPD: pbsrun.mx_mpd" on page 45 of the PBS Professional Reference Guide.
MPICH-MX with `rsh`/`ssh` **Wrapper is deprecated.**	"MPICH-MX mpirun (mpirun.ch_mx) with rsh/ssh: pbsrun.ch_mx" on page 44 of the PBS Professional Reference Guide
MVAPICH **Wrapper is deprecated.**	"MVAPICH1 mpirun: pbsrun.mvapich1" on page 48 of the PBS Professional Reference Guide.
MVAPICH2	"MVAPICH2 mpiexec: pbsrun.mvapich2" on page 49 of the PBS Professional Reference Guide.
Intel MPI	"Intel MPI mpirun: pbsrun.intelmpi" on page 47 of the PBS Professional Reference Guide.

10.11.8 Wrapping Multiple MPIs with the Same Name

You may want more than one MPI environment with the same name, for example a 32-bit and a 64-bit version of MPICH2.

1. Create two new MPICH2 initialization scripts by copying that for MPICH2:

```
# cd $PBS_EXEC/lib/MPI
# cp pbsrun.mpich2.init.in pbsrun.mpich2_32.init.in
# cp pbsrun.mpich2.init.in pbsrun.mpich2_64.init.in
```

2. Then wrap them:

```
# pbsrun_wrap <path to 32-bit MPICH2>/bin/mpirun  pbsrun.mpich2_32
# pbsrun_wrap <path to 64-bit MPICH2>/bin/mpirun  pbsrun.mpich2_64
```

Calls to `<path to 32-bit MPICH2>/bin/mpirun` will invoke `/usr/pbs/bin/pbsrun.mpich2_32`. The 64-bit version is invoked with calls to `<path to 64-bit MPICH2>/bin/mpirun`.

10.11.9 See Also

See "pbsrun" on page 40 of the PBS Professional Reference Guide for a description of the `pbsrun` script. See "pbsrun_wrap" on page 51 of the PBS Professional Reference Guide for a description of the master wrapping script.

10.12 Unwrapping MPIs Using the pbsrun_unwrap Script

You can also use the matching pbsrun_unwrap command to unwrap the MPIs you wrapped using pbsrun_wrap.

For example, you can unwrap the two MPICH2 MPIs from 10.11.8 above:

```
# pbsrun_unwrap pbsrun.mpich2_32
# pbsrun_unwrap pbsrun.mpich2_64
```

See "pbsrun_unwrap" on page 50 of the PBS Professional Reference Guide.

10.13 Integration By Hand

For MPIs that must be wrapped by hand, PBS supplies wrapper scripts which call the original and use pbs_attach to give MoM control of jobs.

Wrapping an MPI by hand yields the same result as wrapping using pbsrun_wrap, but you must perform the steps by hand.

Wrapping by hand involves the following steps (which are the same steps taken by pbsrun_wrap):

- You rename the original MPI command
- You create a link whose name is the same as the original MPI command; this link calls the wrapper script
- You edit the wrapper script to call the original MPI command
- You make sure that the link to the wrapper script(s) is available to each user's PATH.

The following table lists MPIs, their wrapper scripts, and a link to instructions:

Table 10-3: Scripts for Wrapping MPIs by Hand

MPI Name	Script Name	Link to Instructions
HP MPI	pbs_mpihp	section 10.13.1, "Integrating HP MPI and Platform MPI", on page 460
LAM MPI 6.5.9 **Deprecated**.	pbs_mpilam	section 10.13.3, "Integrating LAM MPI and Open MPI", on page 461
MPICH **Wrapper is deprecated**.	pbs_mpirun	section 10.13.4, "Integrating MPICH-P4", on page 463
Platform MPI	pbs_mpihp	section 10.13.1, "Integrating HP MPI and Platform MPI", on page 460
HPE MPI	mpiexec	section 10.13.5, "Integrating HPE MPI", on page 465

10.13.1 Integrating HP MPI and Platform MPI

PBS supplies a wrapper script for HP MPI and Platform MPI called pbs_mpihp. The pbs_mpihp script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.

You can make pbs_mpihp transparent to users; see the instructions that follow.

10.13.2 Steps to Integrate HP MPI or Platform MPI

Make sure that you wrap the correct instance of the MPI. If a user's job script would call a link, wrap the link. See section 10.10.1, "Wrap the Correct Instance", on page 456.

The pbs_mpirun command looks for a link with the name PBS_EXEC/etc/pbs_mpihp that points to the HP mpirun. The pbs_mpihp command follows this link to HP's mpirun. Therefore, the wrapping instructions are different from the usual. See "pbs_mpihp" on page 76 of the PBS Professional Reference Guide for more information on pbs_mpihp.

1. Rename HP's mpirun:

```
cd <MPI installation location>/bin
mv mpirun mpirun.hp
```

2. Link the user-callable mpirun to pbs_mpihp:

```
cd <MPI installation location>/bin
ln -s $PBS_EXEC/bin/pbs_mpihp mpirun
```

3. Create a link to mpirun.hp from PBS_EXEC/etc/pbs_mpihp. pbs_mpihp will call the real HP mpirun:

```
cd $PBS_EXEC/etc
ln -s <MPI installation location>/bin/mpirun.hp pbs_mpihp
```

10.13.2.1 Setting Up rsh and ssh Commands

When wrapping HP MPI with pbs_mpihp, note that rsh is the default used to start the mpids. If you wish to use ssh or something else, be sure to set the following or its equivalent in $PBS_HOME/pbs_environment:

```
PBS_RSHCOMMAND=ssh
```

10.13.2.2 Restrictions and Caveats for HP MPI and Platform MPI

- The pbs_mpihp script can be used only on HP-UX and Linux.
- The HP mpirun or mpiexec must be in the job submitter's PATH.
- The version of the HP mpirun or mpiexec must be HPMPI or Platform.
- Under the wrapped HP MPI, the job's working directory is changed to the user's home directory.

10.13.3 Integrating LAM MPI and Open MPI

The 7.x LAM MPI and all Open MPI versions allow you to compile the MPI with the PBS TM interface. We recommend compiling 7.x LAM MPI and all Open MPI versions with the TM module. You can either compile the later LAM with TM or wrap it, but not both. (You can wrap the newer versions if you want, but compiling yields better results.)

All versions of LAM MPI and Open MPI can be transparent to the job submitter.

The 6.5.9 version of LAM MPI requires wrapping for integration.

Support for LAM MPI is **deprecated**.

10.13.3.1 Compiling LAM MPI 7.x/Open MPI with the TM Module

To integrate 7.x LAM MPI with PBS, compile it with the --with-boot-tm=/usr/pbs option. Next, check laminfo to confirm that the the SSI line that says *tm* is there.

If the TM interface library is in the standard location, PBS_EXEC/lib/, Open MPI will find it and use it. You need to explicitly configure with TM only if it's in a non-standard location.

To integrate Open MPI with PBS, configure Open MPI with the `--with-tm` command-line option to the `configure` script. For example:

```
./configure --prefix=/opt/openmpi/1.4.4 --with-tm=${PBS_EXEC}
make
make install
```

After you compile LAM MPI or Open MPI on one host, make it available on every execution host that will use it, by means of shared file systems or local copies.

For the Open MPI website information on compiling with the TM option, see:

`http://www.open-mpi.org/faq/?category=building#build-rte-tm`

10.13.3.2 Wrapping LAM MPI 6.5.9

PBS provides wrapper scripts so that you can integrate LAM MPI 6.5.9 with PBS by hand. The `pbs_mpilam` script is used in place of `mpirun`, and the `pbs_lamboot` script replaces `lamboot`. The `pbs_lamboot` and `pbs_mpilam` scripts allow PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes. You make LAM calls transparent to the user by allowing them to use unchanged `lamboot` and `lamhalt` calls in their scripts.

The PBS command `pbs_lamboot` replaces the standard `lamboot` command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts. Usage is the same as for LAM `lamboot`. All arguments except for `bhost` are passed directly to `lamboot`. PBS will issue a warning saying that the `bhost` argument is ignored by PBS since input is taken automatically from $PBS_NODEFILE. The `pbs_lamboot` command can be instructed to boot the hosts using the `tm` module by setting the LAM_MPI_SSI_boot environment variable to *tm*, or by passing an argument to `pbs_lamboot` that contains "`-ssi boot tm`". In this case, the `pbs_lamboot` program does not redundantly consult the $PBS_NODEFILE.

The PBS command `pbs_mpilam` replaces the standard `mpirun` command in a PBS LAM MPI job, for executing programs. It attaches the user's processes to the PBS job. This allows PBS to collect accounting information, and to manage the processes. Usage is the same as for LAM `mpirun`. All options are passed directly to `mpirun`. If the where argument is not specified, `pbs_mpilam` will try to run the user's program on all available CPUs using the *C* keyword.

Make sure that you wrap the correct instance of the MPI. If a user's job script would call a link, wrap the link. See section 10.10.1, “Wrap the Correct Instance”, on page 456.

- You wrap LAM `lamboot` using `pbs_lamboot`.
  a. Install LAM MPI into `/usr/local/lam-6.5.9`.
  b. Rename LAM `lamboot` to *lamboot.lam*:
     ```
     mv /usr/local/lam-6.5.9/bin/lamboot /user/local/lam-6.5.9/bin/lamboot.lam
     ```
  c. Edit `pbs_lamboot` to change "`lamboot`" call to "`lamboot.lam`":
  d. Create a link for `pbs_lamboot` named *lamboot*:
     ```
     cd /usr/local/lam-6.5.9/bin
     ln  -s PBS_EXEC/bin/pbs_lamboot lamboot
     ```

At this point, using "lamboot" will actually invoke pbs_lamboot.

- You wrap LAM mpirun using the pbs_mpilam script.
  a. Install LAM MPI into /usr/local/lam-6.5.9.
  b. Rename LAM mpirun to *mpirun.lam*:
     ```
     mv /usr/local/lam-6.5.9/bin/mpirun /user/local/lam-6.5.9/bin/mpirun.lam
     ```
  c. Edit pbs_mpilam to change "mpirun" call to "mpirun.lam"
  d. Create a link for pbs_mpilam named *mpirun*:
     ```
     cd /usr/local/lam-6.5.9/bin
     ln -s PBS_EXEC/bin/pbs_mpilam mpirun
     ```

For more information on pbs_lamboot and pbs_mpilam, see "pbs_lamboot" on page 68 of the PBS Professional Reference Guide and "pbs_mpilam" on page 78 of the PBS Professional Reference Guide.

10.13.3.3 Setting up rsh and ssh Commands

If you intend to use ssh, you should set either LAMRSH or LAM_SSI_rsh_agent to the value "*ssh -x*", except under SuSE Linux, where it should be *"ssh -n"*.

10.13.3.4 Setting up Environment Variables

Set the LAM_MPI_SSI_boot environment variable to *tm* so that pbs_lamboot boots the hosts from the tm module.

10.13.3.5 Verifying Use of TM Interface

To see whether your Open MPI installation has been configured to use the TM interface:

```
% ompi_info | grep tm
MCA ras: tm (MCA v2.0, API v2.0, Component v1.3)
MCA plm: tm (MCA v2.0, API v2.0, Component v1.3)
```

10.13.3.6 See Also

See www.lam-mpi.org for more information about LAM MPI.

See http://www.open-mpi.org/faq/?category=building#build-rte-tm for information about building Open MPI with the TM option.

10.13.4 Integrating MPICH-P4

Wrapper is deprecated. PBS supplies a wrapper script called pbs_mpirun for integrating MPICH-P4 with PBS by hand. The pbs_mpirun script allows PBS to clean up job processes, track and limit job resource usage, and perform accounting for all job processes.

You can make pbs_mpirun transparent to job submitters. See the following steps.

10.13.4.1 Restrictions

- The pbs_mpirun command can be used only with MPICH using P4 on Linux.
- User names must be identical across hosts.

10.13.4.2 Options for pbs_mpirun

The usage for pbs_mpirun is the same as mpirun except for the listed options. All other options are passed directly to mpirun:

-machinefile

The value for this option is generated by pbs_mpirun. The value used for the -machinefile option is a temporary file created from the PBS_NODEFILE in the format expected by mpirun.

If the -machinefile option is specified on the command line, a warning is output saying "Warning, -machinefile value replaced by PBS".

-np

The default value for the -np option is the number of entries in PBS_NODEFILE.

10.13.4.3 Steps to Integrate MPICH-P4

To make pbs_mpirun transparent to the user, replace standard mpirun with pbs_mpirun. Make sure that you wrap the correct instance of the MPI. If a user's job script would call a link, wrap the link. See section 10.10.1, "Wrap the Correct Instance", on page 456.

- Install MPICH-P4 into <path to mpirun>
- Rename mpirun to mpirun.std:
  mv <path to mpirun>/mpirun <path to mpirun>/mpirun.std
- Create link called mpirun in <path to mpirun> that points to pbs_mpirun
  ln -s <path to pbs_mpirun>/pbs_mpirun mpirun
- Edit pbs_mpirun to change the call to mpirun so that it calls mpirun.std

At this point, using mpirun actually invokes pbs_mpirun.

10.13.4.4 Setting Up Environment Variables and Paths

- For pbs_mpirun to function correctly for users who require the use of ssh instead of rsh, you can do one of the following:
  - Set PBS_RSHCOMMAND in the login environment
  - Set P4_RSHCOMMAND externally to the login environment, then have job submitters pass the value to PBS via qsub(1)'s -v or -V arguments:

    qsub -vP4_RSHCOMMAND=ssh ...

    or

    qsub -V ...
  - Set P4_RSHCOMMAND in the pbs_environment file in PBS_HOME and then advise users to not set P4_RSHCOMMAND in the login environment
- Make sure that PATH on remote machines contains PBS_EXEC/bin. Remote machines must all have pbs_attach in the PATH.
- The PBS_RSHCOMMAND environment variable should not be set by the user.
- When using SuSE Linux, use "ssh -n" in place of "ssh".

10.13.5 Integrating HPE MPI

PBS supplies its own `mpiexec` on machines running supported versions of HPE MPI, in order to provide a standard interface for use by job submitters. This `mpiexec` calls the standard HPE `mpirun`. If users call this `mpiexec`, PBS will manage, track, and cleanly terminate multi-host MPI jobs.

If job submitters call HPE MPI directly, they must use `pbs_attach` in their job scripts in order to give PBS the same control over jobs; see the HPE documentation.

MPI jobs can be launched across multiple machines. PBS users can run an MPI job within a specific partition.

When job submitters use `mpiexec` in their job scripts, HPE MPI is transparent. Jobs run normally whether the PBS-supplied `mpiexec` is called inside or outside of PBS.

10.13.5.1 Supported Platforms

The PBS-supplied `mpiexec` runs on machines running supported versions of HPE MPI.

10.13.5.2 Steps to Integrate HPE MPI

Make sure that the PBS-supplied `mpiexec` is in each user's PATH.

10.13.5.3 Invoking HPE MPI

PBS uses the MPI-2 industry standard `mpiexec` interface to launch MPI jobs within PBS. If executed on a non-HPE system, PBS's `mpiexec` will assume it was invoked by mistake. In this case it will use the value of PATH (outside of PBS) or PBS_O_PATH (inside PBS) to search for the correct `mpiexec` and if one is found, exec it.

10.13.5.4 Using HPE MPI Over InfiniBand

To use InfiniBand, set the MPI_USE_IB environment variable to 1.

10.13.5.5 Using CSA with HPE MPI

PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

10.13.5.6 Prerequisites

- In order to run single-host or multi-host jobs, the HPE Array Services must be correctly configured. An Array Services daemon (`arrayd`) must run on each host that will run MPI processes. For a single-host environment, `arrayd` only needs to be installed and activated. However, for a multi-host environment where applications will run across hosts, the hosts must be properly configured to be an array.
- HPE systems communicating via HPE's Array Services must all use the same version of the `sgi-mpt` and `sgi-arraysvcs` packages. HPE systems communicating via HPE's Array Services must have been configured to interoperate with each other using the default array. See HPE's `array_services`(5) man page.
- "**`rpm -qi sgi-arraysvcs`**" should report the same value for *Version* on all systems.
- "**`rpm -qi sgi-mpt`**" should report the same value for *Version* on all systems.
- "**`chkconfig array`**" must return "*on*" for all systems
- `/usr/lib/array/arrayd.conf` must contain an array definition that includes all systems.
- `/usr/lib/array/arrayd.auth` must be configured to allow remote access:

  The "AUTHENTICATION NOREMOTE" directive must be commented out or removed

Either "AUTHENTICATION NONE" should be enabled or keys should be added to enable the SIMPLE authentication method.

- If any changes have been made to the `arrayd` configuration files (`arrayd.auth` or `arrayd.conf`), the array service must be restarted.
- `rsh(1)` must work between the systems.
- PBS uses HPE's `mpirun(1)` command to launch MPI jobs. HPE's `mpirun` must be in the standard location.
- The location of `pbs_attach(8B)` on each vnode of a multi-vnode MPI job must be the same as it is on the primary execution host vnode.

10.13.5.7 Environment Variables

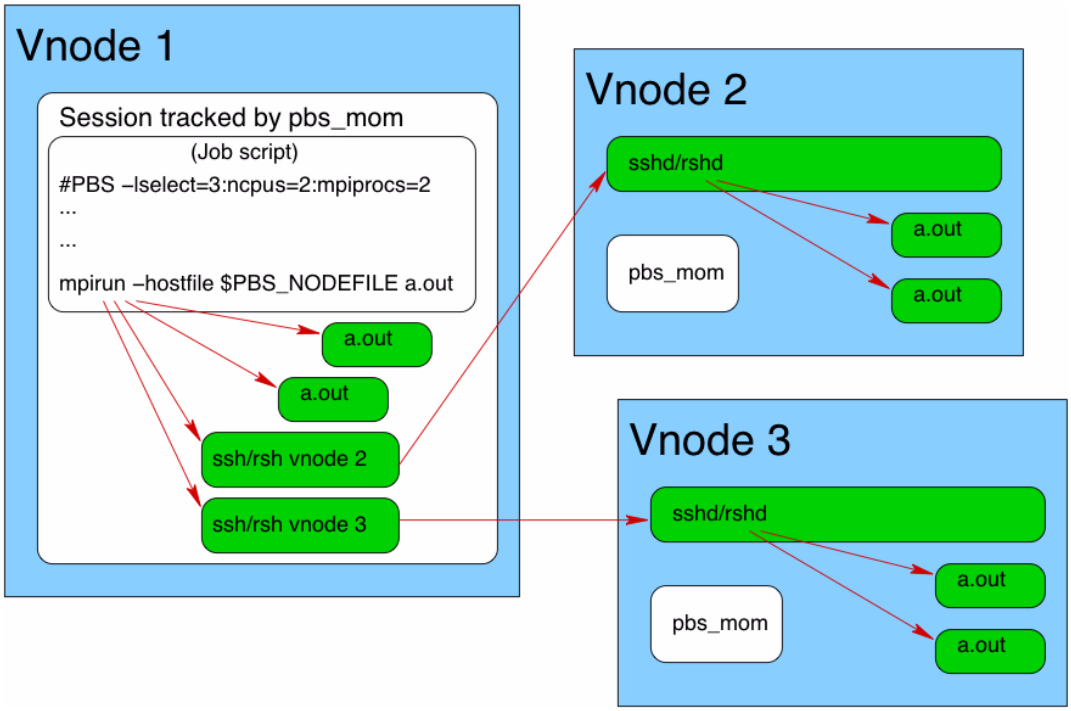
- If the PBS_MPI_DEBUG environment variable's value has a nonzero length, PBS will write debugging information to standard output.
- The PBS_ENVIRONMENT environment variable is used to determine whether `mpiexec` is being called from within a PBS job.
- If it was invoked by mistake, the PBS `mpiexec` uses the value of PBS_O_PATH to search for the correct `mpiexec`.
- To use InfiniBand, set the MPI_USE_IB environment variable to 1.

10.14 How Processes are Started Using MPI and PBS

10.14.1 Starting Processes under Non-integrated MPIs

The following figure illustrates how processes are started on sister vnodes when using a non-integrated MPI:

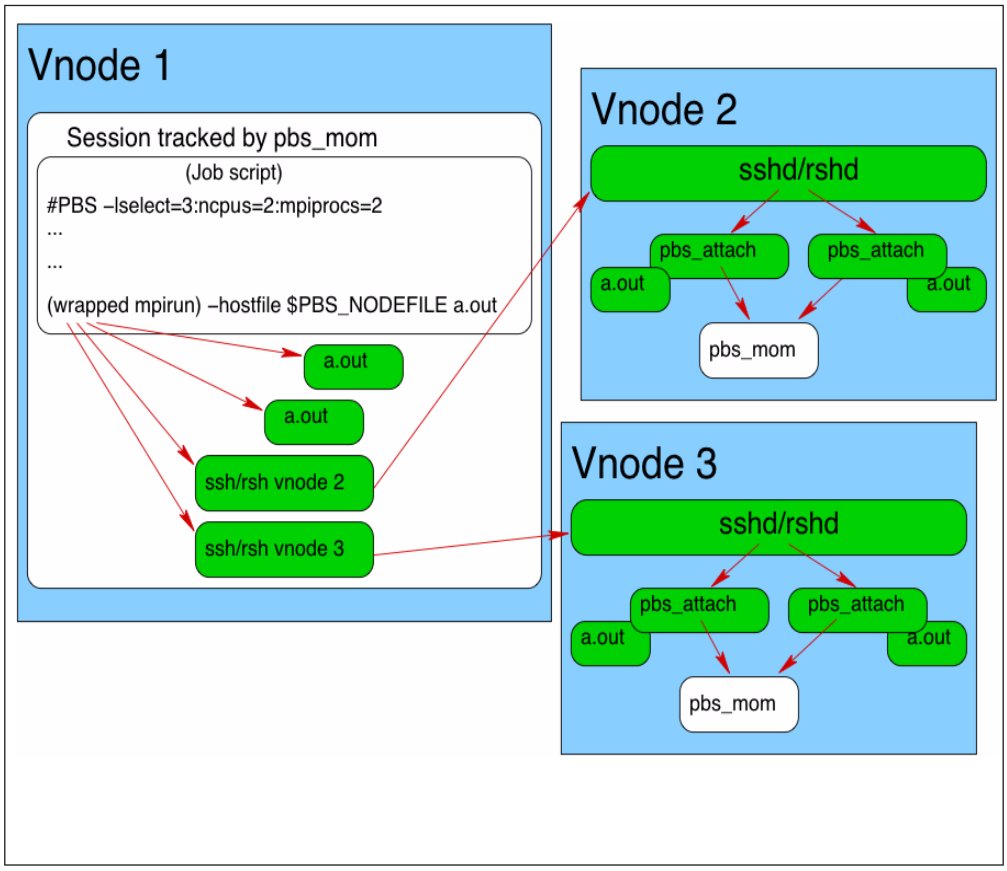
Figure 10-4:PBS does not know about the processes on vnodes 2 and 3, because those processes were generated outside of the scope of PBS.



10.14.2 Starting Processes under Wrapped MPIs

The following figure illustrates how processes are started on sister vnodes when using a wrapped MPI:

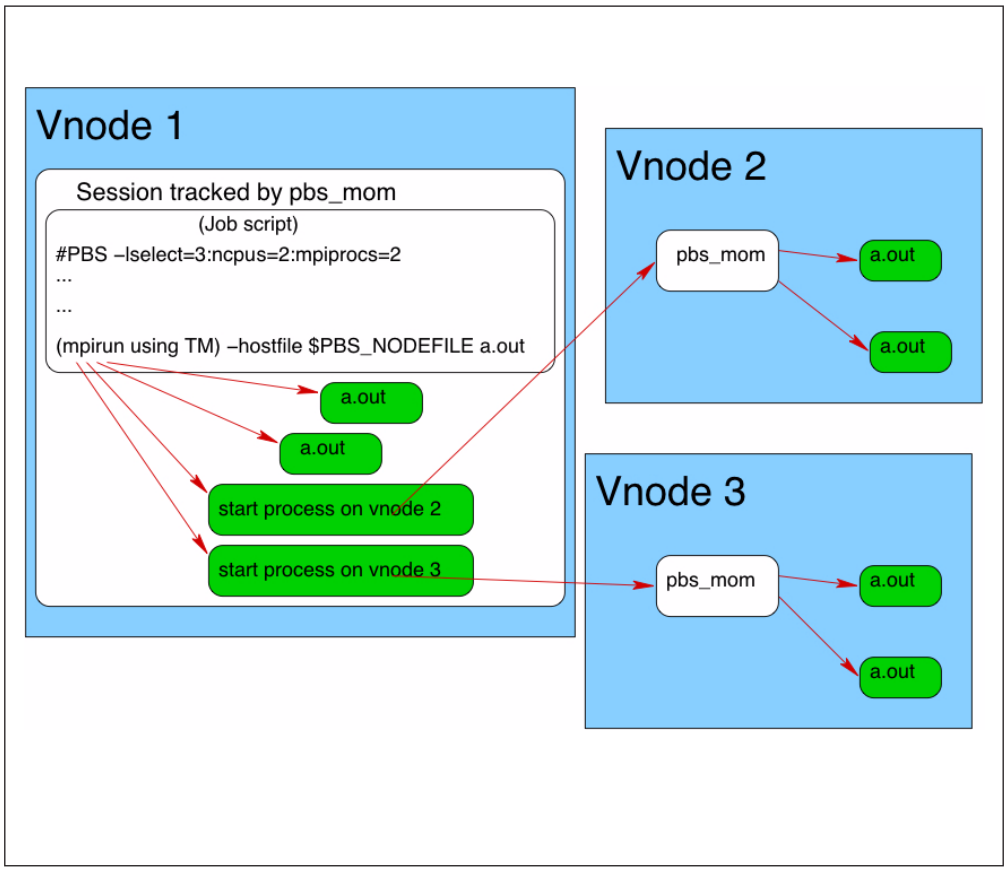
Figure 10-5: PBS knows about processes on vnodes 2 and 3, because `pbs_attach` tells those MoMs which processes belong to which jobs



10.14.3 Starting Processes Under MPIs Employing the TM Interface

The following figure illustrates how processes are started on sister vnodes when using an MPI that employs the TM interface:

Figure 10-6: PBS knows about processes on vnodes 2 and 3, because the TM interface talks directly to `pbs_mom`, and `pbs_mom` starts the processes on vnodes 2 and 3



10.15 Limit Enforcement with MPI

PBS can enforce the following for a job using MPI:

- Per-process limits via `setrlimit(2)` on sister vnodes
  - The `setrlimit` process limit can be enforced only when using an MPI that employs the TM interface directly, which is Open MPI only (and LAM MPI, which is deprecated)
- Limits set via MoM parameters, e.g. cpuburst and cpuaverage, on sister vnodes
  - PBS can enforce these limits using any integrated MPI
- Job-wide limits such as cput, mem
  - PBS can enforce job-wide limits using any integrated MPI

Once a process is started, process limits cannot be changed.

10.16 Restrictions and Caveats for MPI Integration

- Be sure to wrap the correct instance of the MPI. See section 10.10.1, “Wrap the Correct Instance”, on page 456
- Some applications write scratch files to a temporary location in `tmpdir`. The location of `tmpdir` is host-dependent. If you are using an MPI that is not integrated with the PBS TM interface, and your application needs scratch space, the location of `tmpdir` for the job should be consistent across execution hosts. You can specify the root for `tmpdir` in the MoM's $tmpdir configuration parameter. PBS sets the job's TMPDIR environment variable to the temporary directory it creates for the job.

11 Configuring PBS for Cray

11.1 Support for Shasta

PBS runs on Shasta exactly as it does on standard Linux machines. The only information in this chapter that applies to Shasta is contained within this section (Section 11.1, "Support for Shasta"). Each compute node behaves like a standard Linux machine, and runs one MoM. By default, each compute node is represented by one vnode. When you create vnodes on Shasta, use the host shortname as the vnode name.

11.1.1 Shasta Is Different from XC

If you are used to PBS on Cray XC machines, working with Shasta is different. Configuring PBS on Shasta is the same as on a standard Linux machine. You can ignore all of the Cray XC instructions in the rest of this chapter. For example:

- Batch mode and state do not apply
- Special Cray built-in resources do not apply to Shasta
- You don't need to set vntype
- You don't need node_fail_requeue to be zero
- The PBS_alps_inventory_check hook is not used for Shasta
- You don't need vnode_pool
- Hyperthreads are the same as on a standard Linux machine

11.1.1.1 Not Supported on Shasta

- Suspend/resume is not supported on Shasta
- Power awareness is not supported on Shasta

11.1.2 Hook for PBS on Shasta

On Shasta, PBS uses a built-in hook called *PBS_cray_atom*, which runs for execjob_begin and execjob_end events. The hook notifies the Cray when each job starts, and when each job should be deleted. This hook should be enabled by default, but we recommend making sure that it is.

If the hook alarms while running for the execjob_begin event (POST and DELETE), the vnode(s) where the hook was running are marked offline.

If the hook alarms while running for the execjob_end event (DELETE), the hook rejects the action. The default timeout for this hook is 300 seconds.

11.1.2.1 Shasta Hook Configuration File

The configuration file for the PBS_cray_atom hook is formatted as a JSON object. Here is the default configuration file:

```
{
    "post_timeout": 30,
    "delete_timeout": 30,
    "unix_socket_file": "/var/run/jacsd/jacsd.sock"
}
```

11.1.2.1.i Configuration File Parameters

"post_timeout"

Time limit for *POST* requests.

Units: *seconds*

Format: *float*

Default: *30 seconds*

"delete_timeout"

Time limit for *DELETE* requests.

Units: *seconds*

Format: *float*

Default: *30 seconds*

"unix_socket_file"

Path to the UNIX socket file to be used for authentication.

Format: *string*

Default: *"/var/run/jacsd/jacsd.sock"*

11.1.3 Responding to Node Health

On Shasta, Cray tasks take care of marking nodes unavailable or available. If Cray tasks decide that node health is not acceptable, Cray tasks will bring down the PBS MoM on that node. After you restore the node to usability, you must restart the MoM. If there are running jobs, use the `pbs_mom -p` option in order to preserve and track running jobs. See "Impact of Stop-Restart on Running Linux Jobs" on page 169 in the PBS Professional Installation & Upgrade Guide.

11.2 Configuring PBS for Cray XC Series

PBS provides features designed to support the Cray XC series. The rest of the chapter describes the special behavior of PBS on Cray XC systems only.

11.3 Introduction to PBS on Cray XC

PBS provides support for Cray XC systems by providing the following:

- PBS automatically defines vnodes for Cray XC compute nodes
- PBS automatically sets resources and attributes for vnodes representing Cray XC nodes
- PBS automatically creates custom resources that correspond to Cray XC resources
- Cray XC users can submit jobs through PBS using the PBS select and place statements.

11.4 Relationship of PBS Vnodes to Cray XC Nodes

PBS represents each login node as a vnode. A compute node is represented as a single vnode. A PBS MoM runs on each login node; this MoM manages the vnodes representing the compute nodes associated with the login node. On systems with multiple login nodes, each MoM on each login node manages every compute node. When this is the case, each compute node is reported by more than one login node. The Mom attribute of a vnode representing a compute node contains the hostname of each login node reporting the compute node. Each hostname is the FQDN or the short name of the reporting login node, depending on whatever is returned by the DNS.

11.4.1 How PBS Handles Changes in Cray XC Inventory

11.4.1.1 Reporting Changes in Vnode List in Cray XC

If a previously-reported vnode is no longer reported when the vnode list is created, because it is no longer available in the vnode definition file and the inventory, it is missing from the vnode list. The server marks missing vnodes as *stale*.

PBS provides a Boolean MoM configuration option that allows you to specify whether MoM tells the server that a vnode is missing. When the $vnodedef_additive MoM configuration option in `PBS_HOME/mom_priv/config` is *True*, MoM does not tell the server that any vnodes are missing. This means that the server does not mark missing vnodes as *stale*. When $vnodedef_additive is *False*, MoM tells the server that vnodes are missing, and the server marks the missing vnodes as *stale*. The default value for $vnodedef_additive for a MoM managing a Cray XC is *False*. The default value for other systems is *True*.

When a compute node goes down and the ALPS inventory no longer reports it, the vnode is marked *stale* once PBS queries for the inventory. If you bring the compute node back up, you can HUP the MoM in order to make the vnode usable by PBS. Alternatively, if PBS fails to make a Cray XC reservation, MoM will re-read the inventory and re-create the vnode list. A vnode is marked *stale* when any of the MoMs that reported the vnode stop reporting it.

The state of a vnode representing a compute node that is managed by more than one login node is not changed by a MoM going down, unless all of its MoMs are down. A vnode representing a compute node is marked *down* when all of the MoMs that manage the vnode are *down*.

11.4.1.2 When MoMs Report Conflicting Information in Cray XC

When more than one MoM reports information about a vnode, and the information conflicts, PBS uses the most recent information.

11.4.1.3 Vnode Resources on Stale Vnodes in Cray XC

If the ALPS inventory no longer reports information for a vnode, and the vnode is not defined in a vnode definition file, all resource and attribute information for that vnode is removed or set to zero, and the vnode is marked *stale*.

If the ALPS inventory no longer reports information for a vnode, but the vnode is defined in a vnode definition file, the vnode's attributes and resources retain their settings and the vnode is not marked *stale*.

11.4.1.4 Periodically Re-reading ALPS Inventory in Cray XC

PBS comes shipped with a built-in hook that periodically checks to see whether the inventory is consistent across ALPS and PBS. If it is not, it HUPs a MoM on a login node so that PBS has a current copy of the ALPS inventory. This hook is named *PBS_alps_inventory_check*. By default, this hook runs every 300 seconds; you can change the frequency by setting the hook's freq attribute:

```
#qmgr -c "set pbshook <hook name> freq=<new frequency>"
```

11.5 Requirements for Cray XC

- For a compute node to be managed by the PBS MoM, the node must be in batch mode, and in state *UP*.

11.6 Restrictions for Cray XC

- A Cray XC compute node cannot be used by more than one application at the same time.
- PBS does not report cput or mem for jobs running on a Cray XC compute node.

11.7 Resources, Parameters, etc. for Cray XC

PBS provides built-in and custom resources for the Cray XC. PBS also provides some built-in resources for all platforms, but these resources have specific uses on the Cray XC.

11.7.1 Resources for Cray XC

accelerator
accelerator_memory
accelerator_model
energy
eoe
naccelerators
PBScrayhost
PBScraylabel_<label name>
PBScraynid
PBScrayorder
vntype

See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide.

11.7.2 Scheduler Attributes for Cray XC

do_not_span_psets
only_explicit_psets

11.7.3 MoM Configuration Options for Cray XC

$alps_client
$alps_release_jitter
$alps_release_timeout <timeout>
$alps_release_wait_time
pbs_accounting_workload_mgmt <value>
$vnodedef_additive

11.8 Automatic Configuration for Cray XC

11.8.1 Vnode List Creation for Cray XC

You must create a vnode for each login node, but PBS automatically creates the vnodes representing compute nodes. PBS creates a list of vnodes by reading the Cray XC inventory and any vnode definition files. PBS automatically creates one vnode to represent each Cray XC compute node. All NUMA nodes in a compute node are represented by the same vnode. PBS does this automatically when any of the following happens:

- Startup of the MoM
- The MoM is HUPed
- PBS queries the Cray XC for the inventory (e.g., when PBS fails to confirm a Cray XC reservation)
- PBS times out on trying to release an ALPS request.

11.8.2 Automatic Vnode Attribute and Resource Settings for Cray XC

PBS automatically sets the values of certain vnode attributes and resources. Vnode attribute and resource settings are derived from values returned in the inventory, according to the following rules:

resources_available.accelerator
: Set to *True* when this vnode's host has at least one accelerator in state *UP*.

resources_available.accelerator_memory
: Set to the value of basil_accelerator_gpu.memory.

resources_available.accelerator_model
: Set to the value of basil_accelerator_gpu.family.

resources_available.naccelerators
: Number of Accelerator entries for the host that are in state *UP*.

resources_available.host
: Values of node's mpp_host and node_id are concatenated.

Format: <*mpp_host*>_<*node_id*>

Example: Given a compute node where mpp_host = *examplehost* and node_id = *8*, resources_available.host is set to *examplehost_8*.

resources_available.PBScrayhost

On CLE 3.0 and higher, set to value of mpp_host

resources_available.PBScraylabel_<label name>

For the label on a compute node, PBS creates a custom Boolean resource, and sets it to *True* on the vnode representing that compute node. The format for the name of this resource is *PBScraylabel_<label>*. For example, if the label is *Blue*, then the name of the Boolean resource is *PBScraylabel_Blue*.

resources_available.PBScraynid

The value of PBScraynid is set to the value of node_id for this compute node.

resources_available.PBScrayorder

Set to the position in the Cray XC node list of the associated node. If this vnode's associated node was *n*th in the node list, the value of PBScrayorder is *n*.

resources_available.PBScrayseg

Not used.

resources_available.vntype

On compute nodes, set to *cray_compute*

On internal login nodes, set to *cray_login*

Mom vnode attribute

This is the canonical hostname of the login node where MoM runs.

Name of vnode

Value of node's mpp_host and node_id are concatenated.

Format: <*mpp_host*>_<*node_id*>

Example: Given node_id = *8* and mpp_host = *examplehost*, the vnode name is *examplehost_8*.

sharing vnode attribute

Set to *force_exclhost*

11.8.3 Automatic MoM Parameter Settings for Cray XC

$alps_release_jitter <maximum jitter>

Cray XC only. PBS sends requests to ALPS to release a finished job at intervals specified by the sum of $alps_release_wait_time and a randomly generated value between zero and *maximum jitter*, in seconds.

Format: *Float*

Default: 0.12 seconds

$alps_release_timeout <timeout>

Specifies the amount of time that PBS tries to release an ALPS request before giving up. We recommend that the value for this parameter be greater than the value for suspectbegin.

Format: Seconds, specified as positive integer.

Default: *600* (10 minutes)

$alps_release_wait_time <wait time>

Cray XC only. PBS sends requests to ALPS to release a finished job at intervals specified by the sum of *wait time* and a randomly generated value between zero and the maximum specified in $alps_release_jitter, in seconds.

Format: *Float*

Default: 0.4 seconds

$vnodedef_additive MoM configuration option

PBS automatically sets the value of the $vnodedef_additive MoM configuration option to *False* on any MoM on a login node. See section 11.7.3, "MoM Configuration Options for Cray XC", on page 475.

11.8.4 Default Scheduler Attribute Settings for Cray XC

do_not_span_psets

This attribute is set to *False* by default. See section 11.7.2, "Scheduler Attributes for Cray XC", on page 475.

11.9 Recommended Manual Configuration for Cray XC

11.9.1 Configuring Vnode Names on Cray XC

You can use the PBS_MOM_NODE_NAME configuration parameter in `/etc/pbs.conf` to tell MoM what name was used to create the vnode at the server.

11.9.1.1 Requirements for PBS_MOM_NODE_NAME on Cray XC

You can use any resolvable host name for a vnode name when you create a vnode at the server, if you use the PBS_MOM_NODE_NAME configuration parameter to tell MoM about the name that was used to create the vnode at the server.

MoM accepts only a PBS_MOM_NODE_NAME that is known to the resolver of the "`hosts`" map; for example, it appears somewhere in `/etc/hosts` on the execution host, is mentioned in an NIS map, or is a name known to DNS servers, possibly after adding a domain to the search list. If the name cannot be resolved, MoM will refuse to start.

You cannot use dots in PBS_MOM_NODE_NAME.

11.9.1.2 When to Use PBS_MOM_NODE_NAME on Cray XC

You may need to use a name for your parent vnode that is different from the output of the `hostname` command, for reasons such as the following:

- You want vnodes named after their function in the complex, not an accidental hostname that may refer to physical placement of the node. Frequently when that hardware host is not booted, another node with another official hostname needs to assume those functions
- The hostname is maintained by a cluster manager but is a private name that may be unknown to the PBS server or other execution nodes, while the alias is actually a globally valid name
- Name resolution is inconsistent on the complex, i.e. the same name refers to different IP addresses on different hosts in the cluster, or the canonicalized name is different on different hosts

The name you specify in PBS_MOM_NODE_NAME can then be used consistently on the server, in Version 2 configuration files, and in hooks, regardless of the event type; pbs.get_local_nodename() in hooks will return this name, which always matches the name of the parent vnode of the execution host.

Note that it is possible, in some clusters, for PBS_LEAF_NAME and PBS_MOM_NODE_NAME to be associated with different interfaces and IP addresses on the execution host.

11.9.1.3 Avoid Problems with Vnode Naming on Cray XC

If you use a name different from the output of the hostname command to create a vnode, and you don't specify it in PBS_MOM_NODE_NAME, this can create problems when Version 2 configuration files or execution event hooks are used.

- Version 2 configuration files need to use the MoM parent vnode name to change attributes of the parent vnode, but the server knows the parent vnode by another name
- When an exechost_startup or exechost_periodic hook operates on the vnode_list[], the vnode_list[] will contain the vnode named after its local hostname and not the name used on the server to create the vnode, and the vnode_list[] (with its different naming) will be propagated to the server

Both these problems may render the original vnode created on the server stale (and replaced by a vnode created using MoM's view of the parent vnode name), which also means that resources set using `qmgr` on the server will be lost, since they will be set on the stale vnode and not on the vnode MoM created to replace it.

Hooks will face additional problems:

- Vnode naming in pbs.event().vnode_list[] will be inconsistent in different hook events, since a vnode_list[] created locally by MoM will use different naming from that for a vnode_list[] created through the job's exec_vnode and exec_host attributes (which are set by the scheduler and the server)
- In exechost_startup and exechost_periodic hooks, the vnode_list[] does not contain a full representation of the vnode on the server, since not all attributes and resources are propagated; it will be impossible to correctly fetch the missing resources by querying the server, since the vnode will be called differently in pbs.event().vnode_list[] and pbs.server().vnodes
- It will be very hard to determine which portion of a job's exec_vnode attribute corresponds to the local host in hooks, since pbs.get_local_nodename() will return the node's hostname and exec_vnode will use the vnode name used on the server

11.9.2 Set Scheduling Parameters for Cray XC

- If your server/scheduler runs on a non-CLE machine, add the vntype resource to the "`resources:`" line in `<sched_priv directory>/sched_config`. If your server/scheduler runs on a CLE machine, this happens automatically.
- We recommend that you add the naccelerators resource to the "`resources:`" line in `<sched_priv directory>/sched_config`.
- If you want the scheduler to honor the following resources, add them as well:

  PBScrayhost

  PBScraynid

  accelerator_memory

  accelerator_model
- Do **not** add the following resources to the "`resources:`" line:

  PBScrayorder

  nchunk

11.9.2.1 Caveats for Replacing Resources Used for Gating for Cray XC

- There is no support for resources_min.nchunk and resources_max.nchunk. If set, their behavior is undefined.
- If you wish to set resources_min.mpiprocs or resources_max.mpiprocs, you must make sure that mpiprocs can be counted for each job chunk. If the job did not request mpiprocs with each chunk, the job must inherit `mpiprocs = 1` for each chunk.

  Set default_chunk.mpiprocs to 1 on the server:

  ```
  Qmgr: s s default_chunk.mpiprocs = 1
  ```

11.9.3 Keeping Jobs Within One Host for Cray XC

To prevent jobs from being scheduled across multiple Cray XC hosts, you must limit jobs to a single value for PBScrayhost. Do the following:

- Set node_group_enable to *True*:

  ```
  Qmgr: s s node_group_enable=True
  ```
- Set node_group_key to *PBScrayhost*:

  ```
  Qmgr: s s node_group_key=PBScrayhost
  ```
- Set do_not_span_psets to *True*

  ```
  Qmgr: set sched <scheduler name> do_not_span_psets=true
  ```

If a job requests more resources than can be supplied from a single host, and the job does not specify a value for PBScrayhost, the job is scheduled across multiple Cray XC systems. To prevent this from happening, you can set the do_not_span_psets scheduler attribute to *True*, and add *PBSCrayhost* to node_group_key. See section 11.7.2, "Scheduler Attributes for Cray XC", on page 475.

11.9.4 Allowing Scheduling on Nearby Vnodes on Cray XC

To help the scheduler place each job requiring more than one vnode on vnodes that are close to each other, make the scheduler sort the vnodes based on their values for PBScrayorder. The vnodes will be listed in the order that their nodes are listed in the Cray XC inventory. To do this, specify the following in `<sched_priv directory>/config`:

```
node_sort_key: "PBScrayorder LOW"
```

11.9.5 Allowing Users to Request Useful Groups of Nodes on Cray XC

Job submitters can use select and place to request the groups of vnodes they want. However, you must provide the tools. Users may need to group their nodes by the certain criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

For these cases, you can do either of the following:

- Create custom resources, and set them on each vnode so that the important characteristics of the vnode can be requested. For example, if a vnode is fast, create a custom string resource called "*speed*" and set it to *fast* on that vnode.
- Label each node with its important characteristics. For example, if a node is both fast and best for App1, give it two labels, *fast*, and *BestForApp1*. PBS creates custom Boolean resources called *PBScraylabel_<label name>* and sets them to *True* on the appropriate vnodes.

11.9.6 Allowing Users to Request Login Node Groups on Cray XC

If users need to request groups of both esLogin nodes and internal login nodes, do the following:

1. Create a new string value for the vntype resource, for example *cray_compile*.
2. Use `qmgr` to set the value for vntype on the vnodes representing esLogin nodes:

   ```
   qmgr -c "set node esLogin resources_available.vntype+="cray_compile"
   ```

3. Use `qmgr` to add *cray_compile* to the vnodes representing internal login nodes. (resources_available.vntype is automatically set to *cray_login*.)

   ```
   qmgr -c "set node internal_Login resources_available.vntype +="cray_compile""
   ```

If you use `pbsnodes -av` to check resources_available.vntype for internal_Login, it now looks like this:

```
resources_available.vntype=cray_login,cray_compile
```

11.9.7 Set ALPS Reservation Release Timeout on Cray XC

The $alps_release_timeout <timeout> parameter specifies the amount of time that PBS tries to release an ALPS reservation before giving up. After this amount of time has passed, PBS stops trying to release the ALPS reservation, the job exits, and the job's resources are released. PBS sends a HUP to the MoM so that she rereads the ALPS inventory to get the current available ALPS resources.

You can set the amount of time that PBS waits between sending release requests to ALPS via the $alps_release_wait_time <wait time> and $alps_release_jitter <maximum jitter> MoM parameters. PBS generates a random value between zero and the value of $alps_release_jitter, and adds it to the value of $alps_release_wait_time, to determine how long to wait from one request to the next.

We recommend that you set the value for the $alps_release_timeout MoM parameter to a value greater than the value of the suspectbegin Cray XC node health variable. You want to allow the Cray XC node health check to reach its timeout before PBS gives up on trying to release the reservation.

The default for $alps_release_timeout is 600 seconds.

11.9.8 Enable Local Copy on Cray XC

If your site has disabled the use of remote operation functions ("r" commands) and output cannot be returned for jobs running on compute nodes, enable the use of the `cp` command by adding $usecp to the `$PBS_HOME/mom_priv/config` file on each login node.

11.9.9 Prevent Jobs from Being Requeued on Cray XC

We recommend setting the node_fail_requeue server attribute to *0* (zero) for Cray X* series machines. Why? There are three main cases where node_fail_requeue comes into play:

- Network failure between server and primary execution host MoM

  This is what node_fail_requeue *>0* helps with the most, since both the MoM and job processes are still running, so it acts as a buffer to allow the network failure to correct itself before action is taken on the job.

- Primary `pbs_mom` process crashes

  With node_fail_requeue set to a value *>0*, job processes on the Cray XC execution hosts continue to run and the ALPS reservation for the requeued job persists. This leads to PBS getting out of sync with ALPS (ALPS would not free the reservation that the requeued job was using), but PBS sees the resources as free since the job was requeued. The ALPS/PBS sync issue is handled properly by the remaining MoMs re-reading the ALPS inventory and seeing that these resources are unavailable. So we wind up wasting resources in this case. It is possible that the task completing on the Cray XC compute nodes from the original requeued job would still be of value to the job submitter.

- Primary `pbs_mom` host failure (e.g., someone tripped over the power cord)

  With node_fail_requeue set to a value *>0*, job processes continue to run on the Cray XC compute nodes under ALPS, though the running job script disappears upon job requeue. We run into the same case as when `pbs_mom` crashes.

In summary, with node_fail_requeue set to *0* on a Cray XC, the first case is still handled properly since jobs are not requeued due to a temporary network outage between the server and MoM on the login node. The second and third cases do not result in having the computation tasks running on two sets of Cray XC compute nodes (since PBS can't clean them up without the MoM that crashed). This is probably what you want.

For these jobs anything that the job was supposed to do after the `aprun` command in the job script does not occur, but the results from the number crunching would still be available.

11.10 Improving Server/MoM Inventory Performance for Cray XC

You can use the vnode attribute named *vnode_pool* to reduce the communication traffic between server and MoMs.

The vnode_pool attribute allows just one MoM, instead of all, to report inventory upon startup. This allows faster startup and less network communication between the server and the non-reporting MoMs. You use the attribute to tell PBS which MoMs report the same set of vnodes. PBS sees all login nodes with the same setting for vnode_pool as being part of the same Cray XC. Set this attribute to a different value for each Cray XC.

The type is int, and the Python type is int. The default value is zero, i.e. disabled. To enable the feature, set the attribute to any value greater than zero.

The vnode_pool vnode attribute can be set for each login node on a Cray XC running a MoM reporting the compute node inventory for the same Cray XC system. If you set this attribute for one of the MoMs reporting the same inventory, you must set it for all. Set the attribute to the same value on each of these nodes.

If one of the inventory-reporting MoMs goes down, PBS chooses another to report the inventory.

11.10.1 Setting the vnode_pool Attribute on Cray XC

- The vnode_pool attribute can be set only by a PBS Manager, and only in a vnode creation command. Format:

  Qmgr: create node <parent vnode name> vnode_pool=<value>

For example, if you have four login nodes running MoMs reporting the same compute node inventory, set vnode_pool at each of those vnodes to *1*.

- This attribute cannot be set or altered once the vnode is created; the attribute must be set during vnode creation.
- Do not define this attribute in a vnode definition file. To do so will give unspecified results.
- Do not attempt to set this attribute on a non-Cray machine. Setting the attribute on a non-Cray is unsupported and will result in undefined behavior.

11.10.2 Logging for Cray XC

11.10.2.1 MoM Log Messages Related to vnode_pool on Cray XC

- Recorded when a non-inventory-reporting MoM in a vnode pool is sent a Hello message by the Server:
  "Hello (no inventory required) from server at <server address>"
  Log level: PBSEVENT_SYSTEM

11.10.2.2 Server Log Messages Related to vnode_pool on Cray XC

- Recorded when a node is created or being recovered from the database and the specified vnode_pool value is negative or equal to zero:
  "invalid vnode_pool provided"
  Log level: PBSEVENT_ADMIN
- Recorded when an attempt is made to set or alter vnode_pool on an existing node:
  "Unsupported actions for vnode_pool"
  Log level: PBSEVENT_ADMIN
- Recorded when a MoM is first reporting her nodes to the server, and the vnode_pool appears in a vnode definition file:
  "Error <error code> setting attribute vnode_pool in update for vnode <name>"
  Log level: PBSEVENT_SYSTEM
- Recorded when a MoM sends either an IS_UPDATE or IS_UPDATE2 message to the Server and the sending: pbs_mom is in a vnode pool
  "POOL: IS_UPDATE<x> received"
  Log level: PBSEVENT_DEBUG4
- Recorded when an update of the database failed when updating the Mom attribute of a vnode:
  "write_single_node_mom_attr, Failed to update 'Mom' attribute"
  Log level: PBSEVENT_ERROR
- Recorded when the inventory-reporting MoM is set, including each time the inventory-reporting MoM changes:
  "Setting inventory_mom for vnode_pool <x> to <Mom name>"
  Log level: PBSEVENT_DEBUG
- Recorded when a MoM is added to a vnode pool:
  "Mom <Mom name> added to vnode_pool <x>"
  Log level: PBSEVENT_DEBUG3
- Recorded when a node is created or being recovered from the database:
  "vnode_pool value is <x>"

Log level: PBSEVENT_DEBUG3

- Recorded during MoM creation when a MoM is being added to a vnode pool:
  "POOL: cross linking <x> vnodes from <mom>"
  Log level: PBSEVENT_DEBUG4

11.11 Synchronizing PBS with ALPS Inventory on Cray XC

PBS comes shipped with a built-in periodic hook that regularly checks to see whether PBS is in sync with the ALPS inventory. If it is not, it HUPs the MoM on the first login node so that PBS has a current copy of the ALPS inventory.

By default, the hook is enabled on Cray X* series machines, and disabled on all other platforms. The hook runs as Administrator.

This hook is named *PBS_alps_inventory_check*.

By default, this hook runs every 300 seconds; you can change the frequency by setting the hook's freq attribute:

```
#qmgr -c "set pbshook PBS_alps_inventory_check freq=<new frequency>"
```

The timeout for the hook is 90 seconds.

The hook distinguishes multiple Cray XCs associated with a single PBS server by means of the PBScrayhost resource. Each MoM that runs the inventory hook (there will be as many of these MoMs as there are discrete Cray XC systems) will only attempt to inventory its own nodes.

The hook handles the case where there are nodes other than cray_login and cray_compute nodes on a server. It also allows a site to set the vntype resource on these nodes to some other string besides "*cray_compute*" or "*cray_login*". If the hook is running on such a node, it knows that it cannot process the inventory.

The hook ignores "*offline*" nodes for inventory purposes. If a cray_login node is "*offline*", it will be ineligible to be the MoM that runs the inventory hook. If a cray_compute node is "*offline*", then it may be either present or absent in the ALPS inventory without causing a SIGHUP and inventory reread. This allows a site to offline a compute node in PBS to prevent it from being scheduled without having to disable it in ALPS.

11.11.1 Prerequisites for Cray XC

The hook requires that the name of the cray_login parent vnode resolves to one of the addresses of the node it's running on.

11.12 Support for Xeon Phi on Cray XC

11.12.1 Creating Xeon Phi Vnodes on Cray XC

PBS Professional 2021.1 supports Xeon Phi nodes on the Cray XC running CLE 6.0. PBS automatically creates one vnode per Xeon Phi node, and assigns that Xeon Phi node to that vnode. PBS queries the Cray XC for information about the Xeon Phi nodes and automatically sets the following values for the vnode:

- The current_aoe attribute
- The hbmem resource
- The vntype resource to *cray_compute*

11.12.1.1 PBS Vnodes and Segments or NUMA Nodes on Cray XC

Regardless of the number of segments or NUMA nodes per Xeon Phi node, PBS creates one vnode per Xeon Phi node.

11.12.1.2 Indicating Current AOE on Cray XC

PBS indicates the current AOE for each vnode by setting the value of the current_aoe attribute to the concatenated returned values of numa_cfg and hbm_cache_pct. So for example if numa_cfg is *a2a* and hbm_cache_pct is *0*, the value of current_aoe is *a2a_0*. PBS queries BASIL for values for numa_cfg and hbm_cache_pct.

Valid values for numa_cfg: a2a, snc2, snc4, hemi, quad

Valid values for hbm_cache_pct: 0, 25, 50, 100

11.12.1.3 Indicating High-bandwidth Memory on Cray XC

This version of PBS automatically sets the value of resources_available.hbmem to the value of hbm_size_mb returned by BASIL.

11.12.1.4 Validating Xeon Phi Model in Request on Cray XC

You can use a queuejob hook to validate the Xeon Phi model a user requests for a job.

A Xeon Phi model has acceptable values with either of the following:

- No AOE is requested
- Valid values for 'numa_cfg' and 'hbm_cache_pct

11.12.1.4.i Example queuejob Hook for Validating Xeon Phi Models on Cray XC

```
import pbs

e = pbs.event()
j = e.job

try:
    knl_model = j.Resource_List['aoe']
    if knl_model is None:
        res_spec = j.Resource_List['select']
        if res_spec is not None:
            for s in str(res_spec).split('+')[0].split(':'):
                if s[:4] == 'aoe=':
                    knl_model = s.partition('=')[2]
                    break
    if knl_model is None:
        e.accept()
    else
        pbs.logmsg(pbs.LOG_DEBUG, "Job requested knl_model '%s'" % str(knl_model))
        numa_cfg = str(knl_model).split('_')[0]
        hbm_cache_pct = str(knl_model).split('_')[1]
        if num_cfg not in ['a2a', 'snc2', 'snc4', 'hemi', 'quad'] or hbm_cache_pct not in ['0',
'25', '50', '100']:
            e.reject("Invalid knl_model requested '%s'" % str(knl_model))
except SystemExit:
    pass
except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

11.12.2 Configuring Xeon Phi Vnodes on Cray XC

11.12.2.1 Configuration Instructions on Cray XC

You must set the value of resources_available.aoe for each Xeon Phi vnode. You can use the "admin helper" script in Chapter 11, "Using Xeon Phi Configuration Script on Cray XC", on page 485.

In order to use the Xeon Phi AOE, you need to set `capmc` permissions and make sure that capmc initialization succeeded. Contact Cray for this information.

We recommend setting the scheduler's provision_policy configuration parameter to *avoid_provision* because provisioning a new memory model is time-consuming.

11.12.2.2 Using Xeon Phi Configuration Script on Cray XC

We've included below an example of an "admin helper" configuration script called `pbs_config_knl_nodes.py` that sets resources_available.aoe on each Xeon Phi vnode.

We recommend that you do the following:

1. Edit the configuration script so that the server name is correct
2. Run the script at the server host

   The script produces a list of `qmgr` commands.
3. Feed the output of the script into `qmgr`

11.12.2.3 Xeon Phi Configuration Script Contents for Cray XC

The Xeon Phi configuration script is called pbs_config_knl_nodes.py. Here are the contents:

```
import json
import xml.etree.ElementTree as ET
import os
import sys
import subprocess
from string import join
import socket

# server_name = socket.gethostname()
server_name = 'perch'
capmc_dir = '/opt/cray/capmc/default/bin'
apbasil_dir = '/opt/cray/alps/6.1.3-43.3/bin'
basil_req = '<BasilRequest protocol="1.7" method="QUERY" type="SYSTEM"></BasilRequest>'
basil_req_file = '/tmp/pbs_system_query.xml'
cmd1 = ['capmc', 'get_numa_capabilities']
cmd2 = ['capmc', 'get_mcdram_capabilities']
cmd3 = ['apbasil']


# Generate list from numbers
def generate_number_list(number_str):
    tmp_list = number_str.split(',')
    num_list = []
    for item in tmp_list:
        tmp_val = None
        if '-' in item:
        try:
                lower,upper = item.split('-')
                tmp_val = range(int(lower),int(upper)+1)
        num_list += tmp_val
            except ValueError:
            print "Invalid string: ", item
        except:
            print "Something was not right with: ", item
        else:
            try:
                tmp_val = int(item)
        num_list.append(tmp_val)
            except NameError:
                print "Invalid integer: ", item
            except:
                print "Something was not right with: ", item
```

```
        # Return the list of numbers list
        return num_list


    # Add capmc_dir to path
    os.environ['PATH'] = capmc_dir + ":" + apbasil_dir + ":" + os.environ['PATH']

    # capmc commands to configure the knl nodes
    tmp_file = open(basil_req_file, 'w')
    tmp_file.write(basil_req)
    tmp_file.close()


    # identify the available numa settings
    numa_cap = {}
    process = subprocess.Popen(cmd1,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    output,err = process.communicate()
    knl_numa= json.loads(output)

    for item in knl_numa['nids']:
        numa_cap[item['nid']] = item['numa_cfg']


    # identify the available mcdram settings
    mcdram_cap = {}
    process = subprocess.Popen(cmd2,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    output,err = process.communicate()
    knl_mcdram = json.loads(output)
    for item in knl_mcdram['nids']:
        mcdram_cap[item['nid']] = item['mcdram_cfg']


    # identify the current node settings for mcdram and numa cfg
    tmp_file = open(basil_req_file)
    process = subprocess.Popen(cmd3,stdin=tmp_file,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    output,err = process.communicate()
    tmp_file.close()
    root = ET.fromstring(output)
    node_sets = {}
    set_cnt = 0
    for child in root.iter('Nodes'):
        nodes = generate_number_list(child.text.strip())
        node_sets[set_cnt] = {}
        node_sets[set_cnt]['attribs'] = child.attrib
        node_sets[set_cnt]['nids'] = nodes
        set_cnt += 1
```

```
# Loop through the node sets and identify which ones are knls
key_list = []
for key in node_sets:
    if 'attribs' in node_sets[key]:
        if node_sets[key]['attribs']['numa_cfg'] == "":
            key_list.append(key)

for key in key_list:
    node_sets.pop(key, None)

#print "-" * 80
for key in node_sets:
    #print key, node_sets[key]['nids']
    current_model = node_sets[key]['attribs']['numa_cfg']
    current_state = node_sets[key]['attribs']['hbm_cache_pct']
    current_aoe = current_model + "_" + current_state

    # Loop through the nodes that have Xeon Phis and setup them up for provisioning in PBS
    for nid in node_sets[key]['nids']:

        aoe_list = []
        models = ''
        states = ''
        if nid in numa_cap:
            models = numa_cap[nid].split(',')
        if nid in mcdram_cap:
            states = mcdram_cap[nid].split(',')
        for model in models:
            for state in states:
                try:
                    int(state)
                    aoe_list.append("%s_%s" % (model,state))
                except:
                    pass
        print "set node %s_%d_0 resources_available.aoe='%s'" % (server_name, nid,
    join(aoe_list,','))
        print "set node %s_%d_0 provision_enable = True" % (server_name, nid)
```

11.13 Using Hyperthreading on Cray XC

PBS supports hyperthreading on Cray X* series systems using ALPS. On a Cray X* series system using ALPS, PBS assigns values to vnode resources according to the inventory returned from ALPS. PBS sets resources_available.ncpus to the number of compute unit elements returned in the XML inventory. This allows PBS to make ALPS reservations for the compute units of a node, and get all of the hyperthreads associated with those compute units.

For a job to use hyperthreading, the job submitter puts a request for hyperthreading in the `aprun` call in the job script.

You can help job submitters use hyperthreaded vnodes by labeling them with a custom resource.

You can opt to limit the value reported for ncpus to cores only, not hyperthreads, when using cgroups. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

11.13.1 References for Hyperthreading on Cray XC

Check the *Cray Programming Environment User Guide* for Cray's advice on using CPUs and compute units.

11.14 Viewing Cray XC Information

11.14.1 Listing Vnodes on Cray XC

Each vnode appears only once in the output of any command that lists all vnodes, such as `pbs_statnode()`, `pbsnodes -av`, or `qmgr -c "list nodes @default"`.

11.14.2 Contents of Vnode Mom Attribute on Cray XC

When multiple login nodes are defined, each vnode representing a compute node lists every reporting login node in its Mom attribute. The Mom attribute lists multiple fully-qualified host names in a comma-separated list format.

For example, MoM1 reports compute nodes with node_id 1 through 4, and MoM2 reports compute nodes with node_id 3 through 6. In this case, the vnodes representing compute nodes with node_id 1 and 2 list MoM1 in the Mom attribute, vnodes representing compute nodes with node_id 3 and 4 list MoM1 and MoM2 in the Mom attribute, and vnodes representing compute nodes with node_id 5 and 6 list MoM2 in the Mom attribute.

11.14.3 Viewing Vnode Information on Cray XC

Each vnode's jobs attribute lists the jobs that have processes executing on that vnode. Jobs launched from an internal login node, requesting a vntype of *cray_compute* only, are not listed in the internal login node's vnode's jobs attribute. Jobs that are actually running on a login node, which requested a vntype of *cray_login*, do appear in the login node's vnode's jobs attribute.

You can view vnode attributes using the `pbsnodes -av` command.

If esLogin and internal login nodes are grouped by adding a string such as *cray_compile* to the vntype resource, the `pbs_rstat -F` command shows the following:

```
resources_available.vntype=cray_login,cray_compile
```

11.14.4 Effect on Jobs of Stopping and Starting Vnodes on Cray XC

If a job is launched from a login node, and the MoM goes down, the impact on the job depends on the command-line options specified when the MoM is restarted.

If a job is launched from a login node, and the login node goes down, the job does not continue to run.

If there are multiple login nodes, and one login node or its MoM goes down, jobs that were launched from other login nodes are not affected.

11.14.5 Resource Accounting on Cray XC

On CLE 5.2, Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray XC system. PBS performs resource accounting on the login nodes, under the control of their MoMs.

11.14.5.1 Using Comprehensive System Accounting on Cray XC

If CSA is enabled, PBS can request the kernel to write user job accounting data to accounting records. These records can then be used to produce reports for the user.

If PBS finds the CSA shared object libraries, and CSA is enabled, PBS can cause a workload management record to be written for each job. If MoM is configured for CSA support, MoM can issue CSA workload management record requests to the kernel. The kernel writes workload management accounting records associated with the PBS job to the system-wide process accounting file. The default for this file is `/var/csa/day/pacct`.

11.14.5.2 CSA Configuration Parameter on Cray XC

pbs_accounting_workload_mgmt <value>

MoM configuration parameter. Controls whether CSA accounting is enabled. The name does not start with a dollar sign. If set to "*1*", "*on*", or "*true*", CSA accounting is enabled. If set to "*0*", "*off*", or "*false*", CSA accounting is disabled. Values are case-insensitive. Default: "*true*"; enabled.

11.14.5.3 Requirements for CSA on Cray XC

PBS supports CSA on Cray XC machines running CLE 5.2. CSA requires CSA support Linux kernel modules.

On the supported platforms, the PBS MoM is CSA-enabled. If CSA workload management and user job accounting are available, PBS can use them.

11.14.5.4 Configuring MoM for CSA on Cray XC

CSA support is specified in the `pbs_accounting_workload_mgmt` line in MoM's Version 1 configuration file. CSA support is enabled by default; you must explicitly disable it if you want it disabled. If the `pbs_accounting_workload_mgmt` line is absent, CSA is still enabled.

To disable CSA support, modify `$PBS_HOME/mom_priv/config`, by setting `pbs_accounting_workload_mgmt` to *false*, *off*, or *0*.

To enable CSA support, either remove the `pbs_accounting_workload_mgmt` line, or set it to *true*, *on*, or *1*.

After modifying the MoM config file, either restart `pbs_mom` or send it SIGHUP.

11.14.5.5 Enabling Kernel CSA Support on Cray XC

In order for CSA user job accounting and workload management accounting requests to be acted on by the kernel, you need to make sure that the parameters CSA_START and WKMG_START in the `/etc/csa.conf` configuration file are set to "*on*" and that the system reflects this. You can check this by running the command:

```
csaswitch -c status
```

To set CSA_START to *"on"*, use the command:

```
csaswitch -c on -n csa
```

To set WKMG_START to *"on"*, use:

```
csaswitch -c on -n wkmg
```

Alternatively, you can use the CSA startup script `/etc/init.d/csa` with the desired argument (*on*/*off*); see the system's man page for `csaswitch` and how it is used in the `/etc/init.d/csa` startup script.

11.15 Resource Restrictions and Deprecations for Cray XC

The mpp* resources are removed.

The following are not supported, and if set, behavior is undefined.

resources_min.nchunk

resources_max.nchunk.

11.16 Caveats and Advice for Cray XC

11.16.1 Processes Not Suspended on Cray XC

On CLE 5.2UP03, if you are using preemption via suspend/resume, you may see that after a SWITCH request, ALPS continues to start new applications, so that the system is never ready for PBS to suspend job processes. The system can go from RUNNING to EMPTY to RUNNING.

11.16.2 Creating MoM Directories on Cray XC

If you start PBS with PBS_START_MOM = 0 for the first time, PBS does not create MoM directories on that host. If you want to use the machine as an execution host (running a MoM on it), set *PBS_START_MOM* = 1, run `pbs_habitat`, then start MoM.

11.16.3 Error Messages in MoM Logs on Cray XC

There are two circumstances where you may see error messages in the MoM logs:

1. The first circumstance is benign. PBS is trying to release a reservation, and PBS gets acknowledgement when ALPS stops recognizing the reservation. ALPS has canceled the reservation that PBS was trying to cancel. We get the following message:

   "ALPS error: apsched: No entry for resId <reservation ID>"

2. In the second circumstance, a node is busy or unavailable, resulting in these messages:

   "Node;BASIL;ERROR: ALPS error: apsched: resource temporarily unavailable"

   "Node;alps_request_parent;TRANSIENT BASIL error from BACKEND: ERROR: ALPS error: apsched: resource temporarily unavailable"

   "Transient MPP reservation error on create."

   When PBS is trying to make a reservation on a node that has become unavailable, this is benign. MoM will update her vnode information and be able to run jobs.

   Very rarely, these messages can be seen when there is a problem that requires action, for example, when Node Health Checker takes longer than the value of $alps_release_timeout. In this case, set the value of $alps_release_timeout to be greater than the time Node Health Checker requires. Unless you are seeing held jobs, this is probably not what is happening.

11.16.4 Configure Cray XC MoMs According to Rules

Because the vnode sharing attribute must be set using the `pbs_mom -s insert` command, it is not recommended to set the sharing attribute on a Cray XC vnode.

11.16.5 Suspending and Resuming Jobs on Cray XC

On Cray X* series systems, PBS can suspend one or more jobs in order to run a higher priority job.

11.16.5.1 Caveats and Restrictions

- Suspend and resume are supported on Cray XC systems with an Aries interconnect and newer Cray X* series systems
- Suspend and resume are not supported on Cray XC systems with a Gemini interconnect
- On Cray X* series systems, the suspended low priority job and the high priority job must fit into the Cray compute node's memory
- Cray X* series systems can have a maximum number of co-resident jobs on a compute node. See the Cray documentation for more details.
- On a Cray X* series system, a job that requests exclusive access (i.e. `-lplace=excl`) to a node cannot be suspended.

11.16.5.2 Default Behavior on Cray XC

The restrict_res_to_release_on_suspend server attribute is set to *ncpus* by default on Cray X* series systems. This attribute is set in the `pbs_habitat` script which runs when PBS is started for the first time after an install or upgrade.

During suspension of a job PBS releases only ncpus on Cray XC machines.

11.16.5.3 Configuring Suspend and Resume on Cray XC

On Cray X* series systems, PBS issues a request to ALPS to switch an ALPS reservation IN for resume or OUT for suspend.

In order to use suspend and resume on a Cray X* series, you must modify ALPS configuration files. Please refer to Cray's *System Administration Guide* for more details about using suspend and resume on Cray X* series.

You can update the restrict_res_to_release_on_suspend server attribute and add more resources to it. We do not recommend removing ncpus from the list of resource names on Cray X* series systems. See section 5.9.6.2, "Job Suspension and Resource Usage", on page 250.

11.16.5.4 Errors on Cray XC

If ALPS fails to switch a reservation from suspend to resume or resume to suspend, PBS returns the error code 15222 via the `pbs_sigjob()` IFL call. The `qsig` command prints the following error message when ALPS fails to switch a reservation:

```
"qsig: Switching ALPS reservation failed <job id>"
```

11.16.6 Vnode Definition Files Not Recommended on Cray XC

Using vnode definition files on a Cray XC is not recommended. Use `qmgr` where possible instead.

Any attribute and resource settings for a specific vnode in a vnode definition file cause PBS to believe that that vnode is still usable. If PBS reads the Cray XC inventory, and a vnode is not listed in the inventory, but it is listed in a vnode definition file, the vnode is not marked as *stale*. This will cause a problem when the scheduler tries to schedule jobs onto this vnode.

If you create a vnode definition file for a vnode that has more than one MoM, you must make sure that the files are consistent on all of the MoMs that manage the vnode.

Settings in a vnode definition file override those from inventory. Settings in `qmgr` override both vnode definition files and inventory. Be careful not to overwrite information from the inventory when creating a vnode definition file. For example, if the vntype resource for a vnode is set to *cray_login* when PBS reads the inventory, and it is set to *cray_compile* in a vnode definition file, the value for vntype becomes *cray_compile* only.

11.16.7 Use Correct Name When Creating Vnode on Cray XC

When creating a vnode to represent a login node, use the short name returned by the `gethostname` command on the login node. For example, if `gethostname` returns *HostA*, do the following:

```
Qmgr: create node HostA
```

If you create a vnode with a different name from the short name returned by gethostname, the following happens:

- MoM creates a vnode whose name is the short name returned by `gethostname`
- The vnode you created is not recognized by MoM, and is marked *stale*

Do not use the IP address for the vnode name.

11.16.8 Deleting Vnodes on Cray XC

You can delete a parent vnode only if no other vnodes list this vnode in their Mom attributes. In order to delete a parent vnode which is listed in another vnode's Mom attribute, you must first delete the vnode with this vnode in its Mom attribute.

After removing a vnode that is managed by more than one MoM, you must HUP all of the managing MoMs, otherwise the vnode is not marked *stale* by the server.

11.16.9 Do Not Make Vnode Definitions Additive on Cray XC

On a Cray XC MoM, the $vnodedef_additive parameter in `PBS_HOME/mom_priv/config` is set to *False* or 0 by default. Do not unset or change the setting of the $vnodedef_additive parameter.

11.16.10 Do Not Use `configrm` on Cray XC

It is not recommended to use the `configrm` `pbs_tclsh` call.

11.16.11 Using Gating Values As Defaults on Cray XC

For most resources, if the job does not request the resource, and no server or queue defaults are set, the job inherits the maximum gating value for the resource. If this is set at the queue, the queue value of resources_max.<resource name> is used. If this is set only at the server, the job inherits the value set at the server.

11.16.12 Marking Cray XC Vnodes Offline

You can use the `qmgr` command to individually mark each vnode representing a compute node offline. This is independent of the vnodes representing login nodes.

11.16.13 Do Not Use PBS-reserved Resource Names on Cray XC

Do not create resources with names that could be used by PBS to create a Cray XC resource equivalent. For example, do not create a resource with the name *PBScraylabel_small*.

11.16.14 Fewer Chunks for Shorter Scheduling Cycle on Cray XC

The more chunks in each translated job request, the longer the scheduling cycle takes. Jobs that request a value for ncpus effectively direct PBS to use the size of ncpus as the value for ncpus for each chunk, thus dividing the number of chunks by ncpus.

Example 11-1: Comparison of larger vs. smaller chunk size and the effect on scheduling time:

Submit job with chunk size 1 and 8544 chunks:

```
qsub -lselect=8544:ncpus=1 job
```

Job's Resource_List:

```
Resource_List.ncpus = 8544
Resource_List.place = free
Resource_List.select = 8544:vntype=cray_compute
Submit_arguments = -lselect=8544:ncpus=1 job
```

Scheduling took 6 seconds:

```
12/05/2011 16:46:10;0080;pbs_sched;Job;23.example;Considering job to run
12/05/2011 16:46:16;0040;pbs_sched;Job;23.example;Job run
```

To speed up scheduling, you may want to write a submission hook that assigns each job a value for ncpus. We recommend that this value be the value for ncpus for a vnode or for a compute node.

11.16.14.1 Caveats

If you are on a heterogeneous system, forcing ncpus to be set for all requests can cause jobs to wait for particular vnodes and their associated resources, where these jobs would have been able to run across many different-sized vnodes.

Instead, you can use a job submission hook that rejects jobs that don't request values for ncpus.

11.16.15 No cput or mem for Compute Node Jobs on Cray XC

PBS does not report cput or mem for jobs running on a Cray XC compute node.

11.16.16 Set PATH Correctly on Cray XC

PATH must be included in the pbs_environment file. The PATH value is passed on to batch jobs. To maintain security, it is important that PATH be restricted to known, safe directories. Do not include "." in PATH.

11.17 Errors and Logging on Cray XC

11.17.1 Creating Custom Resources on Cray XC

When a custom resource is created for a Cray XC vnode, the server logs a message containing the resource name and type, and the vnode name. This is logged at event class 0x080.

If a custom resource can't be created, the following error message is printed in the server log:

```
error: resource <name> for vnode <name> cannot be defined
```

11.17.2 Job Requests More Than Available on Cray XC

If do_not_span_psets is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  ```
  "Not Running: can't fit in the largest placement set, and can't span placement sets"
  ```
- The following message is printed to the scheduler's log:
  ```
  "Can't fit in the largest placement set, and can't span placement sets"
  ```

11.17.3 Invalid Cray XC Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray XC. The Cray XC width and depth values cannot be calculated from ncpus and mpiprocs values. For example, if ncpus is 2 and mpiprocs is 4, the depth value is calculated by dividing ncpus by mpiprocs, and is one-half. This is not a valid depth value for Cray XC. When a select statement does not meet Cray XC requirements, and the Cray XC reservation fails, the following error message is printed in MoM's log, at log level 0x080:

```
Fatal MPP reservation error preparing request
```

11.17.4 Unequal ompthreads and ncpus on Cray XC

If the value of ompthreads does not match the value of ncpus when PBS is constructing exec_vnode for a job, the following is printed in the MoM log, at event class 0x080:

```
"ompthreads <value> does not match ncpus <value>"
```

12 Support for HPE

12.1 Briefly, How PBS Manages Cpusets

As of version 2020.1, PBS uses the standard MoM on HPE machines, and uses the cgroups hook to manage cpusets on HPE machines. See Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

PBS automatically examines the topology of the machine, and creates child vnodes to represent subsets of the machine. PBS also organizes the machine's vnodes into placement sets. When PBS runs a job on an HPE execution host, the cgroups hook creates the cpuset in which the job runs, and destroys the cpuset after the job is finished.

12.2 Cpusets and Vnodes

The PBS MoM represents a machine as a set of vnodes. Each vnode is visible via commands such as `pbsnodes`. Each vnode must have its own logical memory pool, so you get one vnode per logical memory pool. All of the vnodes on one multi-vnode host are managed by one instance of `pbs_mom`.

A cpuset is a group of CPUs and memory nodes around which an inescapable wall has been placed. The OS manages a cpuset so that processes executing within the cpuset are typically confined to use only the resources defined by the cpuset.

12.3 Requirements for Managing Cpusets

If you want PBS to manage the cpusets on a machine:

- Use the cgroups hook
- You must use a supported version of HPE MPI
- You use the PBS start/stop script to start MoM instead of `pbs_mom`

12.4 Where to Use Cpusets

Use PBS to manage your cpusets wherever you want jobs to be fenced into their own CPUs and memory. This can also be useful on other machines, such as the HPE 8600, depending on the individual machine.

12.5 Settings for sharing Attribute

The cgroups hook sets the sharing attribute for each vnode as follows:

- On MC990X and Superdome Flex, the hook sets the sharing attribute for the parent vnode to *default_shared*
- On MC990X and Superdome Flex, the hook sets the sharing attribute for all other vnodes to *default_shared*
- On 8600, the hook sets the sharing attribute for each vnode to *default_shared*

12.5.1 Creating Vnodes

We recommend using the cgroups hook to manage the machine and create any child vnodes. If you use the cgroups hook, do not create vnodes via a Version 2 configuration file. However, if you are not using the cgroups hook, you can create your vnode definitions by hand. You can have MoM create any child vnodes via a Version 2 configuration file. See section 3.3, “Creating Vnodes”, on page 38.

12.5.1.1 Caveats for Creating Vnodes

Do not attempt to create more than one vnode per logical memory pool. Your jobs will not run correctly.

12.5.2 Configuring Vnodes

If necessary, you can modify child vnodes created by the hook, by using an exechost_startup hook or via a Version 2 configuration file. See section 3.4, “Configuring Vnodes”, on page 41.

12.6 Comprehensive System Accounting

PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

13 Support for NEC SX-Aurora TSUBASA

13.1 Vnodes for NEC SX-Aurora TSUBASA

The basic hardware unit for NEC SX-Aurora TSUBASA is a *vector host* (a standard x86 server) connected to a set of accelerators called *vector engines* (VEs) via optional PCIe. The unit can consist of one or more NUMA nodes. Each unit uses one or more host channel adapters to communicate with other units and with the rest of the world.

The increasing order of communication overhead is first within a vector engine, then between vector engines via a shared PCIe, then between vector engines via PCIes on a common vector host, and finally between vector engines on separate vector hosts.

PBS creates topology-aware vnodes by grouping each PCIe with its associated vector host and vector engines together into one vnode. If there is no PCIe, PBS groups the vector host and its VEs into a vnode. A NUMA node without its own PCIe is in its own vnode.

PBS automatically creates vnodes to represent the host topology when you start PBS on execution hosts after the built-in PBS_sx_aurora hook is enabled.

PBS tries to do topology-aware scheduling by grouping job processes on vector engines in a way that produces the lowest communication overhead. When a job requests vector engines, PBS tries to assign vector engines from a single vnode to minimize communication overhead between vector engines.

For how to request resources on NEC SX-Aurora TSUBASA, see "Submitting Jobs to NEC SX-Aurora TSUBASA", on page 201 of the PBS Professional User's Guide.

13.2 Terminology

HCA

Host channel adapter. Network interconnect used by vnode. Each vector host can have one or more HCAs.

Vector engine, VE

Accelerator associated with vector host. Executes parallel and/or vectorized numeric operations.

Vector host, VH

Standard x86 server. Performs tasks such as I/O.

VE offloading

Main operations that take place on the vector host offload parallel and/or vectorized numeric operations to vector engines. In offloading, NEC MPI launches processes on the VH, and those processes then launch other job processes on VEs assigned to the job. See "Using VE Offloading", on page 206 of the PBS Professional User's Guide.

13.3 Resources for SX-Aurora TSUBASA

nves

Host-level consumable integer. Allows you to specify the number of vector engines per chunk. PBS sets the available VEs on a vnode in resources_available.nves. The default for resources_available.nves is number of VEs attached to the PCIe. The out-of-the-box default value for a job request is zero; PBS assigns a value of zero unless the administrator has set the value otherwise.

nhcas

Chunk-level non-consumable integer. When requested in a job chunk, PBS sets _NEC_HCA_LIST_IO and _NEC_HCA_LIST_MPI environment variables accordingly for that chunk. When not requested for a chunk, PBS sets _NEC_HCA_LIST_IO and _NEC_HCA_LIST_MPI to include all HCAs on a host.

ve_mem

Job-wide string. Used for reporting the maximum memory on vector engines used by job.

ve_cput

Job-wide string. Used for reporting the total CPU time, in seconds, on vector engines used by job.

ncpus

PBS sets the value of resources_available.ncpus on each vnode to (#CPUs on whole host / #vnodes on host) - #VEs on vnode. One CPU per VE is reserved for the VEOS daemon.

mem

PBS sets the value of resources_available.mem on each vnode by dividing the memory of the whole host equally among the vnodes on the host.

13.4 Configuring PBS for NEC SX-Aurora TSUBASA

1. Make sure that the PBS server and MoM daemons are installed and started. See "Installing via RPM on Linux Systems" on page 23 in the PBS Professional Installation & Upgrade Guide.
2. Enable the PBS_sx_aurora hook:

   ```
   sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora enabled=True"
   ```
3. Optional: the default frequency for the hook is 20 seconds. You can set the frequency for the PBS_sx_aurora hook in seconds:

   ```
   sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora freq=<frequency>"
   ```
4. Create the custom resource nves for requesting and managing VEs:

   ```
   sudo /opt/pbs/bin/qmgr -c "c r nves type= long,flag=nhm"
   ```
5. Add the custom nves resource to the {PBS_HOME}/sched_priv/sched_config resources: line:

   ```
   resources: "ncpus, mem, arch, host, vnode, aoe, eoe, nves"
   ```
6. HUP the scheduler(s):

   ```
   kill -HUP <scheduler PID>
   ```
7. Create the custom resource nhcas for managing HCAs:

   ```
   sudo /opt/pbs/bin/qmgr -c "c r nhcas type= long,flag=hm"
   ```
8. Create the custom resources ve_cput and ve_mem for accounting:

   ```
   sudo /opt/pbs/bin/qmgr -c "c r ve_cput type=long, flag=h"
   sudo /opt/pbs/bin/qmgr -c "c r ve_mem type=size, flag=h"
   ```
9. Recommended: sort your vnodes first by nves, then by ncpus. Replace the default node_sort_key in {PBS_HOME}/sched_priv/sched_config with the following:

   ```
   node_sort_key: "nves HIGH unused"
   node_sort_key: "ncpus HIGH unused"
   ```
10. Set NEC-specific environment variables for all hosts. Set the following in each host's {PBS_HOME}/pbs_environment file:

    ```
    NMPI_LAUNCHER_EXEC={PBS_EXEC}/bin/pbs_tmrsh
    NMPI_TTY_COMPAT=ON
    ```
11. Restart PBS so that the PBS_sx_aurora hook can create vnodes on all hosts in the cluster:

    ```
    sudo systemctl restart pbs
    ```

13.5 Debugging on NEC SX-Aurora TSUBASA

- To see all the DEBUG and INFO messages logged by the hook, increase the log level by setting the $logevent parameter to *4095* (0xffffffff) in the MoM's configuration file, PBS_HOME/mom_priv/config.
- If you need to detect failed devices faster, you can change the hook's frequency. The frequency is specified in seconds. The default hook frequency is 20 seconds. You can set the frequency for the PBS_sx_aurora hook:

  ```
  sudo /opt/pbs/bin/qmgr -c "set pbshook PBS_sx_aurora freq=<frequency>"
  ```

13.6 Suspending and Resuming Jobs

On the SX-Aurora TSUBASA, partial process swapping (PPS) means copying part of the memory on VEs being used by VE processes onto memory on the VH (the PPS buffer), then later copying the PPS buffer on the VH back to memory on the VEs.

PPS allows high-priority jobs to run before current lower-priority VE processes finish, by suspending and swapping out those lower-priority processes, then resuming the swapped-out VE processes after the high-priority jobs finish.

Swapping out a VE process means copying part of the memory area being used by the VE process onto the PPS buffer, and freeing the memory area so that other, higher-priority, VE processes can use it.

Swapping in a VE process means copying the memory area in the PPS buffer used for the swapped-out VE process back to memory on the VE.

When PBS runs a higher-priority job by swapping out a lower-priority job, it does the following:

1. Suspend the execution of VE processes and VH processes belonging to the lower priority job
2. Swap out the VE processes of the lower-priority job, using SX-Aurora TSUBASA interfaces
3. Run the higher-priority job
4. After completion of the higher-priority job, swap the VE processes of the lower-priority job back
5. Resume the execution of the lower-priority job

13.7 Job Accounting on NEC SX-Aurora TSUBASA

When PBS writes accounting records, PBS records nves in both Resource_List.nves and resources_assigned.nves. PBS also writes ve_mem and ve_cput as part of the value of the job's resources_used attribute.

14 Managing Jobs

14.1 Routing Jobs

You can route jobs to various places and by various criteria. You can reject submission of jobs that request too much of a given resource. You can force jobs into the correct queues. You can have all jobs submitted to a routing queue, then route them to the correct execution queues. You can use peer scheduling to have jobs executed at other PBS complexes. You can use hooks to move jobs. For information on routing jobs, see section 4.9.39, “Routing Jobs”, on page 205.

14.2 Limiting Number of Jobs Considered in Scheduling Cycle

If you limit the number of jobs in execution queues, you can speed up the scheduling cycle. You can set an individual limit on the number of jobs in each queue, or a limit at the server, and you can apply these limits to generic and individual users, groups, and projects, and to overall usage. You specify this limit by setting the queued_jobs_threshold queue or server attribute. See section 5.15.1.9, “How to Set Limits at Server and Queues”, on page 296.

If you set a limit on the number of jobs that can be queued in execution queues, we recommend that you have users submit jobs to a routing queue only, and route jobs to the execution queue as space becomes available. See section 4.9.39, “Routing Jobs”, on page 205.

14.3 Allocating Resources to Jobs

You can make sure that jobs request or inherit any resources required to manage those jobs. If a job does not request a resource, you can make sure that the resource is allocated to the job anyway.

In order for limits to be effective, each job must request each limited resource. For a complete description of how limits work, see section 5.15, “Managing Resource Usage”, on page 287.

You can create custom resources specifically to allocate them to jobs. These resources can be visible, alterable, and requestable by users, or invisible, unalterable, and unrequestable, or visible but unalterable and unrequestable. For instructions on creating invisible or unrequestable resources, see section 5.14.2.3.vi, “Resource Permission Flags”, on page 260.

You can alter a job's resource request using the following methods:

- You can set defaults for resources at the server or at each queue. This way, you can have jobs inherit specific values for the resources by routing them to special queues, where they inherit the defaults. For how jobs inherit resources, see section 5.9.4, “Allocating Default Resources to Jobs”, on page 247. For how to specify default resources, see section 5.9.3, “Specifying Job Default Resources”, on page 245.

For how resource defaults change when a job is moved, see section 5.9.4.3, "Moving Jobs Between Queues or Servers Changes Defaults", on page 248.

- You can use a hook to assign a specific resource value to a job, if a job requests the wrong value for a resource. For how to use a hook to assign a resource to a job, see the PBS Professional Hooks Guide. For examples of using hooks to assign resources to jobs, see *PBS Professional Plugins (Hooks) Guide*.
- You can use the `qalter` command to change a job's resource request. For how to use the qalter command, see "qalter" on page 127 of the PBS Professional Reference Guide.
- You can set default arguments the `qsub` command via the default_qsub_arguments server attribute. For how to use default arguments to `qsub`, see "Server Attributes" on page 281 of the PBS Professional Reference Guide.

14.3.1 Viewing Resources Allocated to a Job

14.3.1.1 The exec_vnode Attribute

The exec_vnode attribute displayed via `qstat` shows the resources allocated from each vnode for the job.

The exec_vnode line looks like:

```
exec_vnode = (<vnode name>:ncpus=W:mem=X)+(<vnode name>:ncpus=Y:mem=Z)
```

For example, a job requesting

```
-l select=2:ncpus=1:mem=1gb+1:ncpus=4:mem=2gb
```

gets an exec_vnode of

```
exec_vnode =  (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb) +(VNC:ncpus=4:mem=2gb)
```

Note that the vnodes and resources required to satisfy a chunk are grouped by parentheses. In the example above, if two vnodes on a single host were required to satisfy the last chunk, the exec_vnode might be:

```
exec_vnode = (VNA:ncpus=1:mem=1gb)+(VNB:ncpus=1:mem=1gb)
    +(VNC1:ncpus=2:mem=1gb+VNC2:ncpus=2:mem=1gb)
```

Note also that if a vnode is allocated to a job because the job requests an arrangement of *exclhost*, only the vnode name appears in the chunk. For example, if a job requesting

```
-l select 2:ncpus=4 -l place = exclhost
```

is placed on a host with 4 vnodes, each with 4 CPUs, the exec_vnode attribute looks like this:

```
exec_vnode = (VN0:ncpus=4)+(VN1:ncpus=4)+(VN2)+(VN3)
```

14.3.1.2 The schedselect Attribute

The resources allocated from a vnode are only those specified in the job's schedselect attribute. This job attribute is created internally by starting with the select specification and applying any server and queue default_chunk resource defaults that are missing from the select statement. The schedselect job attribute contains only vnode-level resources. The exec_vnode job attribute shows which resources are allocated from which vnodes. See "Job Attributes" on page 328 of the PBS Professional Reference Guide.

14.3.1.3 Resources for Requeued Jobs

When a job is requeued due to an error in the prologue or initialization, the job's exec_host and exec_vnode attributes are cleared. The only exception is when the job is checkpointed and must be rerun on the exact same system. In this case, the exec_host and exec_vnode attributes are preserved.

14.4 Grouping Jobs By Project

14.4.1 PBS Projects

In PBS, a project is a way to organize jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's project attribute specifies the job's project. Each job can be a member of up to one project.

Projects are not tied to users or groups. One user or group may run jobs in more than one project. For example, user Bob runs JobA in ProjectA and JobB in ProjectB. User Bill runs JobC in ProjectA. User Tom runs JobD in ProjectB. Bob and Tom are in Group1, and Bill is in Group2.

14.4.2 Assigning Projects to Jobs

A job's project can be set in the following ways:

- At submission, using the `qsub -P` option; see "qsub" on page 213 of the PBS Professional Reference Guide
- After submission, via the `qalter -P` option; see "qalter" on page 127 of the PBS Professional Reference Guide
- Via a hook; see the PBS Professional Hooks Guide

14.4.3 Managing Resource Use by Project

PBS can apply limits to the amount of resources used by jobs in projects, or the number of queued and running jobs belonging to projects. See section 5.15.1, "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues", on page 287.

14.4.4 Managing Jobs by Project

You can arrange for the jobs belonging to a project to run on designated hardware; see section 4.4.4, "Allocating Resources by User, Project or Group", on page 80. You can also run jobs belonging to a project in designated time slots; see section 4.4.6, "Scheduling Jobs into Time Slots", on page 84. For more information on routing by project, see section 4.9.39, "Routing Jobs", on page 205.

14.4.5 Viewing Project Information

Each job's project, if any, is specified in its project attribute. To see the value of this attribute, use the `qstat -f` option. See "qstat" on page 197 of the PBS Professional Reference Guide.

14.4.6 Selecting Jobs by Project

You can select jobs according to their project using the `qselect -P` option. See "qselect" on page 186 of the PBS Professional Reference Guide.

14.4.7 Default Project Value

The default value for a job's project attribute is "*_pbs_project_default*". Any job submitted without a specified value for the project attribute is given the default value. If you explicitly set the value to "*_pbs_project_default*", the server prints a warning message saying that the value has been set to the default. If you unset the value of the attribute in a hook, the value becomes the default value. Using `qalter -P ""` sets the value to the default.

14.4.8 Error Messages

When a job would exceed a limit by running, the job's comment field is set to an error message. See "Run Limit Error Messages" on page 387 of the PBS Professional Reference Guide.

14.5 Job Prologue and Epilogue

As of 2020.1, the prologue and epilogue are **deprecated**.

You can run a site-supplied script or program before and/or after each job runs. This allows initialization or cleanup of resources, such as temporary directories or scratch files. The script or program that runs before the job is the *prologue*; the one that runs after the job is the *epilogue*.

The primary purpose of the prologue is to provide a site with some means of performing checks prior to starting a job. The epilogue can be used to requeue a checkpointed job. See section 9.3.7.3, "Requeueing via Epilogue", on page 429.

If you have any execjob_prologue hooks, they supersede the prologue, and run when the prologue would run, and if you have any execjob_epilogue hooks, they supersede the epilogue, and run when the epilogue would run.

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

You can run a shell script as your prologue or epilogue, or you can use an execjob_prologue and/or execjob_epilogue hook to do the work. If you already have a shell script prologue and/or epilogue, you can run each via an appropriate execjob_prologue or execjob_epilogue hook. We show how to do this in section 14.5.2, "Using Hooks for Prologue and Epilogue", on page 512.

14.5.1 Using Shell Scripts for Prologue and Epilogue

Only one prologue and one epilogue may be used per PBS server. The same prologue and/or epilogue runs for every job in the complex.

Each script may be either a shell script or an executable object file.

14.5.1.1 When Shell Prologue and Epilogue Run

The prologue runs before the job is executed. The epilogue runs after the job terminates, including normal termination, job deletion while running, error exit, or even if `pbs_mom` detects an error and cannot completely start the job. If the job is deleted while it is queued, then neither the prologue nor the epilogue is run. If the job is discarded while running, for example when the server loses contact with the MoM, the epilogue does not run.

If a prologue or epilogue script is not present, MoM continues in a normal manner.

14.5.1.2 Where Shell Prologue and Epilogue Run

When multiple vnodes are allocated to a job, these scripts are run only by the MoM on the primary execution host.

The prologue runs with its current working directory set to `PBS_HOME/mom_priv`, regardless of the setting of SAND-BOX.

The epilogue runs with its current working directory set to the job's staging and execution directory. This is also where the job shell script is run.

14.5.1.3 Shell Prologue and Epilogue Location

Both the prologue and the epilogue must reside in the PBS_HOME/mom_priv directory.

14.5.1.4 Shell Prologue and Epilogue Requirements

In order to be run, the script must adhere to the following rules:

- The script must be in the PBS_HOME/mom_priv directory
- The prologue must have the exact name "prologue" under Linux, or "prologue.bat" under Windows
- The epilogue must have the exact name "epilogue" under Linux, or "epilogue.bat" under Windows
- The script must be written to exit with one of the zero or positive exit values listed in section 14.5.1.12, “Shell Prologue and Epilogue Exit Codes”, on page 511. The negative values are set by MoM
- Under Linux, the script must be owned by root, be readable and executable by root, and cannot be writable by anyone but root
- Under Windows, the script's permissions must give "Full Access" to the local Administrators group on the local computer

14.5.1.5 Shell Prologue and Epilogue Environment Variables

The prologue and epilogue run with the following set in their environment:

- The contents of the pbs_environment file
- The PBS_JOBDIR environment variable

TMPDIR is not set in the prologue environment or the epilogue environment.

14.5.1.6 Shell Prologue and Epilogue Permissions

Both the prologue and epilogue are run under root on Linux, or under an Admin-type account on Windows, and neither is included in the job session.

14.5.1.7 Shell Prologue and Epilogue Arguments

The prologue is called with the following arguments:

Table 14-1: Arguments to Prologue

Argument	Description
argv[1]	Job ID
argv[2]	User name under which the job executes
argv[3]	Group name under which the job executes

The epilogue is called with the following arguments:

Table 14-2: Arguments to Epilogue

Argument	Description
argv[1]	Job ID
argv[2]	User name under which the job executes
argv[3]	Group name under which the job executes
argv[4]	Job name
argv[5]	Session ID
argv[6]	Requested built-in resources (job's Resource_List)
argv[7]	List of resources used (job's resources_used) gathered from the primary execution host only
argv[8]	Name of the queue in which the job resides
argv[9]	Account string, if one exists
argv[10]	Exit status of the job

14.5.1.8 Shell Epilogue Argument Caveats

Under Windows and with some Linux shells, accessing argv[10] in the epilogue requires a shift in positional parameters. To do this, the script must do the following:

1. Call the arguments with indices 0 through 9
2. Perform a shift /8
3. Access the last argument using %9%

For example:

```
cat epilogue
> #!/bin/bash
>
> echo "argv[0] = $0" > /tmp/epiargs
> echo "argv[1] = $1" >> /tmp/epiargs
> echo "argv[2] = $2" >> /tmp/epiargs
> echo "argv[3] = $3" >> /tmp/epiargs
> echo "argv[4] = $4" >> /tmp/epiargs
> echo "argv[5] = $5" >> /tmp/epiargs
> echo "argv[6] = $6" >> /tmp/epiargs
> echo "argv[7] = $7" >> /tmp/epiargs
> echo "argv[8] = $8" >> /tmp/epiargs
> echo "argv[9] = $9" >> /tmp/epiargs
> shift
> echo "argv[10] = $9" >> /tmp/epiargs
```

14.5.1.9 Standard Input to Shell Prologue and Epilogue

Both scripts have standard input connected to a system-dependent file. The default for this file is `/dev/null`.

14.5.1.10 Standard Output and Error for Shell Prologue and Epilogue

The standard output and standard error of the scripts are connected to the files which contain the standard output and error of the job. There is one exception: if a job is an interactive PBS job, the standard output and error of the epilogue is pointed to `/dev/null` because the pseudo-terminal connection used was released by the system when the job terminated.

14.5.1.11 Shell Prologue and Epilogue Timeout

When the scheduler runs a job, it waits until the prologue has ended. To prevent an error condition within the prologue or epilogue from delaying PBS, MoM places an alarm around the script's/program's execution. The default value is *30 seconds*. If the alarm timeout is reached before the script has terminated, MoM will kill the script. The alarm value can be changed via the $prologalarm MoM configuration parameter. See section 9.6.6, “Prologue & Epilogue Running Time”, on page 447.

14.5.1.12 Shell Prologue and Epilogue Exit Codes

Normally, the prologue and epilogue programs should exit with a zero exit status. The prologue and epilogue should be written to exit with one of the zero or positive values listed here. When there is a problem with the script, MoM sets the exit value to one of the negative values. Exit status values and their impact on the job are listed in the following table:

Table 14-3: Prologue and Epilogue Exit Codes

Exit Code	Meaning	Prologue	Epilogue
-4	The script timed out (took too long).	The job will be requeued.	Ignored
-3	The `wait(2)` call waiting for the script to exit returned with an error.	The job will be requeued	Ignored
-2	The input file to be passed to the script could not be opened.	The job will be requeued.	Ignored
-1	The script has a permission error, is not owned by root, and/or is writable by others than root.	The job will be requeued.	Ignored
0	The script was successful.	The job will run.	Ignored
1	The script returned an exit value of *1*.	The job will be aborted.	Ignored
>1	The script returned a value greater than one.	The job will be requeued.	Ignored
2	The script returned a value of *2*.	The job will be requeued.	If the job was checkpointed under the control of PBS, the job is requeued.

MoM records in her log any case of a non-zero prologue or epilogue exit code, at event class 0x0001.

14.5.1.13 Shell Prologue and Epilogue Limitations and Caveats

- Consider having your epilogue write a lock file so that it can detect whether it is being run more than once for a job.
- You must exercise great caution in setting up the prologue to prevent jobs from being flushed from the system.
- Interactive-batch jobs cannot be requeued if the epilogue exits with a non-zero status. When this happens, these jobs are aborted.
- The prologue and epilogue cannot be used to modify the job environment or to change limits on the job.
- If any execjob_prologue hooks exist, they are run, and the prologue is not run.
- If any execjob_epilogue hooks exist, they are run, and the epilogue is not run.
- If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead.

14.5.2 Using Hooks for Prologue and Epilogue

You can run execjob_prologue and execjob_epilogue hooks to do whatever setup and cleanup you need before and after jobs run. Note that these hooks supersede the shell prologue and epilogue and prevent them from running. See "execjob_prologue: Event Just Before Execution of Top-level Job Process" on page 97 in the PBS Professional Hooks Guide, and "execjob_epilogue: Event Just After Killing Job Tasks" on page 104 in the PBS Professional Hooks Guide.

However, you can run your shell prologue and epilogue using execjob_prologue and execjob_epilogue hooks, and we provide an example hook called `run_pelog_shell.py`. The hook is included in `$PBS_EXEC/unsupported` as `run_pelog_shell.py`, along with its configuration file, `run_pelog_shell.ini`. You can see the contents at "execjob_prologue and execjob_epilogue Hook Examples" on page 266 in the PBS Professional Hooks Guide.

You can use this hook when the execjob_prologue and execjob_epilogue events are used in other hooks, such as the cgroups hook, and you still want to run the classic prologue and epilogue scripts we describe in section section 14.5.1, “Using Shell Scripts for Prologue and Epilogue”, on page 508. Additionally, the hook introduces parallel prologue and epilogue shell scripts.

On the primary execution host (the first host listed in PBS_NODEFILE), the standard naming convention of 'prologue' and 'epilogue' apply. Parallel prologues and epilogues use the naming conventions 'pprologue' and 'pepilogue', respectively, but run only on the secondary execution hosts. On Windows, parallel prologues and epilogues expect a '.bat' extension, which results in 'pprologue.bat' and 'pepilogue.bat'. This hook does the normal checks PBS does to start a prologue, such as permissions, etc., for UNIX. This hook also allows you to use a parallel prologue/epilogue (pprologue/pepilogue).

Parallel prologues will not run until a task associated with the job (i.e. via `pbs_attach`, `pbs_tmrsh`) begins on the secondary execution hosts. Parallel epilogues run only if the prologue ran successfully on the primary execution host. Only the primary execution host will have a value for resources_used in epilogue argument $7.

We assume the same requirements as listed for prologues/epilogues for running all types of prologue and epilogue shell scripts in section 14.5.1.4, “Shell Prologue and Epilogue Requirements”, on page 509.

By default, parallel prologue/epilogue is set to *False*. To enable parallel behavior, edit the configuration file and set ENABLE_PARALLEL to *True*.

The hook kills the prologue/epilogue 5 seconds before the hook_alarm timeout. At this point the job is requeued/deleted depending on the value of DEFAULT_ACTION. The hook_alarm time defaults to 30 seconds, giving the prologue/epilogue approximately 25 seconds to complete.

14.5.2.1 Installing Prologue and Epilogue Hooks

You could create a single hook that runs on both the execjob_prologue and the execjob_epilogue events, but to ensure execution order we separate them into the individual events by creating two separate hooks that use the same hook script.

Edit run_pelog_shell.ini to make configuration changes, then create and import the hook as we show here.

As root, run the following:

```
qmgr << EOF
create hook run_prologue_shell
set hook run_prologue_shell event = execjob_prologue
set hook run_prologue_shell enabled = true
set hook run_prologue_shell order = 1
set hook run_prologue_shell alarm = 35
import hook run_prologue_shell application/x-python default run_pelog_shell.py
import hook run_prologue_shell application/x-config default run_pelog_shell.ini

create hook run_epilogue_shell
set hook run_epilogue_shell event = execjob_epilogue
set hook run_epilogue_shell enabled = true
set hook run_epilogue_shell order = 999
set hook run_prologue_shell alarm = 35
import hook run_epilogue_shell application/x-python default run_pelog_shell.py
import hook run_epilogue_shell application/x-config default run_pelog_shell.ini
EOF
```

Any further configuration changes to run_pelog_shell.ini require re-importing the file to both hooks:

```
qmgr << EOF
import hook run_prologue_shell application/x-config default run_pelog_shell.ini
import hook run_epilogue_shell application/x-config default run_pelog_shell.ini
EOF
RERUN=14
DELETE=6
DEBUG=False
```

We show the defaults for the following constants in run_pelog_shell.ini. We show the contents of the file in "execjob_prologue and execjob_epilogue Hook Examples" on page 266 in the PBS Professional Hooks Guide. You can set them to match site preferences:

```
ENABLE_PARALLEL=False
VERBOSE_USER_OUTPUT=False
DEFAULT_ACTION=RERUN
TORQUE_COMPAT=False
```

14.6 Linux Shell Invocation

When PBS starts a job, it invokes the user's login shell, unless the user submitted the job with the -S option. PBS passes the job script, which is a shell script, to the login process.

PBS passes the name of the job script to the shell program. This is equivalent to typing the script name as a command to an interactive shell. Since this is the only line passed to the script, standard input will be empty to any commands. This approach offers both advantages and disadvantages:

14.6.1 Advantages

- Any command which reads from standard input without redirection will get an EOF.
- The shell syntax can vary from script to script. It does not have to match the syntax for the user's login shell. The first line of the script, even before any #PBS directives, should be

  #!/shell

  where *shell* is the full path to the shell of choice, `/bin/sh, /bin/csh, ...`

  The login shell will interpret the `#!` line and invoke that shell to process the script.

14.6.2 Disadvantages

- An extra shell process is run to process the job script.
- If the script does start with a `#!` line, the wrong shell may be used to interpret the script and thus produce errors.
- If a non-standard shell is used via the -S option, it will not receive the script, but its name, on its standard input.

14.7 When Job Attributes are Set

The attributes of a job are set at various points in the life of the job. For a description of each job attribute, see "Job Attributes" on page 328 of the PBS Professional Reference Guide.

14.7.1 Job Attributes Set By `qsub` Command

Before the job is passed to the server, the `qsub` command sets these job attributes, in this order:

1. Attributes specified as options on the command line
2. Attributes specified in #PBS directives within the job script
3. Job attributes specified in the default_qsub_arguments server attribute
4. If the following job attributes have not already been set, they are set as follows:
   - Job_Name: set to the file name of the job script, or to "*STDIN*" if the script is entered via standard input
   - Checkpoint: set to "*u*" for unspecified.
   - Hold_Types: set to "*n*"
   - Join_Path: set to "*n*"
   - Keep_Files: set to "*n*"
   - Mail_Points: set to "*a*" for abort
   - Priority: set to *0* (zero)
   - Rerunnable: set to *True*
   - run_count: can be set by job submitter
   - Variable_List: the `qsub` command sets the following variables and appends them to the existing value of Variable_List: PBS_O_HOME, PBS_O_LANG, PBS_O_LOGNAME, PBS_O_PATH, PBS_O_MAIL, PBS_O_SHELL, PBS_O_WORKDIR, PBS_O_TZ, and PBS_O_SYSTEM
   - Submit_arguments: set to any submission arguments on the command line

14.7.2 Job Attributes Set at Server

When the job is passed from the qsub command to the server, the raw job information is available to any job submission hooks, which can alter the information. Once the job is at the server, the server sets the following attributes:

- Job_Owner: set to <username>@<submission host name>
- Variable_List: the following are added to the job's Variable_List attribute: PBS_O_QUEUE, PBS_O_HOST
- Output_Path: if not yet specified, the Output_Path attribute is set
- Error_Path: if not yet specified, the Error_Path attribute is set
- Rerunable: if the job is interactive, the Rerunable attribute is set to *False*
- run_count: incremented each time job is run
- project: if unset, the project attribute is set to "*_pbs_project_default*".
- Read-only attributes: the server sets the job's read-only attributes; see "Job Attributes" on page 328 of the PBS Professional Reference Guide
- Resource_List: adjusted to include inherited resources specified in the queue and server Resources_Default attributes, if those resources are not yet in the list
- Comment set when job is sent for execution or rejected; see section 14.7.3.1, "Comment Set When Running Job", on page 515

14.7.3 Attributes Changed by Operations on Jobs

14.7.3.1 Comment Set When Running Job

Before the server sends the job to an execution host, the server sets the job's comment to "Job was sent for execution at <time> on <execvnode>".

After the server gets a confirmation from the MoM, the server updates the job's comment to "Job run at <time> on <execvnode>".

If the MoM rejects the job, the server changes the job comment to "Not Running: PBS Error: Execution server rejected request".

14.7.3.2 Attributes Changed When Moving Job

If you move a job to a different queue or server, any default resources from the current queue or server are removed, and new defaults are inherited. See section 5.9.4.3, "Moving Jobs Between Queues or Servers Changes Defaults", on page 248. For information on the qmove command, see "qmove" on page 172 of the PBS Professional Reference Guide.

14.7.3.3 Attributes Changed When Altering Job

When the qalter command is used to alter a job, the changes to the job are changes to the equivalent job attributes. See "qalter" on page 127 of the PBS Professional Reference Guide.

14.7.3.4 Attributes Changed When Requeueing or Rerunning a Job

When a job is requeued or rerun, its exec_vnode and/or exec_host attributes may be changed. The job may end up running on different vnodes. See "qrerun" on page 178 of the PBS Professional Reference Guide.

Each time a job is run, its run_count attribute is incremented by the server.

14.7.3.5 Attributes Changed by Holding or Releasing a Job

When a job is held using the `qhold` command, or released using the `qrls` command:

- The Hold_Types attribute reflects the change
- The job_state attribute may be changed

See "Job Attributes" on page 328 of the PBS Professional Reference Guide and "qhold" on page 147 of the PBS Professional Reference Guide.

14.7.3.6 Attributes Changed by Suspending or Resuming a Job

When a job is suspended or resumed using the `qsig` command, the job's job_state attribute reflects the change in state. See "qsig" on page 192 of the PBS Professional Reference Guide.

14.8 Job Termination

A job can be terminated for the following reasons:

- You or the submitter can use `qdel` to kill the job
- The job can be preempted and requeued
- The job can go over a limit and be killed
- The job is submitted to a routing queue, and can never be routed (accepted by a destination queue)
- The server is restarted and the job cannot be recovered
- The job specifies a dependency that fails or is terminated
- The job is killed by a signal

14.8.1 Normal Job Termination

When there is no $action terminate script and a running job is terminated, via the `qdel <job ID>` command, because of a server shutdown, or because the job has exceeded a limit, PBS waits for a configurable amount of time between sending a SIGTERM and a SIGKILL signal to the job. The amount of time is specified in the kill_delay queue attribute. The default value for this attribute is *10 seconds*. PBS takes the following steps.

For a single-vnode job:

1. PBS sends the job a SIGTERM
2. PBS waits for the amount of time specified in the kill_delay queue attribute
3. PBS sends the job a SIGKILL

For a multi-vnode job:

1. The primary execution host MoM sends a SIGTERM to all processes on the primary execution host
2. If any of the processes of the top task of the job are still running, PBS waits a minimum of *kill_delay* seconds
3. The primary execution host MoM sends a SIGKILL to all remaining job processes on the primary execution host
4. The subordinate MoMs send a SIGKILL to all their processes belonging to this job

14.8.2 Using the qdel Command to Terminate a Job

You can delete a job using the qdel command. See "qdel" on page 140 of the PBS Professional Reference Guide.

qdel <job ID>

If there is an $action terminate script, it is used to terminate the job.

If there is no $action terminate script, the SIGTERM-delay-SIGKILL sequence described in section 14.8.1, "Normal Job Termination", on page 516 is used to terminate the job.

This command does not terminate provisioning jobs.

qdel -Wforce <job ID>

If MoM is reachable, MoM sends the job a SIGKILL signal, and files are staged out. If MoM is unreachable, the server discards the job. The job may or may not continue to run on the execution host(s).

This command terminates provisioning jobs.

14.8.3 Killing Job Processes

If you need to kill job processes, you can use the printjob command to find the job's session ID, and then kill those processes. See "printjob" on page 125 of the PBS Professional Reference Guide.

14.8.4 Hooks and Job Termination

If you qdel a job, any execjob_preterm hooks run on all the hosts allocated to a job. On the primary execution host, the hook executes when the job receives a signal from the server for the job to terminate. On a sister host, this hook executes when the sister receives a request from the primary execution host MoM to terminate the job, just before the sister signals the task on this host to terminate.

The execjob_preterm hook does not run for any other job termination. For example, it does not run on a qrerun or when a job goes over its limit.

See "execjob_preterm: Event Just Before Killing Job Tasks" on page 103 in the PBS Professional Hooks Guide.

14.8.5 Configuring Site-specific Job Termination

The default behavior of PBS is for MoM to terminate a job under the following circumstances:

- The job's usage of a resource exceeds the limit requested
- The job is deleted by the server on shutdown
- The job is deleted via the qdel command

MoM normally uses SIGTERM, waits for the amount of time specified in the queue's kill_delay attribute, then issues a SIGKILL. See section 14.8, "Job Termination", on page 516.

You may want PBS to run your own job termination script in place of the normal action. The termination script is run in place of a SIGTERM. The termination script runs only on the primary execution host. After the top job process is terminated, a KILL signal is sent to any other job processes running on other hosts.

You can define the desired termination behavior by specifying the script you want to run in the $action terminate parameter in the Version 1 configuration file. The $action terminate parameter takes this form:

$action terminate <timeout> ! <path to script> [args]

Where

<*timeout*> is the time, in seconds, allowed for the script to complete. A value of *zero* (*0*) indicates infinite time is allowed for the script to run.

<*path to script*> is the path to the script. If it is a relative path, it is evaluated relative to the PBS_HOME/mom_priv directory.

<*args*> are optional arguments to the script. Values for <*args*> may be any string not starting with a percent sign ("*%*").

Arguments with a percent sign, making up any of the following keywords, are replaced by MoM with the corresponding value:

Table 14-4: $action terminate Keywords

Keyword	Value Used by MoM
%jobid	Job ID
%sid	Session ID of task (job)
%uid	Execution UID of job
%gid	Execution GID of job
%login	Login name associated with UID
%owner	Job owner in form *name@host*
%auxid	Auxiliary ID (system-dependent)

14.8.5.1 Requirements for Termination Script

The script should exit with a value of *zero* when the job is terminated successfully. If the script exits successfully (with a zero exit status and before the time-out period), PBS does not send any signals or attempt to terminate the job. It is the responsibility of the termination script in this situation to ensure that the job has been terminated.

The script should exit with a non-zero value if the job was not successfully terminated. If the script exits with a non-zero exit status, the job is sent SIGKILL by PBS.

If the script does not complete in the time-out period, it is aborted and the job is sent SIGKILL.

14.8.5.2 Examples of Configuring Termination

Linux:

Example 14-1: To use a 60-second timeout, run PBS_HOME/mom_priv/endjob.sh, and pass the job's session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.sh %sid %uid %jobid
```

Example 14-2: To use an infinite timeout, run the system kill command with the signal *13*, and pass the job's session ID:

```
$action terminate  0 !/bin/kill -13 %sid
```

Windows:

Example 14-3: To use a 60-second timeout, run endjob.bat, and pass the job's session ID, user ID, and PBS jobs ID to the script:

```
$action terminate 60 !endjob.bat %sid %uid %jobid
```

Example 14-4: To use an infinite timeout, run the pbskill command, and pass the job's session ID:

```
$action terminate  0 !"C:/Program Files/PBS Pro/exec/bin/pbskill" %sid
```

14.8.5.3 Caveats and Restrictions on Termination

Under Windows, <*path to script*> must have a "`.bat`" suffix since it will be executed under the Windows command prompt `cmd.exe`. If the <*path to script*> specifies a full path, be sure to include the drive letter so that PBS can locate the file. For example, `C:\winnt\temp\terminate.bat`. The script must be writable by no one but an Administrator-type account.

14.8.6 Killing Jobs with a Signal

You or the job owner can kill a job by sending a kill signal to a job via `qsig`.

If a job is terminated via a signal while it is in the process of being sent to the execution host, the following happens:

- PBS writes a server log message:
  `Job;<job ID>;Terminated on signal <signal number>`
- The job is requeued
- If `qrun` is used to run the job, `qrun` does not set the job's comment

14.9 Job Exit Status Codes

The exit status of a job may fall in one of three ranges, listed in the following table:

Table 14-5: Job Exit Status Ranges

Exit Status Range	Reason	Description
X < 0	The job could not be executed	See Table 14-6, “Job Exit Codes,” on page 520
0 <=X < 128	Exit value of shell or top process	This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the `.logout` script if the user has such a script (`csh`). The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.
X >=128	Job was killed with a signal	This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9). The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see `wait(2)` or `waitpid(2)` for more information), are the value of the signal that killed the job. To interpret (or "decode") the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a SIGTERM (e.g. 143 - 128 = 15, signal 15 is SIGTERM). See the `kill(1)` manual page for a mapping of signal numbers to signal name on your operating system.

The exit status of jobs is recorded in the PBS server logs and the accounting logs.

Negative exit status indicates that the job could not be executed. Negative exit values are listed in the table below:

Table 14-6: Job Exit Codes

Exit Code	Name	Description
0	*JOB_EXEC_OK*	Job execution was successful
-1	*JOB_EXEC_FAIL1*	Job execution failed, before files, no retry
-2	*JOB_EXEC_FAIL2*	Job execution failed, after files, no retry
-3	*JOB_EXEC_RETRY*	Job execution failed, do retry
-4	*JOB_EXEC_INITABT*	Job aborted on MoM initialization
-5	*JOB_EXEC_INITRST*	Job aborted on MoM initialization, checkpoint, no migrate
-6	*JOB_EXEC_INITRMG*	Job aborted on MoM initialization, checkpoint, ok migrate
-7	*JOB_EXEC_BADRESRT*	Job restart failed
-10	*JOB_EXEC_FAILUID*	Invalid UID/GID for job
-11	*JOB_EXEC_RERUN*	Job was rerun
-12	*JOB_EXEC_CHKP*	Job was checkpointed and killed
-13	*JOB_EXEC_FAIL_PASSWORD*	Job failed due to a bad password
-14	*JOB_EXEC_RERUN_ ON_SIS_FAIL*	Job was requeued (if rerunnable) or deleted (if not) due to a communication failure between the primary execution host MoM and a Sister
-15	JOB_EXEC_QUERST	Requeue job for restart from checkpoint
-16	JOB_EXEC_FAILHOOK_RERUN	Job execution failed due to hook rejection; requeue for later retry
-17	JOB_EXEC_FAILHOOK_DELETE	Job execution failed due to hook rejection; delete the job at end
-18	JOB_EXEC_HOOK_RERUN	A hook requested for job to be requeued
-19	JOB_EXEC_HOOK_DELETE	A hook requested for job to be deleted
-20	JOB_EXEC_RERUN_MS_FAIL	Job requeued because server couldn't contact the primary execution host MoM

14.9.1 Job Exit Status Between 0 and 128 (or 256)

This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the `.logout` script if the user has such a script (`csh`).

14.9.2 Job Exit Status >= 128 (or 256)

This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9).

The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see `wait(2)` or `waitpid(2)` for more information), are the value of the signal that killed the job.

To interpret (or "decode") the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a SIGTERM (e.g. 143 - 128 = 15, signal 15 is SIGTERM). See the `kill(1)` manual page for a mapping of signal numbers to signal name on your operating system.

14.9.3 Logging Job Exit Status

The exit status of jobs is recorded in the PBS server logs and the accounting logs.

14.9.4 Exit Status of Interactive Jobs

The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.

14.10 Rerunning or Requeueing a Job

You can re-run a job using the `qrerun` command. To re-run a job means to kill it, and requeue it in the execution queue from which it was run. See "qrerun" on page 178 of the PBS Professional Reference Guide.

14.10.1 Requeueing a Job on a Dead Node

Before you requeue a job on a node you know to be dead, use `qmgr` to mark the node as *Down*. When the node is marked *Down*, `qrerun` the job.

14.10.2 Output from a Re-run Job

When you re-run a job, the job's existing standard output and error files are copied back to the server host and stored in `PBS_HOME/spool`. They are then sent with the job to MoM when the job is again run. The output of a job that is re-run is appended to the output from prior runs of the same job.

14.10.3 Caveats for `qrerun`

- Jobs lose their queue wait time when they are requeued, including when they are checkpointed or requeued during preemption.

14.10.4 Requeueing Caveats

- When requeueing a job fails, for example because the queue does not exist, the job is deleted.
- If a job's run_count attribute is already at the limit (20), and you requeue the job, the job will be held the next time the scheduler tries to run it.

14.10.5 Caveats for Jobs Started by PBS

PBS attempts to run a job a certain number of times before placing a hold on the job. You cannot prevent a job from being held after this number of attempts. You must explicitly release the hold.

14.11 Job IDs

14.11.1 Format of Job IDs

Job Identifier

<sequence number>[.<server name>][@<server>]

Job Array Identifier

Job array identifiers are a sequence number followed by square brackets:

<sequence number>[][.<server name>][@<server>]

Example:

1234[]

Note that some shells require that you enclose a job array ID in double quotes.

14.11.2 Range of IDs

The largest allowed value for a job ID or job array ID is set in the max_job_sequence_id server attribute. Minimum allowed is *9999999*. Maximum allowed is *999999999999*. After this has been reached, job IDs start again at zero.

14.11.3 Job IDs and Moving Jobs

If a job is qmoved from one server to another, the job's ID does not change.

14.11.4 Job IDs and Requeueing and Checkpoint/Restart

If a job is requeued without being checkpointed, or checkpointed and requeued, it keeps its original job ID.

14.12 Where to Find Job Information

Information about jobs is found in `PBS_HOME/server_priv/jobs` and `PBS_HOME/mom_priv/jobs`.

14.12.1 Deleted Jobs

If PBS tries to requeue a job and cannot, for example when the queue doesn't exist, the job is deleted.

14.12.2 Failed Jobs

Once a job has experienced a certain number of failures, PBS holds the job.

14.12.3 Job Information When Server is Down

When the PBS server is down, you can use the `pbs_dataservice` command to start the PBS data service by hand, and then run the `printjob` command at the server host. See "pbs_dataservice" on page 60 of the PBS Professional Reference Guide and "printjob" on page 125 of the PBS Professional Reference Guide.

14.12.4 Job Information on Execution Host

You can use the `printjob` command to look at job information on the execution host. See "printjob" on page 125 of the PBS Professional Reference Guide.

14.13 Job Directories

PBS jobs use two kinds of directories. The first is the job's staging and execution directory. Files are staged into and out of this directory, and this is the directory where the job script executes.

The second is the job's temporary directory, where the job can create scratch files if necessary. The root of this directory is specified in the $tmpdir MoM configuration parameter. PBS creates the temporary directory, then sets the TMPDIR job environment variable to the path of the temporary directory. The job can then use this environment variable. See section 14.14.3, "Creation of TMPDIR", on page 527.

14.13.1 Staging and Execution Directories for Job

A job's *staging and execution directory* is the directory to which input files are staged, and from which output files are staged. It is also the current working directory for the job script, for tasks started via the `pbs_tm()` API, and for the epilogue.

Each PBS user may submit several jobs at once. Each job may need to have data files staged in or out. Each execution host needs a staging and execution directory for jobs. PBS can provide a job-specific staging and execution directory on each execution host for each job. The job's sandbox attribute controls whether PBS creates a staging and execution directory for each job, or uses the user's home directory for staging and execution.

PBS stages files to and from the primary execution host only. If the execution hosts use a shared file system, the staging and execution directory is available everywhere the job needs it.

When a job uses a job-specific staging and execution directory created by PBS, PBS does not require the job's owner to have a home directory on the execution host(s), as long as each MoM's $jobdir_root configuration option is set, and is set to something other than the user's home directory.

When you set $`jobdir_root` to a shared (e.g. NFS) directory, tell MoM it is shared by setting the *shared* directive after the directory name:

$jobdir_root <directory name> shared

Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

Staging is specified via the job's stagein and stageout attributes. The format is the following:

execution_path@[storage_host:]storage_path

The *execution_path* is the path to the staging and execution directory. On stagein, *storage_path* is the path where the input files normally reside, and on stageout, *storage_path* is the path where output files will end up.

14.13.1.1 The sandbox Job Attribute

If the job's sandbox attribute is set to *PRIVATE*, PBS creates a job-specific staging and execution directory for that job. If sandbox is unset, or is set to *HOME*, PBS uses the user's home directory as the job's staging and execution directory. Using the server's default_qsub_arguments attribute, you can specify the default for the sandbox attribute for all jobs. By default, the sandbox attribute is not set.

The user can set the sandbox attribute via `qsub`, for example:

```
qsub -Wsandbox=PRIVATE
```

The `-Wsandbox` option to `qsub` overrides default_qsub_arguments. The job's sandbox attribute cannot be altered while the job is executing.

Table 14-7: Effect of Job sandbox Attribute on Location of Staging and Execution Directory

Job's sandbox attribute	Effect
not set	Job's staging and execution directory is the user's home directory
HOME	Job's staging and execution directory is the user's home directory
PRIVATE	Job's staging and execution directory is created under the directory specified in MoM $jobdir_root configuration option. If $jobdir_root is unset, the staging and execution directory is created under the user's home directory.

14.13.1.2 Options, Attributes and Environment Variables Affecting Staging

The environment variable PBS_JOBDIR is set to the pathname of the staging and execution directory on the primary execution host. PBS_JOBDIR is added to the job script process, any job tasks created by the `pbs_tm()` API, the prologue and epilogue, and the MoM $action scripts.

The job's jobdir attribute is read-only, and is also set to the pathname of the staging and execution directory on the primary execution host. The jobdir attribute can be viewed using the `-f` option to `qstat`.

The following table lists the options, attributes, etc., affecting staging:

Table 14-8: Options, Attributes, Environment Variables, etc., Affecting Staging

Option, Attribute, Environment Variable, etc.	Effect
MoM's $jobdir_root option	Directory under which PBS creates job-specific staging and execution directories. Defaults to user's home directory if unset. If $jobdir_root is unset, the user's home directory must exist. If $jobdir_root does not exist when MoM starts, MoM will abort. If $jobdir_root does not exist when MoM tries to run a job, MoM will kill the job. Permissions on the directory specified in this option must be *1777*. When you set $jobdir_root to a shared (e.g. NFS) directory, tell MoM it is shared by setting the *shared* directive after the directory name: *$jobdir_root <directory name> shared* Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.
MoM's $usecp option	Tells MoM where to look for files in a shared file system; also tells MoM that she can use the local copy agent for these files.
Job's sandbox attribute	Determines which directory PBS uses for the job's staging and execution. If value is *PRIVATE*, PBS uses a job-specific directory it creates under the location specified in the MoM $jobdir_root configuration option. If value is *HOME* or is unset, PBS uses the user's home directory for staging and execution. User-settable per-job via `qsub -W` or through a PBS directive. See the `pbs_mom.8B` man page.
Job's stagein attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W`.
Job's stageout attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W`.
Job's jobdir attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via `qstat -f`.
Job's Keep_Files attribute	Determines whether output and/or error files remain on execution host. User-settable per job via `qsub -k` or through a PBS directive. If the Keep_Files attribute is set to *o* and/or *e* (output and/or error files remain in the staging and execution directory) and the job's sandbox attribute is set to *PRIVATE*, standard out and/or error files are removed when the staging directory is removed at job end along with its contents.
Job's PBS_JOBDIR environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, `pbs_tm` job tasks, prologue and epilogue, and MoM $action scripts.
Job's TMPDIR environment variable	Location of job-specific scratch directory.

Table 14-8: Options, Attributes, Environment Variables, etc., Affecting Staging

Option, Attribute, Environment Variable, etc.	Effect
PBS_RCP string in `pbs.conf`	Location of `rcp` command
PBS_SCP string in `pbs.conf`	Location of `scp` command; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.
Server's default_qsub_arguments attribute	Can contain a default for job's sandbox (and other) attributes.

14.13.1.3 Getting Information About the Job Staging and Execution Directory

The job's jobdir attribute is viewable via `qstat` or the equivalent API while a job is executing. The value of jobdir is not retained if a job is rerun; it is undefined whether jobdir is visible or not when the job is not executing.

14.13.1.4 Example of Setting Location for Creation of Staging and Execution Directories

To make it so that jobs with `sandbox=PRIVATE` have their staging and execution directories created under `/scratch`, as `/scratch/<job-specific_dir_name>`, put the following line in MoM's configuration file:

```
$jobdir_root /scratch
```

14.13.1.5 Staging and Execution Directory Caveats

If the user home directory is NFS mounted, and you want to use `sandbox=PRIVATE`, then root must be allowed write privilege on the NFS filesystem on which the users' home directories reside.

The directory specified in MoM's $jobdir_root parameter must have permissions set to *1777*.

When you set $jobdir_root to a shared (e.g. NFS) directory, tell MoM it is shared by setting the *shared* directive after the directory name:

$jobdir_root <directory name> shared

Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

14.14 The Job Lifecycle

14.14.1 Sequence of Events for Start of Job

This is the order in which events take place on an execution host at the start of a job:

1. Application licenses are checked out
2. Any job-specific staging and execution directories are created:
   - PBS_JOBDIR and job's jobdir attribute are set to pathname of staging and execution directory
   - Files are staged in
3. TMPDIR is created
4. The job's cpusets are created
5. The prologue is executed
6. The job script is executed

14.14.2 Sequence of Events for End of Job

This is the order in which events generally take place at the end of a job:

7. The job script finishes
8. The epilogue is run
9. The obit is sent to the server
10. Any specified file staging out takes place, including `stdout` and `stderr`
11. Files staged in or out are removed
12. Any job-specific staging and execution directories are removed
13. Job files are deleted
14. Application licenses are returned to pool
15. The job's cpusets are destroyed

14.14.3 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for this job. The root of TMPDIR is set by MoM to the value of MoM's $tmpdir configuration option. PBS sets TMPDIR to the pathname of the job-specific temporary scratch directory. This directory is for the use of the job, not PBS. This directory and its contents are removed when the job is finished.

The recommended TMPDIR configuration is to have a separate, local directory on each host. If the temporary scratch directory cannot be created, the job is killed.

14.14.4 Choice of Staging and Execution Directories

If the job's sandbox attribute is set to *PRIVATE*, PBS creates job-specific staging and execution directories for the job. If the job's sandbox attribute is set to *HOME*, or is unset, PBS uses the user's home directory for staging and execution. The staging and execution directory may be shared (e.g., cross-mounted) among all the hosts allocated to the job, or each host may use a separate directory. This is true whether or not the directory is the user's home directory.

14.14.4.1 Choosing Job-specific Staging and Execution Directories

When PBS creates a job-specific staging and execution directory, it does so under the directory specified in the MoM configuration option $jobdir_root. If the $jobdir_root option is not set, job-specific staging and execution directories are created under the user's home directory.

If the staging and execution directory is accessible on all of the job's execution hosts, these hosts will log the following message at the 0x0400 event class:

```
"the staging and execution directory <full path> already exists".
```

If the staging and execution directory is not cross-mounted so that it is accessible on all the job's execution hosts, each secondary host also creates a directory using the same base name as was used on the primary host.

If the staging and execution directory cannot be created the job is aborted. The following error message is logged at 0x0001 event class:

```
"unable to create the job directory <full path>".
```

When PBS creates a directory, the following message is logged at 0x0008 event class:

```
"created the job directory <full path>"
```

14.14.4.1.i Job-specific Staging and Execution Directory Caveats

- You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution. The pathname to each directory on each node may be different, since each depends on the corresponding MoM's $jobdir_root.
- The permissions for the directory specified in $jobdir_root must be *1777*.
- When you set $jobdir_root to a shared (e.g. NFS) directory, tell MoM it is shared by setting the *shared* directive after the directory name:

  $jobdir_root <directory name> shared

  Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

14.14.4.2 Choosing User Home Directory as Staging and Execution Directory

If the job's sandbox attribute is unset or is set to *HOME*, PBS uses the user's home directory for the job's staging and execution directory.

The user must have a home directory on each execution host. The absence of the user's home directory is an error and causes the job to be aborted.

14.14.5 Setting PBS_JOBDIR and jobdir Job Attribute

PBS sets PBS_JOBDIR and the job's jobdir attribute to the pathname of the staging and execution directory.

14.14.6 Staging Files Into Staging and Execution Directories

PBS evaluates `execution_path` and `storage_path` relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is the user's home directory or a job-specific directory created by PBS. PBS stages files to the primary execution host only. Staging is done as the job owner.

PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See section 15.6, “Setting File Transfer Mechanism”, on page 559.

14.14.7 Running the Prologue

The MoM's prologue is run on the primary host as root, with the current working directory set to `PBS_HOME/mom_priv` and with PBS_JOBDIR set in its environment.

14.14.8 Job Execution

PBS runs the job script on the primary host as the user. PBS also runs any tasks created by the job via the `pbs_tm()` API as the user. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in their environment. The job attribute jobdir is set to the pathname of the staging and execution directory on the primary host.

14.14.9 Standard Out, Standard Error and TMPDIRs

The job's `stdout` and `stderr` files are created directly in the job's staging and execution directory on the primary execution host.

14.14.9.1 Output and Error with Job-specific Staging and Execution Directories

If the `qsub -k` option is used, the `stdout` and `stderr` files will **not** be automatically copied out of the staging and execution directory at job end; they will be deleted when the directory is automatically removed.

14.14.9.2 Output and Error with User Home Directory as Staging and Execution Directory

If the `-k` option to `qsub` is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

14.14.10 Running the Epilogue

PBS runs MoM's epilogue script on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with PBS_JOBDIR set in its environment.

14.14.11 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `execution_path` and `storage_path` relative to PBS_JOBDIR. Files that cannot be staged out are saved in `PBS_HOME/undelivered`. PBS stages files out from the primary execution host only. Staging is done as the job owner.

PBS uses local file transfer mechanisms where possible. For remote file transfers, PBS uses the mechanism you specify. See section 15.6, "Setting File Transfer Mechanism", on page 559.

When the job is done, PBS writes the final job accounting record and purges job information from the server's database.

14.14.11.1 Staging Out with Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. If no errors are encountered during stageout and all stageouts are successful, the staging and execution directory and all of its contents are removed, on all execution hosts.

Files to be staged out are deleted all together, only after successful stageout of all files. If any errors are encountered during stageout, no files are deleted on the primary execution host, and the execution directory is not removed.

If PBS created job-specific staging and execution directories on secondary execution hosts, those directories and their contents are removed at the end of the job, regardless of stageout errors.

14.14.11.2 Staging Out with User Home Directory as Staging and Execution Directory

Files that are successfully staged out are deleted immediately, without regard to files that were not successfully staged out.

14.14.12 Removing TMPDIRs

PBS removes all TMPDIRs, along with their contents.

14.15 Managing Job History

14.15.1 Introduction

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

You can configure whether PBS preserves job history, and for how long, by setting values for the job_history_enable and job_history_duration server attributes.

14.15.2 Definitions

Moved jobs

Jobs which were moved to another server

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

History jobs

Jobs which will no longer execute at this server:

- Moved jobs
- Finished jobs

14.15.3 Job History Information Preserved by PBS

PBS can keep all job attribute information, including the following kinds of job history information:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that are running at another server
- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

14.15.4 Period When PBS Preserves Job History

PBS preserves history for the specified history duration beginning from the time a job finishes or is deleted.

After the duration has expired, PBS deletes the job history information and it is no longer available.

14.15.5 Configuring Job History Management

To configure job history, you enable it and you set the job history duration. You configure PBS to manage job history using the following server attributes:

job_history_enable

Enables or disables job history management. Setting this attribute to *True* enables job history management.

Format: Boolean.

Default: *False*

job_history_duration

Specifies the length of time that PBS will keep each job's history.

Format: duration: [[hours:]minutes:]seconds[.milliseconds]

Default: *Two weeks* (336:00:00)

14.15.5.1 Enabling Job History

To enable job history management, set the server's job_history_enable attribute to *True*:

```
Qmgr: set server job_history_enable=True
```

14.15.5.2 Setting Job History Duration

To set the length of time that job history is preserved, set the server's job_history_duration attribute to the desired duration:

```
Qmgr: set server job_history_duration=<duration>
```

If the job history duration is set to *zero*, no history is preserved.

If job history is enabled and job history duration is unset, job history information is kept for the default 2 weeks.

14.15.6 Changing Job History Settings

14.15.6.1 Disabling Job History

If job history is being preserved, and you unset the job_history_enable server attribute, PBS deletes all job history information. This information is no longer available.

14.15.6.2 Enabling Job History

If job history is not being preserved, and you set the job_history_enable server attribute, PBS begins preserving job history information for any jobs that are queued or running.

14.15.6.3 Modifying Job History Duration

Every job's history duration is set to the current value of the job_history_duration server attribute.

Example 14-5: Reducing job history duration:

The value of job_history_duration was "00:10:00" when a job finished execution. After 2 minutes, you change the duration to "00:06:00". This job's history is kept for a total of 6 minutes.

Example 14-6: Increasing job history duration:

The value of job_history_duration was "00:10:00" when a job finished execution. After 8 minutes you change the duration to "00:30:00". This job's history is kept for a total of 30 minutes.

Example 14-7: Increasing job history duration:

The value of job_history_duration was "00:10:00" when a job finished execution. After 11 minutes you change the duration to "00:30:00". This job's history is kept for a total of 10 minutes. The job's history is deleted after it is kept for 10 minutes.

14.15.7 Backward Compatibility

To have PBS behave as it did before the job history management feature was introduced, disable job history management. Do one of the following:

- Set the server's job_history_enable attribute to *False*:
  Qmgr: set server job_history_enable=False
- Unset the server's job_history_enable attribute:
  Qmgr: unset server job_history_enable
- Set the value of the server's job_history_duration attribute to *zero*, by doing one of the following:
  Qmgr: set server job_history_duration=0
  Qmgr: set server job_history_duration=00:00
  Qmgr: set server job_history_duration=00:00:00

14.15.8 Logging Moved Jobs

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the `qmove` command
- Job was submitted to a routing queue, then routed to a destination queue at another server

When a job is moved, the server logs the event in the server log and the accounting log. The server log messages are logged at log level 0x0008.

Format for the server log file:

```
7/08/2008 16:17:38;0008;Server@serverhost1;Job; 97.serverhost1.domain.com;Job moved to
    destination: workq@serverhost2
```

Format for the accounting log entry:

```
7/08/2008 16:17:38;M;97.serverhost1.domain.com;destination=workq@serverhost2
```

Record type: *M* (moved job)

14.15.9 Deleting Moved Jobs and Job Histories

You can use the `qdel -x` option to delete job histories. This option also deletes any specified jobs that are queued, running, held, suspended, finished, or moved. When you use this, you are deleting the job and its history in one step. If you use the `qdel` command without the `-x` option, you delete the job, but not the job history, and you cannot delete a moved or finished job. See "qdel" on page 140 of the PBS Professional Reference Guide.

14.15.10 Job History Caveats

- Enabling job history requires additional memory for the server. When the server is keeping job history, it needs 8kb-12kb of memory per job, instead of the 5kb it needs without job history. Make sure you have enough memory: multiply the number of jobs being tracked by this much memory. For example, if you are starting 100 jobs per day, and tracking history for two weeks, you're tracking 1400 jobs at a time. On average, this will require 14.3M of memory.
- If the server is shut down abruptly, there is no loss of job information. However, the server will require longer to start up when keeping job history, because it must read in more information.

14.16 Environment Variables

The settings in `$PBS_HOME/pbs_environment` are available to user job scripts. You must HUP the MoM if you change the file. This file is useful for setting environment variables for `mpirun` etc. For a list of environment variables used by PBS, see "PBS Environment Variables" on page 399 of the PBS Professional Reference Guide.

14.17 Adjusting Job Running Time

14.17.1 Shrink-to-fit Jobs

PBS allows you or the job submitter to adjust the running time of a job to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the min_walltime and max_walltime resources. PBS chooses the actual walltime. Any job that requests min_walltime is a **shrink-to-fit** job.

For a complete description of using shrink-to-fit jobs, see section 4.9.42, "Using Shrink-to-fit Jobs", on page 211.

14.18 Managing Number of Run Attempts

PBS has a built-in limit of 21 for the number of times the server can try to run a job or subjob. When the job or subjob goes over this limit, it gets a System hold. The number of tries is recorded in the job or subjob's run_count attribute. The run_count attribute starts at zero, and the job or subjob is held when run_count goes above 20. When a subjob's run_count attribute goes above 20, it and its parent job array get a System hold. You can use `qrls` on the parent array to release the parent array and indirectly release the subjobs. See "qrls" on page 180 of the PBS Professional Reference Guide.

Job submitters can set a non-negative value for run_count on job submission, and can use `qalter` to raise the value of run_count. A PBS Manager or Operator can use `qalter` to raise or lower the value of run_count.

14.19 Managing Amount of Memory for Job Scripts

By default, starting with version 13.1, PBS limits the size of any single job script to 100MB. You can set a different limit using the jobscript_max_size server attribute. The format for this attribute is *size*, and the units default to bytes. You can specify the units. For example:

```
Qmgr: set server jobscript_max_size = 10mb
```

Job script size affects server memory footprint. If a job submitter wants to use a really big script, they can put it in shared storage and call it from a short script, or they can run a small job script that stages in the big script, then calls it.

14.20 Allowing Interactive Jobs on Windows

1. Make sure that file and printer sharing is enabled. This is off by default.
2. Make sure that the ephemeral port range in your firewall is open on both the submission and execution hosts. Check your OS documentation for the correct range.
3. Make sure that IPC$ share is enabled. You should be able to run the following command from the submission host:

```
net use \\<execution_host>\IPC$
```

The output should look like this:

```
> net use \\myhost\IPC$
c:\Users\pbsuser>net use \\myhost\IPC$
Local name
Remote name       \\myhost\IPC$
Resource type     IPC
Status            Disconnected
# Opens           0
# Connections     1
The command completed successfully.
```

14.20.1 Configuring PBS for Remote Viewer on Windows

Job submitters can run interactive GUI jobs so that the GUI is connected to the primary execution host for the job. The job submitter runs a GUI application over a remote viewer. On Windows, PBS supports any remote viewer, such as Remote Desktop or X.

You can specify the remote viewer that PBS will use by setting a pbs.conf parameter on each submission host. See section 14.20.2, “Specifying Remote Viewer at Submission Hosts”, on page 535.

On an execution host that will launch a GUI application for an interactive job, MoM must run in a LocalSystem account. See section 14.20.3, “Configuring MoM to Run in LocalSystem Account on Windows”, on page 536.

A password is usually required when a Remote Desktop client tries to connect to an execution host. However you can configure Single Sign-on for Remote Desktop using the current login at the client host. See section 14.20.4, “Configuring Single Sign-on for Remote Desktop on Windows”, on page 536.

14.20.2 Specifying Remote Viewer at Submission Hosts

You can specify which remote viewer PBS should use when a job submitter runs a GUI job remotely. On each submission host, set the PBS_REMOTE_VIEWER parameter in pbs.conf to point to the remote viewer you want, or to a script that launches the desired remote viewer. If this parameter is unset, PBS uses the native Windows Remote Desktop client as the remote viewer. The line in pbs.conf should have this form:

```
PBS_REMOTE_VIEWER = <remote viewer client>
```

Example 14-8: Using the remote desktop client as the remote viewer:

```
PBS_REMOTE_VIEWER=mstsc /v
```

Example 14-9: Using the VNC viewer client as the remote viewer:

```
PBS_REMOTE_VIEWER=vncviewer.exe
```

Example 14-10: Launching a remote viewer via a script:

```
PBS_REMOTE_VIEWER=launch_remote_viewer.bat
```

14.20.3 Configuring MoM to Run in LocalSystem Account on Windows

On an execution host that will launch a GUI application for an interactive job, MoM must run in a LocalSystem account. To run MoM in a LocalSystem account, take the following steps:

1. Log in as administrator
2. Open `services.msc`
3. Right-click on the `pbs_mom` service and open "*properties*"
4. In the "*Log on*" tab, select "*Local System Account*"
5. Check "*Allow service to interact with desktop*"
6. Click *OK*

14.20.4 Configuring Single Sign-on for Remote Desktop on Windows

14.20.4.1 Configuring Submission Hosts for Single Sign-on

You can configure single sign-on using domain or local group policy. Follow these steps:

1. Log on to your local machine as an administrator
2. Start the Group Policy Editor:

   ```
   gpedit.msc
   ```

3. Navigate to "*Computer Configuration\Administrative Templates\System\Credentials Delegation*"
4. Double-click the "*Allow Delegating Default Credentials*" policy
5. Enable the policy
6. Click on the "*Show*" button to get to the list of servers
7. Add "TERMSRV/<server name>" to the server list.
8. You can add any number of server names to the list. A server name can be a hostname or an IP address. You can use one wildcard (*) per name. To store credentials for everything, use just a wildcard.
9. Confirm your changes by clicking on the "*OK*" button until you get back to the main Group Policy Object Editor dialog.
10. Repeat steps 3 through 7 for the following policies:
    a. "*Allow Delegating Default Credentials with NTLM-only Server Authentication*"
    b. "*Allow Delegating Saved Credentials with NTLM-only Server Authentication*"
    c. "*Allow Delegating Saved Credentials*"
11. In the Group Policy editor, navigate to *Computer Configuration -> Administrative Templates -> Windows Components -> Remote Desktop Services -> Remote Desktop Connection Client*
12. For the entry labeled "*Do not allow passwords to be saved*", change to *Disabled*
13. Force the policy to be refreshed immediately on the local machine. Run the following at a command prompt:

    ```
    gpupdate /force
    ```

14.20.4.2 Configuring Execution Hosts for Single Sign-on

The PBS execution host is the Remote Desktop server.

If the execution host is a Windows server, for example Windows Server 2008 R2, follow these steps:

1. Start Server Manager
2. Expand *Roles->Remote Desktop Services* and select *RD Session Host Configuration*
3. In the right pane in *Connections*, right-click *RDP-TCP Connection Name* and choose *Properties*
4. On the *Log on Settings* tab make sure "*Always prompt for password*" is unchecked
5. On the *General* tab choose the *Security* layer: *Negotiate* or *SSL (TLS 1.0)*
6. Click *OK*

If the execution host is not a Windows server, follow these steps:

1. Open the Group Policy Editor:
   `gpedit.msc`
2. Navigate to *Computer Configuration->Administrative Templates->Windows Components->Remote Desktop Services->Remote Desktop Session Host->Security*
3. Set "*Always prompt for password upon connection*" to "*Disabled*"

14.21 Releasing Unneeded Vnodes from Jobs

If you want to prevent unnecessary resource usage, you can release unneeded hosts or vnodes from jobs. You can use the `pbs_release_nodes` command or the release_nodes_on_stageout job attribute:

- You can use the `pbs_release_nodes` command at the command line, or submitters can use it or in their job scripts to release vnodes when the command is issued. You can use this command to release specific vnodes that are not on the primary execution host, or all vnodes that are not on the primary execution host. You can also use it to release all hosts or vnodes except for what you specify, which can be either a count of hosts to keep, or a select specification describing the vnodes to keep. You cannot use the command to release vnodes on the primary execution host. See "pbs_release_nodes" on page 92 of the PBS Professional Reference Guide.
- You can set the job's release_nodes_on_stageout attribute to *True* so that PBS releases all of the job's vnodes that are not on the primary execution host when stageout begins. You must set the job's stageout attribute as well. See "Job Attributes" on page 328 of the PBS Professional Reference Guide.
- You can use the default_qsub_arguments server attribute to specify that all jobs are submitted with release_nodes_on_stageout set by default.

For details, see "Releasing Unneeded Vnodes from Your Job", on page 128 of the PBS Professional User's Guide.

14.21.1 Caveats and Restrictions for Releasing Vnodes

- The job must specify a stageout parameter in order to release vnodes on stageout. If the job does not specify stageout, release_nodes_on_stageout has no effect.
- You can release only vnodes that are not on the primary execution host. You cannot release vnodes on the primary execution host.
- The job must be running (in the *R* state).
- The `pbs_release_nodes` command is not supported on vnodes tied to Cray X* series systems (vnodes whose vntype has the "cray_" prefix).
- If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.
- If a vnode on a multi-vnode host is assigned exclusively to a job, and the vnode is released, the job will show that the vnode is released, but the vnode will still show as assigned to the job in `pbsnodes -av` until the other vnodes on that host have been released. If a vnode on a multi-vnode machine is not assigned exclusively to a job, and the vnode is released, it shows as released whether or not the other vnodes on that host are released.
- If you specify release of a vnode on which a job process is running, that process is terminated when the vnode is released.

14.22 Tolerating Vnode Faults

PBS lets you allocate extra vnodes to a job so that the job can successfully start and run even if some vnodes fail. See section 9.5, “Vnode Fault Tolerance for Job Start and Run”, on page 434.

14.23 Managing Job Array Behavior

You can set a limit on the number of simultaneously running subjobs from any one array job by setting the job's max_run_subjobs attribute, either via `qalter -Wmax_run_subjobs=<new value> <job ID>`, or in a queuejob or modifyjob hook.

You can set a limit on the size of job arrays, either at the server, via the server max_array_size attribute, or at each queue, via the server max_array_size attribute.

Note that the `qrun` command overrides the limit on the number of simultaneously running subjobs for an array job set in the max_run_subjobs job attribute.

14.24 Recommendations

We recommend that as much as possible, you avoid huge numbers of jobs and subjobs. We recommend consolidating jobs where possible.

15 Administration

This chapter covers information on the maintenance and administration of PBS, and is intended for the PBS administrator. Topics covered include starting and stopping PBS, event logging, and accounting.

15.1 The PBS Configuration File

During the installation of PBS Professional, the installation script creates a configuration file named `pbs.conf`. This configuration file controls which daemons are to run on the local system, the directory tree location, and various runtime configuration options. Each host in a complex should have its own `pbs.conf` file.

15.1.1 Location of Configuration File

The configuration file is located in one of the following:

Linux:

```
/etc/pbs.conf
```

Windows:

```
[PBS Destination Folder]\pbs.conf
```

where `[PBS Destination Folder]` is the path specified when PBS is installed on the Windows platform, for example:

```
C:\Program Files\PBS\pbs.conf
```

or

```
C:\Program Files (x86)\PBS\pbs.conf
```

You can set the value of PBS_CONF_FILE in your environment in order to specify an alternate location for `pbs.conf`.

15.1.2 Format of Configuration File

Each line in the `/etc/pbs.conf` file gives a value for one parameter, or is a comment, or is blank. The order of the elements is not important.

15.1.2.1 Specifying Parameters

When you specify a parameter value, do not include any spaces in the line. Format for specifying a parameter value:

<parameter>=<value>

For example, to specify a value for PBS_START_MOM on the local host:

```
PBS_START_MOM=1
```

15.1.2.2 Comment Lines in Configuration File

You can comment out lines you are not using. Precede a comment with the hashmark ("#"). For example:

```
#This is a comment line
```

15.1.3 Example of Configuration File

The following is an example of a `pbs.conf` file for a host which is to run the server, the scheduler, and a MoM. The server runs on the host named Host1.ExampleDomain.

```
PBS_EXEC=/opt/pbs/M.N.P.S
PBS_HOME=/var/spool/PBS
PBS_START_SERVER=1
PBS_START_MOM=1
PBS_START_SCHED=1
PBS_SERVER=Host1.ExampleDomain
```

15.1.4 Contents of Configuration File

The `/etc/pbs.conf` file contains configuration parameters for PBS. The following table describes the parameters you can use in the `pbs.conf` configuration file:

Table 15-1: Parameters in `pbs.conf`

Parameter	Description
PBS_AUTH_METHOD	Specifies default authentication method and library to be used by PBS. Used only at authenticating client. Case-insensitive. Default value: *resvport* To use `MUNGE`, set to *munge*
PBS_BATCH_SERVICE_PORT	Port on which server listens. Default: 15001
PBS_BATCH_SERVICE_PORT_DIS	DIS port on which server listens.
PBS_COMM_LOG_EVENTS	Communication daemon log mask. Default: *511*
PBS_COMM_ROUTERS	Tells a `pbs_comm` the location of the other `pbs_comms`.
PBS_COMM_THREADS	Number of threads for communication daemon.
PBS_REMOTE_VIEWER	Specifies remote viewer client. If not specified, PBS uses native Remote Desktop client for remote viewer. Set on submission host(s). Supported on Windows only.
PBS_CORE_LIMIT	Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string "unlimited". If unset, core file size limit is inherited from the shell environment.
PBS_DATA_SERVICE_PORT	Used to specify non-default port for connecting to data service. Default: 15007

Table 15-1: Parameters in `pbs.conf`

Parameter	Description
PBS_ENCRYPT_METHOD	Specifies method and library for encrypting and decrypting data in client-server communication. Used only at authentication client side. Case-insensitive. To use TLS encryption in client-server communication, set this parameter to *tls*. No default; if this is not set, PBS does not encrypt or decrypt data.
PBS_ENVIRONMENT	Location of `pbs_environment` file.
PBS_EXEC	Location of PBS `bin` and `sbin` directories.
PBS_HOME	Location of PBS working directories.
PBS_LEAF_NAME	Tells endpoint what hostname to use for network. The value does not include a port, since that is usually set by the daemon. By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation. TPP internally resolves the name to a set of IP addresses, so you do not affect how `pbs_comm` works.
PBS_LEAF_ROUTERS	Location of endpoint's `pbs_comm` daemon(s).
PBS_LOCALLOG=<value>	Enables logging to local PBS log files. Valid values: 0: no local logging 1: local logging enabled Only available when using `syslog`.
PBS_LOG_HIGHRES_TIMESTAMP	Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is *HH:MM:SS*. With microsecond logging, format is *HH:MM:SS:XXXXXX*. Does not affect accounting log. Not applicable when using `syslog`. Overridden by environment variable of the same name. Valid values: *0*, *1*. Default: *0* (no microsecond logging)
PBS_LR_SAVE_PATH	Path where Undo Live Recorder stores daemon execution recordings. Default: `$PBS_HOME/spool`. See "Sending Daemon Execution Recordings to Altair" on page 663 in the PBS Professional Administrator's Guide

Table 15-1: Parameters in `pbs.conf`

Parameter	Description
PBS_MAIL_HOST_NAME	Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). For how this is used in email address, see section 2.2.2, "Specifying Mail Delivery Domain", on page 20.
PBS_MANAGER_SERVICE_PORT	Port on which MoM listens. Default: 15003
PBS_MOM_HOME	Location of MoM working directories.
PBS_MOM_NODE_NAME	Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by `gethostname()`. If you use the IP address for a vnode name, set `PBS_MOM_NODE_NAME=<IP address>` in `pbs.conf` on the execution host. Dots are not allowed in this parameter unless they are part of an IP address.
PBS_MOM_SERVICE_PORT	Port on which MoM listens. Default: 15002
PBS_OUTPUT_HOST_NAME	Host to which all job standard output and standard error are delivered. If specified in `pbs.conf` on a job submission host, the value of PBS_OUTPUT_HOST_NAME is used in the host portion of the job's Output_Path and Error_Path attributes. If the job submitter does not specify paths for standard output and standard error, the current working directory for the `qsub` command is used, and the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies only a file path for standard output and standard error, the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies paths for standard output and standard error that include host names, the specified paths are used. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Delivering Output and Error Files" on page 60 in the PBS Professional Administrator's Guide.
PBS_PRIMARY	Hostname of primary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.
PBS_RCP	Location of `rcp` command if `rcp` is used.
PBS_SCHEDULER_SERVICE_PORT	Port on which default scheduler listens. Default value: 15004

Table 15-1: Parameters in `pbs.conf`

Parameter	Description
PBS_SCHED_THREADS	Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable. Overridden by `pbs_sched -t` option and PBS_SCHED_THREADS environment variable. Default: 1
PBS_SCP	Location of `scp` command if `scp` is used; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.
PBS_SECONDARY	Hostname of secondary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
PBS_SERVER	Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER. Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.
PBS_SERVER_HOST_NAME	The FQDN of the server host. Used by clients to contact server. Overridden by PBS_PRIMARY and PBS_SECONDARY failover parameters. Overrides PBS_SERVER parameter. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Contacting the Server" on page 60 in the PBS Professional Administrator's Guide.
PBS_START_COMM	Set this to *1* if a communication daemon is to run on this host.
PBS_START_MOM	Default is *0*. Set this to *1* if a MoM is to run on this host.
PBS_START_SCHED	**Deprecated**. Set this to *1* if default scheduler is to run on this host. Overridden by scheduler's scheduling attribute.
PBS_START_SERVER	Set this to *1* if server is to run on this host.
PBS_SUPPORTED_AUTH_METHODS	Specifies supported authentication methods for client-server communication. Used by authenticating server (PBS server, scheduler, MoM, or comm); ignored at client. Case-insensitive. If this parameter is set, PBS accepts only the methods listed. Format: comma-separated list of authentication methods. Default value: *resvport* Example: *munge,GSS*

Table 15-1: Parameters in `pbs.conf`

Parameter	Description
PBS_SYSLOG=<value>	Controls use of `syslog` facility under which the entries are logged. Valid values: 0: no syslogging 1: logged via LOG_DAEMON facility 2: logged via LOG_LOCAL0 facility 3: logged via LOG_LOCAL1 facility ... 9: logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=<value>	Filters `syslog` messages by severity. Valid values: 0: only LOG_EMERG messages are logged 1: messages up to LOG_ALERT are logged ... 7: messages up to LOG_DEBUG are logged
PBS_TMPDIR	Location of temporary files/directories used by PBS components.

For information on how to use the `pbs.conf` file when configuring PBS for failover, see section 9.2.5.2, "Configuring the pbs.conf File for Failover", on page 409.

15.1.5 Configuration File Caveats and Recommendations

- Each parameter in `pbs.conf` can also be expressed as an environment variable.
- Environment variables override `pbs.conf` parameter settings.
- When you change a setting in a `pbs.conf` file, you must restart the daemon that reads the file.
- If you specify a location for `PBS_HOME` in the shell environment, make sure that this agrees with that specified in `pbs.conf`.
- Do not change a hostname without updating the corresponding Version 2 configuration file.
- Use a name for the server in the PBS_SERVER variable in the `pbs.conf` file that is not longer than 255 characters. If the short name for the server resolves to the correct host, you can use this in `pbs.conf` as the value of PBS_SERVER. However, if the fully-qualified domain name is required in order to resolve to the correct host, then this must be the value of the PBS_SERVER variable.
- If you set PBS_LEAF_NAME on a primary or secondary server host, make sure that PBS_PRIMARY and PBS_SECONDARY match PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on a server host, make sure that PBS_PRIMARY and PBS_SECONDARY match the hostnames of the server hosts.
- The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation.
- Do not include shell-style comments in the configuration file.
- When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

15.2 Environment Variables

PBS sets environment variables for different purposes: some variables are used by the daemons, commands, and jobs, and some environment variables are set individually for each job. Each parameter in `pbs.conf` can also be expressed as an environment variable. Environment variables override `pbs.conf` parameters.

15.2.1 Environment Variables For Daemons, Commands, and Jobs

The PBS installer creates an environment file called `pbs_environment`. This file is used by the daemons, commands, and jobs:

- Each PBS daemon initializes its environment using this environment file
- Several commands use environment variables to determine things like the name of the default server. The environment file is useful for setting environment variables for `mpirun`, etc.
- Jobs inherit the contents of this environment file before they acquire settings from `.profile` and `.login` files. Job scripts can use the environment variables set in the job's environment.

You can edit the environment file.

15.2.1.1 Contents of Environment File

When this file is created, it contains the following:

```
TZ=<local timezone, e.g. US/Pacific>
PATH=/bin:/usr/bin
```

For a list of PBS environment variables, see "PBS Environment Variables" on page 399 of the PBS Professional Reference Guide.

To support X forwarding, edit MoM's PATH variable to include the directory containing the `xauth` utility.

15.2.1.2 Location of Environment File

The PBS environment file is located here:

```
PBS_HOME/pbs_environment
```

15.2.1.3 Environment File Requirements

You must restart each daemon after making any changes to the environment file.

15.2.1.4 Editing Configuration Files Under Windows

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

15.2.2 Job-specific Environment Variables

For each job, the `qsub` command creates environment variables beginning with PBS_O_, and puts them in the job's environment. They are not written to `pbs_environment`. The server sets some of these environment variables if the `qsub` command does not set them.

For each job, the MoM on the primary execution host creates a file of the hosts to be used by the job. The node file is put in the job's environment, but the host list is not written to `pbs_environment`. The location of the node file is specified in the PBS_NODEFILE environment variable, which is set for the job only. See "The Job Node File", on page 77 of the PBS Professional User's Guide.

Some environment variables are set by commands. The PBS `mpiexec` script sets PBS_CPUSET_DEDICATED.

For a list of environment variables used and set by the `qsub` command, see "Environment Variables" on page 230 of the PBS Professional Reference Guide.

15.3 Event Logging

PBS provides event logging for the server, the scheduler, the communication daemon, and each MoM. You can use logfiles to monitor activity in the PBS complex.

15.3.1 PBS Events

The amount and type of output in the PBS event logfiles depends on the specified log filters for each component. Each PBS daemon can be directed to record only messages pertaining to certain levels of importance, called log levels. The specified log levels are logically "or-ed" to produce a mask representing the events to be logged by the daemon. The hexadecimal value for each log level is shown in Table 15-2, "PBS Events and Log Levels," on page 547. When events appear in the log file, they are tagged with their hexadecimal value, without a preceding "0x".

15.3.2 Event Logfiles

Each PBS daemon writes a separate event logfile. Each multisched writes its own logfile. By default, each daemon writes a file that has the current date as its name in the `PBS_HOME/<component>_logs` directory. The location of the logfile can be overridden with the -L option to each daemon's command. For example, to override the server's logfile location:

```
pbs_server -L <new path>
```

Whenever a new log file is opened, the daemon logs PBS_LEAF_NAME, PBS_MOM_NODE_NAME, and the hostname. The daemon also logs all network interfaces, listing each interface and all of the hostnames associated with that interface. In addition, it logs the PBS version and the build information.

Each daemon closes the day's log file and opens a new log file on the first message written after midnight. If no messages are written, the old log file stays open. Each daemon closes and reopens the same logfile when the daemon receives a SIGHUP.

Each daemon writes its version and build information to its event logfile each time it is started or restarted, and also when the logfile is automatically rotated out. The version and build information appear in individual records. These records contain the following substrings:

```
pbs_version = <PBSPro_stringX.stringY.stringZ.5-digit seq>
build = <status line from config.status, etc>
```

Example:

```
pbs_version = PBSPro_9.2.0.63106
build = '--set-cflags=-g -O0' --enable-security=KCRYPT ...
```

If the daemon cannot write to its log file, it writes the error message to the console. Some errors that appear before the daemon has backgrounded itself may appear on standard error.

The maximum number of characters in the message portion of a log entry is 4096.

15.3.3 Log Levels

PBS allows specification of the types of events that are logged for each daemon. Each type of log event has a different log level. All daemons use the same log level for the same type of event.

The following table lists the log level for each type of event.

Table 15-2: PBS Events and Log Levels

Name	Decimal	Hex	Event Description
PBSEVENT_ERROR	1	0x0001	Internal PBS errors
PBSEVENT_SYSTEM	2	0x0002	System (OS) errors, such as malloc failure
PBSEVENT_ADMIN	4	0x0004	Administrator-controlled events, such as changing queue attributes
PBSEVENT_JOB	8	0x0008	Job related events, e.g. submitted, ran, deleted
PBSEVENT_JOB_USAGE	16	0x0010	Job resource usage
PBSEVENT_SECURITY	32	0x0020	Security related events
PBSEVENT_SCHED	64	0x0040	When the scheduler was called and why
PBSEVENT_DEBUG	128	0x0080	Common debug messages
PBSEVENT_DEBUG2	256	0x0100	Debug event class 2
PBSEVENT_RESV	512	0x0200	Reservation-related messages
PBSEVENT_DEBUG3	1024	0x0400	Debug event class 3. Debug messages rarer than event class 2.
PBSEVENT_DEBUG4	2048	0x0800	Debug event class 4. Limit-related messages.

15.3.3.1 Specifying Log Levels

Each daemon uses an integer representation of a bit string to specify its log levels. The bit string can be decimal (or hexadecimal, for the MoM). Each daemon's log levels are specified in a a bit string that includes the events to be logged. You can specify each multisched's log levels individually.

For example, if you want the server to log all events except those at event classes 512, 1024, and 2048 (hex 0x0200, 0x0400, and 0x0800), you would use a log level of 511. This is 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1. If you want to log events at event classes 1, 2, and 16, you would set the log level to 19.

The following table shows the log event parameter for each daemon:

Table 15-3: Daemon Log Event Specification Parameters

PBS	Parameter/ Attribute	Reference	How to Make Parameter Take Effect
Server	log_events attribute	"Server Attributes" on page 281 of the PBS Professional Reference Guide	Takes effect immediately with `qmgr`

Table 15-3: Daemon Log Event Specification Parameters

PBS	Parameter/ Attribute	Reference	How to Make Parameter Take Effect
MoM	$logevent parameter	"Contents of MoM Configuration File" on page 240 of the PBS Professional Reference Guide	Requires SIGHUP to MoM
Scheduler	log_events attribute	"Configuration Parameters" on page 250 of the PBS Professional Reference Guide	Takes effect immediately with `qmgr`
communi-cation	PBS_COMM_LOG_EVENTS parameter in `pbs.conf`	"Daemon Log Mask" on page 46 in the PBS Professional Installation & Upgrade Guide	Restart the communication daemon

When reading the PBS event logfiles, you may see messages of the form "Type 19 request received from PBS_Server...". These "type codes" correspond to different PBS batch requests. See "Request Codes" on page 395 of the PBS Professional Reference Guide.

15.3.3.1.i Specifying Server Log Events

You can specify the server's log events by setting the server's log_events attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the `qmgr` command:

```
Qmgr: set server log_events = <value>
```

The new value takes effect immediately.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to *1025* (1024 +1, or 0x0401). To include all events, set the value to *4095* or *-1*. The default value for this attribute is 511. It can be set by Operators and Managers only. See "Server Attributes" on page 281 of the PBS Professional Reference Guide.

You can set the server's log level when you start the server using `pbs_server -e <log level>`. Note that you can specify a hexadecimal value this way, but not via the server's log_events attribute. When you use the `-e <log level>` option to `pbs_server`, that sets the server's log_events attribute to the corresponding integer value.

15.3.3.1.ii Specifying MoM Log Events

Each MoM's log events are specified in the $logevent parameter in that MoM's configuration file `PBS_HOME/mom_priv/config`. The parameter is an integer representation of a bit string, where the integer includes all events to be logged. For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to *1025* (1024 +1, or 0x0401). To set the value, add the $logevent line in `PBS_HOME/mom_priv/config`, then HUP the MoM. To include all events, set the value to *4095* (0xffffffff). The default value used by MoM is 975 (0x03cf). This parameter can be set by root only. See section 3.2, "Contents of MoM Configuration File", on page 240.

15.3.3.1.iii Specifying Scheduler Log Events

You can specify log events for the scheduler and for each multisched by setting each scheduler's log_events attribute. The attribute is an integer representation of a bit string, where the integer includes all events to be logged. To set the value, use the `qmgr` command:

```
Qmgr: set sched <scheduler name> log_events = <value>
```

The new value takes effect immediately.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to *1025* (1024 +1, or 0x0401). To include all events, set the value to *4095* or *-1*. The default value for this attribute is 767. It can be set by Operators and Managers only. See "Scheduler Attributes" on page 298 of the PBS Professional Reference Guide.

15.3.3.1.iv Specifying Communication Daemon Log Events

The communication daemon's log events are specified in the PBS_COMM_LOG_EVENTS parameter in `/etc/pbs.conf`. This parameter is an integer representation of a bit string, where the integer includes all events to be logged. HUP the daemon after you set the parameter.

For example, to log only debug event class 3 (1024, or 0x0400) and internal PBS errors (1, or 0x0001), set the value to *1025* (1024 +1, or 0x0401). To include all events, set the value to *2047* (or *-1*). The default value for this attribute is 511 (0x1ff). See "Logging and Errors with TPP" on page 54 in the PBS Professional Installation & Upgrade Guide and "Contents of Configuration File" on page 540.

15.3.4 Event Logfile Format and Contents

15.3.4.1 Event Logfile Format

Each component event logfile is a text file with each entry terminated by a new line. The format of an entry is:

<logfile date and time>;<event code>;<server name>;<object type>;<object name>;<message>

- The *logfile date and time* field is a date and time stamp in the format:

  mm/dd/yyyy hh:mm:ss[.xxxxxx]

If microsecond logging is enabled, microseconds are logged using the *.xxxxxx* portion. Microseconds may be preceded by zeroes. Microsecond logging is controlled per host via the PBS_LOG_HIGHRES_TIMESTAMP configuration parameter or environment variable.

- The *event code* is a bitmask for the type of event which triggered the event logging. It corresponds to the bit position, 0 to n, of each log event in the event mask of the PBS component writing the event record. See section 15.3.1, "PBS Events", on page 546 for a description of the event mask.
- The *server name* is the name of the server which logged the message. This is recorded in case a site wishes to merge and sort the various logs in a single file.
- The *object type* is the type of object which the message is about. All messages are associated with an *object type*. The following lists each possible *object type*:

Table 15-4: List of Event Logfile Object Types

Object Type	Usage
Svr	for server
Que	for queue
Job	for job
Req	for request
Fil	for file
Act	for accounting string
Node	for vnode or host
Resv	for reservation
Sched	for scheduler

- The *object name* is the name of the specific object.
- The *message* field is the text of the log message.

15.3.4.2 Scheduler Commands

These commands tell a scheduler why a scheduling cycle is being started. These commands appear in the server's logfile. Each has a decimal value, shown below. The following table shows commands from the server to a scheduler.

Table 15-5: Commands from Server to Scheduler

Value	Event Description
1	New job enqueued
2	Job terminated
3	Scheduler time interval reached
4	Cycle again after scheduling one job
5	Scheduling command from operator or manager
7	Configure
8	Quit (`qterm -s`)

Table 15-5: Commands from Server to Scheduler

Value	Event Description
9	Ruleset changed
10	Schedule first
11	A reservation's start time has been reached
12	Schedule a job (`qrun` command has been given)
13	Stopped queue is started
14	Job moved into local queue (queue at this server)
15	eligible_time_enable is turned on
16	PBS attempting to reconfirm degraded reservation

15.3.5 Logging Job Usage

PBS can log per-vnode cputime usage. The primary execution host MoM logs cputime in the format "hh:mm:ss" for each vnode of a multi-vnode job. The log level of these messages is 0x0100.

Under Linux, to append job usage to standard output for an interactive job, use a shell script for the epilogue which contains the following:

```
#!/bin/sh
tracejob -sl $1 | grep 'cput'
```

This behavior is not available under Windows.

15.3.6 Managing Log Files

15.3.6.1 Disk Space for Log Files

It is important not to run out of disk space for logging. You should periodically check the available disk space, and check the size of the log files PBS is writing, so that you know how fast you are using up disk space. Make sure that you always have more than enough disk space available for log files.

15.3.6.2 Dividing Up Log Files

You may wish to divide a day's logging up into more than one file. You may want to create a logfile that contains only the entries of interest. You can specify a file for a daemon's event log. See section 15.3.6.3, “Specifying Log File Path”, on page 552. The next sections describe how to break up your log files.

15.3.6.2.i Dividing Log Files on Linux

On Linux systems, all daemons close and reopen the same named log file when they are sent a SIGHUP. The process identifier (PID) of each daemon is available in its lock file in its home directory. You can move the current log file to a new name and send SIGHUP to restart the file using the following commands:

```
cd $PBS_HOME/<daemon>_logs
mv <current log file> <archived log file>
kill -HUP 'cat ../<daemon>_priv/<daemon>.lock'
```

15.3.6.2.ii Dividing Log Files on Windows

On Windows systems, you can rotate the event log files by stopping the service for which you want to rotate the logfile, moving the file, and then restarting that service. For example:

```
cd "%PBS_HOME%\mom_logs"
net stop pbs_mom
move <current log file> <archived log file>
net start pbs_mom
```

15.3.6.3 Specifying Log File Path

You may wish to specify an event logfile path that is different from the default path. Each daemon has an option to specify a different path for the daemon's event logfile. This option is the -L logfile option, and it is the same for all daemons. For example, to start the scheduler so that it logs events in `/scratch/my_sched_log`:

```
pbs_sched -L /scratch/my_sched_log
```

See the `pbs_server(8B)`, `pbs_sched(8B)`, and `pbs_mom(8B)` manual pages.

15.3.7 Extracting Logged Information

You can use the `tracejob` command to extract information from log files, such as why a job is not running or when a job was queued. The `tracejob` command can read both event logs and accounting logs. See the `tracejob(8B)` manual page.

15.3.8 Using the Linux `syslog` Facility

Each PBS component logs various event classes of information about events in its own log file. While having the advantage of a concise location for the information from each component, the disadvantage is that in a complex, the logged information is scattered across each execution host. The Linux syslog facility can be useful.

If your site uses the syslog subsystem, PBS may be configured to make full use of it. The following entries in pbs.conf control the use of syslog by the PBS components:

Table 15-6: Entries in pbs.conf for Using Syslog

Entry	Description
PBS_LOCALLOG=x	Enables logging to local PBS log files. Only possible when logging via syslog feature is enabled. 0 = no local logging 1 = local logging enabled
PBS_SYSLOG=x	Controls the use of syslog and syslog facility under which the entries are logged. If x is: *0* - no syslogging *1* - logged via LOG_DAEMON facility *2* - logged via LOG_LOCAL0 facility *3* - logged via LOG_LOCAL1 facility ... *9* - logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=y	Controls the severity level of messages that are logged; see /usr/include/sys/syslog.h. If y is: *0* - only LOG_EMERG messages are logged *1* - messages up to LOG_ALERT are logged ... *7* - messages up to LOG_DEBUG are logged

15.3.8.1 Caveats

- PBS_SYSLOGSEVR is used in addition to PBS's log_events mask which controls the class of events (job, vnode, ...) that are logged.
- If you use syslog, you cannot have daemons log events at microsecond resolution.

15.4 Managing Machines

15.4.1 Offlining Hosts and Vnodes

For using hooks to offline vnodes, see "Offlining Bad Vnodes" on page 66 in the PBS Professional Hooks Guide.

To offline an entire host, use the pbsnodes command. Use the name of the parent vnode, which is usually the name of the host:

```
pbsnodes -o <name of parent vnode>
```

All vnodes on this host are offlined.

To offline a single vnode, use the qmgr command, with the name of the vnode:

```
qmgr -c "set node foo[3] state=offline"
```

15.4.1.1 Caveats of Offlining

If you set a vnode with no running jobs *offline*, the server will not attempt to communicate with the vnode. Therefore, the server will not notice that the vnode is up until you clear the *offline* state. For example, a vnode that is both *down* and *offline* will not be marked up by the server until you clear the *offline* state.

15.4.2 Performing Maintenance on Powered-up Vnodes

15.4.2.1 Reserving Vnodes for Maintenance

You can create maintenance reservations using `pbs_rsub --hosts <host list>`. Maintenance reservations are designed to make the specified hosts available for the specified amount of time, regardless of what else is happening:

- You can create a maintenance reservation that includes or is made up of vnodes that are down or offline.
- Maintenance reservations ignore the value of a vnode's resv_enable attribute.
- PBS immediately confirms any maintenance reservation.
- Maintenance reservations take precedence over other reservations; if you create a maintenance reservation that overlaps an advance or standing job reservation, the overlapping vnodes become unavailable to the job reservation, and the job reservation is in conflict with the maintenance reservation. PBS looks for replacement vnodes; see "Reservation Fault Tolerance" on page 432 in the PBS Professional Administrator's Guide.

PBS will not start any new jobs on vnodes overlapping or in a maintenance reservation. However, jobs that were already running on overlapping vnodes continue to run; you can let them run or requeue them.

You cannot specify place or select for a maintenance reservation; these are created by PBS:

- PBS creates the reservation's placement specification so that hosts are assigned exclusively to the reservation. The placement specification is always the following:

  -lplace=exclhost
- PBS sets the reservation's resv_nodes attribute value so that all CPUs on the reserved hosts are assigned to the maintenance reservation. The select specification is always the following:

  -lselect=host=<host1>:ncpus=<number of CPUs at host1>+host=<host2>:ncpus=<number of CPUs at host2>+...

Maintenance reservations are prefixed with *M*. A maintenance reservation ID has the format:

M<sequence number>.<server name>

You cannot create a recurring maintenance reservation.

Creating a maintenance reservation does not trigger a scheduling cycle.

You must have manager or operator privilege to create a maintenance reservation.

15.4.2.2 Putting Vnodes into Maintenance State

You may want to perform maintenance on a vnode while it is powered up, but you don't want job processes running on it. You can suspend a job on a vnode and put the vnode into a *maintenance* state, where the scheduler will not start any new jobs on the vnode, using `qsig -s admin-suspend <job ID>`. You must suspend each job on the vnode; if you suspend only one, the rest will keep running. When you *admin-suspend* a job, all of the job's vnodes are put into the *maintenance* state, the job goes into the *S* state, and the job's processes are suspended.

Once the maintenance is finished, you can resume the *admin-suspend*ed jobs using `qsig -s admin-resume <job ID>`. The *admin-resume* signal directly resumes the job, without waiting for the scheduler. Once all of the vnode's jobs are *admin-resume*d, the vnode leaves the *maintenance* state.

You can see the list of jobs that were running on a vnode then *admin-suspend*ed in the maintenance_jobs vnode attribute. This attribute is a list of job IDs, and is readable only by managers.

15.4.2.2.i Resource Release on Suspension

When you *admin-suspend* a job, resources are released according to how you have configured the restrict_res_to_release_on_suspend server attribute; see section 5.9.6.2, "Job Suspension and Resource Usage", on page 250. However, no new jobs will run while the job is suspended.

15.4.2.2.ii Caveats for admin-suspend and admin-resume

- We recommend that before you *admin-suspend* any job, you disable scheduling and wait for the current scheduling cycle to finish. The scheduler queries vnode state only at the beginning of the scheduling cycle. If a vnode goes into the *maintenance* state after the start of the cycle, the scheduler could still schedule jobs onto that vnode.
- The *suspend* and *resume* signals are not interchangeable with the *admin-suspend* and *admin-resume* signals. For example, if a job is suspended via normal the *suspend* signal (`qsig -s suspend <job ID>`), it cannot be resumed with the *admin-resume* signal.

  Similarly, if a job is suspended with the *admin-suspend* signal, it cannot be resumed with the *resume* signal. Either request will be rejected with the following message:

  `"Job can not be resumed with the requested resume signal"`
- If there are multiple jobs on a vnode, we recommend using either *suspend* and *admin-suspend*, but not both. If you have a *suspend*ed job on a vnode that was in the *maintenance* state but is no longer, the scheduler could run jobs on the resources owned by the *suspend*ed job.
- If you want to perform maintenance on a vnode that has no jobs running on it, we recommend putting the vnode into the *offline* state before performing the maintenance.
- Any reservations on vnodes in the *maintenance* state are marked *degraded*. PBS searches for alternate vnodes for those reservations.
- Any vnode which had only *admin-suspend*ed subjobs will stay in the *maintenance* state after a server restart.

15.4.3 Changing Hostnames or IP Addresses

- Do not change a hostname without updating the corresponding Version 2 configuration file.
- Do not change the IP address or hostname of a machine in the complex while PBS is running. Stop PBS (server, scheduler, and MoMs), change the IP address, and restart PBS.

To change a hostname or IP address:

1. Make sure no jobs are running
2. Stop all PBS daemons
3. Make a backup of `PBS_HOME`
4. Change the hostname or IP address
5. If you are using the IP address as a vnode name, update PBS_MOM_NODE_NAME in `pbs.conf` on the execution host to the new IP address.
6. Restart all PBS daemons
7. If a host has a corresponding Version 2 configuration file, make sure that it is consistent with the new hostname
8. If you are running `nscd`, restart `nscd` on all hosts

15.4.4 Discovering Last Reboot Time of Server

Under Linux, you can find the timestamp of the most recent time PBS started up in `/var/tmp/pbs_boot_check`.

The permission of this file is set to 0644; only the PBS init script should modify this file. Do not modify this file. If you do so, you violate the configuration requirements of PBS.

This file is not available under Windows.

15.4.5 Changing Network Configuration

If you change any network configuration, restart the PBS daemons.

15.4.6 Replacing or Reimaging Nodes

When PBS_HOME is removed on a node by reimaging, etc., make sure that the server knows that there are no legitimate jobs on the node. Send each job `qsig -s SIGNULL` after the node is up again, which causes the server to contact the MoM and discover that any jobs are gone as far as MoM is concerned. The server then requeues and reruns any of MoM's gone jobs. Otherwise zombie jobs will ensure that no new jobs are scheduled to the node even after it's been reimaged.

MoM depends on its PBS_HOME to know which jobs are gone. When a node goes down on PBS complexes with either diskless nodes or nodes with integrated disk drives, sometimes the cluster manager will reimage the node before the complex reintegrates the node. In that case, PBS_HOME is gone, so MoM no longer knows about any jobs she was managing. The server will never get any updates or obits for those jobs, so they'll stay in state *R*.

If the reimaging process is longer than node_fail_requeue, the server will requeue the jobs, but your complex may use node_fail_requeue set to 0 for very good reasons, for example if there are Cray or HPE NUMA machines in the complex.

15.4.7 Restricting User Access to Execution Hosts

PBS provides a facility to prevent users who are not running PBS jobs from using machines controlled by PBS. You can turn this feature on by using the $restrict_user MoM directive. This directive can be fine-tuned by using the $restrict_user_exceptions and $restrict_user_maxsysid MoM directives. This feature can be set up host by host.

- A user requesting exclusive access to a set of hosts (via `place=excl`) can be guaranteed that no other user will be able to use the hosts assigned to his job, and PBS will not assign any unallocated resources on the vnode to another job.
- A user requesting non-exclusive access to a set of hosts can be guaranteed that no non-PBS users are allowed access to the hosts.
- A privileged user can be allowed access to the complex such that they can log into a host without having a job active.
- An abusive user can be denied access to the complex hosts.

The administrator can find out when users try to access hosts without going through PBS. The administrator can ensure that application performance is consistent on a complex controlled by PBS. PBS will also be able to clean up any job processes remaining after a job finishes running.

For a vnode with access restriction turned on:

- Any user not running a job who logs in or otherwise starts a process on that vnode will have his processes terminated.
- A user who has logged into a vnode where he owns a job will have his login terminated when the job is finished.
- When MoM detects that a user that is not exempt from access restriction is using the system, that user's processes are killed and a log message is output:

  ```
  01/16/2006 22:50:16;0002;pbs_mom;Svr;restrict_user;
  killed uid 1001 pid 13397(bash) with log event class PBSE_SYSTEM.
  ```

You can set up a list of users who are exempted from the restriction via the $restrict_user_exceptions directive. This list can contain up to 10 usernames.

Example 15-1: Turn access restriction on for a given node:

```
$restrict_user True
```

Example 15-2: Limit the users affected to those with a user ID greater than 500:

```
$restrict_user_maxsysid 500
```

Example 15-3: Exempt specific users from the restriction:

```
$restrict_user_exceptions userA, userB, userC
```

Note that a user who has a job running on a particular host will be able to log into that host.

15.4.7.1 Windows Restriction

The user access restriction feature is not supported on Windows.

15.5 Managing the Data Service

15.5.1 PBS Monitors Data Service

PBS monitors its connection to the data service. If the connection is broken (for example, because the data service is down), PBS tries to reestablish the connection. If necessary, PBS restarts the data service.

If failover is configured, and PBS cannot reestablish a connection, PBS quits.

If failover is not configured, PBS attempts to reestablish the connection until it succeeds.

When the server is stopped, it stops the data service.

15.5.2 Data Service Accounts

On Linux, PBS uses a PBS data service management account and an internal data service account. They are described here: "Create PBS Data Service Management Account" on page 23 in the PBS Professional Installation & Upgrade Guide.

15.5.3 Data Service Account Password

The default password for the internal data service account is a random password that is generated at installation, and which is known only to the PBS server.

15.5.3.1 Setting Data Service Account Name and Password

Changing the password is necessary only if you want to manually log into the database to check data or change something. Otherwise it is not necessary.

Use the `pbs_ds_password` command to change the password of the data service internal user account (not the PBS data service management account).

You can change the user account and/or password for the data service account using the `pbs_ds_password` command. Use this command if you need to change the user account or update the password for the data service account. You must be root or administrator to run the `pbs_ds_password` command. See "pbs_ds_password" on page 61 of the PBS Professional Reference Guide.

To change the data service account name:

```
pbs_ds_password -C <new user account>
```

To change the data service account password:

```
pbs_ds_password
```

15.5.3.2 Caveats

- When you specify a new name for the data service account, there must already be a data service management account with that name
- The account name cannot be changed while the data service is running.
- Do not delete `PBS_HOME/server_priv/db_password`. Doing so will prevent the `pbs_ds_password` command from being able to function.
- Do not change the data service password using any method other than the `pbs_ds_password` command.
- If you change the data service account after installing PBS, and then you want to upgrade PBS, you must change it back in order to perform the upgrade. After the upgrade, you can change it again. This is covered in the upgrading instructions.
- If you type in a password, make sure it does not contain restricted characters. The `pbs_ds_password` command generates passwords containing the following characters:

  0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&()_+*

  When creating a password manually, do not use \ (backslash) or ‘ (backquote). This can prevent certain commands such as `pbs_server`, `pbs_ds_password`, and `printjob` from functioning properly, as they rely on connecting to the database. The format is also described in "PBS Password" on page 357.

15.5.4 Starting and Stopping the Data Service

PBS automatically starts and stops the data service. However, you can start, stop, or check the status of the PBS data service using the `pbs_dataservice` command. See "pbs_dataservice" on page 60 of the PBS Professional Reference Guide.

To start the data service:

```
pbs_dataservice start
```

To stop the data service:

```
pbs_dataservice stop
```

To get the status of the data service:

```
pbs_dataservice status
```

15.5.4.1 Caveats for Starting and Stopping Data Service

- **Do not** start or stop the data service using anything except the `pbs_dataservice` command. Start or stop the data service using only the `pbs_dataservice` command.
- The data service cannot be stopped while the PBS server is running.

15.5.5 Changing Data Service Port

You can change the port that the data service listens on by changing the setting of the PBS_DATA_SERVICE_PORT entry in `pbs.conf`.

15.5.5.1 Caveats

- The PBS daemons must not be running when the port is changed.
- The data service must not be running when the port is changed.

15.5.6 File Ownership

The files under `PBS_HOME/datastore` are owned by the data service user account.

15.6 Setting File Transfer Mechanism

15.6.1 File Transfer in PBS

If PBS finds `scp` in the system PATH at install time, it sets the PBS_SCP parameter in `pbs.conf` to the path to `scp`.

MoM does the work of transferring files, using the mechanism you specify. MoM transfers files when she stages them in or out for a job, and when she delivers output and error files. MoM always tries to determine whether the source and destination for a file transfer are both local. If they are, she uses the local copy mechanism (`/bin/cp` on Linux, and `xcopy` on Windows). You can use the $usecp MoM configuration parameter to tell MoM which local directories are mapped to mounted directories, so that she can use the local copy mechanism.

For remote copying, PBS uses `scp` or `rcp`. In most installations, PBS is configured to use `scp` by default; at installation PBS looks for `scp` in the system PATH, and if it finds `scp` in the system PATH, PBS sets PBS_SCP in `pbs.conf`. You can configure PBS to use `rcp` for remote copying. You can also tell PBS to use any script or command for remote file transfer, such as `rsync`, `gsiftp`, etc.; see section 15.6.3.3, "Configuring MoM to Use Different Flags, a Script, or a Different Command", on page 563.

PBS ships with a version of `rcp` called `pbs_rcp`. On Windows, PBS uses this version by default.

PBS does not impose limitations on the size of files being transferred. Any limitations are caused by the commands themselves. The `pbs_rcp` command should be as fast as other implementations of `rcp`.

15.6.1.1 Configuration Parameters Affecting File Transfer

You configure MoM's file transfer mechanisms using the following:

- Local copy: the $usecp MoM configuration parameter
- Remote copy: the PBS_SCP and PBS_RCP entries in `pbs.conf`

15.6.1.2 How MoM Chooses File Transfer Method

If MoM knows that she is performing a local file transfer, she uses her local copy mechanism.

If MoM is transferring a remote file, she chooses according to the following:

- If PBS finds `scp` in the system PATH at install time, it sets the PBS_SCP parameter in `pbs.conf` to the path to `scp`.
- If a command is specified in PBS_SCP, MoM uses the command specified in PBS_SCP.
- If a command is specified in PBS_RCP, and PBS_SCP is not defined, MoM uses the command specified in PBS_RCP.
- If no `pbs.conf` parameters are defined, MoM uses `pbs_rcp`.

15.6.1.2.i When Multiple Attempts Are Required

If necessary, MoM tries to transfer a file multiple times, with an increasing delay between each attempt:

- If MoM is using her local copy mechanism, she tries it up to four times
- If MoM is using the entry in PBS_SCP:
  - She first tries this, and if it fails, she tries `rcp`, `pbs_rcp`, or the entry in PBS_RCP if it is configured
  - She repeats this sequence four times
- If MoM is using `rcp`, `pbs_rcp`, or the entry in PBS_RCP, she tries it up to four times

15.6.1.3 Options Passed to File Transfer Commands

15.6.1.3.i Options Passed on Linux

MoM automatically uses these options on Linux:

Table 15-7: File Transfer Mechanism Options on Linux

Distance	Mechanism	Options
Remote	PBS_RCP entry, `rcp`, or `pbs_rcp`	`-rp`
Remote	PBS_SCP entry	`-Brvp`
Local	`/bin/cp`	`-rp`

15.6.1.3.ii Options Passed on Windows

MoM automatically uses these options on Windows:

Table 15-8: File Transfer Mechanism Options on Windows

Distance	Mechanism	Options
Remote	PBS_RCP entry, `rcp`, or `pbs_rcp`	`-E -r`
Remote	PBS_SCP entry	`-Brv`
Local	`xcopy`	`/e/i/q/y`

15.6.2 Configuring MoM for Local Copy

MoM uses her local copy mechanism whenever she knows she will perform a local copy. To tell her which directories can be treated as local, specify the mappings between local and mounted directories in MoM's $usecp configuration parameter.

15.6.2.1 Configuring the $usecp MoM Parameter

This tells MoM where to look for files in a shared file system, so that she can use the local copy agent for these files. This is useful when you have common mount points across execution hosts.

Format:

$usecp <hostname>:<source directory> <destination directory>

You can use a wildcard ("*") as the first element only, to replace *hostname*.

MoM uses a local copy mechanism to transfer files when staging or delivering output, under the following circumstances:

- The destination is a network mounted file system
- The source and destination are both on the local host
- The *source directory* can be replaced with the *destination directory* on *hostname*

You can map multiple directories. Use one line per mapping.

You must HUP MoM after making this change.

15.6.2.1.i Linux and $usecp

MoM uses `/bin/cp` for the local copy mechanism on Linux.

Format:

$usecp <hostname>:<source directory> <destination directory>

Use trailing slashes on both the source and destination directories.

Example 15-4: Configuring $usecp on Linux:

```
$usecp *:/home/ /home/
$usecp *.example.com:/home/ /home/
$usecp *:/home/user/ /home/user/
$usecp *:/data/ /data/
$usecp HostA:/users/work/myproj/ /sharedwork/proj_results/
```

15.6.2.1.ii Windows and $usecp

MoM uses `xcopy` for the local copy mechanism on Windows.

Format:

$usecp <host name>:<drive name>:<directory> <drive name>:<directory>

When a network location is mapped to a local drive, you can cover all host names and case-sensitivity using entries similar to these:

```
$usecp *:Q: Q:
$usecp *:q: q:
```

Using this mapping, when MoM sees files with this format:

<hostname>:Q:<file path>

or

<hostname>:q:<file path>

she passes them to the copy command with this format:

Q:<file path>

or

q:<file path>

Example 15-5: Mapping locations with different directory names:

```
$usecp HostB:C:/xxxxx C:/yyyyy
```

15.6.3 Configuring MoM for Remote Copy

15.6.3.1 Configuring MoM to use `scp` or PBS_SCP Entry

By default MoM uses `scp` for remote copying, if the `scp` command is in the system PATH when PBS is installed. If you want MoM to use `scp` for remote copying on Windows or if PBS_SCP was not set in `pbs.conf` by default on Linux, follow the steps below.

1. Make sure that `scp` and `ssh` are installed on each host involved in the file transfer.
2. If you use plain `scp` without a wrapper script, MoM calls it with the `-B` option, which requires passwordless authentication. If you have not done so, set up passwordless authentication on all machines involved in file transfer. See section 15.6.7.1, "Enabling Passwordless Authentication", on page 565.
3. To use `scp` on Windows, set up passwordless authentication on all machines involved in file transfer. See section 15.6.7.1, "Enabling Passwordless Authentication", on page 565.
4. Set PBS_SCP to the absolute path to `scp`.
5. If the MoM is already running, restart the MoM.

The PBS_SCP `pbs.conf` entry is the absolute path to a command or script used for remote transfer. When PBS_SCP is defined, this entry overrides PBS_RCP, and MoM tries PBS_SCP first for remote transfers.

MoM calls the command this way:

```
$PBS_SCP -Brvp <path to source> <username>@<destination>.<host>:<path to destination>
```

You cannot specify options inside the PBS_SCP entry.

15.6.3.2 Configuring MoM to use `rcp`, `pbs_rcp` or PBS_RCP Entry

If you want MoM to use `rcp`, you will need to remove PBS_SCP from the `pbs.conf` file and restart MoM.

If you want MoM to use a different `rcp`, or another mechanism such as a script:

1. Make sure that `rcp` and `rsh` are installed on each host involved in the file transfer.
2. Specify the absolute path to the command or script in the PBS_RCP entry in `pbs.conf`.
3. If MoM is running, restart MoM.

The PBS_RCP `pbs.conf` entry is the absolute path to a command or script used for remote transfer. If MoM is unable to copy using the PBS_SCP entry, she uses the entry in PBS_RCP as an alternate method.

MoM calls the command this way:

```
$PBS_RCP -rp <path to source> <username>@<destination>.<host>:<path to destination>
```

You cannot specify options inside the PBS_RCP entry.

15.6.3.3 Configuring MoM to Use Different Flags, a Script, or a Different Command

If you want MoM to use different flags to rcp or scp, or a different command, or your own script, for remote file transfer:

1. If needed, write a script that does what you need
2. Specify the path to the command or script in PBS_SCP in pbs.conf
3. If the MoM is already running, restart the MoM.

When MoM calls PBS_SCP, she calls it with the -Brvp (Linux) or -Brv (Windows) flags. This means that when you are writing a script, the arguments being passed to the script are:

$1 -Brvp or -Brv

$2 path to source

$3 path to destination

You choose which arguments the script passes to the command inside the script. If you are using a different command, make sure that you pass the correct flags to it.

Example 15-6: Pass desired options to scp by writing a wrapper script for scp that contains the desired options, and pointing PBS_SCP to the wrapper script. In this case, we don't use the default -Brvp, which is passed to the script as $1. The script does not pass $1 to scp; instead, it specifies -Br. We do pass in the source and destination as $2 and $3.

In pbs.conf:

```
PBS_SCP=/usr/bin/scp_pbs
```

In /usr/bin/scp_pbs:

```
#!/bin/sh
/usr/bin/scp -Br $2 $3
```

Example 15-7: Use rsync by writing a wrapper script that passes all arguments except for the first (-Brvp) to rsync, and pointing PBS_SCP to the wrapper script. In this case, the script passes all but the first argument to rsync as $*. We get rid of the first argument using the shift command.

In pbs.conf:

```
PBS_SCP=/usr/bin/rsync_pbs
```

In /usr/bin/rsync_pbs:

```
#!/bin/sh
shift
/usr/bin/rsync -avz -e ssh $*
```

For remote copying, MoM tries the PBS_SCP entry in pbs.conf first. If you configure both PBS_RCP and PBS_SCP with scripts or commands, put the script or command that you want MoM to try first in PBS_SCP.

15.6.4 Allowing Direct Write of Standard Output and Error to /dev/null

Standard output and standard error are normally written to a location such as /var/spool, then copied to their final location. To avoid creating these files at all, and to avoid copying them, you need two things:

In MoM's version 1 configuration file, add this:

```
$usecp $<MoM hostname>:/dev/null /dev/null
```

Job submitters can use direct write to send them to /dev/null:

```
qsub -koed -o /dev/null -e /dev/null
```

15.6.5 Troubleshooting File Transfer

15.6.5.1 Problems with rcp

When using rcp, the copy of output or staged files can fail for the following reasons:

- The user lacks authorization to access the specified system
- Under Linux, if the user's .cshrc prints any characters to standard output, e.g. contains an echo command, the copy will fail

You may encounter a strange hang in stageout, with jobs stuck in the *E* state for a long time. This can happen because rcp may be trying to connect to a port that's already in use. If your standard copy mechanism is scp, and you don't want to let PBS fall back on pbs_rcp, do one of the following:

- You can move pbs_rcp
- If you specify PBS_SCP, set PBS_RCP to /bin/false in pbs.conf
- If you are using CM/PAS and specify PBS_SCP in /etc/conf, put the PBS_SCP line after the PBS_RCP line

15.6.5.2 Problems with Directory Access

Local and remote delivery of output may fail for the following additional reasons:

- A directory in the specified destination path does not exist
- A directory in the specified destination path is not searchable by the user
- The target directory is not writable by the user

15.6.6 Advice on Improving File Transfer Performance

15.6.6.1 Avoiding Server Host Overload

Avoid staging files from the server host, unless you can isolate the daemons from the effects of CPU and memory usage by scp/ssh, by using a mechanism such as cpusets. Consider the impact from a large job array that causes many files to be staged from the server host. Instead, use a shared filesystem. See section 15.6.6.2, “Avoiding Remote Transfers in Large Complexes”, on page 564.

15.6.6.2 Avoiding Remote Transfers in Large Complexes

If you are running a very large HPC complex, consider using MoM's $usecp directive to avoid rcp and scp transfers. Instead, have your users place input files on a shared filesystem before submitting jobs, write their output to the shared filesystem, and keep as much as possible out of stdout and stderr.

15.6.6.3 Improving Performance for ssh

If network bandwidth is a limiting factor, you can use compression to improve performance. However, if CPU usage and/or memory are limiting factors, do not use compression, because compression also requires CPU and memory.

You can use compression ciphers that minimize the CPU and memory load required, for example arcfour or blowfish-cbc:

```
ciphers         arcfour,blowfish-cbc
```

15.6.6.4 Improving Performance when Staging Similar Files

If you are staging in many similar files, for example, for job arrays, you can use `rsync` in a wrapper script. Follow the instructions in section 15.6.3.3, "Configuring MoM to Use Different Flags, a Script, or a Different Command", on page 563.

15.6.6.5 Avoiding Limits on `ssh` Connections

To prevent `scp` requests being denied when using `ssh`, you can set higher limits on incoming `ssh` connections. By default `ssh` is configured to treat more than 10 incoming connections (plus 10 in the authentication phase) as a denial-of-service attack, even on machines that could service many more requests.

Set higher limits in `/etc/ssh/sshd_config` for servers that are meant to service a lot of incoming openSSH sessions, but only on machines that have enough CPU and memory to service all of the requests.

See the MaxSessions and MaxStartups parameters in the man page for `sshd_config`. You can make these at least as large as the number of hosts in the cluster plus 10, assuming that any MoM only has one `scp` session open at any one time.

15.6.6.6 Alternatives to Changing `ssh` Limits

To avoid having to change limits on incoming `ssh` connections, you can do the following:

- Use a mounted directory and employ $usecp MoM parameters. See section 15.6.6.2, "Avoiding Remote Transfers in Large Complexes", on page 564.
- Use compression to service more requests with the same amount of hardware resources. See section 15.6.6.3, "Improving Performance for ssh", on page 564.

15.6.6.7 Getting Around Bandwidth Limits

If you have bandwidth limits, you can use a command such as `gsiftp`, which allows you to specify the bandwidth you want to use for file transfer. Follow the instructions in section 15.6.3.3, "Configuring MoM to Use Different Flags, a Script, or a Different Command", on page 563.

15.6.7 General Advice on File Transfer

15.6.7.1 Enabling Passwordless Authentication

You must enable passwordless authentication so that job files can be staged in and out. You must also choose and set a file transfer mechanism such as `rcp` or `scp` for remote file copying. Before you set up the remote file copy mechanism, enable passwordless authentication for it.

Enable passwordless authentication for each machine in the complex, and for any machine from which or to which files will be transferred.

You can use any authentication method you want, such as a `shosts.equiv` file, an authorized keys file, or `.rhosts` authentication. You can choose a cipher and use encryption; balance the CPU time required by encryption with the CPU time required by MoMs and job tasks.

PBS requires that `rsh`/`rcp` and/or `ssh`/`scp` works between each pair of hosts where files will be transferred. Test whether you have succeeded by logging in as root, and using your chosen file transfer mechanism to copy a file between machines.

15.6.7.2 Using scp for Security

Unless your complex is a closed system, we recommend using scp instead of rcp, because scp is more secure.

15.6.7.3 Avoiding Asynchronous Writes to NFS

Asynchronous writes to an NFS server can cause reliability problems. If using an NFS file system, mount the NFS file system synchronously (without caching.)

15.6.7.4 Returning Output on Cray

If your site has disabled the use of remote operation functions ("r" commands) and output cannot be returned for jobs running on compute nodes, enable the use of the cp command by adding $usecp to the $PBS_HOME/mom_priv/config file on each login node. See section 15.6.2, “Configuring MoM for Local Copy”, on page 561.

15.6.7.5 Editing the pbs.conf File Under Windows

You can edit the pbs.conf file by calling the PBS program named "pbs-config-add". For example, on Windows systems:

```
\Program Files (x86)\PBS\exec\bin\pbs-config-add "PBS_SCP=\winnt\scp.exe"
```

Do not edit pbs.conf directly; this could reset the permission on the file, which could prevent other users from running PBS.

15.6.7.6 The pbs_rcp Command

15.6.7.6.i Exit Values for pbs_rcp

The pbs_rcp command exits with a non-zero exit status for any error. This tells MoM whether or not the file was delivered.

15.6.7.7 Caveats

- Output is not delivered if the path specified by PBS_SCP or PBS_RCP in pbs.conf is incorrect.
- When a job is rerun, its stdout and stderr files are sent to the server and stored in PBS_HOME/spool. When the job is sent out for execution again, its stdout and stderr are sent with it. The copy mechanism used for these file transfers is internal to PBS; you cannot alter it or manage it in any way.

15.7 Some Performance Tips

15.7.1 Improving Scheduling Performance

- The scheduler can run asynchronously, so it doesn't wait for each job to be accepted by MoM, which means it also doesn't wait for an execjob_begin hook to finish. For short jobs, this can give you better scheduling performance. The scheduler runs asynchronously by default when the complex is using TPP mode, and can run asynchronously only when the complex is using TPP mode. To run the scheduler asynchronously, set the throughput_mode scheduler attribute to *True*. For details on TPP mode, see "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide; for job throughput, see section 4.5.7.1, "Improving Throughput of Jobs", on page 97.
- If you limit the number of jobs queued in execution queues, you can speed up the scheduling cycle. See section 4.5.7.2, "Limiting Number of Jobs Queued in Execution Queues", on page 98.
- Avoid using dynamic resources where possible; see section 5.4.4, "Static vs. Dynamic Resources", on page 234
- We give advice on minimizing the impact hooks can have on scheduling in "Scheduling Impact of Hooks" on page 72 in the PBS Professional Hooks Guide.

15.7.2 Improving Communication Performance

- We give recommendations for improving communication daemon performance in "Recommendations for Maximizing Communication Performance" on page 51 in the PBS Professional Installation & Upgrade Guide.
- You can use placement sets to keep job processes topologically close to one another; see section 4.9.32, "Placement Sets", on page 168.
- See our recommendations on file transfer performance improvement in section 15.6.6, "Advice on Improving File Transfer Performance", on page 564.

15.7.3 Improving Hook Speed

- See our hook performance recommendations in "Performance Considerations" on page 73 in the PBS Professional Hooks Guide.

15.8 Temporary File Location for PBS Components

You can configure where all PBS components put their temporary files and directories on each system. You may want to avoid using the usual temporary file locations of `/tmp` and `/var/tmp`, because users tend to fill these up.

15.8.1 Default Location for Temporary Files

By default, on Linux platforms, PBS components put their temporary files and directories in `/var/tmp`. PBS uses this location because it is persistent across restarts or crashes, allowing diagnosis of a problem, whereas the contents of `/tmp` may be lost.

On Windows, the default location is `C:\WINNT\TEMP` if it is present, or `C:\WINDOWS\TEMP`.

15.8.2 Configuring Temporary File Location for PBS Components

You configure the location of temporary files and directories for PBS components by setting the value of the PBS_TMPDIR configuration parameter in the `/etc/pbs.conf` file on each system. Set this parameter to the directory to be used for storing temporary files and directories by all PBS components on that system.

After you set the location of temporary files and directories, restart all PBS components:

```
<path to init.d>/init.d/pbs restart
```

The location for temporary files and directories for PBS components is determined by the following settings, in order of decreasing precedence:

1. $tmpdir in `mom_priv/config` (affects `pbs_mom` only, not other components)
2. PBS_TMPDIR (for Linux) or TMP (for Windows) environment variable
3. PBS_TMPDIR in PBS configuration file
4. If none of the preceding settings are present, PBS uses default values:
   - `/var/tmp` (for Linux)
   - `C:\WINNT\TEMP` or `C:\WINDOWS\TEMP` (for Windows)

15.8.3 Requirements

- The specified directory must exist.

  If the configured temporary file location does not exist, PBS prints the following error message:

  ```
  <command>: No such file or directory (2) in chk_file_sec, Security violation "<directory>"
      resolves to "<directory>"
  <command>: Unable to configure temporary directory.
  ```
- The directory must be globally readable and writable.
- On Linux systems, the directory must have the sticky bit set in the file permissions.
- The directory must not present a security risk:
  - All parent directories of the configured temporary directory must be owned by a UID less than 11 and a GID less than 10.
    - If the assigned owner has write permission, the UID must be 10 or less.
    - If the assigned group has write permission, the GID must be 9 or less.
  - Each parent directory must not be writable by "other".

  If a PBS component detects a security risk for a file or directory, it prints the following messages and exits:

  ```
  <command>: Not owner (1) in chk_file_sec, Security violation "<directory>" resolves to
      "<directory>"
  <command>: Unable to configure temporary directory.
  ```

15.8.4 Advice and Recommendations for Temporary File Location

- Make sure that the location you choose for temporary files is cleaned periodically.
- In the past, some PBS components defaulted to `/tmp` for storing temporary files. All components now default to `/var/tmp`, which is most likely a persistent storage location. You should take this into account and adjust the cleaning of `/var/tmp` accordingly.
- If a PBS component prints a security error message and exits, fix the security problem and restart the component.

15.9 Administration Caveats

15.9.1 General Caveats

Do not manually delete files in PBS private directories.

15.9.2 Windows Caveats

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

15.10 Support for Globus

Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus.

15.11 Support for Hyperthreading

On Linux machines that have Hyper-Threading Technology, PBS can end up reporting and using the number of logical processors, instead of the number of physical CPUs, as the value for resources_available.ncpus.

PBS does not control how CPUs are allocated to processes within a job. That is handled by the OS kernel.

15.11.1 Linux Machines with HTT

On Linux, PBS uses the number of CPUs shown in `/proc/cpuinfo`. If the CPUs are hyper-threaded and hyper-threading is enabled, the number of virtual and physical CPUs is different.

15.11.2 Windows Machines with HTT

On Windows, PBS calls the `CPUCount` Windows function, which reports whether hyper-threading is enabled. If hyper-threading is enabled, MoM uses the number of physical CPUs. If hyper-threading is not enabled, MoM uses the number of CPUs reported by the OS. MoM logs whether or not hyper-threading is enabled.

15.11.3 Using Number of Physical CPUs

If you do not wish to use hyper-threading, you can configure PBS to use the number of physical CPUs. Do this by setting resources_available.ncpus to the number of physical cpus:

```
Qmgr: set node <vnode name> resources_available.ncpus=<number of physical CPUs>
```

15.11.4 Hyperthreading Caveats

On a cpusetted system, NEVER change the value for resources_available.ncpus, resources_available.vmem, or resources_available.mem.

15.12 How To...

15.12.1 How to Drain Jobs

You can drain jobs from the entire complex by setting up dedicated time. Do not allow jobs in the dedicated time queue. See section 4.9.10, “Dedicated Time”, on page 125.

You can drain jobs from a specific set of vnodes by creating a reservation that blocks out those vnodes for the desired amount of time. See section 4.9.37, “Reservations”, on page 197.

15.12.2 How to Find Out Which Daemons Should Be Running

On the host in question, look in /etc/pbs.conf, or the location pointed to in the PBS_CONF_FILE environment variable. Check the settings that specify whether each daemon should run. 1 means the daemon should run.

```
PBS_START_MOM
PBS_START_COMM
PBS_START_SERVER
PBS_START_SCHED
```

16 Configuring and Using PBS with Cgroups

16.1 Chapter Contents

16.1 Chapter Contents . . . AG-571
16.2 Introduction to Cgroups . . . AG-571
16.3 Why Use Cgroups? . . . AG-572
16.3.1 What PBS Can Do With Cgroups . . . AG-573
16.3.2 Examples of Using Cgroups . . . AG-573
16.4 How PBS Uses Cgroups . . . AG-573
16.4.1 Vnode Creation via Cgroups Hook . . . AG-573
16.4.2 Job Life Cycle with Cgroups . . . AG-574
16.4.3 Cgroup Subsystems . . . AG-574
16.5 Configuring Cgroups . . . AG-576
16.5.1 Prerequisites for Cgroups Hook . . . AG-576
16.5.2 Enabling and Tuning Hook According to Host and/or Vnode Type . . . AG-577
16.5.3 Cgroups Hook Configuration Parameters . . . AG-579
16.5.4 Finish Up . . . AG-599
16.5.5 Managing GPUs or Xeon Phi via Cgroups . . . AG-600
16.6 Configuring MPI for Cgroups . . . AG-604
16.6.1 Steps to Integrate MPI with PBS via ssh . . . AG-605
16.7 Managing Jobs with Cgroups . . . AG-606
16.7.1 Requesting Memory . . . AG-606
16.7.2 Limit Enforcement . . . AG-606
16.7.3 Examples of Requesting Cores and Hyperthreads . . . AG-606
16.7.4 Spawning Job Processes . . . AG-606
16.8 Caveats and Errors . . . AG-607
16.8.1 Interactions Between Suspend/resume and the cpuset Subsystem . . . AG-607
16.8.2 Caveats for Shrinking a Job on a Host . . . AG-607
16.8.3 Caveats for Using CUDA . . . AG-607
16.8.4 Do Not Change ncpus When cpuset Subsystem is Enabled . . . AG-607
16.8.5 Cgroups Hook Prevents Epilogue from Running . . . AG-607
16.8.6 Errors . . . AG-608

16.2 Introduction to Cgroups

The term cgroup (pronounced see-group, short for control groups) refers to a Linux kernel feature that was introduced in version 2.6.24.

A cgroup may be used to manage access to system resources and to account for resource usage. The root cgroup is the ancestor of all cgroups and provides access to all system resources. When a cgroup is created, it inherits the configuration of its parent. When a process assigned to a cgroup creates a child process, the child is automatically assigned to its parent's cgroup.

Once created, a cgroup may be configured to restrict access to a subset of its parent's resources. These restrictions may include such things as memory, NUMA nodes, and devices. These different resource classes are grouped into categories referred to as *cgroup subsystems*. When processes are assigned to a cgroup, the kernel enforces all configured restrictions.

In Cgroups v1, supported by this cgroups hook, the kernel supports a number of cgroup subsystems. A cgroup subsystem is a kernel component that modifies the behavior of the processes in a cgroup. Subsystems are sometimes also known as *cgroup resource controllers* or *cgroup controllers*. We describe PBS support for cgroup subsystems in this chapter.

PBS provides a hook that allows you to take advantage of cgroups. When the cgroups hook is enabled, it runs on every node assigned to the job. When a job is started, the hook creates a set of directories for the configured subsystems based on the resource requirements of the job and then places the job process within the cgroup.

While the job is running, the kernel, not PBS, enforces resource restrictions, based on the cgroup settings written by the hook earlier. The cgroups hook can be configured to periodically poll the job's cgroup and update resource usage. When the job finishes, the hook writes final resource usage to the job's resources_used attribute, and removes the cgroup directories it created to house the job.

16.3 Why Use Cgroups?

Without cgroups, Linux can define sets of processes as related, but cannot define a set of loosely coupled processes as a single entity. Without cgroups, PBS uses Linux sessions to track job processes, but less accurately than with cgroups. Linux sessions impose the following limitations:

- Restrictions on how processes have to be related: sessions must encompass a parent process and nothing else but its progeny; it is impossible to merge a job session and another session created by for example `sshd`. However, PBS can manage more than one session per host.
- Inability to set resource usage restrictions for the entire set of processes belonging to a job; while you can set per-process limits on memory or CPU time, you cannot limit a session or a group of sessions belonging to a job
- Inability to make job sessions be inescapable containers: the `setsid` call or command will make processes leave the current session and create their own unrelated session; users and applications can use it without restriction.

With cgroups, all of these issues can be avoided:

- Since cgroups offer a well-established and general way of grouping processes together, cgroup controllers can implement precise resource usage accounting, resource usage limits, and process control for the entire cgroup. The cgroup also persists until it is explicitly destroyed, allowing some resource usage counters in a cgroup to outlive the processes that were members of the cgroup.
- Root can add processes to a cgroup regardless of the relationship between the processes, so a daemon can force processes spawned by OS services to join an existing cgroup. A process or thread spawned by a process appears at first in the cgroup of the parent process, and a non-root process cannot use any library call to escape the cgroup unless another cgroup grants that process permission to change cgroups.

16.3.1 What PBS Can Do With Cgroups

- More correctly identify which processes are part of a PBS job, even when libraries mislead PBS by creating processes that move into their own sessions that are not registered with PBS.
- Make processes spawned through external systemd daemons, including `ssh`, fully join a job
- Prevent job processes from using more resources than specified; for example disallow bursting above limits set according to the resources requested by the job when it was submitted
- Keep job processes within defined memory and CPU boundaries, ensuring there is minimal interference between jobs sharing a host, in order to provide consistent job run times.
- More accurately track and account for resource usage, even at the end of the job, when the job processes have exited and can no longer be investigated
- Enable or disable access to devices
- Ensure jobs leave enough resources for OS processes, to avoid disrupting OS services and turning nodes into "black hole nodes" that will no longer correctly run jobs

In addition to leveraging the kernel's cgroups support, the cgroups hook can also optionally discover the hardware configuration of a host and create child vnodes that reflect the hardware configuration, for example a number of CPU sockets with locally attached memory and GPU device.

This allows configuring the server and scheduler for optimal job performance. This functionality is an extension of the functionality formerly supported by the cpuset MoM (since the cpuset controller is now just one of the cgroup controllers in the kernel).

16.3.2 Examples of Using Cgroups

- Limit all of a job's processes to 6 CPUs and 6GB of RAM on vnode A
- Ensure that if two MPI jobs share a host, they do not pin processes to the same CPUs
- Limit access to GPU devices to only those assigned to the job by the cgroups hook
- Partition a host into a number of socket-aligned vnodes for optimal placement of jobs
- Pair processor, memory, and coprocessors like Xeon Phi and GPUs for optimal job placement

16.4 How PBS Uses Cgroups

16.4.1 Vnode Creation via Cgroups Hook

The hook creates a child vnode for each NUMA node on a host when all of the following conditions are true:

- The hook is enabled on the host
- At least one subsystem is enabled on the host
- The vnode_per_numa_node parameter is set to *true*
- The hook finds a NUMA node on the host

For example, if the host named "myhost" has one NUMA node, you end up with two vnodes to represent the host: the parent vnode, named "myhost", and a vnode to represent the NUMA node, named "myhost[0]".

16.4.1.1 Caveats for Vnode Creation

- Make sure that you run the cgroups hook only after you have created the parent vnode.
- If the cgroups hook creates vnodes for a host, do not use any other method to create child vnodes for that host. For example, do not create special vnodes for GPUs.

16.4.2 Job Life Cycle with Cgroups

16.4.2.1 Running Single-host Jobs with Cgroups

When PBS runs a single-host job, the following happens:

1. PBS creates a cgroup on the host assigned to the job for each enabled subsystem. PBS assigns resources (CPUs, memory, and optionally co-processors such as GPUs or Xeon Phis) to the job on the host. It sets the required limits on resource usage in the cgroups created.
2. PBS places the top job process in each created cgroup. Cgroup semantics then automatically ensure that the progeny of the top job process is also confined to the correct cgroups.
3. The cgroup periodic hook collects resource usage information about the cgroups and hands it over to MoM; MoM uses these values when it reports job resource usage to the server to set the resources_used attribute.
4. When the job has finished, the cgroups hook reports CPU and memory usage to MoM and cleans up the cgroups.

16.4.2.2 Running Multi-host Jobs with Cgroups

When PBS runs a multi-host job, the following happens:

1. PBS creates a cgroup on each host assigned to the job. PBS assigns resources to the job and sets the required cgroup limits.
2. PBS places the top job process in the cgroup on the first node (the primary execution host). Any child processes that the job spawns on the primary execution host remain in the cgroup, even if they use calls to change Linux session.
3. The job creates processes on the remote hosts. If PBS integration of multihost applications has been done correctly, those processes will either:
   - Be created as children of MoM through a call to the TM API `tm_spawn` call (possibly indirectly through the use of `pbs_tmrsh`)
   - Be spawned by an external service and then registered as part of the job through a call to `tm_attach` (possibly indirectly through the use of `pbs_attach`)

   In both cases, the cgroups hook migrates the processes into the correct cgroups; the kernel then ensures their progeny remain in the cgroups.
4. On each host, the cgroup periodic hook collects usage information about the cgroups and hands it over to MoM; the primary execution host MoM periodically queries the resource usage on sister nodes, sums the contributions of all the nodes, and reports it to the server, which publishes it in the resources_used attribute.
5. When the job has finished, on each host, the cgroups hook reports final CPU and memory usage to the local MoM and cleans up the job cgroups. The primary execution host MoM collects resource usage information from all nodes, sums it across nodes, and sends it to the server, which makes a final update of the resources_used attribute .

16.4.3 Cgroup Subsystems

The cgroups hook can manage the subsystems listed in "Subsystems Managed by the Cgroups Hook" on page 575.

For other subsystems, listed in "Subsystems Used by Other Software" on page 575, the hook can only create and destroy per-job cgroups and move job processes into those cgroups; apart from that, it does not manage any values in those cgroups. Some of these subsystems are used by other software.

16.4.3.1 Cgroup Subsystems Managed by the Cgroups Hook

The cgroups hook can manage the following subsystems:

Table 16-1: Subsystems Managed by the Cgroups Hook

Subsystem	Functionality Used by Cgroups Hook
cpu	Set quota for CPU usage and/or set CPU usage shares for each job, to allow the Linux CFS scheduler to implement fair sharing See section 16.5.3.7, "cpu Subsystem", on page 587.
cpuacct	Monitor CPU resource usage See section 16.5.3.5, "cpuacct Subsystem", on page 584.
cpuset	Allocate and restrict jobs to specific CPU and optionally NUMA memory domains See section 16.5.3.6, "cpuset Subsystem", on page 585.
devices	Restrict job access to only specific character and block devices. See section 16.5.3.8, "devices Subsystem", on page 589.
memory	Set kernel-enforced per-job limits for memory usage; monitor memory usage. See section 16.5.3.9, "memory Subsystem", on page 591.
memsw	Set kernel-enforced per-job limits for swap usage; monitor swap usage. See section 16.5.3.10, "memsw Subsystem", on page 594.
hugetlb	Set kernel-enforced per-job limits for huge pages usage; monitor huge pages usage. See section 16.5.3.11, "hugetlb Subsystem", on page 596.

16.4.3.2 Cgroup Subsystems Not Managed by Cgroups Hook

When subsystems not managed by the cgroups hook are enabled, the cgroups hook only creates per-job cgroup directories, ensures job processes are moved into them, and deletes the per-job cgroup when the job ends. Managing any parameters is left to other hooks or software.

If you run another hook, for example the container hook, that expects per-job cgroups to have been created by the cgroups hook for a set of subsystems, make sure you enable these subsystems on the host.

The cgroups hook can create and destroy cgroups for, and move processes into, the following subsystems:

Table 16-2: Subsystems Used by Other Software

Subsystem	Functionality Used by Other Software
freezer	Can be used to suspend entire job
blkio	Can track and limit usage and bandwidth to specific disk block devices
pids	Can be used to track and limit number of processes in a job

Table 16-2: Subsystems Used by Other Software

Subsystem	Functionality Used by Other Software
net_cls	Can be used to tag network packets sent by the job with a job-specific class identifier that can be used in e.g. firewall configuration
net_prio	Can be used to dynamically set the priority of network traffic generated by each job
perf_events	Allow the `perf` tool to monitor a PBS Professional job as a group

16.5 Configuring Cgroups

You manage the behavior of the cgroups hook across your complex by setting parameters in the cgroups hook configuration file.

16.5.1 Prerequisites for Cgroups Hook

16.5.1.1 Ensure that Cgroups are Available

Many Linux distributions have cgroups available by default; for others you may need to install and enable cgroups. We provide some tips here for making sure that cgroups are available on your system:

- Verify that you have cgroups configured on your system:

  ```
  cat/proc/mounts|grep cgroup
  ```

  You should see cpuset, cpuacct, memory, etc. enabled.

  Each subsystem that will be enabled on the host by the configuration file should be listed (except memsw; see below).

  If you do not have cgroups available, install them.

  You may need to set your kernel flags so that they enable cgroup support.
- A cgroup subsystem may be disabled at boot time. To check for this, look for "cgroup_disable" entries in `/proc/cmdline` and take appropriate actions to remove it from the kernel command line parameters.
- The memsw subsystem is not a separate controller but an option of the memory controller that is present or absent depending on the kernel options at boot time.

  If you plan to enable the memsw subsystem, to verify whether swap accounting and limits are available:

  ```
  cat /proc/mounts | grep cgroup | grep memory | awk 'BEGIN {FS=" "} {print $2}' | head -n 1 | xargs
      --replace=dir ls dir/memory.memsw.usage_in_bytes
  ```

  If this lists a file, then memory plus swap accounting is turned on. If instead, `ls` reports it cannot access the file, then kernel support for it is disabled; in some kernels it is disabled by default.

  If memsw support is disabled, either add "`swapaccount=1`" to the kernel command line parameters to enable it and reboot the node, or ensure that the memsw section of the configuration file is disabled for the host.
- Make sure that cgroups will survive a reboot. Test whether cgroups survive by rebooting the host, then look to see whether cgroups are available. If they are not available, refer to the documentation for your Linux distribution.

16.5.1.2 Ensure that PBS Is Already Installed and Started

Make sure that PBS is installed and started.

16.5.2 Enabling and Tuning Hook According to Host and/or Vnode Type

You can use just one configuration file across a complex containing hosts with different configurations. You can enable the hook for a specific subset of the hosts in your complex. You can also tune the hook by enabling each subsystem independently according to host. For example if you have some hosts with swap, and some without, and some hosts with GPUs and some without, you can enable the memsw subsystem only for the hosts that have swap, and enable the devices subsystem only for the hosts that have GPUs.

You can similarly tune the hook for any parameter that takes *true* and *false*. So for example you can tune the soft_limit parameter in the memory subsystem so that soft_limit evaluates to *true* for certain hosts.

16.5.2.1 Vnode Types for Cgroups Hook

You can label each host to reflect its characteristics, then use the label when specifying which hosts are included in a subset. The labeling mechanism is a single string, in a file on MoM's host in `PBS_HOME/vntype`. We refer to this string as a "*vnode type*", and the hook stores the value of the string in the variable "vntype". You can define any vnode type you need. Write your vntypes using only alphanumerical characters and the delimiters ".", "-" and "_".

16.5.2.1.i Vnode Type File and vntype Resource

The resources_available.vntype vnode resource and the `vntype` file contents are related but different tools. Do not try to set the `vntype` file contents by changing resources_available.vntype for the vnode (this will not work). If you want the value of resources_available.vntype for the vnode to reflect the contents of the `vntype` file on the execution host, you can propagate the file string to the resource by setting the propagate_vntype_to_server parameter in the hook's configuration file to *True*.

16.5.2.2 Tuning Where Hook, Subsystems, and Parameters are Enabled

For each of the *true*/*false* parameters, and for the swappiness parameter, you can specify whether a host or vnode type is in, or not in, the list for which the parameter evaluates to *true*. A list is one or more comma-separated host or vnode names, specified using one of these:

"vntype in:"

"vntype not in:"

"host in:"

"host not in:"

Whitespace around the entries is ignored. You can use hostnames or vntypes, or Python `fnmatch` sequences, which allows "*" or "?" wildcards. Do not use commas inside an entry.

Example 16-1: If you have four vntypes "compute_swap", "compute_noswap", "gpu_swap", "gpu_noswap", you can set the swappiness parameter for the memory subsystem using

```
"swappiness" : "vntype in: *_swap"
```

and set the vnode_per_numa_node parameter in the main section using:

```
"vnode_per_numa_node" : "vntype in: gpu_*"
```

16.5.2.2.i Enabling the Hook and Subsystems

For the hook and each subsystem, the enabled parameter can be modified using the exclude_vntypes, exclude_hosts, include_hosts, and run_only_on_hosts parameters. In the following hierarchy, each parameter modifies the previous parameters. Always specify these parameters in this order in the configuration file:

1. enabled
2. exclude_hosts
3. exclude_vntypes
4. include_hosts
5. run_only_on_hosts

16.5.2.2.ii exclude_vntypes

Modifies the enabled parameter. JSON list of patterns for vntypes to exclude from enabled group. For example, to include all hosts except those without cgroups (marked with "no_cgroup" in their vntype):

```
"enabled" : true,
"exclude_vntypes": [ "*no_cgroup*"]
```

This is equivalent to:

```
"enabled" : "vntype not in: *no_cgroup*"
```

16.5.2.2.iii exclude_hosts

Modifies the enabled parameter. List of hosts to exclude from membership group. For example, to enable all hosts with GPUs (marked with "gpu" in their vntype), except for the hosts marked for testing:

```
"enabled" : "vntype in: gpu"
"exclude_hosts" : [ "gpu_test*" ]
```

16.5.2.2.iv include_hosts

Modifies and overrides the enabled, exclude_vntypes, and exclude_hosts parameters. List of hosts to include in membership group, despite having been among those excluded. For example, to include two "thin" hosts in the cpuset subsystem list, despite the fact that they did not qualify to be enabled in the "fat" group:

```
"cpuset"  : {
    "enabled" : "vntype in: fat"
    "include_hosts" : [ "test_cpuset_thin01", "test_cpuset_thin02" ]
```

This is equivalent to:

```
"cpuset"  : {
    "enabled" : "vntype not in: thin"
    "include_hosts" : [ "test_cpuset_thin01", "test_cpuset_thin02" ]
```

16.5.2.2.v run_only_on_hosts

Modifies the enabled, exclude_vntypes, exclude_hosts, and include_hosts parameters. Provides additional restriction on hosts otherwise qualified for inclusion in membership list. Any host included must pass all membership tests. For example, to include only hosts that are both "willing" and among the list of "able01", "able02", and "able03":

```
"enabled" : "vntype in: willing"
"run_only_on_hosts" : [ "able01", "able02", "able03" ]
```

16.5.2.2.vi Hook and Subsystem Enablement Tuning Parameters

The following parameters let you tune whether the hook and each subsystem is enabled on any host and/or vntype. You can set each parameter in this table for the entire hook and for each subsystem individually. Here we show the defaults in the configuration file, and in the code, as they apply to whether the hook itself is enabled. We do not list the defaults for each subsystem; they may be different.

Table 16-3: Cgroups Hook Global and Subsystem Membership Configuration Parameters

Parameter Name	Default Value: Configuration File	Default Value: Hook	Description
enabled	*true*	*True*	When *true*, the cgroups hook is enabled on the host. Can be modified using the exclude_vntypes, exclude_hosts, include_hosts, and run_only_on_hosts parameters. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
exclude_hosts	*[]*	*[]*	Modifies the enabled parameter. If not empty, specifies a list of hosts where the hook should be disabled. See section 16.5.2.2.iii, "exclude_hosts", on page 578
exclude_vntypes	*["no_cgroups"]*	*[]*	Modifies the enabled parameter. Specifies a list of vnode types for which the cgroups hook should be disabled. See section 16.5.2.2.ii, "exclude_vntypes", on page 578
include_hosts			Modifies the enabled and exclude_hosts parameters. If not empty, specifies a list of hosts where the hook should be enabled despite earlier exclusion. See section 16.5.2.2.iv, "include_hosts", on page 578
run_only_on_hosts	*[]*	*[]*	Overrides the enabled, exclude_hosts, exclude_vntypes, and include_hosts parameters. If not empty, specifies a list of hosts limiting the hosts for which the cgroups hook should be enabled to the matching hosts. See section 16.5.2.2.v, "run_only_on_hosts", on page 578

16.5.3 Cgroups Hook Configuration Parameters

The cgroups hook configuration file contains parameters that the hook uses as guides for its behavior, and parameters that the hook uses when it sets values in the cgroups directories. For any parameter that is unset in the configuration file, the cgroups hook uses defaults built into the hook. This file must conform to JSON syntax.

The cgroups hook configuration file is named *pbs_cgroups.json* before it is imported as the hook configuration file. After the file is imported as the hook configuration file, PBS names it *pbs_cgroups.CF*.

16.5.3.1 Global Parameters for Cgroups Hook

Here are the cgroups hook configuration parameters, except for the membership tuning parameters described above. Please note that some parameters may be different from those shown here:

Table 16-4: Cgroups Hook Configuration File Global Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled	*true*	*True*	When *true*, the cgroups hook is enabled on the host. Can be modified using the exclude_vntypes, exclude_hosts, include_hosts, and run_only_on_hosts parameters. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
cgroup_lock_file		*"/var/spool/pbs/mom_priv/cgroups.lock"*	This file ensures that only one hook event can manipulate the cgroups at any one time. The filesystem on which this file resides must support file locking.
cgroup_prefix	*"pbs_jobs"*	*"pbs_jobs"*	The parent directory under each cgroup subsystem where job cgroups are created. If the memory subsystem is located at `/sys/fs/cgroup/memory`, the memory cgroup for job 1.foo is found in the `/sys/fs/cgroup/memory/pbs_jobs.service/jobid/1.foo` directory.
kill_timeout	*10*	*10*	Specifies the maximum number of seconds the cgroups hook spends attempting to kill job processes before destroying cgroups
server_timeout	*15*	*15*	Specifies the maximum number of seconds the cgroups hook spends attempting to fetch node comments from the server
nvidia-smi		*"/usr/bin/nvidia-smi"*	The location of the `nvidia-smi` command on nodes supporting NVIDIA GPU devices. See section 16.5.5, "Managing GPUs or Xeon Phi via Cgroups", on page 600.
online_offlined_nodes	*true*	*false*	When enabled, if the periodic hook manages to confirm there are no orphan groups, it will online vnodes again if it can confirm they were earlier offlined by the cgroups hook. See section 16.5.3.4, "Automatic Onlining of Fixed Vnodes", on page 584.
periodic_resc_update	*true*	*false*	When this parameter is set to *true*, the hook periodically posts updates of the job's resource usage on this host to MoM. When this parameter is set to *false*, the usage is only sent to MoM when the job ends

Table 16-4: Cgroups Hook Configuration File Global Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
use_hyperthreads	*false*	*false*	When set to *true*, all CPU threads are made available to jobs. When *false*, only the first hyperthread of each core is made visible to jobs. See section 16.5.3.3, "Configuring Hyperthreading Support", on page 582.
ncpus_are_cores	*false*	*false*	When *true*, resources_available.ncpus of a vnode is the number of cores, and the hook assigns all threads of each core to a job. When *false*, resources_available.ncpus is the number of CPU threads available to jobs, and the hook assigns individual CPU threads to jobs. See section 16.5.3.3, "Configuring Hyperthreading Support", on page 582.
vnode_per_numa_node	*false*	*false*	When set to *true*, each NUMA node is represented by a separate vnode, and the host is managed by a resourceless parent vnode. When set to *false*, the entire host is represented by a single vnode. See section 16.5.3.2, "Setting vnode_per_numa_node", on page 581.
propagate_vntype_to_server		*true*	When set to *true*, the contents of the vntype file on the local host are propagated to the resources_available.vntype vnode resource. When set to *false*, the vntype file contents are not propagated to the vnode resources_available.vntype resource. See section 16.5.2.1.i, "Vnode Type File and vntype Resource", on page 577.

We show a sample file in section 16.5.3.12, "Sample Cgroups Hook Configuration File", on page 598. You can also export and look at the installed PBS cgroups hook configuration file:

```
qmgr -c "export hook pbs_cgroups application/x-config default" >pbs_cgroups.json
```

You can edit this file and change its parameters, then read it back in:

```
qmgr -c "import hook pbs_cgroups application/x-config default pbs_cgroups.json"
```

Note that if your configuration file is incomplete or not present, hook behavior may differ from what is expected.

16.5.3.2 Setting vnode_per_numa_node

When this is *false*, all resources of the host are presented to the server as a single vnode (the parent vnode). In a large complex, minimizing the number of vnodes makes it faster for the scheduler to select nodes for jobs, and if large parallel jobs span a set of small hosts used exclusively by one job at a time, there is little advantage in making subdivisions of the host visible to the server and scheduler.

However, on clusters where execution hosts run more than one job at a time, you can take advantage of hosts made up of a number of separate NUMA nodes. (A NUMA node is a set of CPUs with uniform access speed and latency to a set of local memory and PCIe resources. Usually a NUMA node maps to a socket in a multi-socket computer, but some processors integrate more than one NUMA node on a single socket. On both AMD and Intel processors, the number of NUMA nodes per socket also depends on BIOS configuration, which allows tuning the size of a NUMA node to the workload. The cgroups hook does not decide how many NUMA nodes there are; it relies on the Linux kernel's view of the NUMA nodes on a host.)

When this parameter is *true*, the scheduler is able to improve application performance in these ways:

- Run small jobs only on single-NUMA-node vnodes
- Give parallel applications smaller than a host exclusive use of their NUMA nodes
- Run jobs that request GPUs on vnodes where the CPUs are on the socket directly connected to the GPU's PCIe bus

When vnode_per_numa_node is *true*, the host is presented to the server as a parent vnode that has no resources, plus a number of child vnodes that are aligned to NUMA nodes and hold specific CPU, memory, and coprocessor resources such as GPUs or Intel Xeon Phi processors.

The scheduler can still spread a single chunk across several vnodes on the same host. To ensure that a job is placed on only one NUMA node, use `-lplace=group=vnode`. You can also group using custom resources that identify larger sets of well-connected vnodes.

The main drawback to enabling vnode_per_numa_node is the increase in the number of vnodes in a cluster, which may slow down the scheduler, and make the output of pbsnodes larger and more difficult to interpret. A second drawback is that certain classes of jobs will no longer fit in one vnode, making it more complex to ensure they are still placed in a well-connected set of vnodes.

16.5.3.3 Configuring Hyperthreading Support

Hyperthreading can increase the throughput for some applications; it can also allow the operating system to retain access to some idle CPU threads even when PBS jobs use every core. If you disable hyperthreading in the BIOS, the kernel must share the only visible thread in each core with PBS jobs. On the other hand,

- Hyperthreading makes it more complicated to run applications that get no performance benefits from it, especially on clusters where some nodes are hyperthread-enabled and others are not
- On a host that runs more than one job, for parallel applications, hyperthreading that is not tightly managed can have severe negative impacts on performance if the threads of a single core are running processes from unrelated applications

This is why the cgroups hook supports three different models of hyperthreading support:

"No hyperthreads" behavior

use_hyperthreads disabled

In this model PBS makes only the first thread of each core visible to PBS jobs, so if your workload cannot leverage hyperthreading well, you don't need to disable hyperthreading in the BIOS. The other CPU threads are still usable by the operating system, which means throughput is better than if hyperthreading support is disabled in the BIOS.

The value of resources_available.ncpus reflects the number of cores associated with a vnode, minus the cores whose threads have been marked as reserved by using exclude_cpus in the cpuset section of the configuration file.

This model is different from "Assign whole cores to jobs" behavior only when the cpuset subsystem is enabled.

"Default behavior"

use_hyperthreads enabled and ncpus_are_cores disabled

This mode mimics the behavior you would get without the cgroups hook, as well as the behavior of the former cpuset MoM.

The value of resources_available.ncpus reflects the number of CPU threads available on a vnode. For applications to request all threads of N cores, they must request 2*N ncpus on 2-way hyperthreaded hosts, or N ncpus on hosts where hyperthreading is disabled. The cgroups hook tries to allocate those threads from the minimum number of cores.

The main use case of this mode is a workload consisting of many single-threaded jobs for which running one job per thread rather than one job per core improves throughput. For example, on a host with a total of 24 2-way hyperthreaded cores, you can run 48 unrelated jobs instead of 24. The 48 jobs will run slower than if you ran only 24, but the total throughput might still be greater than if you had run 24 jobs.

"Assign whole cores to jobs" behavior

use_hyperthreads enabled and ncpus_are_cores enabled

In this model, hyperthreads are exposed to applications within a job, but one CPU of ncpus maps to all the threads of a single core. The cgroups hook assigns each CPU core exclusively to one job, but within a job the processes see all CPU threads of the assigned cores and can either choose to ignore hyperthreading or leverage it.

This is the easiest model to use when you do not need to map more than one job on a single core to increase throughput.

The value of resources_available.ncpus reflects the number of cores associated with a vnode, minus the cores whose threads have been marked as reserved by using exclude_cpus in the cpuset section of the configuration file.

This model is often preferred.

This model is different from "No hyperthreads" behavior only when the cpuset subsystem is enabled.

16.5.3.3.i Mixing Hyperthreading Models in a Complex

If you have a mixed workload or complex where you want to run some high-throughput single-threaded jobs, and some that take advantage of hyperthreading:

- You can tune your hook configuration file in order to partition your complex, where:
  - Some hosts are configured with ncpus_are_cores disabled, to run high-throughput single-threaded workloads
  - Other hosts are configured with ncpus_are_cores enabled
- You can allow jobs to request hyperthreaded hosts based on whether or not ncpus_are_cores is enabled

You can include hyperthreading information in each host's vnode type string in its vntype file, so that the cgroups hook parameters use_hyperthreads and ncpus_are_cores evaluate correctly to *true* or *false* for each host. You can set a resource on each host to indicate how hyperthreads are handled on that host, so that jobs can request the hyperthreading they want. You can also use this resource to associate hosts with queues, so that job submitters can specify a queue instead of requesting a resource. The string in the vntype file is probably going to be used for multiple characteristics such as GPUs, swap, etc., so you probably don't want to propagate it to resources_available.vntype.

Example 16-2: You use the custom resource ht to indicate whether and what kind of hyperthreads are available, and you use "ht" or "nohyper" in the vntype file to indicate whether hyperthreads exist. You can associate queues with hosts according to host configuration. If you have some hosts with hyperthreading and some without, some hosts with all threads of a core assigned to one job and some hosts without, and some hosts with GPUs and some without, you might end up with the following:

Table 16-5: Example of Mixing Hyperthreading Models

Host	use_hyperthreads	ncpus_are_cores	vntype file	resources_available.ht	queue name
hosta	*true*	*true*	gpu_ht	ht_by_core	ht_core_q
hostb	*true*	*true*	compute_ht	ht_by_core	ht_core_q
hoste	*true*	*false*	gpu_ht	ht_by_thread	ht_thread_q

Table 16-5: Example of Mixing Hyperthreading Models

Host	use_hyperthreads	ncpus_are_cores	vntype file	resources_available.ht	queue name
hostf	*true*	*false*	compute_ht	ht_by_thread	ht_thread_q
hostc	*false*	*true*	gpu_nohyper	core	core_q
hostd	*false*	*true*	compute_nohyper	core	core_q
hostg	*false*	*false*	gpu_nohyper	core	core_q
hosth	*false*	*false*	compute_nohyper	core	core_q

In our example, none of the job submitters care about whether their jobs run on hosts with GPUs; the GPU machines are here just to illustrate how you might use the vntype file string for more than one aspect of a host.

Jobs that want hyperthreads and want all of the threads for each core can request "ht=ht_by_core" in the select statement, or they can request or be routed to the queue named "ht_core_q".

Single-threaded jobs that are I/O bound and don't mind sharing a core can request "ht=ht_by_thread", or they can request or be routed to the queue named "ht_thread_q".

Single-threaded jobs that want non-hyperthreaded cores can request "ht=core", or they can request or be routed to the queue named "core_q".

16.5.3.4 Automatic Onlining of Fixed Vnodes

When cleaning up a job, if the cgroups hook fails to kill all processes within a cgroup, it cannot destroy the job's cgroups. If that happens, it offlines the vnodes on the host to prevent the scheduler from sending new jobs to the vnodes; since resources_assigned is no longer accurate for the vnodes, the cgroups hook might reject the jobs. The cgroups hook then periodically attempts to clean up these orphaned cgroups.

When the online_offlined_nodes parameter is enabled, the hook automatically onlines the vnodes once no more orphaned cgroups exist, if the node comment confirms that the cgroups hook offlined the vnodes. When this parameter is disabled, it leaves the node offline; you can manually online vnodes again later after confirming the host is healthy.

16.5.3.5 cpuacct Subsystem

The cpuacct subsystem enables per-job CPU time accounting, through per-cgroup usage counters provided by the kernel. Advantages:

- The kernel maintains per-cgroup usage counters, so MoM doesn't have to sum the usage of each job process
- MPI libraries or applications that use `setsid` to detach processes from existing sessions do not cause inaccurate resource usage accounting, since the processes are still seen as part of the job
- A session spawning a child session that registers itself to PBS (not necessary with cgroups) does not cause some CPU time to be counted twice incorrectly at the end of the job
- Usage accounting does not rely on MoM polling, and usage accumulated after the last poll cycle is still counted. (When the cpuacct subsystem is not used, CPU time for `tm_attach`ed processes is counted only until the last MoM poll cycle)
- Short MoM polling cycles are not required for accurate accounting at the end of the job

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 579.

16.5.3.6 cpuset Subsystem

The cpuset subsystem restricts jobs to specific CPUs and optionally memory sockets allocated to them by the hook.

Advantages:

- Libraries know which set of CPUs are available for process pinning, so when a vnode is shared between multiple jobs, libraries don't try to pin more than one job to the same CPUs. For example, if you have two 4-CPU jobs and an 8-CPU vnode, they won't both try to pin themselves on CPUs 0, 1, 2, and 3.
- Strict job isolation is enforced; it is impossible for a job to steal CPU resources from another job on the same host, since the Linux scheduler will only run processes on the designated CPUs. If a job requests N CPUs and then creates N*2 processes, the job's processes will compete with each other for CPU resources, instead of disturbing other jobs.
- Since job processes are restricted to specific CPUs and not left to wander over the entire host, the default "first touch node local" memory allocation policy can minimize memory latency; for jobs that do not span NUMA nodes, pinning processes to CPUs to avoid non-local memory accesses is no longer even necessary.

Disadvantages: if you want to overcommit CPU resources, for example by setting resources_available.ncpus to 128 on a host that has 64 CPU threads, you cannot use the cpuset subsystem.

Note that there can be a problem when using the cpuset subsystem and preemption via suspend and resume. See section 16.8.1, "Interactions Between Suspend/resume and the cpuset Subsystem", on page 607.

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, "Hook and Subsystem Enablement Tuning Parameters", on page 579.

16.5.3.6.i Using Memory Fences for Job Memory Requests

You can set memory fences around jobs by setting the mem_fences parameter to *true*. When *true*, the cgroups hook sets the cpuset.mems to only the NUMA nodes assigned to the job, preventing jobs from using memory from other NUMA nodes. When *false*, cpuset.mems encompasses all NUMA nodes present on the host. This parameter is *true* by default.

Using job memory fences maximizes application performance, and mimics the behavior of the former cpuset MoM most closely.

Recommendations when using memory fences for job requests:

- Precisely match job chunk specifications to vnodes, and/or run only a combination of jobs that use `-lplace=group=vnode` for small jobs and `-lplace=excl` for larger jobs. Do not allow the scheduler to split chunks across several vnodes. If a job requests -lncpus=1:mem=2GB and the ncpus are allocated on vnode node[0] but the memory is partially allocated on vnode node[1], memory fences are not going to enforce node local memory allocations. Memory fences can cause jobs to fail if some jobs straddle vnodes and share vnodes with jobs entirely confined to one vnode. For example, if job B is allocated 1GB of memory on node node[0] and 63GB of memory on node[1], the kernel may let the job allocate 64GB on node[0] instead, causing memory allocations for jobs confined to node[0] to fail through no fault of their own.
- When erecting memory fences and using huge pages, install utilities to ensure the correct amount of huge pages are present on NUMA nodes allocated to jobs; if the huge pages still available are on the wrong nodes, jobs may fail.

Without memory fences, processes in a cpuset still have a strong preference to allocate memory on the NUMA node of the first process to access the memory, unless memory on that NUMA node is depleted, when the kernel will satisfy requests using memory on other NUMA nodes.

Disabling memory fences is the safest option to ensure that jobs do not fail because they cannot request enough memory. This can happen when another job, possibly one using more than one vnode, grabs the memory first.

The drawback of not having memory fences is that when unanticipated off-node allocations do happen, a job will not fail but will silently use remote memory and run slower; you may prefer these jobs to fail, so that you can address the root cause rather than just run jobs too slowly, especially for large parallel jobs that should never cause such off-node allocations.

If you wish to rely on first touch local node memory placement to work most of the time and would rather see jobs still run rather than fail when remote memory allocations become inevitable, disable memory fences.

Recommendations when not using memory fences for job requests:

- If you disable memory fences, you must appropriately set kernel tunables governing how the OS uses memory for caching files; see section 16.5.3.6.iii, “Memory Spreading for OS File Caching”, on page 586.
- When using large memory pages (see section 16.5.3.11, “hugetlb Subsystem”, on page 596) you must ensure that enough large pages are always available on the NUMA nodes where they are needed to ensure best performance.

16.5.3.6.ii Using Memory Fences for OS File Caching

Pages allocated by the kernel to cache files accessed by processes have their own fences, controlled by the mem_hardwall parameter. In theory, placing these on remote NUMA nodes has less deleterious effects, both because memory latency to these pages is less important, and also because a job on foreign NUMA nodes can in theory reuse these pages fairly easily, since the kernel can discard the cached contents of the file to mark the memory free again (but possibly only after writing out dirty cache to disk, which may take a while).

By default mem_hardwall is set to *false*, which allows the operating system the freedom to use memory on all nodes to cache files for any process. If you want to isolate jobs to get more repeatable performance, you can enable the mem_hardwall parameter, so that a cgroup can cache files using pages on only the NUMA nodes assigned to it.

16.5.3.6.iii Memory Spreading for OS File Caching

By default memory_spread_page is disabled for a cpuset, which means that file cache allocated for a process in the cgroup will preferentially be allocated on the NUMA node where the process that causes the file to be cached is currently running. This minimizes I/O latency to the buffer cache, but may create hotspots on certain NUMA nodes where file cache is concentrated, reducing the free memory immediately available for application memory allocations. If the kernel cannot reuse the file cache rapidly enough, the concentration of file cache may cause memory to be allocated off-node, which can slow application performance.

You can enable the memory_spread_page parameter to reduce file cache hotspots by spreading file cache allocation across the NUMA nodes that can be used by the cgroup. This may cause a slight increase in latency for accessing cached files.

To control whether pages are spread only on the NUMA nodes assigned to the job, or on the entire host, use the mem_hardwall parameter. Set this to *true* to spread pages only on the NUMA nodes assigned to the job.

16.5.3.6.iv Allowing Zero CPU Jobs

Some job submitters may want to run "weightless" jobs that consume few CPU resources; these jobs are assigned zero CPUs. The cpuset subsystem allows these jobs to run in the parent cpuset that has access to all CPUs and memories. However, these processes break down the barriers that otherwise ensure jobs are isolated from other jobs, so you may want to disable support for these by setting the allow_zero_cpus parameter in the cpuset subsystem to *false*.

If you want to allow zero-CPU jobs while ensuring that they don't take up too much of your resources, enable the cpu subsystem.

16.5.3.6.v Excluding CPUs

You can exclude CPUs so that they are not used by jobs by listing them in the exclude_cpus parameter. If the cpuset subsystem is enabled, the CPUs you specify in exclude_cpus are not assigned to jobs. Note, however, that if the cpuset subsystem is disabled, CPUs are still excluded from jobs, but only by reducing the count of CPUs available; you cannot control which CPUs are excluded.

16.5.3.6.vi cpuset Subsystem Configuration Parameters

Table 16-6: cpuset Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled	*true*	*false*	When set to *true*, the hook creates a cpuset for each job, taking into account the number of ncpus and mem resources assigned by the scheduler to this host's vnodes. When set to *false*, the kernel is free to schedule processes and allocate memory based on the system configured policies. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
exclude_cpus	*[]*	*[]*	Specifies CPU thread IDs not to be assigned to jobs Format: JSON list Default: Empty list, no CPU thread IDs excluded See section 16.5.3.6.v, "Excluding CPUs", on page 586
mem_fences	*true*	*true*	When *true*, the cgroups hook sets the cpuset.mems to only the NUMA nodes assigned to the job, preventing other NUMA nodes from satisfying memory requests from the job. When *false*, cpuset.mems encompasses all NUMA nodes present on the host. See section 16.5.3.6.i, "Using Memory Fences for Job Memory Requests", on page 585
mem_hardwall	*false*	*false*	Specifies whether kernel allocations for file caching should be restricted to the memory nodes in the cpuset. By default, all NUMA nodes can be used for caching files accessed by job processes. When set to *true*, the buffer cache for this job will be constrained to the NUMA nodes listed in cpuset.mems for the job. See section 16.5.3.6.ii, "Using Memory Fences for OS File Caching", on page 586
memory_spread_page	*false*	*false*	Specifies whether file system buffers should be spread evenly across the memory nodes allocated to the cpuset. By default, no attempt is made to spread memory pages for these buffers evenly, and buffers are placed on the same node on which the process that created them is running. See section 16.5.3.6.iii, "Memory Spreading for OS File Caching", on page 586
allow_zero_cpus		*true*	Specifies whether zero CPU jobs should be allowed to run or refused when the cpuset subsystem is enabled. See section 16.5.3.6.iv, "Allowing Zero CPU Jobs", on page 586

16.5.3.7 cpu Subsystem

The cpu subsystem is an alternative way to control which processes get access to CPU resources. It cannot isolate jobs

with the precision of the cpuset subsystem, but you can use it in some specific circumstances:

- When you use the cpuset subsystem while allowing zero CPU jobs, but want to ensure that the Linux scheduler favors the jobs requesting one or more CPUs.
- When one of the features you use in PBS does not interoperate well with cpusets, for example when using suspend/resume when both the high-priority workload and the low-priority workload might have jobs that share vnodes. See section 16.8.1, “Interactions Between Suspend/resume and the cpuset Subsystem”, on page 607.
- When you want to overcommit CPU resources (impossible if the CPUs are assigned to jobs via the cpuset subsystem) but still want jobs to get access to CPU resources according to the ncpus requested.

The cpu subsystem implements two different mechanisms:

- Linux scheduler fair sharing, where different shares are assigned to cgroups, according to requested ncpus. If there are CPU resource conflicts, the Linux scheduler favors cgroups that have used less than their allotted share
- Hard quotas that can be enforced if too much CPU usage is detected.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 579.

16.5.3.7.i cpu Subsystem Caveats

- The cpu subsystem controls are often imprecise when hyperthreading is enabled, since the Linux kernel CFS scheduler sees each CPU thread as 100% of a CPU, regardless of how slow or fast it runs (which depends on usage of the other threads of the core). When ncpus_are_cores is enabled, quotas are multiplied by the number of threads per core, which can be misleading.
- There is no strict isolation between jobs; the Linux scheduler enforces a quota only on the set of processes, not individual processes. Linux fair sharing is not as efficient as cpusets in isolating jobs from rogue jobs. The cpu subsystem will slow down the rogue job, but since throttling may be uneven, a rogue process may still interfere with other jobs; if the hapless victims belong to a parallel application, that whole application may be affected, including its processes on CPUs where there is no conflict, since these will be forced to wait for the application process that was slowed down.

16.5.3.7.ii cpu Subsystem Configuration Parameters

Table 16-7: cpu Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled		*false*	When set to *true*, the hook creates a cpu subsystem directory for each job, and assigns a number of shares and optionally a quota based on ncpus. When set to *false*, no per-job cgroup is created for this subsystem. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
cfs_period_us		*100000*	Time in microseconds between periodic checks by the Linux scheduler to compute usage and check quotas. Lowering this makes computation more precise and throttles rogues faster, but uses more OS scheduler overhead, with less CPU resources left for jobs.
cfs_quota_fudge_factor		*1.03*	Sets quota slightly above theoretically valid value. Setting a CPU usage quota to 100% of a CPU still throttles applications, due to rounding errors. The default is appropriate for the default cfs_period_us, but should be raised if cfs_period_us is lowered.
enforce_per_period_quotas		*false*	Specifies whether hard quotas are set. When *false*, only shares are set, and a job can use more CPU resources than it requested provided other jobs leave CPU resources idle. When *true*, hard CPU usage quotas are set
zero_cpus_shares_fraction		*0.002*	Specifies fraction of shares allotted for a CPU to a "zero-CPU" job. The default, *0.002*, is the minimum allowed by the kernel, and ensures that such a job gets CPU resources only if they are left idle by other jobs.
zero_cpus_quota_fraction		*0.2*	Specifies fraction of a CPU allotted to a "zero-CPU" job before it is throttled by its hard quota. Default fraction: *one-fifth of a CPU*

16.5.3.8 devices Subsystem

The devices subsystem is used to grant or restrict access to devices on the system, restricting the job to use specific devices, including GPU and Intel Xeon Phi ("MIC") devices assigned to the job. If MICs and/or GPUs are available in the complex and the devices subsystem is enabled, the PBS cgroups hook creates the nmics and/or ngpus resources if they are not already present.

Since detecting the GPU and/or MIC resources assigned to existing jobs relies on the devices cgroups for these jobs, enable this subsystem if you want the cgroups hook to manage GPU and/or MIC assignments.

For examples of how to use this subsystem, see section 16.5.5, “Managing GPUs or Xeon Phi via Cgroups”, on page 600.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, “Hook and Subsystem Enablement Tuning Parameters”, on page 579.

16.5.3.8.i Allowing Access to Devices

The allow parameter specifies how access to devices will be controlled. The list consists of entries in one of the following formats:

- A single string entry, used verbatim. For example:
  - "b *:* rwm" allows full access (read, write, and `mknod`) to all block devices
  - "c *:* rwm" allows full access to all character devices
- A list containing two strings. For example:
  - ["mic/scif","rwm"] looks for the major and minor number of the mic/scif device and allows full access.
  - If `ls /dev/mic` reported

    ```
    "crw-rw-rw- 1 root root 244, 1 Mar 30 14:50 scif"
    ```

    then the line added to the allow file looks like

    ```
    "c 244:1rwm"
    ```
- A list containing three strings. For example:
  - ["nvidiactl","rwm", "*"] looks for the major number of the nvidiactl device and allows full access to all devices with that major number.
  - If `ls /dev/nvidiactl` reported

    ```
    "crw-rw-rw- 1 root root 284, 1 Mar 30 14:50 nvidiactl"
    ```

    then the line added to the allow file looks like

    ```
    "c 284:* rwm"
    ```

16.5.3.8.ii devices Subsystem Configuration Parameters

Table 16-8: devices Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled	*false*	*false*	When set to *true*, the hook configures the devices subsystem based on the number of nmics and ngpus requested by the job. Refer to the allow parameter for additional information. When set to *false*, no cgroup is created for the devices subsystem.
allow	*[* *"b *:* rwm",* *"c *:* rwm"* *]*	*[]*	Specifies how access to devices will be controlled. The list consists of entries in one of the following formats: • A single string entry, used verbatim. Example: "b *:* rwm" • A list containing two strings. Example: ["mic/scif","rwm"] • A list containing three strings. Example: ["nvidiactl","rwm", "*"] See section 16.5.3.8.i, "Allowing Access to Devices", on page 590

16.5.3.9 memory Subsystem

The memory subsystem allows you to monitor and limit the amount of physical memory used by all of the processes of a job on a host.

By default this subsystem is enabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, "Hook and Subsystem Enablement Tuning Parameters", on page 579.

Advantages to enabling the memory subsystem:

- With cgroups, jobs and operating systems are protected from any attempt by a job to use too much memory. Without cgroups, enforcing memory limits relies on monitoring and after-the-fact interventions to kill processes; MoM may not be able to act in time to protect the health of the host from being compromised through excessive memory usage.
- Accurate monitoring and records of memory usage. Without cgroups, memory usage such as resources_used.mem, which is supposed to capture peak usage, relies instead on periodic polling, which can miss the high-water mark.
- The accuracy of memory usage monitoring is unaffected by processes that leave the Linux sessions registered as part of the job; without cgroups, this behavior breaks memory usage accounting when using some precompiled MPI libraries.
- Because memory usage reporting does not depend on polling, MoM can be configured to poll less often, which reduces the load on the PBS datastore.
- Jobs are prevented from rampantly filling host memory with kernel-allocated file cache. Instead, because kernel-allocated file cache for job file access is considered job memory, when the job hits its memory limit, job memory requests are fulfilled by reclaiming file cache allocated by the job earlier, or even by temporarily moving some of the job to swap until this can be done. If necessary, job processes will hang until memory can be allocated without crossing the memory usage limit.
- Recommended: you can reserve enough memory for the operating system and the file cache required for OS services to run well. If you do this, jobs are prevented from starving operating system services of memory resources.

16.5.3.9.i Reserving Memory

If you want to reserve memory so that it is not assigned to jobs, you can set a percent of physical memory using reserve_percent, and add a fixed amount to that using reserve_amount. The reserve_amount parameter sets a specific amount of available physical memory that is not to be assigned to jobs.

Reserving memory decreases the amount of resources_available.mem that MoM advertises to the server as being available for each vnode, as well as the amount of memory the cgroups hook is willing to assign to jobs.

For most HPC compute nodes with a minimum of 32GB, we recommend at least 2GB of memory for the operating system. Since there is no minimum amount of memory specified for a MoM host, the defaults are conservative, but the default value is too small for most clusters.

16.5.3.9.ii Effect of Cgroups Hook on the mem Resource

The cgroups hook changes how much memory must be requested for job I/O and accounted in resources_used.mem.

Without the cgroups hook, file content cached in memory need not be included in a job's memory request, and is not accounted for in resources_used.mem. With the cgroups hook, reported memory usage reported includes memory for cached files accessed by the job.

Jobs requesting memory can use that amount both for physical memory and for caching. For example, when using the cgroups hook, a job that requests 20GB and uses 16GB but reads a 50GB file can hold only 4GB of the file in cache at a time. So if a job requires 32GB of application memory but also requires 5GB of private file cache to perform adequately, then it needs to request 37GB.

Memory is accounted accurately with cgroups. For jobs with multiple processes all accessing the same memory, without the cgroups hook the amount of memory reported as used is multiplied by the number of processes, but with the cgroups hook, the memory is only counted once. However, memory usage includes file cache placed into memory by job I/O, but not file cache merely accessed by the job but placed into memory earlier by unrelated processes.

It may be necessary to use other tools to determine application usage not involving file cache instead of resources_used.mem.

Applications using direct I/O to filesystems, meaning they bypass the buffer cache, are unaffected; these jobs do not see a change in resources_used.mem with and without cgroups.

16.5.3.9.iii Assigning a Default Amount of Memory to Jobs

The default parameter is the amount of memory assigned to the job if it doesn't request any memory. Because the scheduler does not know about this allocation, do not make this overly large, otherwise the cgroups hook may reject jobs because there isn't enough available memory. Instead, set large defaults for job memory resource requests (Resource_List.mem, default_chunk.mem, etc.) via a queuejob hook or defaults at the server or queue.

This value is not communicated to the server or scheduler; setting this value has no effect on the job's resource request.

16.5.3.9.iv Managing Use of Swap by Jobs

If your execution nodes have no swap or if you do not want your jobs to be able to swap to disk, set the swappiness parameter to *0* or *false*. In this case you can disable the memsw subsystem.

When this is zero or false, if a job cannot have its memory requests satisfied by claiming free physical memory or reclaiming memory from the page cache, then the Out of Memory killer will step in and kill job processes instead of allowing swap usage.

If you want your jobs to swap only when it is necessary to avoid job failure, and not to proactively move infrequently-used job pages to swap, set the swappiness parameter to *1* or *true*.

You can use the membership tuning tools described in section 16.5.2, “Enabling and Tuning Hook According to Host and/or Vnode Type”, on page 577 to specify swappiness for hosts or vnodes.

Not recommended for HPC workloads: if you set this parameter to larger values such as *60*, the kernel will move infrequently-accessed user pages in order to free memory for file caching. We recommend using larger values only if you have at least as much swap as there is physical memory not assigned to jobs in the memsw subsystem. You can either not enable the memsw subsystem, or enable it but ensure there is enough swap and set reserve_amount in the memsw section to at least the physical memory of the host.

16.5.3.9.v Setting Memory Soft Limits

You can limit whether PBS imposes hard memory limits on job processes.

If you set the soft_limit parameter to *false*, PBS uses hard memory limits which prevent the processes from ever exceeding their requested memory usage. If a job accesses more memory than it has requested, some of the job's memory is moved to swap or the job is killed.

If you set this parameter to *true*, PBS uses soft memory limits; the memory requested by the job is set as a hint to the kernel as to what usage is expected, but no hard limit is enforced; when the kernel experiences memory shortage it uses the limits to select the cgroups from which memory is moved to swap.

16.5.3.9.vi Setting Aside Memory for Kernel Drivers

Some kernel drivers (notably, GPFS or Lustre filesystem) may lower the memory available on the host or NUMA nodes by a small number of MB, typically 32MB or 64MB, after MoM has started.

This may reduce the amount of memory actually available at a vnode to less than the amount the scheduler thinks is available, and if jobs requiring the full amount are scheduled on that vnode, the hook will reject those jobs.

To compensate, you can hide some memory from the server. You set vnode_hidden_mb to for example "32" or "64".

The amount listed in the vnode_hidden_mb parameter lowers the memory advertised to the server when the exechost_startup hook is run, but not the memory that the cgroups hook itself is willing to assign to jobs.

16.5.3.9.vii memory Subsystem Configuration Parameters

Table 16-9: memory Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled	*true*	*false*	Boolean. When *True*, enables the memory subsystem. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
default	*"256MB"*	*"0MB"*	Amount of memory assigned to the job if it doesn't request any memory. Recommendation: keep this small. See section 16.5.3.9.iii, "Assigning a Default Amount of Memory to Jobs", on page 592.
swappiness		*1*	Sets the memory.swappiness value for the cgroup. When set to *false* or *0*, jobs are not allowed to use swap. When set to *true* or *1*, kernel is allowed to only swap if absolutely required. When set to larger values, kernel is allowed to proactively swap. Not recommended for HPC workloads. See section 16.5.3.9.iv, "Managing Use of Swap by Jobs", on page 592.
reserve_amount	*"64MB"*	*"0MB"*	A specific amount of available physical memory that is not to be assigned to jobs. The actual amount of memory reserved is reserve_amount plus the amount specified by reserve_percent. See section 16.5.3.9.i, "Reserving Memory", on page 592.
reserve_percent	*"0"*	*"0"*	The percentage of available physical memory that is not to be assigned to jobs. The actual amount of memory reserved is reserve_amount plus the amount specified by reserve_percent. See section 16.5.3.9.i, "Reserving Memory", on page 592.
soft_limit	*false*	*false*	When set to *false*, PBS uses hard memory limits which prevent the processes from ever exceeding their requested memory usage. When set to *true*, PBS uses soft memory limits; the memory requested by the job is set as a hint to the kernel as to what usage is expected, but no hard limit is enforced. See section 16.5.3.9.v, "Setting Memory Soft Limits", on page 593.
vnode_hidden_mb		*"1"*	Amount of memory per vnode to hide from the server but keep available to the hook, to compensate for memory reduction by kernel drivers. See section 16.5.3.9.vi, "Setting Aside Memory for Kernel Drivers", on page 593.

16.5.3.10 memsw Subsystem

The memsw subsystem allows you to monitor and limit swap used by all of the job processes on a host. This subsystem restricts a job to a specific amount of memory + swap. If a job exceeds the limit, processes associated with that job are killed.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, "Hook and Subsystem Enablement Tuning Parameters", on page 579.

16.5.3.10.i Reserving Swap

If you want to reserve swap so that it is not assigned to jobs, you can set a percent of swap using reserve_percent, and add a fixed amount to that using reserve_amount. The reserve_amount parameter sets a specific amount of swap that is not to be assigned to jobs.

Reserving swap decreases the amount of resources_available.vmem that MoM advertises to the server as being available for each vnode, as well as the amount of vmem the cgroups hook is willing to assign to jobs.

16.5.3.10.ii memsw Subsystem Configuration Parameters

Table 16-10: memsw Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
enabled	*false*		Boolean. When *true*, enables the memsw subsystem. When *false*, the subsystem is disabled, and jobs reaching their memory limit are allowed to use swap in an unrestrained fashion (unless memory.swappiness was set to *0*) until resources are depleted. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
default	*"256MB"*	*0*	Specifies the amount of memory + swap assigned to the job if it doesn't request any memory. This value is not communicated to the server or scheduler; setting this value has no effect on the job's resource request.
reserve_amount	*"64MB"*	*0MB*	An amount of available swap that is not to be assigned to jobs. The amount reserved is the amount determined by reserve_percent plus reserve_amount. See section 16.5.3.10.i, "Reserving Swap", on page 595.
reserve_percent	*"0"*	*0*	Percentage of available swap that is not to be assigned to jobs. The amount reserved is the amount determined by reserve_percent plus reserve_amount. See section 16.5.3.10.i, "Reserving Swap", on page 595.

16.5.3.10.iii Scheduling on the vmem Resource

To allow the scheduler to take resources_available.vmem and resources_assigned.vmem on nodes and vmem requested by jobs into account when deciding where and when to schedule jobs, list "vmem" on the "resources:" line in $PBS_HOME/sched_priv/sched_config.

16.5.3.10.iv Effect of memsw Subsystem on the vmem Resource

Enabling the memsw subsystem changes how much vmem must be requested and how vmem is accounted in resources_used.vmem. The value of resources_available.vmem at a host or vnode reflects the disk swap that can be assigned to jobs. If there is more than one vnode per NUMA node, swap resources are split equally over the number of NUMA nodes reported.

If this subsystem is enabled, requesting the vmem resource sets a limit for the job's memory plus swap usage. For example, a job requesting -lselect=1:ncpus=16:mem=8GB:vmem=64GB is allowed to use 8GB in physical memory plus 56GB of memory resident in swap.

The value of resources_used.vmem reflects the job's memory plus swap usage across all nodes.

The way physical memory is accounted changes with the cgroups hook; see section 16.5.3.9.ii, "Effect of Cgroups Hook on the mem Resource", on page 592.

16.5.3.10.v Caveat for Enabling memsw Subsystem

The cgroups hook prevents a job from using more physical memory than it has requested, which means that a swap shortage cannot always be made up with physical memory.

If the memory and memsw subsystems are enabled, a job can fall into a trap where the scheduler thinks there is enough swap at a host or vnode, but there is not. The reason is that there is no separate resource for swap; resources_available.vmem is the sum of physical memory plus swap.

For example, suppose a node has 64GB of physical memory and 2GB of swap. With no memory reservations in the cgroups hook configuration file, resources_available.mem is approximately 64GB and resources_available.vmem is approximately 66GB.

If you submit a job with -lselect=1:mem=2GB:vmem=10GB, the scheduler sees enough available vmem and enough available mem on the node. But when the job runs, if it does indeed use 10GB of memory, it will fail. The memory cgroup will limit the job memory resident in physical memory to 2GB, but there is only 2GB of swap, so even though there is enough memory plus swap, the job will not be able to use 8GB of swap to make up the remainder of the 10GB.

The hook will try to catch one common case: if the explicitly requested vmem is larger than the requested mem, then nodes without any swap will refuse to run the job; the cgroups hook will reject the request to run the job. Administrators and job submitters must ensure that such jobs do not land on nodes without swap, for example by using resources to tag nodes accordingly and setting the proper requests based on those tags.

To avoid the problem of running out of swap when enabling both memory and memsw:

- Set reserve_amount in the memsw section to a value that is equal to resources_available.mem for the host; this makes any job specification safe, but requires that swap resources are larger than the physical memory on the host.
- Do not use jobs that leave a lot of physical memory on the host unrequested if Resource_List.vmem > Resource_List.mem: only use jobs like that when the physical memory cannot fit the job, to use swap as extra memory. This will drastically reduce the amount of swap you need to reserve as not visible to jobs in the configuration file.
- Write an exechost_startup hook, or a script running on the server and using `qmgr`, that initializes a resources_available.swap based on the difference between resources_available.vmem and resources_available.mem. Use a queuejob hook that sets requested swap in all chunks requested to the value of (vmem requested - mem requested), if applicable, and make the scheduler honor the swap resource by adding it to the `resources:` line in `sched_config`. This removes the need to reserve a lot of swap space as invisible to PBS.

16.5.3.11 hugetlb Subsystem

The hugetlb subsystem lets you manage the amount of huge page memory used in a cgroup.

While this subsystem imposes limits on the huge pages that can be used by a job, it does not create huge pages on the different NUMA nodes. That must be done separately, so that the cgroups hook exechost_startup portion can report huge pages available on the different nodes.

You can control how many huge pages you want on a set of nodes dynamically:

```
numactl -m <node list> echo X >/proc/sys/vm/nr_hugepages_mempolicy.
```

You can see the number of huge pages currently available here:

```
/sys/devices/system/node/node[0-9]*/hugepages/
```

The free_hugepages and surplus_hugepages pseudofiles are read-only.

Writing to nr_hugepages tells the system to adjust the number of persistent huge pages on the NUMA node, but if existing memory that is free is too fragmented, it may have to repurpose buffer cache pages for it (which may take time if the pages are dirty and need to be pushed to the filesystem) or even move existing used memory to swap.

By default this subsystem is disabled. You can tune the parameters that modify whether this subsystem is enabled; the parameters, but not their defaults, are listed in section 16.5.2.2.vi, "Hook and Subsystem Enablement Tuning Parameters", on page 579.

16.5.3.11.i Reserving Huge Page Memory

If you want to reserve huge page memory (hpmem) so that it is not assigned to jobs, you can set a percent of huge page memory using reserve_percent, and add a fixed amount to that using reserve_amount. The reserve_amount parameter sets a specific amount of huge page memory that is not to be assigned to jobs.

Reserving huge page memory decreases the amount that MoM advertises to the server as being available for each node, as well as the amount that the cgroups hook is willing to assign to jobs.

16.5.3.11.ii Caveat for hugetlb Subsystem

When a job spans more than one vnode, it may split its allocation of huge page memory across NUMA nodes differently from how PBS assigned the memory. This can lead to other jobs on those NUMA nodes not having enough huge page memory.

You can:

- Use utilities to check whether huge pages are available on the correct NUMA nodes just before you launch applications
- Use a memory-fences-safe workload
- Disable memory fences to allow huge pages to be allocated off-node

16.5.3.11.iii hugetlb Subsystem Configuration Parameters

Table 16-11: hugetlb Subsystem Configuration Parameters

Parameter Name	Default Value: Config File	Default Value: Hook	Description
default	*0MB*		The amount of huge page memory assigned to the cgroup when the job does not request hpmem.
enabled	*false*		When set to *true*, the hook registers a limit that restricts the amount of hugepage memory processes may access. When set to *false*, no limit is registered. See section 16.5.2, "Enabling and Tuning Hook According to Host and/or Vnode Type", on page 577.
reserve_amount	*0MB*		An amount of available huge page memory (hpmem) that is not to be assigned to jobs. The amount reserved is the amount determined by reserve_percent plus reserve_amount. See section 16.5.3.11.i, "Reserving Huge Page Memory", on page 597.
reserve_percent	*0*		The percentage of available huge page memory (hpmem) that is not to be assigned to jobs. The amount reserved is the amount determined by reserve_percent plus reserve_amount. See section 16.5.3.11.i, "Reserving Huge Page Memory", on page 597

16.5.3.12 Sample Cgroups Hook Configuration File

Here we show a sample cgroups hook configuration file similar to the default configuration file:

```
{
    "enabled"                : "vntype not in: no_cgroups",
    "cgroup_prefix"          : "pbs_jobs",
    "periodic_resc_update"   : true,
    "vnode_per_numa_node":   : false,
    "online_offlined_nodes"  : true,
    "use_hyperthreads"       : true,
    "ncpus_are_cores"        : true,
    "cgroup":{
        "cpuacct":{
            "enabled"        : true
        },
        "cpuset":{
            "enabled"        : true,
            "exclude_cpus"   : [0,8],
            "mem_fences"     : true,
```

```
            "mem_hardwall"     : false,
            "memory_spread_page"   : true
        },
        "devices":{
            "enabled"          : false,
            "allow":[
                "b*:*rwm",
                "c*:*rwm"
            ]
        }
        "memory":{
            "enabled"          : true,
            "soft_limit"       : false,
            "default"          : "256MB",
            "reserve_percent"  : "0",
            "reserve_amount"   : "3GB"
        },
        "memsw":{
            "enabled"          : false,
            "default"          : "256MB",
            "reserve_percent"  : "0",
             "reserve_amount"  : "1GB"
        },
        "hugetlb":{
            "enabled"          : false,
            "default"          : "0MB",
            "reserve_percent"  : "0",
            "reserve_amount"   : "0MB"
        }
    }
}
```

16.5.4 Finish Up

16.5.4.1 Enable cgroups hook

The cgroups hook and its default configuration file are already imported. You must enable the cgroups hook as root:

1. 1. Log in as root
2. 2. Enable the cgroups hook on the server host:

```
Qmgr: set hook pbs_cgroups enabled = true
```

16.5.4.2 HUP or Restart MoM

HUP or restart each MoM:

```
kill -HUP <MoM PID>
```

or

```
<path to PBS start/stop script>/pbs restart
```

or

```
systemctl restart pbs
```

16.5.4.3 Enable Use of Resources by the Scheduler

Modify the resources: line in <sched_priv directory>/sched_config:

- The nmics and ngpus resources are automatically created, but you have to add them to the resources: line in <sched_priv directory>/sched_config.
- If you have configured huge page memory, and it is enabled in the cgroups hook, PBS creates the hpmem resource, but you need to add it to the resources: line in <sched_priv directory>/sched_config.
- If you have configured the memsw subsystem, add "vmem" to the resources: line in <sched_priv directory>/sched_config.

Set resource flags:

- If the cgroups hook creates the nmics and ngpus resources, you may need to set their flags.You also need to set the flags for the vmem and hpmem resources. Set the flags for the nmics, ngpus, vmem, and hpmem resources to "*nh*":

```
Qmgr: set resource nmics,ngpus,vmem,hpmem flag=nh
```

16.5.5 Managing GPUs or Xeon Phi via Cgroups

Integration with Linux cgroups allows PBS to automatically detect and configure GPUs and Xeon Phi processors. Since some of the details are different, we will proceed by describing the case with GPUs.

16.5.5.1 Managing GPUs via Cgroups

PBS can restrict jobs to specific allocated GPUs. If you set vnode_per_numa_node to *true* in the cgroups hook configuration file, PBS takes advantage of topology and associates GPUs with the closest memory and CPUs in the system.

If GPUs are available in the complex and the devices subsystem is enabled, the PBS cgroups hook creates the ngpus resource if it is not already present, and discovers and sets values for it.

16.5.5.1.i Configuration Steps

Here we summarize the configuration steps that allow the cgroups hook to manage your GPUs.

- Modify the resources: line in <sched_priv directory>/sched_config.

The ngpus resource is automatically created, but you have to add it to the resources: line in <sched_priv directory>/sched_config.

- Set the flags for the ngpus resource to "*nh*":

  ```
  Qmgr: set resource ngpus flag=nh
  ```

- Export the installed PBS cgroups hook configuration file:

  ```
  qmgr -c "export hook pbs_cgroups application/x-config default" >pbs_cgroups.json
  ```

- Edit this file and change its parameters:
  - If nvidia-smi is someplace other than /usr/bin/nvidia-smi, add the nvidia-smi global parameter with the absolute path to nvidia-smi. Be sure to add the comma to the end of the line. For example:

    ```
    "nvidia-smi"    : "/usr/bin/nvidia/nvidia-smi",
    ```

  - Add the "nvidiactl" value to the allow parameter of the devices subsystem section. Be sure to add the comma to the end of the previous line:

    ```
    "c : rwm",
    ["nvidiactl", "rwm", "*"]
    ```

  - Make sure that the vnode_per_numa_node global parameter is set to *true*
  - Enable the devices subsystem (it is disabled by default); see section 16.5.3.8, “devices Subsystem”, on page 589

    ```
    "enabled" : true,
    ```

  - Enable the cpuset subsystem (it is enabled by default); see section 16.5.3.6, “cpuset Subsystem”, on page 585

    ```
    "enabled" : true,
    ```

- Read the configuration file back in:

  ```
  qmgr -c "import hook pbs_cgroups application/x-config default pbs_cgroups.json"
  ```

- HUP or restart each MoM:

  ```
  kill -HUP <MoM PID>
  ```

  or

  ```
  <path to PBS start/stop script>/pbs restart
  ```

  or

  ```
  systemctl restart pbs
  ```

If device isolation is not enabled for GPUs, the hook assigns devices to jobs, and sets the CUDA_VISIBLE_DEVICES environment variable for processes created as children of MoM according to the devices assigned to the job on the host on which the process runs.

16.5.5.1.ii Isolating NVIDIA GPUs

For NVIDIA GPU isolation to work, you need to restrict access to only those devices assigned to the job. In the "allow" subsection of the devices section of the configuration file, do not use the broad "c *:* rwm".

Make sure that the "allow" section excludes read and write access for the 195 major number ("c 195:* m"), which is what all NVIDIA devices use. Preserve mknod ("m") access, since other software such as the container hook may need "m" access.

You also need to include ["nvidia-uvm", "rwm"], since it is part of how the driver determines isolation, and possibly other global NVIDIA devices used by NVIDIA tools. You may also have to add other devices, such as those required for MPI library access to Infiniband devices.

The cgroups hook assigns the correct NVIDIA GPUs when a job requests ngpus.

Here is an example of the devices section of the cgroups hook configuration file, configured to allow NVIDIA GPU isolation:

```
"devices":{
    "enabled"      : true,
        "allow"    :
        [
            "b *:* m",
            "c *:* m",
            "c 195:* m",
            "c 136:* rwm",
            ["infiniband/rdma_cm", "rwm"],
            ["fuse", "rwm"],
            ["net/tun", "rwm"],
            ["tty", "rwm"],
            ["ptmx", "rwm"],
            ["console", "rwm"],
            ["null", "rwm"],
            ["zero", "rwm"],
            ["full", "rwm"],
            ["random", "rwm"],
            ["urandom", "rwm"],
            ["cpu/0/cpuid", "rwm", "*"],
            ["nvidia-modeset", "rwm"],
            ["nvidia-uvm", "rwm"],
            ["nvidia-uvm-tools", "rwm"],
            ["nvidiactl", "rwm"]
        ]
},
```

Here we detail the 136 device:

```
root@myhost:~# ls -l /dev/pts
crw------- 1 altair tty  136, 0 Jul 22 20:38 0
c--------- 1 root   root   5, 2 Jul 22 19:18 ptmx
```

Here we detail the 195 devices:

```
root@myhost:~# ls -l /dev/nvidia*
crw-rw-rw- 1 root root 195,   0 Jul 22 19:18 /dev/nvidia0
crw-rw-rw- 1 root root 195,   1 Jul 22 19:18 /dev/nvidia1
crw-rw-rw- 1 root root 195,   2 Jul 22 19:18 /dev/nvidia2
crw-rw-rw- 1 root root 195,   3 Jul 22 19:18 /dev/nvidia3
crw-rw-rw- 1 root root 195, 255 Jul 22 19:18 /dev/nvidiactl
crw-rw-rw- 1 root root 195, 254 Jul 22 19:35 /dev/nvidia-modeset
crw-rw-rw- 1 root root 238,   0 Jul 22 19:35 /dev/nvidia-uvm
crw-rw-rw- 1 root root 238,   1 Jul 22 19:35 /dev/nvidia-uvm-tools
```

16.5.5.2 Environment Variables for CUDA and Xeon Phi

When you use Xeon Phi co-processors, PBS sets the OFFLOAD_DEVICES environment variable during job initialization, for each process that is a child of MoM.

When you use CUDA devices, PBS sets the CUDA_VISIBLE_DEVICES environment variable for each process that is a child of MoM.

16.5.5.2.i Using CUDA_VISIBLE_DEVICES with Multihost Jobs

On each host, the correct value for CUDA_VISIBLE_DEVICES references unique identifiers for the GPUs on that host, which means that the value of CUDA_VISIBLE_DEVICES is different on each host of a multihost job. As a result, you cannot just propagate the value of CUDA_VISIBLE_DEVICES visible in the job script to all processes of a multihost job; you need to get the correct value for each host.

PBS sets the correct value for the CUDA_VISIBLE_DEVICES environment variable for the job script on the primary execution host of the job.

For job processes on remote hosts, the CUDA_VISIBLE_DEVICES environment variable is also correctly set if the processes are spawned as children of MoM through the Task Management API (the TM API). To spawn processes through the TM API, you can use `pbs_tmrsh`, or an MPI library that was compiled with TM API support, or an MPI library that was configured to use `pbs_tmrsh` as the remote process launcher; see Chapter 10, "Using MPI with PBS", on page 451.

For other jobs that use an MPI library that does not spawn processes through the TM API (for example HPE MPI, for which remote processes are spawned by array services) or that use other mechanisms such as `ssh`, processes on sister hosts are not children of MoM, and MoM is unable to set the CUDA_VISIBLE_DEVICES environment variable directly for these processes. Processes can use `pbs_attach` to join the job, but `pbs_attach` cannot set environment variables for existing processes.

For these non-MoM-child processes on sister hosts it is necessary to fetch the correct value for the CUDA_VISIBLE_DEVICES environment variable from a file PBS wrote for the job. The location of that file depends on PBS_HOME (or PBS_MOM_HOME if it is set) and the job ID. In the rest of this section we describe how to find the value for CUDA_VISIBLE_DEVICES using PBS_HOME. See examples $PBS_HOME is different and PBS_MOM_HOME is not defined and $PBS_HOME is different and PBS_MOM_HOME is defined.

If PBS_HOME/PBS_MOM_HOME is the same on all hosts for the job, the file that contains the CUDA_VISIBLE_DEVICES is in the same location on each host, and you can pass the location of the file to processes on sister hosts via an environment variable. The PBS_NODEFILE environment variable is set to the path to the file containing the GPU identifiers for CUDA_VISIBLE_DEVICES; see the example $PBS_HOME is the same everywhere.

If PBS_HOME may be different on the different hosts:

- Pass the job ID to the sister host(s):
  - Transmit the job ID from the primary execution host to the sister(s):

    On the primary execution host, make sure that `/etc/ssh/ssh_config` contains "SendEnv PBS_JOBID".

    On the sister host(s):
    - Make sure that `/etc/ssh/sshd_config` contains "AcceptEnv PBS_JOBID".
    - Restart `sshd` if necessary.
- On the sister host, look up PBS_HOME in `/etc/pbs.conf` (in a shell process, you can source `/etc/pbs.conf` directly).

Once you know PBS_HOME and PBS_JOBID, you know the name of the file that contains the CUDA_VISIBLE_DEVICES assignment. The file is located at `$PBS_HOME/aux/$PBS_JOBID.env` .

Example 16-3: `$PBS_HOME` is different and PBS_MOM_HOME is not defined

Look in or source `/etc/pbs.conf`. `$PBS_HOME/aux/$PBS_JOBID.env` is the file to read.

If $PBS_HOME is /var/spool/node2, and $PBS_JOBID is 332.tc72:

```
echo $PBS_HOME/aux/$PBS_JOBID.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_HOME/aux/$PBS_JOBID.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

Example 16-4: $PBS_HOME is different and PBS_MOM_HOME is defined

$PBS_MOM_HOME/aux/$PBS_JOBID.env is the file to read.

If $PBS_MOM_HOME is /var/spool/node2, and $PBS_JOBID is 332.tc72:

```
echo $PBS_MOM_HOME/aux/$PBS_JOBID.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_MOM_HOME/aux/$PBS_JOBID.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

Example 16-5: $PBS_HOME is the same everywhere

$PBS_NODEFILE.env is the file to read.

If $PBS_HOME is /var/spool/node2, and $PBS_JOBID is 332.tc72:

```
echo $PBS_NODEFILE.env
/var/spool/node2/aux/332.tc72.env
cat $PBS_NODEFILE.env
CUDA_VISIBLE_DEVICES=GPU-232cc436-c5b4-6bd9-c5bc-6820334123d7
CUDA_DEVICE_ORDER=PCI_BUS_ID
```

16.6 Configuring MPI for Cgroups

In order to capture job processes and put their PIDs in cgroups, PBS needs the MPI to tell it about those processes. An MPI that is integrated with PBS does this. You need to make sure that your MPI is integrated with PBS. If you are already using an MPI that is integrated with PBS, you do not need to perform this step. OpenMPI, MVAPICH2, and MPICH behave well if they have been compiled with support for the TM API and linked with the PBS libraries. Intel MPI also behaves well if it has been integrated with PBS.

However, if your MPI uses `ssh` and is not integrated with PBS, you can use `pbs_attach` to capture processes started with `ssh`. In section 10.1, “Integration with MPI”, on page 451, we describe integrating MPIs with PBS. If your MPI is not integrated with PBS, cgroups cannot help you manage spawned processes.

If your MPI is not integrated with PBS, you might notice that jobs are running significantly slower, or jobs are crashing with errors such as "Unable to set CPU", or "Unable to join process"; the MPI may be trying to pin all processes to CPU 0 or crashing.

Wrapping `ssh` is sufficient for all precompiled MPIs to work.

16.6.1 Steps to Integrate MPI with PBS via ssh

The following is a helpful example of integrating MPI with PBS via ssh:

- On each host in the PBS complex, edit /etc/ssh/ssh_config:
  - Add the following as the last SendEnv line, after the other SendEnv lines:

    SendEnv PBS_JOBID
- On each execution host in the PBS complex, edit /etc/ssh/sshd_config:
  - Add the following as the last AcceptEnv line, after the other AcceptEnv lines:

    AcceptEnv PBS_JOBID
- On each host in the PBS complex, restart sshd:

  /etc/init.d/sshd restart
- On each host in the PBS complex, edit /etc/ssh/sshrc to include the following lines:

```
#!/bin/sh
if read proto cookie && [ -n "$DISPLAY" ]; then
        if [ `echo $DISPLAY | cut -c1-10` = 'localhost:' ]; then
                # X11UseLocalhost=yes
                echo add unix:`echo $DISPLAY |
                    cut -c11-` $proto $cookie
        else
                # X11UseLocalhost=no
                echo add $DISPLAY $proto $cookie
        fi | xauth -q -
fi
string=$*
# Make sure the following points to your $PBS_EXEC/bin
pbs_bin="/opt/pbs/bin"
attach_cmd="pbs_attach"
#echo "PBS_JOBID: $PBS_JOBID"
#echo "$*"
if [ -n "$PBS_JOBID" ]; then
    # Check to see whether the command is already calling pbs_attach
    if [ "${string/$attach_cmd}" = "$string" ] ; then
        echo "Attaching $PPID to $PBS_JOBID"
        $pbs_bin/$attach_cmd -j $PBS_JOBID -p $PPID 2> /dev/null
        exit 0
    fi
fi
```

- Test to ensure that it works as expected and PBS can capture PIDs:
  a. Make sure cgroups are enabled
  b. Submit an interactive PBS job
  c. ssh into a host belonging to the job and verify that the job process PID was added to the tasks file for the desired subsystem, e.g. cpuacct/pbspro/<job ID>/tasks

16.7 Managing Jobs with Cgroups

16.7.1 Requesting Memory

The default amount of memory assigned by cgroups to jobs that do not request it is 256MB. If this value does not work for your site, either change the default value if you need to only slightly more, or assign a default value using a queuejob hook or a server default. See section 16.5.3.9.iii, "Assigning a Default Amount of Memory to Jobs", on page 592.

16.7.2 Limit Enforcement

When a job is killed due to hitting a cgroup limit, you will see something like the following in the job's `stdout`:

```
mpirun noticed that process rank 0 with PID 115249 on node node0042 exited on signal 9 (Killed).
```

The hook will also attempt to find OOM killer messages in the kernel `dmesg` buffer. If it finds them it prints more specific errors in the MoM log, and in the job's `stderr`, if the cgroup limit violation occurs on the first node.

The messages will contain either "Cgroup memory limit exceeded" or "Cgroup memsw limit exceeded" and will attempt to print the corresponding kernel `dmesg` buffer messages (if found), which usually identify the process that was killed.

16.7.3 Examples of Requesting Cores and Hyperthreads

Assume we have 2-way hyperthreaded processors.

When hyperthreading is enabled on a system and ncpus_are_cores is disabled, each core is associated with two threads. In this case, if a job submitter wants all threads of a hyperthreaded core, they should request ncpus in multiples of *2*.

When hyperthreading is not enabled, or if it is and ncpus_are_cores is enabled, a job submitter should request just the number of cores

For example, a job submitter requests the following on a cluster with 2-way hyperthreaded CPUs on all nodes:

```
-lselect=1:ncpus=2
```

Result:

- If hyperthreading is not enabled, this nets two cores. If the cpuset subsystem is enabled, only the first thread of each core is visible in the job cpuset.
- If hyperthreading is enabled and ncpus_are_cores is enabled, this also nets two cores, with a total of four threads visible in the job's cpuset.
- If hyperthreading is enabled and ncpus_are_cores is disabled, this nets two threads, with an attempt to assign the two threads from a single core; this may not succeed if other jobs on the vnode have requested odd numbers of ncpus.

16.7.4 Spawning Job Processes

When a job process is spawned using `tm_spawn`, the execjob_launch cgroups hook runs. The execjob_launch hook can set environment variables correctly and set per-process limits for the processes.

When a job process is spawned outside of PBS and `pbs_attach` is used to make the process join the job, the execjob_attach cgroups hook runs, but it is unable to set the environment for the job or set per-process limits for the process.

16.8 Caveats and Errors

16.8.1 Interactions Between Suspend/resume and the cpuset Subsystem

The cgroups hook is in general compatible with suspend/resume. But when using preemption via suspend and resume, unless vnodes are allocated exclusively to jobs in the class of preempting jobs, the class of preempted jobs, or both, since the scheduler is unaware of the CPU assignments made by MoM, it is possible for the scheduler to resume a low-priority job even though the job's cpuset still overlaps with that of a running high priority job.

To avoid this, use any of these methods:

- Disable the cpuset subsystem, and use only the cpu controller to limit excessive CPU resource usage by jobs. The lack of CPU isolation may still cause the jobs to interfere on some workloads.

  Also, not using the cpuset subsystem may require you to disable process pinning in your applications to share a vnode between more than one job, to allow the Linux CFS scheduler to move processes to CPU threads that are free.
- Ensure that each job in either the preempting workload or the preempted workload has exclusive access to all vnodes it uses.
- Use one of the recent hook events triggered on job resumption to either reject resumption of a job when a conflict is detected, or to migrate the cpusets to CPUs no longer assigned to active jobs. If migrating the cpuset, take a lock on the cgroups lock file.

16.8.2 Caveats for Shrinking a Job on a Host

If the cpuset subsystem is enabled, ensure that if the cgroups hook helps to shrink a job, no processes are running on the host.

16.8.3 Caveats for Using CUDA

To use CUDA version earlier than 7.0, you must allow access to all devices. Otherwise, the NVIDIA commands will fail. With CUDA 7.0 or greater, you do not need to allow access to all devices.

16.8.4 Do Not Change ncpus When cpuset Subsystem is Enabled

Do not change the value of the ncpus resource from that reported by MoM if the cpuset subsystem is enabled. Otherwise the cgroups hook will attempt to use CPUs that don't exist, and jobs will fail.

16.8.5 Cgroups Hook Prevents Epilogue from Running

If you are running the cgroups hook, any epilogue script will not run. The cgroups hook has an execjob_epilogue event which takes precedence over an epilogue script, so if you are running the cgroups hook, make your epilogue script into an execjob_epilogue hook instead. See section 14.5.2, “Using Hooks for Prologue and Epilogue”, on page 512.

16.8.6 Errors

When a job is trying to access a Xeon Phi device on a vnode, but the device is not accessible by the job cgroup, you will see the following error in the job error output:

```
Error getting SCIF driver version
```

17 Configuring PBS for SELinux

17.1 Overview of PBS Support for MLS-compliant SELinux

With this version of PBS Professional, we offer a separate package named PBSPro_2021.1.0-RHEL7_x86_64_selinux.tar.gz that supports SELinux enforcement mode used with one or more MLS policies on RHEL 7. In this chapter, we describe how to use SELinux PBS only.

17.2 Terminology

In this section, we use the following terms:

DCID
Director of Central Intelligence

MLS
Multi-level Security

PL4
Protection Level 4

17.3 How Support for SELinux Works

17.3.1 Security Context

SELinux PBS collects the security context at the time a request is made, records it in the server, and associates it with a user's job when the job is passed to the MoM. MoMs then use the information to create user processes with the correct context.

17.3.2 Authorization

Job submitters can alter, delete, and hold only those jobs with a security context which matches that of the requester.

PBS managers and operators are expected to have sufficient SELinux privilege to operate on all users' jobs.

17.3.3 Authentication

When authenticating a user, this version of PBS captures the user's SELinux security context, collecting the SELinux user identity, role, type, and MLS levels.

The only supported authentication method when using PBS with SELinux is the default of *resvport*.

17.3.4 Instantiation

When a MoM creates a user process, she creates it with the security context of the user who submitted the job. For each job, PBS records the security context of the job submitter in the security_context job attribute.

17.4 Enforcement of Permissions

PBS does not make the permission decisions required by MLS or DCID PL4. These decisions are made by the SELinux mechanisms delivered with the OS.

17.4.1 Policy Files

Altair supplies policy files describing the permissions for PBS. These files are in SELinux format.

The SELinux policy framework includes the permissions PBS needs to impersonate users. The policy framework is encapsulated in a set of policy files for this product.

SELinux policy files for PBS are located in `$PBS_EXEC/selinux/`. These policy files are the following:

`pbs.fc`

`pbs.if`

`pbs.te`

`pbs_dontaudit.te`

PBS policy files must be inspected and vetted by site security personnel before installation. See section 2.1.2, "Installation Steps", on page 8.

17.4.2 Location of pbs_mom.pamd File

The `pbs_mom.pamd` file is shipped in `$PBS_EXEC/selinux`.

For polyinstantiation to work properly, copy `$PBS_EXEC/selinux/pbs_mom.pamd` to the `/etc/pam.d/` directory.

17.5 Special Attributes and Directories

`$PBS_EXEC/selinux`
: Directory that contains SELinux policy files and miscellaneous data.

security_context
: Job attribute. Contains security context of job submitter. Set by PBS to the security context of the job submitter at the time of job submission. Visible to all. If not present when a request is submitted, an error occurs, a server message is logged, and the request is rejected.

  Format: String in SELinux format

  Default: Unset

17.6 Prerequisites

Because PBS managers and operators may need access to information requiring elevated privilege such as job names or security contexts, PBS requires that PBS managers and operators have sufficient SELinux privilege to operate on all users' jobs.

You can enable SELinux only on RHEL 7 hosts.

17.7 Caveats and Restrictions

- You cannot upgrade from a non-SELinux compliant PBS version to this version; you must do a fresh install.
- This version of PBS cannot interoperate with non-SELinux compliant clients (`qsub`, `qstat`, etc.).
- This version of PBS cannot interoperate with non-SELinux compliant complexes; you cannot do peer scheduling, route jobs, etc.
- You cannot mix compliant and non-compliant daemons
- In order to use SELinux with containers, you will probably need to adjust your site policy.
- The X forwarding that is enabled by PBS does not work with this version of PBS. You may be able to set up X forwarding via normal methods, without the help of PBS. Omit the "-X" `qsub` argument, and forward the DISPLAY information into the job itself.
- If a user runs `qstat` on a non-server host, all jobs are reported regardless of the value of query_other_jobs. To protect against this, set the server's acl_hosts attribute to exclude all non-server hosts:

  set server acl_hosts="-<non-server host>,-<non-server host>"

  For example:

  `set server acl_hosts ="-submithost1,-submithost2,-submithost3"`

  Set the server's acl_host_enable attribute to *True* to enable restricting host access:

  `set server acl_host_enable=true`
- An SELinux-enabled PBS complex cannot interoperate with non-SELinux-enabled PBS complexes.

17.8 Installing PBS For Use With SELinux

17.8.1 Pre-installation Requirements

- An SELinux run-time environment must be present.
- The SELinux run-time environment must be able to compile and install new policies.
- The site must configure polyinstantiation as needed.
- You must install, start, and stop PBS using an account that is not subject to polyinstantiation.
- If PBS will run on more than one host, you must ensure that the MLS label is passed through and trusted among the PBS peer services. Add the following line to each PBS node's `/etc/netlabels.rules`:

  `map add default address:<site-specific address/mask> protocol:cipsov4,32`

17.8.2 Installing in Non-default Location

The policy files, for example pbs.fc, contain full paths that assume default locations for PBS_HOME, PBS_EXEC, and PBS_CONF_FILE. If you will install PBS in a non-default location, make sure that you set these correctly. We recommend that you review these anyway.

17.8.3 Installation Steps

1. Make sure you are using an account that is not subject to polyinstantiation.
2. Make sure you have sufficient privilege to install a policy.
3. Extract the policy files:

   ```
   # rpm2cpio pbs-<version>.x86_64.rpm | cpio -id './opt/pbs/selinux/pbs*.[fit][cfe]'
   ```

4. Site security personnel inspect the policy files and PBS package.
5. Install the PBS policy module. We provide the following instruction for convenience:

   ```
   # make -f /usr/share/selinux/devel/Makefile && /usr/sbin/semodule -i pbs-mine.pp pbs_dontaudit.pp
   ```

6. Install the PBS package, but do not start PBS:

   ```
   # rpm -i pbspro-server-<version>.el7.x86_64.rpm
   ```

7. Put the PBS PAM module in place. Polyinstantiation is controlled by the PAM session module. You need a PBS PAM module in order to allow pbs_mom to polyinstantiate user jobs.
8. Copy pbs_mom.pamd from $PBS_EXEC/selinux/ on the PBS server installation host to /etc/pam.d/pbs_mom on all execution hosts.
9. Make sure that /etc/pam.d/pbs_mom is readable by all.

17.8.4 Starting SELinux PBS

Use the following instructions to start PBS. Do **not** use the standard instructions for starting PBS. Use the exact instructions below:

```
# systemctl start pbs.service
```

17.8.5 Post-installation Steps

17.8.5.1 Configure Job Privacy

Once PBS has been started, set the query_other_jobs server attribute to *False*.

17.8.5.2 Set Privacy for PBS Logs

Protect the PBS service logs from being read or written by anyone except root. The logs are found here:

$PBS_HOME/mom_logs

$PBS_HOME/sched_logs

$PBS_HOME/server_logs

$PBS_HOME/comm_logs

Change permissions on the logs:

```
chmod 700 $PBS_HOME/mom_logs $PBS_HOME/sched_logs $PBS_HOME/server_logs $PBS_HOME/comm_logs
```

17.8.6 Configuring Ports Used by PBS

If client commands such as qstat, qsub, etc., frequently fail with error code 15007, the likely problem is that the port in the 512-1023 range that PBS is trying to use to communicate has other SELinux policy rules set up which prohibit use by PBS.

For example, in this illustration PBS tried to use port 548, which is listed like this in semanage:

```
dhcpd_port_t                   tcp      547, 548, 647, 847, 7911
dhcpd_port_t                   udp      67, 547, 548, 647, 847
```

In our illustration, the audit log shows something like this:

```
type=AVC msg=audit(1437660325.774:5672072): avc:  denied  { name_bind } for  pid=40996
    comm="pbs_iff" src=548 scontext=staff_u:sysadm_r:sysadm_t:s1
    tcontext=system_u:object_r:dhcpd_port_t:s0 tclass=tcp_socket
```

When this happens, you need to let PBS know which ports in the 512 to 1023 range may be used by PBS. Specify at least 128 ports for use by PBS. Use the semanage command to appropriately label the ports that are usable by PBS:

```
semanage port -a -t pbsd_iff_port_t -p tcp <port range>
```

See semanage-port(8).

17.9 Configuring PBS for SELinux

17.9.1 Configure File Staging Utilities

PBS needs to use file staging utilities that can preserve security labels when staging files in and out. Files to be used for a job must have the user's context. Copying files without preserving the context results in a file that cannot be used by the job. When copying a local file, MoM automatically preserves the context. You must configure MoM to preserve context when copying remote files. You can use the following option for authenticated file transfer between Linux systems. The -xattrs option preserves context:

```
rsync -e ssh -xattrs
```

The following is another option for preserving context:

```
cp --preserve=context
```

MoM performs remote staging using the file transfer methods that you specify, or the default if none is specified. You specify the method you want by putting the path in the PBS_SCP and PBS_RCP variables in pbs.conf. First MoM tries the path in PBS_SCP, then the path in PBS_RCP, so put the same path in both. The following is a summary of how to tell MoM to use rsync:

- Write a script that passes the desired options and arguments to rsync
- Edit pbs.conf, and specify the path to the script in PBS_RCP and PBS_SCP
- If the MoM is running, HUP the MoM

17.9.1.1 Steps to Configure Utility

1. Write a wrapper script named `rsync_pbs` that passes all arguments except for the first (`-Brvp` or `-rp`) to `rsync`.

   MoM uses the `-Brvp` flags when calling PBS_SCP, and the `-rp` flags when calling PBS_RCP. The arguments that were being passed, and that you can borrow, are the following:

   $1 `-Brvp` or `-rp`

   $2 path to source

   $3 path to destination

   For example, our script passes all but the first argument to `rsync` as $*. We get rid of the first argument using the `shift` command.

   In `pbs.conf`:

   ```
   PBS_SCP=/usr/bin/rsync_pbs
   ```

   In `/usr/bin/rsync_pbs`:

   ```
   #!/bin/sh
   shift
   /usr/bin/rsync -e ssh -xattrs $*
   ```

2. Configure both PBS_RCP and PBS_SCP with the path to `rsync_pbs`.
3. If the MoM is already running, HUP the MoM.

17.10 Managing an SELinux System

17.10.1 Checking Security Context

PBS writes the security context for a job in its security_context attribute. For example, if you need to compare the security context of jobs and users:

Find job security context:

```
bash-4.2$ qstat -f | grep security
security_context = user_u:user_r:user_t:s3:c1,c2
```

Find user security context:

```
bash-4.2$ id -Z
user_u:user_r:user_t:s3:c1,c2
```

18 Configuring PBS for Containers

18.1 Introduction

PBS supplies a built-in hook that runs jobs and applications inside containers. The hook launches separate container(s) for each job, and runs the job with the same submission or job script commands and environment as it would have outside the container(s). The same job environment variables are exported inside the container(s), and file staging and job output and error files are handled the same way as outside a container.

Users can run multi-vnode, multi-host, and interactive jobs in containers, and PBS tracks resource usage for these jobs. The PBS container(s) use cgroups to constrain the resources that the job can use, track resource usage, and pin and isolate resources. The PBS container(s) use the cgroups hook to assign GPUs correctly; see Chapter 16, "Configuring and Using PBS with Cgroups", on page 571.

When the job finishes, PBS removes the container(s).

Jobs are matched to hosts running container daemons via a custom string array resource, which indicates which container engines are available on each host. You create this resource, and tell the container hook which engines are available on each host by setting the container_resource_name parameter in the hook's configuration file to the name of the custom string array resource. The default for the container_resource_name parameter is "container_engine", so we recommend using that name when you create the resource.

Job submitters can specify the job container image by requesting it or setting the CONTAINER_IMAGE environment variable to the name of the container in which the job should run. A container request in the container_image resource overrides the CONTAINER_IMAGE environment variable. The PBS container hook looks for a container request and monitors job submissions for this environment variable, launches the appropriate container, and starts the job in the container.

You can configure a list of allowed registries, and set a default registry.

PBS can perform a registry login in order to pull from registries that require a login.

You can whitelist specific additional arguments to the container engine by listing them in the container_args_allowed container hook parameter. Job submitters can then specify any of these whitelisted arguments in the PBS_CONTAINER_ARGS environment variable.

You can configure the container hook so that it automatically adds job owners to additional groups inside Docker containers. The hook finds the groups on the execution host where the job owner is already a member, and adds the job owner to these groups inside the container. To do this, set the enable_group_add_arg container hook parameter to *True*. This feature applies only to Docker; Singularity users are automatically added to all groups inside containers. Note that for security, we recommend that you never whitelist the `--group-add` container argument in the container_args_allowed hook configuration parameter.

You can set permissions on files mounted inside containers, for example setting files in a container to be read-only.

18.1.1 Container Engines Used by PBS

A PBS server can create Docker and Singularity containers. Each job can specify which container engine to use, but can use only one container engine. You specify the default container engine by setting the container_resource_default_value parameter in the container hook's configuration file to either "docker" or "singularity". In addition, a user can always run a single-node job in a single Singularity container by prepending their scripts, executables, or commands with the Singularity binary.

18.1.1.1 Using nvidia-docker

PBS can invoke `nvidia-docker` if the `nvidia-docker-cmd` line in the hook's configuration file points to the location of the `nvidia-docker` command, and the job requests ngpus inside its select statement.

18.1.1.2 Caching Singularity Images

When an image is downloaded from the container hub, it is saved on the execution host in the cache path. You can set the cache path in the container_cache_path hook configuration parameter. The default value for this parameter is empty, in which case it is treated as if it is `<user home>/.singularity/cache`.

PBS sets the value of the SINGULARITY_CACHEDIR Singularity environment variable to the value of container_cache_path. See the Singularity documentation for more information on this environment variable.

18.1.2 Container Ports

For single-vnode jobs in Docker containers, job submitters can request ports for applications. The container hook maps requested ports to available ports on the host and returns the mapping. You can define which port ranges are available for containers. The job submitter requests ports by listing comma-separated port numbers in the container_ports job resource. Lists of port numbers must be enclosed in single quotes. The hook sets the job's resources_used.container_ports value to comma-separated *<container port>:<host port>* pairs. For example, a job can request `-l container_ports="'2324,8989'"`, and the hook sets the job's resources_used.container_ports to 2324:8080,8989:32771.

18.1.3 Managing How Files and Directories are Mounted in Containers

18.1.3.1 Setting Permissions on Mounted Files

You can set permissions on the files and directories mounted in both Docker and Singularity containers using the mount_paths container hook configuration parameter. The default value for the mount_paths parameter is *["/etc/passwd", "/etc/group"]*.

You can set multiple mounting paths in the mount_paths parameter. Syntax:

"mount_paths": [<first path spec>, <second path spec>, ... <nth path spec>]

where *<path spec>* can be any of:

- A single filename, indicating that source and target have the same name
- ["<source name>", "<target name>"], indicating no restriction
- ["<source name>", "<target name>", "<restriction>"], specifying restriction on file inside container

When you put more than one element in a path spec, for example both a source and a target, enclose the path spec in square brackets.

Example 18-1: Setting /etc/passwd to read-only for a Singularity container:

In the hook configuration file:

```
"mount_paths": [["/etc/passwd","/etc/passwd","ro"]]
```

PBS mounts /etc/passwd in the container with the destination /etc/passwd, and sets the permissions to *ro* (read-only) on the file inside container.

Example 18-2: Docker: setting multiple mount paths, with restriction

In the hook configuration file, we have two path specifications:

```
"mount_paths": [["/etc/passwd","/etc/passwd","readonly", "bind-propagation=rslave"],
    "/etc/group"]
```

The first path spec is ["/etc/passwd","/etc/passwd","readonly"], and PBS passes the following to Docker:

```
docker --mount type=bind,source=/etc/passwd,target=/etc/passwd,readonly,bind-propaga-
    tion=rslave
```

The second path spec is ["/etc/group"], and PBS passes the following to Docker:

```
docker --mount type=bind,source=/etc/group,target=/etc/group
```

18.1.3.2 Allowing or Disallowing Job Work Directory Inside Container

It may be possible to submit a job from a place that is not sensitive on the submission host, but is sensitive on the compute hosts. You can allow or disallow mounting the job's work directory inside Docker or Singularity containers by setting the value of the mount_jobdir hook configuration parameter. Setting this to *True* enables mounting the job's work directory in containers. The default value of this parameter is *True*.

18.1.4 How PBS Uses Container Registries

You can configure a list of allowed (whitelisted) registries by listing them in the allowed_registries hook configuration parameter. PBS checks each job's registry specification against the whitelist. PBS uses only whitelisted registries. If the specified registry is not whitelisted, PBS rejects the job. If the job submitter does not specify a registry, PBS uses the default registry. You specify the default registry by making it the first entry in the allowed_registries parameter. Set the default registry according to the default container engine you are using (*docker* or *singularity*). For Singularity, set the default based on the value of container_image_source (*docker://* or *library://*).

You can whitelist all registries by including *PBS_ALL* in the list.

The default value for allowed_registries is ["docker.io", "SylabsCloud", "PBS_ALL"].

18.1.5 Registry Credential File

PBS can pull images using job owner registry login credentials; this allows job owners to use images from registries where a login is required. If a registry allows you to pull without logging in, PBS allows this.

PBS stores login credentials in a JSON file, in a directory you specify. Make sure that the hook, which runs as root, can read the file.

Example 18-3: PBS uses the job owner's credentials to pull a container image:

```
qsub -v CONTAINER_IMAGE= myregistry.local/MyImage
```

PBS checks whether "myregistry.local" is included in the allowed_registries parameter.

If "myregistry.local" is included, PBS logs in using the credentials that are listed in <job owner>/.container/tokens.json for myregistry.local.

If "myregistry.local" is not included, the job is rejected.

18.1.5.1 Registry Credential Filename

The credential filename has this format:

<job owner>/.container/tokens.json

18.1.5.2 Registry Credential File Format

The file contents have this format:

```
{
   "registry1 <URL>/<endpoint>": {
      "user_id" : "<user ID>" , "passwd" : "<generated OAUTH token/password>"
   },

   "registry2 <URL>/<endpoint>": {
      "user_id" : "<user ID>" , "passwd" : "<generated OAUTH token/password>"
   }
}
```

18.1.5.3 Registry Credential File Default Values

registry: default registry (first element in the allowed_registries parameter)

user_id: job owner; if this is empty, PBS tries instead with the job owner ID

passwd: no password

18.1.5.4 Registry Credential File Location

The registry credential file *base path* is the path to where registry credential files are stored, up to but not including <job owner>/.container/tokens.json. The default base path to registry credential files is /home. You can configure the base path to where registry credential files are stored, by setting the value of the cred_base_path parameter in the hook configuration file.

Example 18-4: You set cred_base_path to "/container/creds/", and your job owner is User1. The full path to the JSON file is:

/container/creds/User1/.container/tokens.json

18.1.5.5 Docker Examples

Example 18-5: Job submission with image specification:

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/pbsuser/test-image
```

PBS looks for "pbsprohub.local" in the allowed_registries hook configuration parameter. If the registry is whitelisted, PBS looks for login credentials for the pbsprohub.local registry in the <job owner>/.container/tokens.json file. PBS logs into the registry and pulls the requested container image.

If the login credentials for the pbsprohub.local registry are not listed in <job owner>/.container/tokens.json and the registry does not require a login, PBS skips logging in and pulls the requested container image.

If "pbsprohub.local" is not listed in the allowed_registries configuration parameter, PBS rejects the job.

Example 18-6: Job submission without image specification:

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/centos:7
```

PBS looks for "pbsprohub.local" in the allowed_registries hook configuration parameter. If the registry is whitelisted, PBS looks for login credentials for the pbsprohub.local registry in the <job owner>/.container/tokens.json file. PBS logs into the registry and pulls the requested container image.

If "pbsprohub.local" is not listed in allowed_registries, PBS uses the default registry. This can lead to possible failure at runtime.

Example 18-7: Job submission without registry specification:

```
qsub -v CONTAINER_IMAGE=pbsuser/test-image
```

Since no registry is specified, PBS uses the default registry. PBS uses the login credentials if they are listed in <job owner>/.container/tokens.json, and pulls <default registry>/pbsuser/test-image.

18.1.5.6 Singularity Examples

Example 18-8: Job submission without registry specification:

```
qsub -v CONTAINER_IMAGE=pbsuser/test-image
```

In the hook configuration file:

```
container_image_source = "docker://"
```

Since no registry is specified in the job request, PBS uses the default registry.

PBS uses the container image found at docker://<default registry>/pbsuser/test-image, and PBS uses login credentials if they are listed in <job owner>/.container/tokens.json.

Example 18-9: Job submission without defined endpoint:

```
qsub -v CONTAINER_IMAGE=pbspro/default/test-image
```

In the hook configuration file:

```
container_image_source = "library://"
```

If "pbspro" is not a defined endpoint, and is not listed in allowed_registries, PBS uses the default registry. Singularity remote endpoints enable secure container sharing. See the Singularity documentation about remote endpoints.

If "pbspro" is a defined endpoint, PBS checks the allowed_registries list, then uses the credentials in <job owner>/.container/tokens.json. PBS uses the "pbspro" endpoint, authenticating via the credentials in <job owner>/.container/tokens.json, by calling the command "singularity remote login".

Example 18-10: Path to image is "shub":

In the hook configuration file:

```
container_image_source = "shub://"
```

Because the Singularity hub is a public repository, PBS does not perform any authorization.

18.2 The PBS Container Hook

PBS has a built-in container hook named "PBS_hpc_container" which does several useful things:

- The `PBS_hpc_container` hook can create Docker and Singularity containers, and it can invoke `nvidia-docker` if it is configured and the job requests ngpus inside the select statement.
- The hook runs for the following events, with these actions:
  - At a queuejob event, the hook adds the name and desired value of the string array resource listing container engines to the job's select statement, if the job does not already specify it. This allows the scheduler to match the job to a host running the selected container daemon.
  - At a queuejob event, the hook checks the allowed_registries parameter. If there are whitelisted registries, it then looks for registry login credentials for the job owner.
  - At an execjob_launch event, the hook launches the job inside the selected container
- The hook starts a container instance from the requested image, and sets up the job's environment. The image is specified via `-lcontainer_image=<container image>` or in the job's CONTAINER_IMAGE environment variable. The hook uses the requested container engine, or if the job does not request a container engine, the hook uses the default set in the container_resource_default_value parameter in the hook's configuration file.
  - The name of the container is the job ID.
  - If the job is interactive, the hook runs the job in the container in interactive mode.
  - If the job has multiple chunks that are scheduled to run on a single host:

    With Docker, the hook runs all of the job's child processes in one container on that host.

    With Singularity, the hook runs each child process in its own container.
  - If the job runs on multiple hosts, the hook ensures that containers created on sister MoMs are network linked to the container running on the primary execution host.
- The hook updates the resources used by the job, and removes the job's container.
- The hook cleans up any orphaned containers left behind by previous jobs on the host.
- The hook can automatically add the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.

18.3 Prerequisites

The required container daemon(s) must be running on all hosts where users will run jobs in containers.

18.4 Configuring PBS for Containers

18.4.1 Create Container Resources

1. For the container image name, create a custom string resource named "container_image":
   ```
   Qmgr: create resource container_image type=string,flag=m
   ```
2. For the container port number, create a custom string array resource named "container_ports":
   ```
   Qmgr: create resource container_ports type=string_array,flag=m
   ```
3. Create the custom string array resource that will list the container engines available on each host. We recommend naming it "container_engine":
   ```
   qmgr -c "create resource container_engine type=string_array,flag=mh"
   ```
4. Set the value of the container_engine resource on each host to the list of available container engines:
   qmgr -c "s n node1 resources_available.container_engine=<list of container engines>"
   For example, if you have Docker on node1, and you have both Docker and Singularity on node2:
   ```
   qmgr -c "s n node1 resources_available.container_engine=docker"
   qmgr -c "s n node2 resources_available.container_engine=docker"
   qmgr -c "s n node2 resources_available.container_engine += singularity"
   ```
5. Add the container_engine resource to the `resources:` line in `PBS_HOME/<sched_priv directory>/sched_config`.
6. HUP the scheduler:
   ```
   kill -HUP <scheduler PID>
   ```

18.4.2 Configure PBS Container Hook

The container hook's configuration file allows parameters that are specific to each container engine.

18.4.2.1 Default Configuration File

```
{
    "container_resource_name": "container_engine",
    "container_resource_default_value": "docker",
    "mount_paths": ["/etc/passwd", "/etc/group"],
    "mount_jobdir": true,
    "cred_base_path": "",
    "allowed_registries": ["docker.io", "SylabsCloud", "PBS_ALL"],
    "docker":{
        "container_cmd": "/usr/bin/docker",
        "nvidia_docker_cmd": "/bin/nvidia-docker",
        "remove_env_keys": [],
        "port_ranges": [],
        "container_args_allowed": [],
        "enable_group_add_arg": false
    },
    "singularity":{
        "container_cmd": "/usr/local/bin/singularity",
        "container_image_source": "",
        "container_cache_path": "",
        "container_args_allowed": []
    }
}
```

The following table shows the parameters:

Table 18-1: PBS Container Hook Configuration File Parameters

Parameter Name	Default Value	Description
allowed_registries	*[]*	Whitelist of registries PBS is allowed to pull from. Put default registry first; PBS uses this when it cannot find the specified registry. Set this to *PBS_ALL* to allow all registries.
container_args_allowed	*[]*	Whitelist of arguments that job submitters are allowed to pass to container engine via the PBS_CONTAINER_ARGS environment variable. Do not whitelist `--env`, `--entrypoint`, `--group-add`
container_cache_path	*[]*	For Singularity. Path to cache directory on execution host where singularity images are cached.
container_cmd	*/usr/bin/docker*	Path to container command. Can be "/usr/bin/docker" or "/usr/local/bin/singularity"

Table 18-1: PBS Container Hook Configuration File Parameters

Parameter Name	Default Value	Description
container_image_source	*[]*	Singularity only. Can be path to existing container image, or URI of container hub. Optional; job submitters can specify path to image. Example: For an image with the path `/home/user1/singularity_images/centos_latest.sif`, set container_image_source to *["/home/user1/singularity_images/"]* Example Singularity hub: ["shub://"] Example Docker hub where Singularity can fetch an image and convert it to SIF: ["docker://"]
container_resource_default_value	*docker*	Default container engine
container_resource_name	*container_engine*	Name of resource that lists available container engines on each host.
cred_base_path	*[]*	Base path to user login credential file, where credential file is `<user ID>/.container/tokens.json`. Default is empty, in which case PBS uses `/home`.
enable_group_add_arg	*false*	The hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs. Applies to Docker only; Singularity automatically adds job owners to all groups.
mount_jobdir	*true*	Boolean for enabling or disabling mounting the job's working directory inside the container
mount_paths	*["/etc/passwd", "/etc/group"]*	Additional paths to mount into container at creation time, with additional options for security. For example ["/opt/mpich"], or [["/etc/passwd","/etc/passwd","ro"]]
nvidia_docker_cmd	*/bin/nvidia-docker*	Path to nvidia-docker command
port_ranges	*[]*	Docker only. Comma-separated ranges of ports, for example ["2001-9999","3500-4500","7600-9500"]
remove_env_keys	*[]*	Docker only. List of environment variables not to export to job container

To configure your PBS container hook, export the configuration file, edit it, and re-import it.

1. Export the PBS container hook's configuration file:
   ```
   #qmgr -c "export pbshook PBS_hpc_container application/x-config default" > container_config.json
   ```
2. Set global parameters in the PBS container hook configuration file to match your site. The configuration file must conform to JSON syntax.
   - Set the alllowed_registries parameter to the list of allowed registries. See section 18.1.4, "How PBS Uses Container Registries", on page 617
   - Set the container_cmd parameter to the path of the container command
   - Set the container_resource_name parameter to the name of the resource that lists available container engines, if you used a name other than "container_engine"
   - Set the container_resource_default_value parameter to the default container engine, if you want it to be different from "docker"
   - Optionally create the credential file(s) `<job owner>/.container/tokens.json` (job submitters may want to do this step)
   - Optionally set the cred_base_path parameter to the credential file path; see section 18.1.5.4, "Registry Credential File Location", on page 618
   - If using Singularity, optionally set the container_cache_path parameter to the path where Singularity images will be stored. See section 18.1.1.2, "Caching Singularity Images", on page 616
   - Optionally set the container_image_source parameter
   - Optionally set mount_paths; see section 18.1.3.1, "Setting Permissions on Mounted Files", on page 616
   - Optionally disallow mounting the job's work directory in containers; see section 18.1.3.2, "Allowing or Disallowing Job Work Directory Inside Container", on page 617
   - If using `nvidia-docker`, set nvidia_docker_cmd
   - If using Docker, set port_ranges to ranges of allowed ports on hosts
   - Optionally set remove_env_keys
   - Optionally set container_args_allowed to a whitelist of arguments that job submitters can pass to the container engine via the PBS_CONTAINER_ARGS environment variable.
     - Do not include "--entrypoint"; entry points are not supported
     - Do not include "--env"; this is not supported
     - Do not include "--group-add"; this poses security risks
   - Optionally set enable_group_add_arg to *True* so that the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.
3. For local users and local groups, include `/etc/passwd` and `/etc/group` in mount paths so the image has the host operating system users and groups defined.
4. To configure PBS to use `nvidia-docker`, make sure it is available in the path specified in the `nvidia-docker-cmd` line in the hook configuration file.

We show a sample configuration file here:

```
{
    "container_resource_name": "container_engine",
    "container_resource_default_value": "docker",
    "mount_paths": ["/etc/passwd", "/etc/group"],
    "mount_jobdir": true,
    "cred_base_path": "",
    "allowed_registries": ["docker.io", "SylabsCloud", "PBS_ALL"],
    "docker":{
        "container_cmd": "/usr/bin/docker",
        "nvidia_docker_cmd": "/bin/nvidia-docker",
        "remove_env_keys": [],
        "port_ranges": [],
        "container_args_allowed": [],
        "enable_group_add_arg": false
    },
    "singularity":{
        "container_cmd": "/usr/local/bin/singularity",
        "container_image_source": "",
        "container_cache_path": "",
        "container_args_allowed": []
    }
}
```

5. Re-import the PBS container hook's configuration file:

```
#qmgr -c "import pbshook PBS_hpc_container application/x-config default container_config.json"
```

18.4.3 Install and Start Container Engines

Install and start Docker and/or Singularity on all hosts where you want to use them. Make sure that users are not part of any Docker groups. Consult documentation for your OS, Docker, and Singularity.

18.4.4 Configure Security Enhancement for Docker

We see a security shortcoming in Docker in the case where multiple users are on the same host, where users can see into each others' containers. We have implemented a security enhancement for this. We allow the job to run inside the container, but we don't add job submitters to the Docker group, and we don't allow job submitters to connect to the Docker container.

Our security enhancement for Docker integration allows jobs to run inside the container, but prevents job submitters from connecting to the Docker container. We use `pbs_container` to accomplish this.

Make `PBS_EXEC/sbin/pbs_container` a part of the Docker group. Set its SGID permissions:

```
chgrp docker PBS_EXEC/sbin/pbs_container
chmod 2755 PBS_EXEC/sbin/pbs_container
```

18.5 Caveats and Restrictions

- To run a shell in a container using anything besides the user's default, the job submitter must specify the shell using the `-S` option to `qsub`.
- Job submitters cannot use old-style resource requests such as `-lncpus` with containers.
- Any entry point in a container is disabled. If job submitters want to run an entry point command, they must include the complete command with its arguments on the command line.
- Make sure that when you are configuring the container hook, if you whitelist any container arguments in the container_args_allowed hook configuration parameter, do not whitelist "--group-add". This would allow job submitters to add themselves to any groups inside the container. Instead, set the enable_group_add_arg hook parameter to *True* so the hook automatically adds the job owner to groups in the container; these are the groups on the execution host to which the job owner already belongs.

18.6 Errors and Logging

Container creation errors are logged in the MoM log files. You can use `tracejob` to display these errors.

19 Accounting

19.1 The Accounting Log File

The PBS server automatically maintains an accounting log file on the server host only.

19.1.1 Name and Location of Accounting Log File

Accounting log files are written on the server host only.

The accounting log filename defaults to PBS_HOME/server_priv/accounting/ccyymmdd where *ccyymmdd* is the date.

You can place the accounting log files elsewhere by specifying the -A option on the pbs_server command line.

The argument to the -A option is the absolute path to the file to be written. If you specify a null string, the accounting log is not opened and no accounting records are recorded. For example, the following produces no accounting log:

```
pbs_server -A ""
```

19.1.2 Managing the Accounting Log File

If you use the default filename including the date, the server closes the file and opens a new file every day on the first write to the file after midnight.

If you use either the default file or a file named with the -A option, the server closes the accounting log upon daemon/service shutdown .and reopens it upon daemon/service startup.

The server closes and reopens the account log file when it receives a SIGHUP signal. This allows you to rename the old log and start recording again on an empty file. For example, if the current date is February 9, 2015 the server will be writing in the file 20150209. The following actions cause the current accounting file to be renamed feb9 and the server to close the file and start writing a new 20150209.

```
cd $PBS_HOME/server_priv/accounting
mv 20150209 feb9
kill -HUP <server PID>
```

19.1.3 Permissions for Accounting Log

The PBS_HOME/server_priv/accounting directory is owned by root, and has the following permissions:

```
drwxr-xr-x
```

19.2 Viewing Accounting Information

To see accounting information, you can do any of the following:

- Use the `tracejob` command to print out all accounting information for a job
- Use the `pbs-report` command to generate reports of accounting statistics from accounting files
- Use the PBS Works front-end tool called PBS Analytics
- Look at the accounting files using your favorite editor or viewer

19.2.1 Using the `tracejob` Command

You can use the `tracejob` command to extract the accounting log messages for a specific job and print the messages in chronological order. The `tracejob` command looks at all log files, so if you want to see accounting information only, use the `-l`, `-m`, and `-s` options to filter out scheduler, MoM, and server log messages. You can use `tracejob` to see information about a job that is running or has finished. For accounting information, use the `tracejob` command at the server host.

```
tracejob <job ID>
```

See "tracejob" on page 235 of the PBS Professional Reference Guide for details about using the `tracejob` command.

19.2.1.1 Permissions for the `tracejob` Command

Root privilege is required when using `tracejob` to see accounting information.

19.3 Format of Accounting Log Messages

The PBS accounting log is a text file with each entry terminated by a newline. There is no limit to the size of an entry.

19.3.1 Log Entry Format

The format of a message is:

<logfile date and time>;<record type>;<ID string>;<message text>

where

logfile date and time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss

record type

A single character indicating the type of record

ID string

The job or reservation identifier

message text

Message text format is blank-separated *keyword=value* fields.

Message text is ASCII text.

Content depends on the record type.

There is no dependable ordering of the content of each message.

There is no limit to the size of an entry.

19.3.2 Space Characters in String Entries

String entries in the accounting log may contain spaces. Under Linux, you must enclose any strings containing spaces with quotes.

Example 19-1: If the value of the Account_Name attribute is "*Power Users*", the accounting entry should look like this, either because you added the quotes or PBS did:

```
user=pbstest group=None account="Power Users"
```

19.3.2.1 Replacing Space Characters in String Entries

You can specify a replacement for the space character in any accounting string via the `-s` option to the `pbs_server` command by doing the following:

1. Bring up the *Services* dialog box
2. Select *PBS_SERVER*
3. Stop the server
4. In the start parameters, use the `-s` option to specify the replacement
5. Start the server

Example 19-2: To replace space characters with "*%20*", bring up the server with "`-s %20`".

In this example, PBS replaces space characters in string entries with "*%20*":

```
user=pbstest group=None account=Power%20Users
```

If the first character of the replacement string argument to the `-s` option appears in the data string itself, PBS replaces that character with its hex representation prefixed by *%*.

Example 19-3: Given a percent sign in one of our string entries:

```
account=Po%wer Users
```

Since % appears in the data string and our replacement string is "%20", PBS replaces % with its hex representation (%25):

```
account="Po%25wer%20Users"
```

19.4 Types of Accounting Log Records

Accounting records for job arrays and subjobs are the same as for jobs, except that subjobs do not have Q (job entered queue) records. PBS writes different types of accounting records for different events. We list the record types, and describe the triggering event and the contents for each record type.

A

Job was aborted by the server. The *message text* contains the explanation for why the job was aborted.

a

Job was altered via `qalter` or a server hook. The message text is a list of <job attrribute>=<new value> pairs, separated by spaces. This record shows only changes to a job attribute made via `qalter` or a server hook. This record does not show changes made by the server, a scheduler, or when MoM changes resources_used.

B

Reservation record, written at the beginning of a reservation period, for all types of reservations. This record is written when the start time of a confirmed reservation is reached. Possible information includes the following:

Table 19-1: B Record: Reservation Information

Entry	Explanation
Authorized_Groups= <*groups*>	Groups who are and are not authorized to submit jobs to the reservation.
Authorized_Hosts= <*hosts*>	Hosts from which jobs may and may not be submitted to the reservation.
Authorized_Users = <*users*>	Users who are and are not authorized to submit jobs to the reservation.
ctime= <*creation time*>	Timestamp; time at which the reservation was created, in seconds since the epoch.
duration = <*reservation duration*>	The duration specified or computed for the reservation, in seconds.
end= <*end of period*>	Time at which the reservation period is to end, in seconds since the epoch.
name= <*reservation name*>	Reservation name, if reservation creator supplied a name string for the reservation.
nodes= <*vnodes*>	Contents of resv_nodes reservation attribute
owner= <*reservation owner*>	Name of party who submitted the reservation request.
queue= <*queue name*>	Name of the reservation queue.
Resource_List= <*requested resources*>	List of resources requested by the reservation. Resources are listed individually , for example: `Resource_List.ncpus = 16 Resource_List.mem = 1048676kb`
start= <*start of period*>	Time at which the reservation period is to start, in seconds since the epoch.

C

Job was checkpointed and requeued. Not written for a snapshot checkpoint, where the job continues to run.

When a job is checkpointed and requeued, PBS writes a C record in the accounting log. The C record contains the job's exit status. If a job is checkpointed and requeued, the exit status recorded in the accounting record is *-12*.

The C record is written for the following:

- Using the `qhold` command
- Checkpointing and aborting a job

c

After vnode release, the c record shows job information for the upcoming phase. See also the u record, which shows the phase that just finished, and the e record, which shows the final phase. The c record's *message text* field contains the following:

Table 19-2: c Record: Upcoming Phase, After Vnode Release

Entry	Explanation
ctime=<*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
etime=<*time when job became eligible to run*>	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
exec_host=<*list of hosts and resources*>	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
exec_vnode=<*vnode_list*>	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode. Parentheses may be missing if the exec_vnode string was entered without them while executing `qrun -H`.
group= <*group name*>	Group name under which the job will execute.
jobname= <*job name*>	Name of the job.
project= <*project name*>	Job's project name at the start of the job.
qtime=<*time*>	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue=<*queue name*>	The name of the queue in which the job resides.
resources_used.<resource name> = <resource value>	List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is `qrerun`, and only in the case where the job did start running.

Resource_List.<resource name>= <*resource value*>	List of resources requested by the job. Resources are listed individually, for example:
	`Resource_List.ncpus =16`
	`Resource_List.mem =1048676kb`

run_count=<*count*>	The number of times the job has been executed.
session=<`job` *session ID*>	Session number of job.
start=<*start time*>	Time when job execution started, in seconds since epoch.
user=<*username*>	The user name under which the job will execute.

D

Job or subjob was deleted by request. If a running job is discarded by PBS, PBS writes a D record, but not an E record. The *message text* contains `requestor=<username>@<hostname>` to identify who deleted the job. The D record is written for the following actions:

Table 19-3: D Record Triggers

Action	Result
`qdel` a non-running job	Write record immediately
`qdel -W force` a job	Kill job, then write record after job is deleted
`qdel -W force` a provisioning job	Kill job, then write record after job is deleted
`qdel` a running job	Kill job, then write record when MoM gets back to us
`qrerun` a job	Kill job, then write record after job is requeued
`qdel` a subjob	Kill subjob, then write record after subjob is deleted
PBS discards a running job, for example due to hardware failure, and node_fail_requeue is triggered	Write record and requeue job

E

Job or subjob ended (terminated execution). If a running job is discarded by PBS, PBS writes a D record, but not an E record. The end-of-job accounting record is not written until all of the job's resources are freed. The E record is written for the following actions:

- When a job array finishes (all subjobs are in the X state). For example, [1] and [2] finish, but we delete [3] while it's running, so the job array [] gets an E record.
- After a multi-node job is discarded (deleted) after nodes go down and node_fail_requeue is triggered
- When MoM sends an obit to the server

The E record can include the following:

Table 19-4: E Record: Job End

Entry	Explanation
account= <*account name*>	Written if job has a value for its Account_Name attribute
accounting_id= <*JID value*>	CSA JID, job container ID; value of job's accounting_id attribute
alt_id= <*alternate job ID*>	Optional alternate job identifier.
array_indices=<*array indices*>	Array indices job array was submitted with, if this is a job array. Not reported for subjobs.
ctime= <*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
eligible_time= <*eligible time*>	Amount of time job has waited while blocked on resources, in seconds.
end= <*job end time*>	Time in seconds since epoch when this accounting record was written. Includes time to stage out files, delete files, and free job resources. The time for these actions is not included in the walltime recorded for the job.

Table 19-4: E Record: Job End

Entry	Explanation
etime= *<time when job became eligible to run>*	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
exec_host= *<list of hosts and resources>*	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
exec_vnode= *<vnodes>*	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode. Parentheses may be missing if the exec_vnode string was entered without them while executing `qrun -H`.
Exit_status= *<exit status>*	The exit status of the job or subjob. See "Job Exit Status Codes" on page 519 in the PBS Professional Administrator's Guide and "Job Array Exit Status", on page 156 of the PBS Professional User's Guide. The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.
group= *<group name>*	Group name under which the job executed. This is the job's egroup attribute.
jobname= *<job name>*	Name of the job
pcap_accelerator	Power cap for an accelerator. Corresponds to Cray capmc set_power_cap --accel setting.
pcap_node	Power cap for a node. Corresponds to Cray capmc set_power_cap --node setting.
pgov	Cray ALPS reservation setting for CPU throttling corresponding to p-governor.
project= *<project name>*	Job's project name when this record is written
qtime=*<time>*	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue= *<queue name>*	Name of the queue from which the job executed
resources_used.<resource name>= *<resource value>*	Resources used by the job as reported by MoM. Typically includes ncpus, mem, vmem, cput, walltime, cpupercent. walltime does not include suspended time. Some resources get special reporting; see section 19.6.2.2, "Reporting Resources Used by Job", on page 652.
Resource_List.<resource name>= *<resource value>*	List of resources requested by the job. Resources are listed individually, for example: `Resource_List.ncpus =16 Resource_List.mem =1048676kb`
resvID=*<reservation ID>*	ID of reservation job is in, if any
resvname=*<reservation name>*	Name of reservation job is in, if any
run_count=*<count>*	The number of times the job has been executed.

Table 19-4: E Record: Job End

Entry	Explanation
session=<*session ID*>	Session number of job.
start=<*start time*>	Time when job execution started, in seconds since epoch.
user=<*user name*>	The user name under which the job executed. This is the job's euser attribute.

e

For a job whose vnodes were released, the e record shows information for the final phase of the job, after vnodes were released. See also the c record, which shows info for an upcoming phase, and the u record, which shows info for the just-finished phase. The e record's *message text* field contains the following:

Table 19-5: e Record: Final Phase After Vnode Release

Entry	Explanation
ctime=<*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
etime=<*time when job became eligible to run*>	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
exec_host=<*list of hosts and resources*>	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
exec_vnode=<*vnode_list*>	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode. Parentheses may be missing if the exec_vnode string was entered without them while executing `qrun -H`.
group= <*group name*>	Group name under which the job will execute.
jobname= <*job name*>	Name of the job.
project= <*project name*>	Job's project name at the start of the job.
qtime=<*time*>	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue=<*queue name*>	The name of the queue in which the job resides.
resources_used.<resource name> = <resource value>	List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is `qrerun`, and only in the case where the job did start running.

Resource_List.<resource name>= <*resource value*>	List of resources requested by the job. Resources are listed individually, for example:
	`Resource_List.ncpus =16`
	`Resource_List.mem =1048676kb`

run_count=<*count*>	The number of times the job has been executed.

Table 19-5: e Record: Final Phase After Vnode Release

Entry	Explanation
session=<job *session ID*>	Session number of job.
start=<*start time*>	Time when job execution started, in seconds since epoch.
user=<*username*>	The user name under which the job will execute.

K

Reservation deleted at request of scheduler or server. The *message text* field contains `requestor=Server@<hostname>` or `requestor=Scheduler@<hostname>` to identify who deleted the resource reservation.

k

Reservation terminated by owner issuing a `pbs_rdel` command. Written for all types of reservations. The *message text* field contains `requestor=<username>@<hostname>` to identify who deleted the reservation.

L

Information about node or socket licenses. Written when the server periodically reports license information from the license server. This line in the log has the following fields:

<Log date>; <record type>; <keyword>; <specification for license>; <hour>; <day>; <month>; <max>

The following table describes each field:

Table 19-6: Licensing Information in Accounting Log

Field	Description
Log date	Date of event
record type	Indicates license info
keyword	license
specification for license	Indicates that this is license info
hour	Number of licenses used in the last hour
day	Number of licenses used in the last day
month	Number of licenses used in the last month
max	Maximum number of licenses ever used. Not dependent on server restarts.

M

Job move record. When a job or job array is moved to another server, PBS writes an M record containing the date, time, record type, job ID, and destination.

Example of an accounting log entry:

```
7/08/2008 16:17:38; M; 97.serverhost1.domain.com; destination=workq@serverhost2
```

When a job array has been moved from one server to another, the subjob accounting records are split between the two servers.

Jobs can be moved to another server for one of the following reasons:

- Moved for peer scheduling
- Moved via the qmove command
- Job was submitted to a routing queue, then routed to a destination queue at another server

P

Provisioning starts for job or reservation.

Format: *<Date and time>;<record type>;<job or reservation ID>; <message text>,* where *message text* contains:

- User name
- Group name
- Job or reservation name
- Queue
- List of vnodes that were provisioned, with the AOE that was provisioned
- Provision event (START)
- Start time in seconds since epoch

Example:

```
"01/15/2009 12:34:15;P;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
    prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=START start_time=1231928746"
```

p

Provisioning ends for job or reservation. Provisioning can end due to either a successful finish or failure to provision.
Format: *<Date and time>;<record type>;<job or reservation ID>; <message text>,* where *message text* contains:

- User name
- Group name
- Job or reservation name
- Queue
- List of vnodes that were provisioned, with the AOE that was provisioned
- Provision event (END)
- Provision status (SUCCESS or FAILURE)
- End time in seconds since epoch

Example printed when job stops provisioning:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
    prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=SUCCESS
    end_time=1231928812"
```

Example printed when provisioning for job failed:

```
"01/15/2009 12:34:15;p;108.mars;user=user1 group=group1 jobname=STDIN queue=workq
    prov_vnode=jupiter:aoe=osimg1+venus:aoe=osimg1 provision_event=END status=FAILURE
    end_time=1231928812"
```

Q

Job entered a queue. Not written for subjobs. PBS writes a new Q record each time the job is routed or moved to a new queue or to the same queue.

The Q record can include the following:

Table 19-7: Q Record: Job Queued

Entry	Explanation
account= <*account name*>	Written if job has a value for its Account_Name attribute
accounting_id= <*JID value*>	CSA JID, job container ID; value of job's accounting_id attribute
array_indices=<*array indices*>	Array indices job array was submitted with, if this is a job array. Not reported for subjobs.
ctime= <*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
depend=<*dependencies*>	The job's dependencies, if any
etime= <*time when job became eligible to run*>	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
group= <*group name*>	Group name under which the job executed. This is the job's egroup attribute.
jobname= <*job name*>	Name of the job
pcap_accelerator	Power cap for an accelerator. Corresponds to Cray capmc set_power_cap --accel setting.
pcap_node	Power cap for a node. Corresponds to Cray capmc set_power_cap --node setting.
pgov	Cray ALPS reservation setting for CPU throttling corresponding to p-governor.
project= <*project name*>	Job's project name when this record is written
qtime=<*time*>	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue= <*queue name*>	Name of the queue from which the job executed
Resource_List.<resource name>= <*resource value*>	List of resources requested by the job. Resources are listed individually, for example: `Resource_List.ncpus =16 Resource_List.mem =1048676kb`
resvID=<*reservation ID*>	ID of reservation job is in, if any
resvname=<*reservation name*>	Name of reservation job is in, if any
user=<*user name*>	The user name under which the job executed. This is the job's euser attribute.

R

Job information written when job is rerun via `qrerun` or node_fail_requeue action, or when MoM is restarted without the `-p` or `-r` options. Not written when job fails to start because the prologue rejects the job. Possible information includes:

Table 19-8: R Record: Job Rerun

Entry	Explanation
account= <*account name*>	Written if job has an "account name" string
accounting_id= <*JID value*>	CSA JID, job container ID
alt_id= <*alternate job ID*>	Optional alternate job identifier.

Table 19-8: R Record: Job Rerun

Entry	Explanation
ctime= *<creation time>*	Timestamp; time at which the job was created, in seconds since the epoch.
eligible_time= *<eligible time>*	Amount of time job has waited while blocked on resources, starting at creation time, in seconds.
end= *<time>*	Time in seconds since epoch when this accounting record was written. Includes time to delete files and free resources.
etime= *<time job became eligible to run>*	Timestamp; time in seconds since epoch when job most recently became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or when a job is held then released. Not affected by qaltering.
exec_host= *<list of hosts and resources>*	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
exec_vnode= *<vnode_list>*	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode. Parentheses may be missing if the exec_vnode string was entered without them while executing `qrun -H`.
Exit_status= *<exit status>*	The exit status of the previous start of the job. See "Job Exit Status Codes" on page 519 in the PBS Professional Administrator's Guide. The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.
group=*<group name>*	The group name under which the job executed.
jobname=*<job name>*	The name of the job.
project=*<project name>*	The job's project name.
qtime=*<time job was queued>*	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue=*<queue name>*	The name of the queue in which the job is queued.
Resource_List.<resource name>= *<resource value>*	List of resources requested by the job. Resources are listed individually, for example: `Resource_List.ncpus =16 Resource_List.mem =1048676kb`
resources_used.<resource name> = <resource value>	List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is `qrerun`, and only in the case where the job did start running.
run_count=*<count>*	The number of times the job has been executed.
session=*<session ID>*	Session ID of job.
start=*<start time>*	Time when job execution started most recently, in seconds since epoch.
user=*<user name>*	The user name under which the job executed.

r

Job has been resumed. Format:

<logfile date and time>;<record type>;<job ID string>;

S

Job execution started. The *message text* field contains the following:

Table 19-9: S Record: Job Start

Entry	Explanation
accounting_id= <*accounting string*>	An identifier that is associated with system-generated accounting data. In the case where accounting is CSA, *accounting string* is a job container identifier or JID created for the PBS job.
array_indices=<*array indices*>	Array indices job array was submitted with, if this is a job array. Not reported for subjobs.
ctime= <*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
etime= <*time when job became eligible to run*>	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "*Q*" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
exec_host= <*list of hosts and resources*>	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
group= <*group name*>	Group name under which the job will execute.
jobname= <*job name*>	Name of the job.
pcap_accelerator	Power cap for an accelerator. Corresponds to Cray capmc set_power_cap --accel setting.
pcap_node	Power cap for a node. Corresponds to Cray capmc set_power_cap --node setting.
pgov	Cray ALPS reservation setting for CPU throttling corresponding to p-governor.
project= <*project name*>	Job's project name at the start of the job.
qtime= <*time*>	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue= <*queue name*>	The name of the queue in which the job resides.
resources_assigned.<resource name>= <*resource value*>	**Not** a job attribute; simply a label for reporting job resource assignment. One resource per entry. Includes all allocated consumable resources. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the *-n* or *-f* flags. A resource is not listed if the job does not request it directly or inherit it by default from queue or server settings. Actual amount of each resource assigned to the job by PBS. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs.

Resource_List.<resource name>= <*resource value*>	List of resources requested by the job. Resources are listed individually, for example:
	`Resource_List.ncpus =16`
	`Resource_List.mem =1048676kb`

Table 19-9: S Record: Job Start

Entry	Explanation
resvID=<*reservation ID*>	ID of reservation job is in, if any
resvname=<*reservation name*>	Name of reservation job is in, if any
session= <*job session ID*>	Session number of job.
start=<*start time*>	Time when job execution started, in seconds since epoch.
user= <*username*>	The user name under which the job will execute.

s

A job's vnode request was trimmed via release_nodes() in an execjob_prologue or execjob_launch hook. The job's tolerate_node_failures attribute must be set to *all* or *job_start*. The *s* record shows the new trimmed vnode request. Format: *<date> <time>;s;<job ID>;user=<job owner> group=<group> project=<project> jobname=<job name> queue=<queue name> ctime=<ctime> qtime=<qtime> etime=<etime> start=<start time> exec_host=<exec_host> exec_vnode=<exec_vnode> Resource_List.<resource>=<value> Resource_List.<resource>=<value>...Resource_List.<resource>=<value>*

T

Job was restarted from a checkpoint file.

U

Unconfirmed advance, standing, job-specific ASAP, or maintenance reservation requested.

- For an advance, job-specific ASAP, or maintenance reservation, the *message text* field has this format:

  U:<reservation ID> requestor=<username>@<hostname>
- For a standing reservation, the *message text* field has this format:

  U:<reservation ID>;requestor=<username>@<hostname> recurrence_rule=<recurrence rule> timezone=<time-zone>

  Example: "U;S56.exampleserver;requestor=pbsuser@exampleserver recurrence_rule=FREQ=HOURLY;COUNT=2 timezone=Asia/Kolkata".

u

After vnode release, the u record shows job information for the phase that just finished. See also the c record, which shows the upcoming phase, and the e record, which shows the final phase. The u record's *message text* field contains the following:

Table 19-10: u Record: Phase Just Finished, After Vnode Release

Entry	Explanation
ctime=<*creation time*>	Timestamp; time at which the job was created, in seconds since the epoch.
etime=<*time when job became eligible to run*>	Timestamp; time in seconds since epoch when job became eligible to run, i.e. was enqueued in an execution queue and was in the "Q" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
exec_host=<*list of hosts and resources*>	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.

Table 19-10: u Record: Phase Just Finished, After Vnode Release

Entry	Explanation
exec_vnode=<*vnode_list*>	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode. Parentheses may be missing if the exec_vnode string was entered without them while executing `qrun -H`.
group= <*group name*>	Group name under which the job will execute.
jobname= <*job name*>	Name of the job.
project= <*project name*>	Job's project name at the start of the job.
qtime=<*time*>	Timestamp; the time that the job entered the current queue, in seconds since epoch.
queue=<*queue name*>	The name of the queue in which the job resides.
resources_used.<resource name> = <resource value>	List of resources used by the job. Written only when the server can contact the primary execution host MoM, as in when the job is `qrerun`, and only in the case where the job did start running.

Resource_List.<resource name>= <*resource value*>	List of resources requested by the job. Resources are listed individually, for example:
	`Resource_List.ncpus =16`
	`Resource_List.mem =1048676kb`

run_count=<*count*>	The number of times the job has been executed.
session=<`job` *session ID*>	Session number of job.
start=<*start time*>	Time when job execution started, in seconds since epoch.
user=<*username*>	The user name under which the job will execute.

Y

Reservation confirmed by the scheduler or altered. Written for all types of reservations.

- Advance, job-specific, or maintenance reservations:

  When an advance, job-specific, or maintenance reservation is confirmed for the first time, the Y record has this format:

  Y; <reservation ID> requestor=<requestor>@<server> start=<requested start time> end=<requested end time> vnodes=(<allotted vnodes>)

  Example: "Y; R123.server requestor=Scheduler@svr start=1497264531 end=1497264651 nodes=(node1:ncpus=3)"

  The Y record is written when an advance, job-specific, or maintenance reservation alter request is confirmed. The Y record has the same format as for the first time the reservation is confirmed, but the requested field(s) are updated with new value(s):

  Y; <reservation ID> requestor=<requestor>@<server> start=<(new/original) start time> end=<(new/original) end time> nodes=(<allotted vnodes>)

  Example: "Y; R123.server requestor=root@hostname start=1497264471 end=1497264651 nodes=(node1:ncpus=3)"

- Standing reservations:

  When a standing reservation is confirmed for the first time, the Y record has this format:

Y; <reservation ID> requestor=<requestor>@<server> start=<requested start time> end=<requested end time> nodes=(<allotted vnodes>) count=<count>

The *nodes* field is specific to the first occurrence.

Example: "Y; R123.server requestor=Scheduler@svr start=1497264531 end=1497264651 nodes=(node1:ncpus=3) count=3"

The Y record is written when a standing reservation alter request is confirmed. The Y record has the same format as as for the first time a standing reservation is confirmed, but the requested field(s) are updated with the new value(s), and the index of the next occurrence is appended. The *nodes* field is specific to the occurrence altered:

Y; <reservation ID> requestor=<requestor>@<server> start=<(new/original) start time> end=<(new/original) end time> nodes=(<allotted vnodes>) count=<count> index=<index of the altered occurrence>

Example: "Y; R123.server requestor=root@hostname start=1497264471 end=1497264651 nodes=(node1:ncpus=3) count=3 index=1"

The *allotted vnodes* is the value of the resv_nodes reservation attribute.

The *count* is the value of the reserve_count reservation attribute.

z

Job has been suspended. The *message string* contains the following:

- Values for the job's resources_used attribute
- Values for the job's resources_released attribute. This attribute is populated only when the server's restrict_res_to_release_on_suspend attribute is set; see section 5.9.6.2, “Job Suspension and Resource Usage”, on page 250.

19.4.1 Accounting Records for Job Arrays

Accounting records for job arrays and subjobs are the same as for jobs, except that subjobs do not have Q (job entered queue) records.

When a job array has been moved from one server to another, the subjob accounting records are split between the two servers.

When a job array goes from *Queued* to *Begin* state, we write one S for the whole job array.

The E record is written when a job array finishes. For example, [1] and [2] finish, but we delete [3] while it's running, so the job array gets an E record.

A job array is finished when all subjobs are in the *X* state.

19.5 Timeline for Accounting Messages

19.5.1 Timeline for Job Accounting Messages

The following is a timeline that shows when job and job array attributes and resources are recorded, and when they are written:

Table 19-11: Timeline for Job Accounting Messages

Job/Job Array Lifecycle	Accounting Record
Job is queued	Q
Job is moved	M
Application licenses are checked out	
Any required job-specific staging and execution directories are created	
PBS_JOBDIR and job's jobdir attribute are set to pathname of staging and execution directory	
If necessary, MoM creates job execution directory	
MoM creates temporary directory	
Files are staged in	
Just after job is sent to MoM	S T
primary execution host MoM tells sister MoMs they will run job task(s)	
MoM sets TMPDIR, JOBDIR, and other environment variables in job's environment	
MoM performs hardware-dependent setup: The job's cpusets are created, ALPS reservations are created	
The job script starts	
MoM runs user program	
Job starts an MPI process on sister vnode	
Job is suspended	z
Job is resumed	r
The job script finishes	
The obit is sent to the server	E, e
Any specified file staging out takes place, including `stdout` and `stderr`	
Files staged in or out are deleted	
Any job-specific staging and execution directories are removed	
The job's cpusets are destroyed	
Job files are deleted	E
Application licenses are returned to pool	

Table 19-11: Timeline for Job Accounting Messages

Job/Job Array Lifecycle	Accounting Record
Job is aborted by server	A
Job is checkpointed and held	C
Job is deleted	D
Periodic license information	L
Job is rerun via `qrerun` or node_fail_requeue	R
Job is restarted from a checkpoint file	T
`qdel` a subjob	D
`qrerun` a job	D
`qdel` a provisioning job with force	D
`qdel` any job with force	D
`qdel` a running job: kill job, then write record when MoM gets back to us	D
`qdel` a non-running job: write record now	D
Job array finishes	E
After a job is discarded after nodes go down on multi-vnode job	E
Job vnodes are released	c, u

19.5.2 Where Job Attributes are Recorded

Some accounting entries for job attributes have different names from their attributes. The following table shows the record(s) in which each job attribute is recorded and the entry name:

Table 19-12: Job Attributes in Accounting Records

Job Attribute	Record	Accounting Entry Name
accounting_id	E, Q, R, S	accounting_id
Account_Name	E, Q, R	account
accrue_type		
alt_id	E, R	alt_id
argument_list		
array		
array_id		
array_index		
array_indices_remaining		
array_indices_submitted	E, Q, S	array_indices
array_state_count		

Table 19-12: Job Attributes in Accounting Records

Job Attribute	Record	Accounting Entry Name
block		
Checkpoint		
comment		
create_resv_from_job		
ctime	c, E, e, Q, R, S, u	ctime
depend	Q	depend
egroup	c, E, e, Q, R, S, u	group
eligible_time	E, R	eligible_time
Error_Path		
estimated		
etime	c, E, e, Q, R, S, u	etime
euser	c, E, e, Q, R, S, u	user
executable		
Execution_Time		
exec_host	c, E, e, R, S, u	exec_host
exec_vnode	c, E, e, R, u	exec_vnode
Exit_status	E, R	Exit_status
forward_x11_cookie		
forward_x11_port		
group_list		
hashname		
Hold_Types		
interactive		
jobdir		
Job_Name	c, E, e, Q, R, S, u	jobname
Job_Owner		
job_state		
Join_Path		
Keep_Files		
Mail_Points		
Mail_Users		
mtime		
no_stdio_sockets		

Table 19-12: Job Attributes in Accounting Records

Job Attribute	Record	Accounting Entry Name
Output_Path		
pcap_accelerator	E, Q, S	
pcap_node	E, Q, S	
pgov	E, Q, S	
Priority		
project	c, E, e, Q, R, S, u	project
pset		
pstate		
qtime	c, E, e, Q, R, S, u	qtime
queue	c, E, e, Q, R, S, u	queue
queue_rank		
queue_type		
release_nodes_on_stageout		
Remove_Files		
Rerunable		
resources_released	z	resources_released
resources_used	c, E, e, R, u, z	resources_used
Resource_List	c, E, e, Q, R, S, u	Resource_List
resource_released_list		
run_count	c, E, e, R, u	run_count
run_version		
sandbox		
schedselect		
sched_hint		
server		
session_id	c, E, e, R, S, u	session
Shell_Path_List		
stagein		
stageout		
Stageout_status		
stime	c, E, e, R, S, u	start
Submit_arguments		
substate		

Table 19-12: Job Attributes in Accounting Records

Job Attribute	Record	Accounting Entry Name
sw_index		
tolerate_node_failures		
topjob_ineligible		
umask		
User_List		
Variable_List		

19.5.3 Timeline for Reservation Accounting Messages

The following table shows the timeline for when reservation accounting messages are recorded:

Table 19-13: Timeline for Reservation Accounting Messages

Reservation Lifecycle	Accounting Record
Unconfirmed reservation is created	U
Reservation is confirmed	Y
Reservation period begins	B, Y
Provisioning for reservation begins	P
Provisioning for reservation ends	p
Reservation period ends	F, Y
Reservation is altered	Y
Reservation is deleted by scheduler or server	K
Reservation is deleted by user via `pbs_rdel`	k

19.5.4 Where Reservation Attributes and Info are Recorded

Some accounting entries for reservation attributes have names that are different from their attributes. The following table shows the record(s) in which each reservation attribute is recorded and the entry name:

Table 19-14: Reservation Attributes/Info in Accounting Records

Reservation Attribute/Info	Record	Accounting Entry Name
Account_Name		
Authorized_Groups	B	Authorized_Groups
Authorized_Hosts	B	Authorized_Hosts
Authorized_Users	B	Authorized_Users
ctime	B	ctime

Table 19-14: Reservation Attributes/Info in Accounting Records

Reservation Attribute/Info	Record	Accounting Entry Name
delete_idle_time		
group_list		
hashname		
interactive		
Mail_Points		
Mail_Users		
mtime		
Priority		
queue	B	queue
reservation requestor	K, k, U, Y	requestor
reserve_count	Y	
reserve_duration	B	duration
reserve_end	B, Y	end
reserve_ID	Q	resvID
reserve_index		
reserve_job		
Reserve_Name	B, Q	name resvname
Reserve_Owner	B	owner
reserve_retry		
reserve_rrule	U (standing reservations)	recurrence_rule
reserve_start	B, Y	start
reserve_state		
reserve_substate		
reserve_type		
Resource_List	B	Resource_List
resv_nodes	B, Y	nodes
server		
timezone	U (standing reservation)	timezone
User_List		
Variable_List		

19.5.4.1 Jobs in Reservations

- The job's queue is recorded in its c, E, e, Q, R, S, and u records.
- The job's reservation name and reservation ID are written in its E, Q, and S records.

19.5.5 How MoM Polling Affects Accounting

MoM periodically polls for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

Example 19-4: MoM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

The output shown by a `qstat` during the window of time between T8 and T16 shows the resource usage up to T8.

If the `qstat` is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to the user if "`qsub -me`" was used in job submission) contains usage through T20.

The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.

If a job ends between MoM poll cycles, resources_used.<resource name> numbers will be slightly lower than they are in reality. For long-running jobs, the error percentage will be minor.

See section 3.1.2, "Configuring MoM Polling Cycle", on page 34 for details about MoM's polling cycle.

19.6 Resource Accounting

Job resources are recorded in the Q (job queued; Resource_List only), S (job start), R (job rerun), E (job end), c (upcoming phase when job vnodes are released), u (just-finished phase when job vnodes are released), s (vnodes trimmed), e (usage during post-release phase), and z (job suspension) records.

Reservation resources are recorded in the B (reservation start) record.

19.6.1 Accounting Log Resource Entry Formats

When reporting resources in the accounting B, c, E, e, R, S, or u records, there is one entry per resource. Each resource is reported on a separate line.

Values for requested resources are written in the same units as those in the resource requests. Values for resources_used and resources_assigned are written in *kb*. A suffix is always written unless the quantity is measured in bytes.

19.6.2 Job Resource Accounting

The following table shows which job and reservation resources are recorded in the accounting log, and lists the records where they are recorded:

Table 19-15: Job Resources in Accounting Log

Resources	Record	Description

Resource_List	c
	E
	e
	Q
	R
	S
	u
		List of resources requested by the job. Resources are listed individually, for example:
		`Resource_List.ncpus =16`
		`Resource_List.mem =1048676kb`

resources_assigned	S	**Not** a job attribute; simply a label for reporting job resource assignment. One resource per entry. Includes all allocated consumable resources. Consumable resources include ncpus, mem and vmem by default, and any custom resource defined with the *-n* or *-f* flags. A resource is not listed if the job does not request it directly or inherit it by default from queue or server settings. Actual amount of each resource assigned to the job by PBS. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs.
resources_released	z	Listed by vnode, consumable resources that were released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server.

Table 19-15: Job Resources in Accounting Log

Resources	Record	Description
resources_used	c E e R (when qrerun) u z	Resources used by the job as reported by MoM. Typically includes ncpus, mem, vmem, cput, walltime, cpupercent. walltime does not include suspended time.
exec_host	c E e R S u	List of job hosts with host-level, consumable resources allocated from each host. Format: *exec_host=<host A>/<index>*<CPUs> [+<host B>/<index> * <CPUs>]* where *index* is task slot number starting at 0, on that host, and *CPUs* is the number of CPUs assigned to the job, *1* if omitted.
exec_vnode	c E e R u	List of job vnodes with vnode-level, consumable resources from each vnode. Format: *(<vnode A>:<resource name>=<resource amount>:...)+(<vnode B>:<resource name>=<resource amount>:...)* *resource amount* is the amount of that resource allocated from that vnode.

19.6.2.0.i Accounting Log Entries for min_walltime and max_walltime

The Resource_List job attribute contains values for min_walltime and max_walltime. For example, if the following job is submitted:

```
qsub -l min_walltime="00:01:00",max_walltime="05:00:00" -l select=2:ncpus=1 job.sh
```

This is the resulting accounting record:

```
…S…….. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18 resources_assigned.ncpus=2
…R…….. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18
…E……. Resource_List.max_walltime=05:00:00 Resource_List.min_walltime=00:01:00
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=pack
    Resource_List.select=2:ncpus=1 Resource_List.walltime=00:06:18…….
```

19.6.2.1 Reporting Resources Assigned to Job

The value reported in the resources_assigned accounting entry is the amount assigned to a job or that a job prevents other jobs from using, which is different from the amount the job requested or used. For example, if a job requests one CPU on a multi-vnode machine that has four CPUs per blade/vnode and that vnode is allocated exclusively to the job, even though the job requested one CPU, it is assigned all 4 CPUs. In this example, resources_assigned reports 4 CPUs, and resources_used reports 1 CPU.

The resources_assigned accounting entry is reported in the S record.

19.6.2.2 Reporting Resources Used by Job

Consumable job resources actually used by the job are recorded in the job's resources_used attribute. Values for resources_used are reported in the c, E, e, u, and z records, and the R record if the job is rerun, but not when the server loses contact with the primary execution host MoM.

You can use hooks to set values for a job's resources_used attribute for custom resources. These custom resources will appear in the accounting log, along with custom resources that are created or set in hooks. Other custom resources will not appear in the accounting log. See "Setting Job Resources in Hooks" on page 49 in the PBS Professional Hooks Guide.

PBS reports resources_used values for string resources that are created or set in a hook as JSON strings in the E record.

- If MoM returns a JSON object (a Python dictionary), PBS reports it in the E record in single quotes:

  ```
  resources_used.<resource_name> = '{ <MoM JSON item value>, <MoM JSON item value>, <MoM JSON item value>, ..}
  ```

  Example: MoM returns { "a":1, "b":2, "c":1,"d": 4} for resources_used.foo_str. We get:

  ```
  resources_used.foo_str='{"a": 1, "b": 2, "c":1,"d": 4}'
  ```

- If MoM returns a value that is not a JSON object, it is reported verbatim in the E record.

  Example: MoM returns "hello" for resources_used.foo_str. We get:

  ```
  resources_used.foo_str="hello"
  ```

19.6.2.3 Freeing Resources

The resources allocated to a job from vnodes are not released until all of those resources have been freed by all MoMs running the job. The end of job accounting record is not written until all of the resources have been freed. The end entry in the job E record includes the time to stage out files, delete files, and free the resources. This does not change the recorded walltime for the job.

19.6.2.4 Releasing Vnodes

When a job's vnodes are released via `pbs_release_nodes`, PBS writes the c and u records, and at the end of the job, PBS writes the e record.

If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.

19.6.3 Reservation Resource Accounting

The following table shows which reservation resources are recorded in the accounting log, and lists the records where they are recorded:

Table 19-16: Reservation Resources in Accounting Log

Resources	Record	Description
exec_host	B	List of vnodes allocated to the reservation

Resource_List	B	List of resources requested by the reservation. Resources are listed individually as, for example:
		`Resource_List.ncpus = 16`
		`Resource_List.mem = 1048676kb`

resv_nodes	B	Contents of resv_nodes reservation attribute

19.6.4 Platform-specific Resource Accounting Tools

19.6.4.1 Resource Accounting on Cray

Jobs that request only compute nodes are not assigned resources from login nodes. PBS accounting logs do not show any login node resources being used by these jobs.

Jobs that request login nodes are assigned resources from login nodes, and those resources appear in the PBS accounting logs for these jobs.

PBS performs resource accounting on the login nodes, under the control of their MoMs.

19.6.4.1.i Using Cray Resource Utilization Reporting

You can use Cray's Resource Utilization Reporting (RUR) to collect statistics on how compute nodes are used, and then use an execjob_epilogue hook to set custom resource values for each job. These custom resources are recorded in the job's E record. You can get more information on Cray's RUR at http://docs.cray.com. See document S-2393-51.

19.6.4.1.ii Using Comprehensive System Accounting

PBS supports Comprehensive System Accounting (CSA) on Cray machines running CLE 5.2. CSA runs on the compute nodes, under the control of the Cray system.

If CSA is enabled, PBS can request the kernel to write user job accounting data to accounting records. These records can then be used to produce reports for the user.

If PBS finds the CSA shared object libraries, and CSA is enabled, PBS can cause a workload management record to be written for each job. If MoM is configured for CSA support, MoM can issue CSA workload management record requests to the kernel. The kernel writes workload management accounting records associated with the PBS job to the system-wide process accounting file. The default for this file is `/var/csa/day/pacct`.

19.6.4.1.iii CSA Configuration Parameter

pbs_accounting_workload_mgmt <value>

MoM configuration parameter. Controls whether CSA accounting is enabled. The name does not start with a dollar sign. If set to "*1*", "*on*", or "*true*", CSA accounting is enabled. If set to "*0*", "*off*", or "*false*", CSA accounting is disabled. Values are case-insensitive. Default: "*true*"; enabled.

19.6.4.1.iv Requirements for CSA

CSA requires CLE 5.2.

CSA requires CSA support Linux kernel modules.

On the supported platforms, the PBS MoM is CSA-enabled. If CSA workload management and user job accounting are available, PBS can use them.

19.6.4.1.v Configuring MoM for CSA

CSA support is specified in the `pbs_accounting_workload_mgmt` line in MoM's Version 1 configuration file. CSA support is enabled by default; you must explicitly disable it if you want it disabled. If the `pbs_accounting_workload_mgmt` line is absent, CSA is still enabled.

To disable CSA support, modify `$PBS_HOME/mom_priv/config`, by setting `pbs_accounting_workload_mgmt` to *false*, *off*, or *0*.

To enable CSA support, either remove the `pbs_accounting_workload_mgmt` line, or set it to *true*, *on*, or *1*.

After modifying the MoM config file, either restart `pbs_mom` or send it SIGHUP.

19.6.4.1.vi Enabling Kernel CSA Support

In order for CSA user job accounting and workload management accounting requests to be acted on by the kernel, you need to make sure that the parameters CSA_START and WKMG_START in the `/etc/csa.conf` configuration file are set to "*on*" and that the system reflects this. You can check this by running the command:

```
csaswitch -c status
```

To set CSA_START to *"on"*, use the command:

```
csaswitch -c on -n csa
```

To set WKMG_START to *"on"*, use:

```
csaswitch -c on -n wkmg
```

Alternatively, you can use the CSA startup script `/etc/init.d/csa` with the desired argument (*on*/*off*); see the system's man page for `csaswitch` and how it is used in the `/etc/init.d/csa` startup script.

19.6.5 Changing Resource Values Reported in Accounting Logs

You can use an execution hook to set a value for resources_used, and this value is then recorded in the accounting log. Bear in mind that by the time an execjob_end hook runs, it's too late to change the accounting log; it's already written.

19.7 Options, Attributes, and Parameters Affecting Accounting

19.7.1 Options to `pbs_server` Command

-A <*accounting file*>

Specifies an absolute path name for the file to use as the accounting file. If not specified, the file is named for the current date in the `PBS_HOME/server_priv/accounting` directory.

Format: *String*

19.7.2 Options to `qsub` Command

-A <*accounting string*>

Accounting string associated with the job. Used for labeling accounting data and/or fairshare. Sets job's Account_Name attribute to <accounting string>.

Format: *String*

-W release_nodes_on_stageout=<value>

When set to *True*, all of the job's vnodes are released when stageout begins.

Cannot be used with vnodes tied to Cray X* series systems.

When the cgroups hook is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

The job's stageout attribute must be set for the release_nodes_on_stageout attribute to take effect.

19.7.3 Options to `qalter` Command

-A <*new accounting string*>

Replaces the accounting string associated with the job. Used for labeling accounting data and/or fairshare. Sets job's Account_Name attribute to *<new accounting string>*. This attribute cannot be altered once the job has begun execution.

Format: *String*

-W release_nodes_on_stageout=<value>

When set to *True*, all of the job's vnodes are released when stageout begins.

Cannot be used with vnodes tied to Cray X* series systems.

When cgroups is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

The job's stageout attribute must be set for the release_nodes_on_stageout attribute to take effect.

19.7.4 Job Attributes

Account_Name

PBS jobs have an attribute called Account_Name. You can use it however you want; PBS does not interpret it.

PBS does not use this accounting string by default. However, you can tell PBS to use the job's accounting string as the owner of the job for fairshare purposes. See section 4.9.19, “Using Fairshare”, on page 138. PBS accepts the string passed by the shell as is.

Any character is allowed if the string is enclosed in single or double quotes. When you specify this string on the command line to a PBS utility or in a directive in a PBS job script, escape any embedded white space by enclosing the string in quotes.

You can set the initial value of a job's Account_Name attribute via the `-A <account string>` option to `qsub`. You can change the value of a job's Account_Name attribute via the `-A <new account string>` option to `qalter`.

Can be read and set by user, Operator, Manager.

Format: String that can contain any character

Default value: none.

Python attribute value type: *str*

accounting_id

Accounting ID for tracking accounting data not produced by PBS. May be used for a CSA job ID or job container ID.

Can be read by User, Operator, Manager.

No default value.

Format: *String*

Python attribute value type: *str*

alt_id

For a few systems, the session ID is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it is recorded in the end-of-job accounting record. May contain white spaces.

On Windows, holds PBS home directory.

Can be read by User, Operator, Manager.

No default value.

Format: *String*

Python attribute value type: *str*

release_nodes_on_stageout

When set to *True*, all of the job's vnodes are released when stageout begins.

Cannot be used with vnodes tied to Cray X* series systems.

When the cgroups hook is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

19.7.5 MoM Parameters

$logevent <*mask*>

Sets the mask that determines which event types are logged by `pbs_mom`. To include all debug events, use *0xffffffff*. See "Log Levels" on page 547 of the PBS Professional Reference Guide.

19.7.5.1 Cray-only MoM Initialization Values

pbs_accounting_workload_mgmt <*value*>

Controls whether CSA accounting is enabled. Name does not start with dollar sign. If set to "*1*", "*on*", or "*true*", CSA accounting is enabled. If set to "*0*", "*off*", or "*false*", accounting is disabled.

Default: "*true*"; enabled.

19.8 Accounting Caveats and Advice

If you use the cgroups hook to manage subsystems and create child vnodes, you get accurate accounting. If not, accuracy depends on whether or not your MPI is integrated with PBS.

19.8.1 Integrate MPIs for Accurate Accounting

PBS Professional is integrated with several implementations of MPI. When PBS is integrated with an MPI, PBS can track resource usage, control jobs, clean up job processes, and perform accounting for all of the tasks run under the MPI.

When PBS is not integrated with an MPI, PBS can track resource usage, clean up processes, and perform accounting only for processes running on the primary host. This means that accounting and tracking of CPU time and memory aren't accurate, and job processes on sister hosts cannot be signaled.

Follow the steps in section 10.1, “Integration with MPI”, on page 451.

19.8.2 MPI Integration under Windows

Under Windows, some MPIs such as MPICH are not integrated with PBS. With non-integrated MPIs, PBS is limited to tracking resources, signaling jobs, and performing accounting only for job processes on the primary vnode.

19.8.3 Using Hooks for Accounting

19.8.3.1 Use Hooks to Record Job Information

For each job, you can use execjob_prologue, execjob_epilogue, or exechost_periodic hooks to set resources_used values for custom resources, which are then recorded in the job's E record.

19.8.3.2 Use Hooks to Manage Job Accounting String

You can use a hook to assign the correct value for each job's Account_Name attribute. This can be useful both for your accounting records and if you use the job's Account_Name attribute as the job's owner for fairshare. See the PBS Professional Hooks Guide and section 4.9.19, “Using Fairshare”, on page 138.

20 Mixed Linux-Windows Operation

20.1 Introduction to Mixed Linux-Windows Operation

You can add Windows execution and client hosts to a Linux PBS complex, creating a *mixed-mode complex*. These Windows hosts must be in an Active Directory domain. Linux systems must use MUNGE rather than reserved-port authentication, and Windows users must be active directory users. Communication should be encrypted using TLS for improved security. The server needs to authenticate both Linux and Windows users. We describe how to set up a mixed-mode complex in this section.

On Windows, MoM automatically sets resources_available.arch to "windows" for the local vnode. Users submitting Windows jobs can request Windows hosts by specifying "windows" for the arch resource. For example:

```
qsub -lselect=1:arch=windows ...
```

Users submitting Windows jobs must cache their passwords at each execution and client host before submitting jobs, and each time their password changes. Job submitters use the pbs_login command to cache their passwords.

20.1.1 Caveats for Mixed Linux-Windows Operation

- You cannot submit a Linux job from a Windows client
- Group limits are not enforced for Windows jobs; for example, "set queue max_queued_res.ncpus = [g:<group name> = <limit>]" has no effect

20.2 Configuration

1. Start with a normal working Linux PBS complex. See the *PBS Professional Installation & Upgrade Guide*.

20.2.1 Configure Authentication

1. Configure MUNGE authentication for Linux clients and pwd for Windows clients.

   The default reserved-port (resvport) method is not secure for mixed-mode operation, because Windows does not have a concept of reserved ports. Follow the instructions in section 8.4, “Authentication for Daemons & Users”, on page 378. After you have integrated MUNGE, put this in the server's /etc/pbs.conf file:

   ```
   PBS_SUPPORTED_AUTH_METHODS=munge, pwd
   ```

2. Restart the PBS daemons. On each Linux host:

   ```
   systemctl restart pbs
   ```

or

```
<path to start/stop script> pbs restart
```

3. Make sure that you can submit jobs and that hooks work.

20.2.2 Windows Hosts and Users in Active Directory Domain

1. Make sure the new Windows execution and client hosts are part of the same Windows Active Directory domain.
2. Make sure that Active Directory Authentication works: verify that the users added to the AD domain can log in to all the Windows hosts.

20.2.3 Allow Linux Authentication of Windows Active Domain Users

You can use various methods to allow Linux hosts to authenticate Windows Active Domain users. We show an example using SSSD here.

1. On the Linux host running the server, and any hosts running extra comms, configure `sssd` so that the users of the Windows domain can log in to the Linux host on which `pbs_server` and `sssd` run. For an example, see section 8.4.5, “Configuring SSSD”, on page 380. For information on configuring `sssd`, see https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/windows_integration_guide/index#sssd-ad-proc and https://access.redhat.com/articles/3023951.

   If you want the Linux host to automatically create a home directory for an Active Directory user if that home directory does not exist at login, you may have to set SELinux to permissive mode. This is optional.

2. Verify that `sssd` is correctly configured.

   a. Run the following commands:

   ```
   id <username>
   su - <username>
   <password>
   ```

   b. As a Windows domain user, `ssh` to the Linux host running `sssd`

20.2.4 Configure User Authorization

We recommend setting flatuid to *False* for the PBS complex, so that users need a `.rhosts` file to enable authorization. For example, to configure the `.rhosts` file so that user User1 can submit jobs from submission host Winclient1, make sure that there is a file named `.rhosts` in User1's home directory on the server host, and that this file contains the following entry:

```
Winclient1 User1
```

20.2.5 Install PBS on Windows Hosts

1. Install MoMs on your Windows execution hosts, and install the PBS client commands on your Windows client hosts. See section 3.7, "Installing PBS on Windows Hosts", on page 37.
2. Configure the remote file copy mechanism to be used by Windows execution hosts:
   - If you will use `scp` for your remote file copy mechanism, configure passwordless `ssh`; see section 15.6.7.1, "Enabling Passwordless Authentication", on page 565.
   - If you will use the $usecp MoM parameter to specify your remote file copy mechanism, you do not need to configure passwordless `ssh`, unless it is required by the MPI implementation you are using.
3. Create the parent vnode for each Windows host; see section 3.3.3, "Creating the Parent Vnode", on page 40:

   ```
   qmgr -c "create node <name of parent vnode>"
   ```

20.2.6 Set Up TLS Encryption

Configure TLS encryption for daemon-daemon communication. For Windows authentication to work securely, we strongly recommend using TLS encryption in the complex. To set up TLS encryption, we need a CA certificate and TLS certificate key pair generated from any system with `openssl` set up. We will use the same key pair on all the server and comm hosts. For an example of how to configure PBS for TLS encryption, see section 8.5.2.2, "Example of Configuring PBS for TLS Encryption", on page 388.

20.3 Troubleshooting Mixed Linux-Windows Complex

- Job comment contains "failed to Impersonate Logged On User on <hostname>:job has bad password"

  The user might not be allowed to log on locally according to the local or global policy settings; MoM is unable to impersonate the job submitter in order to run the job. Check the Windows policy settings; make sure the user is allowed to log in to the execution host.
- Failed to send auth request

  ```
  auth: error returned: 15029
  auth: Failed to send auth request
  No support for requested service.
  cannot connect to server pbsserver3 (errno=15029)
  ```

  Check whether the server's PBS_SUPPORTED_AUTH_METHOD parameter includes pwd; add it if not. Restart the server and try again.
- User not known to the underlying authentication module

  ```
  auth: error returned: 15019
  auth: PAM authentication failed for testuser2 with error: User not known to the underlying
      authentication module
  ```

When this error occurs despite testuser2 existing and giving the correct password, it may be due to SSSD settings. One reason could be that 'use_fully_qualified_domain' is *True* in the SSSD settings. Change that to *False* and verify that the user can log in from the server using the following commands, in our example:

```
id username:
[pbsadmin@pbsserver3 ~]$ id testuser4
uid=775213102(testuser4) gid=775200513(domain users) groups=775200513(domain users)
su  - username:
[pbsadmin@pbsserver3 ~]$ su - testuser2
Password:
Last login: Mon Apr 20 13:40:32 UTC 2020 on pts/1
[testuser2@pbsserver3 ~]$
```

- Windows MoM fails to register

  After installing pbs_mom on Windows and executing win_postinstall.py, even though the postinstall script completes successfully, the server still shows state of the new Windows vnode as "state-unknown, down". In the comm logs there are repeated messages of authentication failure for the service account.

  Check for and delete a stale password file named ".pbs_cred.CR" from the home directory of the PBS service account used to run the Windows MoM.

- Vnode does not go to *free* state, and the following error message appears in the server logs

  ```
  init_pam;libpam.so not found
  validate_auth_data;Failed to initialize the library
  tcp_pre_process;Failed to initialize the library
  wait_request;process socket failed
  ```

  Check whether the pam library is installed. If it is installed, make sure that the PATH variable points to the location where the library is installed. The library name might be libpam.so.*. In this case, create a soft link to the library as shown in the example below:

  ```
  ln -s /usr/lib64/libpam.so.0.83.1 /usr/lib64/libpam.so
  ```

- Files fail to stage out using scp.exe from C:\Windows\System32\OpenSSH

  With OpenSSH (scp.exe) installed in the C:\Windows\System32 folder, stage out failures are observed as shown:

  ```
  12/27/2019 00:25:58;0100;PBS_stage_file;Job;11.mixlin;User pbsuser passworded
  12/27/2019 00:25:58;0080;PBS_stage_file;Job;sys_copy;CreateProcessAsUser(928,
      C:/Windows/System32/OpenSSH-Win64/scp.exe -Brv \PROGRA~2\PBS\home\spool\11.mixlin.OU
      "pbsuser@mixlin:/home/pbsuser/STDIN.o11") under acct pbsuser
      wdir=C:\Users\pbsuser\Documents\PBS Pro}}
  12/27/2019 00:25:58;0080;PBS_stage_file;Fil;sys_copy;command:
      C:/Windows/System32/OpenSSH-Win64/scp.exe -Brv C:/PROGRA~2/PBS/home/spool/11.mixlin.OU
      pbsuser@mixlin:/home/pbsuser/STDIN.o11 status=10002, try=1
  ```

  This is caused by Windows WOW64 filesystem redirection, since PBS is a 32-bit application. See Microsoft article https://docs.microsoft.com/en-us/windows/win32/winprog64/file-system-redirector.

  Replace "Sysetm32" with "Sysnative" in the PBS_SCP parameter in pbs.conf and restart the PBS_MOM service. You do not need to move or reinstall OpenSSH from C:\Windows\System32 path.

21 Problem Solving

Additional information is always available online at the PBS website, www.pbsworks.com. The last section in this chapter tells you how to get additional assistance from the PBS Support staff.

21.1 Debugging Tools

21.1.1 Debugging Commands

The following commands will provide helpful debugging information: `qstat`, `tracejob`, `qmgr`, and `pbsnodes`.

21.1.2 Setting Corefile Size

To set the size of the core file for a PBS daemon, you can set PBS_CORE_LIMIT in `pbs.conf`. Set this on the machine where the daemon runs. This can be set to an integer number of bytes or to the string "unlimited". If this is unset, the limit is inherited from the shell environment, which you can check via `uname -c`.

21.1.3 Using the debuginfo RPM Package

PBS is shipped with debuginfo package(s). When you unzip the PBS product download package the debuginfo package(s) can be found alongside the other PBS packages containing the server etc.

Normally, you do not need to install any debuginfo package(s). You only need to install the debuginfo package(s) when the support team recommends doing so to aid in diagnosing a problem. The contents of the debuginfo package files are automatically installed in the default location for your Linux distribution, typically under `/usr/lib/debug` and `/usr/src/debug`.

The debuginfo package(s) names are platform-dependent. When you install a debuginfo package, nothing additional is installed in `PBS_HOME` or `PBS_EXEC`.

21.1.4 Sending Daemon Execution Recordings to Altair

Altair support can analyze recordings of daemon execution in order to diagnose a problem. If support asks you to create recordings, you can use Undo Live Recorder to capture execution recordings of the server, scheduler(s), MoM(s), or comm(s), then send them to Altair support.

To trigger a daemon to begin or end recording, send it a SIGUSR1:

```
kill SIGUSR1 <daemon PID>
```

The scheduler starts or stops recording only when a scheduling cycle is triggered.

Each daemon writes its recordings in the directory specified in the PBS_LR_SAVE_PATH parameter in its `/etc/pbs.conf` file. This path defaults to `$PBS_HOME/spool`, which is created automatically at install time. You can specify the directory where each daemon writes its recordings; if you change the value of PBS_LR_SAVE_PATH, you need to HUP the daemon so that it rereads its configuration file, and creates the new directory.

The filename for each recording is <daemon name>_<yymmddHHMM>.undo

For example, on November 28, 2020, at 8:14 a.m., a server named Server@examplehost will write a record named "Server@examplehost_202011280814.undo".

When a daemon starts or stops a recording, it writes a log message at log level LOG_INFO. The message contains the date and time, the daemon name, whether the recording was started or stopped, and the filename of the recording.

Contact information is available at www.altair.com and in "Technical Support" on page iii.

21.1.4.1 Caveats and Restrictions

Using Undo Live Recorder is supported on Linux on x86_64 platforms only.

21.1.5 Finding PBS Version Information

Use the `qstat` command to find out what version of PBS Professional you have.

```
qstat -fB
```

In addition, each PBS command will print its version information if given the `--version` option. This option cannot be used with other options.

21.1.6 Troubleshooting and Hooks

PBS is shipped with tools for debugging hooks. See "Debugging Hooks", on page 159 of the PBS Professional Hooks Guide.

You may wish to disable hook execution in order to debug PBS issues. To verify whether hooks are part of the problem, disable each hook by setting its enabled attribute to *False*.

21.2 Security and Permissions Problems

21.2.1 Directory Permission Problems

If for some reason the access permissions on the PBS file tree are changed from their default settings, a component of the PBS system may detect this as a security violation, and refuse to execute. If this is the case, an error message to this effect will be written to the corresponding log file. You can run the `pbs_probe` command to check (and optionally correct) any directory permission (or ownership) problems. See "pbs_probe" on page 81 of the PBS Professional Reference Guide for details on usage of the `pbs_probe` command.

21.2.1.1 Correcting Permissions Problems on Linux

You can use the `pbs_probe` command to detect and repair file and directory permissions problems. You can run `pbs_probe` in report mode or fix mode; in report mode, it reports the errors found; in fix mode, it attempts to fix detected problems, and reports any problems it could not fix.

To fix permissions errors, log into the host you wish to check, and run the following command:

```
pbs_probe -f
```

See the `pbs_probe(8B)` manual page.

21.2.1.2 Correcting Permissions Problems on Windows

You can use the `pbs_mkdirs` command to correct file and directory permissions problems on Windows. The command checks and if necessary repairs the permissions of configuration files such as `pbs_environment` and `mom_priv/config`. You should run the `pbs_mkdirs` command only while the PBS MoMs are stopped.

To repair permissions on an execution host, log into the host and run the following commands:

```
net stop pbs_mom
pbs_mkdirs mom
net start pbs_mom
```

21.3 Troubleshooting Jobs

21.3.1 Job Held Due to Invalid Password

If a job fails to run due to an invalid password, then the job is held with hold type *p* (bad password), its comment field updated with why it failed, and an email is sent to the owner for remedy action. Root or administrator can release the hold via qrls. See "qhold" on page 147 of the PBS Professional Reference Guide and "qrls" on page 180 of the PBS Professional Reference Guide.

21.3.2 Requeueing a Job "Stuck" on a Down Vnode

PBS Professional will detect if a vnode fails when a job is running on it, and will automatically requeue and schedule the job to run elsewhere. If the user marked the job as "not rerunnable" (i.e. via the `qsub -r n` option), then the job will be deleted rather than requeued. If the affected vnode is on the primary execution host, the requeue will occur quickly. If it is another vnode in the set assigned to the job, it could take a few minutes before PBS takes action to requeue or delete the job. However, if the auto-requeue feature is not enabled, or if you wish to act immediately, you can manually force the requeueing and/or rerunning of the job. See section 9.6.2, "Node Fail Requeue: Jobs on Failed Vnodes", on page 443.

If you wish to have PBS simply remove the job from the system, use the "`-Wforce`" option to `qdel:`

qdel -Wforce <job ID>

If instead you want PBS to requeue the job, and have it immediately eligible to run again, use the "`-Wforce`" option to `qrerun`

qrerun -Wforce <job ID>

See "Job Input & Output Files", on page 31 of the PBS Professional User's Guide.

21.3.3 Job Cannot be Executed

If a user receives a mail message containing a job ID and the line "Job cannot be executed", the job was aborted by MoM when she tried to place it into execution. The complete reason can be found in one of two places, MoM's log file or the standard error file of the user's job. If the second line of the message is "See Administrator for help", then MoM aborted the job before the job's files were set up. The reason will be noted in MoM's log. Typical reasons are a bad user/group account, checkpoint/restart file, or a system error. If the second line of the message is "See job standard error file", then MoM had created the job's file and additional messages were written to standard error. This is typically the result of a bad resource request.

21.3.4 Running Jobs with No Active Processes

On very rare occasions, PBS may be in a situation where a job is in the *Running* state but has no active processes. This should never happen as the death of the job's shell should trigger MoM to notify the server that the job exited and end-of-job processing should begin. If this situation is noted, PBS offers a way out. Use the `qsig` command to send SIGNULL, signal 0, to the job. If MoM finds there are no processes then she will force the job into the exiting state. See "qsig" on page 192 of the PBS Professional Reference Guide.

21.3.5 Jobs that Can Never Run

If backfilling is being used, the scheduler looks at the job being backfilled around and determines whether that job can never run.

If backfilling is being used, the scheduler determines whether that job can or cannot run now, and if it can't run now, whether it can ever run. If the job can never run, the scheduler logs a message saying so.

The scheduler only considers the job being backfilled around. That is the only job for which it will log a message saying the job can never run.

This means that a job that can never run will sit in the queue until it becomes the most deserving job. Whenever this job is considered for having small jobs backfilled around it, the error message "resource request is impossible to solve: job will never run" is printed in the scheduler's log file. If backfilling is not being used, this message will not appear.

If backfilling is not being used, the scheduler determines only whether that job can or cannot run now. The scheduler won't determine if a job will ever run or not.

21.3.6 Job Comments for Problem Jobs

PBS can detect when a job cannot run with the current unused resources and when a job will never be able to run with all of the configured resources. PBS can set the job's comment attribute to reflect why the job is not running.

If the job's comment starts with "Can never run", the job will never be able to run with the resources that are currently configured. This can happen when:

- A job requests more of a consumable resource than is available on the entire complex
- A job requests a non-consumable resource that is not available on the complex

For example, if there are 128 total CPUs in the complex, and the job requests 256 CPUs, the job's comment will start with this message.

If the job's comment starts with "Not running", the job cannot run with the resources that are currently available. For example, if a job requests 8 CPUs and the complex has 16 CPUs but 12 are in use, the job's comment will start with this message.

You may see the following comments. R is for "Requested", A is for "Available", and T is for "Total":

```
"Not enough free nodes available"
"Not enough total nodes available"
"Job will never run with the resources currently configured in the complex"
"Insufficient amount of server resource <resource name> (R | A | T | <requested value>
   !=<available values for requested resource>)
"Insufficient amount of queue resource <resource name> (R | A | T | <requested value> !=<available
   values for requested resource>)
"Error in calculation of start time of top job"
"Can't find start time estimate"
```

The "Can Never Run" prefix may be seen with the following messages:

"Insufficient amount of resource <resource name> (R | A | T | <requested value> !=<available values for requested resource>)"

"Insufficient amount of Server resource <resource name> (R | A | T | <requested value> !=<available values for requested resource>)"

"Insufficient amount of Queue resource <resource name> (R | A | T | <requested value> !=<available values for requested resource>)"

"Not enough total nodes available"

"can't fit in the largest placement set, and can't span psets"

21.3.7 Bad UID for Job Execution

For a job to be accepted by the PBS server, the user at the submitting host must pass an ruserok() test.

From the RCMD(3) man page:

> The iruserok() and ruserok() functions take a remote host's IP address or name, respectively, two user names and a flag indicating whether the local user's name is that of the superuser. Then, if the user is NOT the superuser, it checks the /etc/hosts.equiv file. If that lookup is not done, or is unsuccessful, the .rhosts in the local user's home directory is checked to see if the request for service is allowed.

If this file does not exist, is not a regular file, is owned by anyone other than the user or the superuser, or is writable by anyone other than the owner, the check automatically fails. Zero is returned if the machine name is listed in the hosts.equiv file, or the host and remote user name are found in the .rhosts file; otherwise iruserok() and ruserok() return *-1*. If the local domain (as obtained from gethostname(2)) is the same as the remote domain, only the machine name need be specified.

If the server attribute flatuid is set to true, this test is skipped and the job is accepted based on the submitting users name alone (with obvious security implications).

You can run the following command:

```
Qmgr: set server flatuid=true
```

Flatuid or not, to run as a user other than the job owner (the submitter) you must have authorization to do so. Otherwise, any user could run a job as any other user. You authorize for userA to run a job as userB the same way you authorize userA@host1 to run a job as userA on host2 when flatuid is Not SET, i.e. see .ruserok() and .rhosts.

Here is a test program to see if ruserok passes for a given user and host. There are two use cases:

- User submitting job from remote host to server getting unexpected "Bad UID" message. That is, user doesn't have access when he thinks he should.
- User(s) can delete, etc other user(s) jobs. That is, one user is able to act as what he thinks is a different user, server sees them as being equivalent.

Build this with "cc ruserok.c -o ruserok"

Usage (run on the PBS server system):

```
ruserok remote_host remote_user1 local_user2
```

where:

remote_host: the host from which the job is being submitted, or where the PBS client command is issued

remote_user1: the username of the user submitting the job, or issuing the client command

loca_user2: the username of the user remote_user1 is trying to submit the job as, or owner of the job that remote_user1 is trying to act on with the client command

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
    int rc;
    char hn[257];
    if (argc != 4)
        { fprintf(stderr, "Usage: %s remote_host remote_user1 local_user2\n", argv[0]); return 1;
    }
    if (gethostname(hn, 256) < 0)
        { perror("unable to get hostname"); return 2; }
    hn[256] = '\0';
    printf("on local host %s, from remote host %s\n", hn, argv[1]);
    rc = ruserok(argv[1], 0, argv[2], argv[3]);
    if (rc == 0)
        printf("remote user %s is allowed access as local user %s\n", argv[2], argv[3]);
    else
        printf("remote user %s is denied access as local user %s\n", argv[2], argv[3]);
    return 0;
}
```

21.3.8 Windows: Bad UID for Job Execution

If, when attempting to submit a job to a remote server, `qsub` reports:

```
BAD uid for job execution
```

Then you need to add an entry in the remote system's `.rhosts` or `hosts.equiv` pointing to your Windows machine. Be sure to put in all hostnames that resolve to your machine. See section 2.3.6, “User Authorization Under Windows”, on page 15.

If remote account maps to an Administrator-type account, then you need to set up a `.rhosts` entry, and the remote server must carry the account on its acl_roots list.

21.3.9 New Jobs Not Running

If PBS loses contact with the Altair License Server, any jobs currently running will not be interrupted or killed. The PBS server will continually attempt to reconnect to the license server, and re-license the assigned vnodes once the contact to the license server is restored.

No new jobs will run if PBS server loses contact with the ALM license server.

21.3.10 Job Stuck in Exiting State

A job can be stuck in the Exiting state if the user submits the job from a directory where the user does not have write access. You can forcefully delete the job:

21.3.10.1 qdel -Wforce <job ID>

21.4 Troubleshooting Daemons

21.4.1 Server Host Bogs Down After Startup

If the server host becomes unresponsive a short time after startup, the server may be trying to contact the wrong license server.

21.4.1.1 Symptoms

15 seconds to one or two minutes after you start the PBS server, the system becomes unresponsive.

21.4.1.2 Problem

The problem may be caused by the pbs_license_info server attribute pointing to an old FLEX license server. This attribute should point to the new ALM license server. See the *PBS Works Licensing Guide*.

21.4.1.3 Treatment

On some Linux systems, the effects of memory starvation on subsequent responsiveness may be long-lasting. Therefore, instead of merely killing and restarting the PBS server, we recommend rebooting the machine.

Take the following steps:

1. Reboot the machine into single-user mode.
2. Determine the correct value for pbs_license_info and set the PBS_LICENSE_INFO entry in `pbs.conf` to this value.
3. Reboot, or change `runlevel` to multi-user.
4. Using `qmgr`, set the pbs_license_info server attribute to the correct value:

```
# qmgr -c "set server pbs_license_info = <port>@<license server hostname>"
# qmgr -c "set server scheduling= true"
```

5. Stop the PBS server process.
6. Continue normally.

21.4.2 Server Does Not Start

The server may not start due to problems with the data service. Call PBS technical support; see "Technical Support" on page iii. For more on the PBS data service, see "pbs_dataservice" on page 60 of the PBS Professional Reference Guide.

21.4.3 Primary Server Periodically Restarting

If the primary server keeps restarting, an unknown secondary server may be contacting it. This can happen when PBS_PRIMARY and PBS_SECONDARY are missing from pbs.conf, but a secondary server has been started.

21.4.4 PBS Data Service Does Not Start

- You may need to create the data service management account. This must be creating before installing PBS. See "Create PBS Data Service Management Account" on page 23 in the PBS Professional Installation & Upgrade Guide.
- If you see an error message saying "PBS data service is running on another host - cannot start", there may be a problem with the lock file in $PBS_HOME/dataservice/pbs_dblock:
  - Problem during failover between two hosts, where the primary host still has a lock on the file
  - Ungraceful shutdown, where the primary host has an incorrectly, still-locked, lock file; look at the primary server host.
  - File system issues that interfere with the locking, unlocking, and/or access, of the lock file.

21.4.5 Server Dies Inexplicably

Check the data service. When the data service dies, the server automatically goes down too.

21.4.6 Data Service Running When PBS Server is Down

You can use the pbs_dataservice command to stop the data service. See "pbs_dataservice" on page 60 of the PBS Professional Reference Guide.

21.4.7 Scheduler Cannot Reliably Contact Server

If you see a series of 15031 errors, this can happen when PBS_PRIMARY and PBS_SECONDARY are missing from pbs.conf, but a secondary server has been started.

21.4.8 PBS Daemon Will Not Start

If the PBS server, MoM, or scheduler fails to start up, it may be refusing to start because it has detected permissions problems in its directories or on one or more of its configuration files, such as pbs_environment or mom_priv/config.

21.4.9 Troubleshooting Windows Daemon Problems

21.4.9.1 Windows: MoMs Do Not Start

- In the case where the PBS daemons, the Active Directory database, and the domain controller are all on the same host, some PBS MoMs may not start up immediately. If the Active Directory services are not running when the PBS MoMs are started, the MoMs won't be able to talk to the domain controller. This can prevent the PBS MoMs from starting. As a workaround, wait until the host is completely up, then retry starting the failing MoM.

Example:

```
net start pbs_mom
```

- In a domained environment, if the PBS service account is a member of any group besides "Domain Users", the install program will fail to add the PBS service account to the local Administrators group on the install host. Make sure that the PBS service account is a member of only one group, "Domain Users" in a domained environment.
- If the MoM fails to start up because of permission problems on some of its configuration files like `pbs_environment` or `mom_priv/config`, then correct the permission by running:

  ```
  pbs_mkdirs mom
  ```

21.5 Troubleshooting Vnodes

21.5.1 Vnodes Down

The PBS server determines the state of hosts (*up* or *down*), by communicating with MoM on the host. The state of vnodes may be listed by two commands: `qmgr` and `pbsnodes`

```
Qmgr: list node @active
pbsnodes -a
Node jupiter  state = state-unknown, down
```

A vnode in PBS may be marked "*down*" in one of two substates. For example, the state above of vnode "jupiter" shows that the server has not had contact with MoM since the server came up. Check to see if a MoM is running on the vnode. If there is a MoM and if the MoM was just started, the server may have attempted to poll her before she was up. The server should see her during the next polling cycle in 10 minutes. If the vnode is still marked "state-unknown, down" after 10+ minutes, either the vnode name specified in the server's node file does not map to the real network hostname or there is a network problem between the server host and the vnode.

If the vnode is listed as:

```
pbsnodes -a
Node jupiter state = down
```

then the server has been able to ping MoM on the vnode in the past, but she has not responded recently. The server will send a "ping" PBS message to every free vnode each ping cycle, 10 minutes. If a vnode does not acknowledge the ping before the next cycle, the server will mark the vnode down.

21.5.2 Bad Vnode on Startup

If, when the server starts up, one or more vnodes cannot be resolved, the server marks the bad vnode(s) in state "*state-unknown, down*".

21.6 Troubleshooting Client Commands

21.6.1 Windows: Client Commands Slow

PBS caches the IP address of the local host, and uses this to communicate between the daemons. If the cached IP address is invalidated, PBS can become slow. In both scenarios, jobs must be killed and restarted.

21.6.1.1 Scenario: Wireless Router, DHCP Enabled

The system is connected to a wireless router that has DHCP enabled. DHCP returned a new IP address for the server short name, but DNS is resolving the server full name to a different IP address.

The IP address and server full name have become invalid due to the new DHCP address. PBS has cached the IP address of the server full name.

Therefore, the PBS server times out when trying to connect to the scheduler and local MoM using the previously cached IP address. This makes PBS slow.

Symptom:

1. PBS is slow.
   a. Server logs show "Could not contact scheduler".
   b. `pbsnodes -a` shows that the local node is down.
2. First IP addresses returned below don't match:

```
cmd.admin> pbs_hostn -v <server_short_name>
cmd.admin> pbs_hostn -v <server_full_name>
```

Workaround: cache the correct new IP address of the local server host.

1. Add the address returned by `pbs_hostn -v <server_short_name>` (normally the DHCP address) to `%WINDIR%\system32\drivers\etc\hosts file` as follows:
   `<DHCP address> <server_full_name> <server_short_name>`
2. Restart the MoM:

```
cmd.admin> net stop pbs_mom
cmd.admin> net start pbs_mom
```

21.6.2 Windows: qstat Errors

If the `qstat` command produces an error such as:

```
illegally formed job identifier.
```

This means that the DNS lookup is not working properly, or reverse lookup is failing. Use the following command to verify DNS reverse lookup is working

```
pbs_hostn -v hostname
```

If however, `qstat` reports "No Permission", then check `pbs.conf`, and look for the entry "PBS_EXEC". `qstat` (in fact all the PBS commands) will execute the command "`PBS_EXEC\sbin\pbs_iff`" to do its authentication. Ensure that the path specified in `pbs.conf` is correct.

21.6.3 Clients Unable to Contact Server

If a client command (such as `qstat` or `qmgr`) is unable to connect to a server there are several possibilities to check. If the error return is 15034, "No server to connect to", check (1) that there is indeed a server running and (2) that the default server information is set correctly. The client commands will attempt to connect to the server specified on the command line if given, or if not given, the server specified by SERVER_NAME in `pbs.conf`.

If the error return is `15007`, "`No permission`", check for (2) as above. Also check that the executable `pbs_iff` is located in the search path for the client and that it is setuid root. Additionally, try running `pbs_iff` by typing:

```
pbs_iff -t server_host 15001
```

Where *server_host* is the name of the host on which the server is running and 15001 is the port to which the server is listening (if started with a different port number, use that number instead of 15001). Check for an error message and/or a non-zero exit status. If pbs_iff exits with a non-zero status, either the server is not running or was installed with a different encryption system than was pbs_iff.

21.7 Troubleshooting PBS Licenses

21.7.1 Wrong License Server: Out of Memory

If you run out of memory shortly after startup, the server may be looking for the wrong license server. See section 21.4.1, "Server Host Bogs Down After Startup", on page 669.

21.7.2 Unable to Connect to License Server

If PBS cannot contact the license server, the server will log a message:

"Unable to connect to license server at pbs_license_info=..."

If the license server location is incorrectly initialized (e.g. if the host name or port number is incorrect), PBS may not be able to pinpoint the misconfiguration as the cause of the failure to reach a license server.

If PBS cannot detect a license server host and port when it starts up, the server logs an error message:

"Did not find a license server host and port (pbs_license_info=<X>). No external license server will be contacted"

21.7.3 Insufficient Minimum Licenses

If the PBS server cannot get the number of licenses specified in pbs_license_min from the license server, the server will log a message:

"checked-out only <X> CPU licenses instead of pbs_license_min=<Y> from license server at host <H>, port <P>. Will try to get more later."

21.7.4 Wrong Type of License

If the PBS server encounters a proprietary license key that is of the wrong type, the server will log the following message:

"license key #1 is invalid: invalid type or version".

21.8 Crash Recovery

PBS daemons could terminate unexpectedly either because the host machine stops running or because the daemon itself stops running. The daemon may be killed by mistake, or may (rarely) crash. The server may terminate if the filesystem runs out of space.

21.8.1 Recovery When Host Machine Stops

If the host machine stops running, no special steps are required, since PBS will be started when the machine starts.

21.8.1.1 Execution Host Stops

If the host machine is an execution host, any jobs that were running on that host were terminated when the machine stopped, and when MoM is restarted, she will report to the server that those jobs are dead, and begin normal activity. The server will automatically restart any jobs that can be restarted.

Shutting down one host of a multi-host job will cause that job to be killed. The job will have to be rerun; restarting the MoM on the stopped host with the -p option will not help the job. See "pbs_mom" on page 71 of the PBS Professional Reference Guide.

21.8.1.2 Server/scheduler/communication Host Stops

If the host machine is the server/scheduler/communication host, no data is lost and no jobs are lost, because the server writes everything to disk. The server is restarted automatically upon machine startup.

The scheduler is started automatically upon machine startup. The scheduler starts fresh each cycle, so it does not lose data.

21.8.2 Recovery When Daemon Stops

For more detailed information on starting and stopping PBS, see "Starting & Stopping PBS on Linux" on page 159 in the PBS Professional Installation & Upgrade Guide.

21.9 Other Troubleshooting

21.9.1 Problem With Dynamic Resource

If you need to debug a dynamic resource being supplied by an external script, it may help to follow these steps:

1. Set the scheduler's log_events parameter to 4095 (everything is logged)
   ```
   qmgr -c "set sched <scheduler name> log_events = 4095"
   ```
2. Send a SIGHUP to the scheduler (pbs_sched)
3. The scheduler log will contain the value the scheduler reads from the external script

21.9.2 Cannot Create Formula or Hook

You must run qmgr at the server host when operating on the server's job_sort_formula attribute or on hooks. For example, attempting to create the formula at another host will result in the following error:

```
qmgr obj= svr=default: Unauthorized Request  job_sort_formula
```

21.9.3 Windows: PBS Cannot Locate Configuration File

If PBS is installed on a hard drive other than C:, it may not be able to locate the pbs.conf global configuration file. If this is the case, PBS will report the following message:

```
E:\Program Files\PBS\exec\bin>qstat -
pbsconf error: pbs conf variables not found:
PBS_HOME PBS_EXEC
No such file or directory
qstat: cannot connect to server UNKNOWN (errno=0)
```

To correct this problem, set PBS_CONF_FILE to point pbs.conf to the right path. Normally, during PBS Windows installation, this would be set in system autoexec.bat which will be read after the Windows system has been restarted. Thus, after PBS Windows installation completes, be sure to reboot the Windows system in order for this variable to be read correctly.

21.9.4 Filesystem Runs Out of Space

If your filesystem has run out of space, the server may experience errors or may crash. If the server is still running, you need only to free up enough space. If the server has crashed, you must restart it. See "Server: Starting, Stopping, Restarting" on page 163 in the PBS Professional Installation & Upgrade Guide.

21.9.5 Unrecognized Timezone Variable

Problem: you see this message:

```
pbs_rsub: Bad time specification(s)
```

Reason: The time zone is not specified correctly in PBS_TZID. On later Linux updates, the system's zoneinfo files may have some countries represented under different names from those in previous releases. For example, *Asia/Calcutta* has been replaced by *Asia/Kolkata*.

In order to create reservations, the PBS server must recognized the PBS_TZID environment variable at the submission host. The appropriate zone location for the submission host can be obtained from the machine on which the PBS Professional server is installed.

- On Linux platforms, either use the tzselect command, if it is available, or look in the underlying operating system's zone.tab timezone location file, which may be found under /usr/share/zoneinfo/zone.tab. While the PBS server is running and can contact the execution machine, use the Linux tzselect utility to determine the value for PBS_TZID.
- On all other platforms, look in the list of libical supported zoneinfo locations available under $PBS_EXEC/lib/ical/zoneinfo/zones.tab.

21.10 Getting Help

If the material in the PBS manuals is unable to help you solve a particular problem, you may need to contact the PBS Support Team for assistance. The PBS Professional support team can be reached directly via email and phone; contact information is on the inside front cover of each manual.

Index

$alps_release_jitter AG-476
$alps_release_wait_time AG-477
$logevent MoM parameter AG-548
$restrict_user AG-391
$restrict_user_exceptions AG-391
$restrict_user_maxsysid AG-391
.rhosts AG-377

A

accelerator AG-474
accelerator_memory AG-474
accelerator_model AG-474
access
  by group AG-362
  by user AG-362
    effect of flatuid AG-376
  control lists AG-362
  from host AG-362
  to a queue AG-362
  to a reservation AG-362
  to server AG-362
accounting
  account AG-632, AG-637
  alt_id AG-632, AG-637
  authorized_hosts AG-630
  authorized_users AG-630
  ctime AG-630, AG-631, AG-632, AG-634, AG-637, AG-638, AG-639, AG-640
  duration AG-630
  end AG-630, AG-632, AG-638
  etime AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-640
  Exit_status AG-633, AG-638
  group AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
  jobname AG-631, AG-633, AG-637, AG-638, AG-639, AG-641
  name AG-630
  owner AG-630
  qtime AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
  queue AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
  Resource_List AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
  session AG-631, AG-634, AG-635, AG-638, AG-640, AG-641
  start AG-630, AG-631, AG-634, AG-635, AG-638, AG-640, AG-641
  user AG-631, AG-634, AG-635, AG-637, AG-638, AG-640, AG-641
accounting_id AG-632, AG-637
acl_group_enable
  queue attribute AG-370
acl_groups
  queue attribute AG-370
acl_host_enable
  queue attribute AG-370
  server attribute AG-370
acl_hosts
  queue attribute AG-370
  server attribute AG-370
acl_roots AG-394
acl_user_enable
  queue attribute AG-370
  server attribute AG-370
acl_users
  queue attribute AG-370
  server attribute AG-370
ACLs AG-362
  default behavior AG-363
  format AG-363
  group AG-364
  host AG-364
  matching entry AG-365
  modifying behavior AG-363
  overrides AG-376
  removing entity AG-367
  rules for creating AG-367
  user AG-364
  who can create AG-368
activate a power profile AG-319
advance reservation AG-198, AG-630
aggressive_provision AG-329
AOE AG-327
aoe resource
  defining AG-336
application license
  floating AG-276
    definition AG-231
  floating externally-managed AG-276

application licenses AG-274
floating license PBS-managed AG-277
license units and features AG-275
overview AG-257
per-host node-locked example AG-279
types AG-274
ASAP reservation AG-198
authentication AG-609
authorization AG-609
Authorized_Groups reservation attribute AG-371
Authorized_Hosts reservation attribute AG-371
Authorized_Users reservation attribute AG-371
average CPU usage enforcement AG-307
average_cpufactor AG-307
average_percent_over AG-307
average_trialperiod AG-307
avoid_provision AG-329

B

backfill_prime AG-195
basic fairshare AG-139
batch requests AG-548
Boolean
format AG-237
borrowing vnode AG-230, AG-271
built-in resource AG-231

C

checkpoint AG-630, AG-665
preemption via AG-188
chunk AG-231
clienthost AG-391
configuration
file staging AG-613
rsync AG-613
server AG-19
consumable resource AG-231
CPU AG-231
cpuaverage AG-307
cput AG-141
cray_compute AG-483
creating queues AG-22
creation of provisioning hooks AG-338
csaswitch AG-492, AG-654
current_aoe AG-336
current_eoe AG-322
custom resource AG-231
custom resources
application licenses AG-274
floating managed by PBS AG-277
overview AG-257
per-host node-locked AG-279
types AG-274
how to use AG-255
scratch space
overview AG-257
static host-level AG-270
static server-level AG-268
cycle harvesting
ideal_load AG-123
max_load AG-123

D

deactivate a power profile AG-319
debuginfo AG-663
decay AG-142
dedicated time AG-125
defining aoe resource AG-336
defining provisioning policy AG-339
defining resources
multi-vnode machines AG-272
degraded reservation AG-198
department AG-139
DIS AG-540
DNS AG-672
do_not_span_psets AG-475
dynamic fit AG-169
dynamic resource scripts/programs AG-265

E

egroup AG-139
euser AG-139
eligible wait time AG-126
eligible_time AG-127, AG-129, AG-632, AG-638
energy AG-322
enforcement AG-610
eoe AG-315, AG-322
euser AG-139
exec_host AG-630
exiting AG-127
express_queue AG-184

F

fair_share
scheduler parameter AG-138
fairshare AG-138, AG-184
fairshare entities AG-139
fairshare ID AG-140
fairshare_perc AG-151

file staging
configuration AG-613
files
pbs.conf AG-672
policy AG-610
location AG-610
finished jobs AG-531
flatuid server attribute AG-376
float
format AG-237
floating license
definition AG-231
example AG-276
example of externally-managed AG-276
format
Boolean AG-237
float AG-237
size AG-238
string resource value AG-238, AG-243
string_array AG-238, AG-244

G

gethostname AG-391
global resource AG-231
Globus AG-19
group
access AG-362
ACLs AG-364
limit AG-231, AG-289
generic AG-289
individual AG-289

H

hbm_cache_pct AG-484
hbmem AG-483
HCA AG-501
help, getting AG-675
high-bandwidth memory AG-484
history jobs AG-531
hooks
creation of provisioning AG-338
provisioning AG-327
host
access AG-362
ACLs AG-364
host channel adapter AG-501
hosts.equiv AG-377
HUP MoMs AG-483
hyperthreading AG-490

I

ideal_load
cycle harvesting AG-123
indirect resource AG-231, AG-271
ineligible_time AG-127
InfiniBand AG-466
initial_time AG-127
instance AG-198
instance of a standing reservation AG-198
instantiation AG-610

J

job history AG-530
changing settings AG-532
configuring AG-531
enabling AG-532
setting duration AG-532
job that can never run AG-666
job-specific ASAP reservation AG-198
job-specific now reservation AG-198
job-specific reservation AG-198
job-specific start reservation AG-198

L

last_state_change_time
vnode attribute AG-323
last_used_time
vnode attribute AG-323
license
application
floating AG-276
floating
definition AG-231
limit AG-232, AG-288
attributes AG-294
cput AG-305
file size AG-305
generic group limit AG-231, AG-289
generic project limit AG-289
generic user limit AG-231, AG-289
group limit AG-231, AG-289
individual group limit AG-232, AG-289
individual project limit AG-289
individual user limit AG-232, AG-289
overall AG-232, AG-289
pcput AG-305
pmem AG-305
project limit AG-289
pvmem AG-305
user limit AG-232, AG-289
walltime AG-305

limits
generic and individual AG-292
group AG-287
overall limits AG-292
project AG-287
resource usage AG-287
scope AG-290
setting limits AG-296
user AG-287
load_balancing AG-121
local resource AG-232
location
policy files AG-610
log events
MoM AG-548
scheduler AG-548
server AG-548
log levels AG-547
log_events
server attribute AG-548
logs
permissions AG-612

M

maintenance reservation AG-198
Manager
privilege AG-361
managers server attribute AG-361
managing vnode AG-232, AG-271
master provisioning script AG-327, AG-337
master script AG-327, AG-337
matching ACL entry AG-365
max_concurrent_provision AG-339
max_group_res AG-303
max_group_run AG-303
max_group_run_soft AG-303
max_load
cycle harvesting AG-123
max_queuable AG-303
max_queued AG-295
max_queued_res AG-295
max_run AG-294
max_run_res AG-295
max_run_res_soft AG-295
max_run_soft AG-294
max_running AG-303
max_user_res AG-303
max_user_res_soft AG-303
max_user_run AG-303
max_user_run_soft AG-303
max_walltime AG-216
memory
high-bandwidth AG-484
memory-only vnode AG-232
min_walltime AG-216
minimizing power cycles AG-320
MoM
log events AG-548
MPI_USE_IB AG-466
mpiexec AG-465
multihost placement sets AG-170
MUNGE AG-379

N

naccelerators AG-474
natural vnode AG-38
NEC SX-Aurora process swapping AG-504
NEC SX-Aurora TSUBASA AG-501
nhcas AG-502
node_idle_limit
server attribute AG-323
nodes
NUMA AG-484
Xeon Phi AG-483
non-consumable resource AG-232
nonprimetime_prefix AG-195
normal_jobs AG-184
NUMA nodes AG-484
numa_cfg AG-484
nves AG-502

O

Operator
privilege AG-360
operators server attribute AG-361
opt_backfill_fuzzy AG-109
output plugin AG-318
overall limit AG-232, AG-289

P

partial process swapping AG-504
password
invalid AG-665
pbs.conf AG-319, AG-613, AG-672
pbs_accounting_workload_mgmt AG-491, AG-653
PBS_AUTH_METHOD AG-540
PBS_BATCH_SERVICE_PORT AG-540
PBS_BATCH_SERVICE_PORT_DIS AG-540
PBS_COMM_LOG_EVENTS AG-540
PBS_COMM_ROUTERS AG-540
PBS_COMM_THREADS AG-540
PBS_CONF_SYSLOG AG-544, AG-553
PBS_CONF_SYSLOGSEVR AG-544, AG-553
PBS_CORE_LIMIT AG-540
PBS_DATA_SERVICE_PORT AG-540
PBS_ENCRYPT_METHOD AG-541

PBS_ENVIRONMENT AG-541
PBS_EXEC AG-410, AG-541
PBS_EXEC/share AG-610
PBS_HOME AG-410, AG-541
pbs_iff AG-672
PBS_LEAF_NAME AG-541
PBS_LEAF_ROUTERS AG-541
PBS_LOCALLOG AG-541, AG-553
PBS_LOG_HIGHRES_TIMESTAMP AG-541
PBS_LR_SAVE_PATH AG-541
PBS_MAIL_HOST_NAME AG-21, AG-542
PBS_MANAGER_SERVICE_PORT AG-542
pbs_mkdirs AG-665
PBS_MOM_HOME AG-410, AG-542
PBS_MOM_NODE_NAME AG-477, AG-542
PBS_MOM_SERVICE_PORT AG-542
PBS_MPI_DEBUG AG-466
PBS_OUTPUT_HOST_NAME AG-542
PBS_PRIMARY AG-410, AG-542
pbs_probe AG-664
PBS_RCP AG-542, AG-613
PBS_REMOTE_VIEWER AG-540
pbs_rsub AG-371
PBS_SCHED_THREADS AG-543
PBS_SCHEDULER_SERVICE_PORT AG-542
PBS_SCP AG-543, AG-613
PBS_SECONDARY AG-410, AG-543
PBS_SERVER AG-410, AG-543
PBS_SERVER_HOST_NAME AG-543
PBS_START_COMM AG-543
PBS_START_MOM AG-410, AG-543
PBS_START_SCHED AG-410, AG-543
PBS_START_SERVER AG-410, AG-543
PBS_SUPPORTED_AUTH_METHODS AG-543
PBS_TMPDIR AG-544
pbs-alps-inventory-check hook AG-483
PBScrayhost AG-240, AG-474
PBScraylabel AG-474
PBScraylabel_ AG-240
PBScraynid AG-240, AG-474
PBScrayorder AG-240, AG-474
pbsfs AG-142
pcap_accelerator AG-324, AG-633, AG-637, AG-639
pcap_node AG-324, AG-633, AG-637, AG-639
PCIe AG-501
permissions
    logs AG-612
pgov AG-324, AG-633, AG-637, AG-639
p-governor AG-317, AG-324
placement
    task AG-169
placement pool AG-169
placement set AG-169
placement sets
    multihost AG-170
policy
    defining provisioning AG-339
    files AG-610
        location AG-610
power cycles
    minimizing AG-320
power profile
    activate AG-319
    deactivate AG-319
power profiles AG-315
power_off_iteration
    server attribute AG-324
power_provisioning AG-322
    server attribute AG-322, AG-324
    vnode attribute AG-324
poweroff_eligible AG-324
PPS AG-504
preempt_order AG-180
preempt_prio AG-181
preempt_queue_prio AG-181
preempt_sort AG-181
preemption AG-180
preemption via checkpoint AG-188
preemptive scheduling AG-180
preemptive_sched AG-180
primary server AG-542
prime_spill AG-196
primetime_prefix AG-195
privilege
    Manager AG-361
    Operator AG-360
    user AG-360
project AG-289, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
project limit AG-289
    generic AG-289
    individual AG-289
project limits AG-287
prologue AG-319
provision_policy AG-329
provisioning
    creation of hooks AG-338
    defining policy AG-339
    hooks AG-327
    master script AG-337
        writing AG-337
    overview AG-328
    policy AG-329
    rebooting AG-328
    reservations AG-331
    vnode selection AG-329
    vnode states AG-332

pstate AG-323

Q

qdel AG-665
qmgr AG-19, AG-672
qrerun AG-665
qstat AG-672
qsub AG-665
query_other_jobs AG-612
queue AG-30
    access to a AG-362
    ACL AG-363
    attribute
        acl_group_enable AG-370
        acl_groups AG-370
        acl_host_enable AG-370
        acl_hosts AG-370
        acl_user_enable AG-370
        acl_users AG-370
queue_softlimits AG-184
queued jobs AG-289
queued_jobs_threshold AG-295
queued_jobs_threshold_res AG-295
queues
    creating AG-22

R

rcp AG-542
rebooting
    provisioning AG-328
reservation AG-630
    access to a AG-362
    ACL AG-363
    advance AG-198
    ASAP AG-198
    attribute
        Authorized_Groups AG-371
        Authorized_Hosts AG-371
        Authorized_Users AG-371
    control of creation AG-363
    degraded AG-198
    instance AG-198
    job-specific AG-198
        ASAP AG-198
        now AG-198
        start AG-198
    maintenance AG-198
    now AG-198
    reservation ID AG-199
    soonest occurrence AG-198
    standing AG-198
        instance AG-198
        soonest occurrence AG-198
reservations AG-197
    provisioning AG-331
resource AG-232
    built-in AG-231
    consumable AG-231
    custom AG-231
    indirect AG-231, AG-271
    non-consumable AG-232
    shared AG-232, AG-271
resource limits AG-287
resource usage limits AG-287
Resource_List AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
Resource_List.eoe AG-322
resources
    unset AG-161
resources_assigned AG-639
resources_available.eoe AG-322
resources_used.energy AG-319, AG-322
restrict_user AG-391
restrict_user_exceptions AG-391
restrict_user_maxsysid AG-392
resv_enable AG-371
resv_enable server attribute AG-363
roles AG-359
RPM
    debuginfo AG-663
rsync
    configuration AG-613
run_count AG-631, AG-634, AG-641
run_time AG-127
RUR AG-318

S

sched_preempt_enforce_resumption AG-182
scheduler
    log events AG-548
scp AG-543
scratch space AG-257
    dynamic
        host-level AG-274
        server-level AG-273
    static
        host-level AG-274
        server-level AG-274
script
    master provisioning AG-337
    writing provisioning AG-337
secondary server AG-543
security_context job attribute AG-610

server
access to AG-362
ACL AG-363
attribute
acl_host_enable AG-370
acl_hosts AG-370
acl_user_enable AG-370
acl_users AG-370
flatuid AG-376
log_events AG-548
managers AG-361
operators AG-361
resv_enable AG-363
log events AG-548
parameters AG-18
primary AG-542
recording configuration AG-19
secondary AG-543
server attributes
node_idle_limit AG-323
power_off_iteration AG-324
power_provisioning AG-322
server_softlimits AG-184
set_power_cap AG-324
setting limits AG-296
shared resource AG-232, AG-271
shares AG-139
size
format AG-238
sleep
vnode state AG-324
soonest occurrence AG-198
sort key AG-145
sshd AG-605
standing reservation AG-198
start reservation AG-198
starving_jobs AG-184
states
vnodes and provisioning AG-332
static fit AG-169
strict_ordering and backfilling AG-225
string AG-243
string resource value
format AG-238, AG-243
string_array AG-244
format AG-238, AG-244
support team AG-675
SX-Aurora AG-501
syslog AG-552

T

task placement AG-169
TSUBASA AG-501
type codes AG-548

U

unknown node AG-139
unknown_shares AG-138
unset resources AG-161
usage limits AG-287
user
access AG-362
ACLs AG-364
privilege AG-360
roles AG-359
user limit AG-232, AG-289
generic AG-289
individual AG-289
user limits AG-287

V

VE AG-501
VE offloading AG-501
ve_cput AG-502, AG-504
ve_mem AG-502, AG-504
vector engine AG-501
vector host AG-501
version information AG-664
VH AG-501
virtual nodes AG-37
vnode AG-37
borrowing AG-230, AG-271
managing AG-232, AG-271
memory-only AG-232
natural AG-38
selection for provisioning AG-329
states and provisioning AG-332
vnode attributes
last_state_change_time AG-323
last_used_time AG-323
vnode_pool AG-481
vnodedef_additive AG-475

W

Windows
password AG-665
writing provisioning script AG-337

X

X forwarding AG-545
xauth AG-545
Xeon Phi nodes AG-483



Altair PBS Professional 2021.1

Plugins (Hooks) Guide

You are reading the Altair PBS Professional 2021.1

Hooks Guide (HG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation vii

1 New Hook Features 1

1.1 New Features in PBS 2021.1 . . . 1
1.2 Changes in Previous Releases . . . 1
1.3 Deprecations and Removals . . . 3

2 Introduction to Hooks 5

2.1 Introduction to Hooks . . . 5
2.2 Glossary . . . 5
2.3 Prerequisites and Requirements for Hooks . . . 7
2.4 Uses for Hooks . . . 7

3 Quick Start with Hooks 11

3.1 Simple How-to for Writing Hooks . . . 11
3.2 Writing Hooks: Basic Hook Structure . . . 11
3.3 Example of Simple Hook . . . 12
3.4 Importing Hook Configuration File . . . 13
3.5 Creating and Importing Your Hook . . . 13
3.6 Setting Attributes for Your Hook . . . 13

4 Hook Basics 15

4.1 Hook Basics . . . 15
4.2 Viewing Hook Information . . . 22
4.3 Restarting the Python Interpreter . . . 23
4.4 Attributes and Parameters Affecting Hooks . . . 24
4.5 Python Modules and PBS . . . 24
4.6 See Also . . . 26

5 Creating and Configuring Hooks 29

5.1 Creating and Configuring Hooks . . . 29
5.2 Writing Hook Scripts to Operate on PBS Elements . . . 40
5.3 Advice and Caveats for Writing Hooks . . . 68

6 Hook Objects and Methods 75

6.1 The pbs Module . . . 76
6.2 PBS Interface Objects . . . 76
6.3 Event Objects . . . 87
6.4 Server Objects . . . 118
6.5 Queue Objects . . . 121
6.6 Job Objects . . . 122
6.7 The exec_vnode Object . . . 129
6.8 Chunk Objects . . . 131
6.9 Reservation Objects . . . 131
6.10 Vnode Objects . . . 133
6.11 Configuration File Objects . . . 135
6.12 Constant Objects . . . 140
6.13 Object Members and Methods . . . 141

7 Built-in Hooks 155

7.1 Managing Built-in Hooks . . . 155
7.2 Prerequisites . . . 155
7.3 Allowed Operations . . . 155
7.4 Viewing Built-in Hooks . . . 155
7.5 Setting Attributes of Built-in Hooks . . . 156
7.6 Editing and Importing Configuration Files for Built-in Hooks . . . 156
7.7 Restrictions . . . 156
7.8 Replacing a Built-in Hook with Your Own Hook . . . 156
7.9 Errors and Logging when Operating on Built-in Hooks . . . 157

8 Debugging Hooks 159

8.1 The `pbs_python` Hook Debugging Tool . . . 159
8.2 Files for Debugging . . . 159
8.3 Steps to Debug a Hook Using pbs_python . . . 165
8.4 Caveats and Restrictions for pbs_python . . . 165
8.5 Examples of Using `pbs_python` to Debug Hooks . . . 166
8.6 Using Log Messages to Debug Hook Scripts . . . 174
8.7 Checking Hook Syntax using Python . . . 174
8.8 Examples of Debugging Files . . . 174
8.9 Interactive Debugging using pbs_python . . . 221
8.10 Error Reporting and Logging . . . 221

9 Hook Examples 231

Index 289

1 New Hook Features

This chapter briefly lists new features by release, with the most recent listed first. This chapter also lists deprecated elements, such as options, keywords, etc.

The *Release Notes* included with this release of PBS Professional list all new features in this version of PBS Professional, and any warnings or caveats. Be sure to review the Release Notes, as they may contain information that was not available when this book was written.

1.1 New Features in PBS 2021.1

Improved Cgroups Hook

The cgroups hook is improved for 2020.1. See "Configuring and Using PBS with Cgroups" on page 571 in the PBS Professional Administrator's Guide.

Faster Read of Custom Job Resources by Execution Hooks

You can specify which custom job resources are cached at MoMs so that execution hooks can read them faster. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 261 in the PBS Professional Administrator's Guide.

New Post-suspend and Pre-resume Hooks

PBS has two new hook events for just after suspending a job and just before resuming it. See section 6.3.1, "Event Types", on page 87.

Python Version Changed to 3.6

PBS 19.4.1 uses Python version 3.6.

1.2 Changes in Previous Releases

Hooks Support Reliable Job Startup and Run (19.2)

Hooks have been enhanced to allow you to provide jobs with extra vnodes in case of vnode failure. See "Vnode Fault Tolerance for Job Start and Run" on page 434 in the PBS Professional Administrator's Guide.

New Reservation End Hook (19.2)

You can create hooks for the end of a reservation. See "resv_end: Event when Reservation Ends" on page 90 in the PBS Professional Hooks Guide.

Python Version Changed to 2.7.1 (18.2.3)

PBS 18.2.3 uses Python 2.7.1. The use of Python 2.5.1 is deprecated.

Periodic Server Hook (18.2.3)

PBS has a periodic hook that runs at the server. See section 6.3.1.7, "periodic: Periodic Event at Server Host", on page 95.

Hook to Run Job Start Time Estimator (18.2.3)

PBS has a built-in hook named PBS_est that can run the job start time estimator. See "Estimating Job Start Time" on page 131 in the PBS Professional Administrator's Guide.

Configurable Python Interpreter Restarts (18.2.3)

You can configure how often you want the Python interpreter to restart. See section 4.3, "Restarting the Python Interpreter", on page 23.

PBS Can Report Custom Resources Set in Hooks (18.2.3)

MoM can accumulate and report custom resources that are set in a hook. See section 5.2.4.12, "Setting Job Resources in Hooks", on page 49.

The execjob_prologue Hook Runs on All Sister MoMs (18.2.3)

The execjob_prologue hook runs on all sister MoMs. See section 6.3.1.9, "execjob_prologue: Event Just Before Execution of Top-level Job Process", on page 97.

New Hook Events (13.0)

PBS provides three new hook events:

- An execjob_launch hook runs just before MoM runs the user's program
- An execjob_attach hook runs when pbs_attach is called
- An exechost_startup hook runs when MoM starts up or is HUPed

See section 4.1.2, "When Hooks Run", on page 15, section 6.3.1.10, "execjob_launch: Event when Execution Host Receives Job", on page 98, section 6.3.1.11, "execjob_attach: Event when pbs_attach() runs", on page 100, and section 6.3.1.17, "exechost_startup: Event When Execution Host Starts Up", on page 106.

Configuration Files for Hooks (13.0)

You can use configuration files with hooks. See section 5.1.6, "Using Hook Configuration Files", on page 32.

Configuring Vnodes in Hooks (13.0)

You can use hooks to configure vnode attributes and resources. See section 5.2.4.11, "Setting and Unsetting Vnode Resources and Attributes", on page 48.

Adding Custom Resources in Hooks (13.0)

You can use hooks to add custom non-consumable host-level resources. See section 5.2.7, "Adding Custom Non-consumable Host-level Resources", on page 64.

Node Health Hook Features (13.0)

PBS has node health checking features for hooks. You can offline and clear vnodes, and restart the scheduling cycle. See section 5.2.12.4, "Offlining and Clearing Vnodes Using the fail_action Hook Attribute", on page 66 and section 5.2.6, "Restarting Scheduler Cycle After Hook Failure", on page 63.

Hook Debugging Enhancements (13.0)

You can get hooks to produce debugging information, and then read that information in while debugging hooks. See Chapter 8, "Debugging Hooks", on page 159.

Managing Built-in Hooks (13.0)

You can enable and disable built-in hooks. See Chapter 7, "Built-in Hooks", on page 155.

Scheduler Does not Trigger modifyjob Hooks (13.0)

The scheduler does not trigger modifyjob hooks. See Chapter 5, "Creating and Configuring Hooks", on page 29.

runjob Hook can Modify Job Attributes (12.2)

The runjob hook can modify a job's attributes and resources. See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

Execution Event and Periodic Hooks (12.0)

You can write hooks that run at the execution host when the job reaches the execution host, when the job starts, ends, is killed, and is cleaned up. You can also write hooks that run periodically on all execution hosts. See Chapter 5, "Creating and Configuring Hooks", on page 29.

Vnode Access for Hooks (11.0)

Hooks have access to vnode attributes and resources. See Chapter 5, "Creating and Configuring Hooks", on page 29.

Provisioning (10.2)

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application. See Chapter 7, "Provisioning", on page 327.

New Hook Type (10.2)

PBS has a new hook type which can be triggered when a job is to be run. See "Creating and Configuring Hooks" on page 29.

Hooks (10.0)

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code. See Chapter 5, "Creating and Configuring Hooks", on page 29.

1.3 Deprecations and Removals

The use of Python 2.x is **deprecated**. PBS now uses Python 3.6. (19.4.1)

2 Introduction to Hooks

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches, MoM startup checks, workarounds, etc., and extends the capabilities of PBS, without the need to modify source code.

This chapter describes how hooks can be used, how they work, the interface to hooks provided by the pbs module, how to create and deploy hooks, and how to get information about hooks.

Please read the entire chapter, and the "Special Notes (Hooks)" section of the release notes, before writing any hooks.

2.1 Introduction to Hooks

A hook is a block of Python code that PBS executes at certain events, for example, when a job is queued. As long as the Python code conforms to the rules we describe, you can have it do whatever you want. Each hook can *accept* (allow) or *reject* (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. The hook can use a configuration file that you provide. PBS provides an interface for use in hooks. This interface allows hooks to read and/or modify things such as job, server, vnode, and queue attributes, and the event that triggered the hook.

2.1.1 Built-in Hooks

Some functions of standard PBS are accomplished through built-in hooks. We use the keyword *pbshook* with these hooks. These hooks are not designed to be altered, so they have some restrictions placed on them. See Chapter 7, "Built-in Hooks", on page 155.

2.2 Glossary

Accept an action

The hook allows the action to take place.

Action

A PBS operation or state transition. Also called an *event*. For a list of events, see section 6.3.1, “Event Types”, on page 87.

Built-in hook

A hook that is supplied as part of PBS. These hooks cannot be created or deleted by administrators.

Creating a hook

When you "create a hook" using `qmgr`, you're telling PBS that you want it to make you an empty hook object that has no characteristics other than a name.

Event

A PBS operation or state transition. Also called *action*. For a list of events, see section 6.3.1, “Event Types”, on page 87.

Execution event hook, MoM hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks have names prefixed with "*execjob_*".

Failure action

The action taken when a hook fails to execute. Specified in the fail_action hook attribute. See section 5.1.9.2, "Using the fail_action Hook Attribute", on page 37.

Hook configuration file

Configuration file specific to a particular hook. See section 5.1.6, "Using Hook Configuration Files", on page 32.

Importing a hook

When you "import a hook" using `qmgr`, you're telling PBS which Python script to run when the hook is triggered.

Importing a hook configuration file

When you "import a hook configuration file" using `qmgr`, you're telling PBS which file should be stored as the configuration file for the specified hook.

MoM hook, execution event hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks have names prefixed with "*execjob_*"

Non-job event hook

A hook that is not directly related to a specific job. Non-job event hooks are periodic hooks, startup hooks, provisioning hooks, and reservation creation hooks.

pbshook

PBS keyword for a built-in hook.

pbs module

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, object members, and methods. You can operate on these objects using Python code.

Pre-execution event hook, server hook

A hook that runs at the PBS server. A server hook runs before the job is sent to MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

Reject an action

The hook prevents the action from taking place. For example, if a runjob hook rejects a job, the job is requeued.

Server hook, pre-execution event hook

A hook that runs at the PBS server. A server hook runs before the job is sent to MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

2.3 Prerequisites and Requirements for Hooks

- To create a hook under Linux, you must be logged into the primary or secondary server host as root. You must create any hooks at the primary or secondary server host.
- When creating hooks, make sure that each execution host where execution or periodic hooks should run has the $reject_root_scripts MoM parameter set to *False*. The default for this parameter is *False*.
- In order for execution event hooks to function, either the query_other_jobs server attribute must be set to *True*, or root at every execution host must be added to the managers list (root@hostname must be added to the managers server attribute). If you have any hooks running with user set to *pbsuser*, you will have to set query_other_jobs to *True* (you probably don't want to add *pbsuser* to managers).

A normal, non-privileged, user cannot circumvent, disable, add, delete, or modify hooks or the environment in which the hooks are run.

2.4 Uses for Hooks

2.4.1 Routing Jobs

- Route jobs into specific queues or between queues:
  - Automatically route interactive jobs into a particular execution queue
  - Move a job to another queue; for example, if project allocation is used up, move job to "background" queue
- Reject job submissions that do not specify a valid queue, printing an error message explaining the problem
- Enable project-based ACLs for queues to make sure the appropriate job runs in the correct queue

2.4.2 Managing Resource Requests and Usage

- Reject improperly specified jobs:
  - Reject jobs which do not specify walltime
  - Reject jobs that request a number of processors that is not a multiple of 8
  - Reject jobs requesting a specific queue, but not requesting memory
  - Reject jobs whose processors per node is not specified or is not numeric
- Modify job resource requests:
  - Apply default memory limit to jobs that request a specific queue
  - Check on requested CPU and memory and modify these or supply them if missing
  - Adjust for the fact that users ask for 2GB on a machine that has 2GB physical memory, but only 1.8 GB available memory, by changing the memory request to 1.8GB
- Reject parallel jobs for some queues.
- Set default properties, for example, if "myri" is not set, set it to "*False*" to ensure Myrinet is used only for Myrinet jobs.
- Convert from ALPS-specific resource request strings into PBS-specific job requirements.
- Automatically translate old syntax to new syntax.
- Compensate for dissimilar system capabilities; for example, allow users to use more CPUs only if they use old, slow machines.
- Limit reservations submitted by users to a maximum amount of resources and walltime, but do not limit reservations submitted by PBS administrators.
- Define resources and set values.

2.4.3 Ensuring that Jobs Run Properly

- Make sure that jobs, or all jobs in a queue, request exclusive access (`-l place=excl`).
- Reject multi-host jobs, restricting each job to a single machine.
- Put a hold on the job if there isn't enough scratch space when the job is submitted.
- Reject jobs that could cause problems, based on the user and type of job that have caused previous problems. For example, if Bill's Abaqus jobs crash the system, reject new Abaqus jobs from Bill.
- Validate an input deck before the job is submitted.
- Modify a job's dependency list when the job is rejected.
- Modify a job's list of environment variables before it gets to the execution host(s).

2.4.4 Managing Job Output

- Manage where output goes by modifying a job's output path with the job's ID.

2.4.5 Controlling Interactive Jobs

- Control interactive job submission; for example, enable or disable interactive jobs at the server or queue level

2.4.6 Helping Schedule Jobs

- Increase the priority of an array job once the first subjob runs, by modifying the value of a job resource used in the job sorting formula
- Change scheduling according to user and job:
  - Set initial user-dependent coefficients for the scheduling formula. For example, set values of custom resources based on job attributes and user
  - Set whether or not the job is rerunnable, based on user
  - Calculate CPH (CPH == total ncpus * walltime in hours) and set a custom CPH job resource to the value
- Set initial priorities for jobs
- Periodically run the job start time estimator named `pbs_est` at the server. See "Estimating Job Start Time" on page 131 of the PBS Professional Administrator's Guide.

2.4.7 Communicating Information to Users

- Report useful error messages back to the user, e.g., `"You do not have sufficient walltime left to run your job for 1:00:00. Your walltime balance is 00:30:00."`

2.4.8 Managing User Activity

- Reject jobs from blacklisted users
- Prevent users from using `qalter` to change their jobs in any way, allowing only administrators to `qalter` jobs
- Prevent users from bypassing controls: disallow a job being submitted to queueA in a held state and then being moved to queueB where the job would not have passed hook checks for queueB initially. For example, if a queuejob hook disallows interactive jobs for queueB, the administrator also needs to ensure that an interactive job is not initially submitted to queueA and later moved to queueB
- Prevent users from overriding node_group_key with `qsub -lplace = group = X`, or with `qalter`
- Restrict the ability to submit a reservation to PBS administrators only

2.4.9 Enabling Accounting and Validation

- Make sure correct project designation is used: if no project or account string is found, look up username in database to find appropriate project to use and add it as project or account string before submission
- Submit job to correct queue based on project: check for project number and submit job to queues based on project type, e.g. project number 1234 jobs get submitted into "challenge" queue; similarly for "standard" queue, etc
- Validate project before the job executes; if validation fails, do not start job, and print error message. Validation can be based on project name, or for example requested resources, such as CPU hours

2.4.10 Allocation Management

- You can use a job submission (queuejob) hook to check whether an entity has enough resources allocated to accept the job.
- You can use a hook that runs just before the job is sent to the execution host (runjob) to perform allocation management tasks such as deducting requested amounts of resources from an entity's allocation.
- You can use a hook that runs after a job finishes (execjob_epilogue) to perform final allocation management tasks such as allocation reconciliation.

2.4.11 Managing Job Execution

Hooks that run periodically at execution hosts can do the following:

- Modify job environment variables
- Check vnode health
- Report I/O wait time
- Report memory usage integral (MB*time used)
- Report energy usage to run a given job, if you have power sensors on vnodes
- Report actual usage of accelerator hardware (FPGAs, GPUs, etc)
- Interrogate HW performance counters so that you can flag codes that are not running efficiently (e.g. FLOPS < 5% of peak FLOPS)
- Record how much disk space a job has accumulated in PBS_JOBDIR
- Record power usage, energy usage, and disk space usage

Hooks that run just before the user's program executes can do the following:

- Change the job shell or executable
- Change the job shell or executable arguments
- Change the job's environment variables

2.4.12 Configuring Vnodes

Hooks that run when an execution host starts can do the following:

- Configure vnodes on the local host
- Create custom resources for vnodes
- Offline vnodes that are not ready for use
- Return vnodes to use that have been offlined

2.4.13 Provisioning Vnodes

- Provision a vnode with a new AOE. See Chapter 7, "Provisioning", on page 327.

2.4.14 Accepting or Rejecting Job Task Attachment

- Allow or disallow action when MoM is about to attach a process for a job

3
Quick Start with Hooks

3.1 Simple How-to for Writing Hooks

We will go into the details of what goes into a hook later in the chapter, but here we show the basics of how to create a hook. Steps for creating a hook:

1. Log into the server host as root
2. Write the hook script
3. Create an empty hook via `qmgr`
4. Set the attributes of the hook so that it triggers when you want, etc
5. If the hook will use a configuration file:
   a. Write the hook configuration file
   b. Import the hook configuration file
6. Import the hook script into the empty hook. You do not need to restart the MoM, unless it's an exechost_startup hook. Since exechost_startup hooks run only when MoM starts up or is HUPed, if you want the hook to run now, restart or `kill -HUP` the MoM.

3.2 Writing Hooks: Basic Hook Structure

- Import the pbs and sys modules:
  ```
  import pbs
  import sys
  ```
- Use the try... except construction, where you test for conditions in the try block, and accept or reject the event:
  ```
  try:
      …
  except:
  ```
  Consider either rerunning the job or deleting the job inside the `except:` block.
- Treat the SystemExit exception as a normal occurrence, and pass if it occurs:
  ```
  except SystemExit:
      pass
  ```
- Reject the event, or rerun or delete the job, if any other exception occurs:
  ```
  except:
      pbs.event().reject("%s hook failed with %s")
  ```
- If the requestor is the scheduler, and where appropriate, the server or MoM, allow the action to take place:
  ```
  if pbs.event().requestor in ["PBS_Server", "Scheduler", "pbs_mom"]:
      pbs.event().accept()
  ```

The following code fragment is a basic hook skeleton:

```
import pbs
import sys

e=pbs.event()
j=e.job
try:
    if e.requestor in ["Scheduler"]:
    e.accept()
    …

except SystemExit:
    pass

except:
    j.rerun()
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

3.3 Example of Simple Hook

Example 3-1: Set job priority

Set a job's priority

```
import pbs
import sys

    e = pbs.event()

try:
    # Get the hook event information and parameters
    # This will be for the 'modifyjob' event type.

    # Ignore requests from scheduler or server
    if e.requestor in ["PBS_Server", "Scheduler"]:
        e.accept()

    # Get the information for the job being queued
    j = e.job
    # Set the job's priority
    j.Priority = 7
    # accept the event
    e.accept()
except SystemExit:
    pass
except:
    e.reject("Failed to set job priority")
```

3.4 Importing Hook Configuration File

If you want your hook to use a configuration file, you can import the configuration file. A configuration file is not required.

Syntax for importing a configuration file:

```
Qmgr: import hook <hook_name> application/x-config <content-encoding>
   <input_config_file>
```

Here, <*content-encoding*> can be "*default*" (7-bit) or "*base64*".

See section 5.1.6, "Using Hook Configuration Files", on page 32.

3.5 Creating and Importing Your Hook

When you "create a hook" using `qmgr`, you're telling PBS that you want it to make you an empty hook object that has no characteristics other than a name. When you "import a hook" using `qmgr`, you're telling PBS which Python script to run when the hook is triggered.

Syntax for creating a hook:

```
Qmgr: create hook <hook name>
```

Simple syntax for importing a hook:

```
Qmgr: import hook <hook name> application/x-python <content-encoding> <input_file>
```

This uses the script named <*input_file*> as the contents of your hook.

- The <*input_file*> must be encoded with <*content-encoding*>.
- The allowed values for <*content-encoding*> are "*default*" (7 bit) and "*base64*".
- <*input_file*> must be locally accessible to both `qmgr` and the batch server.
- A relative path in <*input_file*> is relative to the directory where `qmgr` was executed.
- If your hook already has a content script, then that is overwritten by this import call.
- If the name of <*input_file*> contains spaces, <*input file*> must be quoted.

3.6 Setting Attributes for Your Hook

Hooks have attributes that control their behavior, such as which events trigger the hook, the time to allow the hook to execute, etc. The only attribute you must set for a simple hook is the event(s) that will trigger the hook. Choose your hook type according to the event you want, by looking in Table 5-1, "Hook Trigger Events," on page 31.

Syntax for setting the hook event(s):

```
Qmgr: set hook <hook name> event = <event name>
Qmgr: set hook <hook name> event = "<event name>, <event name>"
```

For more details on setting hook trigger events, see section 5.1.5, "Setting Hook Trigger Events", on page 31.

You can set the rest of the hook's attributes if you wish. To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

For a list of all the hook attributes, see section 5.1.9.3, "List of Hook Attributes", on page 37.

4
Hook Basics

4.1 Hook Basics

4.1.1 Accepting or Rejecting Actions

Hooks accept (allow) or reject (prevent) actions, modify input parameters, modify job attributes, environment variables, programs, program arguments, and change internal or external values.

Each action can have zero or more hooks. Each hook must either accept or reject its action. All of an action's hooks are run when that action is to be performed. For PBS to perform an action, all hooks enabled for that action must accept the action. If any hook rejects the action, the action is not performed by PBS. If a hook script doesn't call accept() or reject(), and it doesn't encounter an exception, PBS behaves as if the hook accepts the action. An action is always accepted, unless:

- pbs.event().reject() is called
- An unhandled exception is encountered
- The hook alarm has been triggered due to hook timeout being reached

When PBS executes the hooks for an action, it stops processing hooks at the first hook that rejects the action.

4.1.1.1 Examples of Accepting and Rejecting Actions

Example 4-1: Accepting an action: In this example, userA submits a job to queue Queue1, and the job submission action has two hooks: hook1 disallows jobs submitted by UserB, and hook2 disallows jobs being submitted directly to Queue2. Both hook1 and hook2 accept userA's job submission to Queue1, so the submission goes ahead.

Example 4-2: Rejecting an action: In this example, userA uses the `qmove` command to try to move jobA from Queue1 to Queue2. The job move action has two hooks: hook3 disallows jobs being moved into Queue2, and hook4 disallows userB moving jobs out of Queue1. In this example, hook3 rejects the action, so the move operation is disallowed, even though hook4 would have accepted the action.

4.1.2 When Hooks Run

Each type of event has a corresponding type of hook. The following are the events where you can run hooks, with the hook type:

4.1.2.1 Job-related Hooks that Run Before Execution

Hooks that run before a job is received by an execution host (pre-execution event hooks):

queuejob: Queueing a job

modifyjob: Modifying a job, except when scheduler makes the modification (can also run after job is received by execution host)

movejob: Moving a job

runjob: Just before a job is sent to an execution host

4.1.2.2 Job-related Hooks that Run at Execution Host

Hooks that run after a job is received by an execution host (execution event hooks):

execjob_begin: When a job is received by an execution host, after stagein

execjob_prologue: Just before starting a job's shell

execjob_launch: Just before starting the user's program

execjob_attach: When running `pbs_attach()`

execjob_preterm: Just before killing a job

execjob_epilogue: Just after executing or killing a job, but before job is cleaned up

execjob_end: Just after cleaning a job up

execjob_postsuspend: Just after suspending a job

execjob_preresume: Just before resuming a job

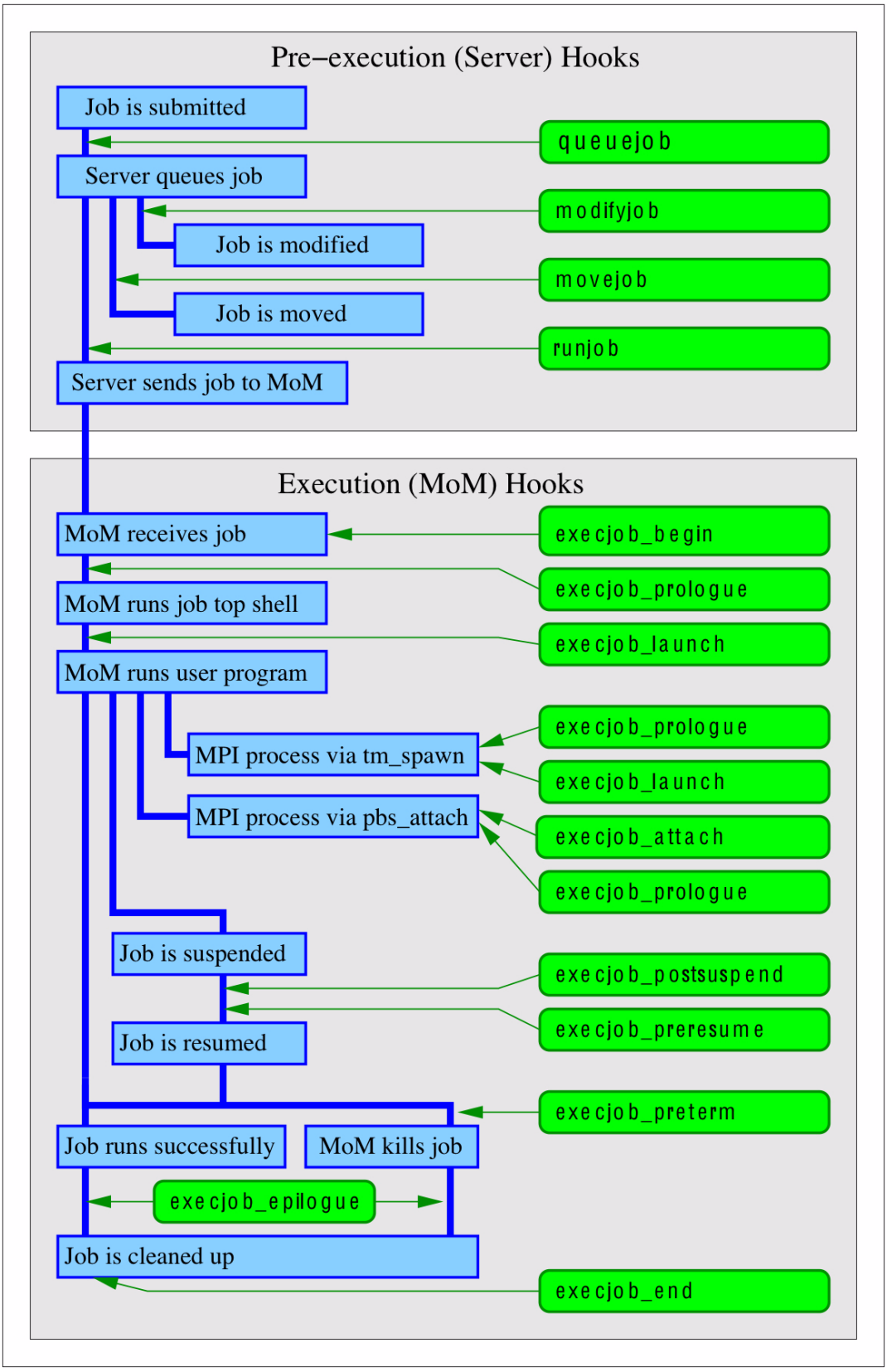


Figure 4-1:Simplified view of trigger timing for job-related hooks

4.1.2.3 Non-job-related Hooks

Hooks that are not directly related to a specific job (non-job event hooks):

resvsub: Submitting a PBS reservation

resv_end: When a PBS reservation ends

provision: Provisioning a vnode

exechost_periodic: Periodically on all execution hosts

exechost_startup: When an execution host is started or receives a HUP

periodic: periodically at the server

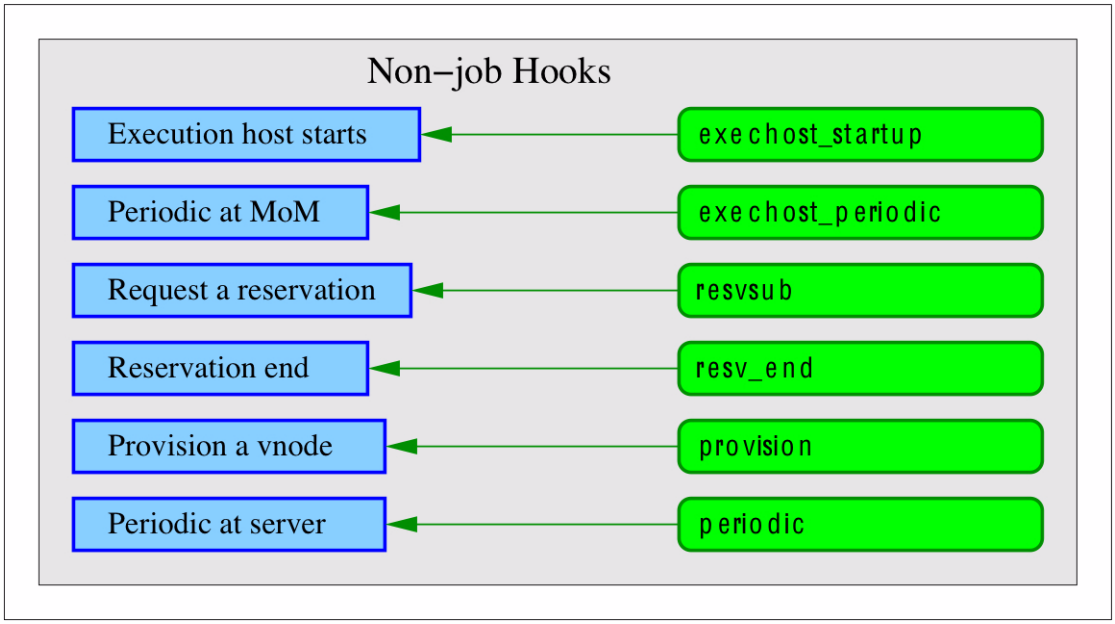


Figure 4-2:Simplified view of trigger timing for non-job-related hooks

4.1.2.4 Each Triggering Event Runs One Hook Instance

Each time an event triggers a hook, the hook runs for that instance of the event. If you have written a hook that runs at job submission, this hook will run for each job that is submitted to this server. Each MoM runs one copy of each of her execution hooks per job. Execution hooks run one per job at the MoM, not one per vnode. For a job that runs on four vnodes of a multi-vnoded machine where all the vnodes are managed by one MoM, where you have written one execution hook, only one instance of the hook runs for that job.

Each time a job goes through a triggering event, PBS runs any relevant hooks. This means that if you run a job, that triggers a runjob hook. If the job is killed and requeued and runs again, the runjob hook runs again.

If the scheduler modifies a job, any modifyjob hooks are not triggered.

When you are using peer scheduling, and a job is pulled from one complex to another, the pulling complex applies its hooks as if the job had been submitted locally, and the furnishing complex applies its movejob hooks. Figure 4-3 shows an example of the hooks that are triggered when a job is moved from a complex containing a movejob hook to a complex containing a queuejob hook.

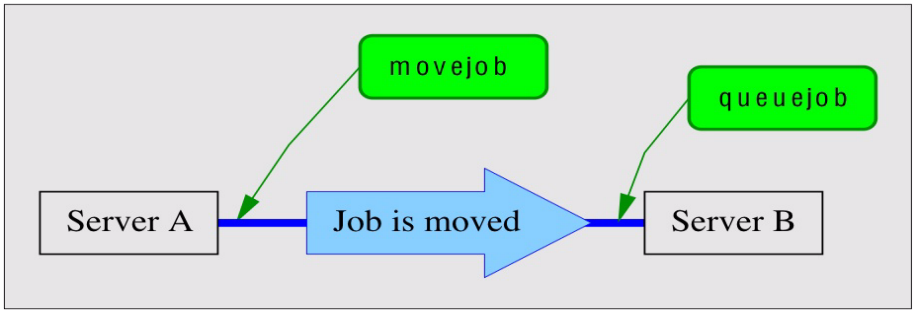


Figure 4-3:Hooks that run when job is moved

4.1.2.5 Execution Event Hook Triggers in Lifecycle of Job

The hooks triggered for an MPI job depend on whether MPI processes are spawned using the PBS TM interface via `tm_spawn()`, or are spawned using `pbs_attach()`. When a process is spawned using `tm_spawn()`, MoM starts the process. When a process uses `pbs_attach()`, `pbs_attach()` starts the process and informs MoM of the process ID.

The following shows where execution event hooks are triggered in the lifecycle of a normal, successful job. We show the timing for hooks on the primary execution host, on a sister vnode where a process is spawned using `tm_spawn()`, and on a sister vnode where a process is spawned using `pbs_attach()`.

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
Application licenses are checked out			
Any required job-specific staging and execution directories are created			
PBS_JOBDIR and job's jobdir attribute are set to pathname of staging and execution directory			
Files are staged in			
	execjob_begin	execjob_begin	execjob_begin
Job is sent to MoM			
	execjob_prologue If there is no execjob_prologue hook, the prologue script runs		
Server writes accounting log "S" record			
Primary execution host tells sister MoMs they will run job task(s)			

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
If necessary, MoM creates work directory			
MoM creates temporary directory for job			
MoM sets TMPDIR, JOBDIR, and other environment variables in job's environment			
MoM performs hardware-dependent setup: The job's cpusets are created, ALPS reservations are created			
	execjob_launch		
The job script starts			
Job starts an MPI process on sister vnode			
		execjob_prologue	execjob_prologue
		execjob_launch, for all tasks on this sister	
			execjob_attach, for all tasks on this sister
Job is suspended			
	execjob_postsuspend	execjob_postsuspend	execjob_postsuspend
	execjob_preresume		
		execjob_preresume (if successful on primary MoM)	execjob_preresume (if successful on primary MoM)
Job is resumed			
The job script finishes			
	execjob_epilogue If there is no execjob_epilogue hook, the epilogue script runs	execjob_epilogue	execjob_epilogue
The obit is sent to the server			
Server writes accounting log "E" record			
Any specified file staging out takes place, including `stdout` and `stderr`			
Files staged in or out are deleted			
Any job-specific staging and execution directories are removed			

Table 4-1: Execution Event Hook Timing

Job Lifecycle	Hooks Are Triggered		
	Primary Execution Host	Sister (`tm_spawn`)	Sister (`pbs_attach`)
The job's cpusets are destroyed			
Job files are deleted			
	execjob_end	execjob_end	execjob_end
Application licenses are returned to pool			

4.1.3 Account Under Which Hooks Run

A hook runs as the administrator or as the job owner, depending on the value of the hook's user attribute. If this is set to *pbsadmin*, the hook runs as the Administrator. If this is set to *pbsuser*, the hook runs as the job owner.

4.1.4 Where Hooks Run

Pre-execution event, periodic, provision, and reservation hooks run on the primary server host, or the secondary server host during failover. For most Linux systems, execution event, startup, and exechost_periodic hooks run on the execution host(s). However, on Cray, execution event, startup, and exechost_periodic hooks run on the host(s) where MoM runs.

4.1.5 Permissions and Location for Hook Creation and Modification

Hooks can be created or modified only by the administrator, and only at the hosts on which the primary and secondary servers run.

4.1.6 Failover

The secondary server uses the same filesystem as the primary server. Any hooks created are stored in the same place and are accessible by both servers, whether the primary or the secondary server is running.

When the secondary server takes over for the primary server after the primary's host has gone down or becomes inaccessible, any hooks created at the primary server continue to function under the secondary server.

If the you create a new hook while the secondary server has control, that hook will persist once the primary server takes over: if the primary server comes back up and takes over, hooks created while the secondary server had control continue to function.

4.1.7 What Hooks Cannot Access or Do

- Hooks cannot read or modify anything not presented in the PBS hook interface
- Hooks cannot modify the server or any queues
- Pre-execution event hooks cannot read or set vnode attributes or resources, except that the runjob hook can set the state attribute for any vnode to be used by the job
- Hooks do not have access to other servers besides the default server:
  - Hooks cannot change the destination server to a non-default server
  - Hooks can allow a job submission or a `qmove` to a non-default server, and can change the destination server from a remote server to the default server
- Hooks cannot directly print to `stdout` or `stderr` or read from `stdin`.
- movejob hooks do not run on `pbs_rsub -Wqmove=<job ID>`

4.1.8 What Hooks Should Not Do

- Hooks should not edit configuration files directly, meaning hooks should not edit the following:

  `PBS_HOME/sched_priv/sched_config`

  `PBS_HOME/sched_priv/fairshare`

  `PBS_HOME/sched_priv/dedicated`

  `PBS_HOME/sched_priv/holidays`

  `/etc/pbs.conf`

  `PBS_HOME/server_priv/resourcedef`

  `PBS_HOME/mom_priv/config`
- Hooks should not execute PBS commands

4.2 Viewing Hook Information

4.2.1 Listing Hooks

To list one hook and its attributes on the current server:

```
Qmgr: list hook <hook name>
```

To list all hooks and their attributes on the current server:

```
Qmgr: list hook
```

4.2.2 Viewing Hook Contents

To view the contents of a hook, export the hook's contents:

qmgr -c "export hook <hook_name> <content-type> <content-encoding>" > [<output_file>]

You cannot export the contents of a built-in hook.

4.2.3 Printing Hook Creation Commands

To view the commands to create one hook, including any configuration file:

```
Qmgr: print hook <hook name>
```

To view the commands to create all the hooks on the default server, including their configuration files:

```
Qmgr: print hook
```

or

```
qmgr -c "print hook"
```

For example, to see the commands used to create hook1 and hook2:

```
# qmgr -c "print hook"
create hook hook1
import hook hook1 application/x-python base64 - cHJpbnQgImhlbGxvLCB3b3JsZCIK

set hook hook1 event=movejob
set hook hook1 alarm=10
set hook hook1 order=5
create hook hook2
import hook hook2 application/x-python base64 - servaJLSDFSESF

set hook hook2 event=queuejob
set hook hook2 alarm=15
set hook hook2 order=60
…
```

4.2.4 Re-creating Hooks

To re-create a hook, including its configuration file, you feed `qmgr` hook descriptions back into `qmgr`. These hook descriptions are the same information that `qmgr` prints out. To print out the statements needed to recreate a hook, use the `print hook` or `print hook <hook name>` `qmgr` commands.

For example, to save information for hook1 and hook2:

```
# qmgr -c "print hook" > hookInfo
```

To re-create hook1 and hook2, with their configuration files:

```
# qmgr < hookInfo
```

4.3 Restarting the Python Interpreter

PBS keeps track of the number of hooks serviced, the number of objects created, and the time since the Python interpreter was last restarted. You can set a limit for the number of hooks created in the python_restart_max_hooks server attribute, a limit for the number of objects created in the python_restart_max_objects server attribute, and a limit for the minimum time interval at which to restart the Python interpreter in the python_restart_min_interval server attribute.

python_restart_max_hooks

The maximum number of hooks to be serviced before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.

Type: integer

Default: *100*

Python type: int

python_restart_max_objects

The maximum number of objects to be created before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.

Type: integer

Default: *1000*

Python type: int

python_restart_min_interval

The minimum time interval before the Python interpreter is restarted. If this interval has elapsed, and either the maximum number of hooks to be serviced (set in python_restart_max_hooks) has been exceeded or the maximum number of objects to be created (set in python_restart_max_objects) has been exceeded, the Python interpreter is restarted.

Type: integer seconds or [[HH:]MM:]SS

Default: *30*

Python type: pbs.duration

4.4 Attributes and Parameters Affecting Hooks

- Each hook's attributes affect the behavior of that hook. Hook attributes are listed in "Hook Attributes" on page 350 of the PBS Professional Reference Guide.
- The $reject_root_scripts MoM parameter controls whether MoM accepts new hook scripts.
- The server attributes that control when the Python interpreter is restarted are listed in Chapter 4, "Restarting the Python Interpreter", on page 23.

4.5 Python Modules and PBS

When you run a hook inside `pbs_python`, the hook has access to modules here:

- In `PBS_EXEC/python`
- In `PBS_EXEC/lib/python/altair`

Your hook can use other modules if you specify them in the hook.

The PBS_EXEC/python modules are in the following directories:

```
PBS_EXEC/python/lib/python36.zip
PBS_EXEC/python/lib/python3.6
PBS_EXEC/python/lib/python3.6/plat-linux2
PBS_EXEC/python/lib/python3.6/lib-tk
PBS_EXEC/python/lib/python3.6/lib-dynload
PBS_EXEC/python/lib/python3.6/site-packages
```

4.5.1 Python Module Caveats

In order to use PBS_EXEC/python/lib/python3.6/site-packages, you must first call the following:

```
import site
```

4.5.2 Modifying Python Modules

If you need to use other modules, we recommend that you put the modules in a different directory from PBS_EXEC/lib/python.

To use other modules besides the ones in PBS_EXEC/lib/python, specify the path in the hook.

If you are adding modules that are not in PBS_EXEC/lib/python, you can do this:

```
import sys
if '/usr/lib64/python3.6' not in sys.path:
    sys.path.append('/usr/lib64/python3.6')
import pbs
```

If you need to include user-defined paths ahead of the default modules, you can do the following. For example, if you put a module in /usr/lib64/python3.6, in /usr/local/lib64/python3.6, and in /usr/local/lib64/custom/python and you want to load them before the PBS-provided modules, add them to your hook this way:

```
import sys
my_paths = ['/usr/lib64/python3.6',
'/usr/local/lib64/python3.6',
'/usr/local/lib64/custom/python']
for my_path in my_paths:
    if my_path not in sys.path:
        sys.path.insert(0, my_path)
import pbs
```

4.5.2.1 Caveats for Modifying Python Modules

If you change a Python module in a pre-execution event hook (queuejob, movejob, modifyjob, runjob), you must restart the server in order to use the new module, because "import" is cached.

4.5.3 List of Modules in pbs_python

The following are the modules that are available via PBS_EXEC/lib/python:

Table 4-2: Modules in pbs_python

Module Name			
cElementTree	handler	pulldom	stringprep
collections	handlers	pydoc	symbol
config	itertools	re	time
copy	linecache	repr	token
copy_reg	math	saxutils	tokenize
cStringIO	md5	select	types
datetime	minicompat	sre	UserDict
dis	minidom	sre_compile	UserList
domreg	NodeFilter	sre_constants	UserString
ElementInclude	opcode	sre_parse	uuid
ElementPath	operator	stat	xmlbuilder
ElementTree	os	statvfs	xmlreader
expat	parser	string	zipfile
expatbuilder	pkgutil	StringIO	_exceptions
expatreader	posixpath	stringold	_socket

4.6 See Also

For a description of the PBS hook APIs, see the PBS Professional Programmer's Guide. Each PBS object's attribute's Python type is listed in its description in "Attributes" on page 277 of the PBS Professional Reference Guide. For example, "Server Attributes" on page 281 of the PBS Professional Reference Guide lists the Python type for the job_sort_formula server attribute.

The following man pages and equivalent sections contain useful information:

Table 4-3: See Also

Man Page	Guide Section
pbs_module(7B)	section 27.3.1, "The pbs Module", on page 110 of the PBS Professional Programmers Guide
pbs_stathook(3B)	section 27.4.2, "The pbs_stathook() API", on page 116 of the PBS Professional Programmers Guide
pbs_hook_attributes(7B)	"Hook Attributes" on page 350 of the PBS Professional Reference Guide
pbs_job_attributes(7B)	"Job Attributes" on page 328 of the PBS Professional Reference Guide

Table 4-3: See Also

Man Page	Guide Section
pbs_server_attributes(7B)	"Server Attributes" on page 281 of the PBS Professional Reference Guide
pbs_queue_attributes(7B)	"Queue Attributes" on page 311 of the PBS Professional Reference Guide
pbs_node_attributes(7B)	"Vnode Attributes" on page 320 of the PBS Professional Reference Guide
qmgr(1B)	"qmgr" on page 149 of the PBS Professional Reference Guide
qsub(1B)	"qsub" on page 213 of the PBS Professional Reference Guide
qmove(1B)	"qmove" on page 172 of the PBS Professional Reference Guide
qalter(1B)	"qalter" on page 127 of the PBS Professional Reference Guide
pbs_rsub(1B)	"pbs_rsub" on page 96 of the PBS Professional Reference Guide
pbs_manager(3B)	"pbs_manager" on page 41 of the PBS Professional Programmers Guide

5 Creating and Configuring Hooks

5.1 Creating and Configuring Hooks

In this chapter we describe how to create and configure site-defined hooks. For information about operating on built-in hooks, see Chapter 7, "Built-in Hooks", on page 155.

5.1.1 Introduction to Creating and Configuring Hooks

Hooks can only be created, run, or modified by the Administrator, and only on the host(s) on which the primary or secondary server runs.

You create hooks using the `qmgr` command to create, delete, import, or export the hook. The `qmgr` command operates on the *hook* object.

Syntax for operating on hooks:

qmgr -c "<command> hook <hook name> [<arguments to command>]"

where

command is *create, delete, set, unset, list, print, import, export*

5.1.1.1 Hook Name Restrictions

- Each hook must have a unique name.
- The name must be alphanumeric, and start with an alphabetic character.
- The name must not begin with "*PBS*".
- The name of a hook can be a legal PBS object name, such as the name of a queue.
- Hook names are case-sensitive.

5.1.2 Overview of Creating and Configuring a Hook

The following is an overview of the steps to create a hook. Each step is described in the following sections. You must be logged into the primary or secondary server host as root.

1. Use the `create hook` `qmgr` command to create an empty hook with the name you specify
2. Import the contents of a hook script into the hook
3. Set the hook's trigger event
4. If the hook will use a configuration file, write and import the configuration file
5. Set the hook's order of execution, if there is another hook for the same event
6. Optionally, set the hook's timeout
7. Make sure that the $reject_root_scripts MoM configuration parameter is set to *False* on all execution hosts where you want hooks to run. The default for this parameter is *False*.

You do not need to restart the MoM.

5.1.2.1 Example of Creating and Configuring a Hook

Create the hook:

```
Qmgr: create hook hook1
```

Import the hook script named `hook1_script.py` into the hook:

```
Qmgr: import hook hook1 application/x-python default /hooks/hook1_script.py
```

Make hook1 a queuejob hook:

```
Qmgr: set hook hook1 event = queuejob
```

Make this the second queuejob hook:

```
Qmgr: set hook hook1 order = 2
```

Set the hook to time out after 60 seconds:

```
Qmgr: set hook hook1 alarm = 60
```

Look at the $reject_root_scripts MoM configuration parameter where you want the hook to run, and make sure it is set to *False*.

5.1.3 Creating Empty Hooks

To create a hook, use the `create hook` command in `qmgr` to create an empty hook with the name you specify:

The create hook `qmgr` command creates an empty hook.

Syntax for creating a hook:

Qmgr: create hook <hook name>

5.1.3.1 Example of Creating an Empty Hook

To create the hook named "hook1", specify a filename, for example `"/hooks/hook1.py"`, that is locally accessible to `qmgr` and the PBS server:

```
Qmgr: create hook hook1
```

5.1.4 Deleting Hooks

To delete a hook, you use the `delete hook` command in `qmgr`.

Syntax for deleting a hook:

```
Qmgr: delete hook <hook name>
```

5.1.4.1 Example of Deleting a Hook

To delete hook hook1:

```
Qmgr: delete hook hook1
```

5.1.5 Setting Hook Trigger Events

To set the events that will cause a hook to be triggered, use the `set hook <hook name> event` command in `qmgr`. You can add triggering events to a hook.

To set one event:

```
Qmgr: set hook <hook name> event = <event name>
```

Designate triggers for a hook by setting <*event name*> to one of the following events:

Table 5-1: Hook Trigger Events

Action (Event)	Event Name
Accepting job into queue	*queuejob*
Modifying job, except when scheduler makes modification	*modifyjob*
Moving job	*movejob*
Before a job is sent to an execution host	*runjob*
Periodically on server host	*periodic*
When a job is received by an execution host, after stagein	*execjob_begin*
When `pbs_attach()` is called	*execjob_attach*
Just before executing a job's top shell	*execjob_prologue*
Just before executing the user's program	*execjob_launch*
Just after suspending a job	*execjob_postsuspend*
Just before resuming a job	*execjob_preresume*
Just after executing or killing a job, but before job is cleaned up	*execjob_epilogue*
Just before killing a job	*execjob_preterm*
Just after cleaning up a job that has finished or been killed	*execjob_end*
When an execution host starts up or receives a HUP	*exechost_startup*
Periodically on all execution hosts	*exechost_periodic*

Table 5-1: Hook Trigger Events

Action (Event)	Event Name
Provisioning a vnode	*provision*
Submitting reservation	*resvsub*
When reservation ends	*resv_end*

To add an event:

```
Qmgr: set hook <hook name> event += <event name>
```

For a detailed description of each event, see section 6.3.1, “Event Types”, on page 87.

5.1.5.1 Example of Setting Hook Trigger Events

To set an event that will cause hook "UserFilter" to be triggered:

```
Qmgr: set hook UserFilter event = queuejob
```

Add another event:

```
Qmgr: set hook UserFilter event += modifyjob
```

Set two events at once:

```
Qmgr: set hook UserFilter event = "queuejob, modifyjob"
```

You must enclose the value in double quotes if it contains a comma.

5.1.6 Using Hook Configuration Files

You can customize the behavior of a hook by providing a configuration file for the hook. You write the hook so that it reads and acts on its configuration file. Hooks are not required to use configuration files. A configuration file can contain whatever information is useful to the hook. A configuration file is just a file of whatever information you want; the way the hook reads and uses the contents of a configuration file is up to you. The hook itself processes the configuration file.

5.1.6.1 Format of Configuration File

PBS supports several file formats for configuration files. The format of the file is specified in its suffix. Formats can be specified in any of the following ways:

- `.ini`
- `.json`
- `.txt` (generic, no special format)
- `.xml`
- No suffix: treat the input file as if it is a `.txt` file
- The dash (-) symbol: configuration file content will be taken from `STDIN`. The content is treated as if it is a `.txt` file.

For example, to import a configuration file in `.json` format:

```
# qmgr -c "import hook <hook_name> application/x-config default input_file.json"
```

5.1.6.2 Importing Configuration File

To provide a configuration file for a hook, you import the configuration file into the hook. The import command is the same as for a hook, except that you set <*content-type*> to "*application/x-config*". Syntax for importing a configuration file:

```
Qmgr: import hook <hook_name> application/x-config <content-encoding>
   <input_config_file>
```

or

 # qmgr -c "import hook <hook_name> application/x-config <content-encoding> <input_config_file>"

where <*content-encoding*> is "*default*" (7-bit) or "*base64*".

This uses the contents of <*input_config_file*> or `stdin` (-) as the contents of configuration file for hook <*hook_name*>.

- The <*input_config_file*> or `stdin` (-) data must have a format <*content-type*> and must be encoded with <*content-encoding*>.
- The allowed values for <*content-encoding*> are "*default*" (7bit) and "*base64*".
- If the source of input is `stdin` (-) and <*content-encoding*> is "*default*", then `qmgr` expects the input data to be terminated by EOF.
- If the source of input is `stdin` (-) and <*content-encoding*> is "base64", then `qmgr` expects input data to be terminated by a blank line.
- <*input_config_file*> must be locally accessible to both `qmgr` and the requested batch server.
- A relative path <*input_config_file*> is relative to the directory where `qmgr` was executed.
- If a hook already has a configuration file, then that is overwritten by this import call.
- If <*input_config_file*> name contains spaces, <*input_config_file*> must be quoted.
- There is no restriction on the size of the hook configuration file.

5.1.6.2.i Examples of Importing Configuration Files

Importing a Python configuration file:

```
# qmgr -c 'import hook hook1 application/x-config default hello.py'
```

Importing a JSON configuration file:

```
# qmgr -c 'import hook hook1 application/x-config default hello.json'
```

5.1.6.3 Exporting Configuration Files

To edit or display the content of a hook configuration file associated with the hook named <hook_name>, export the configuration file. Use the export command:

qmgr -c "export hook <hook_name> application/x-config default" > <output file>

5.1.6.4 How Hooks Find Configuration Files

There are two ways to retrieve a configuration file in a hook.

- PBS puts the configuration file in a location that can be read by the hook, and sets the PBS_HOOK_CONFIG_FILE environment variable to that path. Your hook script can use this path:

```
import os
import ConfigParser

if "PBS_HOOK_CONFIG_FILE" in os.environ:
    config_file = os.environ["PBS_HOOK_CONFIG_FILE"]
    config = ConfigParser.RawConfigParser()
    config.read(os.environ["PBS_HOOK_CONFIG_FILE")
```

- Your hook can use the pbs.hook_config_filename variable, which contains the path to the configuration file. See "pbs.hook_config_filename" on page 135.

  If there is no configuration file, this variable returns *None*.

5.1.6.5 Changing a Hook Configuration File

To replace the content of a hook configuration file, export the file, edit it, and issue another "`import`" hook command with updated <*input_config_file*> content.

5.1.6.6 Validation and Errors

- PBS pre-validates <*input_config_file*> according to its file format, and returns an error in `qmgr`'s `STDERR` if validation fails. For example:

```
# qmgr -c "import hook submit application/x-config default file.json"
"Failed to validate config file, hook 'submit' config file not overwritten"
```

- If the input configuration file given is of unrecognized suffix, the following message is returned in `qmgr`'s `STDERR`.

```
"<input-file> contains an invalid suffix, should be one of: .json .py .txt .xml .ini"
```

- If you import a configuration file and PBS cannot open the file because it is non-existent, has permission problems, or has another system-related error, the following error message is printed in `STDERR`:

```
"qmgr: hook error: failed to open <filename> - <error message>"
```

- If you attempt to export a hook configuration file, but the file is unwriteable due to ownership or permission problems, the following error message is printed to `STDERR`:

```
"qmgr: hook error: <output_file> permission denied"
```

5.1.7 Importing Hooks

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to `qmgr` and the PBS server.

Syntax for importing a hook:

`Qmgr: import hook <hook_name> <content-type> <content-encoding> {<input_file>|-}`

This uses the contents of <*input_file*> or stdin (-) as the contents of hook <*hook_name*>.

- The <*input_file*> or stdin (-) data must have a format <*content-type*> and must be encoded with <*content-encoding*>.
- For script files, the only <*content-type*> currently supported is "*application/x-python*".
- The allowed values for <*content-encoding*> are "*default*" (7 bit) and "*base64*".
- If the source of input is stdin (-) and <*content-encoding*> is "*default*", then qmgr expects the input data to be terminated by EOF.
- If the source of input is stdin (-) and <*content-encoding*> is "*base64*", then qmgr expects input data to be terminated by a blank line.
- <*input_file*> must be locally accessible to both qmgr and the requested batch server.
- A relative path in <*input_file*> is relative to the directory where qmgr was executed.
- If a hook already has a content script, then that is overwritten by this import call.
- If the name of <*input_file*> contains spaces, <*input file*> must be quoted.
- There is no restriction on the size of the hook script.

5.1.7.1 Examples of Importing Hooks

Example 5-1: Given a Python script in ASCII text file "hello.py", this makes its contents into the script contents of hook1:

```
#cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 5-2: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this script the contents of hook1:

```
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

Example 5-3: Read stdin for text containing data until EOF, and make this into the script contents of hook1:

```
# qmgr -c 'import hook hook1 application/x-python default -'
import pbs
pbs.event().job.comment="Hello from stdin"
Ctrl-D
```

Example 5-4: Read stdin for a base64-encoded string of data terminated by a blank line. PBS unencodes the data and makes this script the contents of hook1.

```
# qmgr -c 'import hook hook1 application/x-python base64 -'
cHJpbnQgImhlbGxvLCB3b3JsZCIK
Ctrl-D
```

5.1.8 Exporting Hooks

Syntax for exporting a hook:

qmgr -c "export hook <hook_name> <content-type> <content-encoding>" > <output_file>

This dumps the script contents of hook <*hook_name*> into <*output_file*>, or stdout if <*output_file*> is not specified.

- The resulting <*output_file*> or stdout data is of <*content-type*> and <*content-encoding*>.
- The only <*content-type*> currently supported for scripts is "*application/x-python*".
- The allowed values for <*content-encoding*> are "*default*" (7bit) and "*base64*".
- <*output_file*> must be a path that can be created by qmgr.
- Any relative path in <*output_file*> is relative to the directory where qmgr was executed.
- If <*output_file*> already exists it is overwritten. If PBS is unable to overwrite the file due to ownership or permission problems, then an error message is displayed in stderr.
- If the <*output_file*> name contains spaces, <*output file*> must be enclosed in quotes.

5.1.8.1 Examples of Exporting Hooks

Example 5-5: Dumps hook1's script contents directly into the file "hello.py.out":

```
# qmgr -c "export hook hook1 application/x-python default" > hello.py
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 5-6: To dump the script contents of a hook 'hook1' into a file in "\My Hooks\hook1.py":

```
qmgr -c "export hook hook1 application/x-python default" > "\My Hooks\hook1.py"
```

Example 5-7: Dump hook1's script contents base64-encoded into a file called "hello.py.b64":

```
# qmgr -c "export hook hook1 application/x-python base64" > hello.py.b64
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
```

Example 5-8: Dump hook1's script contents directly to stdout:

```
# qmgr -c "export hook hook1 application/x-python default"
import pbs
pbs.event().job.comment="Hello, world"
```

Example 5-9: Dump hook1's script contents base64-encoded into stdout:

```
# qmgr -c "export hook hook1 application/x-python base64"
 cHJpbnQgImhlbGxvLCB3b3JsZCIK
```

5.1.9 Setting and Unsetting Hook Attributes

You configure a hook using the qmgr command to set or unset its attributes. An unset hook attribute takes the default value for that attribute.

Hook attributes can be viewed via qmgr:

```
Qmgr: list hook <hook name>
```

To set a hook attribute:

```
Qmgr: set hook <hook name> <attribute> = <value>
```

To unset a hook attribute:

```
Qmgr: unset hook <hook name> <attribute>
```

For example, to unset hook1's alarm attribute, causing its value to revert to its default value:

```
Qmgr: unset hook hook1 alarm
```

This causes hook1's alarm to revert to the default of *30 seconds*.

5.1.9.1 Caveats for Setting Hook Attributes

You cannot set the type attribute for a built-in hook.

5.1.9.2 Using the fail_action Hook Attribute

The fail_action hook attribute is a string_array and can take on multiple values:

None
: No action is taken.

offline_vnodes
: After unsuccessful hook execution, offlines the vnodes managed by the MoM executing the hook. Can be set for execjob_begin, execjob_prologue, and exechost_startup hooks only.

clear_vnodes_upon_recovery
: After successful hook execution, clears vnodes previously offlined via *"offline_vnodes"* fail action. Can be set for exechost_startup hooks only.

scheduler_restart_cycle
: After unsuccessful hook execution, restarts scheduling cycle. Can be set for execjob_begin and execjob_prologue hooks only.

Default value: "*None*"

If you specify offlining or clearing vnodes in addition to restarting the scheduler, the scheduler restart happens last. The order of the values is not important.

To set the attribute:

```
# qmgr -c "set hook <hook_name> fail_action = <fail_action value>"
# qmgr -c "set hook <hook_name> fail_action = '<fail_action value>,<fail_action value>'"
```

To add a value to the list of values:

```
# qmgr -c "set hook <hook_name> fail_action += <fail_action value>"
```

To remove a value from the list of values:

```
# qmgr -c "set hook <hook_name> fail_action -=  <fail_action value>"
```

To find out what the values are:

```
# qmgr -c "list hook <hook_name> fail_action"
<hook_name>
    fail_action = <fail_action value>
```

To unset the attribute:

```
# qmgr -c "unset hook <hook_name> fail_action"
```

See section 5.2.12.4, “Offlining and Clearing Vnodes Using the fail_action Hook Attribute”, on page 66 and section 5.2.6, “Restarting Scheduler Cycle After Hook Failure”, on page 63.

5.1.9.3 List of Hook Attributes

Hook attributes are listed in “Hook Attributes” on page 350 of the PBS Professional Reference Guide.

5.1.10 Enabling and Disabling Hooks

A hook is either *enabled*, and will run when its action happens, or is *disabled*, and will not run. Hooks are enabled by default.

Syntax to enable a hook:

```
Qmgr: set hook <hook name> enabled=True
```

Syntax to disable a hook:

```
Qmgr: set hook <hook name> enabled=False
```

5.1.10.1 Example of Enabling and Disabling Hooks

To enable hook1:

```
Qmgr: set hook hook1 enabled=True
```

To disable hook1:

```
Qmgr: set hook hook1 enabled=False
```

5.1.11 Setting the Relative Order of Hook Execution

When there are multiple hooks of the same type for one action, you may wish to specify the order in which these hooks are run. The order in which the hooks for an action are run is determined by each hook's order attribute. Hooks with a lower value for order will run before hooks with a higher value. To set the relative order in which the hooks for an action will be run, set each hook's order attribute.

Syntax:

```
Qmgr: set hook <hook name> order=<ordering>
```

<*ordering*> is an integer. Hooks with lower values for <*ordering*> run before those with higher values; a hook with *order=1* runs before a hook with *order=2*.

Valid values for order:

- Built-in hooks can be from *-1000* to *2000*
- Site hooks can be from *1* to *1000*

The order in which hooks of the same type for unrelated actions execute is undefined. For example, there are two queuejob hooks, Hook1 and Hook2, and userA submits jobA and userB submits jobB. While Hook1 always runs before Hook2 for the same job, the order of execution is undefined for different jobs. So the order could be:

Hook1 (jobB)

Hook1 (jobA)

Hook2 (jobA)

Hook2 (jobB)

5.1.11.1 Example of Setting Relative Order of Hook Execution

To set hookA to run first and hookB to run second:

```
Qmgr: set hook hookA order=2
Qmgr: set hook hookB order=5
```

5.1.11.2 Caveats for Setting Relative Order of Hooks

The order attribute is ignored for exechost_periodic and periodic hooks.

5.1.12 Setting Hook Timeout

You may wish to specify how long PBS should wait for a hook to run. Execution for each hook times out after the number of seconds specified in the hook's alarm attribute. If the hook does not run in the specified time, PBS aborts the hook and rejects the hook's action.

Syntax:

```
Qmgr: set hook <hook name> alarm=<timeout>
```

<*timeout*> is the number of seconds PBS will allow the hook to run.

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show:

```
06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in
    Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>
```

5.1.12.1 Example of Setting Hook Timeout

To set the number of seconds that PBS will wait for hook hook1 to execute before aborting the hook and reject the action:

```
Qmgr: set hook hook1 alarm=20
```

5.1.13 Setting Hook Interval (Frequency)

You can specify the interval at which a periodic hook runs. You can do this only for hooks whose event type is exechost_periodic or periodic.

Syntax:

```
Qmgr: set hook <hook name> freq=<interval>
```

<*interval*> is the number of seconds elapsed between calls to this hook.

5.1.13.1 Example of Setting Hook Interval (Frequency)

To set the number of seconds between calls to an exechost_periodic or periodic hook:

```
Qmgr: set hook hook1 freq=200
```

5.1.14 Setting Hook User Account

You can specify the account under which a hook runs.

Syntax:

```
Qmgr: set hook <hook name> user=<pbsadmin | pbsuser>
```

pbsadmin specifies that the hook runs as root or as administrator.

pbsuser specifies that the hook runs as the job owner.

You can specify that a hook runs as the job owner only for execjob_prologue, execjob_epilogue, and execjob_preterm hooks.

If you do not set the account, it defaults to *pbsadmin*.

5.1.14.1 Example of Setting Hook User Account

To set the account under which a hook runs:

```
Qmgr: set hook hook1 user=pbsuser
```

5.2 Writing Hook Scripts to Operate on PBS Elements

5.2.1 How We Define and Refer to Objects and Methods

5.2.1.1 Scope of Object or Method

When we define an object or method, we show the scope of the object or method. For example, the scope of a job is the pbs module, so we call it a *pbs.job*, and a server has the same scope, so it is a *pbs.server*. Similarly, the logjobmsg() method has module-wide scope, and is defined as *pbs.logjobmsg()*.

However, the scope of a job ID object is the job, not the module, so it is defined as a *pbs.job.id*, and the scope of the job's is_checkpointed() method is the job, so it is defined as *pbs.job.is_checkpointed()*.

5.2.1.2 Referring to Objects

In a hook, you refer to the triggering event using pbs.event(). In a hook that is triggered by a job-related event, such as a movejob or execjob_begin hook, the event has an associated pbs.job object representing the job that triggered the event, and you refer to it using *pbs.event().job*. You can refer to members of that job object using *pbs.event().job.<member>*. For example, to refer to the ID of the job associated with the event, you use *pbs.event().job.id*. To use the is_checkpointed() method on the job associated with the event, you use *pbs.event().job.is_checkpointed()*. You can use shortcuts:

```
e = pbs.event()
j = e.job
c = j.is_checkpointed()
```

5.2.1.3 How to Retrieve Objects: Event vs. Server

Each event has access to specific objects, listed in Table 6-3, “Using Event Object Members in Events,” on page 108. You can manipulate many of these objects through the event. To retrieve the job that triggered an event, you refer to it this way: *pbs.event().job*.

The server has read access to all objects in the pbs module. You refer to these objects through the server. For example, to retrieve a job whose ID is "*1234*" through the server, you use *pbs.server().job("1234")*. **You cannot manipulate an object that is retrieved through the server**.

5.2.1.3.i Retrieving Jobs

The way you retrieve a job determines how much access you have to that job. You can retrieve a job either through the event, via pbs.event().job, or through the server, via pbs.server().job().

If you retrieve a job through an event, the event gives you the job itself, represented as an object. You can see and alter some job attributes for an event-retrieved job object. To get the job object representing the job associated with the current event, on which you can operate, use pbs.event().job. We show which hooks can see and set each job attribute in Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55.

However, if you retrieve a job through the server, the server gives you an instantiated job object that contains a **copy** of the job. You cannot set any job attributes for a server-retrieved job object, and trying to operate on a server-retrieved copy of the job causes an exception. In order to get read-only information about a particular job with ID <*id*>, use *pbs.server().job('<job ID>')*. This returns a read-only copy of the job.

You can see all of the attributes for a server-retrieved job object, except in a queuejob hook. In a queuejob hook, the event gives you the job as it exists before the server sees it, but the server cannot retrieve it, because the job has not yet made it to the server.

5.2.1.3.ii Retrieving Vnodes

Vnode objects behave like job objects. If you retrieve a vnode object through an event, via pbs.event().vnode_list[], you can see some of the vnode's attributes, and set vnode attributes. We show which hooks can see and set each vnode attribute in Table 5-7, "Vnode Attributes Readable & Settable via Events," on page 57.

If you retrieve a vnode object through the server, via pbs.server().vnode(), you have a **copy** of the vnode, and you can see all of the vnode's attributes, but you cannot set any of them.

5.2.1.3.iii Retrieving Queues

You can retrieve queues through the server only, using *pbs.server().queue("<queue name>")*, or using *pbs.server().queues()*. You cannot make any changes to queue objects in hooks. These are read-only.

You can change a job's destination queue, but only to a queue at the local server. Hooks have access only to the local server. Hooks can allow a job submission to a remote server, but they **cannot** specify a remote server. See section 5.3.9.1, "Local Server Only", on page 72. Hooks can specify the destination queue at a local server for a queuejob or movejob event, whether the original destination queue was at the local server or a remote server.

To specify a destination queue at the local server:

```
pbs.event().job.queue = pbs.server().queue("<local_queue>")
```

Do not specify a queue at a remote server in a hook script.

5.2.1.3.iv Retrieving Reservations

In order to get information about a reservation being created in a resvsub event, use *pbs.event().resv*. pbs.server() cannot return information about the reservation, because the reservation has not yet been created.

5.2.2 Recommended Hook Script Structure

5.2.2.1 Catch Exceptions

Your hook script should catch all exceptions except for SystemExit. We recommend that you catch exceptions via try... except and accompany them with a call to pbs.event().reject().

It is helpful if it displays a useful error message in the `stderr` of the command triggering the hook. The error message should show the type of the error and should describe the error.

Here is the recommended script structure:

```
import pbs
import sys

try:
    …

except SystemExit:
    pass

except:
    pbs.event().job.rerun()
    pbs.event().reject("%s hook failed with %s. Please contact \
        Admin" % (pbs.event().hook_name, sys.exc_info()[:2]))
```

5.2.2.1.i Example of Catching Exceptions

This example shows how a coding error in the hook is caught with the except statement, and an appropriate error message is generated. In line 7, the statement k=5/0 generates a divide-by-zero error. The hook script is designed to reject interactive jobs that are submitted to queue "nointer".

```
import pbs
import sys

try:

    batchq = "nointer"
    e = pbs.event()
    j = e.job
    k = 5/0
    if j.queue and j.queue.name == batchq and j.interactive:
        e.reject("Can't submit an interactive job in '%s' queue" %
        (batchq))

except SystemExit:

    pass

except:
    e.reject("%s hook failed with %s. Please contact Admin" % (e.hook_name, sys.exc_info()[:2]))
```

The hook is triggered:

```
% qsub job.scr
qsub: c1 hhook failed with (<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero',)).
    Please contact Admin
```

5.2.2.1.ii Table of Exceptions

The following exceptions may be raised when using the pbs.* objects:

Table 5-2: Exceptions Raised When Using pbs.* Objects

Object	Exception
pbs.BadAttributeValueError	Raised when setting member value of a pbs.* object to an invalid value.
pbs.BadAttributeValueTypeError	Raised when setting member value of a pbs.* object to an invalid type.
pbs.BadResourceValueError	Raised when setting resource value of a pbs.* object to an invalid value.
pbs.BadResourceValueTypeError	Raised when setting resource value of a pbs.* object to an invalid type.
pbs.EventIncompatibleError	Raised when referencing a nonexistent member in pbs.event. Example: calling pbs.event().resv for pbs.event().type of pbs.QUEUEJOB
pbs.UnsetAttributeNameError	Raised when referencing a non-existent member name of a pbs.* object.
pbs.UnsetResourceNameError	Raised when referencing a non-existent resource name of a pbs.* object.
SystemExit	1. Raised when pbs.event().reject() terminates hook execution. 2. Raised when pbs.event().accept() terminates hook execution.

5.2.3 Hook Alarm Calls and Unhandled Exceptions

- An execjob_begin or exechost_startup hook can cause a failure action to take place when the hook script fails due to an alarm call or an unhandled exception. Otherwise, the following happens:

If a pre-execution event or execution event hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that initiates the action results in the following message in stderr:

  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Please inform Admin"

- The following message appears in the appropriate PBS daemon log, logged under PBSEVENT_DEBUG2 event class:

  "<request type> hook <hook_name> encountered an exception, request rejected"

- The job is left unmodified.

- If an exechost_startup hook script encounters an unexpected error causing an unhandled exception, vnode changes do not take effect, but MoM continues to run, and the following message appears at level PBSEVENT_DEBUG2 in mom_logs:

  "exechost_startup hook <hook_name> encountered an exception, request rejected"

  - The following statements will cause an unhandled exception if they appear in a hook script as is:
  - ZeroDivisionError exception raised:

    ```
    val = 5/0
    ```

  - BadAttributeValueError exception raised; pbs.hold_types and strings don't mix:

    ```
    pbs.event().job.Hold_Types = "z"
    ```

  - EventIncompatibleError exception raised for the following runjob event; runjob event has job attribute, not resv attribute:

    ```
    r = pbs.event().resv
    ```

- You can use execjob_begin and exechost_startup hooks to offline vnodes when those hooks encounter alarm calls or unhandled exceptions. See "Offlining and Clearing Vnodes Using the fail_action Hook Attribute" on page 66 of the PBS Professional Reference Guide. You can then clear the offline state from those vnodes later when an exechost_startup hook runs successfully.

- You can use an execjob_begin hook restart the scheduler cycle when the hook encounters an alarm call or unhandled exception. See "Restarting Scheduler Cycle After Hook Failure" on page 63 of the PBS Professional Reference Guide.

For a list of exceptions, see Table 5.2.2.1.ii, "Table of Exceptions," on page 43.

5.2.4 Using Attributes and Resources in Hooks

5.2.4.1 Using Built-in vs. Custom Resources in Hooks

Hooks have more access to built-in resources than they do to custom resources. All hooks can read built-in resources. All event hooks that run at the server can read all custom resources via pbs.event(), as well as via pbs.server(). However, hooks that run at the execution host can read custom resources only via pbs.server(). So for example if a job requests a custom resource, a runjob hook can read the resource, but an execjob_begin hook cannot.

5.2.4.2 Creating and Setting Custom Resources in Hooks

You can create a custom resource only in an exechost_startup hook. You can set a custom resource in a hook that runs at the server or using an exechost_startup hook. To create and set a custom resource in a vnode's resources_available attribute via an exechost_startup hook:

```
# qmgr -c "create hook start event=exechost_startup"
# qmgr -c "import hook start application/x-python default start.py"
# qmgr -c "export hook start application/x-python default"
```

Hook script:

```
import pbs
e=pbs.event()
localnode=pbs.get_local_nodename()

e.vnode_list[localnode].resources_available['foo_i'] = 7
e.vnode_list[localnode].resources_available['foo_f'] = 5.0
e.vnode_list[localnode].resources_available['foo_str'] = "seventyseven"
```

Note that while an exechost_startup hook cannot read an existing custom resource, it can create and set a new one.

When you create a custom job resource in an exechost_startup hook, the *m* flag is set by default. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 261 in the PBS Professional Administrator's Guide.

5.2.4.3 Determining Whether to Use Creation Method to Set Attribute or Resource

The way you set an attribute or resource depends on the type of the attribute or resource:

- If the attribute or resource is a string (str), an integer (int), a Boolean (bool), a long (long), or a floating point (float), you can set it directly:

  pbs.event().job.<attribute name> = <attribute value>

  pbs.event().job.Resource_List["<resource name>"]=<resource value>

  For example:

  ```
  jobA = pbs.event().job
  jobA.Account_Name = "AccountA"
  jobA.Priority = 100
  ```

- However, if the attribute or resource is any other type, you must use the corresponding creation method to instantiate an object of the correct type with the desired value as a formatted input string, then assign the object to the job. For example:

  ```
  pbs.event().job.Hold_Types = pbs.hold_types("uo")
  ```

  For creation methods, see section 6.13.3, "PBS Types and Their Methods", on page 143.

5.2.4.3.i Caveat for Objects Requiring Creation Method

You can operate on these objects only as if they are strings. Use `repr()` on the object to get its full string representation. You can then manipulate this representation using the built-in methods for Python 'str'.

5.2.4.3.ii Python Types not Requiring Creation Method

The following Python types do not require you to use an explicit creation method:

bool

float

int

str

5.2.4.4 How to Unset an Attribute or Resource

To unset an attribute or resource, set *<attribute value>* to *None*:

```
pbs.event().job.<attribute name> = None
```

When you unset an attribute or resource, it takes its default value.

5.2.4.4.i How to Unset an Attribute or Resource Requiring Creation Method

You can unset a job attribute or resource that has a creation method by setting it to *None*.

Example:

```
pbs.event().job.Hold_Types = None
```

5.2.4.5 Using Attributes in Hooks: Reading vs. Setting

All hooks can read, but not set, all job, vnode, server, queue, and reservation attributes via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc.

We list which job attributes can be read or set when the job is retrieved through an event in Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55.

We list which vnode attributes can be read or set when the vnode is retrieved through an event in Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57.

We list which reservation attributes can be read or set when the reservation is retrieved through an event in Table 5-8, “Reservation Attributes Readable & Settable in resvsub and resv_end Hooks,” on page 59.

No hooks can see or set any scheduler attributes.

The job, vnode, or reservation object's attributes appear to the hook as they would be after the event, not before it, for all hooks except runjob hooks.

5.2.4.6 Setting Time Attributes

For the job attributes Execution_Time, ctime, etime, mtime, qtime, and stime, the pbs.job object expects or shows the number of seconds since Epoch. The only one of these that can be set is Execution_Time.

For the reservation attributes reserve_start, reserve_end, and ctime, the pbs.resv object expects and shows the number of seconds since Epoch. The ctime attribute cannot be set.

If you wish to set the value for Execution_Time, reserve_start, or reserve_end using the *[[CCYY]MMDDhhmm[.ss]* format, or to see the value of any of the time attributes in the ASCII time format, load the Python `time` module and use the functions `time.mktime([CCYY, MM, DD, hh, mm, ss, -1, -1, -1])` and `time.ctime()`.

Example:

```
import time
job.Execution_Time = time.mktime([07, 11, 28, 14, 10, 15, -1, -1, -1])
time.ctime(job.Execution_Time)
'Wed Nov 28 14:10:15 2007'
```

If reserve_duration is unset or set to *None*, the reservation's duration is taken from the walltime resource attribute associated with the reservation request. If reserve_duration and walltime are both specified, meaning not set to *None*, reserve_duration will take precedence.

5.2.4.7 Special Characters in Variable_List Job Attribute

When special characters are used in Variable_List job attributes, they must be escaped. For this attribute, special characters are comma (,), single quote ('), double quote ("), and backslash (\). PBS requires each of these to be escaped with a backslash. However, Python requires that double quotes and backslashes also be escaped with a backslash. If the special character inside a string is a single quote, you must enclose the string in double quotes. If the special character inside the string is a double quote, you must enclose the string in single quotes. The following rules show how to use special characters in a Variable_List attribute when writing a Python script:

Table 5-3: How to Use Special Characters in Python Scripts

Character	Example Value	How Value is Represented in Python Script
, (comma)	*a,b*	"a\\,b" or 'a\\,b'
' (single quote)	*c'd*	"c\\'d"
" (double quote)	*f"g"h*	'f\\\"g\\\"h'
\ (backslash)	*\home\dir\files*	"\\home\\dir\\files" or '\\home\\dir\\files'

For example, if the path is:

```
"\Documents and Settings\pbstest\bin:\windows\system32"
```

This is how the path shows up in a script:

```
job.Variable_List["PATH"] = "\\Documents and Settings\\pbstest\\bin:\\windows\\system32"
```

5.2.4.8 Using string_array Attributes and Resources

5.2.4.8.i Handling Literal Values and Special Characters in string_array Format

In order to capture a literal value or special characters in a string_array attribute or resource, enclose the entire string array in single quotes.

For an attribute or resource whose type is string_array and whose value contains one or more commas (","), the whole string must be enclosed in single quotes, outside of its double quotes. For example:

If our string array has a single element consisting of "glad, elated":

```
job.Resource_List["test_string_array"] = '"glad, elated"'
```

If our string array has two elements, where one is "glad, elated" and the other is "happy":

```
job.Resource_List["test_string_array"] = '"glad, elated","happy"'
```

5.2.4.9 Using Resources in Hooks: Reading vs. Setting

All hooks can read, but not set, all job, vnode, server, queue, and reservation resources via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc.

The resources that can be read or set via pbs.event() vary by hook:

We list the job resources that can be read and set via an event in each kind of hook in Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

We list the vnode resources that can be read and set via an event in each kind of hook in Table 5-10, "Vnode Resources Readable & Settable by Hooks via Events," on page 61.

We give an overview of the resources that can be read and set by each hook in Table 5-4, “Overview of Resources Readable & Settable by Hooks via Pre-execution and Provision Events,” on page 52 and Table 5-5, “Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events,” on page 52. In these tables, if we say that a hook can read or set a group of resources, for example the server's resources_available attribute, that means that the hook can read or set all of the resources for that group.

5.2.4.10 Reading Resources in Hooks

PBS resources are represented as objects of type pbs.pbs_resource, where the resource names are the keys. This type is described in section 6.13.3.19, “Method to Create or Set Resource List”, on page 147. Built-in resources are listed in “List of Built-in Resources” on page 259 of the PBS Professional Reference Guide.

You can read a resource through objects such as the server, the event that triggered the hook, or the vnode to which a resource belongs. For example:

```
pbs.server().resources_available["<resource name>"]
pbs.event().job.Resource_List["<resource name>"]
pbs.event().vnode_list[<vnode name>].resources_available["ncpus"]
```

The resource name must be in quotes.

Example: Get the number of CPUs in a job's Resource_List attribute:

```
ncpus=pbs.event().job.Resource_List["ncpus"]
```

5.2.4.10.i Converting walltime to Seconds

If you want to see a job's walltime in seconds:

```
int(pbs.event().job.Resource_List["walltime"])
```

For example:

```
pbs.logmsg(pbs.LOG_DEBUG, "walltime=%d" % (int(pbs.event().job.Resource_List["walltime"])))
```

If walltime is "00:30:15", this results in the following:

```
walltime=1815
```

5.2.4.11 Setting and Unsetting Vnode Resources and Attributes

You can set and unset vnode resources and attributes using the vnode_list[] object in an exechost_startup or exechost_periodic hook. Any changes made this way are merged with those defined in a Version 2 vnode configuration file.

To set the attributes and resources for a particular vnode:

pbs.event().vnode_list[<vnode name>].<attribute> = <value>

pbs.event().vnode_list["<vnode name>"].resources_available["<resource name>"] = <resource value>

Resource names and string values must be quoted.

Some examples:

```
pbs.event().vnode_list["V2"].pcpus = 5
pbs.event().vnode_list["V2"].resources_available["ncpus"]  = 3
pbs.event().vnode_list["V2"].resources_available.["mem"] = pbs.size("100gb")
pbs.event().vnode_list["V2"].arch = "linux"
pbs.event().vnode_list["V2"].state = pbs.ND_OFFLINE
pbs.event().vnode_list["V2"].sharing = pbs.ND_FORCE_EXCL
```

To unset a resource value, specify "*None*" as its value:

pbs.event().vnode_list[<vnode_name>].resources_available[<res>] = None

pbs.event().vnode_list[<vnode_name>].<attribute> = None

5.2.4.12 Setting Job Resources in Hooks

You can set a job's Resource_List in pre-execution event hooks listed in Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

You can use an execution event hook (execjob_prologue, execjob_epilogue, and exechost_periodic) to set the value of a job's resources_used for host-level resources. The values of these resources are then reported in the job's resources_used attribute. For multi-vnode jobs, numeric values are summed and string resources are aggregated on a per-MoM basis.

5.2.4.12.i Steps for Setting Job Resources in Hooks

You can set values for a job's Resource_List or resources_used attributes as follows:

pbs.event().job.Resource_List["<resource name>"] = <resource value>

pbs.event().job.resources_used["<resource name>"] = <resource value>

For example:

```
pbs.event().job.Resource_List["mem"] = 8gb
```

5.2.4.12.ii String Resource Format for Python

Each string value returned by a MoM is a JSON object (a Python dictionary), which is an unordered set of key-value pairs, where each object begins with a left curly brace ("{"), and ends with a right curly brace ("}"). Each key is followed by a colon (":"), and the key-value pairs are separated using a comma (","). The key is enclosed in double quotes (allowing backslash escapes).

5.2.4.12.iii Setting String Job Resources in Hooks

When all values are in JSON format, the resulting string resource value is a union of all dictionary items, shown in `qstat -f` output and accounting logs as:

```
resources_used.<resource_name> = {<MoMA_JSON_item_value>, <MoMB_JSON_item_value>,
    <MoMC_JSON_item_value>, ..}
```

Example 5-10: If MoMA returns '{ "a":1, "b":2 }', MoMB returns '{ "c":1 }', and MoMC returns '{"d":4}' for resources_used.foo_str. We see the following:

```
resources_used.foo_str='{"a": 1, "b": 2, "c":1,"d": 4}'
```

If two or more values have the same value for the key, only one of them is retained, depending on Python's operation of merging dictionary items. We recommend that hook writers make the keys unique; you can do this by using the value returned by pbs.get_local_nodename() as part of the key.

When at least one of the values obtained from a sister MoM is not of JSON format, the string cannot be accumulated, and resources_used remains unset. PBS writes an error message in the MoM logs as follows:

```
"Job <jobid> resources_used.<string_resource> cannot be accumulated: value <input value> from MoM
    <hostname> not JSON-format: <exception_error_message>."
```

5.2.4.12.iv Example of Setting Resources in Hooks

Example 5-11: Using an epilogue hook that runs on all the MoMs, we set different resources_used values depending on whether the hook executes on the primary execution host or a sister MoM:

```
#: qmgr -c "list hook epi"
Hook epi
type = site
enabled = true
event = execjob_epilogue
user = pbsadmin
alarm = 30
order = 1
debug = false
fail_action = none
# qmgr -c "e h epi application/x-python default"
import pbs
e=pbs.event()
pbs.logmsg(pbs.LOG_DEBUG, "executed epilogue hook")
if e.job.in_ms_mom(): #set in MS mom
    e.job.resources_used["vmem"] = pbs.size("9gb")
    e.job.resources_used["foo_i"] = 9
    e.job.resources_used["foo_f"] = 0.09
    e.job.resources_used["foo_str"] = '{"nine":9}'
    e.job.resources_used["cput"] = 10
    e.job.resources_used["foo_assn2"] = '{"vn1":1,"vn2":2,"vn3":3}'
else: # set in sister mom
    e.job.resources_used["vmem"] = pbs.size("10gb")
    e.job.resources_used["foo_i"] = 10
    e.job.resources_used["foo_f"] = 0.10
    e.job.resources_used["foo_str"] = '{"ten":10}'
    e.job.resources_used["cput"] = 20
    e.job.resources_used["foo_assn2"] = '{"vn4":4,"vn5":5,"vn6":6}'
```

Using two execution hosts, submit the following job:

```
% cat job.scr2
PBS -l select=2:ncpus=1
pbsdsh -n 1 hostname
sleep 300


% qsub job.scr2
102.corretja
```

When the job completes, we can see values for resources_used. With server job_history_enabled=*True*, we can check the values in a finished job. Values in bold show resources accumulated from both MoMs:

```
% qstat -x -f 102
...
resources_used.cpupercent = 0
resources_used.cput = 00:00:30
resources_used.vmem = 19gb
resources_used.foo_f = 0.19
resources_used.foo_i = 19
resources_used.foo_str = '{"nine": 9, "ten": 10}'
resources_used.foo_assn2='{"vn1": 1, "vn2": 2 ,"vn3": 3 ,"vn4": 4, "vn5": 5, "vn6": 6}'
resources_used.mem = 0kb
resources_used.ncpus = 2
resources_used.walltime = 00:00:05
```

The accounting logs show the same values:

```
8/03/2016 18:28:13;E;102.corretja;user=alfie group=users project=_pbs_project_default
    jobname=job.scr2 queue=workq ctime=1470263288 qtime=1470263288 etime=1470263288
    start=1470263288 exec_host=corretja/0+nadal/0 exec_vnode=(corretja:ncpus=1)+(nadal:ncpus=1)
    Resource_List.ncpus=2 Resource_List.nodect=2 Resource_List.place=free
    Resource_List.select=2:ncpus=1 session=16986 end=1470263293 Exit_status=143
    resources_used.cpupercent=0 resources_used.cput=00:00:30 resources_used.vmem=19gb
    resources_used.foo_f=0.19 resources_used.foo_i=19 resources_used.foo_str='{"nine": 9, "ten":
    10}' resources_used.foo_assn2='{"vn1": 1, "vn2": 2 ,"vn3": 3 ,"vn4": 4, "vn5": 5, "vn6": 6}'
    resources_used.mem=0kb resources_used.ncpus=2 resources_used.walltime=00:00:05 run_count=1
```

5.2.4.12.v Setting Built-in Job Resource in Hook Prevents MoM from Updating Resource

If you use a hook to set the value of a built-in host-level resource for a specific job, MoM no longer updates the value of the resource for that job; she leaves that to you. You can get MoM to resume updating the resource for that job only by changing the hook so that it doesn't set the resource, and restarting the job.

Under Linux, job resources_used that MoM does not modify if they've been set in a hook are cput, walltime, mem, vmem, ncpus, and cpupercent.

Under Windows, job resources_used that MoM does not modify if they've been set in a hook are cput, walltime, mem, and ncpus.

5.2.4.13 Overview of Readable & Settable Resources

Here we list an overview of which resources can be read or set in hooks. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect. The following table shows which resource categories are readable or settable in pre-execution and provision hooks:

Table 5-4: Overview of Resources Readable & Settable by Hooks via Pre-execution and Provision Events

Resource Category	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	provision
Job Resource_List (Varies; see Table 5-9)	r, s	r, s	r	Table 5-9	---	---	---
Job resources_used	o	r	r	r	r	---	---
Vnode resources_available	---	---	---	---	---	---	---
Vnode resources_assigned	r	r	r	r	r	---	r
Server resources_available	r	r	r	r	r	r	r
Server resources_assigned	r	r	r	r	r	r	r
Server resources_default	r	r	r	r	r	r	r
Server resources_max	r	r	r	r	r	r	r
Queue resources_available	r	r	r	r	r	r	r
Queue resources_assigned	r	r	r	r	r	r	r
Queue resources_default	r	r	r	r	r	r	r
Queue resources_max	r	r	r	r	r	r	r
Queue resources_min	r	r	r	r	r	r	r
Reservation Resource_List	r	r	r	r	r, s	r	---

The following table lists an overview of which resources can be read or set in execjob and exechost hooks.

Table 5-5: Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events

Resource Category	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Job Resource_List (except for custom resources)	r	r	r	r	r	r	r	r	r	r	r
Job resources_used	r, s	r	r, s	r, s	r	r	r	r, s	r, s	r	r, s
Vnode resources_available	r, s	r	r, s	r	r	r	r	r, s	r, s	r, s	r, s
Vnode resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Server resources_available	r	r	r	r	r	r	r	r	r	r	r
Server resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Server resources_default	r	r	r	r	r	r	r	r	r	r	r

Table 5-5: Overview of Resources Readable & Settable by Hooks via execjob_ and exechost_ Events

Resource Category	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Server resources_max	r	r	r	r	r	r	r	r	r	r	r
Queue resources_available	r	r	r	r	r	r	r	r	r	r	r
Queue resources_assigned	r	r	r	r	r	r	r	r	r	r	r
Queue resources_default	r	r	r	r	r	r	r	r	r	r	r
Queue resources_max	r	r	r	r	r	r	r	r	r	r	r
Queue resources_min	r	r	r	r	r	r	r	r	r	r	r
Reservation Resource_List	r	r	r	r	r	r	r	r	r	r	r

5.2.4.14 Caveats for Setting and Unsetting Attributes and Resources

5.2.4.14.i When to Change Reservation Attributes

The only time that a reservation's attributes can be altered is during the creation of that reservation in a resvsub hook.

5.2.4.14.ii Caution About Unsetting Reservation walltime Resource

The walltime resource is used to determine the reservation's reserve_duration parameter when the reservation's reserve_duration attribute is not set or is set to *None*. If a resvsub hook attempts to unset the walltime parameter, for example:

```
pbs.event().resv.Resource_List["walltime"] = None
```

This will result in the following error:

```
% pbs_rsub -R 1800 -l ncpus=1
pbs_rsub: Bad time specification(s)
```

5.2.4.14.iii Changing Job Attributes for a Running Job

When a job is running, only the cput and walltime attributes can be modified. Attempting to change any other attributes for a running job will cause the corresponding `qalter` action to be rejected. For example, if the job is running, this line in a hook will cause `qalter` to be rejected:

```
pbs.event().job.Resource_List["mem"] = pbs.size("10mb")
```

To avoid having the `qalter` action rejected, check to see whether the job is running, and follow up accordingly. For example:

```
e = pbs.event()
if e.job.job_state in [ pbs.JOB_STATE_RUNNING, pbs.JOB_STATE_EXITING, pbs.JOB_STATE_TRANSIT ]:
    e.accept()
```

5.2.4.14.iv Do Not Unset Array Job Indices

Do not unset pbs.event().job.array_indices_submitted for an array job in a modifyjob hook. For example:

```
pbs.event().job.array_indices_submitted = None
```

If the hook script is executed for a job array, the qalter request will fail with the message:

```
Cannot modify attribute while job running  <job array ID>
```

5.2.4.14.v Do Not Create Job or Reservation Variable List

Hooks are not allowed to create job or reservation Variable_List attributes. Hooks can modify the existing Variable_List job attribute which is supplied by PBS, by modifying values in the list. The following are disallowed in a hook:

```
pbs.event().job.Variable_List = dict()
pbs.event().resv.Variable_List = dict()
```

These calls will cause the following exception:

```
04/07/2008 11:22:14;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Error
    evaluating Python script, attribute 'Variable_List' cannot be directly set.
```

To modify the Variable_List attribute:

```
pbs.event().job.Variable_List["SIMULATE"] = "HOOK1"
```

5.2.4.14.vi Changing Vnode state Attribute

A vnode's state can be set within a runjob hook only if the runjob hook execution concludes with a pbs.event().reject() call. This means that if a statement that sets a vnode's state appears in a runjob hook script, it takes effect only if the following is the last line to be executed:

```
pbs.event().reject()
```

To set a vnode's state, the syntax is one of the following:

pbs.vnode.state = <vnode state constant>

pbs.vnode.state += <vnode state constant>

pbs.vnode.state -= <vnode state constant>

where *<vnode state constant>* is one of the constant objects listed in section 6.10.4, “Vnode State Constant Objects”, on page 135.

Examples of changing a vnode's state attribute:

- To offline a vnode:
  ```
  pbs.vnode.state = pbs.ND_OFFLINE
  ```
- To add another value to the list of vnode states:
  ```
  pbs.vnode.state += pbs.ND_DOWN
  ```
- To remove a value from the list of vnode states:
  ```
  pbs.vnode.state -= pbs.ND_OFFLINE
  ```

When a vnode's state attribute has no states set, the vnode's state is equivalent to *free*. This means that you can remove all values, and the vnode will become *free*.

When a vnode's state is successfully set, the following message is displayed and logged at event class 0x0004:

```
Node;<vnode-name>;attributes set: state - <vnode state constant> by <hook_name>
```

You can set a vnode's state attribute in any execution hook and in a periodic hook, and changes to vnode attributes take effect whether the execution hook or periodic hook calls accept() or reject().

5.2.4.14.vii Attribute Change Failure is Silent

If you attempt to change the value for an attribute in an unsupported way, PBS does not warn you that your attempt failed.

5.2.4.14.viii Lengthened walltime Can Interfere with Reservations

If a hook lengthens the walltime of a running job, you run the risk that the new walltime will interfere with existing reservations etc.

5.2.4.14.ix Setting Vnode Resources in Hooks Overwrites Previous Value

When you set resources_available for a vnode, inside or outside of a hook, you are overwriting the previous value. There is no way in a hook to know whether a value was set inside or outside a hook (for example, using `qmgr` or a vnode definition file). There is no way to prevent a value set inside a hook from being modified outside of the hook.

5.2.4.14.x Changing Resources in Accounting Logs

If you use a non-execjob_end execution hook to set a value for resources_used, the new value for resources_used appears in the accounting logs.

5.2.4.14.xi When Setting Resources Has No Effect

- If you use an execjob_end execution hook to set a value for resources_used, it has no effect, because MoM has already sent the final values for resources_used to the server.
- You cannot use a hook to set a server-level resource. PBS ignores these actions in a hook.
- You cannot use the `qmgr` command to set resources_used for a job.

5.2.4.15 Table: Reading & Setting Job Attributes in Hooks

The following table lists the job attributes that can be read or set when the job is retrieved via an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this attribute can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
accounting_id	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Account_Name	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
accrue_type	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
alt_id	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
argument_list	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_id	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array_index	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
array_indices_remaining	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_indices_submitted	r, s	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
array_state_count	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
block	---	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Checkpoint	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
comment	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
create_resv_from_job	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s	r, s

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
ctime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
depend	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
egroup	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
eligible_time	---	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Error_Path	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
estimated	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
etime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
euser	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Executable	r, s	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Execution_Time	r, s	r, s	r	r, s	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
exec_host	---	---	r	---	---	---	---	---	r	r	r	r	r	r	r	r	r	---	r
exec_vnode	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Exit_status	---	r	r	r	r	---	---	---	---	r	---	---	r	r	r	r	---	---	---
group_list	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
hashname	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Hold_Types	r, s	r, s	r	r, s	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
interactive	r, s	r, o	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
jobdir	---	r	r	---	---	---	---	---	---	---	---	---	---	---	---	r	---	---	---
Job_Name	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Job_Owner	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
job_state	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Join_Path	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Keep_Files	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Mail_Points	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Mail_Users	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
max_run_subjobs	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mtime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
no_stdio_sockets	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Output_Path	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Priority	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
project	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
pset	---	---	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
qtime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
queue	r, s	r	r, s	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
queue_rank	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
queue_type	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
release_nodes_on_stageout	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Rerunable	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_released	r	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_released_list	r	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resources_used	---	r	r	r	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	r, s	r, s

Table 5-6: Job Attributes Readable & Settable via Events

Job Attribute	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic
Resource_List (with restrictions; see Table 5-9)	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r
run_count	r, s	r, s	r	r	r	---	---	---	r, s	r	r, s	r, s	r	r	r	r, s	r, s	---	r, s
run_version	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
sandbox	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
schedselect	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
sched_hint	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
server	---	r	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
session_id	---	---	---	---	---	---	---	---	---	---	---	---	r	r	r	r	r	---	---
Shell_Path_List	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
stagein	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
stageout	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
Stageout_status	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
stime	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Submit_arguments	---	---	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
substate	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
sw_index	---	r	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
umask	r, s	r, s	r	r	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r
User_List	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Variable_List	r, s	r, s	r	r, s	r, s	---	---	---	r, s	r	r	r	r	r	r	r	r	---	r, s

5.2.4.16 Table: Reading & Setting Vnode Attributes in Hooks

The following table shows the vnode attributes that can be read or set when the vnode object is retrieved via an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this attribute can be set but the action has no effect. See Table 4-1, “Execution Event Hook Timing,” on page 19 for more information about why some operations have no effect.

Table 5-7: Vnode Attributes Readable & Settable via Events

Vnode Attribute	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
comment	r, s	r, s	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
current_aoe	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---

Table 5-7: Vnode Attributes Readable & Settable via Events

Vnode Attribute	queuejob	modifyjob (before run)	movejob	runjob	resvsub	resv_end	periodic	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
hpcbp_stage_protocol	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_user_name	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
hpcbp_webservice_address	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
in_multivnode_host	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
jobs	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
license	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
license_info	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
Mom	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
name	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
no_multinode_jobs	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
ntype	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
pbs_version	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
pcpus	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
pnames	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
Port	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
Priority	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
provision_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
queue	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
resources_assigned	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r	r	---
resources_available	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
resv	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
resv_enable	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
sharing	---	---	---	---	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
state	---	---	---	r, s	---	---	---	r, s	r	r, s	r, s	r	r	r, s	r, s	r, s	r, s	r, s	---
topology_info	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	r	r	---

5.2.4.17 Table: Reading & Setting Reservation Attributes in resvsub and resv_end Hooks

Reservation attributes can be read and set through an event only in resvsub and resv_end hooks. No other hooks can read or set reservation attributes through an event. All hooks can read, but not set, all reservation attributes by retrieving the reservation object through the server, using pbs.server().resv(). The following table shows the reservation attributes that can be read or set when the reservation object is retrieved via an event, in a resvsub or resv_end hook:

Table 5-8: Reservation Attributes Readable & Settable in resvsub and resv_end Hooks

Reservation Attribute	resvsub	resv_end
Account_Name	r	r
Authorized_Groups	r, s	r
Authorized_Hosts	r, s	r
Authorized_Users	r, s	r
ctime	r	r
group_list	r, s	r
hashname	r	r
interactive	r, s	r
Mail_Points	r, s	r
Mail_Users	r, s	r
mtime	r	r
Priority	r	r
queue	r	r
reserve_count	r	r
reserve_duration	r, s	r
reserve_end	r, s	r
reserve_ID	r	r
reserve_index	r	r
reserve_job	r	r
Reserve_Name	r, s	r
Reserve_Owner	r	r
reserve_retry	r	r
reserve_rrule	r, s	r
reserve_start	r, s	r
reserve_state	r	r
reserve_substate	r	r
reserve_type	r	r

Table 5-8: Reservation Attributes Readable & Settable in resvsub and resv_end Hooks

Reservation Attribute	resvsub	resv_end
Resource_List	r, s	r
resv_nodes	r	r
server	r, s	r
User_List	r	r
Variable_List	r, s	r

5.2.4.18 Table: Reading & Setting Built-in Job Resources in Hooks

The following table shows the built-in members of the job's Resource_List attribute that can be read or set in each type of hook, when retrieving the object through an event.

An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, “Execution Event Hook Timing,” on page 19 for more information about why some operations have no effect. For more about custom resources, see section 5.2.4.1, “Using Built-in vs. Custom Resources in Hooks”, on page 44.

Table 5-9: Built-in Job Resources Readable & Settable by Hooks via Events

Resource in Resource_List	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
accelerator	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
accelerator_memory	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
accelerator_model	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
aoe	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
arch	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
cput	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
exec_vnode	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
file	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
host	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
max_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mem	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
min_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
mpiprocs	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
naccelerators	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nchunk	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
ncpus	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nice	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
nodect	r, s	r, s	r	r	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
nodes	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
ompthreads	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Table 5-9: Built-in Job Resources Readable & Settable by Hooks via Events

Resource in Resource_List	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pcput	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
pmem	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
pvmem	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
site	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
software	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
soft_walltime	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
start_time	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
vmem	r, s	r, s	r	r, s	r	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
vnode	r, s	r, s	r	r, s	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
vntype	r, s	r, s	r	r, s	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
walltime	r, s	r, s	r	r, s	r, s	---	---	---	r	r	r	r	r	r	r	r	r	---	r	---
PBScrayhost	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScraylabel_<label name>	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScraynid	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScrayorder	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
PBScrayseg	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

5.2.4.19 Table: Reading & Setting Vnode Resources in Hooks

The following table shows the built-in members of the vnode's resources_available attribute that can be read or set in each type of hook, when retrieving the object through an event. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this resource can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-10: Vnode Resources Readable & Settable by Hooks via Events

Resource in resources_available	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
accelerator	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
accelerator_memory	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
accelerator_model	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
aoe	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
arch	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
cput	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---

Table 5-10: Vnode Resources Readable & Settable by Hooks via Events

Resource in resources_available	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
exec_vnode	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
file	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
host	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
max_walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
mem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
min_walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
mpiprocs	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
naccelerators	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nchunk	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
ncpus	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nice	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nodect	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
nodes	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
ompthreads	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pcput	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
pvmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
site	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
software	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
start_time	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vmem	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vnode	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
vntype	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
walltime	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayhost	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScraylabel_<label name>	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScraynid	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayorder	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---
PBScrayseg	---	---	---	---	---	---	---	---	r	r	r	r	r	r	r	r	r	r, s	r, s	---

5.2.5 Using select and place in Hooks

All hooks can read, but not set, a job's select and place statements via pbs.server().job(), pbs.server().vnode(), pbs.server().queue(), etc. The following table shows the type of hook that can read or set a job's select and place statements, when retrieving the object through an event. An "*r*" indicates *read*, an "*s*" indicates *set*. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 5-11: Hooks that Can Read & Set Job select and place Statements via Events

Select or Place	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
Job place statement	r, s	r, s	r	r, s	r	---	---	---	---	---	r	r	r	r	r	---	---	---
Job select statement	r, s	r, s	r	r, s	r	---	---	---	---	---	r, s	r, s	r	r	r	---	---	---

5.2.5.1 How to Set select and place in Hooks

You must use the associated creation method to instantiate an object of the correct type with the desired value, then assign the object to the job. Syntax:

job.Resource_List["place"] = pbs.place("[arrangement]:[sharing]:[group]")

job.Resource_List["select"] = pbs.select("[N:]res=val[:res=val][+[N:]res=val[:res=val] ...]")

Example 5-12: Set a job's select and place directives:

```
jobB = pbs.event().job
jobB.Resource_List["place"] = pbs.place("pack:exclhost")
jobB.Resource_List["select"] = pbs.select("2:mem=2gb:ncpus=1+6:mem=8gb:ncpus=16")
```

See "pbs.select()" on page 148 and "pbs.place()" on page 147.

For modifying a job's select statement when allowing jobs access to extra vnodes, see section 6.13.3.24, "Method to Increment select Object Chunks", on page 148 and section 6.6.4.4, "Job Object Method to Release Vnodes", on page 128. For more about making jobs more reliable, see "Vnode Fault Tolerance for Job Start and Run" on page 434 of the PBS Professional Administrator's Guide.

5.2.5.2 Caveats for Using select and place in Hooks

You may want to check resource requests for a queuejob hook If a user submits a job using old `-lnodes` or `-lncpus` syntax, this is translated to a select statement, but only after a queuejob hook has run.

5.2.6 Restarting Scheduler Cycle After Hook Failure

You can restart the scheduler after an execjob_begin hook fails due to an alarm call or unhandled exception, or when the hook fails due to an internal error such as a full disk or not enough memory on the host, for example, a `malloc()` error. To restart the scheduler after failure of an execjob_begin hook, set the value of an execjob_begin or execjob_prologue hook's fail_action attribute to include "*scheduler_restart_cycle*".

```
# qmgr -c "set hook <hook_name> fail_action += scheduler_restart_cycle"
```

See section 5.1.9.2, "Using the fail_action Hook Attribute", on page 37.

5.2.7 Adding Custom Non-consumable Host-level Resources

You can add new custom host-level, non-consumable resources and set their values in resources_available for a vnode by using vnode_list[]in an exechost_startup hook. Any changes made this way are merged with those defined in a Version 2 vnode configuration file. Upon startup, MoM reads configuration files before executing the exechost_startup hook.

To add a new custom host-level resource, and set its value:

v = pbs.event().vnode_list[<vnode name>]

v.resources_available[<new_resource>] = <value>

The type of the resource is inferred from the value assigned to the resource. Python types map to PBS types as shown in the following table:

Table 5-12: Resource Types when Adding via vnode_list

Python Type	Type
int	*Long*
str	*String*
bool	*Boolean*
pbs.size	*Size*
pbs.duration	*Long*
float	*Float*
Any Python type without an explicit match	*String*

You must also make the resource usable by the scheduler: see section 5.14.2.6, “Allowing Jobs to Use a Resource”, on page 265.

To delete a custom resource created in a hook, use qmgr. See section 5.14.2.4.iii, “Deleting Custom Resources”, on page 263.

Example 5-13: Adding custom resources:

If you have these instructions in a hook:

```
vn.resources_available["fab_int"] = 9
vn.resources_available["fab_str"] = "happy"
vn.resources_available["fab_bool"] = False
vn.resources_available["fab_size"]  = pbs.size("7mb")
vn.resources_available["fab_time"] = pbs.duration("00:30:00")
vn.resources_available["fab_float"] = 7.0
```

This is equivalent to the following qmgr commands:

```
qmgr -c "create resource fab_int type=long,flag=h"
qmgr -c "create resource fab_str type=string,flag=h"
qmgr -c "create resource fab_bool type=boolean,flag=h"
qmgr -c "create resource fab_size type=size,flag=h"
qmgr -c "create resource fab_time type=long,flag=h"
qmgr -c "create resource fab_float type=float,flag=h"
```

5.2.8 Printing And Logging Messages

Hooks can log a custom string in the local daemon's log, at message log event class *pbs.LOG_DEBUG (0x0004)*. This is done using the pbs.logjobmsg(job ID, message) facility. See "pbs.logjobmsg()" on page 151.

Hooks can specify a message for use when the corresponding action is rejected. This message is printed to `stderr` by the command that triggered the event, and is printed in the daemon's log. This is done using the pbs.event().reject(<message>) function. See "pbs.event().reject()" on page 116 for information on how to specify a rejection message.

Hooks cannot directly print to `stdout` or `stderr`, or read from `stdin`. See section 5.3.8.1, "Avoid Hook File I/O", on page 71, and section 8.10.2.8, "Hooks Attempting I/O", on page 225.

5.2.9 Capturing Return Code

To capture an application's return code, you capture the return code in Python and then return it from the hook. You can use the Python subprocess module. Here is an example snippet:

```
import sys
if "<path to subprocess module>" not in sys.path:
    sys.path.append("<path to subprocess module>")
import subprocess

try:
    retcode = subprocess.call("mycommand myarg", shell=True)
except OSError:
    retcode = -1

return retcode
```

5.2.10 When You Need Persistent Data

If you need your data to be persistent, your hook(s) must be able to save and retrieve the information. Hooks are stateless, and each invocation of a hook has no knowledge of prior state of jobs, vnodes, etc. If you want to retain state across invocations of a hook, you can have the hook write what you need to a well-known location such as `PBS_HOME`. When the hook is invoked, it can read in the data, and before the hook exits, it can update the data. You can use whatever format you like for the data.

5.2.11 Setting Up Job Environment on Sisters

If you need to set up the job's environment on sister MoMs, use an execjob_begin hook. This hook can set up the desired environment on sister MoMs so that the job can use the new environment.

If job tasks are spawned on sister MoMs via a tightly-integrated MPI that uses `tm_spawn()`, any execjob_prologue and execjob_launch hooks run on the sister MoMs. However, if job tasks are started using `pbs_attach()`, execjob_attach and execjob_prologue (on the first task attached) hooks run on sister MoMs instead. For a detailed description of the order in which hooks run on the primary and secondary execution hosts, see Table 4-1, "Execution Event Hook Timing," on page 19.

The old-style prologue runs only on the primary execution host; you cannot use it to set up the environment on sister MoMs.

All job tasks running on vnodes managed by the same MoM get the same environment.

5.2.12 Offlining Bad Vnodes

5.2.12.1 General Method for Offlining Bad Vnodes

If you need to offline a bad vnode where a hook is running:

```
this_vnode = pbs.event()vnode_list[pbs.get_local_nodename()]
this_vnode.state = pbs.ND_OFFLINE
this_vnode.comment = "offlining this vnode"
```

5.2.12.2 Offlining Vnodes Associated with an Event

For example, in a job-related event, you can offline the vnodes and reject the job:

```
for v in pbs.event().vnode_list.keys():
    pbs.event().vnode_list[v].state = pbs.ND_OFFLINE
    pbs.event().vnode_list[v].comment = "Offlining this vnode"
pbs.event().reject("Job tried to run on bad vnodes")
```

5.2.12.3 Using List of Failed Vnodes to Offline Vnodes that Have Gone Bad During Start or Run

For each execjob_prologue and execjob_launch event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. This list can include those vnodes from sister MoMs that failed to join the job, that rejected an execjob_begin hook or execjob_prologue hook request, or that encountered a communication error while the primary MoM was polling the sister MoM host. PBS records this list in the pbs.event().vnode_list_fail[] hook parameter. This parameter is a *dict* (dictionary of pbs.vnode objects keyed by vnode name).

You can use a hook to walk through this list and offline the bad vnodes. Here is a code snippet:

```
for vn in e.vnode_list_fail:
   v = e.vnode_list_fail[vn]
   pbs.logmsg(pbs.LOG_DEBUG, "offlining %s" % (vn,))
   v.state = pbs.ND_OFFLINE
```

5.2.12.4 Offlining and Clearing Vnodes Using the fail_action Hook Attribute

The way this works is that when a vnode fails a health check in an execjob_begin hook, it is offlined via "offline_vnodes". Once the vnode is offlined, no other jobs are sent to the vnode, so no other execjob_begin hooks will run until the vnode is cleared. You then use "clear_vnodes_upon_recovery" in an exechost_startup hook which runs when the MoM starts up or is HUPed.

5.2.12.4.i Offlining Vnodes Using the fail_action Hook Attribute

You can offline vnodes when an execjob_prologue, execjob_begin or exechost_startup hook fails due to an alarm call or unhandled exception, or when the hook fails due to an internal error such as a full disk or not enough memory on the host, for example, a `malloc()` error.

To offline vnodes upon failure, set the value of the hook's fail_action attribute to include "*offline_vnodes*". This marks the vnodes managed by the hook's MoM as *offline*.

```
# qmgr -c "set hook <hook_name> fail_action += offline_vnodes"
```

When a vnode is offlined using the fail_action attribute, the vnode's comment attribute is set to an explanation:

```
"offlined by hook <hook_name> due to hook error"
```

See section 5.1.9.2, “Using the fail_action Hook Attribute”, on page 37.

5.2.12.4.ii Clearing Vnodes Using the fail_action Hook Attribute

When an exechost_startup hook runs successfully and does not encounter any uncaught exception or alarm timeout, you can clear the *offline* state from vnodes that were previously marked *offline* via fail_action.

To clear the *offline* state from vnodes that were previously offlined via the *"offline_vnodes"* fail_action attribute, set the value of the exechost_startup hook's fail_action attribute to include "*clear_vnodes_upon_recovery*". This clears the *offline* state from the vnodes managed by the hook's MoM.

```
# qmgr -c "set hook <hook_name> fail_action += clear_vnodes_upon_recovery"
```

If you have fixed your execjob_begin script, and want to send jobs again to the vnodes managed by the MoM where the script runs, clear the *offline* states and comments from the vnodes managed by that MoM:

- Clear the *offline* state:

  ```
  # pbsnodes -r <MoM host>
  ```

- Clear the comment:

  ```
  qmgr -c "u n <vn1>,<vn2>,... comment"
  ```

  Or for long lists of vnodes:

  ```
  # qmgr -c "unset node `pbsnodes -vl | awk '{if( NR == 1 ) {printf "%s", $1} else {printf ",%s",
     $1}}'` comment"
  ```

You can write an exechost_periodic hook that monitors the states of the vnodes, so that when it finds offlined vnodes with vnode comment messages matching "`offlined by hook…`", the hook clears the comment and *offline* states.

See section 5.1.9.2, “Using the fail_action Hook Attribute”, on page 37.

5.3 Advice and Caveats for Writing Hooks

5.3.1 Rules for Hook Access and Behavior

The following are rules and recommendations for writing hooks:

- When modifying hooks or their configuration files, do not edit the .CF or .PY files directly. You might think this is a shortcut; it's not. Changes to execution hooks will not be propagated to the MoMs.
- Use only the documented interfaces. Hooks which access PBS information or modify PBS in any way except through these interfaces are erroneous and unsupported.
- Do not attempt to manipulate the hook stored by PBS, except as specified in Chapter 7, "Built-in Hooks", on page 155.
- Don't delete attributes.
- Don't change environment variables set by PBS. See "Environment Variables" on page 230 of the PBS Professional Reference Guide for a list of these environment variables.
- Do not try to access the following (a well-written, portable hook will not depend on any of the following information):
  - Server configuration information: `qmgr`, `resourcedef` and `pbs.conf`
  - Scheduling information: `qmgr`, `sched_config, fairshare, dedicated, holidays`
- Do not write hooks that depend on the behavior of other hooks.
- Do not make assumptions about the value of PATH; use `"import sys"` and modify `sys.path`
- Do not make assumptions about the value of the current working directory.
- For information about `umask`, see"qalter" on page 127 of the PBS Professional Reference Guide, "qsub" on page 213 of the PBS Professional Reference Guide, and "Job Attributes" on page 328 of the PBS Professional Reference Guide.
- Do not depend on order of execution of unrelated hooks. For example, do not depend on one job submission's queuejob hooks running entirely before another job submission's queuejob hooks. It is not guaranteed that all of one job's hooks will finish before another job's hooks start.
- The Resource_List attribute, like others, is a pbs.pbs_resource. These objects support a restricted set of operations. They can reference values by index. Other features, such as has_key(), are not available. See section 6.13.3.19, "Method to Create or Set Resource List", on page 147.
- Hooks which execute PBS commands are erroneous and unsupported. The behavior of executing PBS commands inside a hook is undefined (and is likely to cause the hook to hang).

5.3.2 Check for Parameter Validity

To make hook scripts more robust, check first for the validity of the event parameters before using them, by comparing against *None*:

```
if pbs.event().job != None:
If pbs.event().job_o != None:
If pbs.event().src_queue != None:
If pbs.event().resv != None:
If pbs.event().vnode != None:
If pbs.event().aoe != None:
```

5.3.2.1 Resource Requests and queuejob Hooks

You may want to check resource requests for a queuejob hook If a user submits a job using old `-lnodes` or `-lncpus` syntax, this is translated to a select statement, but only after a queuejob hook has run.

5.3.2.2 Example of Checking Validity

```
% cat t2245.ty
import pbs
e = pbs.event()
if e.type == pbs.QUEUEJOB && (e.job == None):
    e.reject("Event Job parameter is unset!")
elif e.type == pbs.MODIFYJOB && ((e.job == None) || (e.job_o == None)):
    e.reject("Event Job or Job_o parameter is unset!")
elif e.type == pbs.RESVSUB && (e.resv == None):
    e.reject("Event Resv parameter is unset!")
elif e.type == pbs.RUNJOB && (e.job == None):
    e.reject("Event Job parameter is unset!")
```

5.3.3 Make Changes Only On Acceptance

We recommend that your hook does not make changes unless the hook accepts its event. You do not want to have to back changes out upon a reject().

5.3.4 Offline Vnodes when exechost_startup Hook Rejects

We recommend that before calling pbs.event().reject() in an exechost_startup hook, you set the vnodes managed by the local MoM offline with an accompanying comment. This stops jobs from being sent to the affected vnodes. For example:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
    vnlist[v].state = pbs.ND_OFFLINE
    vnlist[v].comment = "bad configuration"
pbs.event().reject("not accepting jobs")
```

5.3.5 Minimize Unnecessary Steps

To speed up your hooks, move any steps to where they are used the fewest times possible. For example, if you retrieve several pieces of information about a job, but only use them if one of them fits a certain criterion, put the bulk of the information-retrieval steps in the section where you do the work on the job.

5.3.6 Use Fast Operations

Some of the examples we provide could be faster. Instead of using "==", you can use the bitwise ampersand operator ("&").

5.3.7 Avoiding Interference with Normal Operation

5.3.7.1 Treat SystemExit as a Normal Occurrence

Both pbs.event().accept() and pbs.event().reject() terminate hook execution by throwing a SystemExit exception. A "try...except" clause without arguments will catch all exceptions. If hook content appears in a "try except " clause, add the following to treat SystemExit as a normal occurrence:

```
except SystemExit:
    pass
```

Here is an example of an except clause that will catch SystemExit:

```
try:
    ...
except:
    ...
```

In the above case, we need to add the except SystemExit, so that it will look like this:

```
try:
    ...
except SystemExit:
    pass
except:
    ...
```

If the existing code has a specified exception, we don't need to add "except SystemExit:", since this hook script is only catching one particular exception and will not match SystemExit. For example:

```
try:
    ...
except pbs.BadAttributeValueError:
    ...
```

5.3.7.2 Allow the Server to Modify Jobs

The server uses the `qalter` command during normal operation to modify jobs. Therefore, if you have a modifyjob hook script, make sure you do not interfere with `qalter` commands issued by the server. Catch these cases by starting the hook with an `if` clause that accepts modification of jobs by PBS:

```
e = pbs.event()
if e.requestor in [ "PBS_Server" ]:
    e.accept()
```

While the scheduler also uses the `qalter` command to modify jobs, this does not trigger any modifyjob hooks.

5.3.7.3 Stay Within the Scheduler Alarm Time

Consider setting hook alarm values in runjob hooks so that they do not unduly delay the scheduler. The scheduler will wait for a hook to finish executing. The scheduler's cycle time has a default value of *20 minutes*, and is specified in the scheduler's sched_cycle_length attribute.

5.3.8 Avoiding Problems

5.3.8.1 Avoid Hook File I/O

When the PBS server is running, `stdout`, `stderr`, and `stdin` are closed. A hook script attempting I/O will get an exception. To avoid this, redirect input and output to a file. See section 8.10.2.8, "Hooks Attempting I/O", on page 225.

5.3.8.2 Avoid Contacting Bad Host

Be careful not to specify a bad host in <job ID> in pbs.event().job.depend. If it references a non-existent or heavily loaded PBS server, the current PBS server could hang for a few minutes as it tries to contact the bad host. For example:

```
pbs.event().job.depend = pbs.depend("after:23.bad_host")
```

The PBS server could hang while trying to contact "bad_host".

5.3.8.3 Avoid os._exit() Python Function

Do not use the os._exit() Python function. It will cause the PBS server to exit.

5.3.8.4 Avoid Attempting to Log Message Using Bad Job ID

If the pbs.logjobmsg() method is passed a bad job ID, it raises a Python ValueError.

5.3.8.5 Avoid Taking Up Lots of Memory

Certain function calls in PBS Python hooks are expensive to use in terms of memory. If they are called repeatedly in loops, they can use up a lot of memory, potentially causing the server to hang or crash. For example, the following is expensive since each iterative call to pbs.server().vnodes() causes internal allocation of memory, which won't be freed until after the hook executes.

In order to avoid this, produce the output only once, save it to memory, and iterate using the copy. For example:

```
vnl = []
vni = pbs.server().vnodes()
for vn in vni:
    pbs.logmsg(pbs.LOG_DEBUG, "found vn.name=%s" %(vn.name))
    vnl.append(vn)
```

The following functions in PBS Python hooks return iterators, and should be used carefully:

- Iterate over a list of jobs:

  pbs.server().jobs()

  pbs.queue.jobs()
- Iterate over a list of queues:

  pbs.server().queues()
- Iterate over a list of vnodes:

  pbs.server().vnodes()
- Iterate over a list of reservations:

  pbs.server().resvs()

5.3.8.6 Testing Vnode State

To see whether a vnode has a particular state set:

```
If v.state == pbs.ND_OFFLINE:
    pbs.logmsg(pbs.LOG_DEBUG, "vnode %s is offline!" % (v.name))
```

5.3.9 Restrictions

5.3.9.1 Local Server Only

Hooks cannot access a server other than the local server. Hooks also cannot specify a non-default server. So for example if a job submission specifies a queue at a server other than the default, the hook can allow that submission, or can change it to the default server, but cannot change it to another non-default server.

5.3.9.2 Dictionary Data Type Restriction

The Python types listed as dictionaries, such as pbs.event().env, support a restricted set of operations. They can reference values by index. Other features, such as has_key(), are not available.

5.3.10 Scheduling Impact of Hooks

5.3.10.1 Effect of runjob Hooks on Preemption

With preemption turned on, the scheduler preempts low-priority jobs to run a high-priority job. If the high-priority job is rejected by a runjob hook, then the scheduler undoes the preemption of the low-priority jobs. Suspended jobs are resumed, and checkpointed jobs are restarted.

5.3.10.2 Effect of runjob Hooks with Strict Ordering

When strict_ordering is set to *True* and backfill_depth is set to *0*, a most-deserving job that is repeatedly rejected by a runjob hook will prevent other jobs from being able to run. A well-written hook would put the job on hold or requeue the job with a later execution time to prevent idling the system.

5.3.10.3 Effect of runjob Hooks with round_robin and by_queue

With round_robin and by_queue set to *True*, a job continually rejected by a runjob hook may prevent other jobs from the same queue from being run. A well-written hook would put the job on hold or requeue the job with a later execution time to allow other jobs in the same queue to be run.

A runjob hook's performance directly affects the responsiveness of the PBS scheduler. Consider carefully the trade-off between the work such a hook needs to do and your scheduler's required performance.

5.3.10.4 Peer Scheduling and Hooks

When a job is pulled from one complex to another, the following happens:

- Hooks are applied at the new complex as if the job had been submitted locally
- Any movejob hooks at the furnishing server are run

5.3.10.5 Performance Considerations

5.3.10.5.i Cost of Accessing Data

- Using pbs.server() to get data about server, queues, jobs, vnodes, or reservations can be slow if run in an execution hook. This is because of the overhead involved when the function has to directly connect to the server and pass requests (via TCP). However, you can speed up reading of custom job resources by setting the m flag. See "Allowing Execution Hooks to Read Custom Job Resources Faster" on page 261 in the PBS Professional Administrator's Guide.
- Making queries to pbs.server().resources_available[] can be slow.

5.3.10.5.ii Cost of Different Hooks

- Any queuejob hooks execute once per job submission
- Any runjob hooks execute once per attempt to run a job, after the scheduler has found a place for it

What this means to the hook writer:

- Your queuejob hooks can generally get away with longer run times
- Any hook that needs to listen to queuejob events needs to be quick to decide whether it is needed or not

For a fast hook, avoid these:

- Running external commands
- Network connections
- File I/O and logging
- Storing information in server or vnode settings
- Using pbs.server().resources_available
- Iterating over the entire set of vnodes or jobs using pbs.server().vnodes() or pbs.server().jobs().

In addition, see section 5.3.5, "Minimize Unnecessary Steps", on page 69 and section 5.3.6, "Use Fast Operations", on page 69.

5.3.10.6 Effect of Hooks on Job Eligible Time

When eligible time is enabled and a job is blocked by a queuejob hook, the job accrues initial_time. When a job is accepted or rejected by modifyjob or movejob hooks, the job continues to accrue whatever kind of time it was accruing. When a job is requeued by a runjob hook or an execution event hook, the scheduler evaluates what kind of time the job should accrue based on resources and policy.

5.3.11 Windows Caveats

5.3.11.1 Special Characters in Pathnames

On Windows, where backslashes may appear in pathnames, escape each backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

5.3.11.2 Importing and Exporting Hooks

If the name of <*input_file*> contains spaces, <*input file*> must be quoted.

5.3.11.3 Modifying Events

On Windows, in a multi-vnoded job, be careful modifying pbs.event().progname and pbs.event().argv[] parameters; some values are tacked on by `pbs_mom` and are required. See section 6.3.3.1, "Modifying progname or argv[] Under Windows", on page 115.

5.3.11.4 Using Sleep in a Hook Script

Under Windows, the PBS server or MoM cannot interrupt a hook script executing the Python time.sleep(). The server needs to be able to interrupt the script if the script reaches its timeout. In order to be able to interrupt the script, create a sleep that incrementally sleeps for 1 second. The server can then interrupt the hook script in between the sleeps. For example:

```
import time
def mysleep(sec):
for i in range(sec):
    time.sleep(1)
mysleep(30)          <-- pseudo sleep for 30 seconds
```

6
Hook Objects and Methods

Contents

6.1 The pbs Module . . . 76
6.2 PBS Interface Objects . . . 76
6.2.1 Table of PBS Interface Objects . . . 76
6.2.2 Maps of Object Members and Methods . . . 85
6.3 Event Objects . . . 87
6.3.1 Event Types . . . 87
6.3.2 Event Object Members . . . 108
6.3.3 Event Object Member Caveats . . . 115
6.3.4 Event-only Methods . . . 116
6.3.5 Event Object Method Caveats . . . 116
6.3.6 Examples of Using Event Objects . . . 117
6.4 Server Objects . . . 118
6.4.1 Server Object Members . . . 118
6.4.2 Setting Server Object Members . . . 119
6.4.3 Examples of Using Server Object Members . . . 119
6.4.4 Server Object Methods . . . 119
6.5 Queue Objects . . . 121
6.5.1 Queue Object Members . . . 121
6.5.2 Queue Object Methods . . . 122
6.5.3 Queue Type Constant Objects . . . 122
6.6 Job Objects . . . 122
6.6.1 Job Object Members . . . 123
6.6.2 Setting Job Attributes . . . 127
6.6.3 Examples of Using Job Object Members . . . 127
6.6.4 Job Object Methods for Execution Hooks . . . 127
6.7 The exec_vnode Object . . . 129
6.7.1 The exec_vnode Object Members . . . 129
6.7.2 Using pbs.vchunk Objects in exec_vnode . . . 130
6.7.3 Restrictions on exec_vnode Objects . . . 131
6.8 Chunk Objects . . . 131
6.8.1 Chunk Object Members and Methods . . . 131
6.9 Reservation Objects . . . 131
6.9.1 Reservation Object Members . . . 132
6.9.2 Reservation State Constant Objects . . . 133
6.10 Vnode Objects . . . 133
6.10.1 Vnode Object Members . . . 134
6.10.2 Vnode Type Constant Objects . . . 134
6.10.3 Vnode Sharing Constant Objects . . . 135
6.10.4 Vnode State Constant Objects . . . 135
6.11 Configuration File Objects . . . 135
6.11.1 Variable Containing Hook Configuration File Path . . . 135
6.11.2 Dictionary of PBS Configuration File Entries . . . 136
6.12 Constant Objects . . . 140
6.13 Object Members and Methods . . . 141

6.13.1 PBS Objects and Object Members 141
6.13.2 Methods Available in Events 142
6.13.3 PBS Types and Their Methods 143
6.13.4 Global Methods 151

6.1 The pbs Module

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, members, and methods. You can operate on the objects and use the methods in your Python code. In order to use the pbs module, you must begin your Python code by importing the pbs module. For example, in a script that modifies a job:

```
import pbs
pbs.event().job.comment="Modified this job"
```

For the contents of the pbs module, see section 4.5, “Python Modules and PBS”, on page 24.

6.2 PBS Interface Objects

The PBS interface contains different kinds of objects:

- Objects to represent PBS entities, e.g. jobs, server, queues, vnodes, reservations, events, log messages, etc.
- Objects to represent job, server, vnode, queue, and reservation attributes.
- Objects to represent arguments to PBS commands, PBS version information, etc.
- Constant objects to represent event types, states, log event classes, queue types, and exceptions.

6.2.1 Table of PBS Interface Objects

PBS provides a set of interface objects for use in hooks. The following table lists all of the PBS objects in alphabetical order. Each of these objects is described in detail later in the chapter.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.acl Represents a PBS ACL . See section 6.13.3.1, “Method to Create or Set ACL”, on page 143.
pbs.args Represents a space-separated list of PBS arguments to commands such as qsub, qdel. See "Method to Create or Set Command Argument List” on page 143.
pbs.argv[] Argument strings to be passed to the program executed for the job. See section 6.3.2.2, “Job Program Arguments Event Member”, on page 109.
pbs.BadAttributeValueError Raised when setting the member value of a pbs.* object and the value given is invalid. See "Table of Exceptions” on page 43

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.BadAttributeValueTypeError Raised when setting the member value of a pbs.* object and the value type is invalid. See "Table of Exceptions" on page 43
pbs.BadResourceValueError Raised when setting the resource value of a pbs.* object and the value given is invalid. See "Table of Exceptions" on page 43
pbs.BadResourceValueTypeError Raised when setting the resource value of a pbs.* object and the value type is invalid. See "Table of Exceptions" on page 43
pbs.checkpoint Represents a job's checkpoint attribute. See "Job Checkpoint Attribute Member" on page 124
pbs.depend Represents a job's dependency attribute. See "Job depend Attribute Member" on page 124.
pbs.duration Represents a time interval. See "Method to Create or Set Duration from Time String or Integer" on page 144.
pbs.email_list Represents the set of users to whom mail may be sent. Example: Job's Mail_Users attribute. See "Method to Create or Set Email List" on page 144
pbs.env[] Dictionary of environment variables. See section 6.3.2.5, "Job Environment Event Member", on page 110.
pbs.event Represents a PBS event. See "Event Objects" on page 87
pbs.EventIncompatibleError Raised when referencing a nonexistent member in pbs.event(). See "Table of Exceptions" on page 43.
pbs.EXECHOST_PERIODIC Type for an exechost_periodic hook event. See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.
pbs.EXECHOST_STARTUP Type for an exechost_startup hook event. See section 6.3.1.17, "exechost_startup: Event When Execution Host Starts Up", on page 106.
pbs.EXECJOB_ATTACH Type for an execjob_attach hook event. See section 6.3.1.11, "execjob_attach: Event when pbs_attach() runs", on page 100.
pbs.EXECJOB_BEGIN Type for an execjob_begin hook event. See section 6.3.1.8, "execjob_begin: Event when Execution Host Receives Job", on page 96.
pbs.EXECJOB_END Type for an execjob_end hook event. See section 6.3.1.16, "execjob_end: Event After Job Cleanup", on page 105.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.EXECJOB_EPILOGUE Type for an execjob_epilogue hook event. See section 6.3.1.15, "execjob_epilogue: Event Just After Killing Job Tasks", on page 104.
pbs.EXECJOB_LAUNCH Type for an execjob_launch hook event. See section 6.3.1.10, "execjob_launch: Event when Execution Host Receives Job", on page 98.
pbs.EXECJOB_POSTSUSPEND Type for an execjob_postsuspend hook event. See section 6.3.1.12, "execjob_postsuspend: Event Just After Suspending Job", on page 101.
pbs.EXECJOB_PRERESUME Type for an execjob_preresume hook event. See section 6.3.1.13, "execjob_preresume: Event Just Before Resuming Job", on page 102.
pbs.EXECJOB_PRETERM Type for an execjob_preterm hook event. See section 6.3.1.14, "execjob_preterm: Event Just Before Killing Job Tasks", on page 103.
pbs.EXECJOB_PROLOGUE Type for an execjob_prologue hook event. See section 6.3.1.9, "execjob_prologue: Event Just Before Execution of Top-level Job Process", on page 97.
pbs.PERIODIC Type for a periodic hook. See section 6.3.1.7, "periodic: Periodic Event at Server Host", on page 95.
pbs.exec_host Represents a job's exec_host attribute. See "job.exec_host" on page 124 .
pbs.exec_vnode Represents a job's exec_vnode attribute. See "job.exec_vnode" on page 124 .
pbs.group_list Represents a list of group names. See "job.group_list" on page 124.
pbs.hold_types Represents the Hold_Types attribute of a job. See "job.Hold_Types" on page 124.
pbs.job Represents a PBS job. See "Job Objects" on page 122
pbs.job_list[] List of pbs.job objects. See "pbs.event().job_list" on page 111.
pbs.job_sort_formula Represents the job_sort_formula server attribute. See "pbs.job_sort_formula()" on page 145
pbs.JOB_STATE_BEGUN Job arrays only. Job array has started. See "Job job_state Attribute Member" on page 125

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.JOB_STATE_EXITING Job is exiting after having run. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_EXPIRED Subjobs only. Subjob is finished (expired). See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_FINISHED Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution. See "Job job_state Attribute Member" on page 125.
pbs.JOB_STATE_HELD Job is held. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_MOVED Job has been moved to another server. See "Job job_state Attribute Member" on page 125.
pbs.JOB_STATE_QUEUED Job is queued, eligible to run or be routed. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_RUNNING Job is running. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_SUSPEND Job is suspended by server. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_SUSPEND_USERACTIVE Job is suspended due to workstation becoming busy. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_TRANSIT Job is in transit. See "Job job_state Attribute Member" on page 125
pbs.JOB_STATE_WAITING Job is waiting for its requested execution time to be reached, or the job's stagein request has failed. See "Job job_state Attribute Member" on page 125
pbs.join_path Represents the job's Join_Path attribute. See "Job Join_Path Attribute Member" on page 125.
pbs.keep_files Represents the Keep_Files job attribute. See "Job Keep_Files Attribute Member" on page 125
pbs.license_count Represents a set of licensing-related counters. Server attribute. See section 6.13.3.14, "Method to Create or Set license_count Object", on page 146.
pbs.LOG_DEBUG Log event class. See "Message Log Level Objects" on page 152
pbs.LOG_ERROR Log event class. See "Message Log Level Objects" on page 152

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.LOG_WARNING Log event class. See "Message Log Level Objects" on page 152
pbs.mail_points Represents the Mail_Points attribute of a job. See "Job Mail_Points Attribute Member" on page 126.
pbs.MODIFYJOB The modifyjob hook event type. Triggered by `qalter` or `pbs_alterjob()` API call. Not triggered by scheduler job modification. See "Event Types" on page 87.
pbs.MOVEJOB The movejob hook event type. Triggered by `qmove` or `pbs_movejob()` API call. See "Event Types" on page 87
pbs.ND_BUSY Represents *busy* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_DEFAULT_EXCL Represents *default_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_DEFAULT_SHARED Represents *default_shared* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_DOWN Represents *down* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_FORCE_EXCL Represents *force_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_FREE Represents *free* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_IGNORE_EXCL Represents *ignore_excl* sharing vnode attribute value. See section 6.10.3, "Vnode Sharing Constant Objects", on page 135.
pbs.ND_JOBBUSY Represents *job-busy* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_JOB_EXCLUSIVE Represents *job-exclusive* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_OFFLINE Represents *offline* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_PBS Represents *pbs* value for vnode ntype attribute. See section 6.10.2, "Vnode Type Constant Objects", on page 134
pbs.ND_PROV Represents *provisioning* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.ND_RESV_EXCLUSIVE Represents *resv-exclusive* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_STALE Represents *stale* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_STATE_UNKNOWN Represents *state-unknown, down* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_UNRESOLVABLE Represents *unresolvable* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.ND_WAIT_PROV Represents *wait-provisioning* vnode state. See section 6.10.4, "Vnode State Constant Objects", on page 135.
pbs.node_group_key Represents the node_group_key attribute. See "Method to Create or Set node_group_key Object" on page 146.
pbs.path_list Represents a list of pathnames. See "Method to Create or Set path_list Object" on page 146.
pbs.pbs_conf[] Dictionary of entries in `pbs.conf`. See "pbs.pbs_conf[]" on page 136.
pbs.pbs_resource List of resource names and values. See section 6.13.3.19, "Method to Create or Set Resource List", on page 147.
pbs.pid Represents the process ID of a process belonging to a job.
pbs.place Represents the place specification when submitting a job. See section 6.13.3.20, "Method to Create or Set place Object", on page 147.
pbs.progname Path of job shell or executable. See section 6.3.2.15, "Job Executable Event Member", on page 112.
pbs.QTYPE_EXECUTION Represents *execution* value for queue_type queue attribute. See "Queue Type Constant Objects" on page 122
pbs.QTYPE_ROUTE Represents *route* value for queue_type queue attribute. See "Queue Type Constant Objects" on page 122
pbs.queue Represents a PBS queue. See "Queue Objects" on page 121
pbs.QUEUEJOB The queuejob hook event type. Triggered by `qsub` or `pbs_submit()` API call. See section 6.3.1.3, "queuejob: Event when Job is Queued", on page 91.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.range Represents a range of numbers referring to job array indices. See section 6.13.3.21, “Method to Create or Set range Object”, on page 148.
pbs.resv Represents a PBS reservation. See "Reservation Objects” on page 131
pbs.RESVSUB The resvsub hook event type. Triggered by `pbs_rsub` or `pbs_submitresv()` API call. See section 6.3.1.1, “resvsub: Event when Reservation is Created”, on page 90.
pbs.RESV_END The resv_end hook event type. Triggered by end of reservation. See section 6.3.1.2, “resv_end: Event when Reservation Ends”, on page 90.
pbs.RESV_STATE_BEING_DELETED The reservation state *RESV_BEING_DELETED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_CONFIRMED The reservation state *RESV_CONFIRMED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_DEGRADED The reservation state *RESV_DEGRADED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_DELETED The reservation state *RESV_DELETED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_DELETING_JOBS The reservation state *RESV_DELETING_JOBS.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_FINISHED The reservation state *RESV_FINISHED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_NONE The reservation state *RESV_NONE.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_RUNNING The reservation state *RESV_RUNNING.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_TIME_TO_RUN The reservation state *RESV_TIME_TO_RUN.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_UNCONFIRMED The reservation state *RESV_UNCONFIRMED.* See "Reservation State Constant Objects” on page 133
pbs.RESV_STATE_WAIT The reservation state *RESV_WAIT.* See "Reservation State Constant Objects” on page 133
pbs.route_destinations Represents route_destinations queue attribute. See "Method to Create or Set route_destinations Object” on page 148.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.RUNJOB The runjob hook event type. Triggered by `qrun` or `pbs_runjob()` API call. See section 6.3.1.6, “runjob: Event Before Job is Received by MoM”, on page 94.
pbs.select Represents the select specification when submitting a job. See "Method to Create or Set select Object” on page 148.
pbs.server Represents the local PBS server. See"Server Objects” on page 118
pbs.size Represents a PBS size type. See "Method to Create or Set size Object” on page 150.
pbs.software Represents a site-dependent software specification resource. See "Method to Create or Set Software Resource Object” on page 150.
pbs.staging_list Represents a list of file stagein or stageout parameters. See "Job stagein and stageout Attribute Members” on page 126.
pbs.state_count Represents a set of job-related state counters. See "Method to Create or Set state_count Object” on page 151.
pbs.SV_STATE_ACTIVE Server state is *Scheduling*. See "Server State Member” on page 119
pbs.SV_STATE_HOT Server state is *Hot_Start*. See "Server State Member” on page 119
pbs.SV_STATE_IDLE Server state is *Idle*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTDEL Server state is *Terminating, Delayed*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTIMM Server state is *Terminating*. See "Server State Member” on page 119
pbs.SV_STATE_SHUTSIG Server state is *Terminating*. See "Server State Member” on page 119
pbs.UnsetAttributeNameError Raised when referencing a non-existent member name of a pbs.* object. See "Table of Exceptions” on page 43
pbs.UnsetResourceNameError Raised when referencing a non-existent resource name of a pbs.* object. See "Table of Exceptions” on page 43
pbs.user_list Represents a list of user names. See section 6.13.3.29, “Method to Create or Set user_list Object”, on page 151.
pbs.vchunk Represents a job chunk. See section 6.8, “Chunk Objects”, on page 131.

Table 6-1: PBS Interface Objects

PBS Interface Object Description
pbs.version Represents version information for PBS. See section 6.13.3.30, “Method to Create or Set PBS Version Object”, on page 151.
pbs.vnode Represents a PBS vnode. See section 6.10, “Vnode Objects”, on page 133.
pbs.vnode_list[] Represents a list of pbs.vnode objects. See section 6.3.2.22, “The Vnode List Event Member”, on page 114
pbs.vnode_list_fail[] Represents a list of unhealthy vnodes as pbs.vnode objects. See section 6.3.2.23, “The Failed Vnode List Event Member”, on page 115
SystemExit Raised when accepting or rejecting an action. See "Table of Exceptions” on page 43

6.2.2 Maps of Object Members and Methods

Figure 6-1 shows a map of the PBS Python objects. All hook event objects have the methods listed in "global methods". Each object also has its own members and methods, as shown. We expand hook event objects in Figure 6-2.

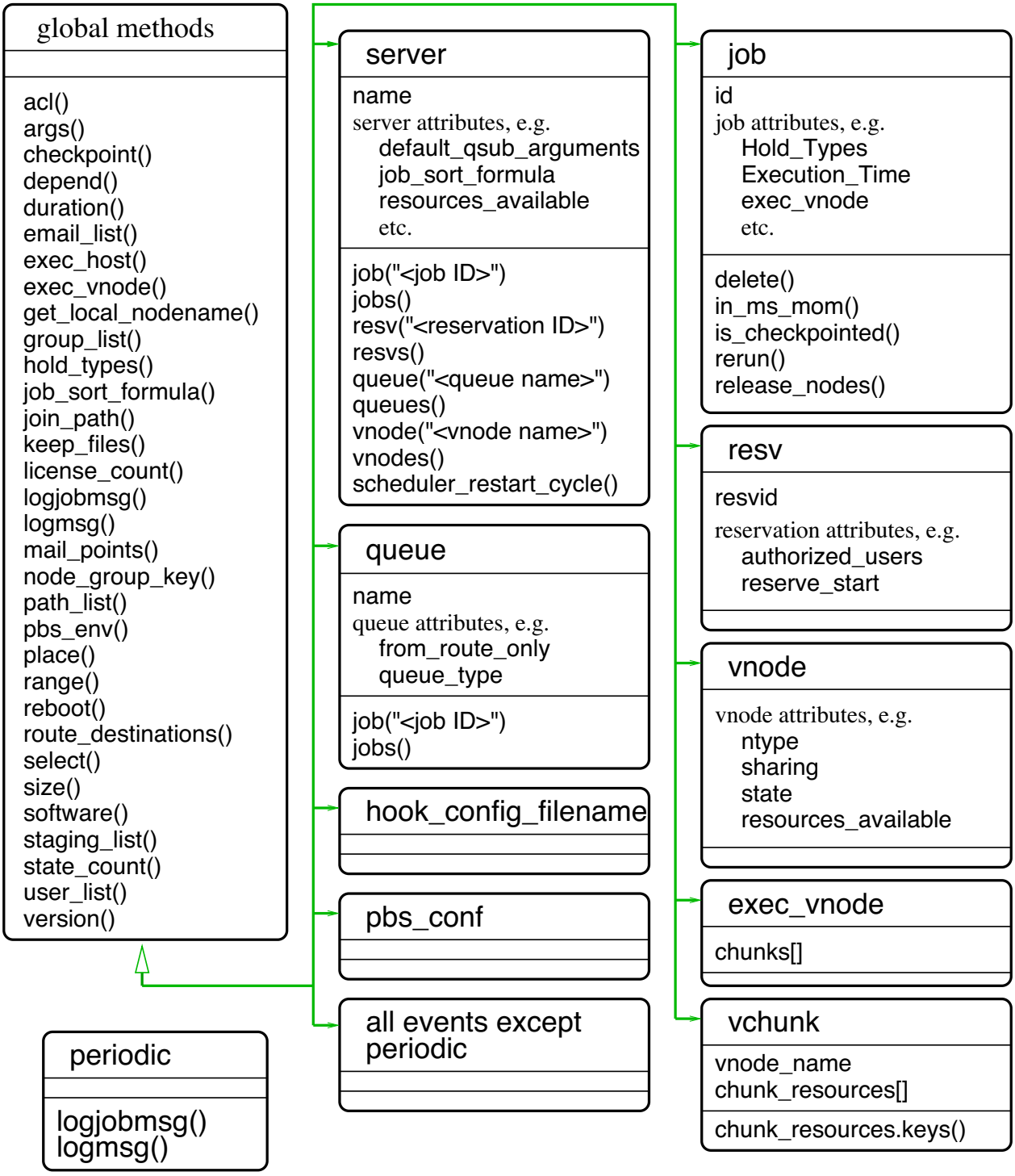


Figure 6-1:Map of members and methods for major PBS objects

Figure 6-2 shows an expanded view of hook event objects. All hook events have the members and methods listed in Figure 6-1, which shows events inheriting global methods. Each type of event also has its own members and/or methods. For example, movejob events have a job member and a src_queue member, in addition to the type, hook_name, requestor, requestor_host, and hook_type members, and the accept(), get_local_nodename(), logjobmsg(), logmsg(), and reject() methods shared by all events. For a description of event objects, see section 6.3, "Event Objects", on page 87.

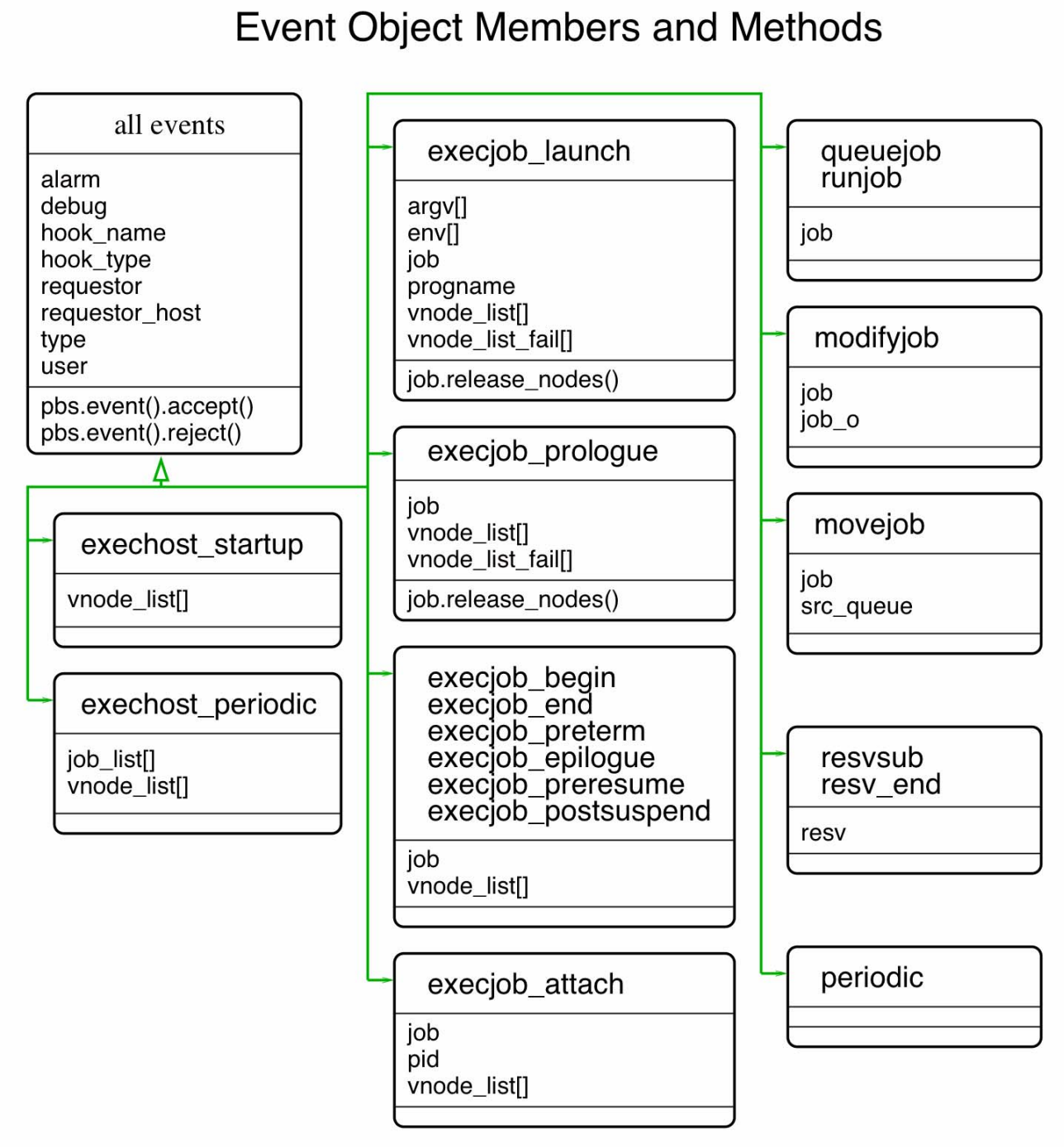


Figure 6-2:Expanded view of event object members and methods

6.3 Event Objects

pbs.event

The event object represents the event that has triggered the hook. You can pass the object to the hook script, and use it in the script. To retrieve objects associated with the event, use this:

pbs.event().<object>

For example, to retrieve the job that triggered an event:

pbs.event().job

There are several types of events. Each type of event is triggered by a different occurrence, and each type has a corresponding hook type. Each type of event has access to different data, and can perform different operations. Some data and operations are common to all events.

Each type of event hook can read and set different job, vnode, and reservation attributes and resources. Each type of event can read different server and queue attributes and resources. We list which attributes and resources can be set for each event in section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.3.1 Event Types

pbs.event().type

The type of the event. Represents the type attribute of the hook. This object can take one or more of the values shown here. The following table summarizes the event types, their constant objects, their triggers, and when and where they run, and gives a pointer to a complete description of the associated hook:

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
resvsub pbs.RESVSUB	Triggered by `pbs_rsub` and the `pbs_submitresv()` API call. A resvsub hook is executed after all processing of `pbs_rsub` input, and just before a reservation is created.	At server	See section 6.3.1.1, "resvsub: Event when Reservation is Created", on page 90.
resv_end pbs.RESV_END	Triggered by end of reservation. A resv_end hook is executed when a reservation ends, just before jobs are deleted from the reservation queue.	At server	See section 6.3.1.2, "resv_end: Event when Reservation Ends", on page 90.
queuejob pbs.QUEUEJOB	Triggered by `qsub` and the `pbs_submit()` API call. Not triggered by requeueing a job (`qrerun`) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down. A queuejob hook is executed after all processing of `qsub` input, and just before the job is queued.	At server	See section 6.3.1.3, "queuejob: Event when Job is Queued", on page 91.

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
modifyjob pbs.MODIFYJOB	Triggered by `qalter`, the `pbs_alterjob()` API call, calculating eligible time, and setting the job's comment. A modifyjob hook is executed after all processing of `qalter` input, and just before the job's attributes are modified. Not triggered when the scheduler modifies a job.	At server	See section 6.3.1.4, "modifyjob: Event when Job is Altered", on page 92.
movejob pbs.MOVEJOB	Triggered by `qmove` and the `pbs_movejob()` API call. Not triggered by `pbs_rsub -Wqmove=<job ID>.` A movejob hook is executed after `qmove` arguments are processed, but before a job is moved from one queue to another.	At server	See section 6.3.1.5, "movejob: Event when Job is Moved", on page 92.
runjob pbs.RUNJOB	Triggered by `qrun` and the `pbs_runjob()` API call. A runjob hook is executed just before a job is sent to an execution host.	At server	See section 6.3.1.6, "runjob: Event Before Job is Received by MoM", on page 94.
periodic pbs.PERIODIC	A periodic hook is executed at specified intervals.	At server	See section 6.3.1.7, "periodic: Periodic Event at Server Host", on page 95
execjob_begin pbs.EXECJOB_BEGIN	An execjob_begin hook is executed when MoM receives the job, after any files or directories are staged in.	On primary MoM host, and if successful, on all sister MoM hosts allocated to job	See section 6.3.1.8, "execjob_begin: Event when Execution Host Receives Job", on page 96.
execjob_prologue pbs.EXECJOB_PROLOGUE	An execjob_prologue hook is executed just before the first job process is started.	On primary MoM host, and on all sister MoM hosts where any job task is spawned or attached	See section 6.3.1.9, "execjob_prologue: Event Just Before Execution of Top-level Job Process", on page 97.
execjob_launch pbs.EXECJOB_LAUNCH	An execjob_launch hook is executed just before the user's program is run.	On primary MoM host, and on all sister MoM hosts where MPI tasks are started with `tm_spawn()`	See section 6.3.1.10, "execjob_launch: Event when Execution Host Receives Job", on page 98

Table 6-2: Event Types and Objects

Event Type & Constant Object	Trigger	Where Run	Description
execjob_attach pbs.EXECJOB_ATTACH	An execjob_attach hook is executed before any execjob_prologue hooks run	On each MoM host where `pbs_attach()` runs	See section 6.3.1.11, "execjob_attach: Event when pbs_attach() runs", on page 100.
execjob_postsuspend pbs.EXECJOB_PRETERM	An execjob_postsuspend hook is executed just after successfully suspending the job	On all MoM hosts allocated to the job	See section 6.3.1.12, "execjob_postsuspend: Event Just After Suspending Job", on page 101.
execjob_preresume pbs.EXECJOB_PRETERM	An execjob_preresume hook is executed just before resuming the job	First on the primary MoM host, and if that is successful, on the sister MoM hosts	See section 6.3.1.13, "execjob_preresume: Event Just Before Resuming Job", on page 102.
execjob_preterm pbs.EXECJOB_PRETERM	An execjob_preterm hook is executed when the job receives a termination signal.	On all MoM hosts allocated to the job	See section 6.3.1.14, "execjob_preterm: Event Just Before Killing Job Tasks", on page 103.
execjob_epilogue pbs.EXECJOB_EPILOGUE	An execjob_epilogue hook is executed after all of the job processes have terminated, after executing or killing a job, but before job is cleaned up	On all MoM hosts allocated to the job	See section 6.3.1.15, "execjob_epilogue: Event Just After Killing Job Tasks", on page 104.
execjob_end pbs.EXECJOB_END	An execjob_end hook is executed on all hosts allocated to a job, at the end of all job processing	On all MoM hosts allocated to the job	See section 6.3.1.16, "execjob_end: Event After Job Cleanup", on page 105.
exechost_startup pbs.EXECHOST_STARTUP	An exechost_startup hook is executed when a MoM starts up or receives a HUP (Linux).	On all MoM hosts in the complex.	See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.
exechost_periodic pbs.EXECHOST_PERIODIC	An exechost_periodic hook is executed at specified intervals	On all MoM hosts in the complex	See section 6.3.1.18, "exechost_periodic: Periodic Events on All Execution Hosts", on page 107.

6.3.1.1 resvsub: Event when Reservation is Created

6.3.1.1.i Modifying Reservation Creation (pbs_rsub)

- When an advance, standing, or job-specific reservation is created via pbs_rsub, resvsub hooks can modify the reservation's attributes that can be set via pbs_rsub
- When an advance, standing, or job-specific reservation is created, resvsub hooks can specify additional attributes that can be specified via pbs_rsub
- The input reservation attributes on which resvsub hooks operate are those that exist after all pbs_rsub processing of command line arguments is completed
- For resvsub hooks, the input reservation attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes & -lncpus) to new select and place syntax

The only time that a reservation can be modified is during its creation. A resvsub event hook can set any settable reservation attribute and any resource that can be specified via pbs_rsub. See Table 5-8, “Reservation Attributes Readable & Settable in resvsub and resv_end Hooks,” on page 59 for a complete list of the reservation attributes that this hook can read and set.

6.3.1.1.ii The resvsub Hook Interface

The type for this event is pbs.RESVSUB.

A resvsub hook is executed after all processing of pbs_rsub input, and just before a reservation is created. The hook is triggered by pbs_rsub and the pbs_submitresv() API call.

A reservation object's attributes appear to a resvsub hook as they would be after the event, not before it.

A pbs.RESVSUB event has the following member, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().resv

A pbs.resv object containing the attributes and resources specified for the reservation being requested. See section 6.9, “Reservation Objects”, on page 131.

A pbs.event().accept() terminates hook execution and allows creation of the reservation, and any changes to reservation resources take effect.

A pbs.event().reject() terminates hook execution and causes the reservation not to be created.

6.3.1.2 resv_end: Event when Reservation Ends

A resv_end event hook can read server and reservation attributes. See Table 5-8, “Reservation Attributes Readable & Settable in resvsub and resv_end Hooks,” on page 59 for a complete list of the reservation attributes that this hook can read.

6.3.1.2.i The resv_end Hook Interface

The type for this event is pbs.RESV_END.

A resv_end hook is executed when a confirmed reservation ends, and just before jobs are deleted from the reservation queue.

A pbs.RESV_END event has the following member, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().resv

A pbs.resv object containing the attributes and resources specified for the reservation being requested. See section 6.9, “Reservation Objects”, on page 131.

A pbs.event().reject() does not interrupt the execution of the process invoking it.

6.3.1.3 queuejob: Event when Job is Queued

6.3.1.3.i Modifying Job Submission (qsub)

- When a job is submitted via qsub, queuejob hooks can modify the following things explicitly specified in the job submission:
  - Job attributes that can be set via qsub
  - Job comment
  - Resources requested by the job
- When a job is submitted via qsub, queuejob hooks can add resource requests to those specified in the job submission
- The input job attributes on which queuejob hooks operate are those that exist after all qsub processing is completed. These input attributes include:
  - Command line arguments
  - Script directives
  - Server default_qsub_arguments
- When a queuejob hook runs at job submission, the hook can affect only that job.
- For queuejob hooks, the input job attributes do not include:
  - Server or queue resources_default or default_chunk.
  - Conversions from old syntax (-lnodes or -lncpus) to new select and place syntax

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44, for a complete listing of attributes and resources that this hook can modify.

6.3.1.3.ii The queuejob Hook Interface

The event type for this event is pbs.QUEUEJOB.

A queuejob hook runs after all processing of qsub input, just before the job reaches the server, and before the job is queued, including when a job is peer queued to a server that has a queuejob hook. (See Figure 4-3.) The hook is triggered by qsub or the pbs_submit() API call. A queuejob hook is not triggered by requeueing a job (qrerun) or on node_fail_requeue, when a job is discarded by the MoM because the execution host went down. A queuejob hook runs once per job array.

In a queuejob event, the event's job object members are as they would be if the job were to be successfully submitted.

A pbs.QUEUEJOB event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object with the attributes and resources specified at submission for the job being queued. See section 6.6, "Job Objects", on page 122.

A pbs.event().accept() terminates hook execution and allows the job to be queued, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be queued. The job is not accepted by the server, and is not assigned a job ID.

6.3.1.3.iii Caveats for queuejob Hook

If a user submits a job using old -lnodes or -lncpus syntax, this is translated to a select statement, but only after a queuejob hook has run. The queuejob hook does have access to the job's resource request.

6.3.1.4 modifyjob: Event when Job is Altered

6.3.1.4.i Modifying Job Change (qalter)

- When a job is changed via qalter, modifyjob hooks can modify the arguments passed to qalter
- When a modifyjob hook runs, it can change the attributes of the job that can be changed via qalter

Before the job runs, this hook can set any job attribute that can be changed via qalter, can set the job's comment, and can set any resource requested by the job.

While the job is running, the only job attributes and resources that the hook can set are those that can be changed via the qalter command: the job's cput and walltime. See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44, for a complete listing of attributes and resources that this hook can modify.

See "qalter" on page 127 of the PBS Professional Reference Guide and "Job Attributes" on page 328 of the PBS Professional Reference Guide.

6.3.1.4.ii The modifyjob Hook Interface

The type for this event is pbs.MODIFYJOB.

A modifyjob hook is executed after all processing of qalter input, and just before the job's attributes are modified. The hook is triggered by the following:

- A qalter command, except when the scheduler calls the command
- The pbs_alterjob() API call
- Calculating eligible time
- Setting the job's comment

A modifyjob hook runs once per job array.

In a modifyjob event hook, the pbs.event().job object's attributes appear to a modifyjob hook as they would be after the job is modified, not before.

A modifyjob event hook shows the original job with all its attributes in pbs.event().job_o.

A pbs.MODIFYJOB event has the following members, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object representing the job being modified. See section 6.6, "Job Objects", on page 122. In this job object, only attributes and resources that are to be modified by the qalter command are populated. In this job object, attributes or resources that are not slated to be modified are not populated. This job object's attributes appear to a modifyjob hook as they would be after the job is modified, not before.

pbs.event().job_o

A pbs.job object representing the original job, before the job was modified via qalter. All attributes and resources are populated. See section 6.3.2.12, "Original Job Event Member", on page 112.

A pbs.event().accept() terminates hook execution and allows the job to be altered, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be altered.

6.3.1.5 movejob: Event when Job is Moved

6.3.1.5.i Modifying Job Move (qmove)

- When a job is moved via qmove, movejob hooks can modify the arguments passed to qmove
- When a movejob hook runs, it can change the job's destination queue to any queue on the default server

A movejob hook can specify only local queues as the destination queue. Whether a job is submitted with a local queue or a remote queue as its destination, a movejob hook can change the destination to a local queue.

The only job attribute that a movejob event hook can set is the job's destination queue.

6.3.1.5.ii The movejob Hook Interface

The type for this event is pbs.MOVEJOB.

The server runs its movejob hooks when any of the following happens:

- This server is the furnishing server when peer scheduling a job
- A job is moved from this server to another server via the `qmove` command
- A job is moved between two queues on this server

A movejob hook is executed after `qmove` arguments are processed, but before a job is moved from one queue to another. This hook is triggered by `qmove` and the `pbs_movejob()` API call. movejob hooks are not triggered by `pbs_rsub -Wqmove=<job ID>`. A movejob hook runs once per job array.

A job object's attributes appear to a movejob hook as they would be after the event, not before it.

The hook shows the job's originating queue in the pbs.event().src_queue object member.

A pbs.MOVEJOB event has the following members, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

A pbs.job object representing the job being moved. See section 6.6, “Job Objects”, on page 122.

Note that pbs.event().job.queue refers to the destination queue, not the current queue.

pbs.event().src_queue

The pbs.queue object representing the originating queue where pbs.event().job came from.

A pbs.event().accept() terminates hook execution and allows the job to be moved, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job not to be moved.

6.3.1.6 runjob: Event Before Job is Received by MoM

6.3.1.6.i Changes Before Job is Sent to MoM (qrun)

When the scheduler runs a job or the administrator runs a job using the qrun command, any runjob hooks are executed.

- On accepting a job, a runjob hook can modify the following:
  - The job's Error_Path attribute
  - The job's Output_Path attribute
  - All of the job's Variable_List attribute members
  - The following Resource_List attribute members:

    cput

    exec_vnode

    file

    max_walltime

    min_walltime

    nice

    pcput

    pmem

    pvmem

    site

    software

    start_time

    walltime
- When a runjob hook rejects a job, it can do the following:
  - Set the job's depend attribute
  - Set any members of the job's Variable_List attribute
  - Place a hold on the job
  - Release a hold on the job
  - Set the job's project attribute
  - Change the time the job is allowed to begin execution
  - Set any of the job's Resource_List attribute members except nodect
  - Change the state of a vnode where the job would have run

See Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55 and Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

A runjob hook can modify a vnode only if the hook rejects the event, and the vnode is in the job's exec_vnode attribute. For a vnode, the hook can modify only the state attribute. The only pre-execution event hook that can change this attribute is a runjob hook.

6.3.1.6.ii The runjob Hook Interface

The event type is pbs.RUNJOB.

A runjob event occurs when one of the following happens:

- The administrator uses the qrun command
- The scheduler chooses to run a job and calls pbs_runjob()

A runjob hook is executed just before a job is sent to the execution host. It is triggered by `qrun` and the `pbs_runjob()` API call. A runjob hook runs once per subjob.

For a runjob hook only, job object attributes appear as they would be before the event takes place.

A pbs.RUNJOB event has the following member, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

A pbs.job object representing the job being run. See section 6.6, "Job Objects", on page 122.

A pbs.event().accept() terminates hook execution and allows the job to be sent to the execution host, and any changes to job attributes or resources take effect.

A pbs.event().reject() terminates hook execution and causes the job to be requeued instead of being sent to the execution host. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

6.3.1.7 periodic: Periodic Event at Server Host

6.3.1.7.i Periodic Events at Server Host

Periodically, at the server host, a periodic hook can:

- Run `qstat`, job start time estimator named `pbs_est`, etc.

6.3.1.7.ii The periodic Hook Interface

This event type is pbs.PERIODIC.

The periodic hook runs periodically on the server host, in the background. The hook begins periodic execution, and the interval timer is restarted, when any of the following happens:

- The hook is enabled
- The hook is imported
- The server starts

The periodic hook runs as *pbsadmin*.

The interval between calls to periodic hooks is specified in the freq hook attribute. See section 5.1.13, "Setting Hook Interval (Frequency)", on page 39.

A call to pbs.event().accept() causes any changes made to objects exposed in the hook to take effect.

A call to pbs.event().reject(<message>) prevents any changes from taking effect.

A call to pbs.event().reject(<message>) causes the following messages to appear in the server log:

```
"run_periodic_hook; request rejected by <hook_name>"
<message>
```

The periodic hook continues to be periodically called whether or not there are errors in hook script execution or a call to the pbs.event().reject() action. To stop the hook from being called, either disable it or delete it:

```
#qmgr -c "s h <periodic hook> enabled=f"
#qmgr -c "d h <periodic hook>"
```

If the periodic hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, all changes do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"periodic hook <hook_name> encountered an exception, request rejected"
"alarm call while running periodic hook '<hook_name>', request rejected"
```

6.3.1.7.iii Caveats for periodic Event Hooks

The order attribute is ignored for periodic hooks. It does **not** guarantee the execution order of a list of periodic hooks.

6.3.1.8 execjob_begin: Event when Execution Host Receives Job

6.3.1.8.i Changes When Job is Received by MoM

When MoM receives a job, an execjob_begin hook can:

- Modify the job's Execution_Time, Hold_Types, Variable_List, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes

6.3.1.8.ii The execjob_begin Hook Interface

This event type is pbs.EXECJOB_BEGIN.

An execjob_begin hook executes on the primary MoM host and then, if successful, executes on all the sister MoM hosts allocated to the job. The hook executes when the host first receives the job, after any files or directories are staged in.

A pbs.EXECJOB_BEGIN event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, "Job Objects", on page 122.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() means the job can proceed with execution, and any changes to job attributes, resources, or the vnode list take effect.

A call to pbs.event().reject(<message>) automatically causes the job to be killed and tells the server to requeue the job. In addition, any changes to job attributes, resources, or vnode list take effect. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

- If the pbs.event().reject(<message>) call is made on a primary execution host, the following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_begin request rejected by <hook_name>"

  <message>

  The rejection message <message> also appears in the STDERR of the program such as qrun invoking pbs_runjob() API:
- If the pbs.event().reject(<message>) call is made on a sister host, the following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_begin request rejected by <hook_name>"

  <message>

  In addition, this message appears in mom_logs on the primary execution host:

  "job_start_error: <hook errno> from node <hostname> could not JOIN_JOB successfully.
- If pbs_runjob() was invoked by the scheduler, the following job comment appears:

  "Not running: PBS Error: <message>"

If the execjob_begin hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is automatically killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts do **not** take effect. In this case, one of the the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"execjob_begin hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_begin hook '<hook_name>', request rejected"
```

6.3.1.9 execjob_prologue: Event Just Before Execution of Top-level Job Process

6.3.1.9.i Changes Before Job Shell is Executed

Just before a job's top shell is executed, an execjob_prologue hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Modify a job's vnode request
- Put a bad vnode in the pbs.event().vnode_list_fail[] list

6.3.1.9.ii The execjob_prologue Hook Interface

This event type is pbs.EXECJOB_PROLOGUE.

An execjob_prologue hook runs on the primary MoM host. If the hook runs successfully on the primary MoM host, an execjob_prologue hook runs on each of the sister MoM hosts allocated to the job. On the primary MoM host, an execjob_prologue hook executes just prior to executing the top-level shell or cmd process of the job. This is where the prologue executes. On a sister MoM host, the hook executes just before the first task of the job on this host is spawned, and before any execjob_launch or execjob_attach hooks. See Table 4-1, “Execution Event Hook Timing,” on page 19.

An execjob_prologue hook overrides a prologue. If an execjob_prologue hook exists and is enabled, MoM executes the hook. Otherwise, she executes the prologue.

A pbs.EXECJOB_PROLOGUE event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.release_nodes()

This method releases unneeded vnodes from a job's vnode request. See section 6.6.4.4, “Job Object Method to Release Vnodes”, on page 128.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

This is a pbs.job object representing the job that is about to run. See section 6.6, “Job Objects”, on page 122.

pbs.event().vnode_list_fail[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job but are marked as unhealthy. See section 6.3.2.23, “The Failed Vnode List Event Member”, on page 115 for information about pbs.event().vnode_list_fail[].

A pbs.event().accept() allows the job to continue its normal execution, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject(<message>) causes the job to be killed, and the owning server to requeue the job. Any changes to job attributes, resources, or vnode list take effect. When a job is requeued by this hook, the scheduler considers it for execution in the next scheduling cycle.

- On the primary execution host, the following job-level mom_logs entries appear:
  "execjob_prologue request rejected by <hook_name>"
  <message>
- On a sister vnode, the following job-level mom_logs entries appear:
  "execjob_prologue request rejected by <hook_name>"
  <message>
- In addition, the following message appears in the STDERR of the program invoking the tm_attach() API, such as the pbs_attach() command:
  "a hook has rejected the task manager request"

If the following setting is specified in the hook script, just before issuing a pbs.event().reject(), the job is deleted instead of being requeued:

```
pbs.event().job.delete()
```

If the user attribute of the execjob_prologue hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_prologue hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job is killed and the server requeues the job. All job changes, vnode changes, or requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

"execjob_prologue hook <hook_name> encountered an exception, request rejected"

"alarm call while running execjob_prologue hook '<hook_name>', request rejected"

The standard output and standard error of an execjob_prologue hook script are not connected to the standard output and standard error of the job.

6.3.1.10 execjob_launch: Event when Execution Host Receives Job

6.3.1.10.i Changes Before User Program is Executed

Just before the user's program is executed, an execjob_launch hook can:

- Change the job's top shell or executable
- Change the arguments to the shell or executable
- Change the job's environment variables
- Modify job and vnode attributes
- Modify a job's vnode request
- Put a bad vnode in the pbs.event().vnode_list_fail[] list

An execjob_launch hook cannot modify anything else.

6.3.1.10.ii The execjob_launch Hook Interface

This event type is pbs.EXECJOB_LAUNCH.

An execjob_launch hook runs on the primary MoM host just before executing the user's program. The hook runs on the sister MoM hosts allocated to the job, just before executing the user's program as specified in a tm_spawn() API call, which is called from pbsdsh and pbs_tmrsh.

Any execjob_launch hooks runs after execjob_prologue hooks.

This hook cannot use any of the job's methods.

A pbs.EXECJOB_LAUNCH event hook has access to the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().argv[]

This is a pbs.argv[] object representing the arguments to the shell or executable. See section 6.3.2.2, "Job Program Arguments Event Member", on page 109.

pbs.event().env

This is a pbs.env[] object representing the job's environment variables. See section 6.3.2.5, "Job Environment Event Member", on page 110.

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, "Job Objects", on page 122.

pbs.event().job.release_nodes()

This method releases unneeded vnodes from a job's vnode request. See section 6.6.4.4, "Job Object Method to Release Vnodes", on page 128.

pbs.event().progname

This is a pbs.progname object representing the job shell or executable. See section 6.3.2.15, "Job Executable Event Member", on page 112.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to the job that caused the execjob_launch hook to execute. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about pbs.event().vnode_list[]. This object is read-only for this event. The vnode objects in vnode_list[] cannot be modified. Attempting to modify them result in the following:

- The execjob_launch hook is terminated
- The job ends prematurely with a non-zero Exit_Status value
- The following PBSEVENT_DEBUG2 message appears in mom_logs:

```
"execjob_launch hook 'launch' encountered an exception,  request rejected"
Can only set progname, argv, env event parameters under execjob_launch hook."
```

pbs.event().vnode_list_fail[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job but are marked as unhealthy. Seesection 6.3.2.23, "The Failed Vnode List Event Member", on page 115 for information about pbs.event().vnode_list_fail[].

A call to pbs.event().accept() means the job can proceed with execution, and any changes to progname, argv[], and env[] take effect. If the hook makes changes to the job's progname, argv[], or env[] parameters, the appropriate PBSEVENT_DEBUG2 message(s) appear in mom_logs for each change in a value:

```
"progname orig: <original_progname>"
"progname new: <updated_progname>"
"argv orig: <original_argv>"
"argv new: <updated_argv>"
"env orig: <original_env>"
"env new: <updated_env>"
```

A call to pbs.event().reject(<message>) causes the job to be terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages to appear in mom_logs:

```
"execjob_launch" request rejected by '<hook_name>'"
<message>
```

If the execjob_launch hook script encounters an unexpected error causing an unhandled exception, the job is terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages appear in mom_logs:

```
"execjob_launch hook <hook_name> encountered an exception, request rejected"
```

If the execjob_launch hook script terminates due to a hook alarm, the job is terminated with a non-zero Exit_Status value, and the following PBSEVENT_DEBUG2 messages appear in mom_logs:

```
"alarm call while running execjob_launch hook '<hook_name>', request rejected"
```

6.3.1.11 execjob_attach: Event when pbs_attach() runs

6.3.1.11.i Event when pbs_attach() Runs

When pbs_attach() is called, an execjob_attach hook can accept or reject the procedure where the process ID is attached to the job.

6.3.1.11.ii The execjob_attach Hook Interface

An execjob_attach hook runs on any MoM host where an MPI process is spawned using pbs_attach(). The execjob_attach hook runs for each process ID.

The execjob_attach hook runs before any execjob_prologue hooks run on behalf of the first task. See Table 4-1, "Execution Event Hook Timing," on page 19.

An execjob_attach hook cannot modify any PBS objects.

A pbs.EXECJOB_ATTACH event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run. See section 6.6, "Job Objects", on page 122. For this hook, this job object is read-only.

pbs.event().pid

This is a Python int representing the process ID whose session ID is being added to the job tasks list. This hook cannot modify the value of the process ID.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), MoM proceeds as usual to add the session ID of the process ID to the job's task list.

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- MoM does not get the session ID
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:

  ```
  "execjob_attach" request rejected by '<hook_name>'"
  <message>
  ```

If the execjob_attach hook script encounters an unhandled exception:

- Hook execution terminates
- MoM does not get the session ID of the process ID
- The following message appears in mom_logs at PBSEVENT_DEBUG2:

  "execjob_attach hook <hook_name> encountered an exception, request rejected"

If the execjob_attach hook script terminates due to a hook alarm, MoM does not get the session ID of the process ID, and the following message appears in mom_logs at PBSEVENT_DEBUG2:

"alarm call while running execjob_attach hook '<hook_name>', request rejected"

6.3.1.11.iii Caveats for execjob_attach Hooks

- Do not attempt to modify pbs.event().pid. If you do:
  - Hook execution is terminated
  - MoM does not get the session ID of the process ID
  - The following messages appear in mom_logs at PBSEVENT_DEBUG2:

    "execjob_attach hook <hook_name> encountered an exception, request rejected"

    "event attribute 'pid' is read-only"
- Do not attempt to modify pbs.event().job or the objects in pbs.event().vnode_list[]. If you do:
  - Hook execution is terminated
  - MoM does not get the session ID of the process ID
  - The following messages appear in mom_logs at PBSEVENT_DEBUG2:

    "execjob_attach hook <hook_name> encountered an exception, request rejected"

    "nothing is settable inside an execjob_attach hook!"

6.3.1.12 execjob_postsuspend: Event Just After Suspending Job

6.3.1.12.i The execjob_postsuspend Hook Interface

This event type is pbs.EXECJOB_POSTSUSPEND.

This hook runs on all of the MoM hosts assigned to a job, after the job has been successfully suspended.

An execjob_postsuspend hook:

- Cannot modify any PBS objects
- Cannot be used to set a fail action
- Must run as *pbsadmin*
- Does not interrupt the flow of suspend/resume

A pbs.EXECJOB_POSTSUSPEND event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that has just been suspended. See section 6.6, “Job Objects”, on page 122. For this hook, this job object is read-only.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), nothing happens.

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:
  "execjob_postsuspend" request rejected by '<hook_name>'"
  <message>

6.3.1.13 execjob_preresume: Event Just Before Resuming Job

6.3.1.13.i The execjob_preresume Hook Interface

This event type is pbs.EXECJOB_PRERESUME.

This hook runs on the primary MoM host when this MoM receives a request to resume a job, and then if this is successful, the primary MoM sends a request to the sisters to resume the job, at which point this hook runs on the sister MoM hosts. All of the execjob_preresume hooks for a job must succeed in order for the job to resume.

An execjob_preresume hook:

- Cannot modify any PBS objects
- Cannot be used to set a fail action
- Must run as *pbsadmin*

A pbs.EXECJOB_PRERESUME event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that has just been suspended. See section 6.6, “Job Objects”, on page 122. For this hook, this job object is read-only.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. The list of vnodes is read-only for this event. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept(), nothing happens

On a call to pbs.event().reject(<message>), the following happens:

- Hook execution terminates
- All MoMs where job processes were running are prevented from resuming the job
- PBS prints the following message in mom_logs at log level PBSEVENT_DEBUG2:
  "execjob_preresume" request rejected by '<hook_name>'"
  <message>

6.3.1.14 execjob_preterm: Event Just Before Killing Job Tasks

6.3.1.14.i Changes Before Job is Killed

Just before a job is killed, an execjob_preterm hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Cause the job to keep running

6.3.1.14.ii The execjob_preterm Hook Interface

This event type is pbs.EXECJOB_PRETERM.

An execjob_preterm hook executes on all the MoM hosts allocated to a job. This hook runs only when a `qdel` has been issued. It does not run for any other job termination. For example, it does not run on a `qrerun` or when a job goes over its limit. On the primary MoM host, the hook executes when the job receives a signal from the server for the job to terminate. On a sister MoM host, this hook executes when the sister receives a request from the primary MoM host to terminate the job, just before the sister signals the task on this host to terminate.

A pbs.EXECJOB_PRETERM event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().job

This is a pbs.job object representing the job that is about to run (or be killed). See section 6.6, "Job Objects", on page 122.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A pbs.event().accept() call allows job cancellation or deletion to happen, and any changes to job attributes, resources, or vnode list take effect.

A pbs.event().reject() call causes the job instance on a vnode to continue running, because the terminate signal is not delivered to the job. Any changes to job attributes, resources, or vnode list take effect.

- On the primary execution host, a pbs.event().reject(<message>) causes the following to appear in the `STDERR` of the program (`qdel`) invoking the `pbs_deljob()` API:

  `"hook rejected request"`
- The following message appears in the MoM log at log event class PBSEVENT_DEBUG2:

  `"execjob_preterm request rejected by <hook_name>"`

  `<message>`
- On a sister host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  `"execjob_preterm request rejected by <hook_name>"`

  `<message>`

If the user attribute of the execjob_preterm hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_preterm hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job continues to run, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"execjob_preterm hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_preterm hook '<hook_name>', request rejected"
```

6.3.1.15 execjob_epilogue: Event Just After Killing Job Tasks

6.3.1.15.i Changes After Job is Executed

Just after a job is executed, an execjob_epilogue hook can:

- Modify the job's Execution_Time, Hold_Types, and resources_used attributes
- Flag the job to be rerun
- Kill the job
- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job's exit status

6.3.1.15.ii The execjob_epilogue Hook Interface

This event type is pbs.EXECJOB_EPILOGUE.

An execjob_epilogue hook executes on all the MoM hosts allocated to the job. On a primary MoM host, the hook executes after all the job tasks/processes on the host have been killed, and basic CPU and memory resource usage information have been logged, but before job processes are cleaned up. This is where the epilogue executes. On a sister MoM host, the hook executes after the sister MoM receives a request to kill the job and has signaled the job tasks to terminate.

When an execjob_epilogue hook modifies the resources_used job attribute, it is modifying only the value counted at the local host. For example, if a job runs on two hosts, using four minutes of CPU time on each host, and the hook changes that to three minutes (and this hook runs at both hosts), the job's final CPU time total is six minutes instead of eight.

An execjob_epilogue hook overrides an epilogue. If an execjob_epilogue hook exists and is enabled, MoM executes the hook. Otherwise, she executes the epilogue.

A pbs.EXECJOB_EPILOGUE event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that just finished. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.Exit_status

A Python int that holds the exit value of the top level shell of the job script. This value is valid only if the hook is executing on a primary execution host.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, “The Vnode List Event Member”, on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() continues the normal end-of-job processing, and any changes to job attributes, resources, or vnode list take effect.

A call to pbs.event().reject() causes the job on the current vnode to exit, and the owning server to completely delete the job. Any changes to job attributes, resources, or vnode list take effect.

- On a primary execution host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_epilogue request rejected by <hook_name>"

  <message>
- On a sister host, a pbs.event().reject(<message>) causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

  "execjob_epilogue request rejected by <hook_name>"

  <message>
- If the following call has been made prior to calling pbs.event().reject(), the owning server requeues the job:

  pbs.event().job.rerun()

If the user attribute of the execjob_epilogue hook is set to *pbsuser*, the hook script executes under the context of the job owner (the value of the euser job attribute).

If the execjob_epilogue hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, this causes the job on the current vnode to exit, and the owning server to completely delete the job. All job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

"execjob_epilogue hook <hook_name> encountered an exception, request rejected"

"alarm call while running execjob_epilogue hook '<hook_name>', request rejected"

The standard output and standard error of an execjob_epilogue hook script are not connected to the standard output and standard error of the job.

6.3.1.16 execjob_end: Event After Job Cleanup

6.3.1.16.i Changes After Job Finishes or is Killed

Just after a job is cleaned up after it finishes execution or is killed, an execjob_end hook can:

- Set attributes and resources on the vnode(s) managed by the MoM where this job executes
- Use the job's exit status

An execjob_end hook cannot effectively modify the job's Execution_Time and Hold_Types attributes. These changes will not be visible to the server, because the job is already cleaned up and reported.

6.3.1.16.ii The execjob_end Hook Interface

This event type is pbs.EXECJOB_END.

An execjob_end hook executes on all the MoM hosts allocated to a job. The hook is executed after a job is cleaned up.

A pbs.EXECJOB_END event has the following members and methods, in addition to those listed in Table 6-3, “Using Event Object Members in Events,” on page 108 and Table 6-14, “Methods Available in Events,” on page 142:

pbs.event().job

This is a pbs.job object representing the job that just ran. See section 6.6, “Job Objects”, on page 122.

pbs.event().job.Exit_status

A Python int that holds the exit value of the top level shell of the job script. This value is valid only when the hook executes on a primary execution host.

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are assigned to this job. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

A call to pbs.event().accept() ends the job, and any changes to job attributes, resources, or vnode list take effect.

A call to pbs.event().reject(<message>) also ends the job, and any changes to job attributes, resources, or vnode list also take effect.

A call to pbs.event().reject(<message>) on a primary execution host causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

```
"execjob_end request rejected by <hook_name>"
<message>
```

A call to pbs.event().reject(<message>) on a sister host causes the following message to appear in the MoM log at log event class PBSEVENT_DEBUG2:

```
"execjob_end request rejected by <hook_name>"
<message>
```

If the execjob_end hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, the job terminates, and all job changes, vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appear in the MoM logs at event class PBSEVENT_DEBUG2:

```
"execjob_end hook <hook_name> encountered an exception, request rejected"
"alarm call while running execjob_end hook '<hook_name>', request rejected"
```

6.3.1.17 exechost_startup: Event When Execution Host Starts Up

6.3.1.17.i Event when Execution Host Starts or Receives HUP

When an execution host starts up or receives a HUP, an exechost_startup hook can:

- Create vnodes on local host
- Create custom resources for vnodes
- Offline vnodes that are not ready for use
- Return vnodes to use that have been previously offlined
- Modify the attributes and resources of the vnodes managed by the local MoM

6.3.1.17.ii The exechost_startup Hook Interface

This event type is pbs.EXECHOST_STARTUP.

The exechost_startup hook runs on a MoM host every time its MoM starts up, or when a Linux `pbs_mom` receives a SIGHUP signal. This hook executes after MoM loads `pbs.conf` values, reads `mom_priv/config` values, and runs platform-specific initializations, for example cpuset initialization, including topology data gathering. If there are Version 2 configuration files, this hook sets vnode definitions from those Version 2 configuration files.

The exechost_startup hook runs independently of jobs; it depends only on MoM startup and HUP.

A pbs.EXECHOST_STARTUP event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are managed by the MoM where the hook runs. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

On a call to pbs.event().accept() or pbs.event().reject(), vnode changes take effect, and MoM continues to run.

A call to pbs.event().reject(<message>) causes the following messages to appear in the MoM log:

```
"exechost_startup" request rejected by hook <hook_name>"
<message>
```

If the exechost_startup hook script encounters an unexpected error causing an unhandled exception:

- Vnode changes do not take effect
- MoM continues to run
- The following message appears at PBSEVENT_DEBUG2 in mom_logs:

```
"exechost_startup hook <hook_name> encountered an exception, request rejected"
```

If the exechost_startup hook script terminates due to a hook alarm, vnode changes do not take effect, MoM continues to run, and the following message appears at PBSEVENT_DEBUG2 in mom_logs:

```
"alarm call while running exechost_startup hook '<hook_name>', request rejected'
```

6.3.1.17.iii Advice on Using exechost_startup Hooks

- We recommend that your hook does not make changes unless the hook accepts its event. You do not want to have to back changes out upon a reject().
- For exceptions, we recommend that you catch them via try... except and accompany them with a call to pbs.event().reject().
- We recommend that before calling pbs.event().reject(), you set the vnodes managed by the local MoM offline with an accompanying comment. This stops jobs from being sent to the affected vnodes. For example:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
    vnlist[v].state = pbs.ND_OFFLINE
    vnlist[v].comment = "bad configuration"
pbs.event().reject("not accepting jobs")
```

6.3.1.18 exechost_periodic: Periodic Events on All Execution Hosts

6.3.1.18.i Periodic Events at Execution Hosts

Periodically, at each execution host, an exechost_periodic hook can:

- Set attributes and resources for any vnode managed by the MoM on the host where the hook runs. This means that an instance of a hook can affect more than one vnode only when the hook is running on a multi-vnode host.
- Set attributes or resources for each job managed by the local MoM.

6.3.1.18.ii The exechost_periodic Hook Interface

This event type is pbs.EXECHOST_PERIODIC.

The exechost_periodic hook runs periodically on all the MoM hosts in the complex.

The interval between calls to exechost_periodic hooks is specified in the freq hook attribute. See section 5.1.13, "Setting Hook Interval (Frequency)", on page 39.

A pbs.EXECHOST_PERIODIC event has the following members and methods, in addition to those listed in Table 6-3, "Using Event Object Members in Events," on page 108 and Table 6-14, "Methods Available in Events," on page 142:

pbs.event().vnode_list[]

This is a dictionary of pbs.vnode objects, keyed by vnode name, listing the vnodes that are managed by the MoM where the hook runs. See section 6.3.2.22, "The Vnode List Event Member", on page 114 for information about using pbs.event().vnode_list[].

pbs.event().job_list[]

List of the pbs.job objects managed by the local MoM. This hook can set the attributes and resources for these jobs. See section 6.3.2.11, "Job List Event Member", on page 111.

A call to pbs.event().accept() or pbs.event().reject(<message>) causes any changes made to vnodes to take effect.

A call to pbs.event().reject(<message>) causes the following messages to appear in the MoM log:

```
"exechost_periodic" request rejected by hook <hook_name>"
<message>
```

The periodic hook continues to be periodically called whether or not there are errors in hook script execution or a call to the pbs.event().reject() action. To stop the hook from being called, either disable it or delete it:

```
#qmgr -c "s h <periodic hook> enabled=f"
#qmgr -c "d h <periodic hook>"
```

If the exechost_periodic hook script encounters an unexpected error causing an unhandled exception, or if the script terminates due to a hook alarm, all vnode changes, requests for host reboot or scheduler cycle restarts, do **not** take effect. In addition, one of the following messages appears in the MoM log at event class PBSEVENT_DEBUG2:

```
"exechost_periodic hook <hook_name> encountered an exception, request rejected"
"alarm call while running exechost_periodic hook '<hook_name>', request rejected"
```

6.3.1.18.iii Caveats for exechost_periodic Event Hooks

The order attribute is ignored for exechost_periodic hooks. It does **not** guarantee the execution order of a list of periodic hooks.

6.3.2 Event Object Members

Some event object members are hook attributes, and some exist as part of the event object but are not hook attributes. The following table summarizes the members for event objects, and shows which event objects have access to each member, and whether the event hook can read and set the member. An "*r*" indicates read, an "*s*" indicates set, and an "*o*" indicates that this member can be set but the action has no effect. See Table 4-1, "Execution Event Hook Timing," on page 19 for more information about why some operations have no effect.

Table 6-3: Using Event Object Members in Events

Event Object Member	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.event().alarm	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().argv[]	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().debug	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().enabled	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().env	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().fail_action	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().freq	---	---	---	---	r, s	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().hook_name	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().hook_type	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Table 6-3: Using Event Object Members in Events

Event Object Member	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.event().job	r, s	r, s	r, s	r, s	r, s	---	---	---	r, s	r	r, s	r	r	r	r, s	r, s	r, s	---	---	---
pbs.event().job_list (jobs)	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	r, s	---
pbs.event().job_o	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().order	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().pid	---	---	---	---	---	---	---	---	---	r	---	---	---	---	---	---	---	---	---	---
pbs.event().progname	---	---	---	---	---	---	---	---	---	---	---	r, s	---	---	---	---	---	---	---	---
pbs.event().requestor	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().requestor_host	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().resv	---	---	---	---	---	---	table 5-8	table 5-8	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().src_queue	---	---	r	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().type	r	r	r	r	r	---	r	r	r	r	r	r	r	r	r	r	r	r	r	r
pbs.event().user	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
pbs.event().vnode_list[]	---	---	---	---	---	---	---	---	r, s	r	r, s	r	r, s	r, s	r, s	r, s	r, s	r, s	r, s	---
pbs.event().vnode_list_fail[]	---	---	---	---	---	---	---	---	---	---	r	r	---	---	---	---	---	---	---	---

An event object (an object returned by pbs.event()) has one or more of the following members:

6.3.2.1 Hook Alarm Event Member

pbs.event().alarm

Hook attribute. Number of seconds to allow a hook to run before the hook times out. Must be greater than zero. See "alarm" on page 350 of the PBS Professional Reference Guide.

Type: *Integer*

6.3.2.2 Job Program Arguments Event Member

pbs.event().argv[]

The list of arguments to be passed to the job script. The arguments can be modified in an execjob_launch hook.

Type: *Python list of strings*

To add another argument to the argument list, append it:

```
pbs.event().argv.append(<new_argument>)
```

To clear out existing argv[] entries and supply a new set of arguments, use the following:

```
pbs.event().argv = []      (sets argv[] to empty list)
pbs.event().argv.append(<arg0>)
pbs.event().argv.append(<arg1>)
…
pbs.event().argv.append(<argN>)
```

On Windows, where backslashes may appear in pathnames, escape each backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS Pro\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS Pro\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

To log the arguments to the program, and update some of them:

```
for a in pbs.event().argv:
    pbs.logmsg(pbs.LOG_DEBUG, "a=%s" % (a,))

argv = pbs.event().argv
argv[1] = "beta"
argv[3] = "gamma"
```

6.3.2.3 Hook Debug Behavior Indicator Event Member

pbs.event().debug

Hook attribute. Specifies whether or not the hook produces debugging files under `PBS_HOME/server_priv/hooks/tmp` or `PBS_HOME/mom_priv/hooks/tmp`. Files are named *hook_<hook event>_<hook name>_<unique ID>.in*, *.data*, and *.out*. See "Producing Files for Debugging" on page 159 in the PBS Professional Hooks Guide, and "debug" on page 350 of the PBS Professional Reference Guide.

Type: *Boolean*

6.3.2.4 Hook Enable or Disable Event Member

pbs.event().enabled

Hook attribute. Specifies whether or not the hook is enabled. See "enabled" on page 350 of the PBS Professional Reference Guide.

Type: *Boolean*

6.3.2.5 Job Environment Event Member

pbs.event().env

The job's environment. Can be modified in an execjob_launch hook.

Type: dictionary of *environment "<variable>=<value>"* entries, with <*variable*> serving as the dictionary key.

To modify a particular environment entry:

```
pbs.event().env[<variable>] = <value>
```

To add more entries to the env[] dictionary:

```
pbs.event().env[<new_var>] = <value>
```

To clear out existing env[] entries and specify a new environment:

```
pbs.event().env = pbs.pbs_env()
pbs.event().env[<var1>] = <value1>
pbs.event().env[<var2>] = <value2>
…
pbs.event[<varN>.] = <valueN>
```

To unset an existing environment variable:

```
pbs.event().env[<var>] = None
```

To embed a comma in an environment variable, escape the value with single quotes:

```
pbs.event().env[<var>] = '"<value>"'
```

On Windows, where backslashes appear in pathnames, either escape the backslash with another backslash, or use the raw ('r') operator to form the string. Both of the following examples will work:

```
e = pbs.event()
e.progname = "C:\\Program Files\\PBS Pro\\exec\\bin\\pbsnodes.exe"
e.progname = r"C:\Program Files\PBS Pro\exec\bin\pbsnodes.exe"
```

See section 6.3.3, "Event Object Member Caveats", on page 115.

Example 6-1: To log the contents of a job's environment variables:

```
for v in pbs.event().env.keys():
    e = pbs.event().env[v]
    pbs.logmsg(pbs.LOG_DEBUG, "env[%s]=%s" % (v,e))
```

6.3.2.6 Failure Action Event Member

pbs.event().fail_action

Hook attribute. Action to take on hook failure or on subsequent successful execution. See "fail_action" on page 352 of the PBS Professional Reference Guide.

6.3.2.7 Frequency Event Member

pbs.event().freq

Hook attribute. Frequency at which to run hook. See "freq" on page 352 of the PBS Professional Reference Guide.

6.3.2.8 Hook Name Event Member

pbs.event().hook_name

Name of the hook being executed.

Type: *str*

6.3.2.9 Hook Type Event Member

pbs.event().hook_type

The type of the hook. The only valid value is "*site*". Represents the Type hook attribute.

Type: *str*

6.3.2.10 Job Event Member

pbs.event().job

The job that triggered the event. A pbs.job object. See section 6.6, "Job Objects", on page 122.

6.3.2.11 Job List Event Member

pbs.event().job_list

The list of jobs managed by the local MoM. Each job is a pbs.job, described in section 6.6, "Job Objects", on page 122.

For a list of settable attributes and resources, see Table 5-6, “Job Attributes Readable & Settable via Events,” on page 55 and Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

Type: dictionary of pbs.job objects

To print the jobs in the list:

```
for k in pbs.event().job_list.keys():
    print pbs.event().job_list[k]
```

To set a job attribute or resource for all jobs in the list:

```
for k in pbs.event().job_list.keys():
    pbs.event().job_list[k].<attribute> = <value>
```

In an exechost_periodic hook, attributes are set after the hook ends in a call to pbs.event().accept() or pbs.event().reject(), but not when the hook encounters an uncaught exception or hits an alarm call.

In an exechost_periodic hook, you can flag a job to be requeued using rerun() or deleted using delete() when the server is notified that the job has terminated.

Example 6-2: Rerun all jobs in this MoM's job list:

```
% cat period.py
import pbs
for k in pbs.event().job_list.keys():
    pbs.event().job_list[k].rerun()
```

6.3.2.12 Original Job Event Member

pbs.event().job_o

This is a pbs.job object representing the original job, before the job was modified via `qalter`. All resources and attributes are populated.

See section 6.6, “Job Objects”, on page 122.

6.3.2.13 Order Event Member

pbs.event().order

Hook attribute. Order in which to run hook. See “order” on page 352 of the PBS Professional Reference Guide.

6.3.2.14 Process ID Event Member

pbs.event().pid

The process ID of a task belonging to a job.

Type: *int*

6.3.2.15 Job Executable Event Member

pbs.event().progname

The path to the job shell or executable. This is settable in an execjob_launch hook as follows:

```
pbs.event().progname = "<path_to_the_script>"
```

When setting the value, specify the full path. Otherwise, the path may not be found, and the shell or executable may not run.

Type: *str*

6.3.2.16 Requestor Event Member

pbs.event().requestor

The requestor of the event.

PBS daemons can request actions. If a daemon requests an action, the requestor member contains one of "*PBS_Server*", "*Scheduler*", or "*pbs_mom*". If the requestor is root, the member contains "*root*".

For Windows systems, if the requestor is the administrator, the member contains the account name of the administrator.

Type: *str*

6.3.2.17 Requestor Host Event Member

pbs.event().requestor_host

The name of the host from which the event was requested.

Type: *str*

6.3.2.18 Reservation Event Member

pbs.event().resv

The reservation being requested in a resvsub event or ending in a resv_end event. See section 6.9, "Reservation Objects", on page 131.

6.3.2.19 Source Queue Event Member

pbs.event().src_queue

The pbs.queue object representing the original queue where pbs.event().job came from.

See section 6.5, "Queue Objects", on page 121.

6.3.2.20 Event Type Event Member

pbs.event().type

Hook attribute. The event type, for example, queuejob or movejob. Valid values: one of the PBS event type constants listed in section 6.3.1, "Event Types", on page 87. See "type" on page 352 of the PBS Professional Reference Guide.

Type: A PBS event type constant, such as pbs.QUEUEJOB, pbs.RESVSUB

6.3.2.21 Event User Event Member

pbs.event().user

Hook attribute. The username under which the hook executes. See "user" on page 352 of the PBS Professional Reference Guide.

Valid values: *pbsadmin*, *pbsuser*.

pbsadmin
: On Linux, this is root. On Windows, this is simply a substitute for the PBS service account; it is not the name of the PBS service account.

pbsuser
: The hook runs under the account of the job owner, which is the value of the euser job attribute. Can be used for execjob_prologue, execjob_epilogue, execjob_preterm events only.

Default value: *pbsadmin*

Type: *String*, *str*

6.3.2.22 The Vnode List Event Member

pbs.event().vnode_list[]

Execution event hooks have access to the list of vnodes assigned to the job. Periodic event hooks have access to the list of vnodes managed by the local MoM. The exechost_startup hook can create and modify the vnodes managed by the local MoM.

When a vnode is in such a list, the hook has access to the attributes and resources of that vnode. Table 6-3, "Using Event Object Members in Events," on page 108 lists which hooks can operate on vnode_list[]. Table 5-7, "Vnode Attributes Readable & Settable via Events," on page 57 and Table 5-10, "Vnode Resources Readable & Settable by Hooks via Events," on page 61 show which hooks can read and/or set each vnode attribute or resource.

When this list is retrieved through an execution event, it is associated with a job, and only vnodes assigned to the job have attributes, resources_available, resources_assigned.ncpus, and resources_assigned.mem filled in; on other vnodes, only pbs.vnode().name is available. See section 6.10, "Vnode Objects", on page 133.

You can use an exechost_startup hook to create vnodes on the host where the hook runs:

```
pbs.event().vnode_list[<new vnode>] = pbs.vnode(<new vnode name>)
```

If you want to use an execjob_ hook to manipulate a vnode that is not assigned to the job, but is still managed by the hook's MoM, you must first instantiate the object for that vnode with the name of the new vnode:

```
pbs.event().vnode_list[<new vnode>] = pbs.vnode(<new vnode name>)
```

Once you have instantiated your new vnode (which must still be managed by your hook's MoM), you can operate on it as shown here:

- To list all vnodes:

```
for v in pbs.event().vnode_list.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "found vnode %s" % (pbs.event().vnode_list[v].name))
```

- To get the vnode managed by the local MoM, use the pbs.get_local_nodename() function to return the local parent vnode name where this hook is executing, and then use pbs.event().vnode_list[<local parent vnode name>].

```
local_node = pbs.get_local_nodename()
```

- To find the other vnodes managed by the hook's MoM:

  a. Query the server for its list of vnodes:

  ```
  pbs.server().vnodes()
  ```

  b. Look in the Mom attribute in the resulting list of vnodes for a match to the output of:

  ```
  pbs.get_local_nodename()
  ```

  On a Cray, the Mom attribute may be a comma-separated list of strings, and you may have to check each one. Furthermore, get_local_nodename() returns the short hostname, and you may need to use instead the canonical hostname. If the name resolution on the server is consistent with the name resolution on the execution nodes, you can use the following, which returns in host_canon_name the name to be matched against Mom attribute of each vnode_list element:

  ```
  import socket
  [...]
  host_short_name = pbs.get_local_nodename()
  host_canon_name, host_aliases, host_addresses = socket.gethostbyname_ex(host_short_name)
  ```

To run your code only for vnodes managed by the specified MoM:

```
vnlist = pbs.event().vnode_list
for v in vnlist.keys():
   if host_canon_name in vnlist[v].Mom.split(","):
      [code]
```

- Setting and unsetting attributes and resources:

  To set the attributes and resources for a particular vnode:

  pbs.event().vnode_list[<vnode name>].<attribute> = <value>

  pbs.event().vnode_list[<vnode name>].<resources_available>["<resource name>"] = <value>

  You can unset a resource value by specifying "*None*" as its value:

  pbs.event().vnode_list[<vnode_name>].resources_available["<resource name>"] = None

  Resource names and string values must be quoted.

  For details and examples, see section 5.2.4.11, "Setting and Unsetting Vnode Resources and Attributes", on page 48.

- You can add new custom host-level, non-consumable resources and their values to resources_available for a vnode:

  vnode_list[<vnode name>].resources_available[<new resource>] = <value>

  For details and examples, see section 5.2.7, "Adding Custom Non-consumable Host-level Resources", on page 64.

You cannot modify a vnode that is managed by a different MoM from where the hook is running. If you try to do this, the following error message appears in the server's log at log event class PBSEVENT_DEBUG2:

```
"<node_host_name>; Not allowed to update <vnode name>, as it is owned by a different mom"
```

A hook that runs as "*pbsuser*" (execjob_prologue, execjob_epilogue, execjob_preterm) is not allowed to manipulate pbs.event().vnode_list[], unless the executing user is a PBS Manager or Operator. If a hook running as an unprivileged user tries to change pbs.event().vnode_list[], the following error message appears in the server's log at log event class PBSEVENT_DEBUG2:

```
"<node_host_name>; Not allowed to update vnodes or to request scheduler restart cycle, if run as
   a non-manager/operator user"
```

6.3.2.23 The Failed Vnode List Event Member

pbs.event().vnode_list_fail[]

For each execjob_prologue and execjob_launch event, PBS records the list of vnodes, with their assigned resources, that are marked as bad by MoM. This list can include those vnodes from sister MoMs that failed to join the job, that rejected an execjob_begin hook or execjob_prologue hook request, or that encountered a communication error while the primary MoM was polling the sister MoM host. PBS records this list in the pbs.event().vnode_list_fail[] hook parameter. For how vnodes are marked as failed, see "Checking Vnodes and Marking Them as Failed" on page 437 of the PBS Professional Administrator's Guide.

Type: *dict* (dictionary of pbs.vnode objects keyed by vnode name)

6.3.3 Event Object Member Caveats

6.3.3.1 Modifying progname or argv[] Under Windows

On Windows, in a multi-vnoded job, be careful modifying pbs.event().progname and pbs.event().argv[] parameters; some values are tacked on by pbs_mom and are required. For example, if a multi-vnode job has in its script:

```
pbsdsh -n 1 cmd.exe /C echo hi
```

This causes an installed execjob_launch hook to execute on the sister MoM specified at node index '*1*'. The execjob_launch hook sees:

```
pbs.event().progname=cmd.exe

pbs.event().argv[0]=cmd.exe
pbs.event().argv[1]=/c
pbs.event().argv[2]=C:/PROGRA~1/PBSPRO~1/exec/sbin/mom_open_demux.exe
pbs.event().argv[3]=174.host1
pbs.event().argv[5]=cmd.exe
pbs.event().argv[6]=/C
pbs.event().argv[7]=echo
pbs.event().argv[8]=hi
```

It is important not to modify pbs.event().progname and pbs.event().argv[0],...,pbs.event().argv[3]. These are automatically added by pbs_mom for execution and collecting output.

You can modify pbs.event().argv[] values starting at index *5*, and you can use pbs.event().argv.extend() to add more arguments. Here we modify values for indices *5* through *8*, and add pbs.event().argv[9], making it "*hello*":

```
pbs.event().argv[5] = "pbsnodes.exe"
pbs.event().argv[6] = "-a"
pbs.event().argv[7] = ""
pbs.event().argv[8] = ""
pbs.event().argv.extend(["hello"])
```

6.3.4 Event-only Methods

6.3.4.1 Event Method for Accepting Event

pbs.event().accept()

Terminates hook execution and causes PBS to perform the associated action.

6.3.4.2 Event Method for Rejecting Event

pbs.event().reject()

pbs.event().reject(["<error message>"][,<error code>])

Terminates hook execution and instructs PBS to not perform the associated action. If the <message> argument is given, it is shown in the appropriate PBS daemon log, and in the stderr of the PBS command that caused this event to take place.

By default, pbs.event().reject() returns *255*. To return an error code other than *255*, specify a value between *2* and *255* in the optional <error code>.

6.3.5 Event Object Method Caveats

pbs.event().accept() terminates hook execution by throwing a SystemExit exception. So if hook content appears in a try…except clause that has no arguments to the except clause, always add the following to treat SystemExit as a normal occurrence:

```
except SystemExit:
    pass
```

See section 5.3.7.1, "Treat SystemExit as a Normal Occurrence", on page 70.

6.3.6 Examples of Using Event Objects

Example 6-3: Inside a hook script, create a PBS event object:

```
e = pbs.event()
```

Example 6-4: Get the event type:

```
type = e.type
```

Example 6-5: Get the user who requested the event action:

```
who = e.requestor
```

Example 6-6: Get the host where the request came from:

```
host = e.requestor_host
```

Example 6-7: The event type is pbs.QUEUEJOB. Get the number of CPUs requested for the job being queued:

```
j = e.job
res = j.Resource_List["ncpus"]
```

Example 6-8: Reset the number of CPUs requested by the job:

```
j.Resource_List["ncpus"] = 1
```

Example 6-9: The event type is pbs.MOVEJOB. Get the request parameters:

```
j = e.job
q = j.queue
```

Example 6-10: Accept an event request:

```
e.accept()
```

Example 6-11: Reject an event request:

```
e.reject("Can't set interactive attribute")
```

Example 6-12: Put a job into a wait state and requeue the job in 3600 seconds (1 hour):

```
import time
...
j.Execution_Time = time.time() + 3600
```

Example 6-13: Put a hold on a job:

```
j = pbs.event().job
j.Hold_Types = pbs.hold_types("u")
j.Hold_Types = pbs.hold_types("uo")
j.Hold_Types += pbs.hold_types("s")
```

or

```
j.Hold_Types = pbs.hold_types("<hold_list>")
```

Example 6-14: Release a hold on a job:

```
j.Hold_Types -= pbs.hold_types("un")
j.Hold_Types -= pbs.hold_types("sp")
j.Hold_Types -= pbs.hold_types("o")
```

or

```
j.Hold_Types -= pbs.hold_types("<hold_list>")
```

6.4 Server Objects

pbs.server

This object represents a PBS server. This object can either represent the local server, or be just a coding construct, not representing an actual server. If it represents the local server, you can read but cannot set its attributes. If it is just a coding construct that does not represent an actual server, you can set its attributes. **You cannot alter the PBS server.** If this server object represents the PBS server, it is the server at which the triggering event is taking place, and at which the hook is executing. The only PBS server available to hooks is the local server.

s = pbs.server(["<name>"])

Creates an instance of a PBS server object. If <*name*> is not specified, the object represents the default server.

You can use pbs.server() to retrieve server, queue, job, vnode, and reservation information, and pass it to a hook script. You **cannot** set attributes or resources for objects that are retrieved through the server via pbs.server().

6.4.1 Server Object Members

Some server object members are server attributes, and some are not. A pbs.server has the following members:

6.4.1.1 Server Name Member

pbs.server().name

The server hostname.

Example: `myhost.mydomain.com`

This member is read-only.

Python type: *str*

6.4.1.2 Server Attribute Members

pbs.server().<attribute name>

The PBS server attribute named <*attribute name*>. The pbs.server object has a member to represent each server attribute, spelled exactly like the attribute. For information about using attributes, see section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

Server attributes are listed in “Server Attributes” on page 281 of the PBS Professional Reference Guide. Attribute creation methods are described in section 6.13.3, “PBS Types and Their Methods”, on page 143.

6.4.1.3 Server State Member

pbs.server().server_state

The server_state server attribute. It can take one of the following values, represented by constant objects:

Table 6-4: Server State Constant Objects

Object	State
pbs.SV_STATE_IDLE	*Idle*
pbs.SV_STATE_ACTIVE	*Scheduling*
pbs.SV_STATE_HOT	*Hot_Start*
pbs.SV_STATE_SHUTDEL	*Terminating, Delayed*
pbs.SV_STATE_SHUTIMM	*Terminating*
pbs.SV_STATE_SHUTSIG	*Terminating*

6.4.2 Setting Server Object Members

If the server object does not represent the PBS server, you can set, but not unset, server object members. If a server object does represent the PBS server, you cannot set values for object members. To set the value for the server attribute named *<attribute name>* to *<attribute value>*, where *s* is an instance of pbs.server:

```
s.<attribute name> = <attribute value>
```

6.4.3 Examples of Using Server Object Members

```
s = pbs.server()
```

Example 6-15: Get server name:

```
name = s.name
```

Example 6-16: Get the value of the server attribute pbs_license_min:

```
min = s.pbs_license_min
```

6.4.4 Server Object Methods

6.4.4.1 Method to Return Job

pbs.server().job('<job ID>')

Returns a pbs.job object for the job with ID <*id*>, residing on the local server. Returns *None* if the job with ID <*id*> does not exist at the server. See section 6.6, "Job Objects", on page 122.

6.4.4.2 Method to Return Job Iterator

pbs.server().jobs()

Returns a Python iterator that iterates over a list of pbs.job objects residing on the local server. Returns an empty iterator if no jobs exist on the local server. See section 6.6, "Job Objects", on page 122.

Example:

```
for j in s.jobs():
   pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.4.4.3 Method to Return Queue

pbs.server().queue("<queue_name>")

Returns a pbs.queue object representing the queue named *<queue name>* that is managed by the local server. See section 6.5, "Queue Objects", on page 121.

A value of *None* is returned if the queue named *<queue_name>* does not exist at the local server.

6.4.4.4 Method to Return Queue Iterator

pbs.server().queues()

Returns a Python iterator that iterates over a list of queue objects managed by the the local server. Returns an empty iterator if no queues exist at the local server. See section 6.5, "Queue Objects", on page 121.

6.4.4.5 Method to Return Reservation

pbs.server().resv("<reservation ID>")

Returns a pbs.resv object for <*reservation ID*> on the local server. Returns *None* if <*reservation ID*> does not exist. See section 6.9, "Reservation Objects", on page 131.

6.4.4.6 Method to Return Reservation Iterator

pbs.server().resvs()

Returns a Python iterator that iterates over a list of pbs.resv objects residing on the local server. Returns an empty iterator if no reservations exist at the local server. See section 6.9, "Reservation Objects", on page 131.

6.4.4.7 Method to Restart Scheduler Cycle

pbs.server().scheduler_restart_cycle()

This directs the current PBS server to tell the scheduler to restart its scheduling cycle.

A hook with its user attribute set to *pbsuser* cannot successfully invoke pbs.scheduler_restart_cycle(), unless the hook's executing user is a PBS Manager or Operator. If this is attempted, the scheduler is not restarted, and the following message appears at log event class PBSEVENT_DEBUG2 in the MoM logs:

```
"<node_host_name>;Not allowed to update vnodes or to request scheduler_restart_cycle, if run as a
   non-manager/operator user"
```

6.4.4.8 Method to Return Named Vnode

pbs.server().vnode("<vnode name>")

Returns a pbs.vnode object representing the vnode with name *<vnode name>* that is managed by the current server.

Returns *None* if *<vnode name>* does not exist.

6.4.4.9 Method to Return Vnode List

pbs.server().vnodes()

Returns a list of pbs.vnode objects managed by current server.

Returns an empty iterator if no vnodes exist at the local server.

Example:

```
for vn in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "found vn %s" % (vn.name))
```

6.5 Queue Objects

pbs.queue

This object represents a PBS queue. This object can either represent an actual PBS queue, or be just a coding construct, not representing an actual queue. If it is just a coding construct, you can set its attributes. If it represents an actual queue, you can read but cannot set its attributes. **You cannot set the attributes of any actual queue in any hook**.

To get information about a particular queue with name <*name*>, you must go through the associated server. Use:

```
q = pbs.server().queue("<name>")
```

To get a list of queues from the server:

```
pbs.server().queues()
```

6.5.1 Queue Object Members

Some queue object members are queue attributes, and some are not.

6.5.1.1 Queue Object Name Member

queue.name

The queue name.

This member is read-only.

Python type: *str*

6.5.1.2 Queue Object Attribute Members

A pbs.queue has a member representing each of its attributes. Each member that is not a *string*, *int*, *bool*, *long*, or *float* has a corresponding creation method; see section 6.13.3, “PBS Types and Their Methods”, on page 143. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

queue.<attribute name>

The queue attribute named <*attribute name*>. Queue attributes are listed in “Queue Attributes” on page 311 of the PBS Professional Reference Guide.

Example 6-17: Get the queue object representing the queue *workq*, and its Priority value:

```
q = s.queue("workq")
prio = q.Priority
```

6.5.1.3 Setting Queue Object Attributes

You can set or unset queue object attributes for queue objects that **don't** represent an actual queue. To set the value of a queue object attribute named <*attribute name*>:

```
pbs.queue.<attribute name> = <attribute value>
```

You cannot set or unset attributes for an actual queue.

6.5.2 Queue Object Methods

6.5.2.1 Method to Return Job

queue.job()

pbs.queue.job("<job ID>")

Returns a pbs.job object representing PBS job with ID <*job ID*>. This job must be residing on the queue. Returns *None* if the job with the specified job ID does not exist, or if the job is not in the queue. See section 6.6, "Job Objects", on page 122.

6.5.2.2 Method to Return Job Iterator

queue.jobs()

Returns a Python iterator that iterates over a list of pbs.job objects representing the jobs on the queue. Returns an empty iterator if no jobs exist on the queue. See section 6.6, "Job Objects", on page 122.

Example:

```
for j in pbs.server().queue("workq").jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.5.3 Queue Type Constant Objects

Queue types are represented by constant objects. The pbs.queue.queue_type member represents the type of the queue. It can take on the following values:

Table 6-5: Queue Type Constant Objects

Object	Queue Type
pbs.QTYPE_EXECUTION	*Execution*
pbs.QTYPE_ROUTE	*Route*

6.6 Job Objects

pbs.job

A job object represents a PBS job.

You can retrieve the job object either through an associated event or through the server. The job object represents one of the following, depending on how it is retrieved:

- The PBS job associated with the event that triggers the hook. To get the job associated with the current event, go through the event that triggered the hook:

  pbs.event().job

  A call to pbs.event().job can return only the job associated with the current event.

When you get a job using pbs.event().job, the hook can read and set the job attributes and resources listed in Table 5-6, "Job Attributes Readable & Settable via Events," on page 55 and Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

- A job at the server at which the hook is executing. To get a particular job with ID <*id*>, go through the server:

  pbs.server().job("<job ID>")

  When you get a job using pbs.server().job("<job ID>"), the hook can read all job attributes and resources, but can set none.

- To get a list of jobs at the server:

  pbs.server().jobs()

For information about a list of jobs visible through events, see section 6.3.2.11, "Job List Event Member", on page 111 and Table 6-3, "Using Event Object Members in Events," on page 108 for the events that can use this list.

All job objects have the same members and methods. Each hook can read or set different attributes and resources. We describe what each type of hook can do in section 6.3, "Event Objects", on page 87.

If you use a hook to make a change to a job, that change is visible to all PBS daemons.

6.6.1 Job Object Members

A pbs.job object has a member to represent each job attribute. Each one is spelled exactly like the corresponding attribute. We list job object members here that require creation methods, require special treatment, or that are not job attributes. All job attribute members that are not listed here are defined this way:

job.<attribute name>

The type of the attribute is given in the attribute description, in "Job Attributes" on page 328 of the PBS Professional Reference Guide.

For information about using job attributes in hooks, see section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

To see which hooks can set which job attributes and resources, see Table 5-6, "Job Attributes Readable & Settable via Events," on page 55 and Table 5-9, "Built-in Job Resources Readable & Settable by Hooks via Events," on page 60.

A pbs.job object also has the id member, which is not a job attribute.

6.6.1.1 Job ID Member

job.id

The PBS job ID.

Read-only.

Python type: *str*

6.6.1.2 Job array_indices_submitted Attribute Member

job.array_indices_submitted

Job attribute. Python type: *range*

See section 6.13.3.21, "Method to Create or Set range Object", on page 148.

6.6.1.3 Job Checkpoint Attribute Member

job.Checkpoint

Job attribute. Python type: *pbs.checkpoint*

See section 6.13.3.3, “Method to Create or Set Checkpoint String”, on page 144.

6.6.1.4 Job depend Attribute Member

job.depend

Job attribute. Python type: *pbs.depend*

See section 6.13.3.4, “Method to Create or Set Dependency Object”, on page 144.

6.6.1.5 Job Execution_Time Attribute Member

job.Execution_Time

Job attribute. Time when the current job is eligible to run. Syntax:

job.Execution_Time = time.mktime([<YY>, <MM>, <DD>, <HH>, <MM>, <SS>, <WEEKDAY>, <YEARDAY> <ISDST>])

For example, the following sets a job's Execution_Time to: *March 1, 2012 at 09:00 am*:

```
job.Execution_Time = time.mktime([2012, 3, 1, 12, 9, 0, -1, -1, -1])
```

Python type: *int*

6.6.1.6 Job exec_host Attribute Member

job.exec_host

Job attribute. Python type: *pbs.exec_host*

See section 6.13.3.7, “Method to Create or Set exec_host Object”, on page 145.

6.6.1.7 Job exec_vnode Attribute Member

job.exec_vnode

Job attribute. Python type: *pbs.exec_vnode*

This complex object is described in section 6.7, “The exec_vnode Object”, on page 129.

See also section 6.13.3.8, “Method to Create or Set exec_vnode Object”, on page 145.

6.6.1.8 Job group_list Attribute Member

job.group_list

Job attribute. Python type: *pbs.group_list*

See section 6.13.3.9, “Method to Create or Set group_list Object”, on page 145.

6.6.1.9 Job Hold_Types Attribute Member

job.Hold_Types

Job attribute. Python type: *pbs.hold_types*

See section 6.13.3.10, “Method to Create or Set hold_types Object”, on page 145.

6.6.1.10 Job job_state Attribute Member

job.job_state

Job attribute. Represents the job's state. Can be compared to the constants representing job states.

Use job state constant objects to test the state of a job. For example:

```
e = pbs.event()
if e.job.job_state == pbs.JOB_STATE_RUNNING :
    e.accept()
```

The job_state member can take on any of the values listed here:

Table 6-6: Job State Objects

Object	State	Description
pbs.JOB_STATE_BEGUN	*B*	Job arrays only: job array has started
pbs.JOB_STATE_EXITING	*E*	Job is exiting after having run
pbs.JOB_STATE_EXPIRED	*X*	Subjobs only; subjob is finished (expired.)
pbs.JOB_STATE_FINISHED	*F*	Job is finished: job executed successfully, job was terminated while running, job execution failed, or job was deleted before execution
pbs.JOB_STATE_HELD	*H*	Job is held.
pbs.JOB_STATE_MOVED	*M*	Job has been moved to another server
pbs.JOB_STATE_QUEUED	*Q*	Job is queued, eligible to run or be routed
pbs.JOB_STATE_RUNNING	*R*	Job is running
pbs.JOB_STATE_SUSPEND	*S*	Job is suspended by PBS so that a higher-priority job can run.
pbs.JOB_STATE_SUSPEND_USERACTIVE	*U*	Job is suspended due to workstation becoming busy
pbs.JOB_STATE_TRANSIT	*T*	Job is in transition (being moved to a new location)
pbs.JOB_STATE_WAITING	*W*	Job is waiting for its requested execution time to be reached, or the job's specified stagein request has failed for some reason.

6.6.1.11 Job Join_Path Attribute Member

job.Join_Path

Job attribute. Python type: *pbs.join_path*

See section 6.13.3.12, “Method to Create or Set join_path Object”, on page 146.

6.6.1.12 Job Keep_Files Attribute Member

job.Keep_Files

Job attribute. Python type: *pbs.keep_files*

See section 6.13.3.13, “Method to Create or Set keep_files Object”, on page 146.

6.6.1.13 Job Mail_Points Attribute Member

job.Mail_Points

Job attribute. Python type: *pbs.mail_points*

See section 6.13.3.15, “Method to Create or Set mail_points Object”, on page 146.

6.6.1.14 Job Mail_Users Attribute Member

job.Mail_Users

Job attribute. Python type: *pbs.email_list*

See section 6.13.3.6, “Method to Create or Set Email List”, on page 144.

6.6.1.15 Job Queue Attribute Member

job.queue

Job attribute. Python type: *pbs.queue*

6.6.1.16 Job Resource_List Attribute Member

job.Resource_List[]

job.Resource_List["<resource name>"]

Job attribute. The job's Resource_List attribute.

Python type: dictionary: *Resource_List["<resource name>"]=<value>* where <*resource name*> is any built-in or custom resource

6.6.1.17 Job resources_used Attribute Member

job.resources_used["<resource name>"]

Job attribute. The job's resources_used attribute, which lists the resources used by the job. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

Python type: dictionary: *resources_used["<resource name>"]=<value>* where <*resource name*> is any built-in or custom resource

6.6.1.18 Job run_count Attribute Member

job.run_count

Job attribute. Execution hooks must run with user = *pbsadmin* to reduce the value of this member. Execution hooks running with user = *pbsuser* cannot reduce the value of this member.

Python type: *int*

6.6.1.19 Job stagein and stageout Attribute Members

job.stagein

job.stageout

Job attribute. Python type: *pbs.staging_list*

See section 6.13.3.27, “Method to Create or Set staging_list Object”, on page 150.

6.6.1.20 Job User_List Attribute Member

job.User_List

Job attribute. Python type: *pbs.user_list*

See section 6.13.3.29, "Method to Create or Set user_list Object", on page 151.

6.6.1.21 Job Variable_List Attribute Member

job.Variable_List[<variable>]

Job attribute. Holds the job's environment variables. Syntax:

job.Variable_List[<variable>] = <value>

Python type: dictionary: *Variable_List["<variable name>"]=<value>*

6.6.2 Setting Job Attributes

How you set a job attribute depends on the type of the attribute; those of type *str*, *int*, *bool*, *long*, and *float* can be set directly. Job attributes of other types require creation methods. Job attribute creation methods are listed in section 6.13.3, "PBS Types and Their Methods", on page 143.

To set job attributes and resources directly:

pbs.event().job.<attribute> = <value>

pbs.event().job.Resource_List["<resource name>"] = <value>

See section 5.2.4.3, "Determining Whether to Use Creation Method to Set Attribute or Resource", on page 45.

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.6.3 Examples of Using Job Object Members

Get the job's Priority value:

```
prio = job.Priority
```

Reset the Priority value of job *j*:

```
job.Priority = 5
```

Get the job's PBS_O_WORKDIR environment variable:

```
workdir = job.Variable_List["PBS_O_WORKDIR"]
```

6.6.4 Job Object Methods for Execution Hooks

Job objects have the following methods. Most methods are available in execjob_ hooks except for the execjob_launch hook, and in the exechost_periodic hook.

6.6.4.1 Job Object Method to Report Checkpoint

job.is_checkpointed()

Returns a Python bool value which is *True* if the job was checkpointed under the control of the PBS MoM.

For example, you could use this in an execjob_epilogue hook, where the hook writer directs the job to be requeued if the job was checkpointed under the control of PBS:

```
# cat epi.py
import pbs
If pbs.event().job.is_checkpointed():
    pbs.event().job.rerun()
    pbs.event().reject("job to be requeued")
# qmgr -c "create hook epi event=execjob_epilogue"
# qmgr - c "import hook epi application/x-python default epi.py"
```

6.6.4.2 Job Object Method to Report Execution Host Role

job.in_ms_mom()

Returns a Python bool value. Returns *True* if this job object is running on the primary execution host.

6.6.4.3 Job Object Method to Delete Job

job.delete()

When this method is used in an execution hook, the job is flagged at the server for deletion after its processes have terminated and any epilogue or execjob_epilogue hook has run.

When this method is used in a non-execution hook script, it raises a Python "NotImplementedError" exception.

If the job.delete() method is used in an execjob_end hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The job.delete() method overrides the job.rerun() method. If both are used, job.delete() takes precedence.

6.6.4.4 Job Object Method to Release Vnodes

job.release_nodes()

job.release_nodes(keep_select=<select specification>)

Automatically selects vnodes that satisfy the new request and are healthy, keeps them in the job's vnode request, and releases all others. The method automatically trims out any vnodes in the pbs.event().vnode_list_fail[] list.

You can call pbs.event().job.release_nodes(keep_select = <desired vnodes>) in an execjob_launch or execjob_prologue hook. Note that despite the method being named "release_nodes", it **keeps** the specified vnodes and **releases** all other vnodes. You can specify the job's original vnode request as the vnodes to keep.

The pbs.event().job.release_nodes() method returns a PBS job object which has the updated values for the job's exec_vnode and Resource_List attributes.

This method is only effective when it runs at the primary MoM.

This method can be used only when it's used for a job whose tolerate_node_failures attribute is set to *job_start* or *all*.

6.6.4.4.i Advice and Recommendations for Using release_nodes Method

- Put the call to this method in an 'if pbs.event().job.in_ms_mom()' clause
- Request vnodes that are a subset of the existing vnode request
- Because an execjob_launch hook is also called when spawning tasks via pbsdsh and tm_spawn, ensure that any execjob_launch hook invoking release_nodes() has 'PBS_NODEFILE' in the pbs.event().env list. The presence of 'PBS_NODEFILE' in the environment ensures that the primary MoM is executing on behalf of starting the top level job, and not spawning a sister task. You can add the following at the top of the hook:

```
if 'PBS_NODEFILE' not in pbs.event().env:
   pbs.event().accept()
...
pbs.release_nodes(keep_select=...)
```

- On Windows, where PBS_NODEFILE always appears in pbs.event().env, put the following at the top of any execjob_launch hook:

```
if any("mom_open_demux.exe") in s for s in e.argv):
   e.accept()
```

6.6.4.4.ii Side Effects of Using release_nodes() Method

When release_nodes() is successfully executed from execjob_prologue or execjob_launch hooks, the following happen:

- PBS generates the *s* accounting record.
- The primary MoM notifies the sister MoMs to update their internal nodes tables, so that the task manager API (e.g. tm_spawn, pbsdsh) will be aware of the change in the future.
- If the pbs_cgroups hook is enabled, the cgroup already created for the job is updated to match the job's new resources. If the kernel rejects the update to the job's cgroup resources, the job is aborted at the execution host, and requeued/rerun at the server.

6.6.4.5 Job Object Method to Re-run Job

job.rerun()

When this method is used in an execution hook, the job is flagged at the server for requeueing after its processes have terminated and any epilogue or execjob_epilogue hook has run.

When this method is used in a non-execution hook script, it raises a Python "NotImplementedError" exception.

If the job.rerun() method is used in an execjob_end hook, it has no effect, because in this case the server has already performed end-of-job processing before the execution hook runs.

The job.delete() method overrides the job.rerun() method. If both are used, job.delete() takes precedence.

6.7 The exec_vnode Object

pbs.exec_vnode

The exec_vnode object represents the job's exec_vnode attribute.

6.7.1 The exec_vnode Object Members

A pbs.exec_vnode object has the following member:

6.7.1.1 The exec_vnode Chunks Member

pbs.exec_vnode.chunks[]

List of pbs.vchunk objects. These objects represent the chunks assigned to a job. See section 6.8, "Chunk Objects", on page 131.

6.7.2 Using pbs.vchunk Objects in exec_vnode

- To get a list of pbs.vchunks in pbs.event().job.exec_vnode:

  pbs.event().job.exec_vnode.chunks

  For example, to log the name of the vnode containing each vchunk:

```
chunklist = pbs.event().job.exec_vnode.chunks
for chunk in chunklist:
    pbs.logmsg(pbs.LOG_DEBUG, "chunk.vnode_name=%s " % (chunk.vnode_name))
```

- To get a pbs.vchunk with a specific index:

  pbs.event().job.exec_vnode.chunks[<index>]

- For example, to get the vchunk in pbs.event().job.exec_vnode with index number *2*:

  pbs.event().job.exec_vnode.chunks[2]

Example 6-18: List the job ID, vnode name, and resources in exec_vnode:

```
j = pbs.event().job
pbs.logmsg(pbs.LOG_DEBUG, "job %s exec_vnode = %s" % (j.id, j.exec_vnode))

chunklist = j.exec_vnode.chunks
for c in chunklist:
    pbs.logmsg(pbs.LOG_DEBUG, "c.vnode_name=%s " % (c.vnode_name))

    for r in c.chunk_resources.keys():
        pbs.logmsg(pbs.LOG_DEBUG, "c.chunk_resources[%s]=%s" % (r,
            c.chunk_resources[r]))
```

Sample output:

```
10:16:53;0006;Server@jobim;Hook;Server@jobim;job 153.jobim exec_vnode  =
    (jobim[2]:ncpus=2:mem=10240kb)+ (jobim[1]:ncpus=2:mem=10240kb) +
    (jobim[3]:ncpus=2:mem=2048kb)
10:16:53;0006;Server@jobim;Hook;Server@jobim;c.vnode_name= jobim[2]
    10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.vnode_name=jobim[1]
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=10240kb
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.vnode_name=jobim[3]
10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[ncpus]=2
    10:16:53;0006;Server@jobim;Hook;Server@jobim; c.chunk_resources[mem]=2048kb
```

6.7.3 Restrictions on exec_vnode Objects

A job's exec_vnode attribute is read-only. You cannot set its value, and you cannot build an exec_vnode object using pbs.vchunk objects.

6.8 Chunk Objects

pbs.vchunk

The pbs.vchunk object represents a chunk specification. It is used in a job's exec_vnode attribute or select statement.

6.8.1 Chunk Object Members and Methods

A pbs.vchunk object has the following members:

6.8.1.1 Chunk Object Vnode Name Member

vchunk.vnode_name

Name of the vnode from which the chunk is taken.

Python type: *str*

6.8.1.2 Chunk Object Chunk Resources Member

vchunk.chunk_resources[]

Resources assigned to the chunk.

Python type: Dictionary containing *<resource name>=<value>* pairs.

Syntax: *chunk_resources['<resource name>'] = <resource value>* where *<resource name>* is any custom or built-in resource.

6.8.1.3 Chunk Object Method to Return chunk_resources Keys

vchunk.chunk_resources.keys()

Returns list of *<resource name>* keys of chunk_resources[]. This list makes it convenient to list all the values of chunk_resources[].

6.9 Reservation Objects

pbs.resv

This represents a PBS reservation. If the reservation is associated with the triggering event, you can read and set reservation attributes and resources in a resvsub hook, and read them in a resv_end hook. See Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59 for a complete list of the reservation attributes and resources that can be set in the resvsub and resv_end hooks. If the reservation is retrieved through the server, and is not associated with the triggering event, you can read all its attributes and resources, but set none.

If you are working with the reservation being created using `pbs_rsub`, you must use *pbs.event().resv*. The server cannot return information about the reservation, because it has not yet been created.

In order to retrieve information about the reservation associated with the triggering action, you must use a reference to the reservation object represented by:

pbs.event().resv

To get a copy of a particular reservation, use:

pbs.server().resv("<reservation name>")

To get a list of the reservations at a server:

pbs.server().resvs()

6.9.1 Reservation Object Members

A pbs.resv object has members that represent reservation attributes, and the resvid member which exists for the job object but is not an attribute of a reservation.

6.9.1.1 Reservation ID Member

resv.resvid

The reservation ID.

Example: "R221.myhost".

This member is read-only.

Python type: *str*

6.9.1.2 Reservation Attribute Members

resv.<attribute name>

The reservation attribute named *<attribute name>*. Each member is spelled exactly like the corresponding attribute.

6.9.1.3 Setting Reservation Object Attribute Values

You can set, but not unset, reservation object attributes.

To see a list of which reservation attributes can be read and set by each hook, see Table 5-8, "Reservation Attributes Readable & Settable in resvsub and resv_end Hooks," on page 59.

Some attributes require creation methods when setting them. See section 5.2.4.3, "Determining Whether to Use Creation Method to Set Attribute or Resource", on page 45. To set a simple reservation object attribute:

```
pbs.resv.<attribute name> = <attribute value>
```

Reservation attribute creation methods are listed in section 6.13.3, "PBS Types and Their Methods", on page 143.

See section 5.2.4, "Using Attributes and Resources in Hooks", on page 44.

6.9.1.4 Examples of Using Reservation Object Attributes

Example 6-19: Get the reservation's owner:

```
owner = pbs.server().resv(<reservation ID>).Reserve_Owner
```

Example 6-20: Reset the reservation's name:

```
pbs.event().resv(<reservation ID>).Reserve_Name = "Resv2008"
```

6.9.2 Reservation State Constant Objects

The pbs.resv.reserve_state member represents the state of the reservation. It can take on the following values, which are represented by constant objects:

Table 6-7: Reservation State Objects

Object	State
pbs.RESV_STATE_NONE	*RESV_NONE*
pbs.RESV_STATE_UNCONFIRMED	*RESV_UNCONFIRMED*
pbs.RESV_STATE_CONFIRMED	*RESV_CONFIRMED*
pbs.RESV_STATE_WAIT	*RESV_WAIT*
pbs.RESV_STATE_TIME_TO_RUN	*RESV_TIME_TO_RUN*
pbs.RESV_STATE_RUNNING	*RESV_RUNNING*
pbs.RESV_STATE_FINISHED	*RESV_FINISHED*
pbs.RESV_STATE_BEING_ALTERED	*RESV_BEING_ALTERED*
pbs.RESV_STATE_BEING_DELETED	*RESV_BEING_DELETED*
pbs.RESV_STATE_DELETED	*RESV_DELETED*
pbs.RESV_STATE_DELETING_JOBS	*RESV_DELETING_JOBS*
pbs.RESV_STATE_DEGRADED	*RESV_DEGRADED*
pbs.RESV_STATE_IN_CONFLICT	*RESV_IN_CONFLICT*

6.10 Vnode Objects

pbs.vnode

Represents a PBS vnode.

The way in which you retrieve a vnode controls what you can do with the vnode. If a vnode is retrieved through an event, using *pbs.event().v node_list[]*, and is managed by the same MoM where the event hook runs, you can set the vnode attributes and resources listed in Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57 and Table 5-10, “Vnode Resources Readable & Settable by Hooks via Events,” on page 61. However, if a vnode is not retrieved through an event, or is not managed by the same MoM where the hook runs, you can read all vnode attributes and resources, but set none.

Execution events have access to the list of vnodes associated with the job. Periodic events have access to the list of vnodes managed by the local MoM. See section 6.3.2.22, “The Vnode List Event Member”, on page 114.

- To retrieve the list of vnodes associated with an execution event or a periodic event:

  pbs.event().vnode_list[]

- To retrieve a specific vnode that is associated with an execution or periodic event, use the list of vnodes associated with the event, and specify the vnode name:

  pbs.event().vnode_list["<vnode name>"]

- To retrieve the vnodes associated with a pre-execution event, get the job's exec_vnode attribute:

pbs.event().job.exec_vnode

- To retrieve the server's list of vnodes:

  pbs.server().vnodes()

- To retrieve a named vnode through the server:

  pbs.server().vnode("<vnode name>")

6.10.1 Vnode Object Members

vnode.<attribute name>

A pbs.vnode object has a member representing each attribute, and each member is spelled exactly like the corresponding attribute. Table 5-7, “Vnode Attributes Readable & Settable via Events,” on page 57 lists which vnode attributes can be set by each hook. See section 5.2.4.3, “Determining Whether to Use Creation Method to Set Attribute or Resource”, on page 45. Attribute creation methods are listed in section 6.13.3, “PBS Types and Their Methods”, on page 143. See section 5.2.4, “Using Attributes and Resources in Hooks”, on page 44.

6.10.1.1 The topology_info Attribute Member

vnode.topology_info

Vnode attribute. The topology_info vnode attribute shows topology information. This attribute is visible only in hooks, and can be used only in hooks.

Python type: *str*

6.10.1.2 Vnode Attribute Restrictions

- The only vnode attribute that can be changed by a pre-execution hook is the state attribute
- The only pre-execution hook that can change the vnode state attribute is the runjob hook
- Execution and periodic hooks can change all settable vnode attributes

6.10.2 Vnode Type Constant Objects

The pbs.vnode.ntype member represents the type of the vnode. It can take on the following values:

Table 6-8: Vnode Type Objects

Object	Type
pbs.ND_PBS	Represents *pbs* value for vnode ntype attribute

6.10.3 Vnode Sharing Constant Objects

The pbs.vnode.sharing member represents the vnode's sharing attribute. It can take on the following values:

Table 6-9: Vnode Sharing Objects

Object	Sharing Value
pbs.ND_DEFAULT_EXCL	Represents *default_excl* vnode sharing attribute value
pbs.ND_DEFAULT_EXCLHOST	Represents *default_exclhost* vnode sharing attribute value
pbs.ND_DEFAULT_SHARED	Represents *default_shared* vnode sharing attribute value
pbs.ND_FORCE_EXCL	Represents *force_excl* vnode sharing attribute value
pbs.ND_FORCE_EXCLHOST	Represents *force_exclhost* vnode sharing attribute value
pbs.ND_IGNORE_EXCL	Represents *ignore_excl* vnode sharing attribute value

6.10.4 Vnode State Constant Objects

The pbs.vnode.state member represents the state of the vnode. It can take on the following values:

Table 6-10: Vnode State Constant Objects

Object	State
pbs.ND_BUSY	Represents *busy* vnode state
pbs.ND_DOWN	Represents *down* vnode state
pbs.ND_FREE	Represents *free* vnode state
pbs.ND_JOBBUSY	Represents *job-busy* vnode state
pbs.ND_JOB_EXCLUSIVE	Represents *job-exclusive* vnode state
pbs.ND_OFFLINE	Represents *offline* vnode state
pbs.ND_PROV	Represents *provisioning* vnode state
pbs.ND_RESV_EXCLUSIVE	Represents resv-exclusive vnode state
pbs.ND_STALE	Represents *stale* vnode state
pbs.ND_STATE_UNKNOWN	Represents *state-unknown, down* vnode state
pbs.ND_UNRESOLVABLE	Represents *unresolvable* vnode state
pbs.ND_WAIT_PROV	Represents *wait-provisioning* vnode state

6.11 Configuration File Objects

6.11.1 Variable Containing Hook Configuration File Path

pbs.hook_config_filename

Contains the path to the hook's configuration file, or *None* if there is no configuration file.

6.11.2 Dictionary of PBS Configuration File Entries

pbs.pbs_conf[]

This is a dictionary of values which represent entries in the pbs.conf file.

This reflects the contents of /etc/pbs.conf on the host where a hook runs, so pre-execution event (server) hooks get the entries on the server host, and execution event (MoM) hooks get the entries on the execution host where the hook runs.

Example of using pbs.pbs.conf[]:

```
pbs.logmsg(pbs.LOG_DEBUG, "pbs home is %s" % (pbs.pbs_conf['PBS_HOME']))
```

If you change /etc/pbs.conf, HUP pbs_mom (Linux) and/or restart pbs_server to rebuild the dictionary with the new contents of pbs.conf.

Each parameter in the pbs.conf file is the key to its dictionary entry. The pbs.conf file can contain the following parameters:

Table 6-11: Parameters in pbs.conf

Parameter	Description
PBS_AUTH_METHOD	Specifies default authentication method and library to be used by PBS. Used only at authenticating client. Case-insensitive. Default value: *resvport* To use MUNGE, set to *munge*
PBS_BATCH_SERVICE_PORT	Port on which server listens. Default: 15001
PBS_BATCH_SERVICE_PORT_DIS	DIS port on which server listens.
PBS_COMM_LOG_EVENTS	Communication daemon log mask. Default: *511*
PBS_COMM_ROUTERS	Tells a pbs_comm the location of the other pbs_comms.
PBS_COMM_THREADS	Number of threads for communication daemon.
PBS_REMOTE_VIEWER	Specifies remote viewer client. If not specified, PBS uses native Remote Desktop client for remote viewer. Set on submission host(s). Supported on Windows only.
PBS_CORE_LIMIT	Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string "unlimited". If unset, core file size limit is inherited from the shell environment.
PBS_DATA_SERVICE_PORT	Used to specify non-default port for connecting to data service. Default: 15007
PBS_ENCRYPT_METHOD	Specifies method and library for encrypting and decrypting data in client-server communication. Used only at authentication client side. Case-insensitive. To use TLS encryption in client-server communication, set this parameter to *tls*. No default; if this is not set, PBS does not encrypt or decrypt data.
PBS_ENVIRONMENT	Location of pbs_environment file.

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_EXEC	Location of PBS `bin` and `sbin` directories.
PBS_HOME	Location of PBS working directories.
PBS_LEAF_NAME	Tells endpoint what hostname to use for network. The value does not include a port, since that is usually set by the daemon. By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation. TPP internally resolves the name to a set of IP addresses, so you do not affect how `pbs_comm` works.
PBS_LEAF_ROUTERS	Location of endpoint's `pbs_comm` daemon(s).
PBS_LOCALLOG=<value>	Enables logging to local PBS log files. Valid values: 0: no local logging 1: local logging enabled Only available when using `syslog`.
PBS_LOG_HIGHRES_TIMESTAMP	Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is *HH:MM:SS*. With microsecond logging, format is *HH:MM:SS:XXXXXX*. Does not affect accounting log. Not applicable when using `syslog`. Overridden by environment variable of the same name. Valid values: *0*, *1*. Default: *0* (no microsecond logging)
PBS_LR_SAVE_PATH	Path where Undo Live Recorder stores daemon execution recordings. Default: `$PBS_HOME/spool`. See "Sending Daemon Execution Recordings to Altair" on page 663 in the PBS Professional Administrator's Guide
PBS_MAIL_HOST_NAME	Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). For how this is used in email address, see section 2.2.2, "Specifying Mail Delivery Domain", on page 20.
PBS_MANAGER_SERVICE_PORT	Port on which MoM listens. Default: 15003
PBS_MOM_HOME	Location of MoM working directories.

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_MOM_NODE_NAME	Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by `gethostname()`. If you use the IP address for a vnode name, set `PBS_MOM_NODE_NAME=<IP address>` in `pbs.conf` on the execution host. Dots are not allowed in this parameter unless they are part of an IP address.
PBS_MOM_SERVICE_PORT	Port on which MoM listens. Default: 15002
PBS_OUTPUT_HOST_NAME	Host to which all job standard output and standard error are delivered. If specified in `pbs.conf` on a job submission host, the value of PBS_OUTPUT_HOST_NAME is used in the host portion of the job's Output_Path and Error_Path attributes. If the job submitter does not specify paths for standard output and standard error, the current working directory for the `qsub` command is used, and the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies only a file path for standard output and standard error, the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies paths for standard output and standard error that include host names, the specified paths are used. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Delivering Output and Error Files" on page 60 in the PBS Professional Administrator's Guide.
PBS_PRIMARY	Hostname of primary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.
PBS_RCP	Location of `rcp` command if `rcp` is used.
PBS_SCHEDULER_SERVICE_PORT	Port on which default scheduler listens. Default value: 15004
PBS_SCHED_THREADS	Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable. Overridden by `pbs_sched -t` option and PBS_SCHED_THREADS environment variable. Default: 1
PBS_SCP	Location of `scp` command if `scp` is used; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_SECONDARY	Hostname of secondary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
PBS_SERVER	Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER. Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.
PBS_SERVER_HOST_NAME	The FQDN of the server host. Used by clients to contact server. Overridden by PBS_PRIMARY and PBS_SECONDARY failover parameters. Overrides PBS_SERVER parameter. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Contacting the Server" on page 60 in the PBS Professional Administrator's Guide.
PBS_START_COMM	Set this to *1* if a communication daemon is to run on this host.
PBS_START_MOM	Default is *0*. Set this to *1* if a MoM is to run on this host.
PBS_START_SCHED	**Deprecated**. Set this to *1* if default scheduler is to run on this host. Overridden by scheduler's scheduling attribute.
PBS_START_SERVER	Set this to *1* if server is to run on this host.
PBS_SUPPORTED_AUTH_METHODS	Specifies supported authentication methods for client-server communication. Used by authenticating server (PBS server, scheduler, MoM, or comm); ignored at client. Case-insensitive. If this parameter is set, PBS accepts only the methods listed. Format: comma-separated list of authentication methods. Default value: *resvport* Example: *munge,GSS*

Table 6-11: Parameters in `pbs.conf`

Parameter	Description
PBS_SYSLOG=<value>	Controls use of `syslog` facility under which the entries are logged. Valid values: 0: no syslogging 1: logged via LOG_DAEMON facility 2: logged via LOG_LOCAL0 facility 3: logged via LOG_LOCAL1 facility ... 9: logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=<value>	Filters `syslog` messages by severity. Valid values: 0: only LOG_EMERG messages are logged 1: messages up to LOG_ALERT are logged ... 7: messages up to LOG_DEBUG are logged
PBS_TMPDIR	Location of temporary files/directories used by PBS components.

6.12 Constant Objects

Constant objects are used to represent PBS elements such as event types, job, server, reservation, and vnode states, log event classes, queue and vnode types, and exceptions. These objects cannot be modified. When the PBS module is imported, the constant objects are imported. PBS uses the following constant objects:

Table 6-12: Constant Objects

Category	Description
Event type objects	The pbs.event().type event member represents the type of the event, for example, pbs.QUEUEJOB or pbs.MOVEJOB. See section 6-2, “Event Types and Objects”, on page 87
Message log event class objects	You can use these objects to indicate log event class when placing messages in the server logs. See section 6-16, “Message Log Level Objects”, on page 152.
Queue type objects	The queue("<queue name>").queue_type member represents the type of the queue. See section 6-5, “Queue Type Constant Objects”, on page 122.
PBS server state objects	The pbs.server().server_state member represents the state of the server. See section 6.4.1.3, “Server State Member”, on page 119.
Job state objects	The job.job_state member represents the job's state. Use these constant objects to test the state of a job. See section 6-6, “Job State Objects”, on page 125.
Reservation state objects	The resv.reserve_state member represents the state of the reservation. See section 6-7, “Reservation State Objects”, on page 133.

Table 6-12: Constant Objects

Category	Description
Vnode state objects	The vnode.state member represents the state of the vnode. See section 6-10, "Vnode State Constant Objects", on page 135.
Vnode sharing objects	The vnode.sharing member represents the vnode's sharing attribute. See section 6-9, "Vnode Sharing Objects", on page 135.
Vnode type objects	The vnode.ntype member represents the type of the vnode. See section 6-8, "Vnode Type Objects", on page 134.

6.13 Object Members and Methods

The relationships between objects and methods are shown in Figure 6-1 and Figure 6-2.

Event object members are listed in Table 6-3, "Using Event Object Members in Events," on page 108.

Non-event objects and object members are listed in Table 6-13, "PBS Objects and Object Members," on page 141.

Table 6-14, "Methods Available in Events," on page 142 shows the methods available for each kind of event.

Each global method is described in section 6.13.3, "PBS Types and Their Methods", on page 143.

Each event-only method is described in section 6.3.4, "Event-only Methods", on page 116.

Each object-only method is described in the section for its object.

6.13.1 PBS Objects and Object Members

The following table lists PBS objects and their members, such as the server or jobs:

Table 6-13: PBS Objects and Object Members

Object	Object Member	Object Sub-member
pbs.hook_config_filename		
pbs.job	job.id	
	job.<attribute name>	
	pbs.exec_vnode (job attribute)	pbs.exec_vnode.chunks[]
pbs.pbs_conf[]		
pbs.queue	queue.<attribute name>	
	queue.name	
pbs.resv	resv.<attribute name>	
	resv.resvid	
pbs.server	pbs.server().<attribute name>	
	pbs.server().name	
pbs.vnode	vnode.<attribute name>	
	pbs.vchunk	vchunk.chunk_resources[]
		vchunk.vnode_name

6.13.2 Methods Available in Events

The following table lists all methods, and shows which event can use each method. A "*y*" means that the hook can use the method, an "*n*" means it cannot, and an "*o*" means that it can but will have no effect:

Table 6-14: Methods Available in Events

Method	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
job.delete()	y	y	y	y	y	n	o	o	y	o	y	o	o	o	o	y	y	o	y	n
job.in_ms_mom()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
job.is_checkpointed()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
job.release_nodes()	n	n	n	n	n	n	n	o	n	n	y	y	n	n	n	n	n	n	n	n
job.rerun()	y	y	y	y	y	n	o	o	y	o	y	o	o	o	o	y	y	o	y	n
pbs.acl()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.args()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.checkpoint()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.depend()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.depend()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.duration()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.email_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.event().accept()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.event().reject()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.exec_host()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.exec_vnode()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.get_local_nodename()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.group_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.hold_types()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.job_sort_formula()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.join_path()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.keep_files()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.license_count()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.logjobmsg()	y	y	y	y	y	y	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.logmsg()	y	y	y	y	y	y	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.mail_points()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.node_group_key()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.path_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.pbs_env()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.place()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.range()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.reboot()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y
pbs.route_destinations()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.select()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.select.increment_chunks()	y	y	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
pbs.server().job('<job ID>')	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().jobs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n

Table 6-14: Methods Available in Events

Method	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_attach	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.server().queue("<queue_name>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().queues()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().resv("<reservation ID>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().resvs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().scheduler_restart_cycle()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().vnode("<vnode name>")	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.server().vnodes()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
pbs.size()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.software()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.staging_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.state_count()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.user_list()	y	y	y	y	y	n	o	o	y	o	y	o	y	y	y	y	y	o	y	n
pbs.version()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
queue.job()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
queue.jobs()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n
vchunk.chunk_resources.keys()	y	y	y	y	y	n	y	o	y	y	y	y	y	y	y	y	y	y	y	n

6.13.3 PBS Types and Their Methods

6.13.3.1 Method to Create or Set ACL

pbs.acl()

pbs.acl("[+|-]<entity>][,...]")

Creates an object representing a PBS ACL, from the specified formatted input string.

6.13.3.2 Method to Create or Set Command Argument List

pbs.args()

pbs.args("<args>")

where <*args*> are space-separated arguments to a command.

Creates an object representing the arguments to the command from the specified formatted input string <*args*>.

Example of setting a command argument list:

```
pbs.args("-Wsuppress_email=N -r y")
```

6.13.3.3 Method to Create or Set Checkpoint String

pbs.checkpoint()

pbs.checkpoint("<checkpoint_string>")

where <*checkpoint_string*> must be one of "*n*", "*s*", "*c*", "*c=mmm*", "*w*", or "*w=mmm*"

Creates an object representing the job Checkpoint attribute, using the specified formatted input string <*checkpoint_string*>.

6.13.3.4 Method to Create or Set Dependency Object

pbs.depend()

pbs.depend("<depend_string>")

<*depend_string*> must be of format *"<type>:<jobid>[,<jobid>...]"*, or *"on:<count>"*.

where <*type*> is one of "*after*", "*afterok*", "*afterany*", "*afternotok*", "*before*", "*beforeok*", "*beforeany*", and "*beforenotok*".

Creates a PBS dependency specification object representing the job depend attribute, using the given <*depend_string*>.

Usage:

pbs.event().job.depend = pbs.depend("<depend_string>")

6.13.3.5 Method to Create or Set Duration from Time String or Integer

pbs.duration()

pbs.duration("[[hours:]minutes:]seconds[.milliseconds]")

Creates a time specification duration instance, returning the equivalent number of seconds from the given time string. Represents an interval or elapsed time in number of seconds. Duration objects can be specified using either a time or an integer. See "Method to Create or Set Duration from Time String or Integer".

pbs.duration(<integer>)

Creates an integer duration instance using the specified number of seconds.

A pbs.duration instance can be operated on by any of the Python int functions. When performing arithmetic operations on a pbs.duration type, ensure the resulting value is a pbs.duration() type, before assigning to a job member that expects such a type.

Example:

```
pbs.event().job.Resource_List["cput"] = pbs.duration(300 + d1) # safe
```

The following will **not** work, since Python evaluates from left to right, and returns result as the type at left (int):

```
d1 = pbs.duration(30)
pbs.event().job.Resource_List["cput"] = 300  + d1
```

6.13.3.6 Method to Create or Set Email List

pbs.email_list()

pbs.email_list("<email_address1>[, <email address2>...]")

Creates an object representing a mail list from the specified formatted input string.

6.13.3.7 Method to Create or Set exec_host Object

pbs.exec_host()

*pbs.exec_host("host/N[*C][+...]")*

Create an object representing the exec_host job attribute, using the specified input string containing host and resource specification.

6.13.3.8 Method to Create or Set exec_vnode Object

pbs.exec_vnode()

pbs.exec_vnode("<vchunk>[+<vchunk> ...]")

<vchunk> is *(<vnodename:ncpus=N:mem=M>)*

Creates an object representing the exec_vnode job attribute, using the input string containing the vnode and resource specification. When the `qrun -H` command is used, or when the scheduler runs a job, the job.exec_vnode object contains the vnode specification for the job.

Example:

```
pbs.exec_vnode("(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z)")
```

This object is managed and accessed via the str() or repr() functions. Example:

```
Python> ev = pbs.server().job("10").exec_vnode
Python> str(ev)                "(vnodeA:ncpus=2:mem=200m)+(vnodeB:ncpus=5:mem=1g)"
```

6.13.3.9 Method to Create or Set group_list Object

pbs.group_list()

pbs.group_list("<group_name>[@<host>][,<group_name>[@<host>]...]")

Creates an object representing a PBS group list from the specified formatted input string.

To use a group list object:

```
job.group_list = pbs.group_list(....)
```

6.13.3.10 Method to Create or Set hold_types Object

pbs.hold_types()

pbs.hold_types("<hold_type_str>")

where *<hold_type_str>* is one of "*u*", "*o*", "*s*", or "*n*".

Creates an object representing the Hold_Types job attribute from the specified formatted input string.

6.13.3.11 Method to Create or Set job_sort_formula Object

pbs.job_sort_formula()

pbs.job_sort_formula("<formula string>")

where *<formula string>* is a string containing a math formula. See section 4.9.21, "Using a Formula for Computing Job Execution Priority", on page 149.

Creates an object representing the job_sort_formula server attribute from the specified formatted input string.

6.13.3.12 Method to Create or Set join_path Object

pbs.join_path()

pbs.join_path({"oe"|"eo"|"n"})

Creates an object representing the Join_Path job attribute from the specified formatted input string.

6.13.3.13 Method to Create or Set keep_files Object

pbs.keep_files()

pbs.keep_files("<Keep_Files option>")

where *<Keep_Files option>* is one of "n", "d", "o", "e", "oe", "eo".

Creates an object representing the Keep_Files job attribute from the specified formatted input string.

6.13.3.14 Method to Create or Set license_count Object

pbs.license_count()

pbs.license_count("Avail_Global:<value> Avail_Local:<value> Used:<value> High_Use:<value>")

Instantiates an object representing a license_count attribute from the specified formatted input string.

6.13.3.15 Method to Create or Set mail_points Object

pbs.mail_points()

pbs.mail_points("<mail points string>")

where *mail points string* is "*a*", "*b*", and/or "*e*", optionally with *"j"*, or "*n*".

Creates a pbs.mail_points object representing a Mail_Points attribute from the specified formatted input string.

6.13.3.16 Method to Create or Set node_group_key Object

pbs.node_group_key()

pbs.node_group_key("<resource(s)>")

Creates a pbs.node_group_key object representing the resource(s) to be used for node grouping, using the specified resource(s). The input string is a comma-separated, quoted list of resources.

6.13.3.17 Method to Create or Set path_list Object

pbs.path_list()

pbs.path_list("<path>[@<host>][,<path>@<host> ...]")

Creates an object representing a PBS pathname list from the specified formatted input string.

To use a path list object:

```
job.Shell_Path_List = pbs.path_list(....)
```

6.13.3.18 Method to Create or Set Job Environment Object

pbs.pbs_env()

Creates an empty environment variable list.

For example, to clear an environment variable list:

```
pbs.event().env = pbs.pbs_env()
```

6.13.3.19 Method to Create or Set Resource List

pbs.pbs_resource()

pbs.pbs_resource(<resource list name>)

Creates a pbs.pbs_resource object with the specified name.

To set values for a pbs.pbs_resource object:

<resource list name>['<resource name>']=<resource value>

For example:

```
Resource_List['ncpus']=8
Resource_List['mem']=pbs.size("10gb")
Resource_List['walltime']=pbs.duration('00:45:00')
```

A pbs.pbs_resource is similar to a dictionary, but you cannot use direct traversal. To loop through entries:

for r in <list name>.keys():

...

For example:

```
for r in <Resource_List>.keys():
    pbs.logmsg(pbs.LOG_DEBUG, "Resource_List[%s]=%s" % (r, Resource_List[r]))
```

which produces the following log message:

```
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[walltime]=00:45:00
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[mem]=10gb
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List[ncpus]=7
```

A str(<object of type pbs.pbs_resource>) produces output of the form:

<resource name>=<value>,<resource name>=<value>, ...

To do the equivalent of str():

```
pbs.logmsg(pbs.LOG_DEBUG, "Resource_List is %s (%s)" % (Resource_List, type(Resource_List)))
```

This produces the following log message:

```
03/08/2018 18:47:16;0006;pbs_python;Hook;pbs_python;Resource_List is
   mem=10gb,ncpus=7,walltime=00:45:00 (<class 'pbs.v1._base_types.pbs_resource'>)
```

6.13.3.20 Method to Create or Set place Object

pbs.place()

pbs.place("[arrangement]:[sharing]:[group]")

arrangement can be "*pack*", "*scatter*", "*free*", "*vscatter*"

sharing can be *"shared"*, *"excl", "exclhost"*

group can be of the form "*group=<resource>*"

[arrangement], *[sharing]*, and *[group]* can be given in any order or combination.

Creates a place object representing the job's place specification from the specified formatted input string.

Example:

```
pl = pbs.place("pack:excl")
s = repr(pl)            (or s = `pl`)
letter = pl[0]          (assigns 'p' to letter)
s = s + ":group=host"   (append to string)
pl = pbs.place(s)       (update original pl)
```

6.13.3.21 Method to Create or Set range Object

pbs.range()

pbs.range("<start>-<stop>:<step>")

Creates a PBS object representing a range of values from the specified formatted input string. Can be used to create a job.array_indices_submitted object. See section 6.6.1.2, "Job array_indices_submitted Attribute Member", on page 123.

Example:

```
pbs.range("1-30:3")
```

6.13.3.22 Method to Create or Set route_destinations Object

pbs.route_destinations()

pbs.route_destinations("<queue_spec>[,<queue_spec>,...]")

where *<queue_spec>* is *queue_name[@server_host[:port]]*

Creates an object that represents a route_destinations routing queue attribute from the specified formatted input string.

6.13.3.23 Method to Create or Set select Object

pbs.select()

pbs.select("[N:]res=val[:res=val]...[+[N:]res=val[:res=val] ...]")

Creates a select object representing the job's select specification from the specified formatted input string.

Example:

```
sel = pbs.select("2:ncpus=1:mem=5gb+3:ncpus=2:mem=5gb")
s = repr(sel)               (or s = `sel`)
letter = s[3]               (assigns 'c' to letter)
s = s + "+5:scratch=10gb"   (append to string)
sel = pbs.select(s)         (reset the value of sel)
```

6.13.3.24 Method to Increment select Object Chunks

pbs.select.increment_chunks()

pbs.select.increment_chunks(<increment specification>)

Creates a select object representing the job's new select specification, which has been padded from the original according to the *increment specification.*

You can pad all chunks, but you do not pad the primary vnode request itself; the job can only request one primary vnode. So when a job requests 3:ncpus=8+4:ncpus=1, the non-paddable primary vnode is considered to be a separate request of 1:ncpus=8, and the paddable part is the remaining 2:ncpus=8+4:ncpus=1.

:

Table 6-15: Behavior for increment specification

Value of increment specification	Behavior
Integer amount Format: can be with or without quotes (a number or a numeric string), e.g. 5 or "5"	Adds specified number of vnodes to each chunk in the job's vnode request. Examples: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks(2) returns: "1:ncpus=3:mem=1gb+3:ncpus=2:mem=2gb+4:ncpus=1:mem=3gb" Calling my_select.increment_chunks("3") returns: "1:ncpus=3:mem=1gb+4:ncpus=2:mem=2gb+5:ncpus=1:mem=3gb"
Percentage amount Format: a quoted numeric string ending in a per-cent sign, e.g. "10%"	Adds specified percent of vnodes to each chunk in the job's vnode request. Resulting amounts are rounded up. Example: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks("23.5%") returns: "1:ncpus=3:mem=1gb+2:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" The first chunk, which is a single chunk, is left as is, and the second and third chunks are increased by 23.5 %. 1.24 is rounded up to 2, and 2.47 is rounded up to 3.
Per-chunk specification Format: {<*chunk index*> : <*increment*>, ...} where the increment can be an integer or a per-centage	Adds specified amount or percent to specified chunk(s). Chunk index starts at *0*. Examples: Given this initial select statement: my_select=pbs.select("ncpus=3:mem=1gb+1:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks({0: 0, 1: 4, 2: "50%"}) returns: "1:ncpus=3:mem=1gb+5:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" There is no increase (0) for chunk 1, we give 4 additional chunks to chunk 2, and we increase chunk 3 by 50%, resulting in 3. Given this initial select statement: my_select=pbs.select("5:ncpus=3:mem=1gb+4:ncpus=2:mem=2gb+2:ncpus=1:mem=3gb") Calling my_select.increment_chunks("50%") or my_select.increment_chunks({0: "50%", 1: "50%", 2: "50%}) returns: "7:ncpus=3:mem=1gb+6:ncpus=2:mem=2gb+3:ncpus=1:mem=3gb" The primary vnode is broken out as "1:ncpus=3:mem=1gb" and is left as is. The "50%" increase is applied to the remaining portion, "4:ncpus=3:mem=1gb". After the increase is applied, the original first chunk is re-created from the primary vnode and the padded remains of the first chunk to make 7. Chunk 2 gets 6 and chunk 3 gets 3.

6.13.3.24.i Example of Padding Chunks

The following code snippet illustrates padding a job's vnode request by one extra vnode per chunk:

```
import pbs
e=pbs.event()
j = e.job
new_select = e.job.Resource_List["select"].increment_chunks(1)
e.job.Resource_List["select"] = new_select
```

6.13.3.25 Method to Create or Set size Object

pbs.size()

You can create a pbs.size object using either a byte count or a suffix:

pbs.size(<integer>)

Creates a PBS size object using integer byte count, storing the value as the number of bytes. Size objects can be specified using either an integer or a string. See the "pbs.size("<integer><suffix>")" creation method.

pbs.size("<integer><suffix>")

Creates a PBS size object using the specified suffix. The suffix must be a multiplier defined in the table shown in "Size" on page 360 of the PBS Professional Reference Guide. The size of a word is the word size on the execution host. Size objects can be specified using either an integer or a string.

To operate on pbs.size instances, use the "+" and "-" operators.

To compare pbs.size instances, use the "==", "!=", ">", "<", ">=", and "<=" operators.

Example: the sizes are normalized to the smaller of the 2 suffixes. In this case, "10gb" becomes "10240mb" and is added to "10mb":

```
sz = pbs.size("10gb")
sz = sz + 10mb
10250mb
```

Example: the following returns *True* because sz is greater than 100 bytes:

```
if  sz > 100:
    gt100 = True
```

6.13.3.26 Method to Create or Set Software Resource Object

pbs.software()

pbs.software("<software info string>")

Creates an object representing a site-dependent software resource from the specified formatted input string.

6.13.3.27 Method to Create or Set staging_list Object

pbs.staging_list()

pbs.staging_list("<filespec>[,<filespec>,...]")

where *<filespec>* is *<execution_path>@<storage_host>:<storage_path>*

Creates an object representing a job file staging parameters list from the specified formatted input string.

To use a staging list object:

```
job.stagein = pbs.staging_list(....)
```

6.13.3.28 Method to Create or Set state_count Object

pbs.state_count()

pbs.state_count("Transit:<U> Queued:<V> Held:<W> Running:<X> Exiting:<Y> Begun:<Z>")

Instantiates an object representing a state_count attribute from the specified formatted input string.

6.13.3.29 Method to Create or Set user_list Object

pbs.user_list()

pbs.user_list("<user>[@<host>][,<user>@<host>...]")

Creates an object representing a PBS user list from the specified formatted input string.

To use a user list object:

```
job.User_List = pbs.user_list(....)
```

6.13.3.30 Method to Create or Set PBS Version Object

pbs.version()

pbs.version("<pbs version string>")

Creates an object representing the PBS version string from the specified formatted input string.

6.13.4 Global Methods

6.13.4.1 Method to Get Local Vnode Name

pbs.get_local_nodename()

This returns a Python str whose value is the name of the local parent vnode.

If you want to refer to the vnode object representing the current host, you can pass this vnode name as the key to pbs.event().vnode_list[]. For example:

```
Vn = pbs.event().vnode_list[pbs.get_local_nodename()]
```

6.13.4.2 Method to Log Job-related String

pbs.logjobmsg()

pbs.logjobmsg(job ID, message)

where *job ID* must be an existing or previously existing job ID and where *message* is an arbitrary string.

This puts a custom string in the log of the PBS daemon running the hook, so if the method is being run by a server hook such as queuejob, it prints to the server log, but if the method is being run at an execution host hook such as execjob_prologue, it prints to the MoM log.

The `tracejob` command can be used to print out the job-related messages logged by a hook script.

Messages are logged at log event class pbs.LOG_DEBUG. See Table 6.13.4.4, “Message Log Level Objects,” on page 152.

6.13.4.3 Method to Log String

pbs.logmsg()

pbs.logmsg(log event class, message)

where *message* is an arbitrary string, and where *log event class* can be one of the message log event class constants shown in Table 6-16, “Message Log Level Objects,” on page 152.

This puts a custom string in the log of the PBS daemon running the hook, so if the method is being run by a server hook such as queuejob, it prints to the server log, but if the method is being run at an execution host hook such as execjob_prologue, it prints to the MoM log.

Example:

```
for j in pbs.server().jobs():
   pbs.logmsg(pbs.LOG_DEBUG, "found job %s" % (j.id))
```

6.13.4.4 Message Log Level Objects

You can use the following objects to indicate log level when placing messages in the server logs.

Table 6-16: Message Log Level Objects

Object	Decimal	Hex	PBS Log Event Filter	Name and Event Category
pbs. EVENT_ERROR	1	0x0001	error	PBSEVENT_ERROR Internal errors
pbs. EVENT_SYSTEM	2	0x0002	system	PBSEVENT_SYSTEM system errors
pbs. EVENT_ADMIN	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_WARNING	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_ERROR	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs.LOG_DEBUG	4	0x0004	admin	PBSEVENT_ADMIN Administrative events
pbs. EVENT_JOB	8	0x0008	job	PBSEVENT_JOB Job-related events
pbs. EVENT_JOB_USAGE	16	0x0010	job_usage	PBSEVENT_JOB_USAGE Job accounting info
pbs. EVENT_SECURITY	32	0x0020	Security	PBSEVENT_SECURITY Security violations
pbs. EVENT_SCHED	64	0x0040	sched	PBSEVENT_SCHED Scheduler events

Table 6-16: Message Log Level Objects

Object	Decimal	Hex	PBS Log Event Filter	Name and Event Category
pbs. EVENT_DEBUG	128	0x0080	debug	PBSEVENT_DEBUG Common debug messages
pbs. EVENT_DEBUG2	256	0x0100	debug2	PBSEVENT_DEBUG2 Uncommon debug messages
pbs. EVENT_RESV	512	0x0200	resv	PBSEVENT_RESV Reservation-related events
pbs. EVENT_DEBUG3	1024	0x0400	debug3	PBSEVENT_DEBUG3 Less common than PBSEVENT_DEBUG2
pbs. EVENT_DEBUG4	2048	0x0800	debug4	PBSEVENT_DEBUG4 Less common than debug3
pbs. EVENT_FORCE	4096	0x8000	(No filter applies)	PBSEVENT_FORCE Forces a message to be logged

6.13.4.5 Method to Reboot Host

pbs.reboot()

pbs.reboot([<command>])

This stops hook execution, so that remaining lines in the hook script are not executed, and starts the tasks that would normally begin after the hook is finished, such as flagging the current host to be rebooted. The MoM logs show the following:

```
<hook name> requested for host to be rebooted
```

We recommend that before calling pbs.reboot(), you set any vnodes managed by this MoM offline, and requeue the current job, if this hook is not an exechost_periodic hook. For example:

```
for  v in pbs.event().vnode_list.keys():
    pbs.event().vnode_list[v].state = pbs.ND_OFFLINE
    pbs.event().vnode_list[v].comment = "Rebooting host"
pbs.event().job.rerun()
pbs.reboot()
```

The effect of the call to pbs.reboot() is not instantaneous. The reboot happens after the hook executes, and after any of the other actions such as pbs.event().job.rerun(), pbs.event().delete(), and pbs.event().vnode_list[] take effect.

A hook with its user attribute set to *pbsuser* cannot successfully invoke pbs.reboot(), even if the owner is a PBS Manager or Operator. If this is attempted, the host is not rebooted, and the following message appears at log event class PBSEVENT_DEBUG2 in the MoM logs:

```
<hook_name>; Not allowed to issue reboot if run as user.
```

The <*command*> is an optional argument. It is a Python str which is executed instead of the reboot command that is the default for the system. For example:

```
pbs.reboot("/usr/local/bin/my_reboot -s 10 -c 'going down in 10'")
```

The specified <*command*> is executed in a shell on Linux/UNIX or via cmd on Windows.

7 Built-in Hooks

7.1 Managing Built-in Hooks

PBS comes shipped with built-in hooks that implement features or patch bugs. You can operate on these hooks via `qmgr`. The `qmgr` keyword for built-in hooks is "*pbshook*". These hooks are named with the "*PBS*" prefix.

7.2 Prerequisites

You can operate on built-in hooks only from an account that has root access to the PBS server host.

When operating on a built-in hook, use the keyword "*pbshook*", not "*hook*".

7.3 Allowed Operations

You can perform a limited set of operations on built-in hooks. You can do the following:

- View attributes
- Set all attributes except for type
- Edit configuration files
- Replace with your own hook

7.4 Viewing Built-in Hooks

You can view attributes of built-in hooks:

```
# qmgr -c "list pbshook"
Hook PBS_example_hook
    type = pbs
    enabled = false
    event = queuejob,resvsub
    user = pbsadmin
    alarm = 90
    order = 1000
```

7.5 Setting Attributes of Built-in Hooks

You can set all attributes except for the type attribute for a built-in hook. For example, you can enable and disable built-in hooks:

```
# qmgr -c "set pbshook <built-in hook name> enabled=true"
# qmgr -c "set pbshook <built-in hook name> enabled=false"
```

If you disable a built-in hook, the following message is printed to qmgr's STDERR:

```
"WARNING: Disabling a PBS hook results in an unsupported configuration!"
```

7.6 Editing and Importing Configuration Files for Built-in Hooks

You can edit and re-import a configuration file for a built-in hook. Get the contents of the configuration file by exporting the file:

```
#qmgr -c "export pbshook <hook name> application/x-config default" > config_file_save
```

Edit the file (here, config_file.save), then re-import it:

```
# qmgr -c "import pbshook <hook name> application/x-config <content-encoding> default
    config_file.save"
```

7.7 Restrictions

- You cannot create or delete a built-in hook. Attempting to do so results in the following error being printed to qmgr's STDERR:
  ```
  Invalid request
  ```
- You cannot import or export content of a built-in hook. Attempting to do so results in the following error being printed to qmgr's STDERR:
  ```
  <content-type> must be application/x-config
  ```
- You cannot display the commands to re-create a built-in hook: using `qmgr -c "print pbshook"` won't work.

7.8 Replacing a Built-in Hook with Your Own Hook

You can replace a built-in hook with your own hook. For example, to replace a built-in exechost_startup hook:

1. Disable the built-in hook:
   ```
   # qmgr -c "set pbshook <built-in startup hook> enabled=false"
   ```
2. Create your own site-defined hook instead:
   ```
   # qmgr -c "create hook <your startup hook> event=exechost_startup"
   # qmgr -c "import hook <your startup hook> application/x-python default <your startup script>
   ```

7.9 Errors and Logging when Operating on Built-in Hooks

- If you try to operate on a built-in hook from an account that does not have root or Admin access, the following error message is issued to STDERR:

  "unable to generate a hook_tempfile from <filepath> - Permission denied"

  <user>@<host> is unauthorized to access hooks data from server <hostname>"

- If you try to import or export a built-in hook, you will see one of the following messages on STDERR:

  # qmgr -c "import pbshook <hook name> application/x-python default my_hook.py"

  <content-type> must be application/x-config

  or

  #qmgr -c "export pbshook <hook name> application/x-python default"

  <content-type> must be application/x-config

8 Debugging Hooks

8.1 The pbs_python Hook Debugging Tool

You can use the pbs_python wrapper that is shipped with PBS to debug hooks. Either:

- Use the --hook option to pbs_python to run pbs_python as a wrapper to Python, employing the pbs_python options. With the --hook option, you cannot use the standard Python options. The rest of this section covers how to use pbs_python with the --hook option.
- Do not use the --hook option, so pbs_python runs the Python interpreter, with the standard Python options, and without access to the pbs_python options.

Usage for pbs_python:

pbs_python --hook [-e <log event mask>] [-i <event input_file>] [-L <log dir>] [-l <log file>] [-o <hook execution record>] [-r <resourcedef file>] [-s site data file] [<python script>]

For a complete description of pbs_python, see "pbs_python" on page 83 of the PBS Professional Reference Guide.

8.2 Files for Debugging

You can get each hook to write out debugging files, and then modify the files and use them as debugging input to pbs_python. Alternatively, you can write the files yourself.

Debugging files can contain information about the event, about the site, and about what the hook changed. You can use these as inputs to a hook when debugging.

8.2.1 Producing Files for Debugging

To get a hook to write out event and site debugging files, and a hook execution record, set its debug attribute to *True* (the default is *False*). The files are named *hook_<event type>_<hook name>_<random integer>.in*, *.data*, and *.out*. The <*random integer*> is the same for all output files for one run of a hook. The <*random integer*> is different for each run.

The hook writes these files:

- *Event file*, containing the values that populate the pbs.event() objects in the hook and any other top level pbs objects like pbs.get_local_nodename(), and job and job list information. This file always contains the event type. The file is named *hook_<event type>_<hook name>_<random integer>.in*. Can be passed to pbs_python using -i <event file> option. See section 8.2.5, "Event File", on page 160.
- *Site data file*, containing the values that populate the pbs.server() objects: server, queue, vnode, etc. information. The site data file is named *hook_<event type>_<hook name>_<random integer>.data*. Can be passed to pbs_python using -s <site data file> option. This file is populated only when the hook calls pbs.server(). See section 8.2.6, "Site Data File", on page 164.
- *Hook execution record*, listing whether the hook accepted or rejected the event, and whatever was changed by the hook, named *hook_<event type>_<hook name>_<random integer>.out*. See section 8.2.7, "Hook Execution Record File", on page 164.

So for example an execjob_begin hook named *BeginHook* will produce files, if PBS chooses "*15223*" as its random integer, named *hook_execjob_begin_BeginHook_15223.in*, *hook_execjob_begin_BeginHook_15223.data*, and *hook_execjob_begin_BeginHook_15223.out*.

8.2.2 Locations for Debugging Files

These files are written to these locations:

- Pre-execution hooks: PBS_HOME/server_priv/hooks/tmp
- All exechost_* and execjob_* hooks: PBS_HOME/mom_priv/hooks/tmp

8.2.3 Format for Debugging Files

File format for debugging files is text. For example:

```
pbs.event().job.Hold_Types=u
pbs.event().job.Job_Name=STDIN
pbs.event().job.Checkpoint=u
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.Mail_Points=a
pbs.event().job.Priority=0
pbs.event().job.Rerunable=TRUE
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=2gb
pbs.get_local_nodename()=mars.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=mars.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

8.2.4 Time Limit for Debugging Files

PBS deletes hook .in, .data, and .out files in PBS_HOME/*/hooks/tmp that are older than 20 minutes. If you need to keep any of these files, copy them to another location.

8.2.5 Event File

The event file must contain the event type, and can contain any relevant information about the triggering event, the current job, or list of jobs.

When the hook writes it, this file contains the values that populate the pbs.event() objects in the hook and any other top level pbs objects such as the local vnode, the job, and the list of jobs. When a hook writes this file, it includes pbs.event().type and the result of get_local_nodename(). Each kind of hook writes different additional pbs.event() information.

The file is named *hook_<event type>_<hook name>_<random integer>.in*. It can be passed to `pbs_python` using the `-i <event file>` option.

The following table shows which information is written to the event file by each kind of hook:

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
pbs.get_local_nodename()	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
alarm	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
argv[]											y								
env											y								
freq																		y	
hook_name	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
hook_type	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
job.Checkpoint									y	y	y	y	y	y	y	y			
job.egroup									y	y	y	y	y	y	y	y			
job.Error_Path									y	y	y	y	y	y	y	y			
job.euser									y	y	y	y	y	y	y	y			
job.exec_vnode									y	y	y	y	y	y	y	y			
job.Exit_Status												y	y	y					
job.id									y	y	y	y	y	y	y	y			
job.jobdir												y	y	y	y	y			
job.job_kill_delay									y	y	y	y	y	y	y	y			
job.Job_Name									y	y	y	y	y	y	y	y			
job.Job_Owner									y	y	y	y	y	y	y	y			
job.job_state										y	y	y	y	y	y	y			
job.Join_Path									y	y	y	y	y	y	y	y			
job.Keep_Files									y	y	y	y	y	y	y	y			
job.mtime									y	y	y	y	y	y	y	y			
job.Output_Path									y	y	y	y	y	y	y	y			
job.Priority										y	y	y	y	y	y	y			
job.project									y	y	y	y	y	y	y	y			
job.queue									y	y	y	y	y	y	y	y			
job.resources_used[cpupercent]												y	y	y	y	y			
job.resources_used[cput]												y	y	y	y	y			
job.resources_used[mem]												y	y	y	y	y			
job.resources_used[ncpus]												y	y	y	y	y			
job.resources_used[vmem]												y	y	y		y			
job.resources_used[walltime]												y	y	y	y	y			
job.Resource_List[file]									y	y	y	y	y	y	y	y			
job.Resource_List[ncpus]									y	y	y	y	y	y	y	y			
job.Resource_List[place]									y	y	y	y	y	y	y	y			
job.run_count									y	y	y	y	y	y	y	y			
job.run_version									y	y	y	y	y	y	y	y			
job.schedselect									y	y	y	y	y	y	y	y			

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
job.server									y	y	y	y	y	y	y	y			
job.session_id												y	y	y	y	y			
job.substate										y	y	y	y	y	y	y			
job.Variable_List									y	y	y	y	y	y	y	y			
job_list["<job ID>"].Checkpoint																		y	
job_list["<job ID>"].egroup																		y	
job_list["<job ID>"].Error_Path																		y	
job_list["<job ID>"].euser																		y	
job_list["<job ID>"].exec_vnode																		y	
job_list["<job ID>"].hashname																		y	
job_list["<job ID>"].jobdir																		y	
job_list["<job ID>"].job_kill_delay																		y	
job_list["<job ID>"].Job_Name																		y	
job_list["<job ID>"].Job_Owner																		y	
job_list["<job ID>"].job_state																		y	
job_list["<job ID>"].Join_Path																		y	
job_list["<job ID>"].Keep_Files																		y	
job_list["<job ID>"].mtime																		y	
job_list["<job ID>"].Output_Path																		y	
job_list["<job ID>"].project																		y	
job_list["<job ID>"].queue																		y	
job_list["<job ID>"].resources_used[cpupercent]																		y	
job_list["<job ID>"].resources_used[cput]																		y	
job_list["<job ID>"].resources_used[mem]																		y	
job_list["<job ID>"].resources_used[ncpus]																		y	
job_list["<job ID>"].resources_used[walltime]																		y	
job_list["<job ID>"].Resource_List[file]																		y	
job_list["<job ID>"].Resource_List[ncpus]																		y	
job_list["<job ID>"].Resource_List[place]																		y	
job_list["<job ID>"].run_count																		y	
job_list["<job ID>"].run_version																		y	
job_list["<job ID>"].schedselect																		y	
job_list["<job ID>"].server																		y	
job_list["<job ID>"].session_id																		y	
job_list["<job ID>"].substate																		y	
job_list["<job ID>"].Variable_List																		y	
job_list["<job ID>"]._msmom																		y	
job_list["<job ID>"]._stderr_file																		y	
job_list["<job ID>"]._stdout_file																		y	
progname											y								

Table 8-1: Event File by Hook

Event Information Written by Hooks: pbs.event.<list item>	queuejob	modifyjob (before run)	movejob	runjob (on reject)	runjob (on accept)	periodic	resvsub	resv_end	execjob_begin	execjob_prologue	execjob_launch	execjob_postsuspend	execjob_preresume	execjob_end	execjob_epilogue	execjob_preterm	exechost_startup	exechost_periodic	provision
requestor	y	y	y	y	y		y	y	y	y	y	y	y	y	y	y	y	y	
requestor_host	y	y	y	y	y		y	y	y	y	y	y	y	y	y	y	y	y	
resv.reserve_end							y	y											
resv.reserve_start							y	y											
resv.Variable_List							y	y											
type	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
user	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	y	
vnode_list["<local vnode name>"] .pbs_version									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .pcpus									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .resources_assigned[mem]									y	y	y	y	y	y	y	y			
vnode_list["<local vnode name>"] .resources_assigned[ncpus]									y	y	y	y	y	y	y	y			
vnode_list["<local vnode name>"] .resources_available[arch]									y	y	y	y	y	y	y	y		y	
vnode_list["<local vnode name>"] .resources_available[file]																		y	
vnode_list["<local vnode name>"] .resources_available[mem]									y	y	y	y	y	y	y	y	y	y	
vnode_list["<local vnode name>"] .resources_available[ncpus]									y		y	y	y	y	y	y	y	y	

For example, an event file created by a queuejob hook contains this data:

```
pbs.get_local_nodename()=jupiter.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

The equivalent command is:

```
Qmgr: list hook
```

8.2.5.1 Caveats

When the execjob_epilogue or execjob_end hook writes resources such as resources_used to the event file, it is writing about only the resources on the local host.

8.2.6 Site Data File

The site data file can contain any relevant information about the server, queues, vnodes, and jobs at the server.

This file is populated only when the hook calls pbs.server().

When the hook writes it, this file contains the values that populate the server, queues, vnodes, reservations, and jobs, with all attributes and resources for which there are values.

The site data file is named *hook_<event type>_<hook name>_<random integer>.data*. It can be passed to `pbs_python` using the `-s <site data file>` option.

The following commands give equivalent information:

```
qstat -Bf
qstat -Qf
qstat -f
pbsnodes -av
```

For example, here are some representative parts of a site data file:

```
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
...
ppbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
...
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
...
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
...
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
```

8.2.7 Hook Execution Record File

The hook execution record file is produced when the hook runs. This file lists the following:

- Whether the event was accepted or rejected
- Any job values that were changed by the hook, showing the new values

This file is named *hook_<event type>_<hook name>_<random integer>.out*.

8.3 Steps to Debug a Hook Using pbs_python

When you debug a hook using pbs_python, give it the following information:

- Use the --hook option to pbs_python so that you can use the other pbs_python options
- Specify event information by using -i <event file>. At a minimum, include the type of the event, but you can also include job and job list information. Information about the event can be one of these:
  - An event information file (.in) written by the hook
  - A file written by you
  - Interactive input

  See section 8.2.5, "Event File", on page 160.
- Optionally, provide site data. Site data includes data about the server, queues, vnodes, etc. You specify site data in a file by using -s <site data file name>. If you do not specify the -s option, pbs_python connects to the server and obtains live data about the site. See section 8.2.6, "Site Data File", on page 164. Site data can come from one of these sources:
  - A site data file (.data) written by the hook
  - A file written by you
  - Interactive input
  - Live data from the server
- If you have added any custom resources, specify the PBS_HOME/server_priv/resourcedef file with the -r option to pbs_python. Make sure you specify the whole path to the file. For example:

  ```
  pbs_python --hook -r $PBS_HOME/server_priv/resourcedef -i <input_file> <hook.py>
  ```
- If your hook uses a configuration file, set the environment variable PBS_HOOK_CONFIG_FILE to the file's pathname before calling pbs_python. See section 5.1.6, "Using Hook Configuration Files", on page 32.
- Run pbs_python on the hook:

  ```
  pbs_python --hook -s <site data> -i <event file> <hook script>
  ```

8.4 Caveats and Restrictions for pbs_python

- When you run a hook inside pbs_python, it has access to the extended set of PBS_EXEC/python modules listed in section 4.5, "Python Modules and PBS", on page 24. When you run pbs_python at the command line (without --hook), the hook does not have access to the PBS_EXEC/lib set of modules.
- If PBS has attempted to run a job multiple times in the 20 minute window, you may need to check the timestamp of hook debugging files (e.g. ls -lt) to figure out which files were produced during a particular hook run.
- The site data file is populated only when the hook calls pbs.server().

8.5 Examples of Using `pbs_python` to Debug Hooks

Example 8-1: Basic periodic hook, with updates to vnodes:

- Input file:

```
% cat hook.input
pbs.event().type=exechost_periodic
pbs.event().vnode_list["host1"].state=free
pbs.get_local_nodename()=host1
```

- Hook file:

```
$ cat test.py
import pbs

e = pbs.event()

pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["ncpus"]=7
pbs.event().vnode_list[pbs.get_local_nodename()].resources_available["mem"]=pbs.size("7gb")
```

- Run:

```
$ pbs_python --hook -i hook.input test.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["host1"].resources_available[ncpus]=7
pbs.event().vnode_list["host1"].resources_available[mem]=7gb
```

Example 8-2: A queuejob hook:

- Input file:

```
$ cat qjob.input
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().type=queuejob
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().alarm=40
pbs.event().job.Job_Name=pact
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[mem]=1mb
```

- Hook file:

```
$ cat qjob.py
import pbs

e = pbs.event()

e.job.Priority = 7
e.job.Account_Name = "mammoth"
e.job.Resource_List["ncpus"] = 5
```

```
e.job.Resource_List["mem"] = pbs.size("5gb")
```

- Run:

```
% pbs_python --hook -i qjob.input qjob.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().job.Priority=7
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=5gb
pbs.event().job.Account_Name=mammoth
```

Example 8-3: Reservation hook:

- Input file:

```
% cat rsub.input
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().type=resvsub
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().alarm=40
pbs.event().resv.Reserve_Name=my_resv
pbs.event().resv.Resource_List[ncpus]=1
pbs.event().resv.Resource_List[mem]=1mb
```

- Hook file:

```
% cat rsub.py
import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()
r = e.resv

print_attribs(r)

r.Resource_List["select"] = pbs.select("1:ncpus=1:mem=5mb")
r.Resource_List["place"] = pbs.place("pack:shared")

# group_list = pbs.group_list
r.group_list = pbs.group_list("Everyone,Everyone@host2,group1@jobim")

# Mail_Points= pbs.mail_points
r.Mail_Points = pbs.mail_points("a")
```

```
# User_List = pbs.user_list
r.User_List =  pbs.user_list("pbstest,pbstest@host2")

# Authorized_Users = pbs.acl
r.Authorized_Users =  pbs.acl("pbstest,user1,Administrator")

# Authorized_Groups = pbs.acl
r.Authorized_Groups =  pbs.acl("Everyone,group1,group2")
```

- Run:

```
% pbs_python  --hook -i rsub.input rsub.py
pbs.event().accept=True
pbs.event().reject=False
pbs.event().resv.group_list=Everyone,Everyone@host2,group1@jobim
pbs.event().resv.User_List=pbstest,pbstest@host2
pbs.event().resv.Resource_List[select]=1:ncpus=1:mem=5mb
pbs.event().resv.Resource_List[place]=pack:shared
pbs.event().resv.Mail_Points=a
pbs.event().resv.Authorized_Users=pbstest,user1,Administrator
pbs.event().resv.Authorized_Groups=Everyone,group1,group2
```

Example 8-4: A modifyjob hook:

- Hook script:

```
$ cat modifyjob.py

import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "------> printing job %s" % (e.job_o.id))
print_attribs(e.job_o)
e.job.Priority = 5
e.job.Resource_List["file"] = pbs.size("7gb")
e.job.Variable_List["FILE"] = "7gb"
```

- Use the `pbs_python` debugging tool.

Ensure you have the following input file:

```
% cat hook.input
pbs.event().type=modifyjob
pbs.event().job.id=0.host1
pbs.event().job.Variable_List=A=b
```

- Run the hook:

```
% pbs_python --hook -i hook.input modifyjob.py
```

- The following are printed:

```
pbs.event().accept=True
pbs.event().reject=False

pbs.event().job.Variable_List=A=b,FILE=7gb
pbs.event().job.Priority=5
pbs.event().job.Resource_List[file]=7gb
```

- The `pbs_python` log file shows this:

```
% cat <yyyymmdd>

01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;----------------> printing job 0.host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Error_Path =
host1.example.com:/home/user1/STDIN.e0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;job_state = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;schedselect =1:ncpus=1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;ctime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;egroup = pbs
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Variable_List =A=b,FILE=7gb
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;etime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;comment = Not Running:
Could not run job - nodes are not licensed or unable to obtain 1 cpu licenses. avail_licenses=0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;substate = 10
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue_rank = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;euser = user1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;queue_type = 1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Output_Path =
host1.example.com:/home/user1/STDIN.o0
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Hold_Types = n
```

```
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Join_Path = n
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;mtime = 1357387083
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;id = 0.host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Resource_List =
    select=1:ncpus=1,nodect=1,ncpus=1,place=free
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 07:50:41;0006;pbs_python;Hook;pbs_python;Job_Owner = user1@host1.example.com
```

Example 8-5: A movejob hook which prints job ID, src_queue, and movejob event parameters, and sets src_queue:

- Hook script:

```
$ cat movejob.py

import pbs

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing src_queue %s" % (e.src_queue.name))
print_attribs(e.src_queue)
pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing job %s" % (e.job.id))
print_attribs(e.job)
e.job.queue = pbs.server().queue("workq2")
```

- Use the pbs_python debugging tool:

```
Use the following input file:
% cat hook.input2
pbs.event().type=movejob
pbs.event().job.id=<existing-job-id>
```

where <existing-job-id> must be some arbitrary job currently existing in the queue *workq*. Submit one (`qsub -h`) if it doesn't exist.

- Run the hook:

  % pbs_python --hook -i hook.input2 movejob.py

- The following is printed:

  pbs.event().accept=True

  pbs.event().reject=False

  pbs.event().src_queue=workq2

- The `pbs_python` log file shows this:

  % cat <yyyymmdd>

```
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------> printing
src_queue workq
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;name = workq
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;----------------> printing
job 0.host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;qtime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Error_Path =
    host1.example.com:/home/user1/STDIN.e0
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;job_state = 1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;ctime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Variable_List =
PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/user1,PBS_O_HOST=host1.example.com,PBS
    _O_LOGNAME=user1,PBS_O_WORKDIR=/home/user1,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/opt/pbs/bin:/o
    pt/pbs/python/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/opt/pbs/bin:/opt/pbs/python/bin:/opt/p
    bs/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/user1/bin:/us
    r/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/base
    /cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/user1/bin:/home/user1/bin
    :/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/local/purify/
    base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin,PBS_O_QUEUE=workq,PBS_O_MAI
    L=/var/spool/mail/user1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;etime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;comment = Not Running:
Could not run job - nodes are not licensed or unable to obtain 1 cpu licenses. avail_licenses=0
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;substate = 10
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Output_Path =
host1.example.com:/home/user1/STDIN.o0
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Hold_Types = n
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Join_Path = n
```

```
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;mtime = 1357387083
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;id = 0.host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Resource_List =
select=1:ncpus=1,nodect=1,ncpus=1,place=free
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 11:23:36;0006;pbs_python;Hook;pbs_python;Job_Owner =
user1@host1.example.com
```

Example 8-6: A runjob hook to print attributes:

- Hook script:

```
$ cat runjob.py

import pbs
import time

def print_attribs(pbs_obj):
    for a in pbs_obj.attributes:
        v = getattr(pbs_obj, a)
        if v and str(v) != "":
            pbs.logmsg(pbs.LOG_DEBUG, "%s = %s" % (a,v))

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "----------------> printing job %s" % (e.job.id))
print_attribs(e.job)
e.job.Hold_Types = pbs.hold_types("us")
e.job.Execution_Time = time.mktime([15, 11, 28, 14, 10, 15, -1, -1, 01])
e.job.project="looper"

pbs.event().reject("not allowed to run at this time!")
```

- Use the following input file:

```
% cat hook.input3
pbs.event().type=runjob
pbs.event().job.id=<existing-job-id>
```

where <*existing-job-id*> must be some arbitrary job currently existing in the server. Submit one (`qsub -h`) if it doesn't exist.

- Run the hook:

  `# pbs_python --hook -i hook.input3 runjob.py`

- The execution record contains the following:

```
pbs.event().reject=True
pbs.event().accept=False
pbs.event().reject_msg=not allowed to run at this time!
pbs.event().job.Execution_Time=1448745015
pbs.event().job.Hold_Types=us
pbs.event().job.project=looper
```

- The `pbs_python` log file shows:

```
% cat <yyyymmdd>

01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;----------------> printing
job 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;qtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Error_Path =
host1.example.com:/home/user1/bugs/sp260361/STDIN.e5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;job_state = 2
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;ctime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Rerunable = 1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Variable_List =
PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/user1,PBS_O_HOST=host1.example.com,PBS
    _O_LOGNAME=user1,PBS_O_WORKDIR=/home/user1/bugs/sp260361,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/
    opt/pbs/bin:/opt/pbs/python/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/opt/pbs/bin:/opt/pbs/pyt
    hon/bin:/opt/pbs/tcltk/bin:/home/user1/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home
    /user1/bin:/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr/loc
    al/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin:/home/user1/bin:/
    home/user1/bin:/usr/local/rational/releases/purify.i386_linux2.2003a.06.15.FixPack.0194:/usr
    /local/purify/base/cots/flexlm.10.8.0.1/i386_linux2:/home/user1/PbsTestLab/bin,PBS_O_QUEUE=w
    orkq,PBS_O_MAIL=/var/spool/mail/user1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Checkpoint = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Submit_arguments =
<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:Argument>
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;queue = workq
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Name = STDIN
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;substate = 20
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Mail_Points = a
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Priority = 0
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;project = _pbs_project_default
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Output_Path =
host1.example.com:/home/user1/bugs/sp260361/STDIN.o5
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Hold_Types = u
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Join_Path = n
```

```
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;mtime = 1357424154
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;id = 5.host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Resource_List =
select=1:ncpus=1,nodect=1,ncpus=1,place=pack
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Keep_Files = n
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;_connect_server = host1
01/05/2013 14:49:39;0006;pbs_python;Hook;pbs_python;Job_Owner =
user1@host1.example.com
```

8.6 Using Log Messages to Debug Hook Scripts

The following steps may help you avoid errors in hook scripts:

1. Create a hook, and import its content.
2. Temporarily set the server's log_events to a higher value such as *2047* to see plenty of logging.
3. Do a test run of the hook script, by causing events (e.g. `qsub`, `qalter`, `qmove`, `pbs_rsub`) that invoke the hook script. Check for error messages in the server logs.
4. Correct the hook script, re-import the fixed code, and rerun the test.
5. Once the hook script is running fine, then set the server's log_events back to the default (i.e. *511*).

8.7 Checking Hook Syntax using Python

You can check hook syntax using Python. If you run Python on the hook, the hook cannot import the pbs module. If the first error you see is a failure to import the pbs module, Python did not find any syntax errors.

8.8 Examples of Debugging Files

Example 8-7: We show several hooks and their debugging files. Our example hooks are queuejob, exechost_startup, exechost_periodic, execjob_begin, and execjob_launch.

Given the following two jobs in the system:

```
TestUser@jupiter:~/jobs> qstat
Job id            Name             User              Time Use S Queue
----------------  ---------------- ----------------  -------- - -----
501.jupiter          STDIN            TestUser                 0 H workq
502.jupiter          STDIN            TestUser                 0 H workq
```

Given the following reservations:

```
TestUser@jupiter:~/jobs> pbs_rstat
Resv ID    Queue    User     State             Start / Duration / End
--------------------------------------------------------------------
R503.jupiter. R503     TestUser@ CO          Today 08:00 / 1800 / Today 08:30
R504.jupiter. R504     TestUser@ CO          Today 09:00 / 1800 / Today 09:30
```

Given the following set of vnodes:

```
TestUser@jupiter:~/jobs> pbsnodes -av
jupiter
    Mom = jupiter.example.com
    Port = 15002
    pbs_version = PBSPro_10.0
    ntype = PBS
    state = free
    pcpus = 1
    resv = R504.jupiter.example.com, R503.jupiter.example.com
    resources_available.arch = linux
    resources_available.host = jupiter
    resources_available.mem = 8gb
    resources_available.ncpus = 8
    resources_available.vnode = jupiter
    resources_assigned.accelerator_memory = 0kb
    resources_assigned.mem = 0kb
    resources_assigned.naccelerators = 0
    resources_assigned.ncpus = 0
    resources_assigned.vmem = 0kb
    resv_enable = True
    sharing = default_shared
mars
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_10.0
    ntype = PBS
    state = free
    pcpus = 1
    resources_available.arch = linux
    resources_available.host = mars
    resources_available.mem = 8gb
    resources_available.ncpus = 8
    resources_available.vnode = mars
    resources_assigned.accelerator_memory = 0kb
    resources_assigned.mem = 0kb
    resources_assigned.naccelerators = 0
    resources_assigned.ncpus = 0
    resources_assigned.vmem = 0kb
    resv_enable = True
    sharing = default_shared
```

queuejob hook attributes:

```
Hook qjob
    type = site
    enabled = true
    event = queuejob
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

queuejob hook contents:

```
import pbs
e=pbs.event()

e.job.Priority=7
e.job.Resource_List["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Submit the job:

```
% qsub job.scr
```

Here are the resulting *.in, *.data, and *.out files:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr /var/spool/PBS/server_priv/hooks/tmp
-rw-r--r-- 1 root root   241 Sep 17 03:54 hook_queuejob_qjob_1410940476.in
-rw-r--r-- 1 root root 18619 Sep 17 03:54 hook_queuejob_qjob_1410940476.data
-rw-r--r-- 1 root root   805 Sep 17 03:54 hook_queuejob_qjob_1410940476.out
```

List the queuejob hook event file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.in
pbs.get_local_nodename()=jupiter.example.com
pbs.event().type=queuejob
pbs.event().hook_name=qjob
pbs.event().hook_type=site
pbs.event().requestor=TestUser
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the queuejob hook site data file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
pbs.server().resources_assigned[nodect]=0
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().FLicenses=32
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:32 Used:0 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
```

```
pbs.server().job(501.jupiter.example.com).Priority=7
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).hop_count=1
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
pbs.server().queue(workq).resources_assigned[nodect]=0
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
```

```
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/STDIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).hop_count=1
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:Argument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
```

```
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=0
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).hop_count=1
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
```

```
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).hop_count=1
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the queuejob hook execution record file:

```
jupiter:/var/spool/PBS/server_priv/hooks/tmp # cat hook_queuejob_qjob_1410940476.out
pbs.event().job.Rerunable=1
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser
    ,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=
    /usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/b
    in:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs
    /bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.event().job.Checkpoint=u
pbs.event().job.Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hpcpa:Argument>
pbs.event().job.Job_Name=job.scr
pbs.event().job.Mail_Points=a
pbs.event().job.Priority=7
pbs.event().job.Hold_Types=n
pbs.event().job.Join_Path=n
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Keep_Files=n
```

The exechost_startup hook attributes:

```
Hook start
    type = site
    enabled = true
    event = exechost_startup
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

The exechost_startup hook contents:

```
import pbs
e=pbs.event()


e.vnode_list[pbs.get_local_nodename()].resources_available["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Restart pbs_mom. Upon startup, the exechost_startup hook writes the following files:

```
jupiter:/home/TestUser/jobs # cd /var/spool/PBS/mom_priv/hooks/tmp
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
total 24
-rw-r--r-- 1 root root   455 Sep 17 04:02 hook_exechost_startup_start_11607.in
-rw-r--r-- 1 root root   115 Sep 17 04:02 hook_exechost_startup_start_11607.out
-rw-r--r-- 1 root root 12389 Sep 17 04:02 hook_exechost_startup_start_11607.data
```

List the exechost_startup hook event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.in
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.get_local_nodename()=jupiter
pbs.event().type=exechost_startup
pbs.event().hook_name=start
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the exechost_startup hook site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=2
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=0
pbs.server().resources_assigned[nodect]=0
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().FLicenses=32
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:32 Used:0 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
```

```
pbs.server().job(501.jupiter.example.com).Priority=7
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=2
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=0
pbs.server().queue(workq).resources_assigned[nodect]=0
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
```

```
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=0
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
```

```
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the exechost_startup hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_startup_start_11607.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["jupiter"].resources_available[file,size]=7gb
```

The exechost_periodic hook attributes:

```
Hook period
    type = site
    enabled = true
    event = exechost_periodic
    user = pbsadmin
    alarm = 30
    freq = 30
    order = 1
    debug = true
    fail_action = none
```

The contents of the exechost_periodic hook:

```
jupiter:/home/TestUser/jobs # qmgr -c "e h period application/x-python default"
import pbs
e=pbs.event()

e.vnode_list[pbs.get_local_nodename()].resources_available["file"] = pbs.size("7gb")

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

In our example, we have two jobs running on the execution host:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # qstat
Job id         Name        User        Time Use  S Queue
------------   ----------- ----------- --------  - -----
501.jupiter       STDIN       TestUser      0        H workq
502.jupiter       STDIN       TestUser      0        H workq
506.jupiter       STDIN       TestUser      00:00:00 R workq
507.jupiter       STDIN       TestUser      00:00:00 R workq
```

The *.in, *.out, and *.data files end up here:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
-rw-r--r-- 1 root root  6885 Sep 17 04:09 hook_exechost_periodic_period_11753.in
-rw-r--r-- 1 root root  1387 Sep 17 04:09 hook_exechost_periodic_period_11753.out
-rw-r--r-- 1 root root 19039 Sep 17 04:09 hook_exechost_periodic_period_11753.data
```

List the exechost_periodic event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.in
pbs.event().freq=30
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.event().vnode_list["jupiter"].resources_available[file]=7gb
pbs.event().job_list["506.jupiter.example.com"].Job_Name=STDIN
pbs.event().job_list["506.jupiter.example.com"].Job_Owner=TestUser@jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].resources_used[cpupercent]=0
pbs.event().job_list["506.jupiter.example.com"].resources_used[cput]=00:00:00
pbs.event().job_list["506.jupiter.example.com"].resources_used[mem]=3880kb
pbs.event().job_list["506.jupiter.example.com"].resources_used[ncpus]=1
pbs.event().job_list["506.jupiter.example.com"].resources_used[vmem]=32192kb
pbs.event().job_list["506.jupiter.example.com"].resources_used[walltime]=00:00:13
pbs.event().job_list["506.jupiter.example.com"].job_state=T
pbs.event().job_list["506.jupiter.example.com"].queue=workq
pbs.event().job_list["506.jupiter.example.com"].server=jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].Checkpoint=u
pbs.event().job_list["506.jupiter.example.com"].Error_Path=jupiter.example.com:/home/TestUser/jo
    bs/STDIN.e506
pbs.event().job_list["506.jupiter.example.com"].exec_host2=jupiter.example.com:15002/0
pbs.event().job_list["506.jupiter.example.com"].exec_vnode=(jupiter:ncpus=1)
pbs.event().job_list["506.jupiter.example.com"].Join_Path=n
pbs.event().job_list["506.jupiter.example.com"].Keep_Files=n
pbs.event().job_list["506.jupiter.example.com"].mtime=1410941347
pbs.event().job_list["506.jupiter.example.com"].Output_Path=jupiter.example.com:/home/TestUser/j
    obs/STDIN.o506
pbs.event().job_list["506.jupiter.example.com"].Resource_List[file]=7gb
pbs.event().job_list["506.jupiter.example.com"].Resource_List[ncpus]=1
pbs.event().job_list["506.jupiter.example.com"].Resource_List[place]=pack
pbs.event().job_list["506.jupiter.example.com"].schedselect=1:ncpus=1
pbs.event().job_list["506.jupiter.example.com"].session_id=11683
pbs.event().job_list["506.jupiter.example.com"].jobdir=/home/TestUser
pbs.event().job_list["506.jupiter.example.com"].substate=0
pbs.event().job_list["506.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PB
    S_O_LANG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/
    gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:
    /usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_
    QUEUE=workq,PBS_O_HOST=jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].euser=TestUser
pbs.event().job_list["506.jupiter.example.com"].egroup=users
pbs.event().job_list["506.jupiter.example.com"].hashname=506.jupiter.example.com
pbs.event().job_list["506.jupiter.example.com"].cookie=000000002CEAFC4E0000000043354104
```

```
pbs.event().job_list["506.jupiter.example.com"].run_count=1
pbs.event().job_list["506.jupiter.example.com"].job_kill_delay=10
pbs.event().job_list["506.jupiter.example.com"].project=_pbs_project_default
pbs.event().job_list["506.jupiter.example.com"].run_version=1
pbs.event().job_list["506.jupiter.example.com"]._msmom=True
pbs.event().job_list["506.jupiter.example.com"]._stdout_file=/var/spool/PBS/spool/506.jupiter.example.com.OU
pbs.event().job_list["506.jupiter.example.com"]._stderr_file=/var/spool/PBS/spool/506.jupiter.example.com.ER
pbs.event().job_list["507.jupiter.example.com"].Job_Name=STDIN
pbs.event().job_list["507.jupiter.example.com"].Job_Owner=TestUser@jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].resources_used[cpupercent]=0
pbs.event().job_list["507.jupiter.example.com"].resources_used[cput]=00:00:00
pbs.event().job_list["507.jupiter.example.com"].resources_used[mem]=3892kb
pbs.event().job_list["507.jupiter.example.com"].resources_used[ncpus]=1
pbs.event().job_list["507.jupiter.example.com"].resources_used[vmem]=32192kb
pbs.event().job_list["507.jupiter.example.com"].resources_used[walltime]=00:00:10
pbs.event().job_list["507.jupiter.example.com"].job_state=T
pbs.event().job_list["507.jupiter.example.com"].queue=workq
pbs.event().job_list["507.jupiter.example.com"].server=jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].Checkpoint=u
pbs.event().job_list["507.jupiter.example.com"].Error_Path=jupiter.example.com:/home/TestUser/jobs/STDIN.e507
pbs.event().job_list["507.jupiter.example.com"].exec_host2=jupiter.example.com:15002/1
pbs.event().job_list["507.jupiter.example.com"].exec_vnode=(jupiter:ncpus=1)
pbs.event().job_list["507.jupiter.example.com"].Join_Path=n
pbs.event().job_list["507.jupiter.example.com"].Keep_Files=n
pbs.event().job_list["507.jupiter.example.com"].mtime=1410941350
pbs.event().job_list["507.jupiter.example.com"].Output_Path=jupiter.example.com:/home/TestUser/jobs/STDIN.o507
pbs.event().job_list["507.jupiter.example.com"].Resource_List[file]=7gb
pbs.event().job_list["507.jupiter.example.com"].Resource_List[ncpus]=1
pbs.event().job_list["507.jupiter.example.com"].Resource_List[place]=pack
pbs.event().job_list["507.jupiter.example.com"].schedselect=1:ncpus=1
pbs.event().job_list["507.jupiter.example.com"].session_id=11716
pbs.event().job_list["507.jupiter.example.com"].jobdir=/home/TestUser
pbs.event().job_list["507.jupiter.example.com"].substate=0
pbs.event().job_list["507.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=workq,PBS_O_HOST=jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].euser=TestUser
pbs.event().job_list["507.jupiter.example.com"].egroup=users
pbs.event().job_list["507.jupiter.example.com"].hashname=507.jupiter.example.com
pbs.event().job_list["507.jupiter.example.com"].cookie=000000003C3AB5AC000000007A31CFD4
```

```
pbs.event().job_list["507.jupiter.example.com"].run_count=1
pbs.event().job_list["507.jupiter.example.com"].job_kill_delay=10
pbs.event().job_list["507.jupiter.example.com"].project=_pbs_project_default
pbs.event().job_list["507.jupiter.example.com"].run_version=1
pbs.event().job_list["507.jupiter.example.com"]._msmom=True
pbs.event().job_list["507.jupiter.example.com"]._stdout_file=/var/spool/PBS/spool/507.jupiter.ex
    ample.com.OU
pbs.event().job_list["507.jupiter.example.com"]._stderr_file=/var/spool/PBS/spool/507.jupiter.ex
    ample.com.ER
pbs.get_local_nodename()=jupiter
pbs.event().type=exechost_periodic
pbs.event().hook_name=period
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the exechost_periodic site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=4
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:2 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=2
pbs.server().resources_assigned[nodect]=2
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().FLicenses=30
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:30 Used:2 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
```

```
pbs.server().job(501.jupiter.example.com).Priority=7
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(506.jupiter.example.com).Job_Name=STDIN
pbs.server().job(506.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(506.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(506.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(506.jupiter.example.com).resources_used[mem]=3880kb
pbs.server().job(506.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(506.jupiter.example.com).resources_used[vmem]=32192kb
pbs.server().job(506.jupiter.example.com).resources_used[walltime]=00:00:13
pbs.server().job(506.jupiter.example.com).job_state=R
pbs.server().job(506.jupiter.example.com).queue=workq
pbs.server().job(506.jupiter.example.com).server=jupiter.example.com
pbs.server().job(506.jupiter.example.com).Checkpoint=u
pbs.server().job(506.jupiter.example.com).ctime=1410941347
pbs.server().job(506.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e506
pbs.server().job(506.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(506.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(506.jupiter.example.com).Hold_Types=n
pbs.server().job(506.jupiter.example.com).Join_Path=n
pbs.server().job(506.jupiter.example.com).Keep_Files=n
pbs.server().job(506.jupiter.example.com).Mail_Points=a
pbs.server().job(506.jupiter.example.com).mtime=1410941347
pbs.server().job(506.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o506
pbs.server().job(506.jupiter.example.com).Priority=7
pbs.server().job(506.jupiter.example.com).qtime=1410941347
pbs.server().job(506.jupiter.example.com).Rerunable=True
pbs.server().job(506.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(506.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(506.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(506.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(506.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(506.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(506.jupiter.example.com).stime=1410941347
pbs.server().job(506.jupiter.example.com).session_id=11683
pbs.server().job(506.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(506.jupiter.example.com).substate=42
pbs.server().job(506.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(506.jupiter.example.com).euser=TestUser
pbs.server().job(506.jupiter.example.com).egroup=users
pbs.server().job(506.jupiter.example.com).hashname=506.jupiter.example.com
pbs.server().job(506.jupiter.example.com).queue_rank=188
pbs.server().job(506.jupiter.example.com).queue_type=E
pbs.server().job(506.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:09 on
    (jupiter:ncpus=1)
pbs.server().job(506.jupiter.example.com).etime=1410941347
pbs.server().job(506.jupiter.example.com).run_count=1
pbs.server().job(506.jupiter.example.com).project=_pbs_project_default
pbs.server().job(506.jupiter.example.com).run_version=1
pbs.server().job(507.jupiter.example.com).Job_Name=STDIN
pbs.server().job(507.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(507.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(507.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(507.jupiter.example.com).resources_used[mem]=3892kb
pbs.server().job(507.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(507.jupiter.example.com).resources_used[vmem]=32192kb
pbs.server().job(507.jupiter.example.com).resources_used[walltime]=00:00:10
pbs.server().job(507.jupiter.example.com).job_state=R
pbs.server().job(507.jupiter.example.com).queue=workq
pbs.server().job(507.jupiter.example.com).server=jupiter.example.com
pbs.server().job(507.jupiter.example.com).Checkpoint=u
pbs.server().job(507.jupiter.example.com).ctime=1410941350
pbs.server().job(507.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e507
pbs.server().job(507.jupiter.example.com).exec_host=jupiter/1
pbs.server().job(507.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(507.jupiter.example.com).Hold_Types=n
pbs.server().job(507.jupiter.example.com).Join_Path=n
pbs.server().job(507.jupiter.example.com).Keep_Files=n
pbs.server().job(507.jupiter.example.com).Mail_Points=a
pbs.server().job(507.jupiter.example.com).mtime=1410941350
pbs.server().job(507.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o507
pbs.server().job(507.jupiter.example.com).Priority=7
```

```
pbs.server().job(507.jupiter.example.com).qtime=1410941350
pbs.server().job(507.jupiter.example.com).Rerunable=True
pbs.server().job(507.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(507.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(507.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(507.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(507.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(507.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(507.jupiter.example.com).stime=1410941350
pbs.server().job(507.jupiter.example.com).session_id=11716
pbs.server().job(507.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(507.jupiter.example.com).substate=42
pbs.server().job(507.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(507.jupiter.example.com).euser=TestUser
pbs.server().job(507.jupiter.example.com).egroup=users
pbs.server().job(507.jupiter.example.com).hashname=507.jupiter.example.com
pbs.server().job(507.jupiter.example.com).queue_rank=189
pbs.server().job(507.jupiter.example.com).queue_type=E
pbs.server().job(507.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:09 on
    (jupiter:ncpus=1)
pbs.server().job(507.jupiter.example.com).etime=1410941350
pbs.server().job(507.jupiter.example.com).run_count=1
pbs.server().job(507.jupiter.example.com).project=_pbs_project_default
pbs.server().job(507.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=4
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:2 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=2
pbs.server().queue(workq).resources_assigned[nodect]=2
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
```

```
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=506.jupiter.example.com/0, 507.jupiter.example.com/1
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=2
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
```

```
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the exechost_periodic hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_exechost_periodic_period_11753.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().vnode_list["jupiter"].resources_available[file,size]=7gb
pbs.event().job_list["506.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O
    _WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/
    mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/ope
    nmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/ga
    mes:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job_list["506.jupiter.example.com"]._delete=False
pbs.event().job_list["506.jupiter.example.com"]._rerun=False
pbs.event().job_list["507.jupiter.example.com"].Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bi
    n/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O
    _WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/
    mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/ope
    nmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/ga
    mes:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job_list["507.jupiter.example.com"]._delete=False
pbs.event().job_list["507.jupiter.example.com"]._rerun=False
```

Attributes of the execjob_begin hook:

```
Hook begin
    type = site
    enabled = true
    event = execjob_begin
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

Contents of the execjob_begin hook:

```
import pbs
e=pbs.event()

e.job.Priority=7
e.job.Variable_List["Monsieur"] = "Shlomi"

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

We submit a job:

```
% qsub job.scr
```

The resulting execjob_begin debug files are here:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # ls -ltr
-rw-r--r-- 1 root root  2263 Sep 17 04:15 hook_execjob_begin_begin_11883.in
-rw-r--r-- 1 root root   585 Sep 17 04:15 hook_execjob_begin_begin_11883.out
-rw-r--r-- 1 root root 15327 Sep 17 04:15 hook_execjob_begin_begin_11883.data
```

List the execjob_begin event file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_execjob_begin_begin_11883.in
pbs.event().job.id=509.jupiter.example.com
pbs.event().job.Job_Name=job.scr
pbs.event().job.Job_Owner=TestUser@jupiter.example.com
pbs.event().job.queue=workq
pbs.event().job.server=jupiter.example.com
pbs.event().job.Checkpoint=u
pbs.event().job.Error_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.e509
pbs.event().job.exec_host2=jupiter.example.com:15002/0
pbs.event().job.exec_vnode=(jupiter:ncpus=1)
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.mtime=1410941704
pbs.event().job.Output_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.o509
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[place]=pack
pbs.event().job.schedselect=1:ncpus=1
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,PBS_O_HOME=/home/TestUser
    ,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_PATH=
    /usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/b
    in:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs
    /bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=workq,PBS_O_HOST=jupiter.e
    xample.com
pbs.event().job.euser=TestUser
pbs.event().job.egroup=users
pbs.event().job.hashname=509.jupiter.example.com
pbs.event().job.run_count=1
pbs.event().job.job_kill_delay=10
pbs.event().job.project=_pbs_project_default
pbs.event().job.run_version=1
pbs.event().job._msmom=True
pbs.event().job._stdout_file=
pbs.event().job._stderr_file=
pbs.event().vnode_list["jupiter"].resources_assigned[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_assigned[mem]=0kb
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.get_local_nodename()=jupiter
pbs.event().type=execjob_begin
pbs.event().hook_name=begin
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
```

```
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the execjob_begin site data file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat  hook_execjob_begin_begin_11883.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=3
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=1
pbs.server().resources_assigned[nodect]=1
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().FLicenses=31
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:31 Used:1 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
```

```
pbs.server().job(501.jupiter.example.com).Priority=7
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(509.jupiter.example.com).Job_Name=job.scr
pbs.server().job(509.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(509.jupiter.example.com).job_state=R
pbs.server().job(509.jupiter.example.com).queue=workq
pbs.server().job(509.jupiter.example.com).server=jupiter.example.com
pbs.server().job(509.jupiter.example.com).Checkpoint=u
pbs.server().job(509.jupiter.example.com).ctime=1410941704
pbs.server().job(509.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/job
    .scr.e509
pbs.server().job(509.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(509.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(509.jupiter.example.com).Hold_Types=n
pbs.server().job(509.jupiter.example.com).Join_Path=n
pbs.server().job(509.jupiter.example.com).Keep_Files=n
pbs.server().job(509.jupiter.example.com).Mail_Points=a
pbs.server().job(509.jupiter.example.com).mtime=1410941704
pbs.server().job(509.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/jo
    b.scr.o509
pbs.server().job(509.jupiter.example.com).Priority=7
pbs.server().job(509.jupiter.example.com).qtime=1410941704
pbs.server().job(509.jupiter.example.com).Rerunable=True
pbs.server().job(509.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(509.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(509.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(509.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(509.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(509.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(509.jupiter.example.com).substate=41
pbs.server().job(509.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
```

```
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(509.jupiter.example.com).euser=TestUser
pbs.server().job(509.jupiter.example.com).egroup=users
pbs.server().job(509.jupiter.example.com).hashname=509.jupiter.example.com
pbs.server().job(509.jupiter.example.com).queue_rank=190
pbs.server().job(509.jupiter.example.com).queue_type=E
pbs.server().job(509.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:15 on
    (jupiter:ncpus=1)
pbs.server().job(509.jupiter.example.com).etime=1410941704
pbs.server().job(509.jupiter.example.com).run_count=1
pbs.server().job(509.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hp
    cpa:Argument>
pbs.server().job(509.jupiter.example.com).project=_pbs_project_default
pbs.server().job(509.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=3
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=1
pbs.server().queue(workq).resources_assigned[nodect]=1
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
```

```
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=509.jupiter.example.com/0
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=1
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
```

```
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the execjob_begin hook execution record file:

```
jupiter:/var/spool/PBS/mom_priv/hooks/tmp # cat hook_execjob_begin_begin_11883.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOM
    E=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/T
    estUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PB
    S_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/T
    estUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin
    :/opt/pbs/bin:/opt/pbs/bin
pbs.event().job.Priority=7
```

Attributes of the execjob_launch hook:

```
Hook launch
    type = site
    enabled = true
    event = execjob_launch
    user = pbsadmin
    alarm = 30
    order = 1
    debug = true
    fail_action = none
```

Contents of the execjob_launch hook:

```
import pbs
e=pbs.event()

e.progname = "/bin/sleep"
e.argv[1] = "30"

s=pbs.server()
for j in s.jobs():
    pbs.logmsg(pbs.LOG_DEBUG, "got j %s" % (j.id,))

for q in s.queues():
    pbs.logmsg(pbs.LOG_DEBUG, "got q %s" % (q.name,))

for v in s.vnodes():
    pbs.logmsg(pbs.LOG_DEBUG, "got vnode %s" % (v.name,))

for r in s.resvs():
    pbs.logmsg(pbs.LOG_DEBUG, "got resv %s" % (r.resvid))
```

Submit a job:

```
% qsub job.scr
```

The execjob_launch hook writes the *.in, *.data, and *.out files in /var/spool/PBS/spool:

```
jupiter:/var/spool/PBS/spool # ls -ltr /var/spool/PBS/spool
-rw------- 1 TestUser users  3489 Sep 17 04:24 hook_execjob_launch_launch_12135.in
-rw------- 1 TestUser users  1045 Sep 17 04:24 hook_execjob_launch_launch_12135.out
-rw------- 1 TestUser users 15906 Sep 17 04:24 hook_execjob_launch_launch_12135.data
```

List the execjob_launch hook event file:

```
cat hook_execjob_launch_launch_12135.in
pbs.event().progname=/bin/bash
pbs.event().argv[0]=-bash
pbs.event().env=TZ=US/Eastern,PATH=/bin:/usr/bin,PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,HOME=/home/TestUser,LOGNAME=TestUser,PBS_JOBNAME=job.scr,PBS_JOBID=511.jupiter.example.com,PBS_QUEUE=workq,SHELL=/bin/bash,USER=TestUser,PBS_JOBCOOKIE=00000000434AB4BA000000000BDC62D3,PBS_NODENUM=0,PBS_TASKNUM=1,PBS_MOMPORT=15003,OMP_NUM_THREADS=1,NCPUS=1,PBS_NODEFILE=/var/spool/PBS/aux/511.jupiter.example.com,PBS_TMPDIR=/var/tmp/pbs.511.jupiter.example.com,PBS_JOBDIR=/home/TestUser,PBS_ENVIRONMENT=PBS_BATCH,ENVIRONMENT=BATCH
pbs.event().job.id=511.jupiter.example.com
pbs.event().job.Job_Name=job.scr
pbs.event().job.Job_Owner=TestUser@jupiter.example.com
pbs.event().job.job_state=T
pbs.event().job.queue=workq
pbs.event().job.server=jupiter.example.com
pbs.event().job.Checkpoint=u
pbs.event().job.Error_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.e511
pbs.event().job.exec_host2=jupiter.example.com:15002/0
pbs.event().job.exec_vnode=(jupiter:ncpus=1)
pbs.event().job.Join_Path=n
pbs.event().job.Keep_Files=n
pbs.event().job.mtime=1410942248
pbs.event().job.Output_Path=jupiter.example.com:/home/TestUser/jobs/job.scr.o511
pbs.event().job.Priority=7
pbs.event().job.Resource_List[file]=7gb
pbs.event().job.Resource_List[ncpus]=1
pbs.event().job.Resource_List[place]=pack
pbs.event().job.schedselect=1:ncpus=1
pbs.event().job.substate=0
pbs.event().job.Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.event().job.euser=TestUser
pbs.event().job.egroup=users
pbs.event().job.hashname=511.jupiter.example.com
pbs.event().job.cookie=00000000434AB4BA000000000BDC62D3
pbs.event().job.run_count=1
pbs.event().job.job_kill_delay=10
pbs.event().job.project=_pbs_project_default
```

```
pbs.event().job.run_version=1
pbs.event().job._msmom=True
pbs.event().job._stdout_file=/var/spool/PBS/spool/511.jupiter.example.com.OU
pbs.event().job._stderr_file=/var/spool/PBS/spool/511.jupiter.example.com.ER
pbs.event().vnode_list["jupiter"].resources_assigned[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_assigned[mem]=0kb
pbs.event().vnode_list["jupiter"].pcpus=1
pbs.event().vnode_list["jupiter"].resources_available[ncpus]=1
pbs.event().vnode_list["jupiter"].resources_available[mem]=757388kb
pbs.event().vnode_list["jupiter"].resources_available[arch]=linux
pbs.event().vnode_list["jupiter"].pbs_version=PBSPro_10.0
pbs.get_local_nodename()=jupiter
pbs.event().type=execjob_launch
pbs.event().hook_name=launch
pbs.event().hook_type=site
pbs.event().requestor=pbs_mom
pbs.event().requestor_host=jupiter.example.com
pbs.event().user=pbsadmin
pbs.event().alarm=30
```

List the execjob_launch hook site data file:

```
jupiter:/var/spool/PBS/spool # cat hook_execjob_launch_launch_12135.data
pbs.server().server_state=Active
pbs.server().server_host=jupiter.example.com
pbs.server().scheduling=True
pbs.server().total_jobs=3
pbs.server().state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0 Begun:0
pbs.server().managers=TestUser@*
pbs.server().default_queue=workq
pbs.server().log_events=511
pbs.server().mail_from=adm
pbs.server().query_other_jobs=True
pbs.server().resources_default[ncpus]=1
pbs.server().default_chunk[ncpus]=1
pbs.server().resources_assigned[mem]=0mb
pbs.server().resources_assigned[ncpus]=1
pbs.server().resources_assigned[nodect]=1
pbs.server().scheduler_iteration=600
pbs.server().flatuid=True
pbs.server().FLicenses=31
pbs.server().resv_enable=True
pbs.server().node_fail_requeue=310
pbs.server().max_array_size=10000
pbs.server().pbs_license_min=1
pbs.server().pbs_license_max=2147483647
pbs.server().pbs_license_linger_time=3600
pbs.server().license_count=Avail_Global:0 Avail_Local:31 Used:1 High_Use:2
pbs.server().pbs_version=PBSPro_10.0
pbs.server().eligible_time_enable=False
pbs.server().max_concurrent_provision=5
pbs.server().job(501.jupiter.example.com).Job_Name=STDIN
pbs.server().job(501.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(501.jupiter.example.com).job_state=H
pbs.server().job(501.jupiter.example.com).queue=workq
pbs.server().job(501.jupiter.example.com).server=jupiter.example.com
pbs.server().job(501.jupiter.example.com).Checkpoint=u
pbs.server().job(501.jupiter.example.com).ctime=1410940219
pbs.server().job(501.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e501
pbs.server().job(501.jupiter.example.com).Hold_Types=u
pbs.server().job(501.jupiter.example.com).Join_Path=n
pbs.server().job(501.jupiter.example.com).Keep_Files=n
pbs.server().job(501.jupiter.example.com).Mail_Points=a
pbs.server().job(501.jupiter.example.com).mtime=1410940219
pbs.server().job(501.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o501
```

```
pbs.server().job(501.jupiter.example.com).Priority=7
pbs.server().job(501.jupiter.example.com).qtime=1410940219
pbs.server().job(501.jupiter.example.com).Rerunable=True
pbs.server().job(501.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(501.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(501.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(501.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(501.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().job(501.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(501.jupiter.example.com).substate=20
pbs.server().job(501.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(501.jupiter.example.com).euser=TestUser
pbs.server().job(501.jupiter.example.com).egroup=users
pbs.server().job(501.jupiter.example.com).queue_rank=185
pbs.server().job(501.jupiter.example.com).queue_type=E
pbs.server().job(501.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(501.jupiter.example.com).project=_pbs_project_default
pbs.server().job(502.jupiter.example.com).Job_Name=STDIN
pbs.server().job(502.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(502.jupiter.example.com).job_state=H
pbs.server().job(502.jupiter.example.com).queue=workq
pbs.server().job(502.jupiter.example.com).server=jupiter.example.com
pbs.server().job(502.jupiter.example.com).Checkpoint=u
pbs.server().job(502.jupiter.example.com).ctime=1410940221
pbs.server().job(502.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/STD
    IN.e502
pbs.server().job(502.jupiter.example.com).Hold_Types=u
pbs.server().job(502.jupiter.example.com).Join_Path=n
pbs.server().job(502.jupiter.example.com).Keep_Files=n
pbs.server().job(502.jupiter.example.com).Mail_Points=a
pbs.server().job(502.jupiter.example.com).mtime=1410940221
pbs.server().job(502.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/ST
    DIN.o502
pbs.server().job(502.jupiter.example.com).Priority=7
pbs.server().job(502.jupiter.example.com).qtime=1410940223
pbs.server().job(502.jupiter.example.com).Rerunable=True
pbs.server().job(502.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(502.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(502.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(502.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(502.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(502.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(502.jupiter.example.com).substate=20
pbs.server().job(502.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,PBS_O_HOME=/home/TestUser,PBS_O_LOGNAME=TestUser,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LA
    NG=en_US.UTF-8,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/mpi/gcc/op
    enmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/g
    ames:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,PBS_O_MAIL=/var/spool/mail/TestUser,PBS_O_QUEUE=
    workq,PBS_O_HOST=jupiter.example.com
pbs.server().job(502.jupiter.example.com).euser=TestUser
pbs.server().job(502.jupiter.example.com).egroup=users
pbs.server().job(502.jupiter.example.com).queue_rank=186
pbs.server().job(502.jupiter.example.com).queue_type=E
pbs.server().job(502.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>-h</jsdl-hpcpa:A
    rgument>
pbs.server().job(502.jupiter.example.com).project=_pbs_project_default
pbs.server().job(511.jupiter.example.com).Job_Name=job.scr
pbs.server().job(511.jupiter.example.com).Job_Owner=TestUser@jupiter.example.com
pbs.server().job(511.jupiter.example.com).resources_used[cpupercent]=0
pbs.server().job(511.jupiter.example.com).resources_used[cput]=00:00:00
pbs.server().job(511.jupiter.example.com).resources_used[mem]=0kb
pbs.server().job(511.jupiter.example.com).resources_used[ncpus]=1
pbs.server().job(511.jupiter.example.com).resources_used[vmem]=0kb
pbs.server().job(511.jupiter.example.com).resources_used[walltime]=00:00:00
pbs.server().job(511.jupiter.example.com).job_state=R
pbs.server().job(511.jupiter.example.com).queue=workq
pbs.server().job(511.jupiter.example.com).server=jupiter.example.com
pbs.server().job(511.jupiter.example.com).Checkpoint=u
pbs.server().job(511.jupiter.example.com).ctime=1410942249
pbs.server().job(511.jupiter.example.com).Error_Path=jupiter.example.com:/home/TestUser/jobs/job
    .scr.e511
pbs.server().job(511.jupiter.example.com).exec_host=jupiter/0
pbs.server().job(511.jupiter.example.com).exec_vnode=(jupiter:ncpus=1)
pbs.server().job(511.jupiter.example.com).Hold_Types=n
pbs.server().job(511.jupiter.example.com).Join_Path=n
pbs.server().job(511.jupiter.example.com).Keep_Files=n
pbs.server().job(511.jupiter.example.com).Mail_Points=a
pbs.server().job(511.jupiter.example.com).mtime=1410942250
pbs.server().job(511.jupiter.example.com).Output_Path=jupiter.example.com:/home/TestUser/jobs/jo
    b.scr.o511
pbs.server().job(511.jupiter.example.com).Priority=7
pbs.server().job(511.jupiter.example.com).qtime=1410942249
pbs.server().job(511.jupiter.example.com).Rerunable=True
pbs.server().job(511.jupiter.example.com).Resource_List[file]=7gb
pbs.server().job(511.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().job(511.jupiter.example.com).Resource_List[nodect]=1
pbs.server().job(511.jupiter.example.com).Resource_List[place]=pack
pbs.server().job(511.jupiter.example.com).Resource_List[select]=1:ncpus=1
```

```
pbs.server().job(511.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().job(511.jupiter.example.com).stime=1410942250
pbs.server().job(511.jupiter.example.com).session_id=12134
pbs.server().job(511.jupiter.example.com).jobdir=/home/TestUser
pbs.server().job(511.jupiter.example.com).substate=42
pbs.server().job(511.jupiter.example.com).Variable_List=PBS_O_SYSTEM=Linux,PBS_O_SHELL=/bin/bash
    ,Monsieur=Shlomi,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_O_LOGNAME=Test
    User,PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_LANG=en_US.UTF-8,PBS_O_QUEUE=workq,PBS_O_MAIL=/
    var/spool/mail/TestUser,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/usr/lib64/m
    pi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/b
    in:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin
pbs.server().job(511.jupiter.example.com).euser=TestUser
pbs.server().job(511.jupiter.example.com).egroup=users
pbs.server().job(511.jupiter.example.com).hashname=511.jupiter.example.com
pbs.server().job(511.jupiter.example.com).queue_rank=192
pbs.server().job(511.jupiter.example.com).queue_type=E
pbs.server().job(511.jupiter.example.com).comment=Job run at Wed Sep 17 at 04:24 on
    (jupiter:ncpus=1)
pbs.server().job(511.jupiter.example.com).etime=1410942249
pbs.server().job(511.jupiter.example.com).run_count=1
pbs.server().job(511.jupiter.example.com).Submit_arguments=<jsdl-hpcpa:Argument>job.scr</jsdl-hp
    cpa:Argument>
pbs.server().job(511.jupiter.example.com).project=_pbs_project_default
pbs.server().job(511.jupiter.example.com).run_version=1
pbs.server().queue(workq).queue_type=Execution
pbs.server().queue(workq).total_jobs=3
pbs.server().queue(workq).state_count=Transit:0 Queued:0 Held:2 Waiting:0 Running:1 Exiting:0
    Begun:0
pbs.server().queue(workq).resources_assigned[mem]=0mb
pbs.server().queue(workq).resources_assigned[ncpus]=1
pbs.server().queue(workq).resources_assigned[nodect]=1
pbs.server().queue(workq).enabled=True
pbs.server().queue(workq).started=True
pbs.server().queue(R503).queue_type=Execution
pbs.server().queue(R503).total_jobs=0
pbs.server().queue(R503).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R503).acl_user_enable=True
pbs.server().queue(R503).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R503).resources_max[ncpus]=1
pbs.server().queue(R503).resources_max[walltime]=00:30:00
pbs.server().queue(R503).resources_available[ncpus]=1
pbs.server().queue(R503).resources_available[walltime]=00:30:00
pbs.server().queue(R503).enabled=True
pbs.server().queue(R503).started=False
pbs.server().queue(R504).queue_type=Execution
pbs.server().queue(R504).total_jobs=0
```

```
pbs.server().queue(R504).state_count=Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
    Begun:0
pbs.server().queue(R504).acl_user_enable=True
pbs.server().queue(R504).acl_users=TestUser@jupiter.example.com
pbs.server().queue(R504).resources_max[ncpus]=1
pbs.server().queue(R504).resources_max[walltime]=00:30:00
pbs.server().queue(R504).resources_available[ncpus]=1
pbs.server().queue(R504).resources_available[walltime]=00:30:00
pbs.server().queue(R504).enabled=True
pbs.server().queue(R504).started=False
pbs.server().vnode(jupiter).Mom=jupiter.example.com
pbs.server().vnode(jupiter).Port=15002
pbs.server().vnode(jupiter).pbs_version=PBSPro_10.0
pbs.server().vnode(jupiter).ntype=0
pbs.server().vnode(jupiter).state=0
pbs.server().vnode(jupiter).pcpus=1
pbs.server().vnode(jupiter).jobs=511.jupiter.example.com/0
pbs.server().vnode(jupiter).resv=R504.jupiter.example.com, R503.jupiter.example.com
pbs.server().vnode(jupiter).resources_available[arch]=linux
pbs.server().vnode(jupiter).resources_available[file]=7gb
pbs.server().vnode(jupiter).resources_available[host]=jupiter
pbs.server().vnode(jupiter).resources_available[mem]=8gb
pbs.server().vnode(jupiter).resources_available[ncpus]=8
pbs.server().vnode(jupiter).resources_available[vnode]=jupiter
pbs.server().vnode(jupiter).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(jupiter).resources_assigned[mem]=0kb
pbs.server().vnode(jupiter).resources_assigned[naccelerators]=0
pbs.server().vnode(jupiter).resources_assigned[ncpus]=1
pbs.server().vnode(jupiter).resources_assigned[vmem]=0kb
pbs.server().vnode(jupiter).resv_enable=True
pbs.server().vnode(jupiter).sharing=1
pbs.server().vnode(mars).Mom=mars.example.com
pbs.server().vnode(mars).Port=15002
pbs.server().vnode(mars).pbs_version=PBSPro_10.0
pbs.server().vnode(mars).ntype=0
pbs.server().vnode(mars).state=0
pbs.server().vnode(mars).pcpus=1
pbs.server().vnode(mars).resources_available[arch]=linux
pbs.server().vnode(mars).resources_available[file]=7gb
pbs.server().vnode(mars).resources_available[host]=mars
pbs.server().vnode(mars).resources_available[mem]=8gb
pbs.server().vnode(mars).resources_available[ncpus]=8
pbs.server().vnode(mars).resources_available[vnode]=mars
pbs.server().vnode(mars).resources_assigned[accelerator_memory]=0kb
pbs.server().vnode(mars).resources_assigned[mem]=0kb
pbs.server().vnode(mars).resources_assigned[naccelerators]=0
```

```
pbs.server().vnode(mars).resources_assigned[ncpus]=0
pbs.server().vnode(mars).resources_assigned[vmem]=0kb
pbs.server().vnode(mars).resv_enable=True
pbs.server().vnode(mars).sharing=1
pbs.server().resv(R503.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R503.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).reserve_type=2
pbs.server().resv(R503.jupiter.example.com).reserve_state=2
pbs.server().resv(R503.jupiter.example.com).reserve_substate=2
pbs.server().resv(R503.jupiter.example.com).reserve_start=1410955200
pbs.server().resv(R503.jupiter.example.com).reserve_end=1410957000
pbs.server().resv(R503.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R503.jupiter.example.com).queue=R503
pbs.server().resv(R503.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R503.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R503.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R503.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R503.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).server=jupiter.example.com
pbs.server().resv(R503.jupiter.example.com).ctime=1410940237
pbs.server().resv(R503.jupiter.example.com).mtime=1410940237
pbs.server().resv(R503.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R503.jupiter.example.com).euser=TestUser
pbs.server().resv(R503.jupiter.example.com).egroup=users
pbs.server().resv(R504.jupiter.example.com).Reserve_Name=NULL
pbs.server().resv(R504.jupiter.example.com).Reserve_Owner=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).reserve_type=2
pbs.server().resv(R504.jupiter.example.com).reserve_state=2
pbs.server().resv(R504.jupiter.example.com).reserve_substate=2
pbs.server().resv(R504.jupiter.example.com).reserve_start=1410958800
pbs.server().resv(R504.jupiter.example.com).reserve_end=1410960600
pbs.server().resv(R504.jupiter.example.com).reserve_duration=1800
pbs.server().resv(R504.jupiter.example.com).queue=R504
pbs.server().resv(R504.jupiter.example.com).Resource_List[ncpus]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[walltime]=00:30:00
pbs.server().resv(R504.jupiter.example.com).Resource_List[nodect]=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[select]=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).Resource_List[place]=free
pbs.server().resv(R504.jupiter.example.com).schedselect=1:ncpus=1
pbs.server().resv(R504.jupiter.example.com).resv_nodes=(jupiter:ncpus=1)
pbs.server().resv(R504.jupiter.example.com).Authorized_Users=TestUser@jupiter.example.com
pbs.server().resv(R504.jupiter.example.com).server=jupiter.example.com
```

```
pbs.server().resv(R504.jupiter.example.com).ctime=1410940250
pbs.server().resv(R504.jupiter.example.com).mtime=1410940250
pbs.server().resv(R504.jupiter.example.com).Variable_List=PBS_O_LOGNAME=TestUser,PBS_O_HOST=jupi
    ter.example.com,PBS_O_MAIL=/var/spool/mail/TestUser
pbs.server().resv(R504.jupiter.example.com).euser=TestUser
pbs.server().resv(R504.jupiter.example.com).egroup=users
```

List the execjob_launch hook execution record file:

```
jupiter:/var/spool/PBS/spool # cat hook_execjob_launch_launch_12135.out
pbs.event().accept=True
pbs.event().reject=False
pbs.event().progname=/bin/sleep
pbs.event().argv[0]=sleep
pbs.event().env=PBS_O_SYSTEM=Linux,PBS_JOBCOOKIE=00000000434AB4BA000000000BDC62D3,PBS_O_SHELL=/b
    in/bash,PBS_O_HOME=/home/TestUser,PBS_O_HOST=jupiter.example.com,PBS_NODENUM=0,PBS_O_LOGNAME
    =TestUser,PBS_JOBID=511.jupiter.example.com,PBS_JOBNAME=job.scr,PBS_O_LANG=en_US.UTF-8,USER=
    TestUser,PATH=/bin:/usr/bin,HOME=/home/TestUser,PBS_QUEUE=workq,PBS_O_MAIL=/var/spool/mail/T
    estUser,PBS_TMPDIR=/var/tmp/pbs.511.jupiter.example.com,ENVIRONMENT=BATCH,PBS_NODEFILE=/var/
    spool/PBS/aux/511.jupiter.example.com,SHELL=/bin/bash,PBS_ENVIRONMENT=PBS_BATCH,Monsieur=Shl
    omi,OMP_NUM_THREADS=1,NCPUS=1,PBS_JOBDIR=/home/TestUser,PBS_O_QUEUE=workq,PBS_MOMPORT=15003,
    PBS_O_WORKDIR=/home/TestUser/jobs,PBS_O_PATH=/usr/local/bin:/usr/local/bin:/usr/local/bin:/u
    sr/lib64/mpi/gcc/openmpi/bin:/home/TestUser/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/u
    sr/X11R6/bin:/usr/games:/opt/pbs/bin:/opt/pbs/bin:/opt/pbs/bin,LOGNAME=TestUser,PBS_TASKNUM=
    1,TZ=US/Eastern
```

8.9 Interactive Debugging using pbs_python

You can perform interactive debugging by leaving out the hook name and supplying event input information and/or site data information. For example, to interactively debug with event input and site data information:

```
pbs_python --hook -i MyEventInputFile -s MySiteData
```

You get a pbs_python prompt, and in order to end the session, issue a pbs.event().accept() or pbs.event().reject():

```
>>import pbs
>>print pbs.event().job.id
1234.examplehost
>>pbs.event().accept()
```

8.10 Error Reporting and Logging

Hook errors are printed to stderr for the command (qsub, qalter, pbs_rsub, or qmove) that triggered the hook. If the hook provides a custom error message, that message is treated the same way.

Hooks can log custom strings to the log file of the daemon from which the hook is executing. When logging a message, a hook uses message logging methods to specify the message, and constant objects to specify the log event class. See "pbs.logmsg()" on page 152, and section 6.13.4.4, "Message Log Level Objects", on page 152.

When the PBS server starts, it prints to the server logs both the Python version integrated with the server, and a list of all the hook names registered with the server.

To see only hook-related 0x0400 messages in the MoM logs, such as "<hook name>;started", "<hook_name>;finished", set the $logevent MoM parameter to *0x400* in the MoM configuration file.

To see all the different types of MoM log messages, set $logevent to *0xffff*.

The default value for the $logevent MoM parameter is *975*, so that the following log events are captured. See "Log Levels" on page 375 of the PBS Professional Reference Guide for more about log levels.

PBSEVENT_ERROR

PBSEVENT_SYSTEM

PBSEVENT_ADMIN

PBSEVENT_JOB

PBSEVENT_JOB_USAGE

PBSEVENT_SECURITY

PBSEVENT_DEBUG

PBSEVENT_DEBUG2

PBSEVENT_RESV

8.10.1 Errors During Creation and Deployment

8.10.1.1 Hook Name Matches Existing Hook

Creating a hook whose name matches that of an existing hook: the following error message is printed in stderr and in the server logs:

```
"hook error: hook name <hook_name> already registered, try another name"
```

8.10.1.2 Using a Hook Name that Starts with "*PBS*"

Using a hook name that starts with "*PBS*": the hook name is rejected with the following error in qmgr's stderr, as well as in the server logs:

```
"hook error: cannot use PBS as a prefix - it is reserved for PBS hooks"
```

8.10.1.3 Deleting a Non-Existent Hook

Deleting a non-existent hook: the following is returned in qmgr's stderr and server logs:

```
"qmgr: hook error: <non-existent hook name> does not exist"
```

8.10.1.4 Specifying a Non-Existent Event Type

Specifying a non-existent event type: an error message is printed to qmgr's stderr and also to the server logs:

Example:

```
Qmgr: set hook hook1 event="mom_checkpoint"
"hook error: invalid argument to event. Should be one of: queuejob, modifyjob, resvsub, movejob,
    runjob, provision, execjob_begin, execjob_prologue, execjob_epilogue, execjob_preterm,
    execjob_end, exechost_periodic, execjob_launch, exechost_startup, execjob_attach or "" for no
    event."
"qmgr: hook error returned from server"
```

8.10.1.5 Using a Bad Hook Value

Putting in a bad hook value: an error is printed to qmgr's stderr and also to the server logs:

Example:

```
Qmgr: set hook hook2 order=1025
"qmgr obj=hookA svr=default: order given (1025) is outside the acceptable range of [1, 1000] for
   type 'site'."
"qmgr: hook error returned from server"
```

8.10.1.6 Unauthorized User

If qmgr is invoked, and the object being operated on is "*hook*", and the executing user at some host does not have access to the target server's private location for hooks data, then the following error is issued to stderr and server logs:

```
"<user>@<host> is unauthorized to access hooks data from server <hostname>"
```

8.10.1.7 Setting a Bad Hook Type

Setting a bad type to a hook produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: invalid argument to type. Must be site"
```

8.10.1.8 Setting a Bad Alarm Value

Setting a bad alarm value to a hook produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: alarm value of a hook must be > 0"
```

8.10.1.9 Exporting To Non-Writable File

Exporting a hook's content to a file that is not writable due to ownership or permission problems results in the following error message being printed to stderr:

```
"qmgr: hook error: <output_file> permission denied"
```

8.10.1.10 Setting Bad Hook user Attribute

Setting a value for the user attribute of a hook to something other than "*pbsadmin*" produces the following error message in qmgr's stderr and also in the server logs:

```
"hook error: user value of a hook must be pbsadmin, pbsuser"
```

This attribute does not need to be set to the actual name of the PBS service account.

8.10.1.11 Importing From Non-Readable File

Importing a hook where the PBS server is unable to open the input file because the file is non-existent, has a permission problem, or any other system-related error causes the following error message to be printed in stderr and in the server logs:

```
"qmgr: hook error: unable to open <filename> by server run by <user>@<host>: <error message>"
```

Examples:

```
"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostX: permission denied"
"qmgr: hook error: unable to open hook1.py by server run by pbsadmin@hostY: No such file or
   directory"
```

8.10.1.12 Importing or Exporting with Wrong Content Type

Importing or exporting a hook where the <*content-type*> is something other than "*application/x-python*" causes the following error message to be printed in stderr and in the server logs:

"qmgr: hook error: <content_type> must be 'application/x-python'"

Importing/exporting a hook where the <*content-encoding*> is something other than "*default*" or "*base64*" causes the following error message to be printed in stderr and on the server logs:

"qmgr: hook error: <content_encoding> must be 'default' or 'base64'"

An import call on a hook that already has a content script results in the following informational message being printed in stdout and server logs:

"qmgr: hook <hook_name> contents overwritten by file <hook input file>

8.10.1.13 Setting Vnode State to Invalid Value

Setting a vnode's state attribute to an invalid value causes the pbs.BadAttributeValueError exception to be raised.

8.10.1.14 Creating a Hook with Same Name as Existing Hook

You may find that when you remove a hook, it may take some time for the hook to be completely purged. If you run "qmgr -c 'create hook <hook_name>'" where a previous hook of the same <*hook_name*> still exists, you will see the following message:

"hook name <hook_name> is pending delete, try another name"

Either specify another name for the hook, or retry the qmgr request again later, after the previous hook is completely purged.

8.10.2 Errors And Messages During Hook Execution

8.10.2.1 Successful Operation of runjob Hook

When a hook successfully sets an attribute, one of the following is written to the server's log:

<job ID>; '<hook name>' hook set job's <attribute name> = <value>

or

Job held by '<hook name>' hook on <timedate>

8.10.2.2 Unsuccessful Operation for runjob Hook

When a hook fails to set an attribute, the following is written to the server's log:

<job ID>; '<hook name>' hook failed to set job's <attribute name> = <value>

8.10.2.3 Rejecting an Action

If a hook rejects an action by calling the pbs.event().reject() function:

- The following messages are printed to stderr of the command that triggered the hook:

  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>:<'msg' value passed to pbs.event().reject()>"

where 'msg' is the message passed (if any) as input to pbs.event().reject().

- The following messages are printed in the appropriate PBS daemon log, logged at event class 0x0400:

  "<user>@<host>…<request type> request rejected by <hook name> "<user>@<host> …<request type> <'msg' value passed to pbs.event().reject()>"

8.10.2.4 Triggering an Alarm

If the alarm was triggered while executing a hook:

- The command that initiated the request gets the following messages in its stderr:

  "<command_name>: Request rejected by filter hook <hook_name>" "<command_name>: alarm call while running hook <hook_name>"

- The following entry appears in the appropriate PBS daemon log, logged under event class PBSEVENT_DEBUG2:

  "<user>@<host>…<request type> alarm call while running hook <hook_name>, request rejected"

8.10.2.5 Encountering an Unhandled Exception

If a hook encounters an unhandled exception:

- PBS rejects the corresponding action. The command that triggered the hook gets the following message in stderr:

  "<command_name>: request rejected as filter hook <hook_name> encountered an exception. Inform admin."

- The following message appears on the appropriate PBS daemon log, logged under PBSEVENT_DEBUG2 event class:

  "<request type> hook <hook_name> encountered an exception, request rejected"

See section 5.2.3, “Hook Alarm Calls and Unhandled Exceptions”, on page 43.

8.10.2.6 Starting and Finishing Hook Execution

Whenever hook execution starts or finishes, timestamped 0x0400 event class log messages appear in the appropriate PBS daemon log:

"11/13/2007 00:00:42 …<user>@<host>…<request type> running hook named <hook name>"

"11/13/2007 00:01:42<user@><host>…<request type> <hook_name> finished"

8.10.2.7 Hook Timeout

When a hook timeout is triggered, the hook script gets a Python KeyboardInterrupt from the PBS server. The server logs show the following:

06/17/2008 17:57:16;0001;Server@host2;Svr;Server@host2;PBS server internal error (15011) in Python script received a KeyboardInterrupt, <type 'exceptions.KeyboardInterrupt'>

8.10.2.8 Hooks Attempting I/O

When the PBS server is running, stdout, stderr, and stdin are closed, so that a hook script containing calls to print to standard output or standard error, or to read input from standard input, gets the following exception:

02/24/2008 08:03:34;0086;Server@a-centauri;Svr;Server@a-centauri;Compiling script file: </var/spool/pbs/server_priv/hooks/hook_test.PY>

02/24/2008 08:03:34;0001;Server@a-centauri;Svr;Server@a-centauri;PBS server internal error (15011) in Error evaluating Python script, <type 'exceptions.IOError'>

8.10.2.9 Bad Value for debug Attribute

If you specify an invalid value for a hook's debug attribute, the following error message appears in qmgr's STDERR:

```
"unexpected value '<bad_val>' must be (not case sensitive) true|t|y|1|false|f|n|0"
```

8.10.2.10 Commands Fail Inside Hooks

When a command fails inside a hook, but succeeds outside the hook, the problem may be a difference in the environments.

8.10.2.11 runjob Hook Errors

8.10.2.11.i Modifying Hold, Execution Time, Dependency, or Project of Accepted Job

If a runjob hook accepts an event request, using pbs.event().accept(), but attempts to set a disallowed attribute, the hook request is rejected.

If the hook is triggered by a qrun command, the following message is sent to stderr where the qrun command was run. If the hook is triggered when the scheduler tries to run the job, the following message is written to the job's comment attribute:

```
request rejected by filter hook <hook_name>: cannot modify job after runjob request has been
    accepted.
```

The following message is written to the PBS server log, at log event class PBSEVENT_DEBUG2:

```
<hook name>; Found job <attribute name> attribute flagged to be set
runjob request rejected by <hook name>: cannot modify job after runjob request has been accepted.
```

8.10.2.11.ii Modifying Disallowed Attributes of Rejected Job

If a runjob hook rejects an event request, using pbs.event().reject(), but attempts to do any of the above, the following message is written to the PBS server log, at log event class 0x0100:

```
runjob request rejected by <hook name>: cannot modify job attribute
<attribute name> after runjob request has been rejected.
```

8.10.2.11.iii Modifying Vnode

If a runjob hook event is accepted via a pbs.event().accept() call, and yet an attempt is made to modify a vnode's state, then the hook request is rejected. The following message is sent to the stderr of qrun, and becomes the job's comment:

```
request rejected by filter hook <hook_name>: cannot modify vnode after runjob request has been
    accepted.
```

The following message appears in the PBS server log, logged at event class PBSEVENT_DEBUG2:

```
runjob request rejected by <hook name>: cannot modify a vnode after runjob request has been
    accepted.
```

8.10.2.11.iv runjob Hook Referencing Wrong Parameter

If a runjob hook attempts to reference a pbs.event() parameter other than pbs.event().job, the exception pbs.EventIncompatibleError is raised.

8.10.2.11.v Attempting to Set Restricted Resource

A runjob hook cannot set the value of a Resource_List member other than those listed in Table 5-9, “Built-in Job Resources Readable & Settable by Hooks via Events,” on page 60.

Setting any of the wrong resources results in the following:

- The hook request is rejected
- The following message is sent to the STDERR of qrun, or after the failed pbs_runjob():

  " request rejected by filter hook: '<hook name>' hook failed to set job's Resource_List.<resc_name> = <resc_value> (not allowed)"
- The scheduler updates the affected job's comment attribute with the above message.
- The following message appears in the server's log, logged at level PBSEVENT_DEBUG2:

  "runjob request rejected: '<hook name>' hook failed to set job's Resource_list.<resc_name> = <resc_value> (not allowed)"

8.10.2.12 Special Errors Requiring Support

If you encounter any of the following log messages, an internal failure has occurred during hook setup. Please contact PBS Professional support:

```
04/15/2011 17:55:23;0100;Server@jobim;Hook;<hook_name>t3;Encountered an error while setting event
04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_job, partially populated python job object
04/15/2011 17:55:23;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_server, partially populated python server object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_queue, partially populated python queue object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_vnode, partially populated python vnode object
04/15/2011 17:55:26;0001;Server@jobim;Svr;Server@jobim;PBS server internal error (15011) in
    _get_resv, warning: partially populated python resv object
```

8.10.3 Errors During Startup

If the server starts up and encounters a hook that has no content (no script was imported into the hook), PBS displays the following warning:

```
"failed to stat <path_server_priv_hooks>/<hook_name>.PY"
"failed to allocate storage for python script
<path_server_priv_hooks>/<hook_name>.PY"
```

8.10.4 Errors in Hook Updates

Updates to hooks are asynchronous with respect to jobs. During an update, some jobs may run on updated MoMs while others run on MoMs that are not yet updated. A multi-host job that started running before the update may find itself running on some MoMs that are updated and some that are not. In addition, a multi-host job that starts during the update may start on updated and non-updated MoMs. When a job triggers a hook, the hook that runs is the current hook, not the hook that was there when the job started. If you change, delete, or add a hook while a job is running, and the job subsequently triggers the hook, that job will encounter whatever changes have propagated to the MoM.

- If a job runs where a hook update is incomplete, PBS prints the following to the server's log file:

  "vnode <node_name>'s parent mom <mom_host>:<mom_port> has a pending copy hook or delete hook request"

Bear in mind that hooks are updated asynchronously with respect to jobs, so a multi-host job that started before the update may encounter an incompletely updated hook.

- As PBS copies or deletes execution or periodic hooks to the MoMs, the following messages are printed in the server's log file at 2047:

  "successfully sent hook file <filename> to <mom_hostname>"

  "successfully sent rescdef file <filename> to <mom host name>"

  "successfully deleted hook file <filename> from <mom host name>"

  "successfully deleted rescdef file <filename> from <mom host name>"

  "failed to copy hook file <filename> to <mom host name>"

  "failed to copy rescdef file <filename> to <mom host name>"

  "failed to delete hook file <filename> from <mom host name>"

  "failed to delete rescdef file <filename> from <mom host name>"

- You may find that when you remove a hook, it takes some time for the hook to be completely purged. If you run "`qmgr -c 'create hook <hook_name>'`" where a previous hook of the same <*hook_name*> still exists, you will see the following message:

  "hook name <hook_name> is pending delete, try another name"

  Either specify another name for the hook, or retry the `qmgr` request again later, after the previous hook is completely purged.

- If a hook tries to use a resource that is not yet propagated, this will cause an exception, which if unhandled, may delete the job. Write your hooks so that they trap exceptions and deal gracefully with the job. For example, you can use pbs.event().job.rerun(). Custom resources are propagated to MoMs under the following circumstances:
  - When you install PBS on a multi-vnoded machine
  - When you add MoMs, resources are propagated to those MoMs
  - When you create a custom resource inside a hook

8.10.5 Hook-related Error Codes

The following are hook-related error codes:

Table 8-2: Hook-related Error Codes

Error Name	Code	Description
PBSE_MOM_INCOMPLETE_HOOK	*15167*	Execution hook not fully transferred to a particular MoM
PBSE_MOM_REJECT_ROOT_SCRIPTS	*15168*	A MoM has rejected a request to copy a hook-related file, or a job script to be executed by root
PBSE_HOOK_REJECT	*15169*	A MoM received a reject result from an execution or periodic hook
PBSE_HOOK_REJECT_RERUNJOB	*15170*	Hook rejection requiring a job to be rerun
PBSE_HOOK_REJECT_DELETEJOB	*15171*	Hook rejection requiring a job to be deleted

8.10.6 Troubleshooting

8.10.6.1 Bad Interpreter Path

If you see the following error:

```
/opt/pbs/bin/pbs_python: bad interpreter: No such file or directory
```

You should check to see whether this is a valid path on this host. Try to `cd` to the job execution directory and execute any command using this interpreter path.

8.10.6.2 Viewing Hook Propagation

You don't need to restart `pbs_mom` for a MoM hook to take effect. If you use `qmgr`, PBS takes care of copying the new hook over to the MoM, in the background. It's possible a job may have seen the old MoM hook before the new hook arrives. After the new hook arrives, you'll see a message in the `server_logs` with the following:

```
vnode <name>'s parent mom <mom_name> has a pending copy hook or delete hook request
```

9
Hook Examples

Contents

9.1 resvsub Hook Examples . . . 232
9-1 Restrict ability to submit reservations to PBS administrators . . . 232
9.2 queuejob Hook Examples . . . 234
9-2 Reject jobs which do not specify walltime. . . . 234
9-3 Reject jobs with CPU requests that are not multiples of 8 . . . 235
9-4 If a user asks for -l ncpus=8:ppn=24, change ncpus to 24 . . . 237
9-5 Calculate and set custom resource . . . 238
9-6 Put interactive jobs in a particular queue . . . 239
9-7 Set job project based on queue where job is submitted . . . 240
9-8 Speed up throughput of interactive jobs . . . 241
9-9 Validate job account . . . 242
9-10 Check job resource request and verify that job can run in this complex . . . 244
9.3 modifyjob Hook Examples . . . 262
9-11 Prevent users from using qalter to change their jobs . . . 262
9-12 Reject jobs requesting a specific queue that do not request mem . . . 263
9.4 periodic Hook Examples . . . 264
9-13 Run job start time estimator . . . 264
9.5 execjob_launch Hook Examples . . . 265
9-14 Modify arguments to job program . . . 265
9.6 execjob_prologue and execjob_epilogue Hook Examples . . . 266
9-15 Run shell script prologue or epilogue. . . . 266
9.7 exechost_startup Hook Examples . . . 279
9-16 Create vnode and set vnode resources . . . 279
9.8 exechost_periodic Hook Examples . . . 281
9-17 Monitor load; offline or free vnode depending on CPU load. . . . 281
9-18 Periodically update resources on vnodes . . . 282
9-19 Log loads on vnodes . . . 284
9-20 Set job attributes and resources . . . 285
9.9 Multi-event Hooks. . . . 286
9-21 Helper function for logging exceptions more completely and flexibly:. . . . 286

9.1 resvsub Hook Examples

Example 9-1: Restrict ability to submit reservations to PBS administrators

Hook type: resvsub

Script NoSub.py on Windows:

```
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]
if who in admin_ulist:
    isadmin=1
else:
    cmd = "net user "  + who + "/domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
        Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*"):
                if li.strip() in admin_glist:
                    isadmin=1
                    break
if e.type == pbs.RESVSUB and not isadmin:
    e.reject("Only admins allowed to create reservations!")
```

Script NoSub.py on Linux:

```
import pbs
import os
e = pbs.event()
r = e.resv
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]

if e.type == pbs.RESVSUB and who not in admin_ulist:
    e.reject("Only admins allowed to create reservations!")
```

Create hook and import script:

```
qmgr -c 'create hook NoSub event="resvsub"'
qmgr -c 'import hook NoSub application/x-python default NoSub.py'
```

9.2 queuejob Hook Examples

Example 9-2: Reject jobs which do not specify walltime

Hook type: queuejob

Script RequireWalltime.py:

```
import pbs
import sys

try:
    e = pbs.event()
    j = e.job
    if j.Resource_List["walltime"] == None :
        e.reject("Job has no walltime requested")

except SystemExit:
    pass

except pbs.UnsetResourceNameError:
    e.reject("Job has no walltime requested")
```

Create hook and import script:

```
qmgr -c 'create hook RequireWalltime event="queuejob"'
qmgr -c 'import hook RequireWalltime application/x-python default RequireWalltime.py'
```

Example 9-3: *Reject jobs with CPU requests that are not multiples of 8*

Hook type: queuejob

Script Multiple8.py:

```
import pbs
import sys

e = pbs.event()
j = e.job

mult_limit = 8

if j.Resource_List["ncpus"] != None:
    try:
        e = pbs.event()
        j = e.job
        R = j.Resource_List["ncpus"] % mult_limit
        if R != 0:
            e.reject("Ncpus resource is not a multiple of %s." % (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
        e.reject("Bad ncpus resource value.")

else:
    R = pbs.event().job.Resource_List
    sel = repr(R["select"])
    tot_ncpus = 0
    for chunk in sel.split("+"):
        nchunks = 1
        for c in chunk.split(":"):
            kv = c.split("=")
            if len(kv) == 1:
                nchunks = kv[0]
            elif len(kv) == 2:
                if kv[0] == "ncpus":
                    tot_ncpus += (int(nchunks) * int(kv[1]))

    try:
        mod = tot_ncpus % mult_limit
        if mod != 0:
            e.reject("Ncpus resource is not a multiple of %s." % \
            (mult_limit,))
    except SystemExit:
        pass
    except (pbs.UnsetResourceNameError, TypeError):
```

```
        e.reject("Bad Ncpus resource value.")
```

Create hook and import script:

```
qmgr -c 'create hook Multiple8 event="queuejob"'
qmgr -c 'import hook Multiple8 application/x-python default Multiple8.py'
```

Example 9-4: *If a user asks for* `-l ncpus=8:ppn=24`*, change* ncpus *to 24*

Hook type: queuejob

Script `ChangeNcpus.py`:

```
import pbs
import sys

try:
    e = pbs.event()
    j = e.job
    j.Resource_List["ncpus"] = max(j.Resource_List["ncpus"],j.Resource_List["ppn"])

except SystemExit:
    pass

except (pbs.UnsetResourceNameError, pbs.BadResourceValError):
    e.reject("Failed to reset ncpus value")
```

Create hook and import script:

```
qmgr -c 'create hook ChangeNcpus event="queuejob"'
qmgr -c 'import hook ChangeNcpus application/x-python default ChangeNcpus.py'
```

Example 9-5: *Calculate and set custom resource*

Hook type: queuejob

Custom resource cph == total ncpus * walltime (in hours). Calculate it and set its value.

You must create the cph resource before using it.

Script CustCPH.py:

```
import pbs

R = pbs.event().job.Resource_List
sel = repr(R["select"])
tot_ncpus = 0
for chunk in sel.split("+"):
    nchunks = 1
    for c in chunk.split(":"):
        kv = c.split("=")
        if len(kv) == 1:
            nchunks = kv[0]
        elif len(kv) == 2:
            if kv[0] == "ncpus":
                tot_ncpus += (int(nchunks) * int(kv[1]))
R["cph"] = tot_ncpus * R["walltime"]
```

Create hook and import script:

```
qmgr -c 'create hook CustCPH event="queuejob"'
qmgr -c 'import hook CustCPH application/x-python default CustCPH.py'
```

Example 9-6: *Put interactive jobs in a particular queue*

Hook type: queuejob

Put job into "interQ" if the job was submitted interactively (using *qsub -I*).

Script IQueue.py:

```
# get the pbs module
import pbs
import sys
try:
    # Get the hook event information and parameters
    # This will be for the 'queuejob' event type.
    e = pbs.event()

    # Get the information for the job being queued
    j = e.job
    if j.interactive:
        # Get the "interQ" queue object
        q = pbs.server().queue("interQ")
        # Reset the job's destination queue
        # parameter for this event
        j.queue = q
        # accept the event
        e.accept()
except SystemExit:
    pass
except:
    e.reject("Failed to route job to queue interQ")
```

Create hook and import script:

```
qmgr -c 'create hook IQueue event="queuejob"'
qmgr -c 'import hook IQueue application/x-python default IQueue.py'
```

Example 9-7: *Set job project based on queue where job is submitted*

Hook type: queuejob

The following is a snippet of a queuejob hook:

```
import pbs
e = pbs.event()
If e.job.queue == None:
    # user did not specify a queue to submit to, so use default
    target_qname = pbs.server().default_queue
else:
    target_qname = e.job.queue.name
If (target_qname == "large") or (target_qname == "medium"):
    e.job.project = "some_large_medium_project"
```

Example 9-8: Speed up throughput of interactive jobs

Hook type: queuejob

Use a queuejob hook that determines whether a job entering the system is an interactive job. If so, it directs the job to the high priority queue specified in 'high_priority_queue', and tells the server to restart the scheduling cycle. You must first define a "high" queue as follows:

```
qmgr -c "create queue high queue_type=e,Priority=150"
qenable high
qstart high
```

The default priority for an express queue is *150*. If you do not want interactive jobs to go into an express queue, set the priority of the queue named "high" to a value greater than ordinary queues but lower than the value for an express queue. See section 4.9.18, “Express Queues”, on page 137.

Instantiate the hook as follows:

```
qmgr -c "create hook rapid_inter event=queuejob"
qmgr -c "import hook rapid_inter application/x-python default rapid_inter.py"
```

Hook script:

```
import pbs

high_priority_queue="high"

e = pbs.event()
if e.job.interactive:
    high = pbs.server().queue(high_priority_queue)
    if high != None:
        e.job.queue = high
        pbs.logmsg(pbs.LOG_DEBUG, "quick start interactive job")
        pbs.server().scheduler_restart_cycle()
```

Example 9-9: Validate job account

This hook reads valid accounts from a JSON file.

```
import os
import simplejson


try:
    import pbs

    pbs_conf = pbs.pbs_conf

except ImportError:
    pass

# Read in the configurations file
pbs_hook_cfg = pbs.hook_config_filename
if pbs_hook_cfg == None:
    pbs.logmsg(pbs.EVENT_DEBUG3,"%s"%os.environ)
    pbs_hook_cfg = os.environ["PBS_HOOK_CONFIG_FILE"]
pbs.logmsg(pbs.EVENT_DEBUG3,"read config file: %s"%pbs.hook_config_filename)
config_file = open(pbs.hook_config_filename).read()

va_cfg = simplejson.loads(config_file)

#pbs.logmsg(pbs.EVENT_DEBUG2,"config file: %s"%va_cfg)

je = pbs.event()
j = pbs.event().job

user=je.requestor

account=j.Account_Name

#pbs.logmsg(pbs.EVENT_DEBUG2,"my Account_Name is: %s"%account)
#pbs.logmsg(pbs.EVENT_DEBUG2,"allowed users for this account are:
    %s"%va_cfg["accounts"][account])

if user in va_cfg["accounts"][account]:
    pbs.logmsg(pbs.EVENT_DEBUG2,"user is allowed to submit to account")
else:
    pbs.logmsg(pbs.EVENT_DEBUG2,"user is NOT allowed to submit to account")
    je.reject("user is unauthorized to submit to this account")
```

Here is an example JSON file with valid accounts:

```
{
   "accounts": {
      "account1":["user1"],
      "account11":["user11"],
      "account2":["user2"],
      "accountall":["user1","user2"]
   }
}
```

Example 9-10: *Check job resource request and verify that job can run in this complex*

```
#!/usr/bin/env python

# -*- coding: utf-8 -*-

############################################################################
# Purpose: To check the request of the job and verify that it will be able to
#          run on this cluster
#
# Setup: Modify the config file. You will need to provide the correct
#    information for your complex
############################################################################

import pbs
import sys
import os
from string import join
import simplejson as json
import traceback
import re
import string

pbs.logmsg(pbs.EVENT_DEBUG, "Entering the check limits hook")
e = pbs.event()

py_base_dir = pbs.pbs_conf['PBS_EXEC'] + "/python/lib/python2.5"
try:
    sys.path.index(py_base_dir + 'site-packages')
except ValueError:
    sys.path = [py_base_dir,
                py_base_dir + 'plat-linux2',
                py_base_dir + 'lib-tk',
                py_base_dir + 'lib-dynload',
                py_base_dir + 'site-packages'] \
        + sys.path

def caller_name():
    return str(sys._getframe(1).f_code.co_name)

# Define error codes

class AdminError(Exception):
    pass

class ConfigError(AdminError):
    pass
```

```
def e_reject(msg):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Message: %s" % (caller_name(), msg))
    pbs.event().reject(msg)

#
# FUNCTION decode_dict
#

def decode_dict(data):
    rv = {}
    for key, value in data.iteritems():
        if isinstance(key, unicode):
            key = key.encode('utf-8')
        if isinstance(value, unicode):
            value = value.encode('utf-8')
        elif isinstance(value, list):
            value = decode_list(value)
        elif isinstance(value, dict):
            value = decode_dict(value)
        rv[key] = value
    return rv

def decode_list(data):
    rv = []
    for item in data:
        if isinstance(item, unicode):
            item = item.encode('utf-8')
        elif isinstance(item, list):
            item = decode_list(item)
        elif isinstance(item, dict):
            item = decode_dict(item)
        rv.append(item)
    return rv

#
# FUNCTION convert_size
#
# Convert a string containing a size specification (e.g. "1m") to a
# string using different units (e.g. "1024k").
#
# This function only interprets a decimal number at the start of the string,
# stopping at any unrecognized character and ignoring the rest of the string.
#
# When down-converting (e.g. MB to KB), all calculations involve integers and
# the result returned is exact. When up-converting (e.g. KB to MB) floating
```

```
# point numbers are involved. The result is rounded up. For example:
#
# 1023MB -> GB yields 1g
# 1024MB -> GB yields 1g
# 1025MB -> GB yields 2g  <-- This value was rounded up
#
# Pattern matching or conversion may result in exceptions.
#

def convert_size(value, units='b'):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "value: %s, units: %s" % (value, units))
    logs = {'b': 0, 'k': 10, 'm': 20, 'g': 30,
            't': 40, 'p': 50, 'e': 60, 'z': 70, 'y': 80}
    try:
        new = units[0].lower()
        if new not in logs:
            new = 'b'
        val, old = re.match('([-+]?\d+)([bkmgtpezy]?)',
                            str(value).lower()).groups()

        val = int(val)
        if val < 0:
            raise ValueError('Value may not be negative')
        if old not in logs.keys():
            old = 'b'
        factor = logs[old] - logs[new]
        val *= 2 ** factor
        slop = val - int(val)
        val = int(val)
        if slop > 0:
            val += 1
        # pbs.size() does not like units following zero
        if val <= 0:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: %s" % str(0))
            return '0'
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: %s" % str(val) + new)
            return str(val) + new
    except:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return value: None")
        return None

#
# FUNCTION size_as_int
#
```

```
# Convert a size string to an integer representation of size in bytes
#


def size_as_int(value):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    return int(convert_size(value).rstrip(string.ascii_lowercase))

#
# FUNCTION caller_name
#
# Return the name of the calling function or method.
#


# Read the config file in json format
def parse_config_file(e, s):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    config = {}

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Server Name: %s" %
               (caller_name(), s.name))

    # Identify the config file and read in the data
    if pbs.hook_config_filename is not None:
        config_file = pbs.hook_config_filename
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Config file is %s" %
                   (caller_name(), config_file))
        try:
            config = json.load(open(config_file, 'r'),
                               object_hook=decode_dict)
        except IOError:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered IOError:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise ConfigError("I/O error reading config file")
        except json.JSONDecodeError:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered DecodeError:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise ConfigError(
                "JSON parsing error reading config file")
        except Exception:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Encountered error:\n %s" %
                       (caller_name(), sys.exec_info()[0]))
            raise
    else:
        raise ConfigError("No configuration file present")
```

```
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Config file:\n %s" %
               (caller_name(), config))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 0 " % (caller_name()))
    # Set some defaults if they are not present
    if 'clusters' not in config:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 1 " % (caller_name()))
        e_reject("Please define the cluster inputs ")
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 1.5 " % (caller_name()))
    if s.name not in config['clusters']:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 2 " % (caller_name()))
        e_reject("Cluster: %s needs to be " % s.name +
                 "defined in the config file: %s" % config['clusters'].keys())
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 2.5 " % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: %s " % (caller_name(), s.name))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: %s " %
               (caller_name(), config["clusters"][s.name]))
    if 'default_queue' not in config["clusters"][s.name]:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 3 " % (caller_name()))
        # Find the default queue
        config["clusters"][s.name]['default_queue'] = s.default_queue
        # e_reject("Please define the default queue for the job")
    else:
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 4 " % (caller_name()))
        try:
            s.queues(config["clusters"][s.name]['default_queue'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s: I am here 5 " % (caller_name()))
        except:
            pbs.logmsg(pbs.EVENT_DEBUG3, "%s not found in pbs complex " %
                       config["clusters"][s.name]['default_queue'])
            config["clusters"][s.name]['default_queue'] = s.default_queue
            pbs.logmsg(pbs.EVENT_DEBUG3, "Changed default queue to %s" %
                       config["clusters"][s.name]['default_queue'])
    if 'site_info' not in config["clusters"][s.name]:
        config["clusters"][s.name]['site_info'] = "not undefined"

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Return Config file:\n %s" %
               (caller_name(), config))
    return config


def chunk_resource_check(e, request, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    pbs.logmsg(pbs.EVENT_DEBUG3, "type: %s" % (type(request[0])))
    pbs.logmsg(pbs.EVENT_DEBUG3, "type: %s" % (type(request[1])))
```

```
resource = request[0]
if resource not in cluster:
    pbs.logmsg(pbs.EVENT_DEBUG3,
               "Resource %s is not defined in the cfg" %
               resource)
    try:
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Trying to return an int for %s" % (request))
        return int(request[1])
    except:
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Returning a string")
        return request[1]
elif resource == 'mem':
    value = pbs.size(request[1])
    clust_res = pbs.size(str(cluster[resource]))
elif isinstance(cluster[resource], int):
    value = int(request[1])
    clust_res = cluster[resource]
elif isinstance(cluster[resource], float):
    value = int(request[1])
    clust_res = cluster[resource]
else:
    pbs.logmsg(pbs.EVENT_DEBUG3, "Not checking resource: %s" % (resource))
    return True

pbs.logmsg(pbs.EVENT_DEBUG3, "Resource: %s" % (resource))
pbs.logmsg(pbs.EVENT_DEBUG3, "R:%s A:%s" %
           (str(request[1]), str(clust_res)))
if value > clust_res:
    pbs.logmsg(pbs.EVENT_DEBUG3, "I am here")
    line = "\nError: You requested %s=%s per " % (resource, value) + \
           "node. This exceeds the available %s " % resource + \
           "on a %s node (%s)\n" % (cluster['name'], clust_res)
    if resource == 'ncpus':
        clust_mem = pbs.size(str(cluster['mem']))
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:%s=%s:mem=%s" % \
                (resource, cluster[resource], clust_mem)
    else:
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:ncpus=%s:%s=%s" % \
                (cluster['ncpus'], resource, clust_res)
    line += "\nIf you still have questions, " + \
            "please refer to %s" % \
```

```
                cluster['site_info']
            pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
            e_reject(line)
            return False
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return %s value: %s" % (resource, value))
        return value


def job_ncpus_check(e, request, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    ncpus = int(request[1])
    if ncpus > int(cluster['ncpus']):
        line = "\nError: You requested ncpus=%d in a chunk. This " % ncpus + \
               "is more than is available on a %s " % cluster['name'] + \
               "compute node (%d).\n" % (int(cluster['ncpus']))
        line += "For example, on %s, use -l " % cluster['name'] + \
                "select=2:ncpus=%s:mpiprocs=%s " % \
                (cluster['ncpus'], cluster['ncpus']) + \
                "to use %d cores\n" % \
                (2 * int(cluster['ncpus']))
        line += "If you still have questions, refer to %s.\n" % \
                cluster['site_info']
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        e_reject(line)
    return ncpus


def job_mem_check(e, request, s, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Request: %s" % (request))
    mem = pbs.size(request[1])

    clust_mem = pbs.size(str(cluster['mem']))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Cluster mem: %s" % (clust_mem))
    if mem > clust_mem:
        pbs.logmsg(pbs.EVENT_DEBUG3, "I am here")
        line = "\nError: You requested %s of memory per " % mem + \
               "node. This exceeds the available memory " + \
               "on a %s node (%s)\n" % (cluster['name'], clust_mem)
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        line += "For example on %s use -l " % cluster['name'] + \
                "select=2:ncpus=%s:mpiprocs=%s:mem=%s" % \
                (cluster['ncpus'], cluster['ncpus'], clust_mem)
        pbs.logmsg(pbs.EVENT_DEBUG3, "line: %s" % (line))
        line += "\nIf you still have questions, " + \
```

```
                "please refer to %s" % \
                cluster['site_info']
            e_reject(line)
        pbs.logmsg(pbs.EVENT_DEBUG3, "Return mem value: %s" % (mem))
        return mem


    def job_size_mem(mem, ncpus, R, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "Check placement: %s" %
                   (repr(R['place'])))
        if repr(R['place']).find('excl') == -1:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set mem for non excl job")
            mem_line = 'mem=%s' % pbs.size(convert_size(ncpus * size_as_int(
                cluster['default_mem_per_core']), "mb"))
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set mem for excl job")
            mem_line = 'mem=%s' % pbs.size(convert_size(
                cluster['ncpus'] * size_as_int(
                    cluster['default_mem_per_core']), "mb"))
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_line: %s" % mem_line)

        return mem_line


    def job_requested_queue(j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
        # Check to see if the queue has been specified, if not specify the default
        # as defined in the config file.
        if hasattr(j.queue, 'name'):
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue: %s" %
                       j.queue.name)
        else:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Set job queue to : %s" %
                       cluster['default_queue'])
            j.queue = s.queue("%s" % cluster['default_queue'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue: %s" %
                       j.queue.name)
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Leaving" % (caller_name()))


    def job_select_cores_only(sel, s, cluster):

        # Initialize local variables
        R = pbs.event().job.Resource_List
        tmp_select = list()
```

```
try:
    tot_ncpus = int(sel)
    tot_mpiprocs = int(sel)
    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %s" % tot_ncpus)
    pbs.logmsg(pbs.EVENT_DEBUG3, "Mem/Core: %s" %
               cluster['default_mem_per_core'])
    tot_mem = pbs.size(convert_size(tot_ncpus *
                                    size_as_int(cluster['default_mem_per_core']), "mb"))

    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d\ttot_mem: %s" %
               (tot_ncpus, tot_mem))

    # Check to see if tot_ncpus > total ncpus on cluster
    if tot_ncpus > cluster['total_cpus']:
        reject_job("total", "ncpus", tot_ncpus, cluster['total_cpus'],
                   s.name, cluster)

    cores_per_node = int(cluster['ncpus'])
    full_nodes = int(tot_ncpus / cores_per_node)
    remaining_cores = tot_ncpus % cores_per_node

    pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d\tfull_nodes: %d\t" %
               (tot_ncpus, full_nodes) +
               "remaining_cores: %d\ttot_mem: %s" %
               (remaining_cores, tot_mem))

    if 'resize_select' in cluster and cluster['resize_select']:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Resizing select statement")
        if full_nodes == 0:
            tmp_select.append("1:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (remaining_cores, remaining_cores, pbs.size(
                               convert_size(remaining_cores * size_as_int(
                                   cluster['default_mem_per_core']), "mb"))))
        elif remaining_cores == 0:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Remaining cores:  0")
            pbs.logmsg(pbs.EVENT_DEBUG3, "Cluster:  %s" % cluster)
            tmp_select.append("%d:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (full_nodes, cores_per_node, cores_per_node,
                               pbs.size(convert_size(
                                   int(cores_per_node) * size_as_int(
                                       cluster['default_mem_per_core']), "mb"))))
            pbs.logmsg(pbs.EVENT_DEBUG3, "tmp_select:  %s" % tmp_select)
        else:
            tmp_select.append("%d:ncpus=%d:mpiprocs=%d:mem=%s" %
                              (full_nodes, cores_per_node, cores_per_node,
                               pbs.size(convert_size(
```

```
                                    cores_per_node * size_as_int(
                                        cluster['default_mem_per_core']), "mb"))))
                tmp_select.append("1:ncpus=%d:mpiprocs=%d:mem=%s" %
                                  (remaining_cores, remaining_cores,
                                   pbs.size(convert_size(
                                    remaining_cores * size_as_int(
                                        cluster['default_mem_per_core']), "mb"))))

            # Replace the old select statement with the new select statement
            R['select'] = pbs.select(join(tmp_select, '+'))
            pbs.logmsg(pbs.EVENT_DEBUG3, "New select Line: %s" % R['select'])
            pbs.logmsg(pbs.EVENT_DEBUG3, "Server: %s" % s.name)
            pbs.logmsg(pbs.EVENT_DEBUG3, "tot_ncpus: %d,\tcluster cores: %s" %
                       (tot_ncpus, int(cluster['ncpus'])))
            return True
        except ValueError:
            return False


def job_requested_resources(e, j, s, cluster, cfg):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

    R = j.Resource_List
    sel = repr(R["select"])

    if R["select"] == None:
        sel = '1'
        pbs.logmsg(pbs.LOG_WARNING, "%s: Requested Resources: %s" %
                   (caller_name(), R))
        if 'accept_empty_select' in cfg['clusters'][s.name]:
            if cfg['clusters'][s.name]['accept_empty_select']:
                return False

    # Calculate the ncpus and memory requested by this job
    tot_ncpus = 0
    tot_mpiprocs = 0
    tot_mem = pbs.size("0kb")
    tot_ngpus = 0
    tot_nmics = 0

    # Initialize a tmp select list
    tmp_select = list()

    pbs.logmsg(pbs.EVENT_DEBUG3, "Select Line: %s" % sel)

    # Check to see if the users just selected ncpus verses a chunk
```

```
# i.e select=32 vs select=1:ncpus=32:mpiprocs=32:mem=32gb

status = job_select_cores_only(sel, s, cluster)
pbs.logmsg(pbs.EVENT_DEBUG3,
           "job_select_cores_only Status: %s" % status)

if not status:
    pbs.logmsg(pbs.EVENT_DEBUG3, "Eval select Line: %s" % R['select'])
    for chunk in sel.split("+"):
        nchunks = 1

        tmp_chunk = chunk.split(":")
        mpiprocs = -1
        ncpus = -1
        mem = 1900

        for c in tmp_chunk:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Chunk: %s" % c)
            kv = c.split("=")
            if len(kv) == 1:
                nchunks = kv[0]
            elif kv[0] == "ncpus":
                pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus: %s" % kv)
                ncpus = chunk_resource_check(e, kv, cluster)
                tot_ncpus += int(nchunks) * ncpus
            elif kv[0] == "ngpus":
                pbs.logmsg(pbs.EVENT_DEBUG3, "ngpus: %s" % kv)
                ngpus = chunk_resource_check(e, kv, cluster)
                tot_ngpus += int(nchunks) * ngpus

            elif kv[0] == "nmics":
                pbs.logmsg(pbs.EVENT_DEBUG3, "nmics: %s" % kv)
                nmics = chunk_resource_check(e, kv, cluster)
                tot_nmics += int(nchunks) * nmics

            elif kv[0] == "mpiprocs":
                pbs.logmsg(pbs.EVENT_DEBUG3, "mpiprocs: %s" % kv)
                mpiprocs = chunk_resource_check(e, kv, cluster)
                tot_mpiprocs += int(nchunks) * mpiprocs

            elif kv[0] == "mem":
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % kv)
                mem = chunk_resource_check(e, kv, cluster)
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % mem)
                pbs.logmsg(pbs.EVENT_DEBUG3, "nchunks: %d" % int(nchunks))
                pbs.logmsg(pbs.EVENT_DEBUG3, "mem (b): %d" %
```

```
                                size_as_int(str(mem)))
                        mem = pbs.size(convert_size(
                                    int(nchunks) * size_as_int(mem), "mb"))
                        pbs.logmsg(pbs.EVENT_DEBUG3, "mem: %s" % mem)
                        tot_mem = tot_mem + mem
                        pbs.logmsg(pbs.EVENT_DEBUG3, "Total mem: %s" % tot_mem)

                # Set up the ncpus, mpiprocs, and mem if not set by the user
                if ncpus == -1:
                    ncpus = 1
                    tmp_chunk.append('ncpus=%d' % ncpus)
                if mpiprocs == -1:
                    tmp_chunk.append('mpiprocs=%d' % ncpus)
                if mpiprocs > ncpus:
                    e_reject("You cannot specify more mpiprocs than ncpus\n" +
                             "You specified: %s" % sel)
                if mem == 1900 and cluster['assign_mem_per_core']:
                    tmp_chunk.append(job_size_mem(mem, ncpus, R, cluster))

                tmp_select.append(join(tmp_chunk, ':'))
                R['select'] = pbs.select(join(tmp_select, '+'))

            pbs.logmsg(pbs.EVENT_DEBUG3, "Check tot_ncpus: %d" %
                       (tot_ncpus))

            # Check to see if tot_ncpus > total ncpus on cluster
            if tot_ncpus > cluster['total_cpus']:
                reject_job("total", "ncpus", tot_ncpus, cluster['total_cpus'],
                           s.name, cluster)
            if pbs.size(tot_mem) > pbs.size(cluster['total_mem']):
                tot_mem = pbs.size(convert_size(tot_mem, "gb"))
                reject_job("total", "mem", tot_mem, cluster['total_mem'],
                           s.name, cluster)

        pbs.logmsg(pbs.EVENT_DEBUG3, "Return resource totals")
        return {'tncpus': tot_ncpus, 'tmem': tot_mem, 'tmpiprocs': tot_mpiprocs,
                'tnmics': tot_nmics, 'tngpus': tot_ngpus}


    def reject_job(ltype, lres, lrequest, rlimit, lname, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

        line = "\nInvalid job request.\n"
        if ltype == "total":
            line += "Job requested %s=%s and the total available for %s " % \
                    (lres, lrequest, lname) + \
```

```
            "is %s=%s\n" % (lres, rlimit)

    elif (lres == "ncpus" or lres == "mem" or lres == "nmics" or
          lres == "ngpus"):
        if lname is not "":
            line += "Job requested %s=%s and the %s limit for the %s %s " % \
                    (lres, lrequest, ltype, lname, ltype) + \
                    "is %s=%s\n" % (lres, rlimit)
        else:
            line += "Job requested %s=%s and the limit for the %s is " % \
                    (lres, lrequest, ltype) + "%s=%s\n" % (lres, rlimit)
    elif lres == "walltime":
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: resource limit %s" %
                   (caller_name(), lres))
        if ltype == "None" or ltype is None:
            line += "Job has not requested a walltime \nPlease add a " + \
                    "walltime and resubmit.\n"
            line += "For example: To request 24 hours add this to the " + \
                    "submission line -lwalltime=24:00:00\n"
        if ltype == "max":
            if lname is not "":
                line += "Job has requested %s walltime which " % lrequest + \
                        "exceeds the %s %s walltime limit of %s\n" % \
                        (lname, ltype, rlimit)
            else:
                line += "Job has requested %s walltime which " % lrequest + \
                        "exceeds the %s walltime limit of %s\n" % \
                        (ltype, rlimit)
            line += "Please change the walltime or queue (depending on " + \
                    "the violated walltime limits) and resubmit.\n"
        if ltype == "min":
            if lname is not "":
                line += "Job requested %s walltime which is " % lrequest + \
                        "less than the %s %s walltime limit of %s\n" % \
                        (lname, ltype, rlimit)
            else:
                line += "Job requested %s walltime which is " % lrequest + \
                        "less than the %s walltime limit of %s\n" % \
                        (ltype, rlimit)
            line += "Please change the walltime or queue (depending on " + \
                    "the violated walltime limits) and resubmit.\n"
    else:
        pbs.logmsg(pbs.EVENT_DEBUG3, "Unknown resource: %s" % (lres))

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: line %s" %
               (caller_name(), line))
```

```
        line += "If you believe that this is a valid " + \
                "job request, please contact the HPC staff\n"
        line += "For more information, please refer to %s\n" % \
                cluster['site_info']
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Ready to reject job" % (caller_name()))
        e_reject(line)


    def check_ncpus_limits(job_res, j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
        ncpus_max_qlim = s.queue(j.queue.name).resources_max['ncpus']
        ncpus_min_qlim = s.queue(j.queue.name).resources_min['ncpus']
        ncpus_max_slim = s.resources_max['ncpus']

        pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_qlim: %s" %
                   (ncpus_max_qlim))
        pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_slim: %s" %
                   (ncpus_max_slim))

        R = j.Resource_List
        pbs.logmsg(pbs.EVENT_DEBUG3, "Job Resource List: %s" % (R))
        if R["ncpus"] != None:
            ncpus_req = R["ncpus"]
        else:
            ncpus_req = job_res['tncpus']
        pbs.logmsg(pbs.EVENT_DEBUG3, "Required ncpus: %s" % (ncpus_req))

        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "Above Find the PBS_GENERIC ncpus limit")

        # Find the PBS_GENERIC ncpus limit
        pbs.logmsg(pbs.EVENT_DEBUG3, "Above ncpus checks:")
        # Check to see if requested ncpus does not violate the limits
        if ((ncpus_max_qlim is not None) and
                (int(ncpus_req) > int(ncpus_max_qlim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_qlim:")
            reject_job("queue", "ncpus", ncpus_req, ncpus_max_qlim,
                       j.queue.name, cluster)
        elif ((ncpus_min_qlim is not None) and
              (int(ncpus_req) < int(ncpus_min_qlim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_min_qlim:")
            reject_job("queue", "ncpus", ncpus_req, ncpus_min_qlim,
                       j.queue.name, cluster)
        elif ((ncpus_max_slim is not None) and
              (int(ncpus_req) > int(ncpus_max_slim))):
            pbs.logmsg(pbs.EVENT_DEBUG3, "ncpus_max_slim:")
```

```
            reject_job("server", "ncpus", ncpus_req, ncpus_max_slim,
                       "", cluster)

        pbs.logmsg(pbs.EVENT_DEBUG3, "Done with ncpus checks")


    def check_mem_limits(job_res, j, s, cluster):
        pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))

        # Mem limit checking section
        pbs.logmsg(pbs.EVENT_DEBUG3, "Looking at mem limits")
        mem_lim = s.queue(j.queue.name).max_run_res['mem']
        pbs.logmsg(pbs.EVENT_DEBUG3,
                   "PBS_GENERIC mem limit: %s" % (mem_lim))
        mem_max_qlim = s.queue(j.queue.name).resources_max['mem']
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_max_qlim: %s" % (mem_max_qlim))

        # Get the requested memory
        R = j.Resource_List
        if R["mem"] != None:
            mem_req = R["mem"]
        else:
            mem_req = job_res['tmem']

        pbs.logmsg(pbs.EVENT_DEBUG3, "Required mem: %s" % (mem_req))

        # Find the PBS_GENERIC mem limit
        if mem_lim is not None:
            tmp_mem_lim = mem_lim.split(',')
            pbs.logmsg(pbs.EVENT_DEBUG3, "Above tmp_mem_lim: %s" %
                       (tmp_mem_lim))
            mem_lim = -1
            for limit in tmp_mem_lim:
                if "PBS_GENERIC=" in limit:
                    mem_lim = pbs.size(limit.split('=')[1].replace(']',
                                                                   ''))
            else:
                mem_lim = -1

        # Check to see if requested mem does not violate the limits
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_lim: %s" % (mem_lim))
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req: %s" % (mem_req))
        if mem_lim != -1 and pbs.size(mem_req) > mem_lim:
            pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req > mem_lim")
            reject_job("user", "mem", mem_req, mem_lim,
                       j.queue.name, cfg['clusters'][s.name])
```

```
    elif mem_max_qlim is not None and pbs.size(mem_req) > mem_max_qlim:
        pbs.logmsg(pbs.EVENT_DEBUG3, "mem_req > mem_max_lim")
        reject_job("queue", "mem", mem_req, mem_max_qlim,
                   j.queue.name, cfg['clusters'][s.name])


def check_walltime(j, s, cluster):
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    R = j.Resource_List
    if R["walltime"] != None:
        wt_req = R["walltime"]
    else:
        wt_req = None
    pbs.logmsg(pbs.EVENT_DEBUG3, "Requested walltime: %s" %
               (wt_req))

    # Get the walltime limits
    limit_name = j.queue.name
    wt_max = s.queue(limit_name).resources_max['walltime']
    wt_min = s.queue(limit_name).resources_min['walltime']
    pbs.logmsg(pbs.EVENT_DEBUG3, "Wall Limit: %s" % (wt_max))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Wall Requested: %s" % (wt_req))
    if wt_max is not None and wt_req > wt_max:
        pbs.logmsg(pbs.EVENT_DEBUG3, "This job should exit: %s" %
                   (wt_req))
    pbs.logmsg(pbs.EVENT_DEBUG3, "Check the walltimes")

    if wt_max is None:
        # Check to see if it is set at the server level
        wt_max = s.resources_max['walltime']
        if wt_max is not None:
            limit_name = "server"

    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: Method called" % (caller_name()))
    pbs.logmsg(pbs.EVENT_DEBUG3, "%s: req: %s, max: %s, min: %s, name: %s" %
               (caller_name(), wt_req, wt_max, wt_min, limit_name))
    if wt_req is None:
        if 'require_walltime' in cluster:
            if cluster['require_walltime']:
                reject_job("None", "walltime", "", "", limit_name, cluster)
    elif wt_max is not None and wt_req > wt_max:
        reject_job("max", "walltime", wt_req, wt_max,
                   limit_name, cluster)
    elif wt_min is not None and wt_req < wt_min:
        reject_job("min", "walltime", wt_req, wt_min,
                   limit_name, cluster)
```

```
        else:
            return True

        return False


    def main():
        pbs.logmsg(pbs.EVENT_DEBUG3, "Entering check limits hook")

        e = pbs.event()
        j = e.job
        s = pbs.server()
        who = e.requestor

        # Read in the config file
        cfg = parse_config_file(e, s)
        cluster = cfg['clusters'][s.name]

        # Check to see if we are running in test mode
        if 'test_mode' in cfg['clusters'][s.name]:
            if cfg['clusters'][s.name]['test_mode']:
                pbs.logmsg(pbs.EVENT_DEBUG3, "Entering check user")
                pbs.logmsg(pbs.EVENT_DEBUG3, "cfg: %s" % cfg['clusters'][s.name])
                if 'test_users' in cfg['clusters'][s.name]:
                    pbs.logmsg(pbs.EVENT_DEBUG3, "check user: %s" % who)
                    if who not in cfg['clusters'][s.name]['test_users']:
                        pbs.logmsg(pbs.EVENT_DEBUG3,
                                   "User %s not in test users %s" %
                                   (who, cfg['clusters'][s.name]['test_users']))
                        e.accept()
                    else:
                        pbs.logmsg(pbs.EVENT_DEBUG3,
                                   "Running hook for User %s" % who)

        # Collect the job requested resources
        pbs.logmsg(pbs.EVENT_DEBUG3, "Ready to look at job requested resources")
        job_res = job_requested_resources(e, j, s, cluster, cfg)
        pbs.logmsg(pbs.EVENT_DEBUG3, "Returned totals: %s" % job_res)

        try:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Default Queue: %s" %
                       cluster['default_queue'])
            job_requested_queue(j, s, cluster)
            pbs.logmsg(pbs.EVENT_DEBUG3, "job queue %s" % j.queue.name)

            # Find the server limits
```

```
            q = s.queue(j.queue.name)
            pbs.logmsg(pbs.EVENT_DEBUG3, "queue %s" % q.name)
            for key in q.attributes.keys():
                exec "a=q.%s" % key

            # check server/queue limits section
            pbs.logmsg(pbs.EVENT_DEBUG3, "Ready to check queue/server limits")
            if job_res is not False:
                if cfg['clusters'][s.name]['check_ncpus']:
                    check_ncpus_limits(job_res, j, s, cluster)

                if cfg['clusters'][s.name]['check_mem']:
                    check_mem_limits(job_res, j, s, cluster)

            if cfg['clusters'][s.name]['check_walltime']:
                check_walltime(j, s, cluster)

        except:
            err = sys.exc_info()[0]
            pbs.logmsg(pbs.EVENT_DEBUG3, "This job had an exception: %s" % err)
            pass

    if __name__ == 'builtins':
        try:
            pbs.logmsg(pbs.EVENT_DEBUG, "Entering the main loop")
            main()

        except SystemExit:
            pass
        except AdminError, exc:
            pbs.logmsg(pbs.EVENT_DEBUG3, "Encountered Admin Error")
            # Something on the system is misconfigured
            pbs.logmsg(pbs.EVENT_DEBUG3,
                       str(traceback.format_exc().strip().splitlines()))
            msg = ("Admin error in %s handling %s event" %
                   (e.hook_name, "queuejob"))
            pbs.logmsg(pbs.EVENT_ERROR, msg)
            e_reject(msg)
        except:
            e_reject("%s hook failed with %s.\nPlease contact your sys admin " +
                     "if this problem persists for more than 10 minutes" %
                     (e.hook_name, sys.exc_info()[:2]))
```

9.3 modifyjob Hook Examples

Example 9-11: Prevent users from using *qalter* to change their jobs

Hook type: modifyjob

Allow only administrators to change jobs.

Script NoAlter.py, on Windows, in a domain:

```
import os
import pbs

e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
isadmin=0
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "Administrator"]

if who in admin_ulist:
    isadmin=1
else:
    cmd = "net user "  + who + " /domain"
    admin_glist = ['Administrators', 'Domain Admins', 'Enterprise
        Admins']
    for line in os.popen(cmd).readlines():
        if line.find("Group") >= 0:
            for li in line.split("*"):
                if li.strip() in admin_glist:
                    isadmin=1
                    break
if e.type == pbs.MODIFYJOB and not isadmin:
    e.reject("Normal users are not allowed to modify their jobs")
```

Script NoAlter.py, on Linux:

```
import pbs
e = pbs.event()
j = e.job
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))
admin_ulist = ["PBS_Server", "Scheduler", "pbs_mom", "root"]
if who not in admin_ulist:
    e.reject("Normal users are not allowed to modify their jobs")
```

Create hook and import script:

```
qmgr -c 'create hook NoAlter event="modifyjob"'
qmgr -c 'import hook NoAlter application/x-python default NoAlter.py'
```

Example 9-12: *Reject jobs requesting a specific queue that do not request mem*

Hook type: modifyjob

Reject jobs requesting workq2 if they don't also request memory.

Script queuespec.py:

```
import pbs
import sys
try:
    e = pbs.event()
    j = e.job
    if j.queue.name == "workq2" and not j.Resource_List["mem"]:
        e.reject("workq2 requires job to have mem specification")

except SystemExit:
    pass

except:
    e.reject("%s hook failed with %s. Please contact
        Admin" % (e.hook_name, sys.exc_info()[:2]))
```

Create hook, import script:

```
qmgr -c 'create hook queuespec event="modifyjob"'
qmgr -c 'import hook queuespec application/x-python default queuespec.py'
```

9.4 periodic Hook Examples

Example 9-13: Run job start time estimator

Hook type: periodic

Run job start time estimator named pbs_est.

Script run_pbs_est.py:

```
import pbs
import time
import os
import subprocess

pbs_est_cmd = os.path.join(pbs.pbs_conf['PBS_EXEC'], 'sbin', 'pbs_est')

e = pbs.event()

pbs.logmsg(pbs.LOG_DEBUG, "Starting job start time estimation task")

exit_stat = subprocess.call([pbs_est_cmd], shell=True)
if exit_stat != 0:
    e.reject("%s exited abnormally with return code %d" % (pbs_est_cmd, exit_stat))
else:
    e.accept()
```

9.5 execjob_launch Hook Examples

Example 9-14: Modify arguments to job program

Hook type: execjob_launch

The argv[] entries can be modified to change the existing arguments to progname.

Given the following hook:

```
# cat launch.py
import pbs
e = pbs.event()
e.argv[1] = "cool"

# qmgr -c "create hook launch event=execjob_launch"
# qmgr -c "import hook launch application/x-python default launch.py"
```

So if a job is submitted as follows:

```
% qsub -- /bin/echo uncool
```

When the job is submitted, progname = "/*bin*/*echo*", argv[0] = "/*bin*/*echo*", argv[1]="*uncool*". However, when the job executes, the execjob_launch hook runs, causing "/bin/echo cool" to execute instead of "/bin/echo uncool".

9.6 execjob_prologue and execjob_epilogue Hook Examples

Example 9-15: Run shell script prologue or epilogue.

You can use this hook when the execjob_prologue and execjob_epilogue events are used in other hooks, such as the cgroups hook, and you still want to run the classic prologue and epilogue scripts we describe in section "Using Shell Scripts for Prologue and Epilogue", on page 508 in the PBS Professional Administrator's Guide. Additionally, the hook introduces parallel prologue and epilogue shell scripts.

See "Using Hooks for Prologue and Epilogue", on page 512 in the PBS Professional Administrator's Guide, for configuration and installation instructions.

This hook is included in $PBS_EXEC/unsupported. as run_pelog_shell.py, along with its configuration file, run_pelog_shell.ini.

Configuration File

Here is the contents of run_pelog_shell.ini:

```
[run_pelog_shell]
# Enable parallel prologues/epilogues that run on sister moms. Note that all
# the normal requirements apply, except the scripts should be named pprologue
# and pepilogue.
ENABLE_PARALLEL=False

# Provide verbose hook output to the user's .o/.e file
VERBOSE_USER_OUTPUT=False

# DEFAULT_ACTION can be one of DELETE or RERUN
DEFAULT_ACTION=RERUN

# Enable Torque argument compatibility
TORQUE_COMPAT=False
```

Hook Script

Here is the hook script (the contents of run_pelog_shell.py):

```
import pbs
import os, sys
import time

# Set up a few variables
start_time=time.time()
pbs_event=pbs.event()
hook_name=pbs_event.hook_name
hook_alarm=30 # default, we'll read it from the .HK later
DEBUG=False # default, we'll read it from the .HK later
job=pbs_event.job

# The trace_hook function has been written to be portable between hooks.
def trace_hook(**kwargs):
    """Simple exception trace logger for PBS hooks
    loglevel=<int> (pbs.LOG_DEBUG): log level to pass to pbs.logmsg()
    reject=True: reject the job upon completion of logging trace
    trace_in_reject=<bool> (False): pass trace to pbs.event().reject()
    trace_in_reject=<str>: message to pass to pbs.event().reject() with trace
    Usage:
    try:
        your=code(here)
    except:
        trace_hook()
    """
    import sys

    if 'loglevel' in kwargs:
        loglevel=kwargs['loglevel']
    else:
        loglevel=pbs.LOG_ERROR
    if 'reject' in kwargs:
        reject=kwargs['reject']
    else:
        reject=True
    if 'trace_in_reject' in kwargs:
        trace_in_reject=kwargs['trace_in_reject']
    else:
        trace_in_reject=False

    # Associate hook events with the appropriate PBS constant. This is a list
    # of all hook events as of PBS Pro 13.0. If the event does not exist, it is
    # removed from the list.
    hook_events=['queuejob', 'modifyjob', 'movejob', 'runjob', 'execjob_begin',
```

```
            'execjob_prologue', 'execjob_launch', 'execjob_attach',
            'execjob_preterm', 'execjob_epilogue', 'execjob_end',
            'resvsub', 'resv_end', 'provision', 'exechost_periodic',
            'exechost_startup', 'periodic']

hook_event={}
for he in hook_events:
    # Only set available hooks for the current version of PBS.
    if hasattr(pbs, he.upper()):
        event_code=eval('pbs.'+he.upper())
        hook_event[event_code]=he
        hook_event[he]=event_code
        hook_event[he.upper()]=event_code
        del event_code
    else:
        del hook_events[hook_events.index(he)]

trace={
    'line':      sys.exc_info()[2].tb_lineno,
    'module':    sys.exc_info()[2].tb_frame.f_code.co_name,
    'exception': sys.exc_info()[0].__name__,
    'message':   sys.exc_info()[1].message,
}
tracemsg='%s hook %s encountered an exception: Line %s in %s %s: %s' %(
    hook_event[pbs.event().type], pbs.event().hook_name,
    trace['line'], trace['module'], trace['exception'], trace['message']
)
rejectmsg="Hook Error: request rejected as filter hook '%s' encountered " \
    "an exception. Please inform Admin" % pbs.event().hook_name
if not isinstance(loglevel, int):
    pbs.logmsg(pbs.LOG_ERROR, 'trace_hook() called with invalid argument' \
        ' (loglevel=%s), setting to pbs.LOG_ERROR. ' % repr(loglevel))
    loglevel=pbs.LOG_ERROR

pbs.logmsg(loglevel, tracemsg)

if reject:
    tracemsg+=', request rejected'
    if isinstance(trace_in_reject, bool):
        if trace_in_reject:
            pbs.event().reject(tracemsg)
        else:
            pbs.event().reject(rejectmsg)
    else:
        pbs.event().reject(str(trace_in_reject)+'Line %s in %s %s:\n%s' % (
            trace['line'],trace['module'],trace['exception'],
```

```
                trace['message'] ))

class JobLog:
    """ Class for managing output to job stdout and stderr."""

    def __init__(self):
        PBS_SPOOL=os.path.join(pbs_conf()['PBS_MOM_HOME'], 'spool')
        self.stdout_log=os.path.join(PBS_SPOOL,
                                     '%s.OU' % str(pbs.event().job.id))
        self.stderr_log=os.path.join(PBS_SPOOL,
                                     '%s.ER' % str(pbs.event().job.id))

        if str(pbs.event().job.Join_Path) == 'oe':
            self.stderr_log=self.stdout_log
        elif str(pbs.event().job.Join_Path) == 'eo':
            self.stdout_log=self.stderr_log

    def stdout(self, msg):
        """Write msg to appropriate file handle for stdout"""
        import sys

        try:
            if not pbs.event().job.interactive and pbs.event().job.in_ms_mom:
                logfile=open(self.stdout_log, 'ab+')
            else:
                logfile=sys.stdout

            if DEBUG:
                pbs.logmsg(pbs.EVENT_DEBUG3,
                    '%s;%s;[DEBUG3]: writing %s to %s' %
                        (pbs.event().hook_name,
                         pbs.event().job.id,
                         repr(msg),
                         logfile.name))

            logfile.write(msg)
            logfile.flush()
            logfile.close()
        except IOError:
            trace_hook()

    def stderr(self, msg):
        """Write msg to appropriate file handle for stdout"""
        import sys

        try:
```

```
            if not pbs.event().job.interactive and pbs.event().job.in_ms_mom():
                logfile=open(self.stderr_log, 'ab+')
            else:
                logfile=sys.stderr

            if DEBUG:
                pbs.logmsg(pbs.EVENT_DEBUG3,
                    '%s;%s;[DEBUG3]: writing %s to %s' %
                        (pbs.event().hook_name,
                         pbs.event().job.id,
                         repr(msg),
                         logfile.name))

            logfile.write(msg)
            logfile.flush()
            logfile.close()
        except IOError:
            trace_hook()


# Read in pbs.conf
def pbs_conf(pbs_key=None):
    """Function to return the values from /etc/pbs.conf
    If the PBS python interpreter hasn't been recycled, it is not necessary
    to re-read and re-parse /etc/pbs.conf. This function will simply return
    the variable that exists from the first time this function ran.
    Creates a dict containing the key/value pairs in pbs.conf, accounting for
    comments in lines and empty lines.
    Returns a string representing the pbs.conf setting for pbs_key if set, or
    the dict of all pbs.conf settings if pbs_key is not set.
    """
    import os

    if hasattr(pbs_conf, 'pbs_keys'):
        return pbs_conf.pbs_keys[pbs_key] if pbs_key else pbs_conf.pbs_keys

    if 'PBS_CONF_FILE' in os.environ.keys():
        pbs_conf_file=os.environ['PBS_CONF_FILE']
    elif sys.platform == 'win32':
        if 'ProgramFiles(x86)' in os.environ.keys():
            program_files=os.environ['ProgramFiles(x86)']
        else:
            program_files=os.environ['ProgramFiles']
        pbs_conf_file='%s\\PBS Pro\\pbs.conf' % program_files
    else:
        pbs_conf_file='/etc/pbs.conf'
```

```
    pbs_conf.pbs_keys=dict([line.split('#')[0].strip().split('=') \
        for line in open(pbs_conf_file) \
        if not line.startswith('#') and '=' in line])

    if 'PBS_MOM_HOME' not in pbs_conf.pbs_keys.keys():
        pbs_conf.pbs_keys['PBS_MOM_HOME'] = \
            pbs_conf.pbs_keys['PBS_HOME']

    return pbs_conf.pbs_keys[pbs_key] if pbs_key else pbs_conf.pbs_keys

# Primary hook execution begins here
try:

    def rejectjob(reason, action=DEFAULT_ACTION):
        """Log job rejection and then call pbs.event().reject()"""

        # Arguments to pbs.event().reject() do nothing in execjob events. Log a
        # warning instead, update the job comment, then reject the job.
        if action == RERUN:
            job.rerun()
            reason='Requeued - %s' % reason
        elif action == DELETE:
            job.delete()
            reason='Deleted - %s' % reason
        else:
            reason='Rejected - %s' % reason

        job.comment='%s: %s' % (hook_name, reason)
        pbs.logmsg(pbs.LOG_WARNING, ';'.join([hook_name, job.id, reason]))
        pbs.logjobmsg(job.id, reason) # Add a message that can be tracejob'd
        if VERBOSE_USER_OUTPUT:
            print reason
        pbs_event.reject()

    # For the path to mom_priv, we use PBS_MOM_HOME in case that is set,
    # pbs_conf() will return PBS_HOME if it is not.
    mom_priv=os.path.abspath(os.path.join(
        pbs_conf()['PBS_MOM_HOME'],'mom_priv'))

    # Get the hook alarm time from the .HK file if it exists.
    hk_file=os.path.join(mom_priv,'hooks','%s.HK' % hook_name)
    if os.path.exists(hk_file):
        hook_settings=dict([l.strip().split('=') for l in
                            open(hk_file,'r').readlines()])
        if 'alarm' in hook_settings.keys():
```

```
        hook_alarm=int(hook_settings['alarm'])
    if 'debug' in hook_settings.keys():
        DEBUG=True if hook_settings['debug']=='true' else False

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG] starting.' %
               (hook_name, job.id))

if 'PBS_HOOK_CONFIG_FILE' in os.environ:
    config_file = os.environ["PBS_HOOK_CONFIG_FILE"]
    config=dict([l.split('#')[0].strip().split('=')
                 for l in open(config_file,'r').readlines() if '=' in l])

    # Set the true/false configurations
    if 'ENABLE_PARALLEL' in config.keys():
        ENABLE_PARALLEL=config['ENABLE_PARALLEL'].lower()[0] in ['t', '1']
    if 'VERBOSE_USER_OUTPUT' in config.keys():
        VEROSE_USER_OUTPUT=config['VERBOSE_USER_OUTPUT'].lower()[0] in ['t', '1']
    if 'DEFAULT_ACTION' in config.keys():
        if config['DEFAULT_ACTION'].upper() == 'DELETE':
            DEFAULT_ACTION=DELETE
        elif config['DEFAULT_ACTION'].upper() == 'RERUN':
            DEFAULT_ACTION=RERUN
        else:
            pbs.logmsg(pbs.LOG_WARN,
                '%s;%s;[ERROR] ' % (hook_name, job.id) + \
                'DEFAULT_ACTION in %s.ini must be one ' % (hook_name) + \
                'of DELETE or RERUN.')
    if 'TORQUE_COMPAT' in config.keys():
        TORQUE_COMPAT=config['TORQUE_COMPAT'].lower()[0] in ['t', '1']

# Skip sister mom if parallel pelogs aren't enabled.
if not ENABLE_PARALLEL and not job.in_ms_mom():
    pbs_event.accept()

# Prologues and epilogues have different arguments
if pbs_event.type == pbs.EXECJOB_PROLOGUE:
    event='prologue'
    args=[
        job.id,                         # argv[1]
        job.euser,                      # argv[2]
        job.egroup                      # argv[3]
    ]
    if TORQUE_COMPAT:
        args.extend([
            job.Job_Name,               # argv[4]
```

```
            job.Resource_List,          # argv[5]
            job.queue.name,             # argv[6]
            job.Account_Name or ''      # argv[7]
        ])
elif pbs_event.type == pbs.EXECJOB_EPILOGUE:
    null='null' if not TORQUE_COMPAT else ''
    event='epilogue'
    args=[
        job.id,                         # argv[1]
        job.euser,                      # argv[2]
        job.egroup,                     # argv[3]
        job.Job_Name,                   # argv[4]
        job.session_id,                 # argv[5]
        job.Resource_List,              # argv[6]
        job.resources_used,             # argv[7]
        job.queue.name,                 # argv[8]
        job.Account_Name or null,       # argv[9]
        job.Exit_status                 # argv[10]
    ]
else: # hook has wrong events added
    pbs.logmsg(pbs.LOG_WARNING,
     '%s;%s;[ERROR] PBS event type %s not supported in this hook.' %
      (hook_name, job.id, pbs_event.type))
    pbs_event.accept()

# Handle empty arguments
args=[str(a) if ( a or a == 0 ) else '' for a in args]

if DEBUG: pbs.logmsg(pbs.LOG_DEBUG,
    '%s;%s;[DEBUG] %s event triggered.' % \
    (hook_name, job.id, event))

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG3] args=%s' % \
    (hook_name, job.id, repr(args)))

# execjob_prologue and execjob_epilogue hooks can run on all nodes, so use
# pprologue/pepilogue if available and not on primary execution node.
p='' if job.in_ms_mom() else 'p'

if DEBUG:
    pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;[DEBUG] %s.' %
        (pbs_event.hook_name,
         job.id,
         'in sister mom' if p else 'in the primary execution host'))
```

```
script=os.path.join(mom_priv, p+event)

if sys.platform == 'win32':
    script=script + '.bat'

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG3, '%s;%s;[DEBUG3] script set to %s.' % (
        pbs_event.hook_name, job.id, script))

correct_permissions = False
if not script:
    pbs_event.accept()

if not os.path.exists(script):
    pbs_event.accept()

if sys.platform == 'win32':
    # Windows support is currently not implemented.
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[ERROR] ' % (hook_name, job.id) + \
               'Classic prologues and epilogues on Windows are not ' + \
               'currently implemented in this hook.')
    pbs_event.accept()

else:
    try:
        struct_stat = os.stat(script)
    except OSError:
        rejectjob('Could not stat the %s script (%s).' %
                  (event, script), RERUN)

    # We mask for read and execute on owner make sure no one else can write
    # with 0522 (?r?x?w??w?). With this, permissions such as 0777 masked by
    # 522 will return 522. Acceptable permissions will return 500.
    correct_permissions = bool(struct_stat.st_mode & 0522 == 0500 and
                               struct_stat.st_uid == 0)

if correct_permissions:
    import signal
    import subprocess
    import shlex

    # Correction for subprocess SIGPIPE handling courtesy of Colin Watson:
    # http://www.chiark.greenend.org.uk/~cjwatson/blog/python-sigpipe.html
    def subprocess_setup():
        """subprocess_setup corrects a known bug where python installs a
```

```
    SIGPIPE handler by default. This is usually not what non-Python
    subprocesses expect"""
    signal.signal(signal.SIGPIPE, signal.SIG_DFL)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG2,
        '%s;%s;[DEBUG2] script %s has appropriate permissions.' %
        (hook_name, job.id, script))

# change to the correct working directory (PBS_HOME):
os.chdir(pbs_conf()['PBS_MOM_HOME'])

# add PBS_JOBDIR environment variable, accounting for empty job.jobdir
os.environ['PBS_JOBDIR'] = job.jobdir or ''

shell=""
if sys.platform == 'win32': #win32 is _always_ cmd
    shell="cmd /c"
else:
    # check the script for the interpreter line
    shebang=open(script, 'r').readline().strip().split('#!')
    if len(shebang)==2:
        shell=shebang[1].split()[0]
        if not os.path.exists(shell):
            rejectjob(
                'Interpreter specified in %s (%s) does not exist.' %
                (p+event, shell),
                RERUN)
    else:
        rejectjob('No interpreter specified in %s.' % (p+event), RERUN)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG2,
               '%s;%s;[DEBUG2] interpreter set to "%s".' %
               (hook_name, job.id, shell))

pbs.logmsg(pbs.LOG_DEBUG, '%s;%s;running %s.' %
           (hook_name, job.id, p+event))

# We perform a shlex.split to make sure we capture any #! arguments
cmd=shlex.split('%s %s' % (shell, script))
cmd.extend(args)

if DEBUG:
    pbs.logmsg(pbs.EVENT_DEBUG3,
        '%s;%s;[DEBUG3] cmd=%s' % (hook_name, job.id, repr(cmd)))
```

```
if str(job.Join_Path) in ['oe','eo']:
    proc=subprocess.Popen(
        cmd,
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        preexec_fn=subprocess_setup)
else:
    proc=subprocess.Popen(
        cmd,
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
        preexec_fn=subprocess_setup)

# Wait for the script to gracefully exit.
while time.time() < start_time + hook_alarm - 5:
    if proc.poll() is not None:
        break
    time.sleep(1)

# If we reach the alarm time - 5 seconds, send a SIGTERM
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Terminating %s after %s seconds' % \
               (hook_name, job.id, event, int(time.time() - start_time)))
    os.kill(proc.pid, signal.SIGTERM)
    while time.time() < start_time + hook_alarm - 3:
        if proc.poll() is not None:
            break
        time.sleep(0.5)

# If we reach an alarm time - 3 seconds, send a SIGKILL
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Killing %s after %s seconds' % \
               (hook_name, job.id, event, int(time.time() - start_time)))
    os.kill(proc.pid, signal.SIGKILL)
    while time.time() < start_time + hook_alarm - 1:
        if proc.poll() is not None:
            break
        time.sleep(0.5)

# If we still can't kill the script, log a warning and let pbs kill it
if proc.poll() is None:
    pbs.logmsg(pbs.LOG_WARNING,
               '%s;%s;[WARNING] Unable to kill %s after %s seconds' % \
```

```
                (hook_name, job.id, event, start_time - time.time()))

    # Get the stdout and stderr from the pelog
    (o, e)=proc.communicate()

    if DEBUG:
        pbs.logmsg(
            pbs.EVENT_DEBUG2,
            '%s;%s;[DEBUG2]: stdout=%s, stderr=%s.' %
            (hook_name, job.id, repr(o), repr(e)))

    joblog=JobLog()
    if o:
        joblog.stdout(o)
    if e:
        joblog.stderr(e)

    if proc.returncode:
        return_action=RERUN
        if event == 'prologue':
            return_action=RERUN
            if proc.returncode == 1:
                return_action=DELETE
        elif event == 'epilogue':
            return_action=DELETE
            if proc.returncode == 2:
                return_action=RERUN

        rejectjob(
            '%s exited with a status of %s.' % (p+event, proc.returncode),
            return_action)
    else:
        if DEBUG:
            pbs.logmsg(pbs.LOG_DEBUG,
                '%s;%s;[DEBUG] %s exited with a status of 0.' %
                (hook_name, job.id, p+event))

        if pbs_event.type == pbs.EXECJOB_PROLOGUE and VERBOSE_USER_OUTPUT:
            print '%s: attached as primary execution host.' % \
                pbs.get_local_nodename()

        pbs_event.accept()
else:
    rejectjob("The %s does not have the correct " % (p+event) + \
              'permissions. See the section entitled, ' + \
              '"Prologue and Epilogue Requirements" in the PBS Pro ' + \
```

```
                    "Administrator's Guide.", RERUN)

except SystemExit:
    pass
except:
    trace_hook()
```

9.7 exechost_startup Hook Examples

Example 9-16: Create vnode and set vnode resources

Hook type: exechost_startup

```
% cat startup.py
import pbs
e=pbs.event()
for v in e.vnode_list.keys():
    vn = e.vnode_list[v]
    vn.resources_available["file"] = pbs.size("7gb")
    vn.resources_available["fab_int"] = 9
    vn.resources_available["fab_str"] = "happy"
    vn.resources_available["fab_bool"] = False
    vn.resources_available["fab_size"] = pbs.size("7mb")
    vn.resources_available["fab_time"] = pbs.duration("00:30:00")
    vn.resources_available["fab_float"] = 7.0

e.vnode_list["mars[1]"] = pbs.vnode("mars[1]")
e.vnode_list["mars[1]"].resources_available["ncpus"] = 7
```

Create hook

```
# qmgr -c "create hook start event=exechost_startup"
# qmgr -c "import hook start application/x-python default startup.py"
```

Restart MoM

```
# kill <pbs_mom PID>
```

then

```
# systemctl start pbs
```

or

```
# /etc/init.d/pbs start  (start MoM)
```

Output

```
# pbsnodes -av
mars
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_12.3.0.140813
    ntype = PBS
    state = free
    pcpus = 4
    resources_available.arch = linux
    resources_available.fab_bool = False
    resources_available.fab_float = 7
    resources_available.fab_int = 9
    resources_available.fab_size = 7mb
    resources_available.fab_str = happy
    resources_available.fab_time = 1800
    resources_available.file = 7gb
    resources_available.host = mars
    resources_available.mem = 8gb
    resources_available.ncpus = 5
    resources_available.vmem = 16gb
    resources_available.vnode = mars
    …


mars[1]
    Mom = mars.example.com
    Port = 15002
    pbs_version = PBSPro_12.3.0.140813
    ntype = PBS
    state = free
    resources_available.arch = linux
    resources_available.file = 7gb
    resources_available.host = mars
    resources_available.ncpus = 7      (set in hook)
    resources_available.vnode = mars[1]
```

9.8 exechost_periodic Hook Examples

Example 9-17: Monitor load; offline or free vnode depending on CPU load

Hook type: exechost_periodic

Monitor load average on the local host. Offline or free the vnode representing the host depending on the CPU load.

You can modify values for ideal_load and max_load. Your hook does the following:

If the system's CPU load average rises above max_load, the state of the vnode corresponding to the current host is set to *offline*. This prevents the scheduler from scheduling jobs on this vnode.

If the system's CPU load average falls below ideal_load, the state of the vnode representing the current host is set to *free*. This allows the scheduler to schedule jobs on this vnode.

To instantiate this hook, specify the following:

```
qmgr -c "create hook load_balance event=exechost_periodic,freq=10"
qmgr -c "import hook load_balance application/x-python default load_balance.py"
```

Hook script:

```
import pbs
import os
import re

ideal_load=1.5
max_load=2.0

# get_la: returns a list of load averages within the past 1-minute, 5-minute, 15-minutes range.
def get_la():
    line=os.popen("uptime").read()
    r = re.search(r'load average: (\S+), (\S+), (\S+)$', line).groups()
    return map(float, r)

local_node = pbs.get_local_nodename()

vnl = pbs.event().vnode_list
current_state = pbs.server().vnode(local_node).state
mla = get_la()[0]
if (mla >= max_load) and ((current_state == pbs.ND_OFFLINE) == 0):
    vnl[local_node].state = pbs.ND_OFFLINE
    vnl[local_node].comment = "offlined node as it is heavily loaded"
elif (mla < ideal_load) and ((current_state == pbs.ND_OFFLINE) != 0):
    vnl[local_node].state = pbs.ND_FREE
    vnl[local_node].comment = None
```

Example 9-18: Periodically update resources on vnodes

Hook type: exechost_periodic

Periodically update the values of a set of custom resources for the vnode where the current MoM runs.

The current set includes two size types, which are scratch and home

Prerequisites:

1. Create the following custom resources:

```
qmgr -c "create resource scratch type=size, flag=nh"
qmgr -c "create resource home type=size, flag=nh"
```

2. Add the new resources to the "resources:" line in the sched_config file and restart pbs_sched:

```
% cat PBS_HOME/sched_priv/sched_config resources
ncpus, mem, arch, [...], scratch, home
```

3. Install this hook as follows:

```
qmgr -c "create hook mom_dyn_res event=exechost_periodic,freq=30"
qmgr -c "import hook mom_dyn_res application/x-python default mom_dyn_res.py"
```

The mom_dyn_res.py script:

```
# NOTE:
#  Update the dyn_res[] array below to include any other custom resources
#  to be included in the updates. Ensure that each resource added has an
#  entry in the scheduler's sched_config file.

import pbs
import os
import sys

# get_filesystem_avail_unprivileged: returns available size in kbytes
# (in pbs.size type) to unprivileged users, of the filesystem where
# 'dirname' resides.

def get_filesystem_avail_unprivileged( dirname ):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bavail) / 1024) )

# get_filesystem_avail_privileged: returns available size in kbytes
# (in pbs.size type) to privileged users, of the filesystem where 'dirname'
# resides.

def get_filesystem_avail_privileged( dirname ):
    o = os.statvfs(dirname)
    return pbs.size( "%skb" % ((o.f_bsize * o.f_bfree) / 1024) )

try:
    # Define here the custom resources as key, and the function and its
    # argument for obtaining the value of the custom resource:
```

```
    #    Format: dyn_res[<resource_name>] = [<function_name>,
    #                    <function_argument>]
    # So "<function_name>(<function_argument>)" is called to return the
    # value for custom <resource_name>.
    dyn_res = {}
    dyn_res["scratch"] = [get_filesystem_avail_unprivileged, "/tmp"]
    dyn_res["home"]    = [get_filesystem_avail_unprivileged, "/home"]

    vnl = pbs.event().vnode_list
    local_node = pbs.get_local_nodename()

    for k in dyn_res.keys():
        vnl[local_node].resources_available[k] = dyn_res[k][0](dyn_res[k][1])

except SystemExit:
    pass

except:
    e = pbs.event()
    e.reject("%s hook failed with %s. Please contact Admin" % \
    (e.hook_name, sys.exc_info()[:2]))
```

Example 9-19: Log loads on vnodes

Hook type: exechost_periodic

You must create the custom resources r1m, r5m, and r15m on the vnodes.

```
#cat getload.py


import pbs
import sys
import os
load = os.getloadavg()
r1m = load[0]
r5m = load[1]
r15m = load[2]
e = pbs.event()
mynode = pbs.get_local_nodename()
v = e.vnode_list[mynode]
v.resources_available["r1m"] = r1m
v.resources_available["r5m"] = r5m
v.resources_available["r15m"] = r15m
pbs.logmsg(pbs.LOG_DEBUG,"getloadavg: vnode %s, r1m = %f, r5m = %f, r15m = %f" %
(repr(mynode), r1m, r5m, r15m))
```

Example 9-20: *Set job attributes and resources*

Hook type: exechost_periodic

```
% cat period.py
import pbs
E = pbs.event()
for k in e.job_list.keys():
    e.job_list[k].resources_used["mem"] = pbs.size("7gb")
    e.job_list[k].Variable_List["POLI"] = "negri"
    e.job_list[k].Hold_Types = pbs.hold_types("us")
```

Create the hook:

```
# qmgr -c "create hook period event=exechost_periodic,freq=30"
# qmgr -c "import hook period application/x-python default period.py"
```

Submit several jobs:

```
% qsub job.scr
<job-id1>
% qsub job.scr
<job-id2>
```

As the exechost_periodic hook executes, the jobs get the new values:

```
% qstat -f <job-id1>
…
Resources_used.mem = 7gb
Hold_Types = us
Variable_List = …POLI=negri…
2
% qstat -f <job-id1>
…
Resources_used.mem = 7gb
Hold_Types = us
Variable_List = …POLI=negri…
```

9.9 Multi-event Hooks

Example 9-21: Helper function for logging exceptions more completely and flexibly:

```
# The trace_hook function has been written to be portable between hooks.
def trace_hook(**kwargs):
    """Simple exception trace logger for PBS hooks
    loglevel=<int> (pbs.LOG_DEBUG): log level to pass to pbs.logmsg()
    reject=True: reject the job upon completion of logging trace
    trace_in_reject=<bool> (False): pass trace to pbs.event().reject()
    trace_in_reject=<str>: message to pass to pbs.event().reject() with trace
    Usage:
    try:
        your=code(here)
    except:
        trace_hook()
    """

    import pbs
    import sys

    if 'loglevel' in kwargs:
        loglevel=kwargs['loglevel']
    else:
        loglevel=pbs.LOG_ERROR
    if 'reject' in kwargs:
        reject=kwargs['reject']
    else:
        reject=True
    if 'trace_in_reject' in kwargs:
        trace_in_reject=kwargs['trace_in_reject']
    else:
        trace_in_reject=False

    # Associate hook events with the appropriate PBS constant. This is a list
    # of all hook events as of PBS Pro 13.0. If the event does not exist, it is
    # removed from the list.
    hook_events=['queuejob', 'modifyjob', 'movejob', 'runjob', 'execjob_begin',
                 'execjob_prologue', 'execjob_launch', 'execjob_attach',
                 'execjob_preterm', 'execjob_epilogue', 'execjob_end',
                 'resvsub', 'provision', 'exechost_periodic',
                 'exechost_startup']

    hook_event={}
    for he in hook_events:
        # Only set available hooks for the current version of PBS.
```

```
    if hasattr(pbs, he.upper()):
        event_code=eval('pbs.'+he.upper())
        hook_event[event_code]=he
        hook_event[he]=event_code
        hook_event[he.upper()]=event_code
        del event_code
    else:
        del hook_events[hook_events.index(he)]

trace={
    'line':      sys.exc_info()[2].tb_lineno,
    'module':    sys.exc_info()[2].tb_frame.f_code.co_name,
    'exception': sys.exc_info()[0].__name__,
    'message':   sys.exc_info()[1].message,
}
tracemsg='%s hook %s encountered an exception: Line %s in %s %s: %s' %(
    hook_event[pbs.event().type], pbs.event().hook_name,
    trace['line'], trace['module'], trace['exception'], trace['message']
)
rejectmsg="Hook Error: request rejected as filter hook '%s' encountered " \
    "an exception. Please inform Admin" % pbs.event().hook_name
if not isinstance(loglevel, int):
    pbs.logmsg(pbs.LOG_ERROR, 'trace_hook() called with invalid argument' \
        ' (loglevel=%s), setting to pbs.LOG_ERROR. ' % repr(loglevel))
    loglevel=pbs.LOG_ERROR

pbs.logmsg(loglevel, tracemsg)

if reject:
    tracemsg+=', request rejected'
    if isinstance(trace_in_reject, bool):
        if trace_in_reject:
            pbs.event().reject(tracemsg)
        else:
            pbs.event().reject(rejectmsg)
    else:
        pbs.event().reject(str(trace_in_reject)+'Line %s in %s %s:\n%s' % (
            trace['line'],trace['module'],trace['exception'],
            trace['message'] ))
```

Index

job.resources_used HG-126

A
accept an action HG-5
action HG-5

B
built-in hook HG-5

C
configuration file
    hook HG-6
creating a hook HG-5

D
DIS HG-136

E
event HG-5
events
    exechost_periodic HG-95, HG-106, HG-107
    execjob_begin HG-96, HG-98
    execjob_end HG-105
    execjob_epilogue HG-104
    execjob_preterm HG-103
    execjob_prologue HG-97
    modifyjob HG-92
    movejob HG-93
    queuejob HG-91
    resvsub HG-90
    runjob HG-94
exechost_periodic HG-89
exechost_periodic events HG-95, HG-106, HG-107
execjob_attach HG-89
execjob_begin HG-88
execjob_begin events HG-96, HG-98
execjob_end HG-89
execjob_end events HG-104, HG-105
execjob_epilogue HG-89
execjob_launch HG-88
execjob_postsuspend HG-89
execjob_preresume HG-89
execjob_preterm HG-89
execjob_preterm events HG-103
execjob_prologue HG-88
execjob_prologue events HG-97
execution event hooks HG-6

F
failover and hooks HG-21
failure action HG-6

H
hook
    configuration file HG-6
    creating HG-5
    importing HG-6
hook configuration file HG-6

hooks
and failover HG-21
creating empty hooks HG-30
deleting HG-31
enabling and disabling HG-38
event types HG-15
events
exechost_periodic HG-95, HG-106, HG-107
execjob_begin HG-96, HG-98
execjob_end HG-104, HG-105
execjob_preterm HG-103
execjob_prologue HG-97
modifyjob HG-92
movejob HG-93
queuejob HG-91
resvsub HG-90
runjob HG-94
exechost_periodic events HG-95, HG-106, HG-107
execjob_begin events HG-96, HG-98
execjob_end events HG-104, HG-105
execjob_preterm events HG-103
execjob_prologue events HG-97
execution event HG-6
exporting HG-35
importing HG-34
interface objects HG-76
job attributes HG-55
log level objects HG-152
modifyjob events HG-92
MoM HG-6
movejob events HG-93
non-job event HG-6
overview of creating HG-30
pbs.exec_vnode HG-129
pbs.job HG-122
pbs.queue() HG-121
pbs.resv HG-131
pbs.server() HG-118
pbs.vchunk HG-131
pbs.vnode HG-133
pre-execution event HG-6
queuejob events HG-91
reservation attributes HG-59
resources HG-47
resvsub events HG-90
runjob events HG-94
setting order of execution HG-38
setting timeout HG-39
setting trigger events HG-31
simple how-to HG-11
vnode attributes HG-57

I

importing a hook HG-6
interface objects for hooks HG-76

J

job
attributes in hooks HG-55
job.array_indices_submitted HG-123
job.Checkpoint HG-124
job.delete() HG-128
job.depend HG-124
job.exec_host HG-124
job.exec_vnode HG-124
job.Execution_Time HG-124
job.group_list HG-124
job.Hold_Types HG-124
job.id HG-123
job.in_ms_mom() HG-128
job.is_checkpointed() HG-127
job.job_state HG-125
job.Mail_Points HG-126
job.Mail_Users HG-126
job.rerun() HG-129
job.stagein HG-126
job.stageout HG-126
job.User_List HG-127

L

logging
hooks log level objects HG-152

M

modifyjob HG-88
modifyjob events HG-92
MoM hooks HG-6
movejob HG-88
movejob events HG-93

N

non-job event hooks HG-6

P

pbs module HG-6
pbs.acl() HG-143
pbs.args() HG-143
pbs.checkpoint( ) HG-144
pbs.depend() HG-144
pbs.duration() HG-144
pbs.email_list() HG-144
pbs.event().accept() HG-116
pbs.event().alarm HG-109
pbs.event().hook_name HG-111, HG-112

pbs.event().hook_type HG-111
pbs.event().pid HG-112
pbs.event().reject() HG-116
pbs.event().requestor HG-113
pbs.event().requestor_host HG-113
pbs.event().type HG-113
pbs.exec_host() HG-145
pbs.exec_vnode HG-129
pbs.exec_vnode() HG-145
pbs.get_local_nodename() HG-151
pbs.group_list() HG-145
pbs.hold_types() HG-145
pbs.job HG-122
pbs.job_sort_formula() HG-145
pbs.join_path() HG-146
pbs.keep_files() HG-146
pbs.license_count() HG-146
pbs.logmsg() HG-152
pbs.mail_points() HG-146
pbs.node_group_key() HG-146
pbs.path_list() HG-146
pbs.pbs_env() HG-146
pbs.pid HG-81
pbs.place() HG-147
pbs.queue HG-121
pbs.queue() HG-121
pbs.queue.job() HG-122
pbs.range() HG-148
pbs.reboot() HG-153
pbs.resv HG-131
pbs.route_destinations() HG-148
pbs.select() HG-148
pbs.server() HG-118
pbs.server(). HG-118
pbs.server().job() HG-119
pbs.server().jobs() HG-119
pbs.server().name HG-118
pbs.server().queue() HG-120
pbs.server().queues() HG-120
pbs.server().resv() HG-120
pbs.server().resvs() HG-120
pbs.server().scheduler_restart_cycle() HG-120
pbs.server().vnode() HG-120
pbs.server().vnodes() HG-120
pbs.size() HG-150
pbs.software() HG-150
pbs.staging_list() HG-150
pbs.state_count() HG-151
pbs.user_list() HG-151
pbs.vchunk HG-131
pbs.version() HG-151
pbs.vnode HG-133
PBS_AUTH_METHOD HG-136
PBS_BATCH_SERVICE_PORT HG-136
PBS_BATCH_SERVICE_PORT_DIS HG-136
PBS_COMM_LOG_EVENTS HG-136
PBS_COMM_ROUTERS HG-136
PBS_COMM_THREADS HG-136
PBS_CONF_SYSLOG HG-140
PBS_CONF_SYSLOGSEVR HG-140
PBS_CORE_LIMIT HG-136
PBS_DATA_SERVICE_PORT HG-136
PBS_ENCRYPT_METHOD HG-136
PBS_ENVIRONMENT HG-136
PBS_EXEC HG-137
PBS_HOME HG-137
PBS_LEAF_NAME HG-137
PBS_LEAF_ROUTERS HG-137
PBS_LOCALLOG HG-137
PBS_LOG_HIGHRES_TIMESTAMP HG-137
PBS_LR_SAVE_PATH HG-137
PBS_MAIL_HOST_NAME HG-137
PBS_MANAGER_SERVICE_PORT HG-137
PBS_MOM_HOME HG-137
PBS_MOM_NODE_NAME HG-138
PBS_MOM_SERVICE_PORT HG-138
PBS_OUTPUT_HOST_NAME HG-138
PBS_PRIMARY HG-138
PBS_RCP HG-138
PBS_REMOTE_VIEWER HG-136
PBS_SCHED_THREADS HG-138
PBS_SCHEDULER_SERVICE_PORT HG-138
PBS_SCP HG-138
PBS_SECONDARY HG-139
PBS_SERVER HG-139
PBS_SERVER_HOST_NAME HG-139
PBS_START_COMM HG-139
PBS_START_MOM HG-139
PBS_START_SCHED HG-139
PBS_START_SERVER HG-139
PBS_SUPPORTED_AUTH_METHODS HG-139
PBS_TMPDIR HG-140
pbshook HG-6
pre-execution event hooks HG-6
primary server HG-138

Q

queue. HG-121
queue.job() HG-122
queue.jobs() HG-122
queue.name HG-121
queuejob HG-87
queuejob events HG-91

R

rcp HG-138
reject an action HG-6

reservation
    attributes in hooks HG-59
resources
    in hooks HG-47
resv. HG-132
resv.resvid HG-132
resvsub HG-87
resvsub events HG-90
runjob HG-88
runjob events HG-94

S

scp HG-138
secondary server HG-139
server
    primary HG-138
    secondary HG-139
server hook HG-6
setting hook trigger events HG-31

V

vchunk.chunk_resources.keys() HG-131
vchunk.vnode_name HG-131
vnode
    attributes in hooks HG-57
vnode.topology_info HG-134



Altair PBS Professional 2021.1

Reference Guide

You are reading the Altair PBS Professional 2021.1

Reference Guide (RG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation ix

1 Glossary of Terms 1

2 PBS Commands 21

2.1 Our Command Notation . . . 21
2.2 Requirements for Commands . . . 21
2.3 mpiexec . . . 26
2.4 pbs . . . 28
2.5 pbsdsh . . . 29
2.6 pbsfs . . . 31
2.7 pbsnodes . . . 35
2.8 pbsrun . . . 40
2.9 pbsrun_unwrap . . . 50
2.10 pbsrun_wrap . . . 51
2.11 pbs_account . . . 53
2.12 pbs_attach . . . 55
2.13 pbs_comm . . . 57
2.14 pbs_dataservice . . . 60
2.15 pbs_ds_password . . . 61
2.16 pbs_hostn . . . 63
2.17 pbs_idled . . . 64
2.18 pbs_iff . . . 66
2.19 pbs_interactive . . . 67
2.20 pbs_lamboot . . . 68
2.21 pbs_login . . . 69
2.22 pbs_mkdirs . . . 70
2.23 pbs_mom . . . 71
2.24 pbs_mpihp . . . 76
2.25 pbs_mpilam . . . 78
2.26 pbs_mpirun . . . 79
2.27 pbs_probe . . . 81
2.28 pbs_python . . . 83
2.29 pbs_ralter . . . 86
2.30 pbs_rdel . . . 90
2.31 pbs_release_nodes . . . 92
2.32 pbs_rstat . . . 94
2.33 pbs_rsub . . . 96
2.34 pbs_sched . . . 105
2.35 pbs_server . . . 108
2.36 pbs_snapshot . . . 112
2.37 pbs_tclsh . . . 119
2.38 pbs_tmrsh . . . 120
2.39 pbs_topologyinfo . . . 122
2.40 pbs_wish . . . 124

2.41 printjob . . . 125
2.42 qalter . . . 127
2.43 qdel . . . 140
2.44 qdisable . . . 143
2.45 qenable . . . 145
2.46 qhold . . . 147
2.47 qmgr . . . 149
2.48 qmove . . . 172
2.49 qmsg . . . 174
2.50 qorder . . . 176
2.51 qrerun . . . 178
2.52 qrls . . . 180
2.53 qrun . . . 182
2.54 qselect . . . 186
2.55 qsig . . . 192
2.56 qstart . . . 195
2.57 qstat . . . 197
2.58 qstop . . . 211
2.59 qsub . . . 213
2.60 qterm . . . 233
2.61 tracejob . . . 235
2.62 win_postinstall.py . . . 238

3 MoM Parameters 239
3.1 Syntax of MoM Configuration File . . . 239
3.2 Contents of MoM Configuration File . . . 240

4 Scheduler Parameters 249
4.1 Format of Scheduler Configuration File . . . 249
4.2 Configuration Parameters . . . 250

5 List of Built-in Resources 259
5.1 Resource Data Types . . . 259
5.2 Viewing Resource Information . . . 261
5.3 Resource Flags . . . 262
5.4 Attributes where Resources Are Tracked . . . 263
5.5 Resource Table Format . . . 264
5.6 Resources Built Into PBS . . . 265

6 Attributes 277
6.1 Attribute Behavior . . . 277
6.2 How To Set Attributes . . . 277
6.3 Viewing Attribute Values . . . 278
6.4 Attribute Table Format . . . 279
6.5 Caveats . . . 280
6.6 Server Attributes . . . 281
6.7 Scheduler Attributes . . . 298
6.8 Reservation Attributes . . . 303
6.9 Queue Attributes . . . 311
6.10 Vnode Attributes . . . 320
6.11 Job Attributes . . . 328
6.12 Hook Attributes . . . 350

7 Formats 353
7.1 Non-resource Formats . . . 353
7.2 Resource Formats . . . 359
8 States 361
8.1 Job States . . . 361
8.2 Job Array States . . . 363
8.3 Subjob States . . . 364
8.4 Server States . . . 364
8.5 Vnode States . . . 365
8.6 Reservation States . . . 367
9 The PBS Configuration File 369
9.1 Format of Configuration File . . . 369
9.2 Contents of Configuration File . . . 369
10 Log Levels 375
10.1 Log Levels . . . 375
11 Job Exit Status 377
11.1 Job Exit Status . . . 377
12 Example Configurations 379
12.1 Single Vnode System . . . 379
12.2 Separate Server and Execution Host . . . 380
12.3 Multiple Execution Hosts . . . 380
12.4 Complex Multi-level Route Queues . . . 382
12.5 External Software License Management . . . 384
12.6 Multiple User ACL Example . . . 385
13 Run Limit Error Messages 387
13.1 Run Limit Error Messages . . . 387
14 Error Codes 389
15 Request Codes 395
16 PBS Environment Variables 399
17 File Listing 403
18 Introduction to PBS 411
18.1 Acknowledgements . . . 411
Index 413

1
Glossary of Terms

This chapter describes the terms used in PBS Professional documentation.

Accept an action (Hooks)

A hook *accepts* an action when the hook allows the action to take place.

Access control list, ACL

An *ACL*, or *Access Control List*, is a list of users, groups, or hosts from which users or groups may be attempting to gain access. This list defines who or what is allowed or denied access to parts of PBS such as the server, queues, or reservations. A server ACL applies to access to the server, and therefore all of PBS. A queue's ACL applies only to that particular queue. A reservation's ACL applies only to that particular reservation. See "ACLs" on page 363 in the PBS Professional Administrator's Guide.

Access to a queue

Applies to users, groups, and hosts. Being able to submit jobs to the queue, move jobs into the queue, being able to perform operations on jobs in the queue, and being able to get the status of the queue.

Access to a reservation

Applies to users, groups, and hosts. Being able to place jobs in the reservation, whether by submitting jobs to the reservation or moving jobs into the reservation. It also means being able to delete the reservation, and being able to operate on the jobs in the reservation.

Access to the server

Applies to users, groups, and hosts. Being able to run PBS commands to submit jobs and perform operations on them such as altering, selecting, and querying status. It also means being able to get the status of the server and queues.

Account string

An *account string* is an arbitrary character string of characters that your site may use to provide additional accounting or charge information. The syntax is unspecified except that it must be a single string. When provided on the command line to a PBS utility or in a directive in a PBS job script, any embedded white space must be escaped by enclosing the string in quotes.

Action (Hooks)

A PBS operation or state transition. The actions that hooks can affect are submitting a job, altering a job, running a job, making a reservation, and moving a job to another queue.

Active (Failover)

A server daemon is active when it is managing user requests and communicating with a scheduler and MoMs.

Active Directory (Windows)

Active Directory is an implementation of LDAP directory services by Microsoft to use in Windows environments. It is a directory service used to store information about the network resources (e.g. user accounts and groups) across a domain.

Admin (Windows)

A user logged in from an account that is either:

1. A member of a group having full control over the local computer and the domain controller
2. Allowed to make domain and schema changes to the Active Directory.

Administrator

Same as PBS Administrator.

Linux: person with Manager privilege and root access.

Windows: person with Manager privilege who is a member of the local Administrators group.

A person who administers PBS, performing functions such as downloading, installing, upgrading, configuring, or managing PBS.

Administrator is distinguished from "site administrator", although often these are the same person.

Administrators (Windows)

A group that has built-in capabilities that give its members full control over the local system, or the domain controller host itself.

Advance reservation

A reservation for a specific set of resources for a specified start time and duration in the future. Advance reservations are created by users to reserve resources for jobs. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

ALM license server

The license server that supplies licenses to run a PBS complex. See the *PBS Works Licensing Guide*.

AOE, Application operating environment

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

API

PBS provides an *Application Programming Interface,* or *API,* which is used by the commands to communicate with the server. This API is described in the PBS Professional Programmer's Guide. A site may make use of the API to implement new commands if so desired.

Application checkpoint

The application performs its own checkpointing when it receives the appropriate signal etc.

Array job

See "Job array".

ASAP reservation, job-specific ASAP reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS schedules the reservation to run as soon as possible, and PBS moves the job into the reservation. Created when you use `pbs_rsub -Wqmove=<job ID>` on a queued job.

Attribute

An *attribute* is a data item belonging to an object such as a job, reservation, vnode, queue, hook, scheduler, or server. The attribute's value affects the behavior of or provides information about the object. See Chapter 6, "Attributes", on page 277. You specify attributes via the `qmgr` command.

Backfilling

A scheduling policy where

1. High-priority jobs are scheduled for execution
2. Lower-priority jobs are run if the following conditions are true:

   Resources (that cannot be used by the high-priority jobs) are available

   The lower-priority jobs will not delay the higher-priority jobs

Lower-priority jobs selected for execution are those next in priority order that will fit in the available resources.

Batch, Batch processing

Allowing jobs to be run outside of the interactive login environment.

Borrowing vnode

The vnode where a shared vnode resource is available, but not managed.

Built-in hook

A hook that is supplied as part of PBS. These hooks cannot be created or deleted by administrators. See "Managing Built-in Hooks" on page 155 in the PBS Professional Hooks Guide.

Built-in resource

A resource that is defined in PBS Professional as shipped. Examples of built-in resources are ncpus, which tracks the number of CPUs, and mem, which tracks memory. See "Built-in vs. Custom Resources" on page 233 in the PBS Professional Administrator's Guide.

Checkpoint/Restart

Allows jobs to be checkpointed and restarted. Uses OS-provided or third-party checkpoint/restart facility.

Checkpoint and Abort, checkpoint_abort

The checkpoint script or tool writes a restart file, then PBS kills and requeues the job. The job resumes from the start file when it is executed again.

Child vnode

On a multi-vnode machine, there is one parent vnode and one or more child vnodes. For multi-vnode machines, child vnodes represent hardware. See "Parent vnode" on page 13.

Chunk

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

Chunk-level resource, host-level resource

A resource that is available at the host level, for example, CPUs or memory. Chunk resources are requested inside of a selection statement. The resources of a chunk are to be applied to the portion of the job running in that chunk.

Chunk resources are requested inside a select statement. A single chunk is requested using this form:

-l select=<resource name>=<value>:<resource name>=<value>

For example, one chunk might have 2 CPUs and 4GB of memory:

```
-l select=ncpus=2:mem=4gb
```

To request multiples of a chunk, prefix the chunk specification by the number of chunks:

-l select=[number of chunks]<chunk specification>

For example, to request six of the previous chunk:

```
-l select=6:ncpus=2:mem=4gb
```

To request different chunks, concatenate the chunks using the plus sign ("+"):

-l select=[number of chunks]<chunk specification>+[number of chunks]<chunk specification>

For example, to request two kinds of chunks, one with 2 CPUs per chunk, and one with 8 CPUs per chunk, both kinds with 4GB of memory:

```
-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4GB
```

Chunk set

An identical set of chunks requested in a select statement. The following is a chunk set: 4:ncpus=8:mem=4GB

Cluster

A relatively homogeneous set of systems that are used as if they are a single machine.

Commands

PBS supplies both command line programs that are POSIX 1003.2d conforming and a graphical interface. These are used to submit, monitor, modify, and delete jobs. These client commands can be installed on any system type supported by PBS and do not require the local presence of any of the other components of PBS.

There are three classifications of commands: user commands (which any authorized user can use), Operator commands, and Manager (or administrator) commands. Operator and Manager commands require specific access privileges.

Communication daemon, comm

The daemon which handles communication between the server, scheduler, and MoMs. Executable is `pbs_comm`.

Complex

A PBS complex consists of the machines running one primary server+scheduler (plus, optionally, a secondary backup server+scheduler) and all the machines on which the MoMs (attached to this server+scheduler) are running. A complex can be a heterogeneous mix of system architectures, and can include one or more clusters.

Consumable resource

A consumable resource is a resource that is reduced or taken up by being used. Examples of consumable resources are memory or CPUs. See "Consumable vs. Non-consumable Resources" on page 234 in the PBS Professional Administrator's Guide.

CPU

Has two meanings, one from a hardware viewpoint, and one from a software viewpoint:

1. A core. The part of a processor that carries out computational tasks. Some systems present virtual cores, for example in hyperthreading.
2. Resource required to execute a program thread. PBS schedules jobs according, in part, to the number of threads, giving each thread a core on which to execute. The resource used by PBS to track CPUs is called "*ncpus*". The number of CPUs available for use defaults to the number of cores reported by the OS. When a job requests one CPU, it is requesting one core on which to run.

Creating a hook

When you "create a hook" using `qmgr`, you're telling PBS that you want it to make you an empty hook object that has no characteristics other than a name.

Custom resource

A resource that is not defined in PBS as shipped. Custom resources are created by the PBS administrator or by PBS for some systems. See "Built-in vs. Custom Resources" on page 233 in the PBS Professional Administrator's Guide.

Data service account

Created by PBS on installation. Account that is internal to the data service, with its own data service password. Used by PBS to log into and do operations on the data service. PBS maps this account to the PBS data service management account. Must have same name as PBS data service management account.

Data service management account

Created by administrator. Account with a system password. Data service account maps to the PBS data service management account and both must have the same name.

Default server

The default server is the server which handles any server tasks, such as client requests, unless you specify a different server. By default, PBS provides a default server; you do not need to take any action to have a default server. If you have installed more than one server, you can specify the default using these:

- The PBS_DEFAULT environment variable
- The PBS_SERVER parameter in `/etc/pbs.conf` on the local host

If both are present, PBS_DEFAULT overrides PBS_SERVER.

Server names have the following format:

<hostname>[:<port number>]

where *hostname* is the fully-qualified domain name of the host on which the server is running and *port number* is the port number to which to connect. If you do not specify *port number*, PBS defaults to using 15001.

There is always at least one active server; the default server is the active server unless another server has been made active.

Degraded reservation

An advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with any occurrence are unavailable.

Delegation (Windows)

A capability provided by Active Directory that allows granular assignment of privileges to a domain account or group. So for instance, instead of adding an account to the "Account Operators" group which might give too much access, delegation allows giving the account read access only to all domain users and groups information. This is done via the Delegation wizard.

Deprecate

We use *deprecated* to mean that something such as a feature or a platform is still supported, but will not be supported beginning with a later release. When a feature is no longer supported, we say it has been removed or is obsolete.

Destination, destination identifier, destination queue, destination server

String. One or more queues or a server. Jobs may be queried at or sent to a destination queue. Commands may be directed to a destination queue or server. A destination may be at the default PBS server or at another server.

Destination queue format:

<queue name>

Indicates specified queue at default server.

@<server name>

When moving a job, indicates default queue at that server.

When operating on queues, can indicate all queues at that server.

<queue name>@<server name>

Indicates specified queue at specified server.

Destination server format:

(no server name)

Indicates default server.

@<server name>

Indicates specified server.

@default

Indicates default server.

Directive

A means by which the user specifies to PBS the value of a job submission variable such as number of CPUs, the name of the job, etc. The default start of a directive is "*#PBS*". PBS directives either specify resource requirements or attribute values. See page "Using PBS Directives", on page 16 of the PBS Professional User's Guide.

Domain Admin Account (Windows)

A domain account on Windows that is a member of the "Domain Admins" group.

Domain Admins (Windows)

A global group whose members are authorized to administer the domain. By default, the Domain Admins group is a member of the Administrators group on all computers that have joined a domain, including the domain controllers.

Domain User Account (Windows)

A domain account on Windows that is a member of the Domain Users group.

Domain Users (Windows)

A global group that, by default, includes all user accounts in a domain. When you create a user account in a domain, it is added to this group automatically.

Endpoint

A PBS server, scheduler, or MoM daemon.

Enterprise Admins (Windows)

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make forest-wide changes in Active Directory, such as adding child domains.

Entity, PBS entity

A user, group, or host.

Entity share

Setting job execution and/or preemption priority according to how much of the fairshare tree is assigned to each job's owner.

Event

A PBS operation or state transition. Also called *action*. For a list of events, see "Event Types" on page 87 in the PBS Professional Hooks Guide.

Execution event hook

A hook that runs at an execution host. These hooks run after a job is received by MoM. Execution event hooks have names prefixed with "*execjob_*".

Execution host

A computer which runs PBS jobs. An *execution host* is a system with a single operating system (OS) image, a unified virtual memory space, one or more CPUs and one or more IP addresses. Systems like Linux clusters, which contain separate computational units each with their own OS, are collections of hosts. Systems such as the HPE 8600 are also collections of hosts.

An execution host can be comprised of one or more vnodes. On the HPE 8600, each blade is treated as a vnode. See "Vnode".

Execution queue

A queue from which a job can be executed.

Failover

The PBS complex can run a backup server. If the primary server fails, the secondary takes over without an interruption in service.

Failure action

The action taken when a hook fails to execute. Specified in the fail_action hook attribute. See "Using the fail_action Hook Attribute" on page 37 in the PBS Professional Hooks Guide.

Fairshare

A scheduling policy that prioritizes jobs according to how much of a specified resource is being used by, and has recently been used by, job submitters. Job submitters can be organized into groups and subgroups, so that jobs can also be prioritized according to those groups' resource usage. Users and groups can each be allotted a percentage of total resource usage. See "Using Fairshare" on page 138 in the PBS Professional Administrator's Guide.

File staging

File staging is the transfer of files between a specified storage location and the execution host. See "Stage in" and "Stage out".

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

Floating license

A unit of application license dynamically allocated (checked out) when a user begins using an application on some host (when the job starts), and deallocated (checked in) when a user finishes using the application (when the job ends).

Furnishing queue/complex

In peer scheduling, the queue/complex from which jobs are pulled to be run at another queue/complex

Generic group limit

A limit that applies separately to groups at the server or a queue. This is the limit for groups which have no individual limit specified. A limit for generic groups is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Generic project limit

Applies separately to projects at the server or a queue. The limit for projects which have no individual limit specified. A limit for generic projects is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Generic user limit

A limit that applies separately to users at the server or a queue. This is the limit for users who have no individual limit specified. A separate limit for generic users can be specified at the server and at each queue.

Global resource

A global resource is defined in a resources_available attribute, at the server, a queue, or a host. Global resources can be operated on via the `qmgr` command and are visible via the `qstat` and `pbsnodes` commands. See "Global vs. Local Resources" on page 235 in the PBS Professional Administrator's Guide.

Group

A collection of system users. A user must be a member of at least one group, and can be a member of more than one group.

Group access, Access by group

Refers to access to PBS objects, such as the server, queues, and reservations. A user in the specified group is allowed access at the server, queues, and reservations

Group ID (GID)

Unique numeric identifier assigned to each group. See "Group".

Group limit

Refers to configurable limits on resources and jobs. This is a limit applied to the total used by a group, whether the limit is a generic group limit or an individual group limit.

History jobs

Jobs which will no longer execute at this server:

- Moved jobs
- Finished jobs

Hold

A restriction which prevents a job from being executed. When a job has a hold applied to it, it is in the *Held* (*H*) state. See section 2.46, “qhold”, on page 147.

Hook

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code.

Host

A machine running an operating system. A host can be made up of one or more vnodes. All vnodes of a host share the same value for resources_available.host.

Host access, Access from host

Refers to user access at the server, queues, and reservations from the specified host

Hostname

A hostname is a string. A hostname is of the form <machine name>.<domain name>, where *domain name* is a hierarchical, dot-separated list of subdomains. A hostname cannot contain the following:

- A dot ("."), other than as a subdomain separator
- The commercial at sign, "@", as this is often used to separate a file from the host in a remote file name
- To prevent confusion with port numbers, a hostname cannot contain a colon (":")

HTT

Intel's Hyper-Threading Technology

Idle

A server daemon is idle when it is running, but only accepting handshake messages, not performing workload management.

Importing a hook

When you "import a hook" using `qmgr`, you're telling PBS which Python script to run when the hook is triggered.

Importing a hook configuration file

When you "import a hook configuration file" using `qmgr`, you're telling PBS which file should be stored as the configuration file for the specified hook.

Indirect resource

A shared vnode resource at vnode(s) where the resource is not defined, but which share the resource.

Individual group limit

Applies separately to groups at the server or a queue. This is the limit for a group which has its own individual limit specified. An individual group limit overrides the generic group limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire group. A separate limit can be specified at the server and each queue.

Individual project limit

Applies separately to projects at the server or a queue. Limit for a project which has its own individual limit specified. An individual project limit overrides the generic project limit, but only in the same context, for example, at a particular queue. The limit is applied to the usage across the entire project. A separate limit can be specified at the server and each queue.

Individual user limit

Applies separately to users at the server or a queue. This is the limit for users who have their own individual limit specified. A limit for an individual user overrides the generic user limit, but only in the same context, for example, at a particular queue. A separate limit can be specified at the server and each queue.

Installation account

The account used by the administrator when installing PBS. Not the *pbsadmin* account used by PBS.

Interactive job

A job where standard input and output are connected to the terminal from which the job was submitted.

Job or Batch job

A unit of work managed by PBS. A *job* is a related set of tasks, created and submitted by the user. The user specifies the resources required by the job, and the processes that make up the job. When the user submits a job to PBS, the user is handing off these tasks to PBS to manage. PBS then schedules the job to be run, and manages the running of the job, treating the tasks as parts of a whole. A job is usually composed of a set of directives and a shell script.

Job array

A *job array* is a container for a collection of similar jobs submitted under a single job ID. It can be submitted, queried, modified and displayed as a unit. The jobs in the collection are called subjobs. For more on job arrays, see "Job Arrays", on page 149 of the PBS Professional User's Guide.

Job array identifier

The identifier returned upon success when submitting a job array.

Job array identifiers are a sequence number followed by square brackets:

<sequence number>[][.<server name>][@<server name>]

Example:

```
1234[]
```

Note that some shells require that you enclose a job array ID in double quotes.

The largest value that *sequence number* can be is set in the max_job_sequence_id server attribute. This attribute defaults to *9999999*. Minimum value for this attribute is *9999999*, and maximum is *999999999999*. After maximum for sequence number has been reached, job array IDs start again at *0*.

Job array range

A specification for a set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices. Format:

<sequence number>[<first>-<last>:<step>][.server][@new server]

first is the first index of the subjobs.

last is the last index of the subjobs.

step is the stepping factor.

Job ID, Job identifier

When a job is successfully submitted to PBS, PBS returns a unique identifier for the job. Format:

<sequence number>[.<server>][@<new server>]

The <server> portion indicates the name of the original server where the job was submitted.

The @<new server> portion indicates the current location of the job if it is not at the original server.

The largest value that *sequence number* can be is set in the max_job_sequence_id server attribute. This attribute defaults to *9999999*. Minimum value for this attribute is *9999999*, and maximum is *999999999999*. After maximum for sequence number has been reached, job IDs start again at *0*.

Job name, Job array name

A job name or job array name can be at most 230 characters. It must consist only of alphabetic, numeric, plus sign ("+"), dash or minus or hyphen ("-"), underscore ("_"), and dot or period (".") characters.

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "*STDIN*".

Job state

A job exists in one of the possible states throughout its existence within the PBS system. For example, a job can be queued, running, or exiting. See "States" on page 361.

Job-specific ASAP reservation, ASAP reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS schedules the reservation to run as soon as possible, and PBS moves the job into the reservation. Created when you use `pbs_rsub -Wqmove=<job ID>` on a queued job.

Job-specific now reservation, now reservation

Reservation created for a specific running job. PBS creates a job-specific now reservation on the same resources as the job is using, and moves the job into the reservation. The reservation is created and starts running immediately when you use `pbs_rsub --job <job ID>` on a running job.

Job-specific reservation

A reservation created for a specific job, for the same resources that the job requested.

Job-specific start reservation, start reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS starts the job according to scheduling policy. When the job starts, PBS creates and starts the reservation, and PBS moves the job into the reservation. Created when you use `qsub -Wcreate_resv_from_job=true` on a queued job.

Job Submission Description Language (JSDL)

Language for describing the resource requirements of jobs.

Job-wide resource, server resource, queue resource

A job-wide resource, also called a server-level or queue-level resource, is a resource that is available to the entire job at the server or queue.

A job-wide resource is available to be consumed or matched at the server or queue if you set the server or queue resources_available.<resource name> attribute to the available or matching value. For example, you can define a custom resource called *FloatingLicenses* and set the server's resources_available.FloatingLicenses attribute to the number of available floating licenses.

Examples of job-wide resources are shared scratch space, application licenses, or walltime.

A job can request a job-wide resource for the entire job, but not for individual chunks. Job-wide resources are requested outside of a selection statement, in this form:

-l keyword=value[,keyword=value ...]

where *keyword* identifies either a consumable resource or a time-based resource such as walltime.

A resource request "outside of a selection statement" means that the resource request comes after "`-l`", but not after "`-lselect=`".

Kill a job

To terminate the execution of a job.

Leaf

An endpoint (a server, scheduler, or MoM daemon.)

License Manager Daemon (`lmx-serv-altair`)

Daemon that functions as the license server. ALM license server. See the *PBS Works Licensing Guide*.

License server

Manages licenses for PBS jobs. ALM license server. See the *PBS Works Licensing Guide*.

License Server List Configuration

One form of redundant license server configuration. A collection of "<port number>@<hostname>" settings, pointing to license servers managing Altair licenses. Each server on the list is tried in turn. There could be X licenses on <server1>, Y licenses on <server2>, and Z licenses on <server3>, and the total licenses available would actually be X+Y+Z, but a request must be satisfied only by one server at a time. The first running server is the only server queried. See the *PBS Works Licensing Guide*.

Limit

A maximum that can be applied in various situations:

- The maximum number of jobs that can be queued
- The maximum number of jobs that can be running
- The maximum number of jobs that can be queued and running
- The maximum amount of a resource that can be allocated to queued jobs
- The maximum amount of a resource that can be consumed at any time by running jobs
- The maximum amount of a resource that can be allocated to queued and running jobs

Linux-Windows complex, Windows-Linux complex

A PBS complex with a Linux server/scheduler/comm host and Windows execution and client hosts.

Load balance

Scheduling policy wherein jobs are distributed across multiple hosts to even out the workload on each host.

Local resource

A local resource is defined in a Version 1 MoM configuration file. Local resources cannot be operated on via the `qmgr` command and are not visible via the `qstat` and `pbsnodes` commands. Local resources can be used by a scheduler. See "Global vs. Local Resources" on page 235 in the PBS Professional Administrator's Guide.

Manager

A person who has been granted Manager privilege by being listed in the server's managers attribute. A Manager is authorized to use all restricted capabilities of PBS. A PBS Manager may act upon the server, queues, or jobs. See "Manager" on page 361 in the PBS Professional Administrator's Guide.

Managing vnode

The vnode where a shared vnode resource is defined, and which manages the resource.

Master provisioning script, Master script (Hooks)

The script that makes up the provisioning hook.

Memory-only vnode

Represents a node board that has only memory resources (no CPUs).

Mixed-mode complex

A PBS complex with a Linux server/scheduler/comm host, Linux execution and client hosts, and Windows execution and client hosts.

MoM

The daemon which runs on an execution host, managing the jobs on that host. *MoM* is the informal name for the process called `pbs_mom`. One MoM runs on each execution host.

MoM runs each job when it receives a copy of the job from the server. MoM creates a new session that is as identical to the user's login session as possible. For example under Linux, if the user's login shell is `csh`, MoM creates a session in which `.login` is run as well as `.cshrc`. MoM returns the job's output to the user when directed to do so by the server.

MoM is a reverse-engineered acronym that stands for "Machine Oriented Mini-server".

Monitoring

The act of tracking and reserving system resources and enforcing usage policy. This covers both user-level and system-level monitoring as well as monitoring running jobs. Tools are provided to aid human monitoring of the PBS system as well.

Primary execution host MoM (was Mother Superior)

The MoM on the head or first host of a multihost job. This MoM controls the job, communicates with the server, and controls and consolidates resource usage information. When a job is to run on more than one execution host, the job is sent to the MoM on the primary execution host, which then starts the job.

Moved jobs

Jobs which were moved to another server

No longer used. See "Execution host".

Non-consumable resource

A non-consumable resource is a resource that is not reduced or taken up by being used. Examples of non-consumable resources are Boolean resources and walltime. See "Consumable vs. Non-consumable Resources" on page 234 in the PBS Professional Administrator's Guide.

Non-job event hook

A hook that is not directly related to a specific job. Non-job event hooks are periodic hooks, startup hooks, provisioning hooks, and reservation creation hooks.

Now reservation, job-specific now reservation

Reservation created for a specific running job. PBS creates a job-specific now reservation on the same resources as the job is using, and moves the job into the reservation. The reservation is created and starts running immediately when you use `pbs_rsub --job <job ID>` on a running job.

nppcu

Number of Processors Per Compute Unit - Cray BASIL 1.3 attribute in the RESERVE XML for specifying how many processors of a compute unit should be used.

Object, PBS object

An element of PBS such as the server, a queue, or a reservation

Occurrence of a standing reservation

An instance of the standing reservation.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- While a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See the `qsub(1B)` man page.
- When an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the resv_nodes attribute reported by `pbs_rstat`.

Operator

This term means a person who has been granted Operator privilege by being listed in the server's operators attribute. An Operator can use some but not all of the restricted capabilities of PBS. See "Operator" on page 360 in the PBS Professional Administrator's Guide.

Overall limit

Limit on the total usage. In the context of server limits, this is the limit for usage at the PBS complex. In the context of queue limits, this is the limit for usage at the queue. An overall limit is applied to the total usage at the specified location. Separate overall limits can be specified at the server and each queue.

Owner, Job owner

The user who submitted a specific job to PBS.

Parameter

A *parameter* specifies an element of the behavior of a component of PBS. For example, MoMs have parameters specifying which events to log, or what the maximum load should be. Parameters are specified by editing the component's configuration files.

Parent vnode

For single-vnode machines, the only vnode is the parent vnode.

For multi-vnode machines, there is a vnode called the *parent vnode*. A parent vnode does not correspond to any actual hardware. The parent vnode is used to define any placement set information that is invariant for a given host. The parent vnode is also used to define dynamic host-level resources, and can be used to define shared resources. See "Parent Vnodes and Child Vnodes" on page 38 in the PBS Professional Administrator's Guide. We used to call this vnode the "natural vnode".

For multi-vnode machines, vnodes that represent hardware are called *child vnodes*. See "Child vnode" on page 3.

Node

A host

pbshook

Keyword used by `qmgr` to operate on built-in hooks.

PBS Entity

A user, group, or host

pbs module

The *pbs module* is an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

PBS Object

An element of PBS such as the server, a queue, or a reservation

PBS Administrator

Same as Administrator.

Linux: person with Manager privilege and root access.

Windows: person with Manager privilege who is a member of the local Administrators group.

A person who administers PBS, performing functions such as downloading, installing, upgrading, configuring, or managing PBS.

PBS Administrator is distinguished from "site administrator", although often these are the same person.

pbsadmin (Windows)

The account that is used to execute the PBS MoM, `pbs_mom`, via the Service Control Manager on Windows. This must be "*pbsadmin*".

PBS_HOME

The path containing PBS files. The path under which PBS files are installed on the local system.

Default: `/var/spool/pbs`

PBS_EXEC

The path containing PBS executables. The path under which PBS executables are installed on the local system.

Default: `/opt/pbs`

PBS Professional

A workload management system consisting of a server, a scheduler, and any number of execution hosts each managed by a MoM. PBS accepts batch jobs from users, and schedules them on execution hosts according to the policy chosen by the site. PBS manages the jobs and their output according to site-specified policy.

Peer scheduling

A feature allowing different PBS complexes to automatically run each others' jobs. This way jobs can be dynamically load-balanced across the complexes. Each complex involved in peer scheduling is called a *peer*.

Placement set

A set of vnodes on which jobs can be run, selected so that the job will run as efficiently as possible. Placement sets are used to improve task placement (optimizing to provide a "good fit") by exposing information on system configuration and topology. See "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

Placement set series

The set of placement sets defined by a resource, where each set has the same value for the resource. If the resource takes on N values, there are N placement sets in the series. See "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

Placement pool

All of the placement sets defined at a PBS object. Each queue can have its own placement pool, and the server can have its own placement pool. See "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

Policy, Scheduling policy

The set of rules by which a scheduler selects jobs for execution.

POSIX

Refers to the various standards developed by the Technical Committee on Operating Systems and Application Environments of the IEEE Computer Society under standard P1003.

Preempt

Stop one or more running jobs in order to start a higher-priority job.

Preemption level

Job characteristic used to determine whether a job may preempt another or may be preempted, such as being in an express queue, starving, having an owner who is over a soft limit, being a normal job, or having an owner who is over a fairshare allotment.

Preemption method

The method by which a job is preempted. This can be checkpointing, suspension, or requeueing.

Preemption target

A preemption target is a job in a specified queue or a job that has requested a specified resource. The queue and/or resource is specified in another job's Resource_List.preempt_targets.

Pre-execution event hook

A hook that runs before the job is accepted by MoM. These hooks do not run on execution hosts. Pre-execution event hooks are for job submission, moving a job, altering a job, or just before sending a job to an execution host.

Primary scheduler

The PBS Professional scheduler daemon which is running during normal operation.

Primary execution host

The execution host where a job's top task runs, and where the MoM that manages the job runs.

Primary server

The PBS Professional server daemon which is running during normal operation.

Primetime and non-primetime

An arbitrary, defined set of time slots during which scheduling follows the rules specified for primetime. By default primetime is 24/7, but you can define it to be any desired time slots. If a time slot is not primetime, it is non-primetime, during which scheduling follows non-primetime rules. There are default behaviors for primetime and non-primetime, but you can set up the behavior you want for each type. You can also define primetime and non-primetime queues. Jobs in a primetime queue run only during primetime, and jobs in non-primetime queues run only during non-primetime. See "Using Primetime and Holidays" on page 191 in the PBS Professional Administrator's Guide.

Project

In PBS, a project is a way to group jobs independently of users and groups. A project is a tag that identifies a set of jobs. Each job's project attribute specifies the job's project.

Project limit

This is a limit applied to the total used by a project, whether the limit is a generic project limit or an individual project limit.

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioned vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

Provisioning hook

The hook which performs the provisioning, either by calling other scripts or by running commands

Provisioning tool

A tool that performs the actual provisioning, e.g. HPE Performance Cluster Manager (HPCM).

Pulling queue

In peer scheduling, the queue into which jobs are pulled, and from which they are run

Queue

A *queue* is a named container for jobs at a server. There are two types of queues in PBS: routing queues and execution queues. A *routing queue* is a queue used to move jobs to other queues including those that exist on other PBS servers. A job must reside in an *execution queue* to be eligible to run and remains in an execution queue during the time it is running. In spite of the name, jobs in a queue need not be processed in queue order (first-come first-served or *FIFO*).

Queuing

The collecting together of work or tasks to be run on a computer. Users submit tasks or "jobs" to the resource management system where they are queued up until the system is ready to run them.

Redundant License Server Configuration

Allows licenses to continue to be available should one or more license servers fail. There are two types: 1) license server list configuration, and 2) three-server configuration. See the *PBS Works Licensing Guide*.

Reject an action (Hooks)

An action is *rejected* when a hook prevents the action from taking place.

Requeue

The process of stopping a running job and putting it back into the *queued* ("*Q*") state.

Rerunnable

If a running PBS job can be terminated and then restarted from the beginning without harmful side effects, the job is rerunnable. The job's Rerunnable attribute must be set to *y* in order for PBS to consider a job to be rerunnable.

Reservation degradation

PBS attempts to ensure that reservations run by finding usable vnodes when reservation vnodes become unavailable.

Reservation ID, reservation identifier

When a reservation is successfully submitted to PBS, PBS returns a unique identifier for the reservation.

Format for advance reservation:

R<sequence number>[.server][@new server]

Format for standing reservation:

S<sequence number>[.server][@new server]

Format for maintenance reservation:

M<sequence number>[.server][@new server]

Resource

A *resource* can be something used by a job, such as CPUs, memory, high-speed switches, scratch space, application licenses, or time, or it can be an arbitrary item defined for another purpose. PBS has built-in resources, and allows custom-defined resources. See "Using PBS Resources" on page 229 in the PBS Professional Administrator's Guide.

Restart

A job that was stopped after being checkpointed while previously executing is executed again, starting from the point where it was checkpointed.

Restart File

The job-specific file that is written by the checkpoint script or tool. This file contains any information needed to restart the job from where it was when it was checkpointed.

Restart Script

The script that MoM runs to restart a job. This script is common to all jobs, and so must use the information in a job's restart file to restart the job.

Route a job

When PBS moves a job between queues. PBS provides a mechanism whereby a job is automatically moved from a routing queue to another queue. This is performed by PBS. The resource request for each job in a routing queue is examined, and the job is placed in a destination queue which matches the resource request. The destination queue can be an execution queue or another routing queue.

Routing queue

A queue that serves as a temporary holding place for jobs, before they are moved to another queue. Jobs cannot run from routing queues.

Scheduler

A scheduler is a daemon which implements some or all of the site's job scheduling policy controlling when and where each job is run. A scheduler is a process called `pbs_sched`.

Scheduling

The process of selecting which jobs to run when and where, according to a predetermined policy. Sites balance competing needs and goals on the system(s) to maximize efficient use of resources (both computer time and people time).

Scheduling policy

Scheduling policy determines when each job runs, and and how much of each resource it can use. Scheduling policy consists of a system for determining the priority of each job, combined with a set of limits on how many jobs can be run, and/or how much of each resource can be used.

Schema Admins (Windows)

A group that exists only in the root domain of an Active Directory forest of domains. The group is authorized to make schema changes in Active Directory.

Secondary scheduler

The PBS Professional scheduler daemon which takes over when the primary scheduler is not available.

Secondary server

The PBS Professional server daemon which takes over when the primary server fails.

Sequence number

The numeric part of a job ID, job array ID, or reservation ID, for example, *1234*. The largest value that can be used for a sequence number is set in the max_job_sequence_id job attribute.

Server

The central PBS daemon, which does the following:

- Handles PBS commands
- Receives and creates batch jobs
- Sends jobs for execution

The server is the process called `pbs_server`.

Each PBS complex has one primary server, and if the complex is configured for failover, a secondary server.

The server contains a licensing client which communicates with the licensing server for licensing PBS jobs.

PBS provides a default server; see "Default server" on page 5.

Server name

A server name is an ASCII character string. Format:

<hostname>[:<port number>]

The network routine `gethostbyname` is used to translate this to a network address. The network routine `getservbyname` is used to determine the port number. An alternate port number may be specified by appending a colon (":") and the port number to the hostname.

Shared resource

A vnode resource defined and managed at one vnode, but available for use at others.

Shrink-to-fit job

A job that requests the min_walltime resource. A shrink-to-fit job requests a running time in a specified range, where min_walltime is required, and max_walltime is not. PBS computes the actual walltime.

Sister

Any MoM that is not on the head or first host of a multihost job. A sister is directed by the primary execution host. Also called a *subordinate MoM*.

Sisterhood

All of the MoMs involved in running a particular job.

Site

A location which for our purposes uses (or will use) PBS. A site can employ one or more PBS complexes, each made up of any combination of hardware and software that PBS supports.

Snapshot Checkpoint

The checkpoint script or tool writes a restart file, and the job continues to execute. The job resumes from this start file if the system experiences a problem during the job's subsequent execution.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, it is the next occurrence.

Stage in

The process of moving one or more job-related files from a storage location to the execution host before running the job.

Stage out

The process of moving one or more job-related files from the execution host to a storage location after running the job.

Staging and execution directory

The *staging and execution directory* is a directory on the execution host where the following happens:

- Files are staged into this directory before execution
- The job runs in this directory
- Files are staged out from this directory after execution

A job-specific staging and execution directory can be created for each job, or PBS can use a specified directory, or a default directory. See "Staging and Execution Directories for Job" on page 523 in the PBS Professional Administrator's Guide.

Standing reservation

An advance reservation which recurs at specified times. For example, the user can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

Start reservation, Job-specific start reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS starts the job according to scheduling policy. When the job starts, PBS creates and starts the reservation, and PBS moves the job into the reservation. Created when you use `qsub -Wcreate_resv_from_job=true` on a queued job.

State

The PBS server, vnodes, reservations, and jobs can be in various states, depending on what PBS is doing. For example the server can be *idle* or *scheduling*, vnodes can be *busy* or *free*, and jobs can be *queued* or *running*, among other states. For a complete description of states, see "States" on page 361.

Strict ordering

A scheduling policy where jobs are run according to policy order. If the site-specified policy dictates a particular priority ordering for jobs, that is the order in which they are run. Strict ordering can be modified by backfilling in order to increase throughput. See "Backfilling".

Subject

A process belonging to a job run by an authorized, unprivileged user (a job submitter.)

Subjob

One of the jobs in a job array, e.g. `1234[7]`, where `1234[]` is the job array itself, and 7 is the index. Queued subjobs are not individually listed in the queue; only their job array is listed. Running subjobs are individually listed.

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

Subordinate MoM

Any MoM that is not on the head or first host of a multihost job. A subordinate MoM is directed by the primary execution host. Also called a *sister*.

Task

A process belonging to a job. A POSIX session started by MoM on behalf of a job.

Task placement

The process of choosing a set of vnodes to allocate to a job that will both satisfy the job's resource request (select and place specifications) and satisfy the configured scheduling policy.

Three-server configuration

One form of redundant license server configuration. Means that if any 2 of the 3 license servers are up and running (referred to as a quorum), the system is functional, with 1 server acting as master who can issue licenses. If the master goes down, another server must take over as master. See the *PBS Works Licensing Guide*.

TPP

TCP-based Packet Protocol. Protocol used by `pbs_comm`.

User

Has two meanings:

1. A person who submits jobs to PBS, as differentiated from Operators, Managers and administrators. See "User" on page 360 in the PBS Professional Administrator's Guide.
2. A system user, identified by a unique character string (the user name) and by a unique number (the user ID). Any person using the system has a username and user ID.

User access, Access by user

The specified user is allowed access at the server, queues, and reservations .

User ID, UID

A unique numeric identifier assigned to each user.

User limit

Refers to configurable limits on resources and jobs. A limit placed on one or more users, whether generic or individual.

Vchunk

The part of a chunk that is supplied by one vnode. If a chunk is broken up across multiple vnodes, each vnode supplies a vchunk.

Version 1 configuration file

MoM configuration file containing MoM configuration parameters. See Chapter 3, "MoM Parameters", on page 239.

Version 2 configuration file

Also called vnodedefs file. Vnode configuration file containing vnode attribute and resource settings. Created using `pbs_mom -s insert` command. See "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide.

Virtual processor, VP

PBS can treat a vnode as if it has more processors available than the number of physical processors. When resources_available.ncpus is set to a number higher than the actual number of physical processors, the vnode can be said to have virtual processors. Also called logical processors.

Vnode

A virtual node, or *vnode*, is an abstract object representing a host or a set of resources which form a usable part of an execution host. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes. Each vnode can be managed and scheduled independently. Each vnode in a complex must have a unique name. Vnodes on a host can share resources, such as node-locked licenses.

vnodedefs file

A Version 2 configuration file. Vnode configuration file containing vnode attribute and resource settings. Created using `pbs_mom -s insert` command. See "Version 2 Vnode Configuration Files" on page 42 in the PBS Professional Administrator's Guide.

vp

Virtual processor. The smallest unit of execution resources that can be specified to run a job. Cray refers to these as a CPU (aka a BASIL PE, an Intel thread or an AMD core).

Windows-Linux complex, Linux-Windows complex

A PBS complex with a Linux server/scheduler/comm host and Windows execution and client hosts.

2
PBS Commands

In this chapter, we describe each PBS command, including any options, operands, etc.

2.1 Our Command Notation

Optional Arguments

Optional arguments are enclosed in square brackets. For example, in the `qstat` man page, the `-E` option is shown this way:

qstat [-E]

To use this option, you would type:

```
qstat -E
```

Variable Arguments

Variable arguments (where you fill in the variable with the actual value) such as a job ID or vnode name are enclosed in angle brackets. Here's an example from the `pbsnodes` man page:

pbsnodes -v <vnode>

To use this command on a vnode named "my_vnode", you'd type:

```
pbsnodes -v my_vnode
```

Optional Variables

Optional variables are enclosed in angle brackets inside square brackets. In this example from the `qstat` man page, the job ID is optional:

qstat [<job ID>]

To query the job named "1234@my_server", you would type this:

```
qstat 1234@my_server
```

Literal Terms

Literal terms appear exactly as they should be used. For example, to get the version for a command, you type the command, then "--version". Here's the syntax:

qstat --version

And here's how you would use it:

```
qstat --version
```

2.2 Requirements for Commands

Some PBS commands require root privilege or PBS Operator or Manager privilege in order to run. Some can be executed by anyone, but the output depends upon the privilege of the user.

Most PBS commands require that the server be running; some require that MoMs be running.

The following table lists the commands, and indicates the permissions required to use each, and whether the server or MoM must be running.

Table 2-1: Permission and Daemon Requirements for Commands

Command	Action	Permission Required	Server Must Be Running?	MoM Must Be Running?
mpiexec	Runs MPI programs under PBS on Linux	Any	No	No
pbs	Start, stop, restart, or get the PIDs of PBS daemons	Root on Linux; Admin on Windows	No	No
pbsdsh	Distributes tasks to vnodes under PBS	Any	No	Yes
pbsfs	Show or manipulate PBS fair-share usage data	Any	Yes	No
pbsnodes	Query PBS host or vnode status, mark hosts free or offline, change the comment for a host, or output vnode information	Result depends on permission	Yes	No
pbsrun	General-purpose wrapper script for `mpirun`	Root or PBS administrator only	No	No
pbsrun_unwrap	Unwraps `mpirun`, reversing `pbsrun_wrap`	Root on Linux	No	No
pbsrun_wrap	General-purpose script for wrapping `mpirun` in `pbsrun`	Root on Linux	No	No
pbs_account	For Windows. Manage PBS service account	Admin on Windows	No	No
pbs_attach	Attaches a session ID to a PBS job	Any	Yes	Yes
pbs_comm	Starts the PBS communication daemon	Root on Linux	No	No
pbs_dataservice	Start, stop, or check the status of PBS data service	Root on Linux	No	No
pbs_ds_password	Sets or changes data service user account or its password	Root on Linux; Admin on Windows	No	No
pbs_hostn	Reports hostname and network address(es)	Any	No	No
pbs_idled	Runs PBS daemon that monitors the console and informs `pbs_mom` of idle time	Root or PBS administrator only	No	No
pbs_iff	Tests authentication with the server	Any; useful only to root	Yes	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Action	Permission Required	Server Must Be Running?	MoM Must Be Running?
pbs_interactive	For Windows. Register, unregister, or get the version of PBS_INTERACTIVE service	Administrator only	No	No
pbs_lamboot	PBS front end to LAM's lamboot program	Any	No	No
pbs_login	Caches encrypted user password for authentication	Any	No, for PBS service account Yes, for job submitters	No
pbs_mkdirs	For Windows. Create, or fix the permissions of, the directories and files used by PBS	PBS administrator only	No	No
pbs_mom	Runs the PBS job monitoring and execution daemon	Root on Linux; Admin on Windows	No	No
pbs_mpihp	Runs an MPI application in a PBS job with HP MPI	Any	Yes	Yes
pbs_mpilam	Runs MPI programs under PBS with LAM MPI	Any	Yes	Yes
pbs_mpirun	Runs MPI programs under PBS with MPICH	Any	Yes	Yes
pbs_probe	**Deprecated**. Reports PBS diagnostic information and fixes permission errors	Root or PBS administrator only	No	No
pbs_python	Python interpreter for debugging a hook script from the command line	Any	No	No
pbs_ralter	Modify an existing advance, standing, or job-specific reservation	Job owner or PBS administrator	Yes	No
pbs_rdel	Deletes a PBS advance, standing, or job-specific reservation	Any	Yes	No
pbs_release_nodes	Releases sister hosts or vnodes assigned to a PBS job	Job owner, PBS Manager, Operator, administrator, root on Linux, Admin on Windows	Yes	Yes
pbs_rstat	Shows status of PBS advance, standing, or job-specific reservations	Any	Yes	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Action	Permission Required	Server Must Be Running?	MoM Must Be Running?
pbs_rsub	Creates a PBS advance, standing, or job-specific reservation	Any	Yes	No
pbs_sched	Runs a PBS scheduler	Root on Linux	No	No
pbs_server	Starts a PBS batch server	Root on Linux	No	No
pbs_snapshot	Captures PBS data to be used for diagnostics	Root on Linux	Yes	No
pbs_tclsh	**Deprecated**. TCL shell with TCL-wrapped PBS API	Any	No	No
pbs_tmrsh	TM-enabled replacement for `rsh/ssh` for use by MPI implementations	Any	No	Yes
pbs_topologyinfo	Reports topological information	Root or Windows administrator only	No	No
pbs_wish	**Deprecated**. TK window shell with TCL-wrapped PBS API	Any	No	No
printjob	Prints job information	Root or Windows Administrator only	No	No
qalter	Alters a PBS job	Any	Yes	No
qdel	Deletes PBS jobs	Any	Yes	No
qdisable	Prevents a queue from accepting jobs	Manager or Operator only	Yes	No
qenable	Allows a queue to accept jobs	Manager or Operator only	Yes	No
qhold	Holds PBS batch jobs	Some holds can be set by Operator, Manager, root, or administrator only	Yes	No
qmgr	Administrator's command interface for managing PBS	Any	Yes	No
qmove	Moves a PBS job from one queue to another	Any; managers and operators can move jobs in some cases where unprivileged users cannot	Yes	No
qmsg	Writes message string into one or more job output files	Any	Yes	No
qorder	Swaps queue positions of two PBS jobs	Any	Yes	No

Table 2-1: Permission and Daemon Requirements for Commands

Command	Action	Permission Required	Server Must Be Running?	MoM Must Be Running?
qrerun	Requeues a PBS job	Manager or Operator only	Yes	No
qrls	Releases holds on PBS jobs	Some holds can be released by Operator, Manager, root, or administrator only	Yes	No
qrun	Runs a PBS job immediately	Operator or Manager only	Yes	No
qselect	Selects specified PBS jobs	Any	Yes	No
qsig	Send signal to PBS job	Operator or Manager required to send *admin-suspend*, *admin-resume*, *suspend*, and *resume*. Any privilege for other signals.	Yes	Yes
qstart	Turns on scheduling or routing for the jobs in a PBS queue	Operator or Manager only	Yes	No
qstat	Displays status of PBS jobs, queues, or servers	Result depends on permission	Yes	No
qstop	Prevents PBS jobs in the specified queue from being scheduled or routed	Operator or Manager only	Yes	No
qsub	Submits a job to PBS	Any	Yes	No
qterm	Terminates one or both PBS servers, and optionally terminates scheduler and/or MoMs	Operator or Manager only	Yes	No
tracejob	Extracts and prints log messages for a PBS job	Root or PBS administrator only	No	No
win_postinstall.py	For Windows. Configures PBS services	Administrator	No	No

2.2.1 Windows Requirements

Under Windows, use double quotes when specifying arguments to PBS commands.

2.3 mpiexec

Runs MPI programs under PBS on Linux

2.3.1 Synopsis

mpiexec

mpiexec --version

2.3.2 Description

The PBS `mpiexec` command provides the standard `mpiexec` interface on a system running supported versions of HPE MPI. If executed on a different system, it will assume it was invoked by mistake. In this case it will use the value of PBS_O_PATH to search for the correct `mpiexec`. If one is found, the PBS `mpiexec` will exec it.

The PBS `mpiexec` calls the HPE `mpirun(1)`.

It is transparent to the user; MPI jobs submitted outside of PBS run as they would normally. MPI jobs can be launched across multiple HPE systems. PBS will manage, track, and cleanly terminate multi-host MPI jobs. PBS users can run an MPI job within a specific partition.

If CSA has been configured and enabled, PBS will collect accounting information on all tasks launched by an MPI job. CSA information will be associated with the PBS job ID that invoked it, on each execution host.

If the PBS_MPI_DEBUG environment variable's value has a nonzero length, PBS writes debugging information to standard output.

2.3.3 Usage

The PBS `mpiexec` command presents the `mpiexec` interface described in section "4.1 Portable MPI Process Startup" of the "MPI-2: Extensions to the Message-Passing Interface" document in `http://www.mpifo-rum.org/docs/mpi-20-html/node42.htm`

2.3.4 Options

--version

The `mpiexec` command returns its PBS version information and exits. This option can only be used alone.

2.3.5 Requirements

- System running a supported version of HPE MPI.
- PBS uses HPE's `mpirun(1)` command to launch MPI jobs. HPE's `mpirun` must be in the standard location.
- The location of `pbs_attach()` on each vnode of a multi-vnode MPI job must be the same as it is on the primary execution host vnode.
- In order to run multihost jobs, the HPE Array Services must be correctly configured. HPE systems communicating via HPE's Array Services must all use the same version of the `sgi-arraysvcs` package. HPE systems communicating via HPE's Array Services must have been configured to interoperate with each other using the default array. See HPE's `array_services(5)` man page.

2.3.6 Environment Variables

PBS_ENVIRONMENT

The PBS_ENVIRONMENT environment variable is used to determine whether `mpiexec` is being called from within a PBS job.

PBS_MPI_DEBUG

The PBS `mpiexec` checks the PBS_MPI_DEBUG environment variable. If this variable has a nonzero length, debugging information is written.

PBS_O_PATH

The PBS `mpiexec` uses the value of PBS_O_PATH to search for the correct `mpiexec` if it was invoked by mistake.

2.3.7 Path

PBS' `mpiexec` is located in `PBS_EXEC/bin/mpiexec`.

2.3.8 See Also

The PBS Professional Administrator's Guide, "pbs_attach" on page 55

HPE man pages: HPE's `mpirun(1)`, HPE's `mpiexec_mpt(1)`, HPE's `array_services(5)`

2.4 pbs

Start, stop, restart, or get the PIDs of PBS daemons

2.4.1 Synopsis

pbs [start | stop | restart | status]

2.4.2 Description

The `pbs` command starts, stops or restarts all PBS daemons on the local machine, or reports the PIDs of all daemons when given the *status* argument. Does not affect other hosts.

You can start, stop, restart, or status the PBS daemons using the `systemctl` command; see "Starting & Stopping PBS on Linux" on page 159 in the PBS Professional Installation & Upgrade Guide.

2.4.2.1 Caveats

This command operates only on daemons that are marked as active in `pbs.conf`. For example, if PBS_START_MOM is set to *0* in the local `pbs.conf`, this command will not operate on `pbs_mom`, and will not start, stop, or restart `pbs_mom`.

This command is typically placed in `/etc/init.d` so that PBS starts up automatically.

2.4.2.2 Required Privilege

You need root privilege to use this command to start, stop, or restart PBS daemons.

A non-root user can use this command to get the PIDs of PBS daemons.

2.4.3 Arguments

restart
: All daemons on the local machine are stopped, then they are restarted. PBS reports the name of the license server and the number and type of licenses available.

start
: Each daemon on the local machine is started. PBS reports the number and type of licenses available, as well as the name of the license server. Any running jobs are killed.

status
: PBS reports the PID of each daemon on the local machine.

stop
: Each daemon on the local machine is stopped, and its PID is reported.

2.4.4 See Also

The PBS Professional Administrator's Guide, "pbs_comm" on page 57, "pbs_mom" on page 71, "pbs_sched" on page 105, "pbs_server" on page 108

2.5 pbsdsh

Distributes tasks to vnodes under PBS

2.5.1 Synopsis

pbsdsh [-c <copies>] [-s] [-v] [-o] -- <program> [<program args>]

pbsdsh [-n <vnode index>] [-s] [-v] [-o] -- <program >[<program args>]

pbsdsh --version

2.5.2 Description of `pbsdsh` Command

The `pbsdsh` command allows you to distribute and execute a task on each of the vnodes assigned to your job by executing (spawning) the application on each vnode. The `pbsdsh` command uses the PBS Task Manager, or TM, to distribute the program on the allocated vnodes.

When run without the `-c` or the `-n` option, `pbsdsh` will spawn the program on all vnodes allocated to the PBS job. The spawns take place concurrently; all execute at (about) the same time.

Note that the double dash must come after the options and before the program and arguments. The double dash is only required for Linux.

The `pbsdsh` command runs one task for each line in the $PBS_NODEFILE. Each MPI rank gets a single line in the $PBS_NODEFILE, so if you are running multiple MPI ranks on the same host, you still get multiple `pbsdsh` tasks on that host.

2.5.2.1 Example

The following example shows the `pbsdsh` command inside of a PBS batch job. The options indicate that the user wants `pbsdsh` to run the *myapp* program with one argument (*app-arg1*) on all four vnodes allocated to the job (i.e. the default behavior).

```
#!/bin/sh
#PBS -l select=4:ncpus=1
#PBS -l walltime=1:00:00

pbsdsh ./myapp app-arg1
```

2.5.3 Options to `pbsdsh` Command

-c <copies>

The program is spawned *copies* times on the vnodes allocated, one per vnode, unless *copies* is greater than the number of vnodes. If *copies* is greater than the number of vnodes, it wraps around, running multiple instances on some vnodes. This option is mutually exclusive with `-n`.

-n <vnode index>

The program is spawned only on a single vnode, which is the *vnode index*-th vnode allocated. This option is mutually exclusive with `-c`.

-o

No obit request is made for spawned tasks. The program does not wait for the tasks to finish.

-s
The program is run in turn on each vnode, one after the other.

-v
Produces verbose output about error conditions and task exit status.

--version
The `pbsdsh` command returns its PBS version information and exits. This option can only be used alone

2.5.4 Operands

program
The first operand, *program*, is the program to execute. The double dash must precede *program* under Linux.

program args
Additional operands, *program args*, are passed as arguments to the program.

2.5.5 Standard Error

The `pbsdsh` command writes a diagnostic message to standard error for each error occurrence.

2.5.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "qsub" on page 213, "TM Library Routines" on page 93 in the PBS Professional Programmer's Guide

2.6 pbsfs

Show or manipulate PBS fairshare usage data

2.6.1 Synopsis

Showing usage data:

pbsfs [-c <entity1> <entity2>] [-g <entity>] [-I <scheduler name>] [-p] [-t]

Manipulating usage data:

pbsfs [-d] [-e] [-I <scheduler name>] [-s <entity> <usage value>]

Printing version:

pbsfs --version

2.6.2 Description

You can use the `pbsfs` command to print or manipulate a PBS scheduler's fairshare usage data. You can print the usage data in various formats, described below. Changes made using `pbsfs` take effect in the next scheduling cycle; you do not need to restart or HUP a scheduler for changes to take effect.

We recommend that if you use the options that manipulate usage data, you should do this when a scheduler is not scheduling jobs, because scheduling while changing fairshare usage data may give unwanted results.

2.6.2.1 Prerequisites

The server must be running in order to use the `pbsfs` command.

2.6.2.2 Permissions

You must be root to run the `pbsfs` command; if not, it will print the error message, "`Unable to access fairshare data`".

2.6.3 Options to `pbsfs`

You can safely use the following options while jobs are being scheduled:

(no options)

Same as `pbsfs -p`.

-c <entity1> <entity2>

Compares two fairshare entities.

-g <entity>

Prints a detailed listing for the specified entity, including the path from the root of the tree to the entity.

-I <scheduler name>

Specifies name of scheduler whose data is to be manipulated or shown. Required for multischeds; optional for default scheduler. Name of default scheduler is "default". If not specified, `pbsfs` operates on default scheduler.

-p

Prints the fairshare tree as a table, showing for each internal and leaf vertex the group ID of the vertex's parent, group ID of the vertex, vertex shares, vertex usage, and percent of shares allotted to the vertex.

-t

Prints the fairshare tree in a hierarchical format.

--version

The pbsfs command returns its PBS version information and exits. This option can only be used alone.

It is not recommended to be scheduling jobs when you use the following options:

-d

Decays the fairshare tree by the amount specified in the fairshare_decay_factor scheduler parameter.

-e

Trims fairshare tree to just the entities in the resource_group file. Unknown entities and their usage are deleted; as a result the unknown group has no usage and no children.

-s <entity> <usage value>

Sets *entity*'s usage value to *usage value*. Editing a non-leaf entity is ignored. All non-leaf entity usage values are calculated each time you use the pbsfs command to make changes.

2.6.3.1 Output Formats for pbsfs

The pbsfs command can print output in three different formats:

pbsfs -g <entity>

Prints a detailed listing for the specified entity. Example:

```
pbsfs -g pbsuser3
fairshare entity: pbsuser3
Resgroup: 20
cresgroup: 22
Shares: 40
Percentage: 24.000000%
fairshare_tree_usage: 0.832973
usage: 1000 (cput)
usage/perc: 4167
Path from root:
TREEROOT  :     0       1201 / 1.000 = 1201
group2    :    20       1001 / 0.600 = 1668
pbsuser3  :    22       1000 / 0.240 = 4167
```

pbsfs,

pbsfs -p

Prints the entire tree as a table, with data in columns. Example:

```
pbsfs
Fairshare usage units are in: cput
TREEROOT  : Grp: -1     cgrp: 0    Shares: -1     Usage: 1201   Perc: 100.000%
group2    : Grp: 0      cgrp: 20   Shares: 60     Usage: 1001   Perc: 60.000%
pbsuser3  : Grp: 20     cgrp: 22   Shares: 40     Usage: 1000   Perc: 24.000%
pbsuser2  : Grp: 20     cgrp: 21   Shares: 60     Usage: 1      Perc: 36.000%
group1    : Grp: 0      cgrp: 10   Shares: 40     Usage: 201    Perc: 40.000%
pbsuser1  : Grp: 10     cgrp: 12   Shares: 50     Usage: 100    Perc: 20.000%
pbsuser   : Grp: 10     cgrp: 11   Shares: 50     Usage: 100    Perc: 20.000%
unknown   : Grp: 0      cgrp: 1    Shares: 0      Usage: 1      Perc: 0.000%
```

pbsfs -t

Prints the entire tree as a tree, showing group-child relationships. Example:

```
pbsfs -t
  TREEROOT(0)
      group2(20)
           pbsuser3(22)
           pbsuser2(21)
      group1(10)
           pbsuser1(12)
           pbsuser(11)
      unknown(1)
```

2.6.3.2 Data Output by pbsfs

cresgroup, cgrp

Group ID of the entity

fairshare entity

The specified fairshare tree entity

fairshare usage units

The resource for which a scheduler accumulates usage for fairshare calculations. This defaults to *cput* (CPU seconds) but can be set in a scheduler's configuration file.

fairshare_tree_usage

The entity's effective usage. See "Computing Effective Usage (fairshare_tree_usage)" on page 143 in the PBS Professional Administrator's Guide.

Path from root

The path from the root of the tree to the entity. A scheduler follows this path when comparing priority between two entities.

Percentage, perc

The percentage of the shares in the tree allotted to the entity, computed as fairshare_perc. See "Computing Target Usage for Each Vertex (fairshare_perc)" on page 143 in the PBS Professional Administrator's Guide.

Resgroup, Grp

Group ID of the entity's parent group

Shares
: The number of shares allotted to the entity

usage
: The amount of usage by the entity

usage/perc
: The value a scheduler uses to the pick which entity has priority over another. The smaller the number the higher the priority.

2.6.4 See Also

"Using Fairshare" on page 138 in the PBS Professional Administrator's Guide.

2.7 pbsnodes

Query PBS host or vnode status, mark hosts free or offline, change the comment for a host, or output vnode information

2.7.1 Synopsis

pbsnodes [-o | -r] [-s <server name>] [-C <comment>] <hostname> [<hostname> ...]

pbsnodes [-l] [-s <server name>]

pbsnodes -v <vnode> [<vnode> ...] [-s <server name>]

pbsnodes -a[v] [-S[j][L]] [-F json|dsv [-D <delimiter>]] [- s <server name>]

pbsnodes [-H] [-S[j][L]] [-F json|dsv [-D <delimiter>]] <hostname> [<hostname> ...]

pbsnodes --version

2.7.2 Description

The `pbsnodes` command is used to query the status of hosts or vnodes, to mark hosts *FREE* or *OFFLINE*, to edit a host's comment attribute, or to output vnode information. The `pbsnodes` command obtains host information by sending a request to the PBS server.

2.7.2.1 Using pbsnodes

To list all vnodes:

```
pbsnodes -av
```

To print the status of the specified host or hosts, run `pbsnodes` with no options (except the `-s` option) and with a list of hosts.

To print the command usage, run `pbsnodes` with no options and without a list of hosts.

To remove a vnode from the scheduling pool, mark it *OFFLINE*. If it is marked *DOWN*, when the server next queries the MoM, and can connect, the vnode will be marked *FREE*.

To offline a single vnode in a multi-vnoded system, use:

```
qmgr -c "set node <vnode name> state=offline"
```

2.7.2.2 Output

The order in which hosts or vnodes are listed in the output of the `pbsnodes` command is undefined. Do not rely on output being ordered.

If you print attributes, `pbsnodes` prints out only those attributes which are not at default values.

2.7.2.3 Permissions

PBS Manager or Operator privilege is required to execute `pbsnodes` with the `-o` or `-r` options, to view custom resources which have been created to be invisible to users, and to see some output such as PBS version.

2.7.3 Options to pbsnodes

(no options)

If neither options nor a host list is given, the pbsnodes command prints usage syntax.

-a

Lists all hosts and all their attributes (available and used.)

When used with the -v option, lists all vnodes.

When listing a host with multiple vnodes:

The output for the jobs attribute lists all the jobs on all the vnodes on that host. Jobs that run on more than one vnode will appear once for each vnode they run on.

For consumable resources, the output for each resource is the sum of that resource across all vnodes on that host.

For all other resources, e.g. string and Boolean, if the value of that resource is the same on all vnodes on that host, the value is returned. Otherwise the output is the literal string "<various>".

-C <comment>

Sets the comment attribute for the specified host(s) to the value of *comment*. Comments containing spaces must be quoted. The comment string is limited to 80 characters. Usage:

```
pbsnodes -C <comment> <hostname> [<hostname> ...]
```

To set the comment for a vnode:

```
qmgr -c "s n <vnode name> comment=<comment>"
```

-F dsv [-D <delimiter>]

Prints output in delimiter-separated value format. Optional delimiter specification. Default delimiter is vertical bar ("|").

-F json

Prints output in JSON format.

-H <hostname> [<hostname> ...]

Prints all non-default-valued attributes for specified hosts and all vnodes on specified hosts.

-j

Displays the following job-related headers for specified vnodes:

Table 2-2: Output for -j Option

Header	Width	Description
vnode	15	Vnode name
state	15	Vnode state
njobs	6	Number of jobs on vnode
run	5	Number of running jobs at vnode
susp	6	Number of suspended jobs at vnode
mem f/t	12	Vnode memory free/total
ncpus f/t	7	Number of CPUs at vnode free/total
nmics f/t	7	Number of MICs free/total
ngpus f/t	7	Number of GPUs at vnode free/total
jobs	No restriction	List of job IDs on vnode

Note that nmics is a custom resource that must be created by the administrator if you want it displayed here.

Each subjob is treated as a unique job.

-L

Displays output with no restrictions on column width.

-l

Lists all hosts marked as *DOWN* or *OFFLINE*. Each such host's state and comment attribute (if set) is listed. If a host also has state *STATE-UNKNOWN*, it is listed. For hosts with multiple vnodes, only hosts where all vnodes are marked as *DOWN* or *OFFLINE* are listed.

-o <hostname> [<hostname> ...]

Marks listed hosts as *OFFLINE* even if currently in use. This is different from being marked *DOWN*. A host that is marked *OFFLINE* continues to execute the jobs already on it, but is removed from the scheduling pool (no more jobs are scheduled on it.)

For hosts with multiple vnodes, `pbsnodes` operates on a host and all of its vnodes, where the hostname is resources_available.host, which is the name of the parent vnode.

To offline all vnodes on a multi-vnoded machine:

```
pbsnodes -o <name of parent vnode>
```

To offline a single vnode on a multi-vnoded system, use:

```
Qmgr: qmgr -c "set node <vnode name> state=offline"
```

Requires PBS Manager or Operator privilege.

-r <hostname> [<hostname> ...]

Clears *OFFLINE* from listed hosts.

-S

Displays the following vnode information:

Table 2-3: Output for -S Option

Header	Width	Description
name	15	Vnode name
state	15	Vnode state
OS	8	Value of OS custom resource, if any
hardware	8	Value of hardware custom resource, if any
host	15	Hostname
queue	10	Value of vnode's queue attribute
ncpus	7	Number of CPUs at vnode
nmics	7	Number of MICs at vnode
mem	8	Vnode memory
ngpus	7	Number of GPUs at vnode
comment	No restriction	Vnode comment

Note that nmics and OS are custom resources that must be created by the administrator if you want their values displayed here.

-s <server name>

Specifies the PBS server to which to connect.

-v [<vnode> [<vnode> ...]]

Lists all non-default-valued attributes for each specified vnode.

With no arguments, prints one entry for each vnode in the PBS complex.

With one or more vnodes specified, prints one entry for each specified vnode.

When used with `-a`, lists all vnodes.

--version

The `pbsnodes` command returns its PBS version information and exits. This option can only be used alone.

2.7.4 Operands

<server name>

Specifies the server to which to connect. Default: default server.

<hostname> [<hostname> ...]

Specifies the host(s) to be queried or operated on.

<vnode> [<vnode> ...]

Specifies the vnode(s) to be queried or operated on.

2.7.5 Exit Status

Zero

Success

Greater than zero

- Incorrect operands are given
- `pbsnodes` cannot connect to the server
- There is an error querying the server for the vnodes

2.7.6 See Also

The PBS Professional Administrator's Guide, "qmgr" on page 149

2.8 pbsrun

General-purpose wrapper script for `mpirun`

2.8.1 Synopsis

pbsrun

pbsrun --version

2.8.2 Description

`pbsrun` is a wrapper script for any of several versions of `mpirun`. This provides a user-transparent way for PBS to control jobs which call `mpirun` in their job scripts. The `pbsrun_wrap` script instantiates `pbsrun` so that the wrapper script for the specific version of `mpirun` being used has the same name as that version of `mpirun`.

If the `mpirun` wrapper script is run inside a PBS job, it translates any `mpirun` call of the form:

mpirun [<options>] <executable> [<args>]

into

mpirun [<options>] pbs_attach [<special options to pbs_attach>] <executable> [<args>]

where *special options* refers to any option needed by `pbs_attach` to do its job (e.g. *-j $PBS_JOBID*).

If the wrapper script is executed outside of PBS, a warning is issued about "not running under PBS", but it proceeds as if the actual program had been called in standalone fashion.

The `pbsrun` wrapper script is not meant to be executed directly; instead it is instantiated by `pbsrun_wrap`. It is copied to the target directory and renamed "`pbsrun.<mpirun version/flavor>`" where *mpirun version/flavor* is a string that identifies the `mpirun` version being wrapped (e.g. *ch_gm*).

The `pbsrun` script, if executed inside a PBS job, runs an initialization script, named `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init`, then parses `mpirun`-like arguments from the command line, sorting which options and option values to retain, to ignore, or to transform, before calling the actual `mpirun` script with a "pbs_attach" prefixed to the executable. The actual `mpirun` to call is found by tracing the link pointed to by `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link`.

For all of the wrapped MPIs, the maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

The wrapped MPIs are:

- MPICH-GM's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh` (The wrapper is **deprecated** as of 14.2.1)
- MPICH-MX's `mpirun` (`mpirun.ch_mx`) with `rsh/ssh` (The wrapper is **deprecated** as of 14.2.1)
- MPICH-GM's `mpirun` (`mpirun.mpd`) with MPD (The wrapper is **deprecated** as of 14.2.1)
- MPICH-MX's `mpirun` (`mpirun.mpd`) with MPD (The wrapper is **deprecated** as of 14.2.1)
- MPICH2's `mpirun`
- Intel MPI's `mpirun` (The wrapper is **deprecated** as of 13.0)
- MVAPICH1's `mpirun` (The wrapper is **deprecated** as of 14.2.1)
- MVAPICH2's `mpiexec`

2.8.3 Options

--version

The pbsrun command returns its PBS version information and exits. This option can only be used alone.

2.8.4 Initialization Script

The initialization script, called $PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init, where *mpirun version/flavor* reflects the mpirun flavor/version being wrapped, can be modified by an administrator to customize against the local flavor/version of mpirun being wrapped.

Inside this sourced init script, 8 variables are set:

```
options_to_retain="-optA -optB <val> -optC <val1> val2> ..."
options_to_ignore="-optD -optE <n> -optF <val1> val2> ..."
options_to_transform="-optG -optH <val> -optI <val1> val2> ..."
options_to_fail="-optY -optZ ..."
options_to_configfile="-optX <val> ..."
options_with_another_form="-optW <val> ..."
pbs_attach=pbs_attach
options_to_pbs_attach="-J $PBS_JOBID"
```

2.8.4.1 Initialization Script Options

options_to_retain

Space-separated list of options and values that pbsrun.<mpirun version/flavor> passes on to the actual mpirun call. Options must begin with "-" or "--", and option arguments must be specified by some arbitrary name with left and right arrows, as in "<val1>".

options_to_ignore

Space-separated list of options and values that pbsrun.<mpirun version/flavor> does not pass on to the actual mpirun call. Options must begin with "-" or "--", and option arguments must be specified by arbitrary names with left and right arrows, as in "<n>".

options_to_transform

Space-separated list of options and values that pbsrun modifies before passing on to the actual mpirun call.

options_to_fail

Space-separated list of options that will cause pbsrun to exit upon encountering a match.

options_to_configfile

Single option and value that refers to the name of the configuration file containing command line segments found in certain versions of mpirun.

options_with_another_form

Space-separated list of options and values that can be found in *options_to_retain*, *options_to_ignore*, or *options_to_transform*, whose syntax has an alternate, unsupported form.

pbs_attach

Path to pbs_attach, which is called before the *executable* argument of mpirun.

options_to_pbs_attach

Special options to pass to the pbs_attach call. You may pass variable references (e.g. *$PBS_JOBID*) and they are substituted by pbsrun to actual values.

If pbsrun encounters any option not found in *options_to_retain*, *options_to_ignore*, and *options_to_transform*, it is flagged as an error.

These functions are created inside the init script. These can be modified by the PBS administrator.

```
transform_action () {
# passed actual values of $options_to_transform
args=$*
}


boot_action () {
mpirun_location=$1
}


evaluate_options_action () {
# passed actual values of transformed options
args=$*
}


configfile_cmdline_action () {
args=$*
}

end_action () {
mpirun_location=$1
}
```

transform_action()

The pbsrun.<mpirun version/flavor> wrapper script invokes the function transform_action() (called once on each matched item and value) with actual options and values received matching one of the *options_to_transform*. The function returns a string to pass on to the actual mpirun call.

boot_action()

Performs any initialization tasks needed before running the actual mpirun call. For instance, GM's MPD requires the MPD daemons to be user-started first. This function is called by the pbsrun.<mpirun version/flavor> script with the location of actual mpirun passed as the first argument. Also, the pbsrun.<mpirun version/flavor> checks for the exit value of this function to determine whether or not to progress to the next step.

evaluate_options_action()

Called with the actual options and values that resulted after consulting *options_to_retain*, *options_to_ignore*, *options_to_transform*, and executing transform_action(). This provides one more chance for the script writer to evaluate all the options and values in general, and make any necessary adjustments, before passing them on to the actual mpirun call. For instance, this function can specify what the default value is for a missing -np option.

configfile_cmdline_action()

Returns the actual options and values to be put in before the *option_to_configfile* parameter.

configfile_firstline_action()

Returns the item that is put in the first line of the configuration file specified in the *option_to_configfile* parameter.

end_action()

Called by `pbsrun.<mpirun version/flavor>` at the end of execution. It undoes any action done by transform_action(), such as cleanup of temporary files. It is also called when `pbsrun.<mpirun version/flavor>` is prematurely killed. This function is called with the location of actual `mpirun` passed as first argument.

The actual `mpirun` program to call is the path pointed to by `$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link`.

2.8.4.2 Modifying *.init Scripts

In order for administrators to modify `*.init` scripts without breaking package verification in RPM, master copies of the initialization scripts are named `*.init.in`. `pbsrun_wrap` instantiates the `*.init.in` files as `*.init`. For instance, `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init.in` is the master copy, and `pbsrun_wrap` instantiates it as `$PBS_EXEC/lib/MPI/pbsrun.mpich2.init`. `pbsrun_unwrap` takes care of removing the `*.init` files.

2.8.5 Versions/Flavors of `mpirun`

2.8.5.1 MPICH-GM `mpirun` (`mpirun.ch_gm`) with `rsh/ssh`: `pbsrun.ch_gm`

2.8.5.1.i Syntax

pbsrun.ch_gm <options> <executable> <arg1> <arg2> ... <argn>

Deprecated. The PBS wrapper script to MPICH-GM's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh` process startup method is named `pbsrun.ch_gm`.

If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by `rsh/ssh` so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` were used.

2.8.5.1.ii Options Handling

If executed inside a PBS job script, all `mpirun.ch_gm` options given are passed on to the actual `mpirun` call with these exceptions:

-machinefile <file>

The *file* argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to the user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.8.5.1.iii Wrap/Unwrap

To wrap MPICH-GM's `mpirun` script:

```
# pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.ch_gm pbsrun.ch_gm
```

To unwrap MPICH-GM's `mpirun` script:

```
# pbsrun_unwrap pbsrun.ch_gm
```

2.8.5.2 MPICH-MX `mpirun` (`mpirun.ch_mx`) with `rsh/ssh`: `pbsrun.ch_mx`

2.8.5.2.i Syntax

pbsrun.ch_mx <options> <executable> <arg1> <arg2> ... <argn>

The wrapper is **deprecated**. The PBS wrapper script to MPICH-MX's `mpirun` (`mpirun.ch_gm`) with `rsh/ssh` process startup method is named `pbsrun.ch_mx`.

If executed inside a PBS job, this allows PBS to track all MPICH-MX processes started by `rsh/ssh` so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_mx` were used.

2.8.5.2.ii Options Handling

If executed inside a PBS job script, all `mpirun.ch_gm` options given are passed on to the actual `mpirun` call with some exceptions:

-machinefile <file>

The *file* argument contents is ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to the user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.8.5.2.iii Wrap/Unwrap

To wrap MPICH-MX's `mpirun` script:

```
# pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.ch_mx pbsrun.ch_mx
```

To unwrap MPICH-MX's `mpirun` script:

```
# pbsrun_unwrap pbsrun.ch_mx
```

2.8.5.3 MPICH-GM `mpirun` (`mpirun.mpd`) with MPD: `pbsrun.gm_mpd`

2.8.5.3.i Syntax

pbsrun.gm_mpd <options> <executable> <arg1> <arg2> ... <argn>

The wrapper is **deprecated**. The PBS wrapper script to MPICH-GM's `mpirun` (`mpirun.ch_gm`) with MPD process startup method is called `pbsrun.gm_mpd`.

If executed inside a PBS job, this allows PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_gm` with MPD were used.

2.8.5.3.ii Options Handling

If executed inside a PBS job script, all `mpirun.ch_gm` with MPD options given are passed on to the actual `mpirun` call with these exceptions:

-m <file>

The *file* argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to the user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.8.5.3.iii Startup/Shutdown

The script starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either `rsh` or `ssh` based on the value of the environment variable RSHCOMMAND. The default is `rsh`.

The script also takes care of shutting down the MPD daemons at the end of a run.

2.8.5.3.iv Wrap/Unwrap

To wrap MPICH-GM's `mpirun` script with MPD:

```
# pbsrun_wrap [MPICH-GM_BIN_PATH]/mpirun.mpd pbsrun.gm_mpd
```

To unwrap MPICH-GM's `mpirun` script with MPD:

```
# pbsrun_unwrap pbsrun.gm_mpd
```

2.8.5.4 MPICH-MX `mpirun` (`mpirun.mpd`) with MPD: `pbsrun.mx_mpd`

2.8.5.4.i Syntax

pbsrun.mx_mpd <options> <executable> <arg1> <arg2> ... <argn>

The wrapper is **deprecated**. The PBS wrapper script to MPICH-MX's `mpirun` (`mpirun.ch_mx`) with MPD process startup method is called `pbsrun.mx_mpd`.

If executed inside a PBS job, this allows PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpirun.ch_mx` with MPD were used.

2.8.5.4.ii Options Handling

If executed inside a PBS job script, all `mpirun.mx_mpd` with MPD options given are passed on to the actual `mpirun` call with these exceptions:

-m <file>

The *file* argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to the user to make sure only PBS hosts are specified in the process group file; MPI processes spawned are not guaranteed to be under the control of PBS.

2.8.5.4.iii Startup/Shutdown

The script starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either `rsh` or `ssh` , based on the value of the environment variable RSHCOMMAND. The default is `rsh`.

The script also takes care of shutting down the MPD daemons at the end of a run.

2.8.5.4.iv Wrap/Unwrap

To wrap MPICH-MX's `mpirun` script with MPD:

```
# pbsrun_wrap [MPICH-MX_BIN_PATH]/mpirun.mpd pbsrun.mx_mpd
```

To unwrap MPICH-MX's `mpirun` script with MPD:

```
# pbsrun_unwrap pbsrun.mx_mpd
```

2.8.5.5 MPICH2 `mpirun`: `pbsrun.mpich2`

2.8.5.5.i Syntax

pbsrun.mpich2 [<global args>] [<local args>] <executable> [<args>] [: [<local args>] <executable> [<args>]]

- or -

pbsrun.mpich2 -configfile <configfile>

where *configfile* contains command line segments as lines:

```
[local args] executable1 [args]
[local args] executable2 [args]
[local args] executable3 [args]
```

The PBS wrapper script to MPICH2's `mpirun` is called `pbsrun.mpich2`.

If executed inside a PBS job, this allows PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard MPICH2's `mpirun` were used.

2.8.5.5.ii Options Handling

If executed inside a PBS job script, all MPICH2's `mpirun` options given are passed on to the actual `mpirun` call with these exceptions:

-host and -ghost

For specifying the execution host to run on. Not passed on to the actual `mpirun` call.

-machinefile <file>

The *file* argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

MPICH2's mpirun -localonly <num processes>

For specifying number of processes to run locally. Not supported. The user is advised instead to use the equivalent arguments: `-np <num processes> -localonly`. The reason for this is that the `pbsrun` wrapper script cannot handle a variable number of arguments to an option (e.g. "`-localonly`" has one argument and "`-localonly <num processes>`" has two arguments).

-np

If the user does not specify the `-np` option, no default value is provided by the PBS wrapper scripts. It is up to the local `mpirun` to decide what the reasonable default value should be, which is usually *1*.

2.8.5.5.iii Startup/Shutdown

The script takes care of ensuring that the MPD daemons on each of the hosts listed in $PBS_NODEFILE are started. It also takes care of ensuring that the MPD daemons have been shut down at the end of MPI job execution.

2.8.5.5.iv Wrap/Unwrap

To wrap MPICH2's `mpirun` script:

```
# pbsrun_wrap [<MPICH2 BIN PATH>]/mpirun pbsrun.mpich2
```

To unwrap MPICH2's `mpirun` script:

```
# pbsrun_unwrap pbsrun.mpich2
```

In the case where MPICH2 uses `mpirun.py`, run `pbsrun_wrap` on `mpirun.py` itself.

2.8.5.6 Intel MPI `mpirun`: `pbsrun.intelmpi`

Wrapping Intel MPI, and support for `mpdboot`, are **deprecated**.

2.8.5.6.i Syntax

pbsrun.intelmpi [<mpdboot options>] [<mpiexec options>] <executable> [<prog args>] [: [<mpiexec options>] <executable> [<prog args>]]

- or -

pbsrun.intelmpi [<mpdboot options>] -f <configfile>

where *mpdboot options* are any options to pass to the `mpdboot` program, which is automatically called by Intel MPI's `mpirun` to start MPDs, and *configfile* contains command line segments as lines.

The PBS wrapper script to Intel MPI's `mpirun` is called `pbsrun.intelmpi`.

If executed inside a PBS job, this allows PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard Intel MPI's `mpirun` were used.

2.8.5.6.ii Options Handling

If executed inside a PBS job script, all of the options to the PBS interface to Intel MPI's `mpirun` are passed to the actual `mpirun` call with these exceptions:

-host and -ghost
: For specifying the execution host to run on. Not passed on to the actual `mpirun` call.

-machinefile <file>
: The *file* argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

mpdboot options --totalnum=* and --file=*
: Ignored and replaced by the number of unique entries in $PBS_NODEFILE and name of $PBS_NODEFILE respectively.

arguments to mpdboot options --file=* and -f <mpd_hosts_file>
: Replaced by $PBS_NODEFILE.

-s
: If `pbsrun.intelmpi` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option `-s <spec>`. The user can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by `pbsrun.intelmpi` upon encountering a `-s` option telling users of the supported form.

-np

If the user does not specify the -np option, no default value is provided by the PBS wrap scripts. It is up to the local mpirun to decide what the reasonable default value should be, which is usually *1*.

2.8.5.6.iii Startup/Shutdown

Intel MPI's mpirun itself takes care of starting/stopping the MPD daemons. pbsrun.intelmpi always passes the arguments -totalnum=<number of mpds to start> and -file=<mpd_hosts_file> to the actual mpirun, taking its input from unique entries in $PBS_NODEFILE.

2.8.5.6.iv Wrap/Unwrap

To wrap Intel MPI's mpirun script:

```
# pbsrun_wrap [INTEL_MPI_BIN_PATH]/mpirun pbsrun.intelmpi
```

To unwrap Intel MPI's mpirun script:

```
# pbsrun_unwrap pbsrun.intelmpi
```

2.8.5.7 MVAPICH1 mpirun: pbsrun.mvapich1

2.8.5.7.i Syntax

pbsrun.mvapich1 <mpirun options> <executable> <options>

The wrapper is **deprecated**. The PBS wrapper script to MVAPICH1's mpirun is called pbsrun.mvapich1.

Only one executable can be specified. MVAPICH1 allows the use of InfiniBand.

If executed inside a PBS job, this allows PBS to be aware of all MVAPICH1 ranks and to track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard mpirun were used.

2.8.5.7.ii Options Handling

If executed inside a PBS job script, all mpirun options given are passed on to the actual mpirun call with these exceptions:

-map <list>

The map option is ignored.

-exclude <list>

The exclude option is ignored.

-machinefile <file>

The machinefile option is ignored.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used.

2.8.5.7.iii Wrap/Unwrap

To wrap MVAPICH1's mpirun script:

```
# pbsrun_wrap <path-to-actual-mpirun> pbsrun.mvapich1
```

To unwrap MVAPICH1's mpirun script:

```
# pbsrun_unwrap pbsrun.mvapich1
```

2.8.5.8 MVAPICH2 `mpiexec: pbsrun.mvapich2`

2.8.5.8.i Syntax

pbsrun.mvapich2 <mpiexec args> <executable> <executable's args> [: <mpiexec args> <executable> <executable's args>]

The PBS wrapper script to MVAPICH2's `mpiexec` is called `pbsrun.mvapich2`.

Multiple executables can be specified using the colon notation. MVAPICH2 allows the use of InfiniBand.

If executed inside a PBS job, this allows PBS to be aware of all MVAPICH2 ranks and to track their resources, so that PBS can perform accounting and have complete job control.

If executed outside of a PBS job, it behaves exactly as if standard `mpiexec` were used.

2.8.5.8.ii Options Handling

If executed inside a PBS job script, all `mpiexec` options given are passed on to the actual `mpiexec` call with these exceptions:

-host <hostname>

The *hostname* argument contents are ignored.

-machinefile <file>

The *file* argument contents are ignored and replaced by the contents of the $PBS_NODEFILE.

2.8.5.8.iii Wrap/Unwrap

To wrap MVAPICH2's `mpiexec` script:

```
# pbsrun_wrap <path-to-actual-mpiexec> pbsrun.mvapich2
```

To unwrap MVAPICH2's `mpiexec` script:

```
# pbsrun_unwrap pbsrun.mvapich2
```

2.8.6 Requirements

The `mpirun` being wrapped must be installed and working on all the vnodes in the PBS cluster.

2.8.7 Errors

If `pbsrun` encounters any option not found in *options_to_retain*, *options_to_ignore*, and *options_to_transform*, it is flagged as an error.

2.8.8 See Also

The PBS Professional Administrator's Guide, "pbs_attach" on page 55, "pbsrun_wrap" on page 51, "pbsrun_unwrap" on page 50

2.9 pbsrun_unwrap

Unwraps `mpirun`, reversing `pbsrun_wrap`

2.9.1 Synopsis

pbsrun_unwrap pbsrun.<mpirun version/flavor>

pbsrun_unwrap --version

2.9.2 Description

The `pbsrun_unwrap` script is used to reverse the actions of the `pbsrun_wrap` script.

Use `pbsrun_wrap` to wrap `mpirun`.

Using `pbsrun_unwrap` for Intel MPI is **deprecated** as of 13.0.

2.9.2.1 Syntax

pbsrun_unwrap pbsrun.<mpirun version/flavor>

For example, running the following:

```
pbsrun_unwrap pbsrun.ch_gm
```

causes the following actions:

1. Checks for a link in `$PBS_EXEC/lib/MPI/pbsrun.ch_gm.link`; If one exists, get the pathname it points to, for example:

   ```
   /opt/mpich-gm/bin/mpirun.ch_gm.actual
   ```

2. rm $PBS_EXEC/lib/MPI/pbsrun.mpirun.ch_gm.link
3. rm /opt/mpich-gm/bin/mpirun.ch_gm
4. rm $PBS_EXEC/bin/pbsrun.ch_gm
5. mv /opt/mpich-gm/bin/mpirun.ch_gm.actual /opt/mpich-gm/bin/mpirun.ch_gm

2.9.3 Options

--version

The `pbsrun_unwrap` command returns its PBS version information and exits. This option can only be used alone.

2.9.4 See Also

The PBS Professional Administrator's Guide, "pbs_attach" on page 55, "pbsrun" on page 40, "pbsrun_wrap" on page 51

2.10 pbsrun_wrap

General-purpose script for wrapping mpirun in pbsrun

2.10.1 Synopsis

pbsrun_wrap [-s] <path to actual mpirun> pbsrun.<mpirun version/flavor>

pbsrun_wrap --version

2.10.2 Description

The pbsrun_wrap script is used to wrap any of several versions of mpirun in pbsrun. The pbsrun_wrap script creates a symbolic link with the same path and name as the mpirun being wrapped. This calls pbsrun, which uses pbs_attach to give MoM control of jobs. The result is transparent to the user; when mpirun is called from inside a PBS job, PBS can monitor and control the job, but when mpirun is called from outside of a PBS job, it behaves as it would normally. See "pbs_attach" on page 55 and "pbsrun" on page 40.

Use pbsrun_unwrap to reverse the process.

Using pbsrun_wrap for Intel MPI is **deprecated** as of 13.0.

Available only under Linux.

2.10.2.1 Syntax

pbsrun_wrap [-s] <path to actual mpirun> pbsrun.<mpirun version/flavor>

Any mpirun version/flavor that can be wrapped has an initialization script ending in ".init", found in $PBS_EXEC/lib/MPI:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.init
```

The pbsrun_wrap script instantiates the pbsrun wrapper script as pbsrun.<mpirun version/flavor> in the same directory where pbsrun is located, and sets up the link to actual mpirun call via the symbolic link:

```
$PBS_EXEC/lib/MPI/pbsrun.<mpirun version/flavor>.link
```

For example, running:

```
pbsrun_wrap /opt/mpich-gm/bin/mpirun.ch_gm pbsrun.ch_gm
```

causes the following actions:

1. Save original mpirun.ch_gm script:
   mv /opt/mpich-gm/bin/mpirun.ch_gm /opt/mpich/gm/bin/mpirun.ch_gm.actual
2. Instantiate pbsrun wrapper script as pbsrun.ch_gm:
   cp $PBS_EXEC/bin/pbsrun $PBS_EXEC/bin/pbsrun.ch_gm
3. Link "mpirun.ch_gm" to actually call "pbsrun.ch_gm":
   ln -s $PBS_EXEC/bin/pbsrun.ch_gm /opt/mpich-gm/bin/mpirun.ch_gm
4. Create a link so that "pbsrun.ch_gm" calls "mpirun.ch_gm.actual":
   ln -s /opt/mpich-gm/bin/mpirun.ch_gm.actual $PBS_EXEC/lib/MPI/pbsrun.ch_gm.link

2.10.3 Options

-s

Sets the "strict_pbs" options in the various initialization scripts (e.g. pbsrun.bgl.init, pbsrun.ch_gm.init, etc...) to *1* from the default *0*. This means that the mpirun being wrapped by pbsrun will only be executed if inside a PBS environment. Otherwise, the user gets the error:

```
Not running under PBS exiting since strict_pbs is enabled; execute only in PBS
```

--version

The pbsrun_wrap command returns its PBS version information and exits. This option can only be used alone.

2.10.4 Requirements

The mpirun being wrapped must be installed and working on all the vnodes in the PBS complex.

2.10.5 See Also

The PBS Professional Administrator's Guide, "pbs_attach" on page 55, "pbsrun" on page 40, "pbsrun_unwrap" on page 50

2.11 pbs_account

For Windows. Manage PBS service account

2.11.1 Synopsis

pbs_account [-a <PBS service account name>] [-c [<password>]] [--ci] [--instid <instance ID>] [-o <output path>] [-p [<password>]] [--reg <service path>] [-s] [--unreg <service path>]

2.11.2 Description

The `pbs_account` command is used to manage the PBS service account. It is used to create the account, set or validate the account password, add privileges to the account, and register or unregister the account with the SCM.

2.11.2.1 Permissions

This command can be run by administrators only.

2.11.2.2 Platforms

This command is available on Windows only.

2.11.2.3 Caveats

Using `pbs_account --unreg` and `pbs_account --reg` stops and restarts MoM, which can kill jobs.

2.11.3 Options

-a <account name>

Specifies service account name.

-c [<password>]

- If specified account does not exist, creates the account with the password.
- If specified account exists, validates password against it.

Gives necessary privileges to the specified account: *Create Token Object*, *Replace Process Level Token*, *Log on as a Service*, and *Act as Part of the Operating System*

If password is not specified, user is prompted for password.

--ci

Informational only. Prints actions taken by `pbs_account` while creating PBS service account when operations are performed.

--instid <instance ID>

Specifies the instance ID when registering or unregistering multiple instances of a service. Example:

```
pbs_account --reg "C:\Program Files (x86)\PBS Pro_2\exec\sbin\pbs_mom" --instid 2 -a <username>
    -p <password>

pbs_account --unreg "C:\Program Files (x86)\PBS Pro_2\exec\sbin\pbs_mom" --instid 2
```

-o <output path>

Prints `stdout` and `stderr` messages in specified output path.

-p [<password >]

Updates the PBS service account password. If no password is specified, the user is prompted for a password.

--reg <path to service>

Registers the PBS service with the SCM, instructing it to run the services under the PBS service account. *path to service* must be in double quotes. Restarts MoM.

-s

Adds necessary privileges to the PBS service account. Grants the "Create Token Object", "Replace Process Level Token", "Log On as a Service", and "Act as Part of the Operating System" privileges to PBS service account.

--unreg <path to service>

Unregisters the PBS service with the SCM. *path to service* must be in double quotes. Stops MoM.

(no options)

Prints name of PBS service account, if it exists. Exit value is 0.

2.11.4 Examples

Example 2-1: To create the PBS service account:

```
pbs_account -c -s -p <password>
```

Example 2-2: To change the PBS service account:

```
pbs_account --reg <service path> -a <PBS service account name>
```

Example 2-3: To register the MoM service:

```
pbs_account --reg "\Program Files\PBS\exec\sbin\pbs_mom.exe" -p <password>
```

2.11.5 Exit Value

Zero

Upon success

2.12 pbs_attach

Attaches a session ID to a PBS job

2.12.1 Synopsis

Linux

pbs_attach [-j <job ID>] [-m <port number>] -p <PID>

pbs_attach [-j <job ID>] [-m <port number>] [-P] [-s] <cmd> [<arg> ...]

pbs_attach --version

Windows

pbs_attach [-c <path to script>] [-j <job ID>] [-m <port number>] -p <PID>

pbs_attach [-c <path to script>] [-j <job ID>] [-m <port number>] [-P] [-s] <cmd> [<arg> ...]

pbs_attach --version

2.12.2 Description

The `pbs_attach` command associates the processes in a session with a PBS job by attaching the session ID to the job. This allows PBS MoM to monitor and control those processes.

MoM uses process IDs to determine session IDs, which are put into MoM's task list (attached to the job.) All process IDs in a session are then associated with the job.

When a command *cmd* is given as an operand, the `pbs_attach` process becomes the parent process of *cmd*, and the session ID of `pbs_attach` is attached to the job.

2.12.3 Options to `pbs_attach`

-c <path to script>

Windows only. Specified command is invoked using a new command shell. In order to spawn and attach built-in DOS commands such as `set` or `echo`, it is necessary to open the task using a `cmd` shell. The new command shell, `cmd.exe`, is attached as a task to the PBS job. The `pbs_attach` command spawns a program using a new command shell when attaching a batch script, or when invoked with the `-c` option.

-j <job ID>

The job ID to which the session ID is to be attached. If *job ID* is not specified, a best effort is made to determine the job to which to attach the session.

-m <port number>

The port at which to contact MoM. Default: value of $PBS_MANAGER_SERVICE_PORT from `pbs.conf`.

-p <PID>

Process ID whose session ID is to be attached to the job. Default: process ID of `pbs_attach`. Cannot be used with the `-P` or `-s` options or the *cmd* operand.

-P

Attach sessions of both `pbs_attach` and the parent of `pbs_attach` to job. When used with `-s` option, the sessions of the new `fork()`ed `pbs_attach` and its parent, which is `pbs_attach`, are attached to the job. Cannot be used with the `-p` or `-s` options or the *cmd* operand.

-s

Starts a new session and attaches it to the job; `pbs_attach` calls `fork()`, then the child `pbs_attach` first calls `setsid()` and then calls `tm_attach` to attach the new session to the job. The session ID of the new `pbs_attach` is attached to the job.

--version

The `pbs_attach` command returns its PBS version information and exits. This option can only be used alone.

2.12.4 Operands

cmd

Name of command whose process ID is to be associated with the job.

2.12.5 Exit Status

0

Success

1

Any error following successful command line processing. A message is printed to standard error.

If *cmd* is specified, `pbs_attach` waits for *cmd* to exit, then exits with the exit value of *cmd*.

If *cmd* is not specified, `pbs_attach` exits after attaching the session ID(s) to the job.

2.12.6 See Also

The PBS Professional Administrator's Guide, "pbs_mom" on page 71, "pbs_tmrsh" on page 120, "TM Library", on page 93 of the PBS Professional Programmer's Guide

2.13 pbs_comm

Starts the PBS communication daemon

2.13.1 Synopsis

pbs_comm [-N] [-r <other routers>] [-t <number of threads>]

pbs_comm --version

2.13.2 Description

The PBS communication daemon, `pbs_comm`, handles communication between daemons, except for scheduler-server and server-server communication, which uses TCP. The server, scheduler(s), and MoMs are connected by one or more `pbs_comm` daemons.

See "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide.

Available on Linux only.

2.13.3 Options to pbs_comm

-N

Runs the communication daemon in standalone mode.

-r <other routers>

List of other `pbs_comm` daemons to which this `pbs_comm` must connect. This is equivalent to the `pbs.conf` variable PBS_COMM_ROUTERS. The command line overrides the variable. Format:

<hostname>[:<port number>][,<hostname>[:<port number>]]

-t <number of threads>

Number of threads the `pbs_comm` daemon uses. This is equivalent to the `pbs.conf` variable PBS_COMM_THREADS. The command line overrides the variable. Format:

Integer

--version

Prints the PBS version information and exits. This option can only be used alone.

2.13.4 Configuration Parameters

PBS_LEAF_ROUTERS

Parameter in `/etc/pbs.conf`. Tells an endpoint where to find its communication daemon.

You can tell each endpoint which communication daemon it should talk to. Specifying the port is optional.

Format: *PBS_LEAF_ROUTERS=<hostname>[:<port number>][,<hostname>[:<port number>]]*

PBS_COMM_ROUTERS

Parameter in `/etc/pbs.conf`. Tells a `pbs_comm` where to find its fellow communication daemons.

When you add a communication daemon, you must tell it about the other `pbs_comm`s in the complex. When you inform communication daemons about each other, you only tell one of each pair about the other. Do not tell both about each other. We recommend that an easy way to do this is to tell each new `pbs_comm` about each existing `pbs_comm`, and leave it at that.

Format: *PBS_COMM_ROUTERS=<hostname>[:<port number>][,<hostname>[:<port number>]]*

PBS_COMM_THREADS

Parameter in `/etc/pbs.conf`. Tells `pbs_comm` how many threads to start.

By default, each `pbs_comm` process starts four threads. You can configure the number of threads that each `pbs_comm` uses. Usually, you want no more threads than the number of processors on the host.

Maximum allowed value: *100*

Format: *Integer*

Example:

```
PBS_COMM_THREADS=8
```

PBS_COMM_LOG_EVENTS

Parameter in `/etc/pbs.conf`. Tells `pbs_comm` which log mask to use.

By default, `pbs_comm` produces few log messages. You can choose more logging, usually for troubleshooting. See "Logging and Errors with TPP" on page 54 in the PBS Professional Installation & Upgrade Guide for logging details.

Format: *Integer*

Default: *511*

Example:

```
PBS_COMM_LOG_EVENTS=<log level>
```

PBS_LEAF_NAME

Parameter in `/etc/pbs.conf`. Tells endpoint what name to use for network. The value does not include a port, since that is usually set by the daemon.

By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network.

The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation.

TPP internally resolves the name to a set of IP addresses, so you do not affect how `pbs_comm` works.

Format: *String*

Example:

```
PBS_LEAF_NAME=host1
```

PBS_START_COMM

Parameter in `/etc/pbs.conf`. Tells PBS init script whether to start a `pbs_comm` on this host if one is installed. When set to *1*, `pbs_comm` is started.

Just as with the other PBS daemons, you can specify whether each host should start `pbs_comm`.

Format: *Boolean*

Default: *0*

Example:

```
PBS_START_COMM=1
```

2.13.5 Communication Daemon Logfiles

The `pbs_comm` daemon creates its log files under `$PBS_HOME/comm_logs`. This directory is automatically created by the PBS installer.

In a failover configuration, this directory is in the shared PBS_HOME, and is used by the pbs_comm daemons running on both the primary and secondary servers. This directory must never be shared across multiple pbs_comm daemons in any other case.

The log filename format is *yyyymmdd* (the same as for other PBS daemons).

The log record format is the same as used by other pbs daemons, with the addition of the thread number and the daemon name in the log record. The log record format is as follows:

<date and time>;<event code>;<daemon name>(<thread number>);<object type>;<object name>;<message>

Example:

```
03/25/2014 15:13:39;0d86;host1.example.com;TPP;host1.example.com(Thread 2);Connection from leaf
    192.168.184.156:19331, tfd=81 down
```

2.13.6 Signal Handling by Communication Daemon

The pbs_comm daemon handles the following signals:

HUP

Re-reads the value of $PBS_COMM_LOG_EVENTS from pbs.conf.

TERM

The pbs_comm daemon exits.

2.14 pbs_dataservice

Start, stop, or check the status of PBS data service

2.14.1 Synopsis

pbs_dataservice [start | stop | status]

2.14.2 Description

The `pbs_dataservice` command starts, stops or gets the status of the PBS data service.

2.14.2.1 Permission

Root privilege is required to use this command.

2.14.3 Arguments

start
: Starts the PBS data service.

stop
: Stops the PBS data service.

  Can be used only when the PBS server is not running.

status
: Displays the status of the PBS data service, as follows:

  - Data service running

    `PBS Data Service running`
  - Data service not running

    `PBS Data Service not running`

2.14.4 Exit Status

Zero
: Success

Non-zero
: Failure

2.15 pbs_ds_password

Sets or changes data service user account or its password

2.15.1 Synopsis

pbs_ds_password [-C <username>] [-r]

2.15.2 Description

You can use this command to change the user account or account password for the data service.

2.15.2.1 Passwords

Blank passwords are not allowed.

If you type in a password, make sure it does not contain restricted characters. The `pbs_ds_password` command generates passwords containing the following characters:

0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&()_+*

When creating a password manually, do not use \ (backslash) or ‘ (backquote). This can prevent certain commands such as `pbs_server`, `pbs_ds_password`, and `printjob` from functioning properly, as they rely on connecting to the database. The format is also described in "PBS Password" on page 357.

2.15.2.2 Permissions

On Linux, root privilege is required to use this command. On Windows, Admin privilege is required.

2.15.2.3 Restrictions

Do not run this command if failover is configured. It is important not to inadvertently start two separate instances of the data service on two machines, thus potentially corrupting the database. If failover is configured, stop the secondary server, remove definitions for PBS_PRIMARY and PBS_SECONDARY from `pbs.conf` on the primary server host, start PBS, run `pbs_ds_password`, stop PBS, replace the definitions, and start PBS again.

2.15.3 Options to `pbs_ds_password`

-C <username>

Changes user account for data service to specified account. Specified user account must already exist.

On Linux-based systems, the specified user account must not be root.

On Windows, the specified user account must match the PBS service account (which can be any user account.)

This option cannot be used while the data service is running.

Can be used with the `-r` option to automatically generate a password for the new account.

-r

Generates a random password. The data service is updated with the new password.

Can be used with the `-C` option.

(no options)

Asks the user to enter a new password twice. Entries must match. Updates data service with new password.

2.15.4 Exit Status

Zero
: Success

Non-zero
: Failure

2.16 pbs_hostn

Reports hostname and network address(es)

2.16.1 Synopsis

pbs_hostn [-v] <hostname>

pbs_hostn --version

2.16.2 Description

The `pbs_hostn` command takes a hostname, and reports the results of both the `gethostbyname(3)` and `gethostbyaddr(3)` system calls. Both forward and reverse lookup of hostname and network addresses need to succeed in order for PBS to authenticate a host.

Running this command can assist in troubleshooting problems related to incorrect or non-standard network configuration, especially within clusters.

2.16.3 Options

-v
: Turns on verbose mode.

--version
: The `pbs_hostn` command returns its PBS version information and exits. This option can only be used alone.

2.16.4 Operands

hostname
: The `pbs_hostn` command accepts a *hostname* operand either in short name form, or in fully qualified domain name (FQDN) form.

2.16.5 Standard Error

The `pbs_hostn` command writes a diagnostic message to standard error for each error occurrence.

2.16.6 Exit Status

Zero
: Upon successful processing of all the operands presented to the `pbs_hostn` command.

Greater than zero
: If the `pbs_hostn` command fails to process any operand.

2.17 pbs_idled

Runs PBS daemon that monitors the console and informs pbs_mom of idle time

2.17.1 Linux Synopsis

pbs_idled [-D <display>] [-r <reconnect delay>] [-w <wait time>]

pbs_idled --version

2.17.2 Windows Synopsis

pbs_idled [start | stop]

pbs_idled --version

2.17.3 Linux Description

On Linux, the pbs_idled program monitors an X windows display and communicates the idle time of the display back to PBS. If the mouse is moved or a key is touched, PBS is informed that the vnode is busy.

You should run this program from the system-wide Xsession file, in the background before the window manager is run. If this program is run outside of the Xsession, it needs to be able to make a connection to the X display. See the xhost or xauth man pages for a description of X security.

2.17.4 Windows Description

On Windows, pbs_idled reads its polling interval from a file called idle_poll_time which is created by MoM. The service monitors keyboard, mouse, and console activity, and updates a file called idle_touch when it finds user activity. The idle_touch file is created by MoM.

2.17.5 Linux Options to pbs_idled

-D <display>
: The display to connect to and monitor

-r <reconnect delay>
: Time to wait before we try to reconnect to the X display if the previous attempt was unsuccessful

-w <wait time>
: Interval between times when the daemon checks for events or pointer movement

--version
: The pbs_idled command returns its PBS version information and exits. This option can only be used alone.

2.17.6 Windows Options to pbs_idled

start
: Starts the pbs_idled process.

stop

Stops the pbs_idled process.

--version

The pbs_idled process returns its PBS version information and exits. This option can only be used alone.

2.17.7 See Also

The PBS Professional Administrator's Guide

xhost(1), xauth(1)

2.18 pbs_iff

Tests authentication with the server

2.18.1 Usage

pbs_iff [-t] <server host> <server port>

pbs_iff --version

2.18.2 Description

Called from the pbs_connect() IFL API to authenticate a connection with the PBS server. Designed to be called internally by PBS commands and components, to be used by our IFL layer to talk to the server.

If `pbs_iff` cannot authenticate, it returns an error message.

2.18.2.1 Required Privilege

Can be run by any user.

It's a setuid root binary so it runs as the user who requests a connection to a server but it becomes root so that it can grab a privileged port.

2.18.3 Options to pbs_iff

-t
: Test mode; means test whether `pbs_iff` can authenticate with the server

--version
: Reports version and exits; can only be used alone

2.18.4 Arguments to pbs_iff

daemon host
: Host where server is running

daemon port
: Port on which server is listening; default is 15001

2.18.5 Exit Status

Zero
: If `pbs_iff` is able to contact the server at the specified port

Non-zero
: If `pbs_iff` is unable to contact the server at the specified port

2.19 pbs_interactive

Windows. Register, unregister, or get the version of PBS_INTERACTIVE service

2.19.1 Synopsis

pbs_interactive [R | U]

pbs_interactive --version

2.19.2 Description

The pbs_interactive command registers, unregisters, or gets the version of the Windows PBS_INTERACTIVE service. The service must be registered manually; the installer does not register it.

On Windows, the PBS_INTERACTIVE service itself monitors logging in and out by users, starts a pbs_idled process for each user logging in, and stops the pbs_idled process of each user logging out.

2.19.2.1 Required Privilege

Admin privilege is required to use this command.

2.19.3 Arguments

R

Registers the PBS_INTERACTIVE service.

U

Unregisters the PBS_INTERACTIVE service.

--version

The pbs_interactive command returns its PBS version information and exits. This option can only be used alone.

2.20 pbs_lamboot

Deprecated. PBS front end to LAM's `lamboot` program

2.20.1 Synopsis

pbs_lamboot

pbs_lamboot --version

2.20.2 Description

The PBS command `pbs_lamboot` replaces the standard `lamboot` command in a PBS LAM MPI job, for starting LAM software on each of the PBS execution hosts running Linux 2.4 or higher.

Usage is the same as for LAM's `lamboot`. All arguments except for `bhost` are passed directly to `lamboot`. PBS will issue a warning saying that the `bhost` argument is ignored by PBS since input is taken automatically from $PBS_NODEFILE. The `pbs_lamboot` program will not redundantly consult the $PBS_NODEFILE if it has been instructed to boot the vnodes using the `tm` module. This instruction happens when an argument is passed to `pbs_lamboot` containing "`-ssi boot tm`" or when the LAM_MPI_SSI_boot environment variable exists with the value *tm*.

2.20.3 Options

--version

The `pbs_lamboot` command returns its PBS version information and exits. This option can only be used alone.

2.20.4 Operands

The operands for `pbs_lamboot` are the same as for `lamboot`.

2.20.5 Environment Variables and Path

The PATH on remote machines must contain `PBS_EXEC/bin`.

2.20.6 See Also

The PBS Professional Administrator's Guide, `lamboot(1)`, "TM Library", on page 93 of the PBS Professional Programmer's Guide

2.21 pbs_login

Caches encrypted user password for authentication

2.21.1 Usage

pbs_login

pbs_login -m <PBS service account password>

echo <password>| pbs_login -p

2.21.2 Description

The `pbs_login` command encrypts the password and caches it locally where it can be used by daemons for authorization.

Job submitters must run this command at each submission host each time their password changes.

On Windows, the `win_postinstall` script calls `pbs_login` to store the PBS service account password so that the account user can be authenticated by daemons.

2.21.3 Required Privilege

Can be run by any user.

2.21.4 Options to pbs_login

(no options)

Queries user for password.

-m <PBS service account password>

This option is intended to be used only by the PBS service account, which is the account that is used to execute `pbs_mom` via the Service Control Manager on Windows. This option is used during installation when invoked by the `win_postinstall` script, or by the administrator when the PBS service account password has changed. Stores PBS service account password in the `mom_priv` directory.

-p

Caches user password on client host. Intended to be run by job submitter at client host. Allows job submitter to be authenticated by daemons.

2.22 pbs_mkdirs

For Windows. Create, or fix the permissions of, the directories and files used by PBS

2.22.1 Synopsis

pbs_mkdirs

pbs_mkdirs [mom]

2.22.2 Description

Runs on Windows only. If the directories and files used by PBS exist, the `pbs_mkdirs` command fixes their permissions. If the directories and/or files do not exist, the `pbs_mkdirs` command creates them, with the correct permissions. The `pbs_mkdirs` command always examines the following directories and files:

`pbs.conf`
`PBS_EXEC`
`PBS_HOME/spool`
`PBS_HOME/undelivered`
`PBS_HOME/pbs_environment`

2.22.2.1 Required Privilege

You must have Administrator privilege to run this command.

2.22.3 Options

mom
: The `pbs_mkdirs` command examines the following additional items:

`PBS_HOME/mom_priv`
`PBS_HOME/mom_logs`

(no options)
: The `pbs_mkdirs` command examines all of the files and directories specified for the `mom` option.

2.22.4 See Also

The PBS Professional Administrator's Guide, "pbs_probe" on page 81

2.23 pbs_mom

Runs the PBS job monitoring and execution daemon

2.23.1 Synopsis

pbs_mom [-a <alarm timeout>] [-C <checkpoint directory>] [-c <config file>] [-d <MoM home directory>] [-L <logfile>] [-M <MoM port>] [-N] [-n <nice value>] [-p|-r] [-R <inter-MoM communication port>] [-S <server port>] [-s <options>]

pbs_mom --version

2.23.2 Description

The `pbs_mom` command starts the PBS job monitoring and execution daemon, called *MoM*.

The standard MoM starts jobs on the execution host, monitors and reports resource usage, enforces resource usage limits, and notifies the server when the job is finished. The MoM also runs any prologue scripts before the job runs, and runs any epilogue scripts after the job runs.

The MoM performs any communication with job tasks and with other MoMs. The MoM on the first vnode on which a job is running manages communication with the MoMs on the remaining vnodes on which the job runs.

The MoM manages one or more vnodes. PBS may treat a host as a set of virtual nodes, in which case one MoM manages all of the host's vnodes. See "Configuring MoMs and Vnodes" on page 33 in the PBS Professional Administrator's Guide.

2.23.2.1 Logging

The MoM's log file is in `PBS_HOME/mom_logs`. The MoM writes an error message in its log file when it encounters any error. If it cannot write to its log file, it writes to standard error. The MoM writes events to its log file. The MoM writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

2.23.2.2 Required Permission

The executable for `pbs_mom` is in `PBS_EXEC/sbin`, and can be run only by root on Linux, and Admin on Windows.

2.23.2.2.i HPE Systems Running Supported Versions of HPE MPI

A PBS job can run across multiple machines that run supported versions of HPE MPI.

PBS can run using HPE's MPI (MPT) over InfiniBand. See the PBS Professional Administrator's Guide.

2.23.2.3 Effect on Jobs of Starting MoM

When MoM is started or restarted, her default behavior is to leave any running processes running, but to tell the PBS server to requeue the jobs she manages. MoM tracks the process ID of jobs across restarts.

In order to have all jobs killed and requeued, use the `-r` option when starting or restarting MoM.

In order to leave any running processes running, and not to requeue any jobs, use the `-p` option when starting or restarting MoM.

2.23.3 Options to pbs_mom

-a <alarm timeout>

Number of seconds before alarm timeout. Whenever a resource request is processed, an alarm is set for the given amount of time. If the request has not completed before *alarm timeout*, the OS generates an alarm signal and sends it to MoM.

Format: *Integer*

Default: *10 seconds*

-C <checkpoint directory>

Specifies the path to the directory where MoM creates job-specific subdirectories used to hold each job's restart files. MoM passes this path to checkpoint and restart scripts. Overrides other checkpoint path specification methods. Any directory specified with the -c option must be owned, readable, writable, and executable by root only (*rwx,---,---*, or *0700*), to protect the security of the restart files. See the -d option to pbs_mom and "Specifying Checkpoint Path" on page 428 in the PBS Professional Administrator's Guide.

Format: *String*

Default: PBS_HOME/checkpoint

-c <config file>

MoM will read this alternate default configuration file upon starting. If this is a relative file name it is relative to PBS_HOME/mom_priv. If the specified file cannot be opened, pbs_mom will abort. See the -d option.

MoM's normal operation, when the -c option is not given, is to attempt to open the default configuration file PBS_HOME/mom_priv/config. If this file is not present, pbs_mom will log the fact and continue.

-d <MoM home directory>

Specifies the path of the directory to be used in place of PBS_HOME by pbs_mom. The default directory is given by $PBS_HOME.

Format: *String*

-L <logfile>

Specifies an absolute path and filename for the log file. The default is a file named for the current date in PBS_HOME/mom_logs/. See the -d option.

Format: *String*.

-M <MoM port>

Specifies the port number on which MoM will listen for server requests and instructions. Overrides PBS_MOM_SERVICE_PORT setting in pbs.conf and environment variable.

Format: *Integer port number*.

Default: *15002*.

-n <nice value>

Specifies the priority for the pbs_mom daemon.

Format: *Integer*.

-N

Specifies that when starting, MoM should not detach from the current session.

-p

Specifies that when starting, MoM should allow any running jobs to continue running, and not have them requeued. This option can be used for single-host jobs only; multi-host jobs cannot be preserved. Cannot be used with the -r option. MoM is not the parent of these jobs.

-r

Specifies that when starting, MoM should requeue any rerunnable jobs and kill any non-rerunnable jobs that she was tracking, and mark the jobs as terminated. Cannot be used with the -p option. MoM is not the parent of these jobs.

It is not recommended to use the -r option after a reboot, because process IDs of new, legitimate tasks may match those MoM was previously tracking. If they match and MoM is started with the -r option, MoM will kill the new tasks.

-R <inter-MoM communication port>

Specifies the port number on which MoM will listen for pings, resource information requests, communication from other MoMs, etc. Overrides PBS_MANAGER_SERVICE_PORT setting in pbs.conf and environment variable.

Format: *Integer port number*

Default: *15003*

-S <server port>

Specifies the port number on which pbs_mom initially contacts the server.

Format: *Integer port number*

Default: *15001*

-s <file options>

If you are running the cgroups hook, make sure that the vnode names in any Version 2 configuration file exactly match those in the output of pbsnodes -av.

This option lets you add, delete, and display Version 2 configuration files. Run this command at the host you want to change. The *file options* are used this way:

-s insert <Version 2 filename> <inputfile>

Reads *inputfile* and copies it to a Version 2 vnode configuration file with the filename *Version 2 filename*. For example, to create a Version 2 file named "Myhost_V2":

```
pbs_mom -s insert <Myhost_V2> <myhost_v2_input>
```

If a configuration file with the specified *Version 2 filename* already exists, the operation fails, and pbs_mom prints a diagnostic and exits with a nonzero status. Configuration files whose names begin with the prefix "*PBS*" are reserved. You cannot add a file whose name begins with "*PBS*"; pbs_mom will print a diagnostic message and exit with a nonzero status.

-s remove <Version 2 filename>

Removes the configuration file named *Version 2 filename* if it exists. Example:

```
pbs_mom -s remove <Version 2 filename>
```

If the file does not exist or if you try to remove a file with the reserved "PBS" prefix, the operation fails, and pbs_mom prints a diagnostic and exits with a nonzero status.

-s show <Version 2 filename>

Prints the contents of the named file to standard output. Example:

```
pbs_mom -s show <Version 2 filename>
```

If *Version 2 filename* does not exist, the operation fails and pbs_mom writes a diagnostic and exits with a nonzero status.

-s list

MoM lists the PBS-prefixed and site-defined configuration files in the order in which they are executed. Example:

```
pbs_mom -s list
```

WINDOWS:

Under Windows, use the -N option so that pbs_mom will start up as a standalone program. For example:

```
pbs_mom -N -s insert <Version 2 filename> <inputfile>
```

or

```
pbs_mom -N -s list
```

--version

The pbs_mom command returns its PBS version information and exits. This option can only be used alone.

2.23.4 Files and Directories

$PBS_HOME/mom_priv

Default directory for default configuration files.

$PBS_HOME/mom_priv/config

MoM's default configuration file.

$PBS_HOME/mom_logs

Default directory for log files written by MoM.

$PBS_HOME/mom_priv/prologue

File containing administrative script to be run before job execution.

$PBS_HOME/mom_priv/epilogue

File containing administrative script to be run after job execution.

2.23.5 Signal Handling

pbs_mom handles the following signals:

SIGHUP

The pbs_mom daemon rereads its configuration files, closes and reopens the log file, and reinitializes resource structures.

SIGALRM

MoM writes a log file entry. See the -a <alarm timeout> option.

SIGINT

The pbs_mom daemon exits, leaving all running jobs still running. See the -p option.

SIGKILL

This signal is not caught. The pbs_mom daemon exits immediately.

SIGTERM, SIGXCPU, SIGXFSZ, SIGCPULIM, SIGSHUTDN

The pbs_mom daemon terminates all running children and exits.

SIGPIPE, SIGUSR1, SIGUSR2, SIGINFO

These are ignored.

All other signals have their default behavior installed.

2.23.6 Exit Status

Zero

Upon success

Greater than zero

- If the `pbs_mom` daemon fails to start
- If the `-s insert` option is used with an existing *Version 2 filename*
- If the administrator attempts to add a script whose name begins with "*PBS*"
- If the administrator attempts to use the `-s remove` option on a nonexistent configuration file, or on a configuration file whose name begins with "*PBS*"
- If the administrator attempts to use the `-s show` option on a nonexistent script

2.23.7 See Also

The PBS Professional Administrator's Guide

2.24 pbs_mpihp

Runs an MPI application in a PBS job with HP MPI

2.24.1 Synopsis

pbs_mpihp [-h <hostname>] [-np <number>] [<other HP mpirun options>] <program> [<args>]

pbs_mpihp [<HP mpirun options>] -f <appfile> [-- [<extra args>]]

pbs_mpihp --version

2.24.2 Description

The PBS command `pbs_mpihp` replaces the standard `mpirun` command in a PBS HP MPI job, for executing programs. `pbs_mpihp` is a front end to the HP MPI version of `mpirun`.

When `pbs_mpihp` is invoked from a PBS job, it processes the command line arguments, then calls standard HP `mpirun` to actually start the MPI ranks. The ranks created are mapped onto CPUs on the vnodes allocated to the PBS job. The environment variable MPI_REMSH is set to `$PBS_EXEC/bin/pbs_tmrsh`. This causes the processes that are created to become part of the PBS job.

The path to standard HP `mpirun` is found by checking to see if a link exists with the name `PBS_EXEC/etc/pbs_mpihp`. If this link exists, it points to standard HP `mpirun`. If it does not exist, a call to `mpirun -version` is made to determine whether it is HP `mpirun`. If so, the call is made to "`mpirun`" without an absolute path. If HP `mpirun` cannot be found, an error is output, all temp files are cleaned up and the script exits with value 127.

If `pbs_mpihp` is invoked from outside a PBS job, it passes all of its arguments directly to standard HP `mpirun` without further processing.

2.24.2.1 Configuration

When HP MPI is wrapped with `pbs_mpihp`, "`rsh`" is the default used to start the mpids. If you wish to use "`ssh`" or something else, be sure to set the following in `$PBS_HOME/pbs_environment`:

```
PBS_RSHCOMMAND=ssh
```

or put the following in the job script:

```
export PBS_RSHCOMMAND=<rsh_cmd>
```

2.24.2.2 Usage

Usage is the same as for HP `mpirun`.

`pbs_mpihp <program>` allows one executable to be specified.

`pbs_mpihp -f <appfile>` uses an *appfile* to list multiple executables. The format is described in the HP `mpirun` man page. If this form is used from inside a PBS job, the file is read to determine what executables are to be run and how many processes are started for each.

Executing `pbs_mpihp` with the `-client` option is not supported under PBS.

2.24.3 Options to `pbs_mpihp`

All options except the following are passed directly to HP `mpirun` with no modification.

-client
Not supported.

-f <appfile>
The specified *appfile* is read by `pbs_mpihp`.

-h <hostname>
Ignored by `pbs_mpihp`.

-l <username>
Ignored by `pbs_mpihp`.

-np <number>
Specifies the *number* of processes to run on the PBS vnodes.

--version
The `pbs_mpihp` command returns its PBS version information and exits. This option can only be used alone.

2.24.4 Exit Values

127
If HP `mpirun` cannot be found

2.24.5 See Also

The PBS Professional Administrator's Guide
`mpirun(1)`

2.25 pbs_mpilam

Deprecated. Runs MPI programs under PBS with LAM MPI

2.25.1 Synopsis

pbs_mpilam [<mpilam options>]

pbs_mpilam --version

2.25.2 Description

The PBS command `pbs_mpilam` replaces the standard `mpirun` command in a PBS LAM MPI job.

If used to run a single program, PBS tracks resource usage and controls all user processes spawned by the program. If used to run multiple programs as specified in an application file (no `<where>` argument and no `-np/-c` option), PBS does not manage the spawned user processes of each program.

If the `where` argument is not specified, `pbs_mpilam` will try to run the user's program on all available CPUs using the *C* keyword.

2.25.2.1 Prerequisites

The PATH on remote machines must contain `PBS_EXEC/bin`.

2.25.2.2 Usage

Usage is the same as for LAM `mpirun`. All options are passed directly to `mpirun`.

2.25.3 Options to `pbs_mpilam`

<mpilam options>

The `pbs_mpilam` command uses the same options as `mpirun`.

--version

The `pbs_mpilam` command returns its PBS version information and exits. This option can only be used alone.

2.25.4 See Also

The PBS Professional Administrator's Guide

`mpirun(1)`

2.26 pbs_mpirun

Deprecated. Runs MPI programs under PBS with MPICH

2.26.1 Synopsis

pbs_mpirun [<mpirun options>]

pbs_mpirun --version

2.26.2 Description

The PBS command `pbs_mpirun` replaces the standard `mpirun` command in a PBS MPICH job using P4.

On Windows, this command cannot be used to start job processes or track a job's resource usage.

2.26.2.1 Prerequisite

The PATH on remote machines must contain `PBS_EXEC/bin`.

2.26.2.2 Usage

Usage is the same as for `mpirun`, except for the `-machinefile` option. All other options are passed directly to `mpirun`.

2.26.3 Options to `pbs_mpirun`

<mpirun options>

The options to `pbs_mpirun` are the same as for `mpirun`, except for the `-machinefile` option. This is generated by `pbs_mpirun`. The user should not attempt to specify `-machinefile`.

The value for `-machinefile` is a temporary file created from PBS_NODEFILE in the format:

```
hostname-1[:number of processors]
hostname-2[:number of processors]
hostname-n[:number of processors]
```

where if the number of processors is not specified, it is *1*. An attempt by the user to specify the `-machinefile` option will result in a warning saying "`Warning, -machinefile value replaced by PBS`".

The default value for the `-np` option is the number of entries in PBS_NODEFILE.

--version

The `pbs_mpirun` command returns its PBS version information and exits. This option can only be used alone.

2.26.4 Environment Variables

`pbs_mpirun` modifies P4_RSHCOMMAND and PBS_RSHCOMMAND. Users should not edit these. `pbs_mpirun` copies the value of P4_RSHCOMMAND into PBS_RSHCOMMAND.

2.26.5 See Also

The PBS Professional Administrator's Guide, `mpirun(1)`

2.27 pbs_probe

Deprecated. Reports PBS diagnostic information and fixes permission errors

2.27.1 Synopsis

pbs_probe [-f | -v]

pbs_probe --version

2.27.2 Description

The pbs_probe command reports post-installation information useful for PBS diagnostics, and fixes permission errors.

2.27.2.1 Information Sources

- Information that is supplied on the command line
- The file /etc/pbs.conf
- The file /etc/init.d/pbs
- The values of any of the following environment variables; these may be set in the environment in which pbs_probe is run: PBS_CONF_FILE, PBS_HOME, PBS_EXEC, PBS_START_SERVER, PBS_START_MOM, and PBS_START_SCHED

2.27.2.2 Required Privilege

In order to execute pbs_probe, you must have PBS Operator or Manager privilege.

2.27.3 Options to pbs_probe

(no options)

Run in "report" mode. In this mode pbs_probe reports any permission errors detected in PBS infrastructure files. The command categorizes the errors and writes a list of them by category. Empty categories are not written.

-f

Run in "fix" mode. In this mode pbs_probe examines each of the relevant infrastructure files and, where possible, fixes any permission errors that it detects, and prints a message saying what got changed. If it is unable to fix a problem, it prints a message saying what was detected.

-v

Run in "verbose" mode. In this mode pbs_probe writes a complete list of the infrastructure files that it checked.

--version

The pbs_probe command returns its PBS version information and exits. This option can only be used alone.

2.27.4 Standard Error

The pbs_probe command writes a diagnostic message to standard error for each error occurrence.

2.27.5 Exit Status

Exit code does not reflect results of probe; it reflects whether or not the program ran.

Zero
: When run correctly, whether or not `pbs_probe` finds any problems or errors

Non-negative
: When run incorrectly

2.27.6 See Also

The PBS Professional Administrator's Guide

2.28 pbs_python

Python interpreter for debugging a hook script from the command line

2.28.1 Synopsis

pbs_python --hook [-e <log event mask>] [-i <event input file>] [-L <log dir>] [-l <log file>] [-o <hook execution record>] [-r <resourcedef file>] [-s <site data file>] [<Python script>]

pbs_python <standard Python options>

pbs_python --version

2.28.2 Description

The PBS Python interpreter, `pbs_python`, is a wrapper for Python.

You can use the `pbs_python` wrapper that is shipped with PBS to debug hooks. Either:

- Use the `--hook` option to `pbs_python` to run `pbs_python` as a wrapper to Python, employing the `pbs_python` options. With the `--hook` option, you cannot use the standard Python options. The rest of this section covers how to use `pbs_python` with the `--hook` option.
- Do not use the `--hook` option, so `pbs_python` runs the Python interpreter, with the standard Python options, and without access to the `pbs_python` options.

2.28.2.1 Debugging Hooks

You can get each hook to write out debugging files, and then modify the files and use them as debugging input to `pbs_python`. Alternatively, you can write the files yourself.

Debugging files can contain information about the event, about the site, and about what the hook changed. You can use these as inputs to a hook when debugging.

For a complete description of using `pbs_python` with debugging files, see "Debugging Hooks" on page 159 in the PBS Professional Hooks Guide.

2.28.3 Options to `pbs_python`

--hook

This option is a switch. When you use this option, you can use the PBS Python module (via "`import pbs`"), and the other options described here are available. When you use this option, you cannot use the standard Python options. This option is useful for debugging.

When you do not use this option, you cannot use the other options listed here, but you can use the standard Python options.

-e <log event mask>

Sets the mask that determines which event types are logged by `pbs_python`. To see only debug messages, set the value to *0xd80*. To see all messages, set the value to *0xffff*. The `pbs_python` interpreter uses the same set of mask values that are used for the $logevent <mask> entry in the `pbs_mom` configuration file. See section 2.23, "pbs_mom", on page 71. Available only when `--hook` option is used.

-i <event input file>

Text file containing data to populate pbs.event() objects. Each line specifies an attribute value or a resource value. Syntax of each input line is one of the following:

<object name>.<attribute name>=<attribute value>

<object name>.<resource list>[<resource name>]=<resource value>

Where

<object name> is a PBS object name which can refer to its sub-objects. Examples: *"pbs.event()", "pbs.event().job", "pbs.event().vnode_list["<vnode name>"]"*.

Example input file:

```
pbs.event().hook_name=proto
pbs.event().hook_type=site
pbs.event().type=queuejob
pbs.event().requestor=user1
pbs.event().requestor_host=host1
pbs.event().alarm=40
pbs.event().job.id=72
pbs.event().job.Job_Name=job1
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=6mb
pbs.event().vnode_list["host1"].resources_available["ncpus"] = 5
pbs.event().vnode_list["host1"].resources_available["mem"] = 300gb
```

Available only when `--hook` option is used.

-L <log dir>

Directory holding the log file where pbs.logmsg() and pbs.logjobmsg() write their output. Default is current working directory where `pbs_python` is executed. Available only when `--hook` option is used.

-l <log file>

Log file where pbs.logmsg() and pbs.logjobmsg() write their output. Default file name is current date in *yyyymmdd* format. Available only when `--hook` option is used.

-o <hook execution record>

The hook execution record contains the changes made after executing the hook script, such as the attributes and resources set in any pbs.event() jobs and reservations, whether an action was accepted or rejected, and any pbs.reject() messages.

Example hook execution record:

```
pbs.event().job.Job_Name=job2
pbs.event().job.Resource_List[file]=60gb
pbs.event().job.Resource_List[ncpus]=5
pbs.event().job.Resource_List[mem]=20gb
pbs.event().job.Account_Name=account2
pbs.event().reject=True
pbs.event().reject_msg=No way!
```

Without this option, output goes to `stdout`. Available only when `--hook` option is used.

-r <resourcedef file>

File/path name containing a resource definition specifying a custom resource whose Python type is pbs.resource. Format:

<resource name> type=<typename> [flag=<value>]

Available only when `--hook` option is used.

-s <site data file>

The site data file can contain any relevant information about the server, queues, vnodes, and jobs at the server. This file can be written by a hook or by the administrator.

When the hook writes it, this file contains the values that populate the server, queues, vnodes, reservations, and jobs, with all attributes and resources for which there are values.

The site data file is named *hook_<event type>_<hook name>_<random integer>.data*. It can be passed to `pbs_python` using the `-s <site data file>` option.

Available only when `--hook` option is used.

--version

The `pbs_python` command prints its version information and exits. This option can only be used alone.

2.28.4 Arguments

<Python script>

The hook script to execute. We recommend importing the PBS Python module at the start of the script:

```
import pbs
```

If you do not specify <*Python script*>, you can perform interactive debugging. If you type the following:

```
% pbs_python --hook -i hook.input
```

The interpreter displays a prompt:

```
>>
```

You can type your Python lines at the prompt:

```
>>import pbs
>> e=pbs.event().job
>> print e.id
<job-id>
...
```

2.29 pbs_ralter

Modifies an existing reservation

2.29.1 Summary

Alter an existing advance, standing, or job-specific reservation.

2.29.2 Synopsis

pbs_ralter [-D <duration>] [-E <end time>] [-G <auth group list>] [-I <block time>] [-l select=<select spec>] [-m <mail points>] [-M <mail list>] [-N <reservation name>] [-R <start time>] [-U <auth user list>] <reservation ID>

pbs_ralter -Wforce [-D <duration>] [-E <end time>] [-R <start time>] <reservation ID>

pbs_ralter --version

2.29.3 Description

You can use the `pbs_ralter` command to alter an existing reservation, whether it is an individual job-specific, advance, or maintenance reservation, or the next or current occurrence of a standing reservation. You can change the start time, end time, duration, events that generate mail, mail recipient list, authorized groups, authorized users, select specification, and reservation name.

The PBS Administrator can use the `-Wforce` option to this command to change the start time, end time, or duration of a reservation; this option overrides the scheduler's actions.

After the change is requested, the change is either confirmed or denied. On denial of the change, the reservation is not deleted and is left as is, and the following message appears in the server's log:

```
Unable to alter reservation <reservation ID>
```

When a reservation is confirmed, the following message appears in the server's log:

```
Reservation alter successful for <reservation ID>
```

To find out whether or not the change was allowed:

- Use the `pbs_rstat` command to see whether you altered reservation attribute(s)
- Use the interactive option to check for confirmation after the blocking time has run out
- Check the server log for confirmation or denial messages

Before the change is confirmed or denied, the change is unconfirmed, and the reservation state is *AL*.

Once a reservation change is confirmed, the reservation state is *CO* or *RN*.

If the reservation has not started and it cannot be confirmed on the same vnodes, PBS searches for another set of vnodes.

If the reservation is altered, PBS logs a Y accounting record. See "Types of Accounting Log Records" on page 630 in the PBS Professional Administrator's Guide.

2.29.3.1 Caveats and Restrictions

You cannot change the start time of a reservation if jobs are running in it.

If you change the end time of a reservation so that it ends before a job running in the reservation finishes, the job is killed when the reservation ends.

2.29.3.2 Required Privilege

Without the -Wforce option, this command can be used by the reservation owner or the PBS Administrator.

With the -Wforce option, this command can be used only by the PBS Administrator.

2.29.4 Options to pbs_ralter

-D <duration>

Specifies reservation's new duration. This option can be used even when the reservation is running and has jobs that are submitted to and/or are running in the reservation.

Can be specified with start and/or end time. PBS calculates anything not specified. When specified without start or end time, PBS keeps previous start time.

If you change the duration to less than the time the reservation has already run, PBS deletes the reservation.

Format: *Duration*, as *seconds* or *hh:mm:ss*

-E <end time>

Specifies reservation's new end time. This option can be used even when the reservation is running and has jobs that are submitted to and/or are running in the reservation.

Format: *Datetime*

-G <auth group list>

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's Authorized_Groups attribute to *auth group list*.

This list becomes the acl_groups list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the Authorized_Groups reservation attribute in section 6.8, “Reservation Attributes”, on page 303.

Syntax:

[+|-]<group name>[,[+|-]<group name> ...]

Default: no default; group names are unchanged

-I <block time>

(Capital I) Specifies interactive mode. The pbs_ralter command will block, up to *block time* seconds, while waiting for the reservation's change request to be confirmed or denied.

The value for *block time* must be positive. The pbs_ralter command returns either the status "*CONFIRMED*" or the status "*DENIED*".

Format: *Integer*

Default: *Not interactive*

-l select=<select spec>

(Lowercase L) Specifies new select specification for reservation. New specification must be a subset of the same chunks requested by the original reservation. If reservation is started, cannot be used to release chunks where reservation jobs are running. If reservation is started and degraded, you must release all unavailable chunks in order to alter the reservation select specification.

-m <mail points>

Specifies the set of events that cause mail to be sent to the list of users specified in the `-M <mail list>` option.

Format: String

Syntax: Either of:

- Any combination of "*a*", "*b*", "*c*" or "*e*"
- The single character "*n*"

Table 2-4: Suboptions to -m Option

Character	Meaning
a	Notify if reservation is terminated for any reason
b	Notify when the reservation period begins
c	Notify when the reservation is confirmed
e	Notify when the reservation period ends
n	Send no mail. Cannot be used with any of a, b, c or e.

Default: No default; if not specified, mail events are unchanged.

-M <mail list>

The list of users to whom mail is sent whenever the reservation transitions to one of the states specified in the -m <mail points> option.

Format: *<username>[@<hostname>][,<username>[@<hostname>]...]*

Default: No default; if not specified, user list is unchanged.

-N <reservation name>

Specifies a name for the reservation.

Format: String up to 15 characters in length. It must consist of printable, non-white space characters with the first character alphabetic.

Default: No default; if not specified, reservation name is unchanged.

-R <start time>

Specifies reservation's new start time. This option can be used either when the reservation is not running or there are no jobs are submitted to the reservation. You cannot use this option when a reservation is not empty and has started running.

The specifications for providing the time are the same as for `pbs_rsub`:

If the day, DD, is not specified, it defaults to today if the time hhmm is in the future. Otherwise, the day is set to tomorrow. For example, if you alter a reservation with the specification `-R 1110` at 11:15 a.m., it is interpreted as being for 11:10 a.m. tomorrow. If the month portion, MM, is not specified, it defaults to the current month, provided that the specified day DD, is in the future. Otherwise, the month is set to next month. Similar rules apply to the two other optional, left-side components.

Format: *Datetime*

-U <auth user list>

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's Authorized_Users attribute to *auth user list*.

This list becomes the acl_users attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the Authorized_Users reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<username>[@<hostname>][,[+|-]<username>[@<hostname>]...]

Default: no default; user list is unchanged

-Wforce

Enforces changes made to the reservation start time, end time, or duration, regardless of the actions of the scheduler. Can be used only by the PBS Administrator. Note that with this option you can force PBS to over-subscribe resources, in which case you (the administrator) may need to manage them yourself. Cannot be used to change the start time of a reservation in which jobs are running.

--version

The `pbs_ralter` command returns its PBS version information and exits. This option can only be used alone.

2.29.5 Operands

The `pbs_ralter` command takes a reservation ID.

For an advance or job-specific reservation this has the form:

R<sequence number>[.<server name>][@<remote server>]

For a standing reservation this has the form:

S<sequence number>[.<server name>][@<remote server>]

For a maintenance reservation this has the form:

M<sequence number>[.<server name>][@<remote server>]

@<remote server> specifies a reservation at a server other than the default server.

2.30 pbs_rdel

Deletes a PBS reservation

2.30.1 Synopsis

pbs_rdel <reservation ID>[,<reservation ID>...]

pbs_rdel --version

2.30.2 Description

The `pbs_rdel` command deletes reservations in the order specified.

This command deletes the specified reservations, whether or not they are running, all jobs in the reservations, and the reservation queues.

2.30.2.1 Required Privilege

A reservation may be deleted by its owner, a PBS Operator, or a PBS Manager.

2.30.3 Options

--version
: The `pbs_rdel` command returns its PBS version information and exits. This option can only be used alone.

2.30.4 Operands

The `pbs_rdel` command accepts one or more *reservation ID* operands.

For an advance or job-specific reservation this has the form:

R<sequence number>[.<server name>][@<remote server>]

For a standing reservation this has the form:

S<sequence number>[.<server name>][@<remote server>]

For a maintenance reservation this has the form:

M<sequence number>[.<server name>][@<remote server>]

@<remote server> specifies a reservation at a server other than the default server.

2.30.5 Exit Status

Zero
: Upon success

Greater than zero
: Upon failure to process any operand

2.30.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "pbs_rsub" on page 96, "pbs_rstat" on page 94, "Reservation Attributes" on page 303

2.31 pbs_release_nodes

Releases vnodes assigned to a PBS job

2.31.1 Synopsis

pbs_release_nodes [-j <job ID>] [-k (<number of hosts to keep> | <selection of vnodes to keep>)] <vnode> [<vnode> [<vnode>] ...]

pbs_release_nodes [-j <job ID>] -a

pbs_release_nodes --version

2.31.2 Description

You can use the `pbs_release_nodes` command to release no-longer-needed sister hosts or vnodes assigned to a running job, before the job would normally release them. These vnodes are then available for use by other jobs.

You can specify the names of sister vnodes to be released, or you can release all sister vnodes not on the primary execution host that are assigned to a running job via the `-a` option.

PBS can keep the number of sister hosts you specify, or PBS can release all sister vnodes except for the ones you specify via a select statement.

Can be used on jobs and subjobs, but not on job arrays or ranges of subjobs.

2.31.2.1 Caveats and Restrictions

- You can release only sister hosts or vnodes that are not on the primary execution host. You cannot release vnodes on the primary execution host.
- The job must be running (in the *R* state).
- The `pbs_release_nodes` command is not supported on vnodes tied to Cray X* series systems (vnodes whose vntype has the "cray_" prefix).
- If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.
- You cannot release a partial host. If you try to release some but not all of a host, the job's exec_vnode attribute shows the new, smaller list of vnodes, but the `pbsnodes` command will reveal that the host is still allocated to the job.
- If you specify release of a vnode on which a job process is running, that process is terminated when the vnode is released.

2.31.2.2 Required Privilege

This command can be run by the job owner, the PBS Manager, Operator, and Administrator, as well as root on Linux and Admin on Windows.

2.31.3 Options to pbs_release_nodes

-a

Releases all job vnodes not on the primary execution host. Cannot be used with `-k` option, or with list of vnode names.

-j <job ID>

Specifies the job ID for the job or subjob whose vnode(s) are to be released.

-k <keep number> | <keep selection>

Use *keep number* to specify how many sister hosts to keep.

Use *keep selection* to specify which sister vnodes to keep. The *keep selection* is a select statement beginning with "select=" specifying which vnodes to keep.

The primary execution host and its vnodes are not released.

For example, to release all sister hosts except 8:

```
pbs_release_nodes -k 8
```

To release all sister vnodes except for 4 of the ones marked with "bigmem":

```
pbs_release_nodes -k select=4:bigmem=true
```

Cannot be used with `-a` option or with vnode list argument.

(no options)

With no options, `pbs_release_nodes` uses the value of the PBS_JOBID environment variable as the job ID of the job whose vnodes are to be released.

--version

The `pbs_release_nodes` command returns its PBS version information and exits. This option can only be used alone.

2.31.4 Operands for pbs_release_nodes

The `pbs_release_nodes` command can take as an operand a list of vnodes. Format:

<vnode name> [<vnode name> [<vnode name>] ...]

Cannot be used with the `-a` option.

2.31.5 Usage

This command can be run at the command line, or called inside a job script, where it can use the value of the PBS_JOBID environment variable.

You can release any vnode that appears in the job's exec_vnode attribute that is not on the primary execution host. You can release a particular set of a job's vnodes, or you can release all of a job's non-primary-execution-host vnodes.

To release specific vnodes:

pbs_release_nodes [-j <job ID>] <vnode name> [<vnode name>] ...]

To release all of a job's vnodes that are not on the primary execution host:

pbs_release_nodes [-j <job ID>] -a

To release all except a specified number of vnodes:

pbs_release_nodes -k <number of sister hosts to keep>

To release all vnodes except for those in a select specification:

pbs_release_nodes -k <select specification>

2.32 pbs_rstat

Shows status of PBS reservations

2.32.1 Synopsis

pbs_rstat [-B] [-f|-F] [-S] [<reservation ID>...]

pbs_rstat --version

2.32.2 Description

The `pbs_rstat` command shows the status of all reservations at the PBS server. Denied reservations are not displayed.

2.32.2.1 Required Privilege

This command can be run by a user with any level of PBS privilege. For full output, users without manager or operator privilege cannot print custom resources which were created to be invisible to users.

2.32.3 Output

The `pbs_rstat` command displays output in any of brief, short, or full formats.

See section 6.8, “Reservation Attributes”, on page 303 and section 8.6, “Reservation States”, on page 367.

2.32.4 Options to `pbs_rstat`

-B
: Brief output. Displays each reservation identifier only.

-f, -F
: Full output. Displays all reservation attributes that are not set to the default value. Users without manager or operator privilege cannot print custom resources which were created to be invisible to users.

-S
: Short output. Displays a table showing the name, queue, owner, state, start time, duration, and end time of each reservation.

--version
: The `pbs_rstat` command returns its PBS version information and exits. This option can only be used alone.

(no options)
: Short output. Same behavior as `-S` option.

2.32.5 Operands

The `pbs_rstat` command accepts one or more *reservation ID* operands.

Format for an advance or job-specific reservation:

R<sequence number>[.<server name>][@<remote server>]

Format for a standing reservation:

S<sequence number>[.<server name>][@<remote server>]

Format for a maintenance reservation:

M<sequence number>[.<server name>][@<remote server>]

@<remote server> specifies a reservation at a server other than the default server.

2.32.6 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "pbs_rsub" on page 96, "pbs_rdel" on page 90, "Reservation Attributes" on page 303

2.33 pbs_rsub

Creates a PBS reservation

2.33.1 Synopsis

For advance and standing reservations:

pbs_rsub [-D <duration>] [-E <end time>] [-g <group list>] [-G <auth group list>] [-H <auth host list>] [-I <block time>] [-l <placement>] [-l <resource request>] [-m <mail events>] [-M <mail list>] [-N <reservation name>] [-q <destination>] [-r <recurrence rule>] [-R <start time>] [-u <user list>] [-U <auth user list>] [-W <attribute value list>]

For job-specific now reservations:

pbs_rsub [-I <block time>] [-m <mail events>] [-M <mail list>] --job <job ID>

For maintenance reservations:

pbs_rsub [-D <duration>] [-E <end time>] [-g <group list>] [-G <auth group list>] [-H <auth host list>] [-m <mail events>] [-M <mail list>] [-N <reservation name>] [-q <destination>] [-R <start time>] [-u <user list>] [-U <auth user list>] --hosts <host list>

For version information:

pbs_rsub --version

2.33.2 Description

The `pbs_rsub` command is used to create advance, standing, job-specific now, job-specific ASAP, or maintenance reservations. For creating job-specific start reservations, see "qsub" on page 213.

- An advance reservation reserves specific resources for the requested time period.
- A standing reservation reserves specific resources for recurring time periods.
- A job-specific now reservation reserves the resources being used by a specific job in case the job fails and needs to be re-submitted, allowing it to run again without having to wait to be scheduled. The reservation is created and starts running when a queued job starts running, or immediately when you use `pbs_rsub --job <job ID>` on a running job.
- A job-specific ASAP reservation is created from a queued job via `pbs_rsub -Wqmove=<job ID>`. The reservation runs as soon as possible, and the job is moved into the reservation. The reservation is created using the same resources as the job requested.
- A job-specific start reservation is created immediately using a running job's resources, and the job is moved into the reservation. You create job-specific start reservations using `qsub -Wcreate_resv_from_job=true` on a running job. See the `qsub` command.
- A maintenance reservation reserves the specified hosts for the specified time regardless of other circumstances.

Advance, standing, and job-specific reservations are "job reservations", to distinguish them from maintenance reservations. When a reservation is created, it has an associated queue.

To get information about a reservation, use the `pbs_rstat` command.

To delete a reservation, use the `pbs_rdel` command. Do not use the `qdel` command.

The behavior of the `pbs_rsub` command may be affected by any site hooks. Site hooks can modify the reservation's attributes.

2.33.2.1 Job Reservations

After an advance or standing reservation is requested, it is either confirmed or denied. A job-specific now reservation is created when the job is started and confirmed immediately. A job-specific ASAP reservation is scheduled as soon as possible. Once the reservation has been confirmed, authorized users submit jobs to the reservation's queue via `qsub` and `qmove`.

A confirmed job reservation will accept jobs at any time. The jobs in its queue can run only during the reservation period. Jobs in a single advance reservation or job-specific reservation can run only during the reservation's time slot, and jobs in a standing reservation can run only during the time slots of occurrences of the standing reservation.

When an advance reservation ends, all of its jobs are deleted, whether running or queued. When an occurrence of a standing reservation ends, only its running jobs are deleted; those jobs still in the queue are not deleted.

2.33.2.2 Maintenance Reservations

You can create maintenance reservations using `pbs_rsub --hosts <host list>`. Maintenance reservations are designed to make the specified hosts available for the specified amount of time, regardless of what else is happening:

- You can create a maintenance reservation that includes or is made up of vnodes that are down or offline.
- Maintenance reservations ignore the value of a vnode's resv_enable attribute.
- PBS immediately confirms any maintenance reservation.
- Maintenance reservations take precedence over other reservations; if you create a maintenance reservation that overlaps an advance or standing job reservation, the overlapping vnodes become unavailable to the job reservation, and the job reservation is in conflict with the maintenance reservation. PBS looks for replacement vnodes; see "Reservation Fault Tolerance" on page 432 in the PBS Professional Administrator's Guide.

PBS will not start any new jobs on vnodes overlapping or in a maintenance reservation. However, jobs that were already running on overlapping vnodes continue to run; you can let them run or requeue them.

You cannot specify place or select for a maintenance reservation; these are created by PBS:

- PBS creates the reservation's placement specification so that hosts are assigned exclusively to the reservation. The placement specification is always the following:

  -lplace=exclhost
- PBS sets the reservation's resv_nodes attribute value so that all CPUs on the reserved hosts are assigned to the maintenance reservation. The select specification is always the following:

  -lselect=host=<host1>:ncpus=<number of CPUs at host1>+host=<host2>:ncpus=<number of CPUs at host2>+...

Maintenance reservations are prefixed with *M*. A maintenance reservation ID has the format:

M<sequence number>.<server name>

You cannot create a recurring maintenance reservation.

Creating a maintenance reservation does not trigger a scheduling cycle.

You must have manager or operator privilege to create a maintenance reservation.

2.33.2.3 Requirements

When using `pbs_rsub` to request a standing, advance, or maintenance reservation, you must specify two of the following options: `-R`, `-E`, and `-D`. The resource request `-l walltime` can be used instead of the `-D` option.

If you want to run jobs in a reservation that will request exclusive placement, you must create the reservation with exclusive placement via `-l place=excl`.

2.33.3 Options to `pbs_rsub`

-D <duration>

Specifies reservation duration. If the start time and end time are the only times specified, this duration time is calculated.

Format: *Duration*

Default: none

-E <end time>

Specifies the reservation end time. If start time and duration are the only times specified, the end time value is calculated.

Format: *Datetime.*

Default: none

-g <group_list>

The *group list* is a comma-separated list of group names. The server uses entries in this list, along with an ordered set of rules, to associate a group name with the reservation. The reservation creator's primary group is automatically added to this list.

Format: *<group>@<hostname>[,<group>@<hostname> ...]*

-G <auth group list>

Comma-separated list of names of groups who can or cannot submit jobs to this reservation. Sets reservation's Authorized_Groups attribute to *auth group list*.

This list becomes the acl_groups list for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

Group names are interpreted in the context of the server host, not the context of the host from which the job is submitted.

See the Authorized_Groups reservation attribute in section 6.8, “Reservation Attributes”, on page 303.

Syntax:

[+|-]<group name>[,[+|-]<group name> ...]

Default: No groups are authorized to submit jobs

--hosts <host list>

Space-separated list of hosts to be included in maintenance reservation. PBS creates placement and resource requests. Placement is always *exclhost*, and all CPUs of requested hosts are assigned to maintenance reservation. Cannot be used with the `-l <placement>`, `-l <resource request>`, or `-I <block time>` options.

-H <auth host list>

Comma-separated list of hosts from which jobs can and cannot be submitted to this reservation. This list becomes the acl_hosts list for the reservation's queue. More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. If the reservation creator specifies this list, the creator's host is not automatically added to the list.

See the Authorized_Hosts reservation attribute in section 6.8, “Reservation Attributes”, on page 303.

Format: *[+|-]<hostname>[,[+|-]<hostname> ...]*

Default: All hosts are authorized to submit jobs

-l <block time>

Specifies interactive mode. The `pbs_rsub` command will block, up to *block time* seconds, while waiting for the reservation request to be confirmed or denied.

If *block time* is positive, and the reservation isn't confirmed or denied in the specified time, the ID string for the reservation is returned with the status "`UNCONFIRMED`".

If *block time* is negative, and a scheduler doesn't confirm or deny the reservation in the specified time, the reservation is deleted.

Cannot be used with `--hosts` option. Has no effect when used with `--job` option.

Format: *Integer*.

Default: *Not interactive*.

--job <job ID>

Immediately creates and confirms a *job-specific now reservation* on the same resources as the job (including resources inherited by the job), and places the job in the job-specific now reservation queue. Sets the job's create_resv_from_job attribute to *True*. Sets the now reservation's reserve_job attribute to the ID of the job from which the reservation was created, sets the reservation's Reserve_Owner attribute to the value of the job's Job_Owner attribute, sets the reservation's resv_nodes attribute to the job's exec_vnode attribute, sets the reservation's resources to match the job's schedselect attribute, and sets the reservation's Resource_List attribute to the job's Resource_List attribute.

The now reservation's duration and start time are the same as the job's walltime and start time. If the job is peer scheduled, the now reservation is created in the pulling complex.

Format: Boolean

Default: no default

Example:

```
pbs_rsub --job 1234.myserver
```

Can be used on running jobs only (jobs in the *R* state, with substate *42*).

Cannot be used with job arrays, jobs already in reservations, or other users' jobs.

-l <placement>

The *placement* specifies how vnodes are reserved. The place statement can contain the following elements, in any order:

-l place=[<arrangement>][:[<sharing>]][:[<grouping>]]

where

arrangement

Whether this reservation chunk is willing to share this vnode or host with other chunks from this reservation. One of *free* | *pack* | *scatter* | *vscatter*

sharing

Whether this reservation chunk is willing to share this vnode or host with other reservations or jobs. One of *excl* | *shared* | *exclhost*

grouping

Whether the chunks from this reservation should be placed on vnodes that all have the same value for a resource. Can have only one instance of *group=<resource name>*

free

Place reservation on any vnode(s).

pack

All chunks are taken from one host.

scatter

Only one chunk with any MPI processes is taken from a host. A chunk with no MPI processes may be taken from the same vnode as another chunk.

vscatter

Only one chunk is taken from any vnode. Each chunk must fit on a vnode.

excl

Only this reservation uses the vnodes chosen.

shared

This reservation can share the vnodes chosen.

exclhost

The entire host is allocated to the reservation.

group=<resource name>

Chunks are grouped according to the specified resource. All vnodes in the group must have a common value for *resource*, which can be either the built-in resource host or a custom vnode-level resource.

Resource name must be a string or a string array.

If you want to run jobs in the reservation that will request exclusive placement, you must create the reservation with exclusive placement via `-l place=excl`.

The place statement cannot start with a colon. Colons are delimiters; use them only to separate parts of a place statement, unless they are quoted inside resource values.

Note that vnodes can have sharing attributes that override reservation placement requests.

See section 6.10, “Vnode Attributes”, on page 320.

Cannot be used with `--hosts` option.

-l <resource request>

The *resource request* specifies the resources required for the reservation. These resources are used for the limits on the queue that is dynamically created for the reservation. The aggregate amount of resources for currently running jobs from this queue will not exceed these resource limits. Jobs in the queue that request more of a resource than the queue limit for that resource are not allowed to run. Also, the queue inherits the value of any resource limit set on the server, and these are used for the job if the reservation request itself is silent about that resource. A non-privileged user cannot submit a reservation requesting a custom resource which has been created to be invisible or read-only for users.

Resources are requested by using the `-l` option, either in chunks inside of selection statements, or in job-wide requests using *<resource name>=<value>* pairs.

Requesting resources in chunks:

-l select=[N:]<chunk>[+[N:]<chunk> ...]

where *N* specifies how many of that chunk, and a chunk is of the form:

<resource name>=<value>[:<resource name>=<value> ...]

Requesting job-wide resources:

-l <resource name>=<value>[,<resource name>=<value> ...]

Default: One chunk containing one CPU.

Cannot be used with `--hosts` option.

-m <mail events>

Specifies the set of events that cause mail to be sent to the list of users specified in the `-M <mail list>` option.

Format: string consisting of one of the following:

- Any combination of "*a*", "*b*", "*c*" or "*e*"
- The single character "*n*"

The following table lists the sub-options to the -m option:

Table 2-5: Sub-options to -m Option

Character	Meaning
a	Notify if the reservation is terminated for whatever reason
b	Notify when the reservation period begins
c	Notify when the reservation is confirmed
e	Notify when the reservation period ends
n	Send no mail. Cannot be used with any of *a*, *b*, *c*, or *e*.

Default: "*ac*".

-M <mail list>

The list of users to whom mail is sent whenever the reservation transitions to one of the states specified in the `-m <mail events>` option.

Format: *<username>[@<hostname>][,<username>[@<hostname>]...]*

Default: Reservation owner.

-N <reservation name>

Specifies a name for the reservation.

Format: *Reservation Name*. See "Reservation Name" on page 358.

Default: None.

-q <server>

Specifies the server at which to create the reservation.

Default: Default server

-r <recurrence rule>

Specifies rule for recurrence of standing reservations. Rule must conform to iCalendar syntax, and is specified using a subset of parameters from RFC 2445.

Valid syntax for *recurrence rule* takes one of two forms:

FREQ=<freq spec>;COUNT=<count spec>;<interval spec>

or

FREQ=<freq spec>;UNTIL=<until spec>;<interval spec>

where

freq spec

Frequency with which the standing reservation repeats. Valid values are:

WEEKLY|DAILY|HOURLY

count spec

The exact number of occurrences. Number up to 4 digits in length.

Format: *Integer*.

interval spec

Specifies interval. Format is one or both of:

BYDAY=MO|TU|WE|TH|FR|SA|SU

or

BYHOUR=0|1|2|...|23

When using both, separate them with a semicolon.

Elements specified in the recurrence rule override those specified in the arguments to the -R and -E options. For example, the BYHOUR specification overrides the hourly part of the -R option. For example, `-R 0730 -E 0830 ... BYHOUR=9` results in a reservation that starts at 9:30 and runs for 1 hour.

until spec

Occurrences will start up to but not after date and time specified. Format:

<YYYYMMDD>[T<HHMMSS>]

Note that the year-month-day section is separated from the hour-minute-second section by a capital T.

Requirements:

- The recurrence rule must be on one unbroken line and must be enclosed in double quotes.
- A start and end date must be used when specifying a recurrence rule. See the R and E options.
- The PBS_TZID environment variable must be set at the submission host. The format for PBS_TZID is a timezone location. Examples: `America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Calcutta`. See the PBS Professional User's Guide.
- Spaces are not allowed.

Examples of Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 -r "FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10 2008

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY;BYDAY=MO,TU,WE,TH,FR;UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r "FREQ=WEEKLY;BYDAY=MO,WE,FR;COUNT=3"
```

-R <start time>

Specifies reservation starting time. If the reservation's end time and duration are the only times specified, this start time is calculated.

If the day, *DD*, is not specified, it defaults to today if the time *hhmm* is in the future. Otherwise, the day is set to tomorrow. For example, if you submit a reservation with the specification `-R 1110` at 11:15 a.m., it is interpreted as being for 11:10am tomorrow. If the month portion, *MM*, is not specified, it defaults to the current month, provided that the specified day *DD*, is in the future. Otherwise, the month is set to next month. Similar rules apply to the two other optional, left-side components.

Format: *Datetime*

-u <user list>

Not used. Comma-separated list of user names.

Format: *<username>[@<hostname>][,<username>[@<hostname>] ...]*

Default: None.

-U <auth user list>

Comma-separated list of users who are and are not allowed to submit jobs to this reservation. Sets reservation's Authorized_Users attribute to *auth user list*.

This list becomes the acl_users attribute for the reservation's queue.

More specific entries should be listed before more general, because the list is read left-to-right, and the first match determines access. The reservation creator's username is automatically added to this list, whether or not the reservation creator specifies this list.

If both the Authorized_Users and Authorized_Groups reservation attributes are set, a user must belong to both in order to be able to submit jobs to this reservation.

See the Authorized_Users reservation attribute in section 6.8, "Reservation Attributes", on page 303.

Syntax:

[+|-]<username>[@<hostname>][,[+|-]<username>[@<hostname>]...]

Default: Job owner only.

-W <extended options>

This allows you to define other attributes for the reservation or perform other actions.

delete_idle_time=<allowed idle time>

Deletes the reservation after the specified amount of idle time. Applies to each instance of a standing reservation.

qmove=<job ID> [-I -<timeout>]

Takes as input a queued job, creates a *job-specific ASAP reservation* for the same resources the job requests, and moves the job into the reservation's queue. The reservation is scheduled to run as soon as possible.

When the reservation is created, it inherits its resources from the job, not from the resources requested through the `pbs_rsub` command.

You can use the `-I` option to specify a timeout for the conversion. If you use the `qmove` option to convert a job to a reservation, and the reservation is not confirmed within the timeout period, the reservation is deleted. The default timeout period is *10 seconds*. There is no option for this kind of reservation to be unconfirmed.

To specify the timeout, you must give a negative value for the `-I` option. For example, to specify a timeout of 300 seconds:

```
pbs_rsub -Wqmove=<job ID> -I -300
```

The default value for the delete_idle_time attribute for an ASAP reservation is 10 minutes.

The `-R` and `-E` options to `pbs_rsub` are disabled when using the qmove=<job ID> option.

Some shells require that you enclose a job array ID in double quotes.

Can be used on queued jobs only.

--version

The `pbs_rsub` command returns its PBS version information and exits. This option can only be used alone.

2.33.4 Output

The `pbs_rsub` command returns the reservation identifier.

Format for an advance or job-specific reservation:

R<sequence number>.<server name>

The associated queue's name is the prefix, *R<sequence number>*.

Format for a standing reservation:

S<sequence number>.<server name>

The associated queue's name is the prefix, *S<sequence number>*.

Format for a maintenance reservation:

M<sequence number>.<server name>

2.33.5 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "pbs_rstat" on page 94, "pbs_rdel" on page 90, "Reservation Attributes" on page 303

2.34 pbs_sched

Runs a PBS scheduler

2.34.1 Synopsis

pbs_sched [-a <alarm>] [-c <clientsfile>] [-d <home dir>] [-I <scheduler name>] [-L <logfile>] [-n] [-N] [-p <output file>] [-R <port number>] [-S <port number>] [-t <num threads>]

pbs_sched --version

2.34.2 Description

Runs the default scheduler or a multisched.

2.34.2.1 Required Permission

`pbs_sched` must be executed with root permission.

2.34.3 Options to `pbs_sched`

-a <alarm>

Deprecated. Overwrites value of sched_cycle_length scheduler attribute. Time in seconds to wait for a scheduling cycle to finish.

Format: *Time, in seconds*.

-c <clientsfile>

Add clients to this scheduler's list of known clients. The *clientsfile* contains single-line entries of the form

```
$clienthost <hostname>
```

Each *hostname* is added to the list of hosts allowed to connect to this scheduler. If *clientsfile* cannot be opened, this scheduler aborts. Path can be absolute or relative. If relative, it is relative to `PBS_HOME/sched_priv/`.

-d <home dir>

The directory in which this scheduler will run.

Default: `PBS_HOME/sched_priv`.

-I <scheduler name>

Name of scheduler to start. Required when starting a multisched.

-L <logfile>

The absolute path and filename of the log file. This scheduler writes its PBS version and build information to *logfile* whenever it starts up or *logfile* is rolled to a new file.

See the -d option.

Default: This scheduler opens a file named for the current date in the `PBS_HOME/sched_log` directory.

-n

Tells this scheduler to not restart itself if it receives a sigsegv or a sigbus. A scheduler by default restarts itself if it receives either of these two signals more than five minutes after starting. A scheduler does not restart itself if it receives either one within five minutes of starting.

-N

Runs the scheduler in standalone mode.

-p <output file>

Any output which is written to standard out or standard error is written to *output file*. The pathname can be absolute or relative, in which case it is relative to PBS_HOME/sched_priv.

See the -d option.

Default: PBS_HOME/sched_priv/sched_out

-R <port number>

The port for MoM to use. If this option is not given, the port number is taken from PBS_MANAGER_SERVICE_PORT, in pbs.conf.

Default: *15003*

-S <port number>

The port for this scheduler to use.

Required when starting a multisched.

For the default scheduler, if this option is not specified, the default port is taken from PBS_SCHEDULER_SERVICE_PORT, in pbs.conf.

Default value for default scheduler: *15004*

Default value for multisched: none

-t <num threads>

Specifies number of threads for this scheduler.

Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of *num threads*.

Overrides PBS_SCHED_THREADS environment variable and PBS_SCHED_THREADS parameter in pbs.conf.

Valid values: *>=1*

Default: one thread

--version

The pbs_sched command returns its PBS version information and exits. This option can only be used alone.

2.34.4 Signal Handling

All signals are ignored until the end of the cycle. Most signals are handled in the standard UNIX fashion.

SIGHUP

This scheduler closes and reopens its log file and rereads its configuration file if one exists.

SIGALRM, SIGBUS, etc.

Ignored until end of scheduling cycle. This scheduler quits.

SIGINT and SIGTERM

This scheduler closes its log file and shuts down.

All other signals have the default action installed.

2.34.5 Exit Status

Zero

Upon normal termination

2.34.6 See Also

The PBS Professional Administrator's Guide

2.35 pbs_server

Starts a PBS batch server

2.35.1 Synopsis

pbs_server [-A <acctfile>] [-a <active>] [-C] [-d <config path>] [-e <mask>] [-F <delay>] [-L <logfile>] [-M <MoM port>] [-N] [-p <port number>] [-R <MoM RM port>] [-S <default scheduler port>] [-s <replacement string>] [-t <restart type>]

pbs_server --version

2.35.2 Description

The `pbs_server` command starts a batch server on the local host. Typically, this command is in a local boot file such as `/etc/rc.local`. If the batch server is already running, `pbs_server` exits with an error.

2.35.2.1 Required Permission

To ensure that the `pbs_server` command is not runnable by the general user community, the server runs only if its real and effective UID is zero. You must be root.

2.35.3 Options to `pbs_server`

-A <acctfile>

Specifies an absolute path name for the file to use as the accounting file. If not specified, the file is named for the current date in the `PBS_HOME/server_priv/accounting` directory.

-a <value>

When *True*, the server is in state "*active*" and the default scheduler is called to schedule jobs. When *False*, the server is in state "*idle*" and the default scheduler is not called to schedule jobs. Sets the server's scheduling attribute. If this option is not specified, the server uses the previously specified *value* for the scheduling attribute.

Format: *Boolean*

-C

The server starts up, creates the database, and exits. Windows only.

-d <config path>

Specifies the absolute path to the directory containing the server configuration files, `PBS_HOME`. A host may have multiple servers. Each server must have a different configuration directory. The default configuration directory is specified in $PBS_HOME, and is typically `/var/spool/pbs`.

-e <mask>

Specifies a log event mask to be used when logging. See "log_events" in section 6.6, “Server Attributes”, on page 281.

-F <delay>

Specifies the number of seconds that the secondary server should wait before taking over when it believes the primary server is down. If the number of seconds is specified as *-1*, the secondary will make one attempt to contact the primary and then become active.

Default: *30 seconds*

-L <logfile>

Specifies the absolute path name for the log file. If not specified, the file is named for the current date in the PBS_HOME/server_logs directory. PBS_HOME is specified in the $PBS_HOME environment variable or in /etc/pbs.conf; see the -d option.

-M <MoM port>

Specifies the hostname and/or port number on which the server should connect to MoM. The option argument, *MoM port*, uses the syntax:

[<hostname>][:<port number>]

If *hostname* not specified, the local host is assumed.

If *port number* is not specified, the default port is assumed.

See the -M option in section 2.23, "pbs_mom", on page 71.

Default: *15002*

-N

Runs the server in standalone mode.

-p <port number>

Specifies the port number on which the server is to listen for batch requests. If multiple servers are running on a single host, each must have its own unique port number. This option is for testing with multiple batch systems on a single host.

Format: Integer port number

Default: *15001*

-R <MoM RM port>

Specifies the port number on which the server should query the up/down status of MoM. See the -R option in section 2.23, "pbs_mom", on page 71.

Default: *15003*

-S <default scheduler port>

Specifies the port number to which the server should connect when contacting the default scheduler. The option argument, *default scheduler port*, uses the syntax:

[<hostname>][:<port number>]

If *hostname* not specified, the local host is assumed. If *port number* is not specified, the default port is assumed.

Default: *15004*

-s <replacement string>

Specifies the string to use when replacing spaces in accounting entity names. Only available under Windows.

-t <restart type>

Specifies behavior when the server restarts. The *restart type* argument is one of the following:

cold

All jobs are purged. Positive confirmation is required before this direction is accepted.

create

The server discards any existing configuration files: server, nodes, queues, and jobs, and initializes configuration files to the default values. The default scheduler is idled (scheduling is set to *False*). Any multi-scheds are deleted.

hot
All jobs in the *Running* state are retained in that state. Any job that was requeued into the *Queued* state from the *Running* state when the server last shut down is run immediately, assuming the required resources are available. This returns the server to the same state as when it went down. After those jobs are restarted, normal scheduling takes place for all remaining queued jobs. All other jobs are retained in their current state.

If a job cannot be restarted immediately because of a missing resource, such as a vnode being down, the server attempts to restart it periodically for up to 5 minutes. After that period, the server will revert to a normal state, as if *warm* started, and will no longer attempt to restart any remaining jobs which were running prior to the shutdown.

updatedb
Updates format of PBS data from the previous format to the data service format.

warm
All jobs in the *Running* state are retained in that state. All other jobs are maintained in their current state. The default scheduler typically chooses new jobs for execution. *warm* is the default if `-t` is not specified.

--version
The `pbs_server` command returns its PBS version information and exits. This option can only be used alone.

2.35.4 Files

`$PBS_HOME/server_priv`
Default directory for configuration files.

`$PBS_HOME/server_logs`
Directory for log files recorded by the server.

2.35.5 Signal Handling for pbs_server

When it receives the following signals, the server performs the following actions:

SIGHUP
The current server log and accounting log are closed and reopened. This allows for the prior log to be renamed and a new log started from the time of the signal.

SIGTERM
Causes a rapid orderly shutdown of `pbs_server`, identical to "`qterm -t quick`".

SIGSHUTDN
On systems where SIGSHUTDN is defined, causes an orderly "*quick*" shutdown of the server.

SIGPIPE, SIGUSR1, SIGUSR2
These signals are ignored.

All other signals have their default behavior installed.

2.35.6 Diagnostic Messages

The server records a diagnostic message in a log file for any error occurrence. The log files are maintained in the `server_logs` directory below the home directory of the server. If the log file cannot be opened, the diagnostic message is written to the system console. The server writes its PBS version and build information to the logfile whenever it starts up or the logfile is rolled to a new file.

2.35.7 Stopping the PBS Server

2.35.7.1 Stopping the Server on Linux

Use the `qterm` command (see section 2.60, “qterm”, on page 233):

```
qterm
```

or send a SIGTERM:

```
kill <server PID>
```

2.35.8 Exit Status

Zero
: When the server has run in the background and then exits

Greater than zero
: If the server daemon fails to begin batch operation

2.35.9 See Also

The *PBS Professional Administrator's Guide*

2.36 pbs_snapshot

Linux only. Captures PBS data to be used for diagnostics

2.36.1 Synopsis

pbs_snapshot -h, --help

pbs_snapshot -o <output directory path> [--accounting-logs=<number of days>] [--additional-hosts=<hostname list>] [--basic] [--daemon-logs=<number of days>] [-H <server host>] [-l <log level>] [--map=<file path>] [--obfuscate] [--with-sudo]

pbs_snapshot --version

2.36.2 Description

You use `pbs_snapshot` to capture PBS data for diagnostics. This tool is written in Python and uses PTL libraries, including `PBSSnapUtils`, to extract the data. You can optionally anonymize the PBS data. The `pbs_snapshot` command captures data from all multischeds. The command detects which daemon or daemons are running on the host where it is collecting information, and captures daemon and system data accordingly. If no PBS daemons are running, the command collects system information. The output tarball contains information about the host specified via the `-H` option, or if that is not specified, the local host. If you specify additional hosts, the command creates a tarball for each additional host and includes it as a sub-tarball in the output.

If you want to capture only PBS configuration information, use the `--basic` option.

2.36.2.1 Required Privilege

The `pbs_snapshot` command allows you to use the `sudo` infrastructure provided by the PTL framework to capture root-owned information via `--with-sudo`. All other information is collected as a normal user. If you need to run `pbs_snapshot` as a non-privileged user, and without using the PTL `--with-sudo` infrastructure, you must be root if you want root-owned information to be collected.

2.36.2.2 Restrictions

The `pbs_snapshot` command is not available on Windows.

2.36.3 Options to pbs_snapshot

--accounting-logs=<number of days>

Specifies number of days of accounting logs to be collected; this count includes the current day.

Value of *number of days* must be >=0:

- If number of days is 0, no logs are captured.
- If number of days is 1, only the logs for the current day are captured.

Default: `pbs_snapshot` collects 30 days of accounting logs

--additional-hosts=<hostname list>

Specifies that pbs_snapshot should gather data from the specified list of additional hosts. Launches the pbs_snapshot command on each specified host, creates a tarball there named *<hostname>_snapshot.tgz*, and includes it as a sub-tarball in the output for the main output. If you use the --with-sudo option, each launched copy uses that option as well.

The command does not query the server when it runs at a non-server host.

The command collects a full snapshot, including the following information:

- Daemon logs, for the number of days of logs being captured, specified via the --daemon-logs=<number of days> option
- The PBS_HOME/<daemon>_priv directory
- Accounting logs if server daemon runs on host
- System information

Format for *hostname list* is a comma-separated list of one or more hostnames:

<hostname>[, <hostname> ...]

--basic

Captures PBS configuration information only. Captures the following:

Table 2-6: PBS Configuration Information Captured with --basic Option

Output File	Description of Captured Information
server/qstat_Bf.out	Output of qstat -Bf
server/qstat_Qf.out	Output of qstat -Qf
scheduler/qmgr_lsched.out	Output of qmgr -c 'list sched'
node/pbsnodes_va.out	Output of pbsnodes -va
reservation/pbs_rstat_f.out	Output of pbs_rstat -f
job/qstat_f.out	Output of qstat -f
hook/qmgr_lpbshook.out	Output of qmgr -c 'list pbshook'
sched_priv/ for each scheduler instance	Copy of each scheduler's sched_priv directory
server_priv/resourcedef	Copy of server_priv/resourcedef file
pbs.conf	Copy of /etc/pbs.conf on server host
pbs_snapshot.log	Log of pbs_snapshot execution
ctime	Timestamp of when the snapshot was taken

Can be combined with other options such as --accounting-logs and --daemon-logs in order to capture additional information.

--daemon-logs=<number of days>

Specifies number of days of daemon logs to be collected; this count includes the current day.

Value of *number of days* must be >=0:

- If number of days is 0, no logs are captured.
- If number of days is 1, only the logs for the current day are captured.

Default: `pbs_snapshot` collects 5 days of daemon logs

-h, --help

Prints usage and exits.

-H <hostname>

Specifies hostname for host whose retrieved data is to be at the top level in the output tarball. If not specified, `pbs_snapshot` puts data for the local host at the top level in the output tarball.

-l <log level>

Specifies level at which `pbs_snapshot` writes its log. The log file is `pbs_snapshot.log`, in the output directory path specified using the `-o <output directory path>` option.

Valid values, from most comprehensive to least: *DEBUG2, DEBUG, INFOCLI2*, *INFOCLI, INFO*, *WARNING*, *ERROR*, *FATAL*

Default: *INFOCLI2*

--map=<file path>

Specifies path for file containing obfuscation map, which is a <key>:<value> pair-mapping of obfuscated data. Path can be absolute or relative to current working directory.

Default: `pbs_snapshot` writes its obfuscation map in a file called "`obfuscate.map`" in the location specified via the `-o <output directory path>` option.

Can only be used with the `--obfuscate` option.

--obfuscate

Obfuscates (anonymizes) or deletes sensitive PBS data captured by `pbs_snapshot`.

- Obfuscates the following data: euser, egroup, project, Account_Name, operators, managers, group_list, Mail_Users, User_List, server_host, acl_groups, acl_users, acl_resv_groups, acl_resv_users, sched_host, acl_resv_hosts, acl_hosts, Job_Owner, exec_host, Host, Mom, resources_available.host, resources_available.vnode
- Deletes the following data: Variable_List, Error_Path, Output_Path, mail_from, Mail_Points, Job_Name, jobdir, Submit_arguments, Shell_Path_List

--version

The `pbs_snapshot` command prints its PBS version information and exits. Can only be used alone.

--with-sudo

Uses the PTL `sudo` infrastructure in order capture root-owned information via `sudo`. (Information not owned by root is captured using normal privilege, not root privilege.) With this option, you do not need to prefix your `pbs_snapshot` command with `sudo`, and you do not need root privilege.

2.36.4 Arguments to pbs_snapshot

-o <output directory path>

Path to directory where pbs_snapshot writes its output tarball. Required. Path can be absolute or relative to current working directory.

For example, if you specify "-o /temp", pbs_snapshot writes "/temp/snapshot_<timestamp>.tgz".

The output directory path must already exist.

2.36.5 Output

2.36.5.1 Output Location

You must use the -o <output directory path> option to specify the directory where pbs_snapshot writes its output. The path can be absolute or relative to current working directory. The output directory must already exist. As an example, if you specify "-o /temp", pbs_snapshot writes "/temp/snapshot_<timestamp>.tgz".

2.36.5.2 Output Contents

The pbs_snapshot command writes the output for the local host and each specified remote host as a tarball. Tarballs for remote hosts are included in the main tarball.

The command captures JSON output from qstat -f -F json and pbsnodes -av -F json.

The main tarball contains the following directory structure, files, and tarballs:

Table 2-7: Contents of Snapshot

Directory or File	Directory Contents	Description
server/		
	qstat_B.out	Output of qstat -B
	qstat_Bf.out	Output of qstat -Bf
	qmgr_ps.out	Output of qmgr print server
	qstat_Q.out	Output of qstat -Q
	qstat_Qf.out	Output of qstat -Qf
	qmgr_pr.out	Output of qmgr print resource
server_priv/	Copy of the PBS_HOME/server_priv directory. Core files are captured separately; see core_file_bt/.	
	accounting/	Accounting logs from PBS_HOME/server_priv/accounting/ directory for the number of days specified via --accounting-logs option
server_logs/	Server logs from the PBS_HOME/server_logs directory for the number of days specified via --daemon-logs option	

Table 2-7: Contents of Snapshot

Directory or File	Directory Contents	Description
job/		
	qstat.out	Output of qstat
	qstat_f.out	Output of qstat -f
	qstat_f_F_json.out	Output of qstat -f -F json
	qstat_t.out	Output of qstat -t
	qstat_tf.out	Output of qstat -tf
	qstat_x.out	Output of qstat -x
	qstat_xf.out	Output of qstat -xf
	qstat_ns.out	Output of qstat -ns
	qstat_fx_F_dsv.out	Output of qstat -fx -F dsv
	qstat_f_F_dsv.out	Output of qstat -f -F dsv
node/		
	pbsnodes_va.out	Output of pbsnodes -va
	pbsnodes_a.out	Output of pbsnodes -a
	pbsnodes_avSj.out	Output of pbsnodes -avSj
	pbsnodes_aSj.out	Output of pbsnodes -aSj
	pbsnodes_avS.out	Output of pbsnodes -avS
	pbsnodes_aS.out	Output of pbsnodes -aS
	pbsnodes_aFdsv.out	Output of pbsnodes -aF dsv
	pbsnodes_avFdsv.out	Output of pbsnodes -avF dsv
	pbsnodes_avFjson.out	Output of pbsnodes -avF json
	qmgr_pn_default.out	Output of qmgr print node @default
mom_priv/	Copy of the PBS_HOME/mom_priv directory. Core files are captured separately; see core_file_bt/.	
mom_logs/	MoM logs from the PBS_HOME/mom_logs directory for the number of days specified via --daemon-logs option	
comm_logs/	Comm logs from the PBS_HOME/comm_logs directory for the number of days specified via --daemon-logs option	
sched_priv/	Copy of the PBS_HOME/sched_priv directory, with all files. Core files are not captured; see core_file_bt/.	
sched_logs/	Scheduler logs from the PBS_HOME/sched_log directory for the number of days specified via --daemon-logs option	
sched_priv_<multi-sched name>/	Copy of the PBS_HOME/sched_priv_<multisched name> directory, with all files. Core files are not captured; see core_file_bt/.	

Table 2-7: Contents of Snapshot

Directory or File	Directory Contents	Description
sched_logs_<multi-sched name>/	Multisched logs from the PBS_HOME/sched_log_<multisched name> directory for the number of days specified via --daemon-logs option	
reservation/		
	pbs_rstat_f.out	Output of pbs_rstat -f
	pbs_rstat.out	Output of pbs_rstat
scheduler/		
	qmgr_lsched.out	Output of qmgr list sched
hook/		
	qmgr_ph_default.out	Output of qmgr print hook @default
	qmgr_lpbshook.out	Output of qmgr list pbshook
datastore/		
	pg_log/	Copy of the PBS_HOME/datastore/pg_log directory for the number of days specified via --daemon-logs option
core_file_bt/	Stack backtrace from core files	
	sched_priv/	Files containing the output of thread apply all backtrace full on all core files captured from PBS_HOME/sched_priv
	sched_priv_<multisched name>	Files containing the output of thread apply all backtrace full on all core files captured from PBS_HOME/sched_priv_<multisched name>
	server_priv/	Files containing the output of thread apply all backtrace full on all core files captured from PBS_HOME/server_priv
	mom_priv/	Files containing the output of thread apply all backtrace full on all core files captured from PBS_HOME/mom_priv
	misc/	Files containing the output of thread apply all backtrace full on any other core files found inside PBS_HOME

Table 2-7: Contents of Snapshot

Directory or File	Directory Contents	Description
system/		
	pbs_probe_v.out	Output of `pbs_probe -v`
	pbs_hostn_v.out	Output of `pbs_hostn -v $(hostname)`
	pbs_environment	Copy of PBS_HOME/pbs_environment file
	os_info	Information about the OS
	process_info	List of processes running on the system when the snapshot was taken. Output of `ps -aux | grep [p]bs` on Linux systems, or `tasklist /v` on Windows systems
	ps_leaf.out	Output of `ps -leaf`. Linux only.
	lsof_pbs.out	Output of `lsof | grep [p]bs`. Linux only.
	etc_hosts	Copy of /etc/hosts file. Linux only.
	etc_nsswitch_conf	Copy of /etc/nsswitch.conf file. Linux only.
	vmstat.out	Output of the command `vmstat`. Linux only.
	df_h.out	Output of the command `df -h`. Linux only.
	dmesg.out	Output of the `dmesg` command. Linux only.
pbs.conf	Copy of the pbs.conf file on the server host	
ctime	Contains the time in seconds since epoch when the snapshot was taken	
pbs_snapshot.log	Log messages written by `pbs_snapshot`	
<remote host-name>.tgz	Tarball of output from running the `pbs_snapshot` command at a remote host	

2.36.6 Examples

`pbs_snapshot -o /tmp`

Writes a snapshot to /temp/snapshot_<timestamp>.tgz that includes 30 days of accounting logs and 5 days of daemon logs from the server host.

`pbs_snapshot --daemon-logs=1 --accounting-logs=1 -o /tmp --obfuscate --map=mapfile.txt`

Writes a snapshot to /temp/snapshot_<timestamp>.tgz that includes 1 day of accounting and daemon logs. Obfuscates the data and stores the data mapping in the map file named "mapfile.txt".

2.37 pbs_tclsh

Deprecated. TCL shell with TCL-wrapped PBS API

2.37.1 Synopsis

pbs_tclsh

pbs_tclsh --version

2.37.2 Description

The `pbs_tclsh` command starts a version of the TCL shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in "TCL/tk Interface" on page 103 in the PBS Professional Programmer's Guide.

The `pbs_tclsh` command is used to query MoM. For example:

```
> pbs_tclsh
tclsh> openrm <hostname>
<file descriptor>
tclsh> addreq <file descriptor> "loadave"
tclsh> getreq <file descriptor>
<load average>
tclsh> closereq <file descriptor>
```

2.37.2.1 Required Permission

Root privilege is required in order to query MoM for dynamic resources. Root privilege is not required in order to query MoM for built-in resources and site-defined static resources.

2.37.3 Options

--version

The `pbs_tclsh` command returns its PBS version information and exits. This option can only be used alone.

2.37.4 Standard Error

The `pbs_tclsh` command writes a diagnostic message to standard error for each error occurrence.

2.37.5 See Also

The PBS Professional Administrator's Guide, the PBS Programmer's Guide, "pbs_wish" on page 124

2.38 pbs_tmrsh

TM-enabled replacement for `rsh/ssh` for use by MPI implementations

2.38.1 Synopsis

pbs_tmrsh <hostname> [-l <username>] [-n] <command> [<args> ...]

pbs_tmrsh --version

2.38.2 Description

The `pbs_tmrsh` command attempts to emulate an "`rsh`" connection to the specified host, via underlying calls to the Task Management (TM) API. The program is intended to be used during MPI integration activities, and not by end-users.

Running "`pbs_tmrsh` <hostname> `<command>`" causes a PBS task to be started on *hostname* running *command*.

2.38.2.1 Requirements for Environment Variables

The environment variables used by the two MPI implementations to point to the `rsh` work-alike (MPI_REMSH in the case of HP and P4_RSHCOMMAND for MPICH) must be set in the job environment and point to the full path for `pbs_tmrsh`.

The file `$PBS_HOME/pbs_environment` should contain the environment variable PATH in which to search for the program executable. This applies to both Windows and Linux. It is expected that a full path will be specified for the *command* and the PATH variable will not be needed.

2.38.3 Options

-l <username>
: Specifies the username under which to execute the task. If used, *username* must match the username running the `pbs_tmrsh` command.

-n
: A no-op; provided for MPI implementations that expect to call `rsh` with the "`-n`" option.

--version
: The `pbs_tmrsh` command returns its PBS version information and exits. This option can only be used alone.

2.38.4 Operands

command
: Specifies command to be run as a PBS task.

hostname
: Specifies host on which to run PBS task. The *hostname* may be specified in IP-dot-address form.

2.38.5 Output and Error

Output and errors are written to the PBS job's output and error files, not to standard output/error.

The pbs_tmrsh command writes a diagnostic message to the PBS job's error file for each error occurrence.

2.38.6 Exit Status

The pbs_tmrsh program exits with the exit status of the remote command or with *255* if an error occurred. This is because ssh works this way.

2.38.7 See Also

The PBS Professional Administrator's Guide, "pbs_attach" on page 55, "TM Library Routines", on page 93 of the PBS Professional Programmer's Guide

2.39 pbs_topologyinfo

Reports topological information

2.39.1 Synopsis

pbs_topologyinfo (-a | --all) [(-l | --license) | (-s | --sockets)]

pbs_topologyinfo (-l | --license) <vnode name> [<vnode name> ...]

pbs_topologyinfo (-s | --sockets) <vnode name> [<vnode name> ...]

pbs_topologyinfo -h | --help

2.39.2 Description

The `pbs_topologyinfo` command reports topological information for one or more vnodes. This information is used for licensing purposes. To use the command, you must specify what kind of topological information you want. The command reports only the requested information.

This command must be run on the server host.

2.39.2.1 Usage

`pbs_topologyinfo -al` reports number of node licenses needed for all vnodes.

`pbs_topologyinfo -l <vnode name>` reports number of node licenses needed for *vnode name*.

`pbs_topologyinfo -as` reports socket counts for all vnodes that have reported sockets.

`pbs_topologyinfo -s <vnode name>` reports socket count for vnode *vnode name*.

2.39.2.2 Prerequisites

Before you use this command, the server and MoMs must be configured so that they can contact each other, and must have been run.

2.39.2.3 Required Privilege for `pbs_topologyinfo`

This command can be run only by root or Admin on Windows.

2.39.3 Options for `pbs_topologyinfo`

-a, --all

Reports requested topological information for all vnodes. When this option is used alone, the command does not report any information.

-h, --help

Prints usage and exits.

-l, --license [<vnode name(s)>]

Reports number of node licenses required. If you specify *vnode name(s)*, the command reports node licenses needed for the specified vnode(s) only.

-s, --sockets [<vnode name(s)>]

Reports derived socket counts. If you specify *vnode name(s)*, the command reports socket count information for the specified vnode(s) only.

(no options)

Does not report any information.

2.39.4 Errors

If you specify an invalid vnode name, the command prints a message to standard error.

2.39.5 Operands

vnode name [<vnode name> ...]

Name(s) of vnode(s) about which to report.

2.39.6 Exit Status

0

Success

1

Any error following successful command line processing

2.39.7 Standard Error

If an invalid vnode name is specified, a message is printed to standard error.

2.39.8 See Also

The PBS Professional Administrator's Guide

2.40 pbs_wish

Deprecated. TK window shell with TCL-wrapped PBS API

2.40.1 Synopsis

pbs_wish

pbs_wish --version

2.40.2 Description

The `pbs_wish` command is a version of the TK window shell which includes wrapped versions of the PBS external API. The PBS TCL API is documented in "TCL/tk Interface" on page 103 in the PBS Professional Programmer's Guide.

2.40.3 Options

--version

The `pbs_wish` command returns its PBS version information and exits. This option can only be used alone.

2.40.4 Standard Error

The `pbs_wish` command writes a diagnostic message to standard error for each error occurrence.

2.40.5 See Also

The PBS Professional Administrator's Guide, "pbs_tclsh" on page 119

2.41 printjob

Prints job information

2.41.1 Synopsis

printjob [-a | -s] <job ID>

printjob [-a] <file path> [<file path>...]

printjob --version

2.41.2 Description

Prints job information. This command is mainly useful for troubleshooting, as during normal operation, the "qstat" command is the preferred method for displaying job-specific data and attributes. The server and MoM do not have to be running to execute this command.

2.41.2.1 Usage

For a running job, you can run this command at any host using a job ID, and you can run this command at any execution host where the job is running using a .JB file path.

For a finished job, if job history is enabled, you can run this command at the server using the job ID.

When querying the server, you must use the job ID, and the data service must be running.

Results will vary depending on whether you use the job ID or a .JB file, and on which execution host you query with a .JB file.

2.41.2.2 Permissions

In order to execute `printjob`, you must have root or Windows Administrator privilege.

2.41.3 Options to `printjob`

(no options>
: Prints all job data including job attributes.

-a
: Suppresses the printing of job attributes. Cannot be used with `-s` option.

-s
: Prints out the job script only. Can be used at server or primary execution host. Cannot be used with `-a` option. Must be used with a job ID.

--version
: The `printjob` command returns its PBS version information and exits. This option can only be used alone.

2.41.4 Operands for `printjob`

file path

The `printjob` command accepts one or more *file path* operands at the execution host. Files are found in `PBS_HOME/mom_priv/jobs/` on the primary execution host. File path must include full path to file. Cannot be used with `-s` option.

job ID

The `printjob` command accepts a *job ID* at the server host. The format is described in "Job ID, Job Identifier" on page 355. Data service must be running.

2.41.5 Standard Error

The `printjob` command writes a diagnostic message to standard error for each error occurrence.

2.41.6 Exit Status

Zero

Upon successful processing of all operands presented

Greater than zero

If the `printjob` command fails to process any operand

2.41.7 See Also

The PBS Professional Administrator's Guide, "qstat" on page 197

2.42 qalter

Alters a PBS job

2.42.1 Synopsis

qalter [-a <date and time>] [-A <account string>] [-c <checkpoint spec>] [-e <error path>] [-h <hold list>] [-j <join>] [-k <discard>] [-l <resource list>] [-m <mail events>] [-M <user list>] [-N <name>] [-o <output path>] [-p <priority>] [-P <project>] [-r <y|n>] [-R <remove options>] [-S <path list>] [-u <user list>] [-W <additional attributes>] <job ID> [<job ID> ...]

qalter --version

2.42.2 Description

The `qalter` command is used to alter one or more PBS batch jobs. Each of certain job attributes can be modified using the `qalter` option for that attribute. You can alter a job or a job array, but not a subjob or range of subjobs.

2.42.2.1 Required Privilege

A non-privileged user can alter their own jobs, whether they are queued or running. An Operator or Manager can alter any job, whether it is queued or running.

A non-privileged user can only lower resource requests. An Operator or Manager can raise or lower resource requests.

2.42.2.2 Modifying Resources and Job Placement

A Manager or Operator may lower or raise requested resource limits, except for per-process limits such as pcput and pmem, because these are set when the process starts, and enforced by the kernel. A non-privileged user can only lower resource requests.

The `qalter` command cannot be used by a non-privileged user to alter a custom resource which has been created to be invisible or read-only for users.

If a job is running, the only resources that can be modified are cput, walltime, min_walltime, and max_walltime.

If a job is queued, any resource mentioned in the options to the `qalter` command can be modified, but requested modifications must fit within the limits set at the server and queue for the amount of each resource allocated for queued jobs. If a requested modification does not fit within these limits, the modification is rejected.

A job's resource request must fit within the queue's and server's resource run limits. If a modification to a resource exceeds the amount of the resource allowed by the queue or server to be used by running jobs, the job is never run.

Requesting resources includes setting limits on resource usage and controlling how the job is placed on vnodes.

See Chapter 5, "List of Built-in Resources", on page 259.

2.42.2.2.i Syntax for Modifying Resources and Job Placement

Resources are modified by using the `-l` option, either in chunks inside of selection statements, or in job-wide requests using *<resource name>=<value>* pairs. The selection statement is of the form:

-l select=[<N>:]<chunk>[+[<N>:]<chunk> ...]

where *N* specifies how many of that chunk, and a *chunk* is of the form:

<resource name>=<value>[:<resource name>=<value> ...]

Job-wide <*resource name*>=<*value*> requests are of the form:

-l <resource name>=<value>[,<resource name>=<value> ...]

2.42.2.2.ii The Place Statement

You choose how your chunks are placed using the *place statement*. The *place statement* can contain the following elements, in any order:

-l place=[<arrangement>][: <sharing>][: <grouping>]

where

arrangement

Whether this chunk is willing to share this vnode or host with other chunks from the same job. One of *free* | *pack* | *scatter* | *vscatter*

sharing

Whether this this chunk is willing to share this vnode or host with other jobs. One of *excl* | *shared* | *exclhost*

grouping

Whether the chunks from this job should be placed on vnodes that all have the same value for a resource. Can have only one instance of *group=<resource name>*

free

Place job on any vnode(s).

pack

All chunks are taken from one host.

scatter

Only one chunk with any MPI processes is taken from a host. A chunk with no MPI processes may be taken from the same vnode as another chunk.

vscatter

Only one chunk is taken from any vnode. Each chunk must fit on a vnode.

excl

Only this job uses the vnodes chosen.

shared

This job can share the vnodes chosen.

exclhost

The entire host is allocated to the job.

group=<resource name>

Chunks are grouped according to a resource. All vnodes in the group must have a common value for *resource*, which can be either the built-in resource host or a custom vnode-level resource. The *resource name* must be a string or a string array.

The *place statement* cannot begin with a colon. Colons are delimiters; use them only to separate parts of a place statement, unless they are quoted inside resource values.

Note that vnodes can have sharing attributes that override job placement requests. See section 6.10, “Vnode Attributes”, on page 320.

For more on resources, resource requests, usage limits, and job placement, see "Using PBS Resources" on page 229 in the PBS Professional Administrator’s Guide and "Allocating Resources & Placing Jobs", on page 49 of the PBS Professional User’s Guide.

2.42.2.3 Modifying Attributes

The user alters job attributes via options to the `qalter` command. Each `qalter` option changes a job attribute.

The behavior of the `qalter` command may be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

To modify the max_run_subjobs attribute, use `qalter -Wmax_run_subjobs=<new value> <job ID>`.

2.42.2.4 Caveats and Restrictions for Altering Jobs

- When you lengthen the walltime of a running job, make sure that the new walltime will not interfere with any existing reservations etc.
- If any of the modifications to a job fails, none of the job's attributes is modified.
- A job that is in the process of provisioning cannot be altered.

2.42.3 Options to `qalter`

-a <date and time>

Changes the point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's Execution_Time attribute to *date and time*.

Format: *Datetime*

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day *DD* is the 5th, the month *MM* is set to the current month.

If a specified portion has already passed, the next-larger portion is set to one after the current date. For example, if the day *DD* is not specified, but the hour *hh* is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day *DD* is set to tomorrow.

The job's Execution_Time attribute can be altered after the job has begun execution, in which case it will not take effect until the job is rerun.

-A <account string>

Replaces the accounting string associated with the job. Used for labeling accounting data. Sets job's Account_Name attribute to *account string*. This attribute cannot be altered once the job has begun execution.

Format: *String*

-c <checkpoint spec>

Changes when the job will be checkpointed. Sets job's Checkpoint attribute. An $action script is required to checkpoint the job. This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

The argument *checkpoint spec* can take one of the following values:

c

Checkpoint at intervals, measured in CPU time, set on job's execution queue. If no interval set at queue, job is not checkpointed.

c=<minutes of CPU time>

Checkpoint at intervals of specified number of minutes of job CPU time. This value must be greater than zero. If interval specified is less than that set on job's execution queue, queue's interval is used.

Format: *Integer*

w

Checkpoint at intervals, measured in walltime, set on job's execution queue. If no interval set at queue, job is not checkpointed.

w=<minutes of walltime>

Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less that that set on the job's execution queue, the queue's interval is used.

Format: *Integer*

n

No checkpointing.

s

Checkpoint only when the server is shut down.

u

Unset. Defaults to behavior when interval argument is set to s.

Default: *u*

Format: *String*

-e <error path>

Replaces the path to be used for the job's standard error stream. Sets job's Error_Path attribute to *error path*. Overridden by -k option.

Format: *[<hostname>:]<path>*

The *error path* is interpreted as follows:

path

If *path* is relative, it is taken to be relative to the current working directory of the `qalter` command, where it is executing on the current host.

If *path* is absolute, it is taken to be an absolute path on the current host where the `qalter` command is executing.

hostname:path

If *path* is relative, it is taken to be relative to the user's home directory on the host named *hostname*.

If *path* is absolute, it is the absolute path on the host named *hostname*.

If *path* does not include a filename, the default filename is *<job ID>.ER*

If the -e option is not specified, PBS writes standard error to the default filename, which has this form:

<job name>.e<sequence number>

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

If you use a UNC path, the hostname is optional. If you use a non-UNC path, the hostname is required.

-h <hold list>

Updates the job's hold list. Adds *hold list* to the job's Hold_Types attribute. The *hold list* is a string of one or more characters. The following table shows the holds and the privilege required to set each:

Table 2-8: Hold Types

Hold Type	Meaning	Who Can Set
u	*User*	*Job owner, Operator, Manager, administrator, root*
o	*Other*	*Operator, Manager, administrator, root*
s	*System*	*Manager, administrator, root, PBS (dependency)*
n	*None*	*Job owner, Operator, Manager, administrator, root*
p	*Bad password*	*Administrator, root*

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

-j <join>

Changes whether and how to join the job's standard error and standard output streams. Sets job's Join_Path attribute to *join*.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

Default: *n*; not merged

The *join* argument can take the following values:

Table 2-9: Join Path Options

Value	Meaning
oe	Standard error and standard output are merged into standard output.
eo	Standard error and standard output are merged into standard error.
n	Standard error and standard output are not merged.

-k <discard>

Changes whether and which of the standard output and standard error streams is left behind on the execution host, and whether they are written to their final destinations. Sets the job's Keep_Files attribute to *discard*. Overrides default path names for these streams. Overrides -o and -e options.

This attribute cannot be altered once the job has begun execution.

In the case where output and/or error is retained on the execution host in a job-specific staging and execution directory created by PBS, these files are deleted when PBS deletes the directory.

Default: *n*; neither is retained, and files are not written to final destinations

The *discard* argument can take the following values:

Table 2-10: discard Argument Values

Option	Meaning
e	The standard error stream is retained on the execution host, in the job's staging and execution directory. The filename is <*job name*>.*e*<*sequence number*>
o	The standard output stream is retained on the execution host, in the job's staging and execution directory. The filename is <*job name*>.*o*<*sequence number*>
eo, oe	Both standard output and standard error streams are retained on the execution host, in the job's staging and execution directory.
d	Output and/or error are written directly to their final destination. Overrides the action of leaving files behind on execution host.
n	Neither stream is retained.

-l <resource list>

Allows the user to change requested resources and job placement. Sets job's Resource_list attribute to *resource list*. Uses resource request syntax. Requesting a resource places a limit on its usage. Users without manager or operator privilege cannot alter a custom resource which was created to be invisible or read-only for users. For syntax, see section 2.42.2.2.i, “Syntax for Modifying Resources and Job Placement”, on page 127.

If a requested modification to a resource would exceed the server's or the job queue's limits, the resource request is rejected. Which resources can be altered is system-dependent.

If the job was submitted with an explicit "`-l select=`", vnode-level resources must be qaltered using the "`-l select=`" form. In this case a vnode-level resource *resource* cannot be qaltered with the "`-l <resource name>`" form.

The place statement cannot begin with a colon.

Examples:

1. Submit the job:

   ```
   % qsub -l select=1:ncpus=2:mem=512mb jobscript
   ```

   Job's ID is 230

2. qalter the job using "`-l <resource name>`" form:

   ```
   % qalter -l ncpus=4 230
   ```

   Error reported by `qalter`:

   ```
   qalter: Resource must only appear in "select" specification when select is used: ncpus 230
   ```

3. qalter the job using the "`-l select=`" form:

   ```
   % qalter -l select=1:ncpus=4:mem=512mb 230
   ```

No error reported by `qalter`:

```
%
```

For more on resource requests, usage limits and job placement, see "Allocating Resources & Placing Jobs", on page 49 of the PBS Professional User's Guide.

-m <mail events>

Changes the set of conditions under which mail about the job is sent. Sets job's Mail_Points attribute to *mail events*. The *mail events* argument can be one of the following:

- The single character "*n*"
- Any combination of "*a*", "*b*", and "*e*", with optional "*j*"

The following table lists the sub-options to the `-m` option:

Table 2-11: Sub-options to m Option

Suboption	Meaning
n	No mail is sent.
a	Mail is sent when the job is aborted by PBS.
b	Mail is sent when the job begins execution.
e	Mail is sent when the job terminates.
j	Mail is sent for subjobs. Must be combined with one or more of a, b, or e options

Can be used with job arrays but not subjobs.

Format: *String*

Syntax: *n | [j](one or more of a, b, e)*

Example: -m ja

Default value: *"a"*

-M <user list>

Alters list of users to whom mail about the job is sent. Sets job's Mail_Users attribute to *user list*.

Format: *<username>[@<hostname>][,<username>[@<hostname>],...]*

Default: Job owner.

-N <name>

Renames the job. Sets job's Job_Name attribute to *name*.

Format: *Job Name*. See "Job Name, Job Array Name" on page 355.

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "*STDIN*".

-o <output path>

Alters path to be used for the job's standard output stream. Sets job's Output_Path attribute to *output path*. Overridden by `-k` option.

Format: *[<hostname>:]<path>*

The *output path* is interpreted as follows:

path

If *path* is relative, it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If *path* is absolute, it is taken to be an absolute path on the current host where the command is executing.

<hostname>:<path>

If *path* is relative, it is taken to be relative to the user's home directory on the host named *hostname*.

If *path* is absolute, it is the absolute path on the host named *hostname*.

If *path* does not include a filename, the default filename is:

<job ID>.OU

If the -o option is not specified, PBS writes standard output to the default filename, which has this form:

<job name>.o<sequence number>

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

If you use a UNC path, the hostname is optional. If you use a non-UNC path, the hostname is required.

-p <priority>

Alters priority of the job. Sets job's Priority attribute to *priority*.

This attribute can be altered after the job has begun execution, in which case the new value will not take effect until the job is rerun.

Format: *Host-dependent integer*

Range: [*-1024, +1023*] inclusive

Default: *zero*

-P <project>

Specifies a project for the job. Sets job's project attribute to specified value.

Format: *Project Name*; see "Project Name" on page 357

Default: "*_pbs_project_default*"

-r <y|n>

Changes whether the job is rerunnable. Sets job's Rerunable attribute to the argument. Does not affect how job is handled when the job is unable to begin execution.

See "qrerun" on page 178.

Format: Single character, "*y*" or "*n*".

y
: Job is rerunnable.

n
: Job is not rerunnable.

Default: "*y*".

Interactive jobs are not rerunnable. Job arrays are always rerunnable.

-R <remove options>

Changes whether standard output and/or standard error files are automatically removed upon job completion.

Sets the job's Remove_Files attribute to *remove options*. Overrides default path names for these streams. Overrides -o and -e options.

This attribute cannot be altered once the job has begun execution.

Default: unset; neither is removed

The *remove options* argument can take the following values:

Table 2-12: discard Argument Values

Option	Meaning
e	The standard error stream is removed (deleted) upon job completion
o	The standard output stream is removed (deleted) upon job completion
eo, oe	Both standard output and standard error streams are removed (deleted) upon job completion
unset	Neither stream is removed

-S <path list>

Specifies the interpreter or shell path for the job script. Sets job's Shell_Path_List attribute to *path list*.

The *path list* argument is the full path to the interpreter or shell including the executable name.

Only one path may be specified without a hostname. Only one path may be specified per named host. The path selected is the one whose hostname is that of the server on which the job resides.

This attribute can be altered after the job has begun execution, but in this case the new value will not take effect until the job is rerun.

Format:

<path>[@<hostname>][,<path>@<hostname> ...]

If the path contains spaces, it must be quoted. For example:

```
qsub -S "C:Program Files\PBS Pro\bin\pbs_python.exe" <script name>
```

Default: user's login shell on execution node

Example of using `bash` via a directive:

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

Example of running a Python script from the command line on Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

Example of running a Python script from the command line on Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

-u <user list>

Alters list of usernames. Job will run under a username from this list. Sets job's User_List attribute to *user list*.

Only one username may be specified without a hostname. Only one username may be specified per named host. The server on which the job resides will select first the username whose hostname is the same as the server name. Failing that, the next selection will be the username with no specified hostname. The usernames on the server and execution hosts must be the same. The job owner must have authorization to run as the specified user.

This attribute cannot be altered once the job has begun execution.

Format: *<username>[@<hostname>][,<username>@<hostname> ...]*

Default: Job owner (username on submit host)

-W <additional attributes>

Each sub-option to the -W option allows you to change a specific job attribute.

Format: *-W <attribute name> = <attribute value>[,<attribute name>=<attribute value>...]*

If white space occurs within the *additional attributes* argument, or the equal sign ("=") occurs within an *attribute value* string, that argument or string must be enclosed in single or double quotes. PBS supports setting the following attributes via the -W option:

depend=<dependency list>

Defines dependencies between this and other jobs. Sets the job's depend attribute to *dependency list*. The *dependency list* has the form:

<type>:<arg list>[,<type>:<arg list> ...]

where except for the *on* type, the *<arg list>* is one or more PBS job IDs in the form:

<job ID>[:<job ID> ...]

The types and their argument lists can be:

after: <arg list>

This job may be scheduled for execution at any point after all jobs in *arg list* have started execution.

afterok: <arg list>

This job may be scheduled for execution only after all jobs in *arg list* have terminated with no errors. See section 2.42.6.1, "Warning About Exit Status with csh", on page 139.

afternotok: <arg list>

This job may be scheduled for execution only after all jobs in *arg list* have terminated with errors. See section 2.42.6.1, "Warning About Exit Status with csh", on page 139.

afterany: <arg list>

This job may be scheduled for execution after all jobs in *arg list* have terminated, with or without errors. This job will not run if a job in the *arg list* was deleted without ever having been run.

before: <arg list>

Jobs in *arg list* may begin execution once this job has begun execution.

beforeok: <arg list>

Jobs in *arg list* may begin execution once this job terminates without errors. See section 2.42.6.1, "Warning About Exit Status with csh", on page 139.

beforenotok: <arg list>

If this job terminates execution with errors, jobs in *arg list* may begin. See section 2.42.6.1, "Warning About Exit Status with csh", on page 139.

beforeany: <arg list>

Jobs in *arg list* may begin execution once this job terminates execution, with or without errors.

on: <count>

This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This type is used in conjunction with one of the *before* types listed. *count* is an integer greater than *0*.

runone:<job ID>

(2020.1 Beta) Puts the current job and the job with *job ID* in a set of jobs out of which PBS will eventually run just one. To add a job to a set, specify the job ID of another job already in the set.

Restrictions:

Job IDs in the *arg list* of *before* types must have been submitted with a type of *on*.

To use the *before* types, the user must have the authority to alter the jobs in *arg list*. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly-submitted job depends is performed after the job is queued. If an error is detected, the new job is deleted by the server. Mail is sent to the job submitter stating the error.

Dependency examples:

```
qalter -W depend = afterok:123.host1.domain.com /tmp/script
qalter -W depend= before:234.host1.com:235.host1.com /tmp/script
```

group_list=<group list>

Alters list of group names. Job will run under a group name from this list. Sets job's group_List attribute to *group list*.

Only one group name may be specified without a hostname. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose hostname is the same as the server name. Failing that, the next selection is the group name with no specified hostname. The group names on the server and execution hosts must be the same.

Format: *<group>[@<hostname>][,<group>@<hostname> ...]*

Default: no default

release_nodes_on_stageout=<value>

When set to *True*, all of the job's vnodes not on the primary execution host are released when stageout begins.

Cannot be used with vnodes tied to Cray X* series systems.

When cgroups is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

The job's stageout attribute must be set for the release_nodes_on_stageout attribute to take effect.

Format: *Boolean*

Default: *False*

run_count=<count>

Sets the number of times the server thinks it has run the job. Sets the job's run_count attribute to *count*. Can be altered while job is running. Job is held when the value of this attribute goes over *20*.

Format: Integer greater than or equal to zero

sandbox=<sandbox spec>

Changes which directory PBS uses for the job's staging and execution. Sets job's sandbox attribute to the value of *sandbox spec*.

Format: *String*

Allowed values for *sandbox spec*:

PRIVATE

PBS creates a job-specific directory for staging and execution.

HOME or unset

PBS uses the user's home directory for staging and execution.

stagein=<path list>

stageout=<path list>

Changes files or directories to be staged in before execution or staged out after execution is complete. Sets the job's stagein and stageout attributes to the specified *path list*s. On completion of the job, all staged-in and staged-out files and directories are removed from the execution host(s). A *path list* has the form:

<filespec>[,<filespec>]

where *filespec* is

<execution path>@<hostname>:<storage path>

regardless of the direction of the copy. The *execution path* is the name of the file or directory on the primary execution host. It can be relative to the staging and execution directory on the execution host, or it can be an absolute path.

The "@" character separates *execution path* from *storage path*.

The *storage path* is the path on *hostname*. The name can be relative to the staging and execution directory on the primary execution host, or it can be an absolute path.

If *path list* has more than one *filespec*, i.e. it contains commas, it must be enclosed in double quotes.

If you use a UNC path, the hostname is optional. If you use a non-UNC path, the hostname is required.

umask=<mask value>

Alters the umask with which the job is started. Controls umask of job's standard output and standard error. Sets job's umask attribute to *mask value*.

Format: one to four digits; typically two

The following example allows group and world read of the job's output and error:

```
-W umask=33
```

Default: *077*

--version

The `qalter` command returns its PBS version information and exits. This option can only be used alone.

2.42.4 Operands

The `qalter` command accepts a *job ID* list as its operand. The *job ID* list is a space-separated list of one or more job IDs for normal jobs or array jobs.

Subjobs and ranges of subjobs are not alterable.

Job IDs have the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

Note that some shells require that you enclose a job array ID in double quotes.

2.42.5 Standard Error

The `qalter` command writes a diagnostic message to standard error for each error occurrence.

2.42.6 Exit Status

Zero

Upon successful processing of input

Greater than zero

Upon failure

2.42.6.1 Warning About Exit Status with csh

If a job is run in `csh` and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies.

2.42.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "Job Attributes" on page 328, Chapter 5, "List of Built-in Resources", on page 259

2.43 qdel

Deletes PBS jobs

2.43.1 Synopsis

qdel [-x] [-Wforce| -Wsuppress_email=<N>] <job ID> [<job ID> ...]

qdel --version

2.43.2 Description

The `qdel` command deletes jobs in the order given, whether they are at the local server or at a remote server.

2.43.2.1 Usage

The `qdel` command is used without options to delete queued, running, held, or suspended jobs, while the `-x` option gives it the additional capacity to delete finished or moved jobs. With the `-x` option, this command can be used on finished and moved jobs, in addition to queued, running, held, or suspended jobs.

When this command is used without the `-x` option, if job history is enabled, the deleted job's history is retained. The -x option is used to additionally remove the history of the job being deleted.

If someone other than the job's owner deletes the job, mail is sent to the job's owner, or to a list of mail recipients if specified during `qsub`. See "qsub" on page 213.

If the job is in the process of provisioning, it can be deleted only by using the `-W force` option.

2.43.2.2 How Behavior of `qdel` Command Can Be Affected

The server's default_qdel_arguments attribute may affect the behavior of the `qdel` command. This attribute is settable by the administrator via the `qmgr` command. The attribute may be set to "*-Wsuppress_email=<N>*". The server attribute is overridden by command-line arguments. See section 6.6, "Server Attributes", on page 281.

2.43.2.3 Sequence of Events

1. The job's running processes are killed.
2. The epilogue runs.
3. Files that were staged in are staged out. This includes standard out (.o) and standard error (.e) files.
4. Files that were staged in or out are deleted.
5. The job's temp directory is removed.
6. The job is removed from the MoM(s) and the server.

2.43.2.4 Required Privilege

A PBS job may be deleted by its owner, an Operator, or the administrator. The MoM deletes a PBS job by sending a SIGTERM signal, then, if there are remaining processes, a SIGKILL signal.

2.43.3 Options to `qdel`

(no options)

Can delete queued, running, held, or suspended jobs. Does not delete job history for specified job(s).

-W force

Deletes the job whether or not the job's execution host is reachable. Deletes the job whether or not the job is in the process of provisioning. Cannot be used with the `-Wsuppress_email` option.

If the server can contact the MoM, this option is ignored; the server allows the job to be deleted normally. If the server cannot contact the MoM or the job is in the *E* state, the server deletes its information about the job.

-Wsuppress_email=<N>

Sets limit on number of emails sent when deleting multiple jobs or subjobs.

- If *N* >= *1* and *N* or more *job IDs* are given, *N* emails are sent.
- If *N* >=*1* and less than *N* job identifiers are given, the number of emails is the same as the number of jobs.
- If *N* = *0*, this option is ignored.
- If *N* = *-1*, no mail is sent.

Note that there is no space between "`W`" and "`suppress_email`".

The *N* argument is an integer.

Cannot be used with `-Wforce` option.

-x

Can delete running, queued, suspended, held, finished, or moved jobs. Deletes job history for the specified job(s).

--version

The `qdel` command returns its PBS version information and exits. This option can only be used alone.

2.43.4 Operands

The `qdel` command accepts one or more space-separated *job ID* operands. These operands can be job identifiers, job array identifiers, subjob identifiers, or subjob range identifiers.

Job IDs have the form:

<sequence number>[.<server name>][@<server name>]

Job arrays have the form:

<sequence number>[][.<server name>][@<server name>]

Subjobs have the form:

<sequence number>[<index>][.<server name>][@<server name>]

Ranges of subjobs have the form:

<sequence number>[<first>-<last>][.<server name>][@<server name>]

Job array identifiers must be enclosed in double quotes for some shells.

2.43.5 Standard Error

The `qdel` command writes a diagnostic message to standard error for each error occurrence.

2.43.6 Exit Status

Zero
: Upon successful processing of input

Greater than zero
: Upon error

2.43.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide

2.44 qdisable

Prevents a queue from accepting jobs

2.44.1 Synopsis

qdisable <destination> [<destination> ...]

qdisable --version

2.44.2 Description

The `qdisable` command prevents a queue from accepting batch jobs. Sets the value of the queue's enabled attribute to *False*. If the command is accepted, the queue no longer accepts Queue Job requests. Jobs already in the queue continue to be processed. You can use this to drain a queue of jobs.

2.44.2.1 Required Permission

In order to execute `qdisable`, the user must have PBS Operator or Manager privilege.

2.44.3 Options

--version

The `qdisable` command returns its PBS version information and exits. This option can only be used alone.

2.44.4 Operands

The `qdisable` command accepts one or more space-separated *destination* operands. The operands take any of the following forms:

<queue name>

Prevents specified queue at default server from accepting jobs.

@<server name>

Prevents all queues at specified server from accepting jobs.

<queue name>@<server name>

Prevents specified queue at specified server from accepting jobs.

To prevent all queues at the default server from accepting jobs, use the `qmgr` command:

```
Qmgr: set queue @default enabled=false
```

2.44.5 Standard Error

The `qdisable` command writes a diagnostic message to standard error for each error occurrence.

2.44.6 Exit Status

Zero
: Upon successful processing of all the operands

Greater than zero
: If the `qdisable` command fails to process any operand

2.44.7 See Also

The PBS Professional Administrator's Guide, "qmgr" on page 149, "qenable" on page 145

2.45 qenable

Allows a queue to accept jobs

2.45.1 Synopsis

qenable <destination> [<destination> ...]

qenable --version

2.45.2 Description

The `qenable` command allows a queue to accept batch jobs. Sets the value of the queue's enabled attribute to *True*. If the command is accepted, the *destination* accepts Queue Job requests.

2.45.2.1 Required Privilege

In order to execute `qenable`, the user must have PBS Operator or Manager privilege.

2.45.3 Options

--version

The `qenable` command returns its PBS version information and exits. This option can only be used alone.

2.45.4 Operands

The `qenable` command accepts one or more space-separated *destination* operands. The operands take any of the following forms:

<queue name>

Allows specified queue at default server to accept jobs.

@<server name>

Allows all queues at specified server to accept jobs.

<queue name>@<server name>

Allows specified queue at specified server to accept jobs.

To allow all queues at the default server to accept jobs, use the `qmgr` command:

```
Qmgr: set queue @default enabled=true
```

2.45.5 Standard Error

The `qenable` command writes a diagnostic message to standard error for each error occurrence.

2.45.6 Exit Status

Zero

Upon successful processing of all the operands

Greater than zero

If the `qenable` command fails to process any operand

2.45.7 See Also

The PBS Professional Administrator's Guide, "qmgr" on page 149, "qdisable" on page 143

2.46 qhold

Holds PBS batch jobs

2.46.1 Synopsis

qhold [-h <hold list>] <job ID> [<job ID> ...]

qhold --version

2.46.2 Description

Places one or more holds on a job. A job that has a hold is not eligible for execution. Can be used on jobs and job arrays, but not on subjobs or ranges of subjobs.

If a job identified by *job ID* is in the queued, held, or waiting states, all that occurs is that the hold type is added to the job. The job is then put into the held state if it resides in an execution queue.

If the job is running, the result of the `qhold` command depends upon whether the job can be checkpointed. The job can be checkpointed if the OS supports checkpointing, or if the application being checkpointed supports checkpointing. See the PBS Professional Administrator's Guide. If the job can be checkpointed, the following happens:

- The job is checkpointed and its execution is interrupted.
- The resources assigned to the job are released.
- The job is placed in the held state in the execution queue.
- The job's Hold_Types attribute is set to *u* for *user hold*.

If checkpoint / restart is not supported, `qhold` simply sets the job's Hold_Types attribute to *u*. The job continues to execute.

A job's dependency places a *system* hold on the job. When the dependency is satisfied, the *system* hold is removed. If the administrator sets a *system* hold on a job with a dependency, when the dependency is satisfied, the job becomes eligible for execution.

If the job is in the process of provisioning, it cannot be held.

A hold on a job can be released by the PBS Administrator, root, a Manager, an Operator, or the job owner, when the job reaches the time set in its Execution_Time attribute, or when a dependency clears. See "qrls" on page 180.

2.46.2.1 Effect of Privilege on Behavior

The following table shows the holds and the privilege required to set each:

Table 2-13: Hold Types

Hold Type	Meaning	Who Can Set
u	*User*	*Job owner, Operator, Manager, PBS Administrator, root*
o	*Other*	*Operator, Manager, PBS Administrator, root*
s	*System*	*Manager, PBS Administrator, root, PBS (dependency)*
n	*No hold*	*Job owner, Operator, Manager, PBS Administrator, root*
p	*Bad password*	*PBS Administrator, root*

2.46.3 Options to `qhold`

(no options)

Same as `-h u`. Applies the *user* hold to the specified job(s).

-h <hold list>

Types of holds to be placed on the job(s).

The *hold list* argument is a string consisting of one or more of the letters "*u*", "*o*", or "*s*" in any combination, or one of the letters "*n*" or "*p*".

--version

The `qhold` command returns its PBS version information and exits. This option can only be used alone.

2.46.4 Operands

The `qhold` command can be used on jobs and job arrays, but not on subjobs or ranges of subjobs. The `qhold` command accepts one or more *job IDs* in the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.46.5 Standard Error

The `qhold` command writes a diagnostic message to standard error for each error occurrence.

2.46.6 Exit Status

Zero

Upon successful processing of all operands

Greater than zero

If the `qhold` command fails to process any operand

2.46.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "qrls" on page 180

2.47 qmgr

Administrator's command interface for managing PBS

2.47.1 Synopsis

At shell command line:

qmgr -c '<directive> [-a] [-e] [-n] [-z]'

qmgr -c 'help [<help option>]'

qmgr <return>

qmgr --version

In qmgr session:

<directive> [-a] [-e] [-n] [-z]

help <help option>

2.47.2 Description

The PBS manager command, `qmgr`, provides a command-line interface to parts of PBS. The `qmgr` command is used to create or delete queues, vnodes, resources, and hooks, to set or change vnode, queue, hook, server, or scheduler attributes and resources, and to view information about hooks, queues, vnodes, resource definitions, the server, and schedulers.

For a list of quick summaries of information about syntax, commands, attributes, operators, names, and values, type "`help`" or "`?`" at the `qmgr` prompt. See section 2.47.11, “Printing Usage Information”, on page 170.

2.47.2.1 Modes of Operation

- When you type `qmgr -c '<directive>'`, `qmgr` performs its task and then exits.
- When you type `qmgr <return>`, `qmgr` starts a session and presents you with its command line prompt. The `qmgr` command then reads directives etc. from standard input; see section 2.47.4.1, “Directive Syntax”, on page 151. You can edit the command line; see section 2.47.2.4, “Reusing and Editing the qmgr Command Line”, on page 150.

  For a `qmgr` prompt, type:

  ```
  qmgr <return>
  ```

  You will see the `qmgr` prompt:

  ```
  Qmgr:
  ```

2.47.2.2 Required Privilege

The `qmgr` command requires different levels of privilege depending on the operation to be performed.

All users can list or print attributes except for hook attributes.

PBS Operator or Manager privilege is required in order to set or change vnode, queue, server, or scheduler attributes. PBS Manager privilege is required in order to create or delete queues, vnodes, and resources.

Under Linux, root privilege is required in order to create hooks, or operate on hooks or the job_sort_formula server attribute. Under Windows, this must be done from the installation account.

For domained environments, the installation account must be a local account that is a member of the local Administrators group on the local computer.

Users without manager or operator privilege cannot view custom resources or resource definitions which were created to be invisible to users.

2.47.2.3 When To Run qmgr At Server Host

When operating on hooks or on the job_sort_formula server attribute, the qmgr command must be run at the server host.

2.47.2.4 Reusing and Editing the qmgr Command Line

You can reuse or edit qmgr command lines. The qmgr command maintains a history of commands entered, up to a maximum of 500. You can use the 'history' command to see a numbered list of commands, and the !<n> command to execute the line whose number is *n*. You must not put any spaces between the bang ("!") and the number. For example, to execute the 123rd command, type the following:

```
!123
```

You can see the last *m* commands by typing 'history m'. For example, to see the last 6 commands, type the following:

```
history 6
```

You can use the up and down arrows to navigate through the command history list, and the left and right arrows to navigate within a command line. Within a command line, you can use emacs commands to move forward and backward, and delete characters.

You can edit the qmgr command line using the backspace and delete keys, and you can insert characters anywhere in a command line.

History is maintained across qmgr sessions, so that if you start qmgr, then exit, then restart it, you can reuse your commands from the previous session. If you exit qmgr and then restart it, the command lines are renumbered.

If you enter the same command line more than once in a row, only one occurrence is recorded in the history. If you enter the same command line multiple times, but intersperse other command lines after each line, each occurrence is recorded.

Each user's history is unique to that user on that host.

In the case where an account runs concurrent sessions, the most recent logout of a session overwrites history from previous logouts. For example, if two people are both logged in as root and using qmgr, the second person to log out overwrites the history file.

2.47.2.4.i The qmgr History File

The qmgr command stores and retrieves its history. First, it tries to write its history in the ${HOME}/.pbs_qmgr_history file. If this file or directory location is not writable, the command stores its history in $PBS_HOME/spool/.pbs_qmgr_history_<user name>. If this file is also not writable, the following happens:

- The qmgr command prints error messages once at qmgr startup
- The qmgr command cannot provide history across qmgr sessions

2.47.3 Options to qmgr

The following table lists the options to qmgr:

Table 2-14: qmgr Options

Option	Action
<return>	Starts a qmgr session and presents user with qmgr prompt
-a	Aborts qmgr on any syntax errors or any requests rejected by a server.
-c '<directive>'	Executes a single command (*directive*) and exit qmgr. The *directive* must be enclosed in single or double quote marks, for example: qmgr -c "print server"
-c 'help [<help option>]'	Prints out usage information. See "Printing Usage Information" on page 170
-e	Echoes all commands to standard output
-n	No commands are executed; syntax checking only is performed
-z	No errors are written to standard error
--version	The qmgr command returns its PBS version information and exits. This option can only be used alone

2.47.4 Directives

A qmgr *directive* is a *command* together with the *object(s)* to be operated on, the *attribute(s)* belonging to the object that is to be changed, the *operator*, and the *value(s)* the *attribute(s)* will take. In the case of resources, you can set the *type* and/or *flag*(s).

2.47.4.1 Directive Syntax

A directive is terminated by a newline or a semicolon (";"). Multiple directives may be entered on a single line. A directive may extend across lines by escaping the newline with a backslash ("\").

Comments begin with the "#" character and continue to the end of the line. Comments and blank lines are ignored by qmgr.

2.47.4.1.i Server, Scheduler, Queue, Vnode Directives

Syntax for operating on servers, schedulers, queues, and vnodes:

<command> <object type> [<object name(s)>] [<attribute> <operator> <value>[,<attribute> <operator> <value>,...]]

For information about attributes, see Chapter 6, "Attributes", on page 277.

2.47.4.1.ii Resource Directives

Syntax for operating on resources:

<command> <resource name> [<resource name> ...] [type = <type>][,flag = <flag(s)>]

For information about resources, see "Using PBS Resources" on page 229 in the PBS Professional Administrator's Guide and Chapter 5, "List of Built-in Resources", on page 259.

2.47.4.1.iii Hook-only Directives

The directives here apply only to hooks. Other directives apply to all objects such as queues, resources, hooks, etc.

Syntax for importing and exporting site-defined hooks:

"*import hook <hook name> application/x-python <content-encoding> (<input file> | -)*"

"*export hook <hook name> <content-type> <content-encoding>*" > *[<output file>]*

Syntax for importing site-defined hook configuration file:

"*import hook <hook name> application/x-config <content-encoding> (<input file> | -)*"

Syntax for importing built-in hook configuration file:

"*import pbshook <hook name> application/x-config <content-encoding> (<input file> | -)*"

2.47.4.2 Using Directives

You can use a *directive* from the shell command line or from within the `qmgr` session.

- To use a directive from the command line, enclose the command and its arguments in single or double quotes.

  qmgr -c '<command> <command arguments>'

  For example, to have `qmgr` print server information and exit:

  ```
  qmgr -c "print server"
  ```

- To use a directive from within the `qmgr` session, first start `qmgr`:

  ```
  qmgr <return>
  ```

  The `qmgr` session presents a `qmgr` prompt:

  ```
  Qmgr:
  ```

  At the `qmgr` prompt, enter the directive (a command and its arguments). For example, to enter the same "print server" directive:

  ```
  Qmgr: print server
  ```

2.47.4.3 Commands Used in Directives

Commands can be abbreviated to their minimum unambiguous form. Commands apply to all target objects unless explicitly limited. The following table lists the commands, briefly tells what they do, and gives a link to a full description:

Table 2-15: `qmgr` Commands Used in Directives

Command	Abbr.	Effect	Description
`active`	`a`	Specifies active objects	See section 2.47.6.1, “Making Objects Active”, on page 156
`create`	`c`	Creates object	See section 2.47.6.2, “Creating Objects (Server, Scheduler, Vnode, Queue, Hook)”, on page 157
`delete`	`d`	Deletes object	See section 2.47.6.3, “Deleting Objects”, on page 157
`exit`		Exits (quits) the `qmgr` session	
`export`	`e`	Exports hook or hook configuration file	See section 2.47.10.6, “Exporting Hooks”, on page 169 and section 2.47.10.5.ii, “Exporting Configuration Files”, on page 168

Table 2-15: `qmgr` Commands Used in Directives

Command	Abbr.	Effect	Description
`help or ?`	`h, ?`	Prints usage to `stdout`	See section 2.47.11, "Printing Usage Information", on page 170
`import`	`i`	Imports hook or configuration file	See section 2.47.10.4, "Importing Hooks", on page 167 or section 2.47.10.5.i, "Importing Configuration Files", on page 168
`list`	`l`	Lists object attributes and their values	See section 2.47.8.1, "Listing Objects and Their Attributes", on page 164
`print`	`p`	Prints creation and configuration commands	See section 2.47.8.3, "Printing Creation and Configuration Commands", on page 166
`quit`	`q`	Quits (exits) the `qmgr` session	
`set`	`s`	Sets value of attribute	See section 2.47.7.1, "Setting Attribute and Resource Values", on page 158
`unset`	`u`	Unsets value of attribute	See section 2.47.7.2, "Unsetting Attribute and Resource Values", on page 159

2.47.5 Arguments to Directive Commands

2.47.5.1 Object Arguments to Directive Commands

The `qmgr` command can operate on objects (servers, schedulers, queues, vnodes, resources, hooks, and built-in hooks). Each of these can be abbreviated inside a directive. The following table lists the objects and their abbreviations:

Table 2-16: `qmgr` Objects

Object Name	Abbr.	Object	Can Be Created/Deleted By:	Can Be Modified By:
server	*s*	server	No one (created at installation)	Administrator, Operator, Manager
sched	*sc*	default scheduler	No one (created at installation)	Administrator, Operator, Manager
		multisched	Administrator, Manager	Administrator, Operator, Manager
queue	*q*	queue	Administrator, Operator, Manager	Administrator, Operator, Manager
node	*n*	vnode	Administrator, Operator, Manager	Administrator, Operator, Manager
resource	*r*	resource	Administrator, Manager	Administrator, Manager
hook	*h*	hook	Linux: root Windows: installation account	Linux: root Windows: installation account
pbshook	*p*	built-in hook	No one (created at installation)	Linux: root Windows: installation account

2.47.5.1.i Specifying Active Server

The `qmgr` command operates on objects (queues, vnodes, etc.) at the active server. There is always at least one active server; the default server is the active server unless other servers have been made active. The default server is the server managing the host where the `qmgr` command runs, meaning it is the server specified in that host's `pbs.conf` file. Server names have the following format:

<hostname>[:<port number>]

where *hostname* is the fully-qualified domain name of the host on which the server is running and *port number* is the port number to which to connect. If *port number* is not specified, the default port number, *15001*, is used.

- To specify the default server:

  @default
- To specify a named server:

  @<server name>
- To specify all active servers:

  @active

2.47.5.1.ii Using Lists of Object Names

In a `qmgr` directive, *object name(s)* is a list of one or more names of specific objects. The administrator specifies the name of an object when creating the object. The name list is in the form:

<object name>[@<server>][,<object name>[@<server>] ...]

where *server* is replaced in the directive with "*default*", "*active*", or the name of the server. The name list must conform to the following:

- There must be no space between the object name and the @ sign.
- Name lists must not contain white space between entries.
- All objects in a list must be of the same type.
- Node attributes cannot be used as vnode names.

2.47.5.1.iii Specifying Object Type and Name

You can specify objects in the following ways:

- To act on the active objects of the named type, at the active server:

  <object type>

  For example, to list all active vnodes, along with their attributes, at the active server:

  `Qmgr: list node`
- To act on the active objects of the named type, at a specified server:

  <object type> @<server name> (note space before @ sign)

  For example, to list all active vnodes at the default server, along with their attributes:

  `Qmgr: list node @default`

  For example, to print out all queues at the default server, along with their attributes:

  `qmgr -c "print queue @default"`
- To act on a specific named object:

  <object type> <object name>

  For example, to list Node1 and its attributes:

  `Qmgr: list node Node1`

To list queues workq, slowq, and fastq at the active server:

```
Qmgr: list queue workq,slowq,fastq
```

- To act on the named object at the specified server:

  <object type> <object name>@<server name>

  For example, to list Node1 at the default server, along with the attributes of Node1:

  ```
  Qmgr: list node Node1@default
  ```

  To list queues Queue1 at the default server, Queue2 at Server2, and Queue3 at the active server:

  ```
  Qmgr: list queue Queue1@default,Queue2@Server2,Queue3@active
  ```

2.47.5.2 Operators in Directive Commands

In a `qmgr` directive, *operator* is the operation to be performed with the attribute and its value. Operators are listed here:

Table 2-17: Operators in Directive Commands

Operator	Effect
=	Sets the value of the attribute or resource. If the attribute or resource has an existing value, the current value is replaced with the new value.
+=	Increases the current value of the attribute or resource by the amount in the new value. When used for a string array, adds the new value as another string after a comma.
-=	Decreases the current value of the attribute or resource by the specified amount. When used for a string array, removes the first matching string.

Example 2-4: Set routing destination for queue Queue1 to be Dest1:

```
Qmgr: set queue route_destinations = Dest1
```

Example 2-5: Add new routing destination for queue Queue1:

```
Qmgr: set queue route_destinations += Dest2
```

Example 2-6: Remove new routing destination for queue Queue1:

```
Qmgr: set queue route_destinations -= Dest2
```

When setting numerical resource values, you can use only the equal sign ("=").

2.47.5.3 Windows Requirements For Directive Arguments

Under Windows, use double quotes when specifying arguments to `qmgr`. For example:

```
Qmgr: import hook hook1 application/x-python default "\Documents and
   Settings\pbsuser1\hook1.py"
```

or

```
qmgr -c 'import hook hook1 application/x-python default "\Documents and Set-
   tings\pbsuser1\hook1.py"'
```

2.47.6 Operating on Objects (Server, Scheduler, Vnode, Queue, Hook)

2.47.6.1 Making Objects Active

Making objects active is a way to set up a list of objects, all of the same type, on which you can then use a single command. For example, if you are going to set the same attribute to the same value on several vnodes, you can make all of the target vnodes active before using a single command to set the attribute value, instead of having to give the command once for each vnode. You can make any type of object active except for resources or hooks.

When an object is active, it is acted upon when you specify its type but do not specify names. When you specify any object names in a directive, active objects are not operated on unless they are named in the directive.

You can specify a list of active objects for each type of object. You can have active objects of multiple types at the same time. The active objects of one type have no effect on whether objects of another type are active.

Objects are active only until the `qmgr` command is exited, so this feature can be used only at the `qmgr` prompt.

Each time you make any objects active at a given server, that list of objects replaces any active objects of the same kind at that server. For example, if you have four queues at a particular server, and you make Q1 and Q2 active, then later make Q3 and Q4 active, the result is that Q3 and Q4 are the only active queues.

You can make different objects be active at different servers simultaneously. For example, you can set vnodes N1 and N2 at the default server, and vnodes N3 and N4 at server Server2 to be active at the same time.

To make all objects inactive, quit `qmgr`. When you quit `qmgr`, any object that was active is no longer active.

2.47.6.1.i Using the `active` Command

- To make the named object(s) of the specified type active:

  active <object type> [<object name>[,<object name> ...]]

  Example: To make queue Queue1 active:

  ```
  Qmgr: active queue Queue1
  ```

  Example: To make queues Queue1 and Queue2 at the active server be active, then enable them:

  ```
  Qmgr: active queue Queue1,Queue2
  Qmgr: set queue enabled=True
  ```

  Example: To make queue Queue1 at the default server and queue Queue2 at Server2 be active:

  ```
  Qmgr: active queue Queue1@default,Queue2@Server2
  ```

  Example: To make vnodes N1, N2, N3, and N4 active, and then give them all the same value for their max_running attribute:

  ```
  Qmgr: active node N1,N2,N3,N4
  Qmgr: set node max_running = 2
  ```

- To make all object(s) of the specified type at the specified server active:

  active <object type> @<server name> (note space before @ sign)

  Example: To make all queues at the default server active:

  ```
  Qmgr: active queue @default
  ```

  Example: To make all vnodes at server Server2 active:

  ```
  Qmgr: active node @Server2
  ```

- To report which objects of the specified type are active:

  active <object type>

  The `qmgr` command prints a list of names of active objects of the specified type to `stdout`.

2.47.6.2 Creating Objects (Server, Scheduler, Vnode, Queue, Hook)

- To create one new object of the specified type for each name, and give it the specified name:

  create <object type> <object name>[,<object name> ...] [[<attribute> = <value>] [,<attribute> = <value>] ...]

  Can be used only with multischeds, queues, vnodes, resources, and hooks. Cannot be used with built-in hooks.

  For example, to create a multisched named multisched_1 at the active server:

  Qmgr: create sched multisched_1

  For example, to create a queue named Q1 at the active server:

  Qmgr: create queue Q1

  For example, to create a vnode named N1 and a vnode named N2:

  Qmgr: create node N1,N2

  For example, to create queue Queue1 at the default server and queue Queue2 at Server2:

  Qmgr: create queue Queue1@default,Queue2@Server2

  For example, to create vnodes named N1, N2, N3, and N4 at the active server, and to set their Mom attribute to *Host1* and their max_running attribute to *1*:

  Qmgr: create node N1,N2,N3,N4 Mom=Host1, max_running = 1

  To create a host-level consumable string resource named "foo" that can be read and set by execution hooks:

  Qmgr: qmgr -c "create resource foo type=string,flag=mnh"

All objects of the same type at a server must have unique names. For example, each queue at server Server1 must have a unique name. Objects at one server can have the same name as objects at another server.

You can create multiple objects of the same type with a single command. You cannot create multiple types of objects in a single command.

To create multiple resources of the same type and flag, separate each resource name with a comma:

qmgr -c "create resource <resource>[,<resource> ...] type=<type>,flag=<flag(s)>"

2.47.6.2.i Examples of Creating Objects

Example 2-7: Create queue:

create queue fast priority=10,queue_type=e,enabled = true,max_running=0

Example 2-8: Create queue, set resources:

```
create queue little
set queue little resources_max.mem=8mw,resources_max.cput=10
```

2.47.6.3 Deleting Objects

- To delete the named object(s):

  delete <object type> <object name>[,<object name> ...]

  When you delete more than one object, do not put a space after a comma.

  Can be used only with queues, vnodes, resources, and hooks. Cannot be used with built-in hooks.

  For example, to delete queue Q1 at the active server:

  Qmgr: delete queue Q1

  For example, to delete vnodes N1 and N2 at the active server:

  Qmgr: delete node N1,N2

For example, to delete queue Queue1 at the default server and queue Queue2 at Server2:

```
Qmgr: delete queue Queue1@default,Queue2@Server2
```

For example, to delete resource "foo" at the active server:

```
Qmgr: delete resource foo
```

- To delete the active objects of the specified type:

  delete <object type>

  For example, to delete the active queues:

  ```
  Qmgr: delete queue
  ```

- To delete the active objects of the specified type at the specified server:

  delete <object type> @<server name>

  For example, to delete the active queues at server Server2:

  ```
  Qmgr: delete queue @Server2
  ```

You can delete multiple objects of the same type with a single command. You cannot delete multiple types of objects in a single command. To delete multiple resources, separate the resource names with commas.

For example:

```
Qmgr: delete resource r1,r2
```

You cannot delete a resource that is requested by a job or reservation, or that is set on a server, queue, or vnode.

2.47.7 Operating on Attributes and Resources

You can specify attributes and resources for named objects or for all objects of a type.

2.47.7.1 Setting Attribute and Resource Values

- To set the value of the specified attribute(s) for the named object(s):

  set <object type> <object name>[,<object name> ...] <attribute> = <value> [,<attribute> = <value> ...]

  Each specified attribute is set for each named object, so if you specify three attributes and two objects, both objects get all three attributes set.

- To set the attribute value for all active objects when there are active objects of the type specified:

  set <object type> <attribute> = <value>

- To set the attribute value for all active objects at the specified server when there are active objects of the type specified:

  set <object type> @<server name> <attribute> = <value>

  For example, to set the amount of memory on a vnode:

  ```
  Qmgr: set node Vnode1 resources_available.mem = 2mb
  ```

If the attribute is one which describes a set of resources such as resources_available, resources_default, resources_max, resources_used, etc., the attribute is specified in the form:

<attribute name>.<resource name>

You can have spaces between *attribute=value* pairs.

2.47.7.1.i Examples of Setting Attribute Values

Example 2-9: Increase limit on queue:

```
set queue fast max_running +=2
```

Example 2-10: Set software resource on mynode:

```
set node mynode resources_available.software = "myapp=/tmp/foo"
```

Example 2-11: Set limit on queue:

```
set queue max_running = 10
```

Example 2-12: Set vnode offline:

```
set node state = "offline"
```

2.47.7.2 Unsetting Attribute and Resource Values

You can use the qmgr command to unset attributes of any object, except for the type attribute of a built-in hook.

- To unset the value of the specified attributes of the named object(s):
  unset <object type> <object name>[,<object name> ...] <attribute>[,<attribute>...]
- To unset the value of specified attributes of active objects:
  unset <object type> <attribute>[,<attribute>...]
- To unset the value of specified attributes of the named object:
  unset <object type> <object name> <attribute>[,<attribute>...]
- To unset the value of specified attributes of the named object:
  unset <object type> @<server name> <attribute>[,<attribute>...]

2.47.7.2.i Example of Unsetting Attribute Value

Example 2-13: Unset limit on queue

```
unset queue fast max_running
```

2.47.7.3 Caveats and Restrictions for Setting Attribute and Resource Values

- If the value includes whitespace, commas or other special characters, such as the # character, the value string must be enclosed in single or double quotes. For example:
  ```
  Qmgr: set node Vnode1 comment="Node will be taken offline Friday at 1:00 for memory
     upgrade."
  ```
- You can set or unset attribute values for only one type of object in each command.
- You can use the qmgr command to set attributes of any object, except for the type attribute of a built-in hook.
- You can have spaces between attribute names.
- Attribute and resource values must conform to the format for the attribute or resource type. Each attribute's type is listed in Chapter 6, "Attributes", on page 277. Each format is described in Chapter 7, "Formats", on page 353.
- Most of a vnode's attributes may be set using qmgr. However, some **must** be set on the individual execution host in Version 2 vnode configuration files, NOT by using qmgr. See "Configuring Vnodes" on page 41 in the PBS Professional Administrator's Guide.

2.47.7.4 Setting Custom Resource Type and Flag(s)

You can use the qmgr command to set or unset the type and flag(s) for custom resources.

Resource types can be the following; see section 7.2, “Resource Formats”, on page 359:

string

boolean

string_array

long

size

float

- To set a custom resource type:

  set resource <resource name> type = <type>

  Sets the type of the named resource to the specified *type*. For example:

  ```
  Qmgr: qmgr -c "set resource foo type=string_array"
  ```

2.47.7.4.i Resource Accumulation Flags

When you define a custom resource, you can specify whether it is server-level or host-level, and whether it is consumable or not. This is done by setting resource accumulation flags via `qmgr`. A consumable resource is tracked, or accumulated, in the server, queue or vnode resources_assigned attribute. The resource accumulation flags determine where the value of resources_assigned.<resource name> is incremented.

2.47.7.4.ii Allowable Values for Resource Accumulation Flags

The value of <*resource flags*>, which is the resource accumulation flag for a resource can be one of the following:

Table 2-18: Resource Accumulation Flags

Flag	Meaning
(no flags)	Indicates a queue-level or server-level resource that is not consumable.
fh	The amount is consumable at the host level for only the first vnode allocated to the job (vnode with first task.) Must be consumable or time-based. Cannot be used with Boolean or string resources. . This flag specifies that the resource is accumulated at the first vnode, meaning that the value of resources_assigned.<resource> is incremented only at the first vnode when a job is allocated this resource or when a reservation requesting this resource on this vnode starts.

Table 2-18: Resource Accumulation Flags

Flag	Meaning
h	Indicates a host-level resource. Used alone, means that the resource is not consumable. Required for any resource that will be used inside a select statement. This flag selects hardware. This flag indicates that the resource must be requested inside of a select statement. Example: for a Boolean resource named "green": `Qmgr: create resource green type=boolean, flag=h`
nh	The amount is consumable at the host level, for all vnodes assigned to the job. Must be consumable or time-based. Cannot be used with Boolean or string resources. This flag specifies that the resource is accumulated at the vnode level, meaning that the value of resources_assigned.<resource> is incremented at relevant vnodes when a job is allocated this resource or when a reservation requesting this resource on this vnode starts. This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.
q	The amount is consumable at the queue and server level. When a job is assigned one unit of a resource with this flag, the resources_assigned.<resource> attribute at the server and any queue is incremented by one. Must be consumable or time-based. This flag specifies that the resource is accumulated at the queue and server level, meaning that the value of resources_assigned.<resource> is incremented at each queue and at the server when a job is allocated this resource. When a reservation starts, allocated resources are added to the server's resources_assigned attribute. This flag is not used with dynamic consumable resources. The scheduler will not oversubscribe dynamic consumable resources.

2.47.7.4.iii When to Use Accumulation Flags

The following table shows when to use accumulation flags.

Table 2-19: When to Use Accumulation Flags

Resource Category	Server	Queue	Host
Static, consumable	flag = q	flag = q	flag = nh or fh
Static, not consumable	flag = (none of h, n, q or f)	flag = (none of h, n, q or f)	flag = h
Dynamic	`server_dyn_res` line in `sched_config`, flag = (none of h, n, q or f)	(cannot be used)	MoM config and `mom_resources` line (**deprecated** as of 18.2.1) in `sched_config`, flag = h

2.47.7.4.iv Example of Resource Accumulation Flags

When defining a static consumable host-level resource, such as a node-locked application license, you would use the "n" and "h" flags.

When defining a dynamic resource such as a floating license, you would use no flags.

2.47.7.4.v Resource Accumulation Flag Restrictions and Caveats

- Numeric dynamic resources cannot have the q or n flags set. This would cause these resources to be under-used. These resources are tracked automatically by the scheduler.

2.47.7.4.vi Resource Permission Flags

When you define a custom resource, you can specify whether unprivileged users have permission to view or request the resource, and whether users can `qalter` a request for that resource. This is done by setting a resource permission flag via `qmgr`.

2.47.7.4.vii Allowable Values for Resource Permission Flags

The permission flag for a resource can be one of the following:

Table 2-20: Resource Permission Flags

Flag	Meaning
(no flag)	Users can view and request the resource, and `qalter` a resource request for this resource.
i	"Invisible". Users cannot view or request the resource. Users cannot `qalter` a resource request for this resource.
r	"Read only". Users can view the resource, but cannot request it or `qalter` a resource request for this resource.

2.47.7.4.viii Effect of Resource Permission Flags

- PBS Operators and Managers can view and request a resource, and `qalter` a resource request for that resource, regardless of the i and r flags.
- Users, operators and managers cannot submit a job which requests a restricted resource. Any job requesting a restricted resource will be rejected. If a manager needs to run a job which has a restricted resource with a different value from the default value, the manager must submit the job without requesting the resource, then `qalter` the resource value.
- While users cannot request these resources, their jobs can inherit default resources from resources_default.<resource name> and default_chunk.<resource name>.

  If a user tries to request a resource or modify a resource request which has a resource permission flag, they will get an error message from the command and the request will be rejected. For example, if they try to `qalter` a job's resource request, they will see an error message similar to the following:

  ```
  "qalter: Cannot set attribute, read only or insufficient permission  Resource_List.hps 173.mars"
  ```

2.47.7.4.ix Resource Permission Flag Restrictions and Caveats

- You can specify only one of the i or r flags per resource. If both are specified, the resource is treated as if only the i flag were specified, and an error message is logged at the default log level and printed to standard error.
- Resources assigned from the default_qsub_arguments server attribute are treated as if the user requested them. A job will be rejected if it requests a resource that has a resource permission flag whether that resource was requested by the user or came from default_qsub_arguments.
- The behavior of several command-line interfaces is dependent on resource permission flags. These interfaces are those which view or request resources or modify resource requests:

  `pbsnodes`

  Users cannot view restricted host-level custom resources.

pbs_rstat

Users cannot view restricted reservation resources.

pbs_rsub

Users cannot request restricted custom resources for reservations.

qalter

Users cannot alter a restricted resource.

qmgr

Users cannot print or list a restricted resource.

qselect

Users cannot specify restricted resources via -l Resource_List.

qsub

Users cannot request a restricted resource.

qstat

Users cannot view a restricted resource.

2.47.7.4.x Allowing Execution Hooks to Read Custom Job Resources Faster

You can make it faster for execution hooks to read custom job resources. Execution hooks cannot read custom job resources via the event, only via the server. However, you can cache a copy of a custom job resource at the MoMs for faster local reading by execution hooks, by setting the *m* flag for the resource. The job resources that can be cached are found in the following job attributes:

exec_vnode
Resource_List
resources_used

To create a resource with the *m* flag set, include the flag. For example, to create two host-level consumable resources r1 and r2 of type long that will be cached at MoMs:

```
qmgr -c "create resource r1,r2 type=long,flag=mnh"
```

To unset this flag for r1:

```
qmgr -c "set resource r1 flag=nh"
```

You can combine this flag with any other resource flag. Job resources created in an exechost_startup hook have the m flag set automatically.

2.47.7.4.xi Caveats for Caching Custom Job Resources

Large numbers of job resources that are cached at MoMs can slow things down. If you don't need execution hooks to be able to read a custom job resource often, don't cache the resource at the MoMs.

2.47.7.4.xii Setting Types and Flags for Custom Resources via qmgr

To set the type for a resource:

set resource <resource name> type = <type>

For example:

```
qmgr -c "set resource foo type=string_array"
```

To set the flags for a resource:

set resource <resource name> flag=<flag(s)>

For example:

```
qmgr -c "set resource foo flag=nh"
```

To set the type and flags for a resource:

set resource <resource name> type=<type>, flag=<flag(s)>

For example:

```
qmgr -c "set resource foo type=long,flag=nhi"
```

You can set multiple resources by separating the names with commas. For example:

```
qmgr -c "set resource r1, r2 type=long"
```

You cannot set the *nh*, *fh*, or *q* flag for a resource of type string, string_array, or Boolean.

You cannot set both the *n* and the *f* flags on one resource.

You cannot have the *n* or *f* flags without the *h* flag.

You cannot set both the *i* and *r* flags on one resource.

You cannot unset the type for a resource.

You cannot set the type for a resource that is requested by a current or history job or reservation, or set on a server, queue, or vnode.

You cannot set the flag(s) to *h*, *nh*, *fh*, or *q* for a resource that is currently requested by a current or history job or reservation.

You cannot unset the flag(s) for a resource that is currently requested by a current or history job or a reservation, or set on any server, queue, or vnode.

You cannot alter a built-in resource.

You can unset custom resource flags, but not their type.

2.47.7.4.xiii

2.47.8 Viewing Object, Attribute, and Resource Information

2.47.8.1 Listing Objects and Their Attributes

You can use the `qmgr` command to list attributes of any object, including attributes at their default values.

- To list the attributes, with associated values, of the named object(s):

  list <object type> <object name>[,<object name> ...]
- To list values of the specified attributes of the named object:

  list <object type> <object name> <attribute name>[, <attribute name>]...
- To list attributes, with associated values, of active objects of the specified type at the active server:

  list <object type>
- To list all objects of the specified type at the specified server, with their attributes and the values associated with the attributes:

  list <object type> @<server name>
- To list attributes of the active server:

  list server

  If no server other than the default server has been made active, lists attributes of the default server (it is the active server).
- To list attributes of the specified server:

list server <server name>

- To list attributes of all schedulers:

  list sched

- To list attributes of the specified scheduler:

  list sched <scheduler name>

- To list all hooks, along with their attributes:

  list hook

- To list attributes of the specified hook:

  list hook <hook name>

2.47.8.1.i Examples of Listing Objects and Their Attributes

Example 2-14: List serverA's schedulers' attributes:

```
list sched @serverA
```

Example 2-15: List attributes for default server's scheduler(s):

```
l sched @default
```

Example 2-16: List PBS version for default server's scheduler(s):

```
l sched @default pbs_version
```

Example 2-17: List queues at a specified server:

```
list queue @server1
```

2.47.8.2 Listing Resource Definitions

You can use the `qmgr list` and `print` commands to list resource definitions showing resource name, type, and flag(s).

- To list the name, type, and flag(s) of the named resource(s):

  list resource <resource name>[,<resource name> ...]

  or

  print resource <resource name>[,<resource name> ...]

- To list name, type, and flag(s) of custom resources only:

  list resource

  or

  print resource

  or

  print server (note that this also prints information for the active server)

- To list all custom resources at the specified server, with their names, types, and flags:

  list resource @<server name>

  or

  print resource @<server name>

When used by a non-privileged user, `qmgr` prints only resource definitions for resources that are visible to non-privileged users (those that do not have the *i* flag set).

2.47.8.3 Printing Creation and Configuration Commands

For printing the creation commands for any object except for a built-in hook.

- To print out the commands to create the named object(s) and set their attributes to their current values:

  print <object type> <object name>[,<object name> ...]

  where *object name* follows the name rules in section 2.47.5.1.ii, “Using Lists of Object Names”, on page 154.
- To print out the commands to create the named object and set its attributes to their current values:

  print <object type> <object name> [<attribute name>[, <attribute name>]...]

  where *object name* follows the name rules in section 2.47.5.1.ii, “Using Lists of Object Names”, on page 154.
- To print out the commands to create and configure the active objects of the named type:

  print <object type>
- To print out the commands to create and configure all of the objects of the specified type at the specified server:

  print <object type> @<server name>
- To print out the commands to create each queue, set the attributes of each queue to their current values, and set the attributes of the server to their current values:

  print server

  This is used for the server and queues, but not hooks.

  Prints information for the active server. If there is no active server, prints information for the default server.
- To print out the creation commands for all schedulers:

  print sched
- To print out the creation commands for the specified scheduler:

  print sched <scheduler name>

2.47.8.4 Caveats for Viewing Information

Some attributes whose values are unset do not appear in the output of the `qmgr` command.

Definitions for built-in resources do not appear in the output of the `qmgr` command.

When a non-privileged user prints resource definitions, `qmgr` prints only resource definitions for resources that are visible to non-privileged users (those that do not have the *i* flag set).

2.47.9 Saving and Re-creating Server and Queue Information

To save and recreate server and queue configuration, print the configuration information to a file, then read it back in later. For example, to save your configuration:

```
# qmgr -c "print server" > savedsettings
```

or

```
Qmgr: print server > savedsettings
```

When re-creating queue and server configuration, read the commands back into `qmgr`. For example:

```
qmgr < savedsettings
```

2.47.10 Operating on Hooks

2.47.10.1 Creating Hooks

- To create a hook:

  Qmgr: create hook <hook name>

  For example:

  Qmgr: create hook my_hook

2.47.10.2 Deleting Hooks

- To delete a hook:

  Qmgr: delete hook <hook name>

  For example:

  Qmgr: delete hook my_hook

2.47.10.3 Setting and Unsetting Hook Attributes

- To set a hook attribute:

  Qmgr: set hook <hook name> <attribute> = <value>
- To unset a hook attribute:

  Qmgr: unset hook <hook name> <attribute>

Example 2-18: Unset hook1's alarm attribute, causing hook1's alarm to revert to its default value of 30 seconds:

Qmgr: unset hook hook1 alarm

2.47.10.4 Importing Hooks

For importing the contents of a site-defined hook. Cannot be used with built-in hooks.

To import a hook, you import the contents of a hook script into the hook. You must specify a filename that is locally accessible to `qmgr` and the PBS server.

Format for importing a site-defined hook:

import hook <hook name> application/x-python <content encoding> {<input file> | -}

This uses the contents of *input file* or `stdin` (-) as the contents of hook *hook name*.

- The *input file* or `stdin` (-) data must have a format of *content type* and must be encoded with *content encoding*.
- The allowed values for *content encoding* are "*default*" (7bit) and "*base64*".
- If the source of input is `stdin` (-) and *content encoding* is "*default*", `qmgr` expects the input data to be terminated by *EOF*.
- If the source of input is `stdin` (-) and *content encoding* is "base64", `qmgr` expects input data to be terminated by a blank line.
- *input file* must be locally accessible to both `qmgr` and the requested batch server.
- A relative path *input file* is relative to the directory where `qmgr` was executed.
- If a hook already has a content script, that is overwritten by this import call.
- If the name in *input file* contains spaces as are used in Windows filenames, *input file* must be quoted.

There is no restriction on the size of the hook script.

2.47.10.4.i Examples of Importing Hooks

Example 2-19: Given a Python script in ASCII text file "hello.py", use its contents as the script contents of hook1:

```
#cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
# qmgr -c 'import hook hook1 application/x-python default hello.py'
```

Example 2-20: Given a base64-encoded file "hello.py.b64", qmgr unencodes the file's contents, and then makes this the script contents of hook1:

```
# cat hello.py.b64
cHJpbnQgImhlbGxvLCB3b3JsZCIK
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

Example 2-21: To create a provisioning hook called Provision_Hook, and import the ASCII hook script called "master_provision.py" located in /root/data/:

```
Qmgr: create hook Provision_Hook
Qmgr: import hook Provision_Hook application/x-python default
   /root/data/master_provision.py
```

2.47.10.5 Importing and Exporting Hook Configuration Files

2.47.10.5.i Importing Configuration Files

For importing the contents of a site-defined or built-in hook configuration file. To import a hook configuration file, you import the contents of a file to a hook. You must specify a filename that is locally accessible to qmgr and the PBS server.

Format for importing a site-defined hook configuration file:

import hook <hook name> application/x-config <content encoding> {<config file>|-}

Format for importing a built-in hook configuration file:

import pbshook <hook name> application/x-config <content encoding> {<config file>|-}

This uses the contents of *config file* or stdin (-) as the contents of the configuration file for hook *hook name*.

- The *config file* or stdin (-) data must have a format of *content-type* and must be encoded with *content encoding*.
- The allowed values for *content encoding* are "*default*" (7bit) and "*base64*".
- If the source of input is stdin (-) and *content encoding* is "*default*", qmgr expects the input data to be terminated by *EOF*.
- If the source of input is stdin (-) and *content encoding* is "base64", qmgr expects input data to be terminated by a blank line.
- *config file* must be locally accessible to both qmgr and the requested batch server.
- A relative path *config file* is relative to the directory where qmgr was executed.
- If a hook already has a configuration file, that file is overwritten by this import call.
- If the name in *config file* contains spaces as are used in Windows filenames, *input file* must be quoted.

There is no restriction on the size of the hook configuration file.

2.47.10.5.ii Exporting Configuration Files

Format for exporting a site-defined hook configuration file:

qmgr -c "export hook <hook name> application/x-config default" > {<config file>|-}

Format for exporting a built-in hook configuration file:

qmgr -c "export pbshook <hook name> application/x-config default" > {<config file>|-}

2.47.10.5.iii Hook Configuration File Format

PBS supports several file formats for configuration files. The format of the file is specified in its suffix. Formats can be any of the following:

- .ini
- .json
- .py (Python)
- .txt (generic, no special format)
- .xml
- No suffix: treat the input file as if it is a .txt file
- The dash (-) symbol: configuration file content is taken from STDIN. The content is treated as if it is a .txt file.

Example 2-22: To import a configuration file in .json format:

```
# qmgr -c "import hook my_hook application/x-config default my_input_file.json"
```

2.47.10.6 Exporting Hooks

For exporting the contents of a site-defined hook. Cannot be used with built-in hooks.

Format for exporting a hook:

qmgr -c "export hook <hook name> <content type> <content encoding>" > [<output file>]

This dumps the script contents of hook *hook name* into *output file*, or stdout if *output file* is not specified.

- The resulting *output file* or stdout data is of *content type* and *content encoding*.
- The only *content type* currently supported is "*application/x-python*".
- The allowed values for *content encoding* are "*default*" (7bit) and "*base64*".
- *output file* must be a path that can be created by qmgr.
- Any relative path *output file* is relative to the directory where qmgr was executed.
- If *output file* already exists it is overwritten. If PBS is unable to overwrite the file due to ownership or permission problems, an error message is displayed in stderr.
- If the *output file* name contains spaces like the ones used in Windows file names, *output file* must be enclosed in quotes.

2.47.10.6.i Examples of Exporting Hooks

Example 2-23: Dump hook1's script contents directly into a file "hello.py.out":

```
# qmgr -c "export hook hook1 application/x-python default" > hello.py
# cat hello.py
import pbs
pbs.event().job.comment="Hello, world"
```

Example 2-24: To< dump the script contents of a hook 'hook1' into a file in "\My Hooks\hook1.py":

```
qmgr -c "export hook hook1 application/x-python default" > "\My Hooks\hook1.py"
```

2.47.10.7 Printing Hook Information

- To print out the commands to create and configure all hooks, including their configuration files:

print hook

- To print out the commands to create and configure the specified hook, including its configuration file:

  print hook <hook name>

2.47.10.8 Saving and Re-creating Hook Information

You can save creation and configuration information for all hooks. For example:

```
# qmgr -c "print hook" > hook.qmgr
```

You can re-create all hooks and their configuration files. For example:

```
# qmgr < hook.qmgr
```

2.47.10.9 Restrictions on Built-in Hooks

You cannot do the following with built-in hooks:

- Import a built-in hook
- Export a built-in hook
- Print creation commands for a built-in hook
- Create a built-in hook
- Delete a built-in hook
- Set the type attribute for a built-in hook

2.47.11 Printing Usage Information

You use the `help` command or a question mark ("?") to invoke the `qmgr` built-in help function. You can request usage information for any of the `qmgr` commands, and for topics including attributes, operators, names, and values.

- To print out usage information for the specified command or topic:

  Qmgr: help [<command or topic>]

  or

  Qmgr: ? [<command or topic>]

  For example, to print usage information for the `set` command:

```
qmgr
Qmgr: help set
Syntax: set object [name][,name...] attribute[.resource] OP value
```

2.47.12 Standard Input

When you start a `qmgr` session, the `qmgr` command reads standard input for directives until it reaches end-of-file, or it reads the *exit* or *quit* command.

2.47.13 Standard Output

When you start a `qmgr` session, and standard output is connected to a terminal, `qmgr` writes a command prompt to standard output.

If you specify the `-e` option, `qmgr` echoes the directives it reads from standard input to standard output.

2.47.14 Standard Error

If you do not specify the `-z` option, the `qmgr` command writes a diagnostic message to standard error for each error occurrence.

2.47.15 Exit Status

0
: Success

1
: Error in parsing

2
: Error in execution

3
: Error connecting to server

4
: Error making object active

5
: Memory allocation error

2.47.16 See Also

The PBS Professional Administrator's Guide, Chapter 6, "Attributes", on page 277, Chapter 5, "List of Built-in Resources", on page 259

2.48 qmove

Moves a PBS job from one queue to another

2.48.1 Synopsis

qmove <destination> <job ID> [<job ID> ...]

qmove --version

2.48.2 Description

Moves a job from one queue to another.

The behavior of the qmove command may be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

2.48.2.1 Restrictions

The qmove command can be used on job arrays, but not on subjobs or ranges of subjobs.

Job arrays can only be moved from one server to another if they are in the '*Q*', '*H*', or '*W*' states, and only if there are no running subjobs. The state of the job array is preserved, and the job array will run to completion on the new server.

A job in the *Running*, *Transiting*, or *Exiting* state cannot be moved.

A job in the process of provisioning cannot be moved.

2.48.2.2 Effect of Privilege on Behavior

An unprivileged user can use the qmove command to move a job only when the move would not violate queue restrictions. A privileged user (root, Manager, Operator) can use the qmove command to move a job under some circumstances where an unprivileged user cannot. The following restrictions apply only to unprivileged users:

- The queue must be enabled
- Moving the job into the queue must not exceed the queue's limits for jobs or resources
- If the job is an array job, the size of the job array must not exceed the queue's max_array_size
- The queue cannot have its from_route_only attribute set to *True* (accepting jobs only from routing queues)

2.48.3 Options

--version

The qmove command returns its PBS version information and exits. This option can only be used alone.

2.48.4 Operands

destination

Where job(s) are to end up. First operand. Syntax:

<queue name>

Moves the job(s) into the specified queue at the job's current server.

@<server name>

Moves the job(s) into the default queue at specified server.

<queue name>@<server name>

Moves the job(s) into the specified queue at the specified server.

See Chapter 7, "Formats", on page 353 for destination identifier formats.

job ID

Job(s) and/or job array(s) to be moved to the new destination . The `qmove` command accepts one or more *job ID* operands of the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.48.5 Standard Error

The `qmove` command writes a diagnostic messages to standard error for each error occurrence.

2.48.6 Exit Status

Zero

Upon successful processing of all the operands presented to the `qmove` command.

Greater than zero

If the `qmove` command fails to process any operand.

2.48.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide

2.49 qmsg

Writes message string into one or more job output files

2.49.1 Synopsis

qmsg [-E] [-O] <message string> <job ID> [<job ID> ...]

qmsg --version

2.49.2 Description

Writes a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Also called "sending a message to a job".

The `qmsg` command writes messages into the files of jobs by sending a Message Job batch request to the batch server that owns the job. The `qmsg` command does not directly write the message into the files of the job.

The `qmsg` command can be used on jobs and subjobs, but not on job arrays or ranges of subjobs.

2.49.3 Options

-E
: The message is written to the standard error of each job.

-O
: The message is written to the standard output of each job.

--version
: The `qmsg` command returns its PBS version information and exits. This option can only be used alone.

(no options)
: The message is written to the standard error of each job.

2.49.4 Operands

message string
: The message to be written. String. First operand. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character is added when written to the job's file.

job ID
: The job(s) to receive the message string. This operand follows the *message string* operand. Can be a job or subjob. Cannot be a job array or range of subjobs. The `qmsg` command accepts one or more *job ID* operands.

  Format for job:

  <sequence number>[.<server name>][@<server name>]

  Format for subjob. Note that a subjob has square brackets around its index number:

  <sequence number>[<index>][.<server name>][@<server name>]

2.49.5 Standard Error

The `qmsg` command writes a diagnostic message to standard error for each error occurrence.

2.49.6 Exit Status

Zero
: Upon successful processing of all the operands presented to the `qmsg` command.

Greater than zero
: If the `qmsg` command fails to process any operand.

2.49.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide

2.50 qorder

Swaps queue positions of two PBS jobs

2.50.1 Synopsis

qorder <job ID> <job ID>

qorder --version

2.50.2 Description

Exchanges positions in queue(s) of two jobs, whether in the same or different queue(s).

No attribute of either job, e.g. priority, is changed. The impact of interchanging the order within or between queues is dependent on local job scheduling policy; contact your systems administrator.

2.50.2.1 Restrictions

- A job in the running state cannot be reordered.
- The `qorder` command can be used on job arrays, but not on subjobs or ranges of subjobs.
- The two jobs must be located at the same server.

2.50.2.2 Effect of Privilege on Behavior

For an unprivileged user to reorder jobs, both jobs must be owned by the user. A privileged user (Manager, Operator) can reorder any jobs.

2.50.3 Options

--version

The `qorder` command returns its PBS version information and exits. This option can only be used alone.

2.50.4 Operands

Both operands are job IDs which specify the jobs to be exchanged. The `qorder` command accepts two *job ID* operands of the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

If you specify the server for both jobs, they must be at the same server.

Note that some shells require that you enclose a job array identifier in double quotes.

2.50.5 Standard Error

The `qorder` command writes diagnostic messages to standard error for each error occurrence.

2.50.6 Exit Status

Zero
: Upon successful processing of all the operands presented to the `qorder` command

Greater than zero
: If the `qorder` command fails to process any operand

2.50.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide

2.51 qrerun

Requeues a PBS job

2.51.1 Synopsis

qrerun [-W force] <job ID> [<job ID> ...]

qrerun --version

2.51.2 Description

If possible, kills the specified job(s), then requeues each job in the execution queue from which it was run.

The `qrerun` command can be used on jobs, job arrays, subjobs, and ranges of subjobs. If you give a job array identifier as an argument, the job array is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and those that are completed and deleted. If a you give a subjob or range as an argument, those subjobs are requeued.

2.51.2.1 Restrictions

If a job is marked as not rerunnable, `qrerun` neither kills nor requeues the job. See the `-r` option for the `qsub` and `qalter` commands, and the Rerunable job attribute.

The `qrerun` command cannot requeue a job or subjob which is not running, is held, or is suspended.

2.51.2.2 Required Privilege

PBS Manager or Operator privilege is required to use this command.

2.51.3 Options

-W force

The job is to be requeued even if the vnode on which the job is executing is unreachable, or if the job's substate is *provisioning*.

--version

The `qrerun` command returns its PBS version information and exits. This option can only be used alone.

2.51.4 Operands

The `qrerun` command accepts one or more *job ID* operands of the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

<sequence number>[<index>][.<server name>][@<server name>]

<sequence number>[<index start>-<index end>][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.51.5 Standard Error

The `qrerun` command writes a diagnostic message to standard error for each error occurrence.

2.51.6 Exit Status

Zero
: Upon successful processing of all operands

Greater than zero
: Upon failure to process any operand

2.51.7 See Also

PBS Professional Administrator's Guide, PBS Professional User's Guide

2.52 qrls

Releases holds on PBS jobs

2.52.1 Synopsis

qrls [-h <hold list>] <job ID> [<job ID> ...]

qrls --version

2.52.2 Description

The `qrls` command directly releases or removes holds on batch jobs or job arrays, and indirectly on subjobs with a System hold. You cannot use the command with a specified range of subjobs. If you use `qrls` on a job array which has a System hold because it has one or more subjobs with a System hold, the System hold is removed from the subjobs, then from the job array.

A job may have one or more types of holds which make the job ineligible for execution.

When you `qrls` a job whose Execution_Time attribute is not set to a time in the future, the job changes to the *queued* state. If Execution_Time is in the future, the job changes to the *waiting* state.

Holds can be set by the owner, an Operator, or Manager, when a job has a dependency, or when a job has its Execution_Time attribute set to a time in the future. See "qhold" on page 147.

2.52.2.1 Effect of Privilege on Behavior

The following table shows the holds and the privilege required to release each:

Table 2-21: Hold Types

Hold Type	Meaning	Privilege Required to Release
u	*User*	*Job owner, Operator, Manager, PBS Administrator, root*
o	*Other*	*Operator, Manager, administrator, root*
s	*System*	*Manager, administrator, root, PBS (dependency)*
n	*No hold*	*Job owner, Operator, Manager, administrator, root*
p	*Bad password*	*Administrator, root*

If you try to release a hold for which the you do not have privilege, the entire request is rejected, and no holds are released.

2.52.3 Options

(no options)

Defaults to `-h u`, removing *user* hold.

-h <hold list>

Types of hold to be released for the jobs. The *hold list* option argument is a string consisting of one or more of the letters *u, o,* or *s* in any combination, or one of the letters *n* or *p*.

--version

The `qrls` command returns its PBS version information and exits. This option can only be used alone.

2.52.4 Operands

The `qrls` command can be used directly on jobs and job arrays, but indirectly on subjobs, and cannot be used on ranges of subjobs. The `qrls` command accepts one or more *job ID* operands of the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.52.5 Standard Error

The `qrls` command writes a diagnostic message to standard error for each error occurrence.

2.52.6 Exit Status

Zero

Upon successful processing of all the operands presented to the `qrls` command

Greater than zero

If the `qrls` command fails to process any operand

2.52.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "qhold" on page 147

2.53 qrun

Runs a PBS job immediately

2.53.1 Synopsis

qrun [-a] [-H <vnode specification>] <job ID> [<job ID> ...]

qrun [-a] [-H -] <job ID> [<job ID> ...]

qrun --version

2.53.2 Description

Forces a job to run, regardless of scheduling position or resource requirements.

The `qrun` command can be used on jobs, subjobs, or ranges of subjobs, but not on job arrays. When it is used on a range of subjobs, the non-running subjobs in that range are run.

When preemption is enabled, a scheduler preempts other jobs in order to run this job. Running a job via `qrun` gives the job higher preemption priority than any of the priorities defined in the preempt_prio scheduler parameter. See "Using Preemption" on page 180 in the PBS Professional Administrator's Guide.

2.53.2.1 Required Privilege

In order to execute `qrun`, you must have PBS Operator or Manager privilege.

2.53.2.2 Caveats for qrun

- The job is run without respect for limits, primetime, or dedicated time.
- If you use a `-H <vnode specification>` option to run a job, but specify insufficient vnodes or resources, the job may not run correctly. Avoid using this option unless you are sure.
- If you don't use the -H option, the job must be in the *Queued* state and reside in an execution queue.
- If you do use the -H option, the job must be in the *Queued* or *Suspended* state and reside in an execution queue.
- The `qrun` command cannot be used on a job that is in the process of provisioning.
- If you use the `-H` option, all schedulers are bypassed, and partition boundaries are ignored.
- If you use `qrun` on a subjob, PBS will try to run the subjob regardless of whether the job has hit the limit specified in max_run_subjobs.

2.53.3 Options to `qrun`

-a

The `qrun` command exits before the job actually starts execution.

(no -H option)

The job is run immediately regardless of scheduling policy as long as the following are true:

- The queue in which the job resides is an execution queue.
- Either the resources required by the job are available, or preemption is enabled and the required resources can be made available by preempting jobs that are running.

The `qrun` command by itself, with no `-H` option, overrides the following:

- Limits on resource usage by users, groups, and projects
- Limits on the number of jobs that can be run at a vnode
- Boundaries between primetime and non-primetime, specified in backfill_prime
- Whether the job is in a primetime queue: you can run a job in a primetime slot even when it's not prime-time, or vice versa. Primetime boundaries are not honored.
- Dedicated time: you can run a job in a dedicated time slot, even if it's not in a dedicated time queue, and vice versa. However, dedicated time boundaries are still honored.

The `qrun` command by itself, with no `-H` option, does not override the following:

- Server and queue resource usage limits

(with -H option)

Do **NOT** use this option unless you know exactly what you are doing.

With the `-H` option, all scheduling policies are bypassed and the job is run directly. The job is run immediately on the named or previously-assigned vnodes, regardless of current usage on those vnodes or which scheduler manages those vnodes, with the exception of vnode state. The job is not run and the `qrun` request is rejected if any named vnode is down, already allocated exclusively, or would need to be allocated exclusively and another job is already running on the vnode. The job is run if the vnode is *offline*.

The `-H` option runs jobs that are queued or suspended.

If the `qrun -H` command is used on a job that requests an AOE, and that AOE is not instantiated on those vnodes, the vnodes are provisioned with the AOE.

If the job requests an AOE, and that AOE is not available on the specified vnodes, the job is held.

-H <vnode specification without resources>

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS applies one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H <vnode specification with resources>

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode specification with resources*, using it instead of the original specification from the user. Any single resource specification results in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

-H -

Runs the job on the set of resources to which it is already assigned. You can run a job on the set of resources already assigned to the job, without having to list the resources, by using the **-** (dash) argument to the `-H` option.

--version

The `qrun` command returns its PBS version information and exits. This option can only be used alone.

2.53.4 Operands

Job ID

The `qrun` command accepts a list of job IDs, of the form:

<sequence number>[.<server name>][@<server name>]

<sequence number>[<index>][.<server name>][@<server name>]

<sequence number>[<index start>-<index end>][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

vnode specification

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where vchunk has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

A *vnode name* is the name of the vnode, not the name of the host.

2.53.5 Standard Error

The `qrun` command writes a diagnostic message to standard error for each error occurrence.

2.53.6 Exit Status

Zero

On success

Greater than zero

If the `qrun` command fails to process any operand

2.53.7 See Also

The PBS Professional Administrator's Guide

2.54 qselect

Selects specified PBS jobs

2.54.1 Synopsis

qselect [-a [<op>] <date and time>] [-A <account string>] [-c [<op>] <interval>] [-h <hold list>] [-H] [-J] [-l <resource list>] [-N <name>] [-p [<op>] <priority>] [-P <project>] [-q <destination>] [-r <rerun>] [-s <states>] [-t <time option> [<comparison>] <specified time>] [-T] [-u <user list>] [-x]

qselect --version

2.54.2 Description

The `qselect` command lists those jobs that meet the specified selection criteria. You can compare certain job attribute values to specified values using a comparison operator shown as *op* in the option description.

You can select jobs, job arrays, or subjobs. You can select jobs from one server per call to the command.

Each option acts as a filter restricting which jobs are listed.

You can select jobs according to the values of some of the resources in the Resource_List job attribute. You can also select jobs according the selection directive (although because this is a string, you can only check for equality or inequality.)

Jobs that are finished or moved are listed only when the `-x` or `-H` options are used. Otherwise, job selection is limited to queued and running jobs.

2.54.2.1 Comparison Operations

You can select jobs by comparing the values of certain job attributes to values you specify. The following table lists the comparison operations you can use:

Table 2-22: Comparison Operations

Operation	Type of Comparison
.eq.	The value of the job attribute is equal to the value of the option argument.
.ne.	The value of the job attribute is not equal to the value of the option argument.
.ge.	The value of the job attribute is greater than or equal to the value of the option argument.
.gt.	The value of the job attribute is greater than the value of the option argument.
.le.	The value of the job attribute is less than or equal to the value of the option argument.
.lt.	The value of the job attribute is less than the value of the option argument.

For example, to select jobs whose Priority attribute has a value greater than 5:

```
qselect -p.gt.5
```

Where an optional comparison is not specified, the comparison operation defaults to *.eq*, meaning PBS checks whether the value of the attribute is equal to the option argument.

2.54.2.2 Required Permissions

When selecting jobs according to resource values, users without operator or manager privilege cannot specify custom resources which were created to be invisible to unprivileged users.

2.54.3 Options to `qselect`

(no options)

Lists all jobs at the server which the user is authorized to list (query status of).

-a [<op>] <date and time>

Deprecated. Restricts selection to those jobs whose Execution_Time attribute qualifies when compared to the *date and time* argument. You can select a range of execution times by using this option twice, to compare to a minimum time and a maximum time.

The *date and time* argument has the format:

[[CC]YY]MMDDhhmm[.SS]

where *MM* is the two digits for the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and the optional *SS* is the seconds. *CC* is the century and *YY* the year.

-A <account string>

Restricts selection to jobs whose Account_Name attribute matches the specified *account string* .

-c [<op>] <interval>

Restricts selection to jobs whose Checkpoint interval attribute meets the comparison criteria.

The *interval* argument can take one of the following values:

c

c=<minutes>

n

s

w

w=<minutes>

We give the range of interval values for the Checkpoint attribute the following ordered relationship:

n > s > c=<minutes> > c > u

(Information about w and w=<minutes> is not available.)

For an interval value of "*u*", only "*.eq.*" and "*.ne.*" are valid.

-h <hold list>

Restricts the selection of jobs to those with a specific set of hold types. The holds in the Hold_Types job attribute must be the same as those in the *hold list* argument, but can be in a different order.

The *hold list* argument is a string consisting of the single letter *n*, or one or more of the letters *u, o, p*, or *s* in any combination. If letters are duplicated, they are treated as if they occurred once. The letters represent the hold types:

Table 2-23: Hold Types

Letter	Hold Type
n	None
u	User
o	Other
p	Bad password
s	System

-H

Restricts selection to finished and moved jobs.

-J

Limits selection to job arrays only.

-l <resource list>

Restricts selection of jobs to those with specified resource amounts. Resource must be job-wide, or be mem, ncpus, or vmem.

The *resource list* is in the following format:

<resource name> <op> <value>[,<resource name> <op> <value> ...]

You must specify *op*, and you can use any of the comparison operators.

Because resource specifications for chunks using the select statement, and placement using the place statement, are stored as strings, the only useful operators for these are *.eq.* and *.ne.*

Unprivileged users cannot specify custom resources which were created to be invisible to unprivileged users.

-N <name>

Restricts selection of jobs to those with the specified value for the Job_Name attribute.

-p [<op>]<priority>

Restricts selection of jobs to those with the specified Priority value(s).

-P <project>

Restricts selection of jobs to those matching the specified value for the project attribute.

Format: *Project Name*; see "Project Name" on page 357

-q <destination>

Restricts selection to those jobs at the specified *destination*.

The *destination* may take of one of the following forms:

<queue name>

Restricts selection to the specified queue at the default server.

@<server name>

Restricts selection to the specified server.

<queue name>@<server name>

Restricts selection to the specified queue at the specified server.

If the -q option is not specified, jobs are selected from the default server.

-r <rerun>

Restricts selection of jobs to those with the specified value for the Rerunable attribute . The option argument *rerun* must be a single character, either *y* or *n* .

-s <states>

Restricts job selection to those whose job_state attribute has the specified value(s).

The *states* argument is a character string consisting of any combination of these characters: *B, E, F, H, M, Q, R, S, T, U, W*, and *X*. (A repeated character is accepted, but no additional meaning is assigned to it.)

Table 2-24: Job States

State	Meaning
B	Job array has started execution
E	The *Exiting* state
F	The *Finished* state
H	The *Held* state
M	The *Moved* state
Q	The *Queued* state
R	The *Running* state
S	The *Suspended* state
T	The *Transiting* state
U	Job suspended due to workstation user activity
W	The *Waiting* state
X	The *eXited* state. Subjobs only

Jobs in any of the specified states are selected.

Job arrays are never in states *R, S, T*, or *U*. Subjobs may be in those states.

-t <time option> [<op>] <specified time>

Jobs are selected according to one of their time-based attributes. The *time option* specifies which time-based attribute is tested. You give the *specified time* in *datetime* format. See Chapter 7, "Formats", on page 353.

The *time option* is one of the following:

Table 2-25: Sub-options to the `-t` Option

Time Option	Time Attribute	Option Format(s)	Attribute Description
a	Execution_Time	*Timestamp* Use *datetime* format to specify.	Time at which the job is eligible for execution.
c	ctime	*Timestamp* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.	Time at which the job was created.
e	etime	*Timestamp* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.	Time when job became eligible to run, i.e. was enqueued in an execution queue and was in the "*Q*" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.
g	eligible_time	Use *duration* format to specify.	Amount of eligible time job accrued waiting to run.
m	mtime	*Timestamp* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.	Time that the job was last modified, changed state, or changed locations.
q	qtime	*Timestamp* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.	Time that the job entered the current queue.
s	stime	*Timestamp* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch	Time the job started. Updated when job is restarted. .
t	estimated.start_time	Use *datetime* format to specify. Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.	Job's estimated start time.

To bracket a time period, use the `-t` option twice. For example, to select jobs using stime between noon and 3 p.m.:

```
qselect -ts.gt.09251200 -ts.lt.09251500
```

-T

Limits selection to jobs and subjobs.

-u <user list>

Restricts selection to jobs owned by the specified usernames.

Syntax of *user list*:

<username>[@<hostname>][,<username>[@<hostname>],...]

Selects jobs which are owned by the listed users at the corresponding hosts. Hostnames may be wildcarded on the left end, e.g. "*.nasa.gov". A username without a "@<hostname>" is equivalent to "<username>@*", meaning that it is valid at any host.

-x

Selects finished and moved jobs in addition to queued and running jobs.

--version

The `qselect` command returns its PBS version information and exits. This option can only be used alone.

2.54.4 Standard Output

PBS writes a list of the selected job IDs to standard output. Each job ID is separated by white space. A job ID can represent a job, a job array, or a subjob. Each job ID has one of the forms:

<sequence number>.<server name>[@<server name>]

<sequence number>[].<server name>[@<server name>]

<sequence number>[<index>].<server name>[@<server name>]

@<server name> identifies the server which currently owns the job.

2.54.5 Standard Error

The `qselect` command writes a diagnostic message to standard error for each error occurrence.

2.54.6 Exit Status

Zero

Upon successful processing of all options presented to the `qselect` command

Greater than zero

If the `qselect` command fails to process any option

2.54.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, section 6.11, “Job Attributes”, on page 328, Chapter 5, "List of Built-in Resources", on page 259

2.55 qsig

Send signal to PBS job

2.55.1 Synopsis

qsig [-s <signal>] <job ID> [<job ID> ...]

qsig --version

2.55.2 Description

The `qsig` command sends a signal to all the processes of the specified job(s). The `qsig` command sends a Signal Job batch request to the server which owns the job.

The `qsig` command can be used for jobs, job arrays, subjobs, and ranges of subjobs. If it is used on a range of subjobs, the running subjobs in the range are signaled.

Not all signal names are recognized by `qsig`; if using a signal name does not work, try issuing the signal number instead.

2.55.2.1 Using admin-suspend and admin-resume

If you have a vnode requiring maintenance while remaining powered up, where you don't want jobs running during the maintenance, you can use the special signals *admin-suspend* and *admin-resume* to suspend and resume the jobs on the vnode. When you use *admin-suspend* on a vnode's job(s), the vnode goes into the *maintenance* state, and its scheduler does not schedule jobs on it. You must separately *admin-suspend* each job on the vnode. When its last *admin-suspend*ed job is *admin-resume*d, a vnode leaves the *maintenance* state.

2.55.2.2 Restrictions

The request to signal a job is rejected if:

- The user is not authorized to signal the job
- The job is not in the *running* or *suspended* state
- The requested signal is not supported by the system upon which the job is executing
- The job is in the process of provisioning
- You attempt to use *admin-resume* on a job that was *suspend*ed
- You attempt to use *resume* on a job that was *admin-suspend*ed

2.55.2.3 Required Privilege

Manager or Operator privilege is required to use the *admin-suspend*, *admin-resume*, *suspend*, or *resume* signals. Unprivileged users can use other signals.

2.55.3 Options to qsig

(no options)

PBS sends SIGTERM to the job.

-s <signal>

PBS sends signal *signal* to the job.

--version

The `qsig` command returns its PBS version information and exits. This option can only be used alone.

2.55.3.1 Signals

You can send standard signals to a job, or the special signals described below. The *signal* argument can be in any of the following formats:

- A signal name, e.g. *SIGKILL*
- A signal name without the SIG prefix, e.g. *KILL*
- An unsigned signal number, e.g. *9*

The signal name *SIGNULL* is allowed; in this case the server sends the signal 0 to the job, which has no effect.

2.55.3.1.i Special Signals

The following special signals are all lower-case, and have no associated signal number:

admin-suspend

Suspends a job and puts its vnodes into the *maintenance* state. The job is put into the *S* state and its processes are suspended. When suspended, a job is not executing and is not charged for walltime.

Syntax: *qsig -s admin-suspend <job ID>*

admin-resume

Resumes a job that was suspended using the *admin-suspend* signal, without waiting for its scheduler. Cannot be used on jobs that were suspended with the *suspend* signal. When the last *admin-suspend*ed job has been *admin-resume*d, the vnode leaves the maintenance state.

Syntax: *qsig -s admin-resume <job ID>*

suspend

Suspends specified job(s). Job goes into *suspended (S)* state. When suspended, a job is not executing and is not charged for walltime.

resume

Marks specified job(s) for resumption by its scheduler when there are sufficient resources. If you use `qsig -s resume` on a job that was suspended using `qsig -s suspend`, the job is resumed when there are sufficient resources. Cannot be used on jobs that were suspended with the *admin_suspend* signal.

2.55.4 Operands

The `qsig` command accepts one or more *job ID* operands. For a job, this has the form:

<sequence number>[.<server name>][@<server name>]

For a job array, *job ID* takes this form:

<sequence number>[][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.55.5 Standard Error

The `qsig` command writes a diagnostic message to standard error for each error occurrence.

2.55.6 Exit Status

Zero
: Upon successful processing of all the operands presented to the `qsig` command

Greater than zero
: If the `qsig` command fails to process any operand

2.55.7 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide

2.56 qstart

Turns on scheduling or routing for the jobs in a PBS queue

2.56.1 Synopsis

qstart <destination> [<destination> ...]

qstart --version

2.56.2 Description

If *destination* is an execution queue, the `qstart` command allows a PBS scheduler to schedule jobs residing in the specified queue. If *destination* is a routing queue, the server can begin routing jobs from that queue. Sets the value of the queue's started attribute to *True*.

2.56.2.1 Required Privilege

In order to execute `qstart`, you must have PBS Operator or Manager privilege.

2.56.3 Options

--version

The `qstart` command returns its PBS version information and exits. This option can only be used alone.

2.56.4 Operands

The `qstart` command accepts one or more space-separated *destination* operands. The operands take one of three forms:

<queue name>

Starts scheduling or routing from the specified queue.

@<server name>

Starts scheduling or routing from all queues at the specified server.

<queue name>@<server name>

Starts scheduling or routing from the specified queue at the specified server.

To start scheduling at all queues at the default server, use the `qmgr` command:

```
Qmgr: set queue @default started=true
```

2.56.5 Standard Error

The `qstart` command writes a diagnostic message to standard error for each error occurrence.

2.56.6 Exit Status

Zero
: Upon successful processing of all the operands presented to the `qstart` command

Greater than zero
: If the `qstart` command fails to process any operand

2.56.7 See Also

The PBS Professional Administrator's Guide, "qmgr" on page 149, "qstop" on page 211

2.57 qstat

Displays status of PBS jobs, queues, or servers

2.57.1 Synopsis

2.57.1.1 Displaying Job Status

Default format:

qstat [-E] [-J] [-p] [-t] [-w] [-x] [[<job ID> | <destination>] ...]

Long format:

qstat -f [-F json|dsv [-D <delimiter>]] [-E] [-J] [-p] [-t] [-w] [-x] [[<job ID> | <destination>] ...]

Alternate format:

qstat [-a | -H | -i | -r] [-E] [-G | -M] [-J] [-n [-1]] [-s [-1]] [-t] [-T] [-u <user list>] [-w] [[<job ID> | <destination>] ...]

2.57.1.2 Displaying Queue Status

Default format:

qstat -Q [<destination> ...]

Long format:

qstat -Q -f [-F json|dsv [-D <delimiter>]] [-w] [<destination> ...]

Alternate format:

qstat -q [-G | -M] [<destination> ...]

2.57.1.3 Displaying Server Status

Default format:

qstat -B [<server name> ...]

Long format:

qstat -B -f [-F json|dsv [-D <delimiter>]] [-w] [<server name> ...]

2.57.1.4 Displaying Version Information

qstat --version

2.57.2 Description

The `qstat` command displays the status of jobs, queues, or servers, writing the status information to standard output.

When displaying job status information, the `qstat` command displays status information about all specified jobs, job arrays, and subjobs. You can specify jobs by ID, or by destination, for example all jobs at a specified queue or server.

2.57.2.1 Display Formats

You can use particular options to display status information in a default format, an alternate format, or a long format. Default and alternate formats display all status information for a job, queue, or server with one line per object, in columns. Long formats display status information showing all attributes, one attribute to a line.

2.57.2.2 Displaying Information for Finished and Moved Jobs

You can display status information for finished and moved jobs by using the -x and -H options.

If your job has been moved to another server through peer scheduling, give the job ID as an argument to `qstat`. If you do not specify the job ID, your job will not appear to exist. For example, your job 123.ServerA is moved to ServerB. In this case, you can use:

```
qstat 123
```

or

```
qstat 123.ServerA
```

Specifying the full job name, including the server, avoids the possibility that `qstat` will report on a job named 123.ServerB that was moved to ServerA.

To list all jobs at ServerB, you can use:

```
qstat @ServerB
```

2.57.2.3 Displaying Truncated Data

When the number of characters required would exceed the space available, `qstat` truncates the output and puts an asterisk ("*") in the last position. For example, in default job display format, there are three characters allowed for the number of cores. If the actual output were *1234*, the value displayed would be *12** instead.

2.57.2.4 Required Privilege

Users without Manager or Operator privilege cannot view resources or attributes that are invisible to unprivileged users.

2.57.3 Displaying Job Status

2.57.3.1 Job Status in Default Format

Triggers: no options, or any of the -J, -p, -t, or -x options.

The `qstat` command displays job status in default format when you specify no options, or any of the -J, -p, -t, or -x options. Jobs are displayed one to a line, with these column headers:

```
Job id   Name       User      Time Use S Queue
-------- ---------- --------- -------- - -----
```

Description of columns:

Table 2-26: Description of Default Job Status Columns

Column	Width without -w	Width with -w	Description	
Job ID	17 (22 when max_job_sequence_id > 10 million	30	Job ID assigned by PBS	
Name	16	15	Job name specified by submitter	
User	16	15	Username of job owner	
Time Use or Percent Complete	8	8	The CPU time used by the job. Before the application has actually started running, for example during stage-in, this field is "0". At the point where the application starts accumulating cput, this field changes to "00:00:00". After that, every time the MoM polls for resource usage, the field is updated. The MoM on each execution host polls for the usage of all processes on her host belonging to the job. Usage is summed. The polling interval is short when a job first starts running and lengthens to a maximum of 2 minutes. See "Configuring MoM Polling Cycle" on page 34 in the PBS Professional Administrator's Guide. If you specify -p, the *Time Use* column is replaced with the percentage completed for the job. For a job array this is the percentage of subjobs completed. For a normal job, it is the percentage of allocated CPU time used.	
S	1	1	The job's state. See section 8.1, "Job States", on page 361	
			B	Array job has at least one subjob running
			E	Job is exiting after having run
			F	Job is finished
			H	Job is held
			M	Job was moved to another server
			Q	Job is queued
			R	Job is running
			S	Job is suspended
			T	Job is being moved to new location
			U	Cycle-harvesting job is suspended due to keyboard activity
			W	Job is waiting for its submitter-assigned start time to be reached
			X	Subjob has completed execution or has been deleted
Queue	16	15	The queue in which the job resides	

2.57.3.2 Job Status in Long Format

Trigger: the -f option.

If you specify the -f (full) option, full job status information for each job is displayed in this order:

- The job ID
- Each job attribute, one to a line
- The job's submission arguments
- The job's executable, in JSDL format
- The executable's argument list, in JSDL format

The job attributes are listed as *<name> = <value>* pairs. This includes the exec_host and exec_vnode strings. The full output can be very large.

The exec_host string has this format:

<host1>/<T1><P1>[+<host2>/<T2>*<P2>+...]*

where

T1 is the task slot number (the index) of the job on host1.

P1 is the number of processors allocated to the job from host1. The number of processors allocated does not appear if it is 1.

The exec_vnode string has the format:

(<vnode1>:ncpus=<N1>:mem=<M1>)[+(<vnode2>:ncpus=<N2>:mem=<M2>)+...]

where

N1 is the number of CPUs allocated to that job on *vnode1*.

M1 is the amount of memory allocated to that job on *vnode1*.

2.57.3.3 Job Status in Alternate Format

Triggers: any of the -a, -i, -G, -H, -M, -n, -r, -s, -T, or -u <user list> options.

The qstat command displays job status in alternate format if you specify any of the -a, -i, -G, -H, -M, -n, -r, -s, -T, or -u <user list> options. Jobs are displayed one to a line. If jobs are running and the -n option is specified, or if jobs are finished or moved and the -H and -n options are specified, there is a second line for the exec_host string.

2.57.3.3.i Job Status Alternate Format Output Columns

Alternate format job status output contains the following columns:

```
                                                  Req'd  Req'd   Elap
Job ID Username Queue Jobname SessID NDS TSK Memory Time  S Time
------ -------- ----- ------- ------ --- --- ------ ----- - ----
```

Description of columns:

Table 2-27: Description of Alternate Format Job Status Columns

Column	Width without -w	Width with -w	Description
Job ID	15 (20 when max_job_sequence_id > 10 million)	30	The job ID assigned by PBS
Username	8	15	Username of job owner
Queue	8	15	Queue in which the job resides
Jobname	10	15	Job name specified by submitter
SessID	6	8	Session ID. Appears only if the job is running
NDS	3	4	Number of chunks or vnodes requested by the job
TSK	3	5	Number of CPUs requested by the job
Req'd Memory	6	6	Amount of memory requested by the job
Req'd Time	5	5	If CPU time is requested, shows CPU time. Otherwise, shows walltime
S	1	1	The job's state; see "States" on page 361 for states
Elap Time or Est Start Time	5	5	If CPU time is requested, shows CPU time. Otherwise, shows walltime. If you use the `-P` option, displays estimated start time for queued jobs, replacing the *Elap Time* field with the *Est Start Time* field.

2.57.3.4 Grouping Jobs and Sorting by ID

Trigger: the `-E` option.

You can use the `-E` option to sort and group jobs in the output of `qstat`. The `-E` option groups jobs by server and displays each group by ascending ID. This option also improves `qstat` performance. The following table shows how the `-E` option affects the behavior of `qstat`:

Table 2-28: How -E Option Affects qstat Output

How qstat is Used	Result Without -E	Result With -E
`qstat` (no job ID specified)	Queries the default server and displays result	No change in behavior; same as without `-E` option
`qstat <list of job IDs from single server>`	Displays results in the order specified	Displays results in ascending ID order
`qstat <job IDs at multiple servers>`	Displays results in the order they are specified	Groups jobs by server. Displays each group in ascending order

2.57.4 Displaying Queue Status

2.57.4.1 Queue Status in Default Format

Trigger: the -Q option by itself.

The qstat command displays queue status in default format if the only option is -Q. Queue status is displayed one queue to a line, with these column headers:

```
Queue       Max  Tot  Ena  Str Que  Run  Hld  Wat  Trn  Ext  Type
----------- ---- ---- ---- --- ---- ---- ---- ---- ---- ---- ----
```

Description of columns:

Table 2-29: Description of Default Queue Status Columns

Column	Description
Queue	Queue name
Max	Maximum number of jobs allowed to run concurrently in this queue
Tot	Total number of jobs in the queue
Ena	Whether the queue is enabled or disabled
Str	Whether the queue is started or stopped
Que	Number of queued jobs
Run	Number of running jobs
Hld	Number of held jobs
Wat	Number of waiting jobs
Trn	Number of jobs being moved (transiting)
Ext	Number of exiting jobs
Type	Type of queue: execution or routing

2.57.4.2 Queue Status in Long Format

Trigger: the -q and -f options together.

If you specify the -f (full) option with the -q option, full queue status information for each queue is displayed starting with the queue name, followed by each attribute, one to a line, as <*name*> = <*value*> pairs.

2.57.4.2.i Queue Status: Alternate Format

Triggers: any of the -q, -G, or -M options.

The qstat command displays queue status in the alternate format if you specify any of the -q, -G, or -M options. Queue status is displayed one queue to a line, and the lowest line contains totals for some columns.

These are the alternate format queue status column headers:

```
Queue   Memory CPU Time Walltime Node Run Que Lm State
------- ------ -------- -------- ---- --- --- -- -----
```

Description of columns:

Table 2-30: Description of Queue Alternate Status Columns

Column	Description
Queue	Queue name
Memory	Maximum amount of memory that can be requested by a job in this queue
CPU Time	Maximum amount of CPU time that can be requested by a job in this queue
Walltime	Maximum amount of walltime that can be requested by a job in this queue
Node	Maximum number of vnodes that can be requested by a job in this queue
Run	Number of running and suspended jobs. Lowest row is total number of running and suspended jobs in all the queues shown
Que	Number of queued, waiting, and held jobs. Lowest row is total number of queued, waiting, and held jobs in all the queues shown
Lm	Maximum number of jobs allowed to run concurrently in this queue
State	State of this queue: *E* (enabled) or *D* (disabled), and *R* (running) or *S* (stopped)

2.57.5 Displaying Server Status

2.57.5.1 Server Status in Default Format:

Trigger: the `-B` option.

The `qstat` command displays server status if the only option given is `-B`.

Column headers for default server status output:

```
Server   Max   Tot   Que   Run   Hld   Wat   Trn   Ext   Status
-------- ----- ----- ----- ----- ----- ----- ----- ----- ------
```

Description of columns:

Table 2-31: Description of Server Status Default Display Columns

Column	Description
Server	Name of server
Max	Maximum number of jobs allowed to be running concurrently on the server
Tot	Total number of jobs currently managed by the server
Que	Number of queued jobs
Run	Number of running jobs
Hld	Number of held jobs
Wat	Number of waiting jobs

Table 2-31: Description of Server Status Default Display Columns

Column	Description
Trn	Number of transiting jobs
Ext	Number of exiting jobs
Status	Status of the server

2.57.5.2 Server Status in Long Format

Trigger: the `-f` option.

If you specify the `-f` (full) option, displays full server status information starting with the server name, followed by each server attribute, one to a line, as <*name*> = <*value*> pairs. Includes PBS version information.

2.57.6 Options to `qstat`

2.57.6.1 Generic Job Status Options

-E

Groups jobs by server and displays jobs sorted by ascending ID. When `qstat` is presented with a list of jobs, jobs are grouped by server and each group is displayed by ascending ID. This option also improves `qstat` performance. See section 2.57.3.4, “Grouping Jobs and Sorting by ID”, on page 201.

2.57.6.2 Default Job Status Options

The following options cause job status information to be displayed in default format:

-J

Displays status information for job arrays (not subjobs).

-t

Displays status information for jobs, job arrays, and subjobs. When used with `-J` option, displays status information for subjobs only.

-p

The *Time Use* column is replaced with the percentage completed for the job. For a job array this is the percentage of subjobs completed. For a normal job, it is the percentage of allocated CPU time used.

-x

Displays status information for finished and moved jobs in addition to queued and running jobs.

2.57.6.3 Alternate Job Status Options

The following options cause job status information to be displayed in alternate format:

-a

All queued and running jobs are displayed. If a *destination* is specified, information for all jobs at that *destination* is displayed. If a *job ID* is specified, information about that job is displayed. Always specify this option before the `-n` or `-s` options, otherwise they will not take effect.

-H

Without a job identifier, displays information for all finished or moved jobs. If a *job ID* is given, displays information for that job regardless of its state. If a *destination* is specified, displays information for finished or moved jobs, or specified job(s), at *destination*.

-i

If a *destination* is given, information for queued, held or waiting jobs at that *destination* is displayed. If a *job ID* is given, information about that job is displayed regardless of its state.

-n

The exec_host string is listed on the line below the basic information. If the -l option is given, the exec_host string is listed on the end of the same line. If using the -a option, always specify the -n option after -a, otherwise the -n option does not take effect.

-r

If a *destination* is given, information for running or suspended jobs at that *destination* is displayed. If a *job ID* is given, information about that job is displayed regardless of its state.

-s

Any comment added by the administrator or scheduler is shown on the line below the basic information. If the -l option is given, the comment string is listed on the end of the same line. If using the -a option, always specify the -s option after -a, otherwise the -s option does not take effect.

-T

Displays estimated start time for queued jobs, replacing the *Elap Time* field with the *Est Start Time* field. Jobs with earlier estimated start times are displayed before those with later estimated start times.

Running jobs are displayed before other jobs. Running jobs are sorted by their stime attribute (start time).

Queued jobs whose estimated start times are unset (estimated.start_time = *unset*) are displayed after those with estimated start times, with the unset value shown as a double dash ("--"). Queued jobs with estimated start times in the past are treated as if their estimated start times are unset.

If a job's estimated start time cannot be calculated, the start time is shown as a question mark ("?").

Time displayed is local to the `qstat` command. Current week begins on Sunday.

The following table shows the format for the *Est Start Time* field when the -w option is not used:

Table 2-32: Format for Estimated Start Time Field without -w Option

Format	Job Estimated Start Time	Example
<HH>:<MM>	Today	`15:34`
<2-letter weekday> <HH>	Within 7 days, but after today	`We 15`
<3-letter month name>	This calendar year, but after this week	`Feb`
<YYYY>	Less than or equal to 5 years from today, after this year	`2018`
>5yrs	More than 5 years from today	`>5yrs`

The following table shows the format for the *Est Start Time* field when the -w option is used:

Table 2-33: Format for Estimated Start Time Field with -w Option

Format	Job Estimated Start Time	Example
Today <HH>:<MM>	Today	`Today 13:34`
<Day> <HH>:<MM>	This week, but after today	`Wed 15:34`
<Day> <Month> <Daynum> <HH>:<MM>	This year, but after this week	`Wed Feb 10 15:34`
<Day> <Month> <Daynum> <YYYY> <HH>:<MM>	After this year	`Wed Feb 10 2011 15:34`

When used with the -f option, prints the full timezone-qualified start time.

Estimated start time information can be made unavailable to unprivileged users; in this case, the estimated start time appears to be unset.

-u <user list>

If a *destination* is given, status for jobs at that *destination* owned by users in *user list* is displayed. If a *job ID* is given, status information for that job is displayed regardless of the job's ownership.

Format: *<username>[@<hostname>][, <username>[@<hostname>], ...]* in comma-separated list.

Hostnames may be wildcarded, but not domain names. When no hostname is specified, *username* is for any host.

-w

Can be used with job status in default and alternate formats. Allows display of wider fields up to 120 characters. See section 2.57.3.1, "Job Status in Default Format", on page 198 and section 2.57.3.3, "Job Status in Alternate Format", on page 200 for column widths.

This option is different from the -w option used with the -f long-format option.

-1 (hyphen one)

Reformats qstat output to a single line. Can be used only in conjunction with the -n and/or -s options.

2.57.6.4 Queue Status Options

-Q

Displays queue status in default format. Operands must be *destinations*.

-q

Displays queue status in alternate format. Operands must be *destinations*.

2.57.6.5 Server Status Options

-B

Display server status. Operands must be names of servers.

2.57.6.6 Job, Queue, and Server Status Options

-f [-w]

Full display for long format. Job, subjob, queue, or server attributes displayed one to a line.

JSON output:

PBS reports resources_used values for resources that are created or set in a hook as JSON strings in the output of qstat -f.

If MoM returns a JSON object (a Python dictionary), PBS reports the value as a string in single quotes:

```
resources_used.<resource_name> = '{ <MoM JSON item value>, <MoM JSON item value>, <MoM JSON item value>, ..}'
```

Example: MoM returns { "a":1, "b":2, "c":1,"d": 4} for resources_used.foo_str. We get:

```
resources_used.foo_str='{"a": 1, "b": 2, "c":1,"d": 4}'
```

If MoM returns a value that is not a valid JSON object, the value is reported verbatim.

Example: MoM returns "hello" for resources_used.foo_str. We get:

```
resources_used.foo_str="hello"
```

Optional -w prints each attribute on one unbroken line. Feed characters are converted:

- Newline is converted to backslash concatenated with "n", resulting in "\n"
- Form feed is converted to backslash concatenated with "f", resulting in "\f"

This -w is independent of the -w job output option used with default and alternate formats.

-F dsv [-D <delimiter>]

Prints output in delimiter-separated value format. The default *delimiter* is a pipe ("|"). You can specify a character or a string *delimiter* using the -D argument to the -F dsv option. For example, to use a comma as the delimiter:

```
qstat -f -F dsv -D,
```

If the delimiter itself appears in a value, it is escaped:

- On Linux, the delimiter is escaped with a backslash ("\").
- On Windows, the delimiter is escaped with a caret ("^").

Feed characters are converted:

- Newline is converted to backslash concatenated with "n", resulting in "\n"
- Form feed is converted to backslash concatenated with "f", resulting in "\f"

A newline separates each job from the next. Using newline as the delimiter leads to undefined behavior.

Example of getting output in delimiter-separated value format:

```
qstat -f -Fdsv
Job Id: 1.vbox|Job_Name = STDIN|Job_Owner = root@vbox|job_state = Q|queue = workq|server =
    vbox|Checkpoint = u|ctime = Fri Nov 11 17:57:05 2016|Error_Path = ...
```

-F json

Prints output in JSON format (http://www.json.org/).

Attribute output is preceded by timestamp, PBS version, and PBS server hostname.

Example:

```
qstat -f -F json
{
    "timestamp":1479277336,
    "pbs_version":"14.1",
    "pbs_server":"vbox",
    "Jobs":{
        "1.vbox":{
            "Job_Name":"STDIN",
            "Job_Owner":"root@vbox",
            "job_state":"Q",
...
```

-G

Shows size in gigabytes. Triggers alternate format.

-M

Shows size in megawords. A word is considered to be 8 bytes. Triggers alternate format.

2.57.6.7 Version Information

--version

The qstat command returns its PBS version information and exits. This option can only be used alone.

2.57.7 Operands

2.57.7.1 Job Identifier Operands

The *job ID* is assigned by PBS at submission. Job IDs are used only with job status requests. Status information for specified job(s) is displayed. Formats:

Job ID:

<sequence number>[.<server name>][@<server name>]

Job array ID:

<sequence number>[][.<server name>][@<server name>]

Subjob ID:

<sequence number>[<index>][.<server name>][@<server name>]

Range of subjobs:

<sequence number>[<index start>-<index end>][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.57.7.2 Destination Operands

Name of queue, name of server, or name of queue at a specific server. Formats:

queue name

Specifies name of queue for job or queue display.

- When displaying job status, PBS displays status for all jobs in the specified queue at the default server.
- When displaying queue status, PBS displays status for the specified queue at the default server.

queue name@server name

Specifies name of queue at server for job or queue display.

- When displaying job status, PBS displays status for all jobs in the specified queue at the specified server.
- When displaying queue status, PBS displays status for the specified queue at the specified server.

@server name

Specifies server name for job or queue display.

- When displaying job status, PBS displays status for all jobs at all queues at the specified server.
- When displaying queue status, PBS displays status for all queues at the specified server.

server name

Specifies server name for server display.

- When displaying server status (with the `-B` option) PBS displays status for the specified server.

2.57.8 Standard Error

The `qstat` command writes a diagnostic message to standard error for each error occurrence.

2.57.9 Exit Status

Zero

Upon successful processing of all operands

Greater than zero

If any operands could not be processed

2.57.10 See Also

The PBS Professional User's Guide, the PBS Professional Administrator's Guide, "Attributes" on page 277

2.58 qstop

Prevents PBS jobs in the specified queue from being scheduled or routed

2.58.1 Synopsis

qstop <destination> [<destination> ...]

qstop --version

2.58.2 Description

If *destination* is an execution queue, the `qstop` command stops a scheduler from scheduling jobs residing in *destination*. If *destination* is a routing queue, the server stops routing jobs from that queue. Sets the value of the queue's started attribute to *False*.

2.58.2.1 Required Privilege

You must have PBS Operator or Manager privilege to run this command.

2.58.3 Options

--version

The `qstop` command returns its PBS version information and exits. This option can only be used alone

2.58.4 Operands

The `qstop` command accepts one or more space-separated *destination* operands. The operands take one of three forms:

<queue name>

Stops scheduling or routing from the specified queue.

@<server name>

Stops scheduling or routing from all queues at the specified server.

<queue name>@<server name>

Stops scheduling or routing from the specified queue at the specified server.

To stop scheduling at all queues at the default server, use the `qmgr` command:

```
Qmgr: set queue @default started=false
```

2.58.5 Standard Error

The `qstop` command writes a diagnostic message to standard error for each error occurrence.

2.58.6 Exit Status

Zero

Upon successful processing of all operands presented to the `qstop` command

Greater than zero

If the `qstop` command fails to process any operand

2.58.7 See Also

The PBS Professional Administrator's Guide, "qmgr" on page 149, "qstart" on page 195

2.59 qsub

Submits a job to PBS

2.59.1 Synopsis

qsub [-a <date and time>] [-A <account string>] [-c <checkpoint spec>] [-C <directive prefix>] [-e <path>] [-f] [-h] [-I [-G [-- <GUI application/script>]] | [-X]] [-j <join>] [-J <range> [%<max subjobs]] [-k <discard>] [-l <resource list>] [-m <mail events>] [-M <user list>] [-N <name>] [-o <path>] [-p <priority>] [-P <project>] [-q <destination>] [-r <y | n>] [-R <remove options>] [-S <path list>] [-u <user list>] [-v <variable list>] [-V] [-W <additional attributes>] [-z] [- | <script> | -- <executable> [<arguments to executable>]]

qsub --version

2.59.2 Description

You use the `qsub` command to submit a batch job to PBS. Submitting a PBS job specifies a task, requests resources, and sets job attributes.

The `qsub` command can read from a job script, from standard input, or from the command line. When the user has submitted the job, PBS returns the job identifier for that job. For a job, this is of the form:

<sequence number>.<server name>

For an array job, this is of the form:

<sequence number>[].<server name>

During execution, jobs can be interactive or non-interactive. Interactive jobs are not rerunnable, and if they are blocking, you cannot use their exit status.

Jobs are run as the user and group who submitted the job.

2.59.2.1 Background Process

By default, on the first invocation, `qsub` spawns a background process to manage communication with the PBS server. Later invocations of `qsub` attempt to communicate with this background process. Under certain circumstances, calls to `qsub` when it uses the background process can result in communication problems. You can prevent `qsub` from spawning a background process by using the `-f` option, although this can degrade performance.

2.59.2.2 Where PBS Puts Job Files

By default, PBS copies the `stdout` and `stderr` files from the job back to the current working directory where the `qsub` command is executed. However, you can specify the output paths using the `-o` and `-e` options. You can also specify which and whether these files should be kept on the execution host via the `-k` option, or deleted, using the `-R` option.

See the `-k`, `-o`, `-e`, and `-R` options, and "Managing Output and Error Files", on page 39 of the PBS Professional User's Guide.

2.59.2.3 Submitting Jobs By Using Job Scripts

To submit a PBS job by using a script, you specify a job script on the command line:

qsub [<options>] <script name>

For example:

```
qsub myscript.sh
```

Job scripts are run as the user and group who submitted the job. Job scripts can be written in Python, Linux shells such as `csh` and `sh`, the Windows command batch language, Perl, etc.

A PBS job script consists of the following:

- Optional shell specification
- Any PBS directives
- The user's tasks: programs, commands, or applications
- Optional comments

  Under Windows, comments can contain only ASCII characters. See the PBS Professional User's Guide.

2.59.2.3.i Using Shells and Interpreters

By default, PBS uses your login shell to run your script. You can optionally specify a different shell or interpreter to run your script:

- Via the `-S` option to `qsub`:

  qsub -S <path to shell> <script name>

  For example:

  ```
  qsub -S /bin/bash myscript.sh
  ```

- In the first line of your script. For example:

  ```
  cat myscript.sh
  #!/bin/sh
  #PBS -N MyHelloJob
  print "Hello"
  ```

2.59.2.3.ii Python Job Scripts

You can use the same Python script under Linux or under Windows, if the script is written to be portable. PBS includes a Python package, allowing Python job scripts to run; you do not need to install Python. You can include PBS directives in a Python job script as you would in a Linux shell script. Python job scripts can access Win32 APIs, including the following modules:

Win32api

Win32con

Pywintypes

Example 2-25: We have a Python job script that includes PBS directives:

```
cat myjob.py
#!/usr/bin/python
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

To run a Python job script under Linux, use the Python path on the execution host:

qsub -S <Python path on execution host> <script name>

For example,

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

To run a Python job script under Windows, use the Python path on the execution host:

qsub -S <Python path on execution host> <script name>

For example:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

If the script pathname contains spaces, it must be quoted, for example:

```
qsub -S "C:\Program Files\PBS\bin\pbs_python.exe" <script name>
```

2.59.2.3.iii Linux Shell Job Scripts

Example 2-26: We have a Linux job script named "weatherscript" for a job named "Weather1" which runs the executable "weathersim" on Linux:

```
#!/bin/sh
#PBS -N Weather1
#PBS -l walltime=1:00:00
/usr/local/weathersim
```

To submit the job, the user types:

```
qsub weatherscript <return>
```

2.59.2.3.iv Windows Command Job Scripts

Example 2-27: We have a script named "weather.exe" for a job named "Weather1" which runs under Windows:

```
#PBS -N Weather1
#PBS -l walltime=1:00:00
weathersim.exe
```

To submit the job, the user types:

```
qsub weather.exe <return>
```

In Windows, if you use `notepad` to create a job script, the last line does not automatically get newline-terminated. Be sure to put one explicitly, otherwise, PBS job will get the following error message:

```
More?
```

when the Windows command interpreter tries to execute that last line.

2.59.2.4 Submitting Jobs From Standard Input

To submit a PBS job by typing job specifications at the command line, you type:

qsub [<options>] [-] <return>

then type any directives, then any tasks, followed by:

- Linux: `CTRL-D` on a line by itself
- Windows: `CTRL-Z <return>`

to terminate the input.

The `qsub` command behaves the same both with and without the dash operand.

For example, on Linux:

```
qsub <return>
#PBS -N StdInJob
sleep 100
<CTRL-D>
```

2.59.2.5 Submitting Job Directly by Specifying Executable on Command Line

To submit a job directly, you specify the executable on the command line:

qsub [<options>] -- <executable> [<arguments to executable>] <return>

When you run `qsub` this way, it runs the *executable* directly. It does not start a shell, so no shell initialization scripts are run, and execution paths and other environment variables are not set. There is not an easy way to run your command in a different directory. You should make sure that environment variables are set correctly, and you will usually have to specify the full path to the command.

Example 2-28: To run `myprog` with the arguments *a* and *b*:

```
qsub -- myprog a b <return>
```

Example 2-29: To run `myprog` with the arguments *a* and *b*, naming the job "JobA":

```
qsub -N JobA -- myprog a b <return>
```

2.59.2.6 Requesting Resources and Placing Jobs

Requesting resources includes setting limits on resource usage and controlling how the job is placed on vnodes.

Resources are requested by using the `-l` option, either in job-wide requests using *<resource name>=<value>* pairs, or in chunks inside of *selection statements*. See Chapter 5, "List of Built-in Resources", on page 259.

Job-wide *<resource name>=<value>* requests are of the form:

-l <resource name>=<value>[,<resource name>=<value> ...]

The selection statement is of the form:

-l select=[<N>:]<chunk>[+[<N>:]<chunk> ...]

where *N* specifies how many of that chunk, and a *chunk* is of the form:

<resource name>=<value>[:<resource name>=<value> ...]

You choose how your chunks are placed using the *place statement*. The *place statement* can contain the following elements, in any order:

-l place=[<arrangement>][: <sharing>][: <grouping>]

where

arrangement

Whether this chunk is willing to share this vnode or host with other chunks from the same job. One of *free* | *pack* | *scatter* | *vscatter*

sharing

Whether this this chunk is willing to share this vnode or host with other jobs. One of *excl* | *shared* | *exclhost*

grouping

Whether the chunks from this job should be placed on vnodes that all have the same value for a resource. Can have only one instance of *group=<resource name>*

free

Place job on any vnode(s).

pack

All chunks are taken from one host.

scatter

Only one chunk with any MPI processes is taken from a host. A chunk with no MPI processes may be taken from the same vnode as another chunk.

vscatter

Only one chunk is taken from any vnode. Each chunk must fit on a vnode.

excl

Only this job uses the vnodes chosen.

shared

This job can share the vnodes chosen.

exclhost

The entire host is allocated to the job.

group=<resource name>

Chunks are grouped according to a resource. All vnodes in the group must have a common value for *resource*, which can be either the built-in resource host or a custom vnode-level resource.

resource name must be a string or a string array.

The place statement cannot begin with a colon. Colons are delimiters; use them only to separate parts of a place statement, unless they are quoted inside resource values.

Note that vnodes can have sharing attributes that override job placement requests. See section 6.10, “Vnode Attributes”, on page 320.

For more on resources, resource requests, usage limits, and job placement, see "Using PBS Resources" on page 229 in the PBS Professional Administrator’s Guide and "Allocating Resources & Placing Jobs", on page 49 of the PBS Professional User’s Guide.

2.59.2.6.i Caveats for Requesting Resources

Do not mix old-style resource or vnode specifications with the new *select* and *place* statements. Do not use one in a job script and the other on the command line. Mixing the two will result in an error.

You cannot submit a job requesting a custom resource which has been created to be invisible or read-only for unprivileged users, regardless of your privilege. A Manager or Operator can use the `qalter` command to change a job's request for this kind of custom resource.

2.59.2.7 Setting Attributes

The job submitter sets job attributes by giving options to the `qsub` command or by using PBS directives. Most `qsub` options set a job attribute, and have a corresponding PBS directive with the same syntax as the option. Attributes set via command-line options take precedence over those set using PBS directives. See the PBS Professional User's Guide, `or` section 6.11, “Job Attributes”, on page 328.

2.59.2.8 Running Your Job on First Available Resources (2020.1 Beta)

2020.1 Beta. This feature is subject to drastic change or removal without notice in future releases.

You may want to run a job on whichever resources become available first, even if the job could run on other sets of resources. You may want to start a flexible job as soon as possible on a smaller set of resources rather than waiting longer for a larger set of resources, or you may prefer certain resources but be able to use others (for example, you might prefer a specific processor, but still be able to run on another if that is all that's available).

If you submit a set of jobs where each job has a "runone" dependency on the others, PBS runs only one of the jobs in the "runone set". PBS automatically groups the jobs into a runone set. The jobs in a runone set can run different scripts.

When any of the jobs in the set starts, PBS applies a system hold to the others. The hold on the other jobs is released when the running job is requeued:

- Via `qrerun`
- When node fail requeue is triggered

The other jobs in the set are deleted:

- When a job ends, regardless of its exit status
- When the running job is deleted

To identify a job as a member of the set, give it a "runone" dependency on the previously-submitted member of the set. For example, we have three jobs, each of which runs on different resources. To submit these three jobs as a runone set:

```
qsub -lselect=200:ncpus=16 -lwalltime=1:00:00 myscript.sh
10.myserver
qsub -lselect=100:ncpus=16 -lwalltime=2:00:00 -Wdepend=runone:10 myscript.sh
11.myserver
qsub -lselect=50:ncpus=16 -lwalltime=4:00:00 -Wdepend=runone:10 myscript.sh
12.myserver
```

2.59.2.9 Changing `qsub` Behavior

The behavior of the `qsub` command may be affected by the server's default_qsub_arguments attribute. This attribute can set the default for any job attribute. The default_qsub_arguments server attribute is settable by the administrator, and is overridden by command-line arguments and script directives. See section 6.6, “Server Attributes”, on page 281.

The behavior of the `qsub` command may also be affected by any site hooks. Site hooks can modify the job's attributes, change its routing, etc.

2.59.3 Options to `qsub`

-a <date and time>

Point in time after which the job is eligible for execution. Given in pairs of digits. Sets job's Execution_Time attribute to *date and time*.

Format: *datetime,* expressed as *[[[[CC]YY]MM]DD]hhmm[.SS]*

where *CC* is the century, *YY* is the year, *MM* is the month, *DD* is the day of the month, *hh* is the hour, *mm* is the minute, and *SS* is the seconds.

Each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day *DD* is the 5th, the month *MM* is set to the current month.

If a specified portion has already passed, the next-larger portion is set to one after the current date. For example, if the day *DD* is not specified, but the hour hh is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day *DD* is set to tomorrow.

-A <account string>

Accounting string associated with the job. Used for labeling accounting data. Sets job's Account_Name attribute to *account string*.

Format: *String*

-c <checkpoint spec>

Determines when the job will be checkpointed. Sets job's Checkpoint attribute to *checkpoint spec*. An $action script is required to checkpoint the job.

See "Using Checkpointing", on page 113 of the PBS Professional User's Guide.

The argument *checkpoint spec* can take one of the following values:

c
: Checkpoint at intervals, measured in CPU time, set on job's execution queue. If there is no interval set at the queue, the job is not checkpointed

c=<minutes of CPU time>
: Checkpoint at intervals of specified number of minutes of job CPU time. This value must be greater than zero. If the interval specified is less than that set on the job's execution queue, the queue's interval is used.

  Format: *Integer*

w
: Checkpoint at intervals, measured in walltime, set on job's execution queue. If there is no interval set at the queue, the job is not checkpointed.

w=<minutes of walltime>
: Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less than that set on the job's execution queue, the queue's interval is used.

  Format: *Integer*

n
: No checkpointing.

s
: Checkpoint only when the server is shut down.

u
: Unset. Defaults to behavior when *interval* argument is set to *s*.

Default: *u*

Format: *String*

-C <directive prefix>

Defines the prefix identifying a PBS directive. Default prefix is "*#PBS*".

If the *directive prefix* argument is a null string, `qsub` does not scan the script file for directives. Overrides the PBS_DPREFIX environment variable and the default. The string "PBS_DPREFIX" cannot be used as a PBS directive. Length limit: 4096 characters.

-e <path>

Path to be used for the job's standard error stream. Sets job's Error_Path attribute to *path*. The *path* argument is of the form:

[<hostname>:]<path>

The *path* is interpreted as follows:

path
: If *path* is relative, it is taken to be relative to the current working directory of the `qsub` command, where it is executing on the current host.

  If *path* is absolute, it is taken to be an absolute path on the current host where the `qsub` command is executing.

hostname:path
: If *path* is relative, it is taken to be relative to the user's home directory on the host named *hostname*.

If *path* is absolute, it is an absolute path on the host named *hostname*.

If *path* does not include a filename, the default filename has the form *<job ID>.ER*

If the -e option is not specified, PBS copies the standard error to the current working directory where the qsub command was executed, and writes standard error to the default filename, which has this form:

<job name>.e<sequence number>

If you use a UNC path for output or error files, the *hostname* is optional. If you use a non-UNC path, the *hostname* is required.

This option is overridden by the -k option.

-f

Prevents qsub from spawning a background process. By default, qsub spawns a background process to manage communication with the PBS server. When this option is specified, the qsub process connects directly to the server and no background process is created.

NOTE: Use of this option degrades performance of qsub when calls to qsub are made in rapid succession.

-G [<path to GUI application or script>]

Starts a GUI session. When no application or script is provided, starts a GUI-enabled interactive shell. When an application or script is provided, starts the GUI application or script. Use full path to application or script unless the path is part of the user's PATH environment variable on the execution host. When submission and execution hosts are different, this uses a remote viewer.

Session is terminated when remote viewer, GUI application, or interactive shell is terminated, or when job is deleted.

Can be used only with interactive jobs (the -I option).

Available only under Windows.

-h

Applies a *User* hold to the job. Sets the job's Hold_Types attribute to "*u*".

-I

Job is to be run interactively. Sets job's interactive attribute to *True*. The job is queued and scheduled as any PBS batch job, but when executed, the standard input, output, and error streams of the job are connected to the terminal session in which qsub is running. If a job script is given, only its directives are processed. When the job begins execution, all input to the job is taken from the terminal session. See the PBS Professional User's Guide for additional information on interactive jobs.

Interactive jobs are not rerunnable.

Job arrays cannot be interactive.

When used with -Wblock=true, no exit status is returned.

-j <join>

Specifies whether and how to join the job's standard error and standard output streams. Sets job's Join_Path attribute to *join*.

Default: *n*; not merged

The *join* argument can take the following values:

Table 2-34: Sub-options to -j Option

Suboption	Meaning
oe	Standard error and standard output are merged into standard output.
eo	Standard error and standard output are merged into standard error.
n	Standard error and standard output are not merged.

-J <range> [%<max subjobs>]

Makes this job an array job. Sets job's array attribute to *True*.

Use the *range* argument to specify the indices of the subjobs of the array. *range* is specified in the form *X-Y[:Z]* where *X* is the first index, *Y* is the upper bound on the indices, and *Z* is the stepping factor. For example, 2-7:2 will produce indices of *2*, *4*, and *6*. If *Z* is not specified, it is taken to be *1*. Indices must be greater than or equal to zero.

Use the optional *%max subjobs* argument to set a limit on the number of subjobs that can be running at one time. This sets the value of the max_run_subjobs job attribute to the specified maximum.

Job arrays are always rerunnable.

-k <discard>

Specifies whether and which of the standard output and standard error streams is left behind on the execution host, or written to their final destination. Sets the job's Keep_Files attribute to *discard*. Overrides default path names for these streams. Overrides -o and -e options.

Default: *n*; neither is retained, and files are not written directly to final destinations.

In the case where output and/or error is retained on the execution host in a job-specific staging and execution directory created by PBS, these files are deleted when PBS deletes the directory.

The *discard* argument can take the following values:

Table 2-35: Sub-options to discard Option

Suboption	Meaning
e	The standard error stream is retained on the execution host, in the job's staging and execution directory. The filename is *<job name>.e<sequence number>*
o	The standard output stream is retained on the execution host, in the job's staging and execution directory. The filename is *<job name>.o<sequence number>*
eo, oe	Both standard output and standard error streams are retained on the execution host, in the job's staging and execution directory.
d	Output and/or error are written directly to their final destination. Overrides action of leaving files on execution host.
n	Neither stream is retained.

-l <resource list>

Allows the user to request resources and specify job placement. Sets job's Resource_list attribute to *resource list*. Requesting a resource places a limit on its usage.

For how to request resources and place jobs, see section 2.59.2.6, “Requesting Resources and Placing Jobs”, on page 216.

-m <mail events>

Specifies the set of conditions under which mail about the job is sent. Sets job's Mail_Points attribute to *mail events*. The *mail events* argument can be one of the following:

- The single character "*n*"
- Any combination of "*a*", "*b*", and "*e*", with optional "*j*"

The following table lists the sub-options to the -m option:

Table 2-36: Sub-options to m Option

Suboption	Meaning
n	No mail is sent.
a	Mail is sent when the job is aborted by PBS.
b	Mail is sent when the job begins execution.
e	Mail is sent when the job terminates.
j	Mail is sent for subjobs. Must be combined with one or more of a, b, or e options

Format: *String*

Syntax: *n | [j](one or more of a, b, e)*

Example: -m ja

Default value: *"a"*

-M <user list>

List of users to whom mail about the job is sent. Sets job's Mail_Users attribute to *user list*.

The *user list* argument has the form:

<username>[@<hostname>][,<username>[@<hostname>],...]

Default: Job owner

-N <name>

Sets job's Job_Name attribute and name to *name*.

Format: *Job Name*; see "Job Name, Job Array Name” on page 355

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "*STDIN*".

-o <path>

Path to be used for the job's standard output stream. Sets job's Output_Path attribute to *path*. The *path* argument has the form:

[<hostname>:]<path>

The *path* is interpreted as follows:

path

If *path* is relative, it is taken to be relative to the current working directory of the command, where it is executing on the current host.

If *path* is absolute, it is taken to be an absolute path on the current host where the command is executing.

hostname:path

If *path* is relative, it is taken to be relative to the user's home directory on the host named *hostname*.

If *path* is absolute, it is an absolute path on the host named *hostname*.

If *path* does not include a filename, the default filename has the form *<job ID>.OU*

If the `-o` option is not specified, PBS copies the standard output to the current working directory where the `qsub` command was executed, and writes standard output to the default filename, which has this form:

<job name>.o<sequence number>

If you use a UNC path, the hostname is optional. If you use a non-UNC path, the hostname is required.

This option is overridden by the `-k` option.

-p <priority>

Priority of the job. Sets job's Priority attribute to *priority.*

Format: Host-dependent integer

Range: [-1024, +1023] inclusive

Default: *Zero*

-P <project>

Specifies a project for the job. Sets job's project attribute to *project*.

Format: *Project Name*; see "Project Name" on page 357

Default value: "*_pbs_project_default*".

-q <destination>

Where the job is sent upon submission.

Specifies a queue, a server, or a queue at a server. The destination argument can have one of these formats:

<queue name>

Job is submitted to the specified queue at the default server.

@<server name>

Job is submitted to the default queue at the specified server.

<queue name>@<server name>

Job is submitted to the specified queue at the specified server.

Default: Default queue at default server

-r <y|n>

Declares whether the job is rerunnable. Sets job's Rerunable attribute to the argument value. Does not affect how the job is handled in the case where the job was unable to begin execution.

Format: Single character, "*y*" or "*n*"

Table 2-37: Sub-options to r Option

Suboption	Meaning
y	Job is rerunnable.
n	Job is not rerunnable.

Default: "*y*"

Interactive jobs are not rerunnable. Job arrays are always rerunnable. See "qrerun" on page 178.

-R <remove options>

Specifies whether standard output and/or standard error files are automatically removed (deleted) upon job completion.

Sets the job's Remove_Files attribute to *remove options*. Overrides default path names for these streams. Overrides -o and -e options.

This attribute cannot be altered once the job has begun execution.

Default: *Unset*; neither is removed

The *remove options* argument can take the following values:

Table 2-38: discard Argument Values

Option	Meaning
e	The standard error stream is removed (deleted) upon job completion
o	The standard output stream is removed (deleted) upon job completion
eo, oe	Both standard output and standard error streams are removed (deleted) upon job completion
unset	Neither stream is removed.

-S <path list>

Specifies the interpreter or shell path for the job script. Sets job's Shell_Path_List attribute to *path list*.

The *path list* argument is the full path to the interpreter or shell including the executable name.

Only one path may be specified without a hostname. Only one path may be specified per named host. The path selected is the one whose hostname is that of the server on which the job resides.

Format: *<path>[@<hostname>][,<path>@<hostname> ...]*

Default: User's login shell on execution host

Example of using `bash` via a directive:

```
#PBS -S /bin/bash@mars,/usr/bin/bash@jupiter
```

Example of running a Python script from the command line on Linux:

```
qsub -S $PBS_EXEC/bin/pbs_python <script name>
```

Example of running a Python script from the command line on Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

-u <user list>

List of usernames. Job is run under a username from this list. Sets job's User_List attribute to *user list*.

Only one username may be specified without a hostname. Only one username may be specified per named host. The server on which the job resides will select first the username whose hostname is the same as the server name. Failing that, the next selection is the username with no specified hostname. The usernames on the server and execution hosts must be the same. The job owner must have authorization to run as the specified user.

Format of *user list: <username>[@<hostname>][,<username>@<hostname> ...]*

Default: Job owner (username on submission host)

-v <variable list>

Specifies environment variables and shell functions to be exported to the job. This is the list of environment variables which is added to those already automatically exported. These variables exist in the user's login environment, from which `qsub` is run. The job's Variable_List attribute is appended with the variables in *variable list* and their values. See section 2.59.7, “Environment Variables”, on page 230.

Format: comma-separated list of strings in the form:

<variable>

or

<variable>=<value>

If a *<variable>=<value>* pair contains any commas, the value must be enclosed in single or double quotes, and the *<variable>=<value>* pair must be enclosed in the kind of quotes not used to enclose the value. For example:

```
qsub -v "var1='A,B,C,D'" job.sh
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

Default: No environment variables are added to job's variable list.

-V

All environment variables and shell functions in the user's login environment where `qsub` is run are exported to the job. The job's Variable_List attribute is appended with all of these environment variables and their values.

-W <additional attributes>

The `-W` option allows specification of some job attributes. Some job attributes must be specified using this option. Those attributes are listed below. Format:

-W <attribute name>=<value>[,<attribute name>=<value>...]

If white space occurs within the *additional attributes* argument, or the equal sign "=" occurs within a *value* string, it must be enclosed with single quotes or double quotes.

The following attributes can be set using the `-W` option only:

block=true

The `qsub` command waits for the job to terminate, then returns the job's exit value. Sets job's block attribute to *True*. When used with X11 forwarding or interactive jobs, no exit value is returned. See section 2.59.8, “Exit Status”, on page 231.

create_resv_from_job=<value>

When this job starts, immediately creates and confirms a *job-specific start reservation* on the same resources as the job (including resources inherited by the job), and places the job in the job-specific reservation queue. Sets the job's create_resv_from_job attribute to *True*. Sets the job-specific reservation's reserve_job attribute to the ID of the job from which the reservation was created. The new reservation's duration and start time are the same as the job's walltime and start time. If the job is peer scheduled, the job-specific reservation is created in the pulling complex.

Format: Boolean

Example:

```
qsub myscript.sh -Wcreate_resv_from_job=1
```

Cannot be used with job arrays or jobs being submitted into a reservation.

depend=<dependency list>

Defines dependencies between this and other jobs. Sets the job's depend attribute to *dependency list*. The *dependency list* has the form:

<type>:<arg list>[,<type>:<arg list> ...]

where except for the *on* type, the *arg list* is one or more PBS job IDs, and has the form:

<job ID>[:<job ID> ...]

The type can be:

after: <arg list>

This job may be scheduled for execution at any point after all jobs in *arg list* have started execution.

afterok: <arg list>

This job may be scheduled for execution only after all jobs in *arg list* have terminated with no errors. See section 2.59.8.1, "Warning About Exit Status with csh", on page 232.

afternotok: <arg list>

This job may be scheduled for execution only after all jobs in *arg list* have terminated with errors. See section 2.59.8.1, "Warning About Exit Status with csh", on page 232.

afterany: <arg list>

This job may be scheduled for execution after all jobs in *arg list* have finished execution, with any exit status (with or without errors.) This job will not run if a job in the *arg list* was deleted without ever having been run.

before: <arg list>

Jobs in *arg list* may begin execution once this job has begun execution.

beforeok: <arg list>

Jobs in *arg list* may begin execution once this job terminates without errors. See section 2.59.8.1, "Warning About Exit Status with csh", on page 232.

beforenotok: <arg list>

If this job terminates execution with errors, jobs in *arg list* may begin. See section 2.59.8.1, "Warning About Exit Status with csh", on page 232.

beforeany: <arg list>

Jobs in *arg list* may begin execution once this job terminates execution, with or without errors.

on: <count>

This job may be scheduled for execution after *count* dependencies on other jobs have been satisfied. This type is used in conjunction with one of the *before* types listed. *count* is an integer greater than 0.

runone:<job ID>

(2020.1 Beta) Puts the current job and the job with *job ID* in a set of jobs out of which PBS will eventually run just one. To add a job to a set, specify the job ID of another job already in the set.

Job IDs in the *arg list* of *before* types must have been submitted with a type of *on*.

To use the *before* types, the user must have the authority to alter the jobs in *arg list*. Otherwise, the dependency is rejected and the new job aborted.

Error processing of the existence, state, or condition of the job on which the newly submitted job is performed after the job is queued. If an error is detected, the new job is deleted by the server. Mail is sent to the job submitter stating the error.

Dependency examples:

```
qsub -W depend=afterok:123.host1.domain.com /tmp/script
```

```
qsub -W depend=before:234.host1.com:235.host1.com /tmp/script
```

group_list=<group list>
List of group names. Job is run under a group name from this list. Sets job's group_list attribute to *group list*.

Only one group name may be specified without a hostname. Only one group name may be specified per named host. The server on which the job resides will select first the group name whose hostname is the same as the server name. Failing that, the next selection is the group name with no specified hostname. The group names on the server and execution hosts must be the same. The job submitter's primary group is automatically added to the list.

Under Windows, the primary group is the first group found for the user by PBS when it queries the accounts database.

Format of *group list*: *<group name>[@<hostname>][,<group name>@<hostname> ...]*

Default: Login group name of job owner

pwd

pwd=''

pwd=""

These forms prompt the user for a password. A space between `W` and `pwd` is optional. Spaces between the quotes are optional. Examples:

```
qsub ... -Wpwd <return>
qsub ... -W pwd='' <return>
qsub ... -W pwd=" " <return>
```

Available on supported Linux platforms only.

release_nodes_on_stageout=<value>
When set to *True*, all of the job's vnodes not on the primary execution host are released when stageout begins.

Cannot be used with vnodes tied to Cray X* series systems.

When cgroups is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released.

The job's stageout attribute must be set for the release_nodes_on_stageout attribute to take effect.

Format: *Boolean*

Default: *False*

run_count=<value>
Sets the number of times the server thinks it has run the job. Sets the value of the job's run_count attribute to *value*.

Format: Integer greater than or equal to zero

sandbox=<sandbox spec>
Determines which directory PBS uses for the job's staging and execution. Sets job's sandbox attribute to the value of *sandbox spec*.

Allowed values for *sandbox spec*:

PRIVATE
PBS creates a job-specific directory for staging and execution.

HOME or unset
PBS uses the user's home directory for staging and execution.

Format: *String*

stagein=<path list>

stageout=<path list>

Specifies files or directories to be staged in before execution or staged out after execution is complete. Sets the job's stagein and stageout attributes to the specified *path list*s. On completion of the job, all staged-in and staged-out files and directories are removed from the execution host(s). The *path list* has the form:

<file spec>[,<file spec>]

where *<file spec>* is

<execution path>@<hostname>:<storage path>

regardless of the direction of the copy. The name *execution path* is the name of the file or directory on the primary execution host. It can be relative to the staging and execution directory on the execution host, or it can be an absolute path.

The "@" character separates *execution path* from *storage path*.

The name *storage path* is the path on *hostname*. The name can be relative to the staging and execution directory on the primary execution host, or it can be an absolute path.

If *path list* has more than one *file spec*, i.e. it contains commas, it must be enclosed in double quotes.

If you use a UNC path, the *hostname* is optional. If you use a non-UNC path, the *hostname* is required.

umask=<mask value>

The umask with which the job is started. Sets job's umask attribute to *mask value*. Controls umask of job's standard output and standard error.

The following example allows group and world read of the job's output and error:

```
-W umask=33
```

Format: one to four digits; typically two

Default: *077*

-X

Allows user to receive X output from interactive job.

DISPLAY variable in submission environment must be set to desired display.

Can be used only with interactive jobs: must be used with one of the following:

```
-I
```

```
-W interactive=true (deprecated)
```

Cannot be used with `-v DISPLAY`.

When used with `-Wblock=true`, no exit status is returned.

Can be used with `-V` option.

Not available under Windows.

-z

Job identifier is not written to standard output.

--version

The `qsub` command returns its PBS version information and exits. This option can only be used alone.

2.59.4 Operands

The `qsub` command accepts as operands one of the following:

(no operands)

Same as with a dash. Any PBS directives and user tasks are read from the command line.

<script>

Path to script. Can be absolute or relative to current directory where qsub is run.

-

When you use a dash, any PBS directives and user tasks are read from the command line.

-- <executable> [<arguments to executable>]

A single executable (preceded by two dashes) and its arguments

The executable, and any arguments to the executable, are given on the qsub command line. The executable is preceded by two dashes, "--".

If a script or executable is specified, it must be the last argument to qsub. The arguments to an executable must follow the name of the executable.

When you run qsub this way, it runs the executable directly. It does not start a shell, so no shell initialization scripts are run, and execution paths and other environment variables are not set. You should make sure that environment variables are set correctly.

2.59.5 Standard Output

Job ID for submitted job

If the job is successfully created

(No output)

If the -z option is set

2.59.6 Standard Error

The qsub command writes a diagnostic message to standard error for each error occurrence.

2.59.7 Environment Variables

The qsub command uses the following environment variables:

PBS_DEFAULT

Name of default server.

PBS_DPREFIX

Prefix string which identifies PBS directives.

Environment variables beginning with "*PBS_O_*" are created by qsub. PBS automatically exports the following environment variables to the job, and the job's Variable_List attribute is set to this list:

PBS_ENVIRONMENT

Set to *PBS_BATCH* for a batch job. Set to *PBS_INTERACTIVE* for an interactive job. Created when qsub is run.

PBS_JOBDIR

Pathname of job's staging and execution directory on the primary execution host.

PBS_JOBID

Job identifier given by PBS when the job is submitted. Created when qsub is run.

PBS_JOBNAME

Job name specified by submitter. Created when qsub is run.

PBS_NODEFILE
: Name of file containing the list of vnodes assigned to the job. Created when `qsub` is run.

PBS_O_HOME
: User's home directory. Value of HOME taken from user's submission environment.

PBS_O_HOST
: Name of submit host. Value taken from user's submission environment.

PBS_O_LANG
: Value of LANG taken from user's submission environment.

PBS_O_LOGNAME
: User's login name. Value of LOGNAME taken from user's submission environment.

PBS_O_MAIL
: Value of MAIL taken from user's submission environment.

PBS_O_PATH
: User's PATH. Value of PATH taken from user's submission environment.

PBS_O_QUEUE
: Name of the queue to which the job was submitted. Value is taken from job submission, otherwise default queue.

PBS_O_SHELL
: Value taken from user's submission environment.

PBS_O_SYSTEM
: Operating system, from `uname -s`, on submit host. Value taken from user's submission environment.

PBS_O_TZ
: Timezone. Value taken from user's submission environment.

PBS_O_WORKDIR
: Absolute path to directory where `qsub` is run. Value taken from user's submission environment.

PBS_QUEUE
: Name of the queue from which the job is executed. Created when `qsub` is run.

PBS_TMPDIR
: Pathname of scratch directory for PBS components. Set when PBS assigns it.

2.59.8 Exit Status

For non-blocking jobs:

Zero
: Upon successful processing of input

Greater than zero
: Upon failure of `qsub`

For blocking jobs:

Exit value of job
: When job runs successfully

3
: If the job is deleted without being run

2.59.8.1 Warning About Exit Status with `csh`

If a job is run in `csh` and a `.logout` file exists in the home directory in which the job executes, the exit status of the job is that of the `.logout` script, not the job script. This may impact any inter-job dependencies.

2.59.9 See Also

"Submitting a PBS Job", on page 11 of the PBS Professional User's Guide, "Job Attributes" on page 328, "Resources Built Into PBS" on page 265, and "Requesting Resources", on page 51 of the PBS Professional User's Guide.

2.60 qterm

Terminates one or both PBS servers, and optionally terminates scheduler(s) and/or MoMs

2.60.1 Synopsis

qterm [-f | -F | -i] [-m] [-s] [-t <type>] [<server name>[<server name> ...]]

qterm --version

2.60.2 Description

The `qterm` command terminates a PBS batch server.

Once the server is terminating, no new jobs are accepted by the server, and no jobs are allowed to begin execution. The impact on running jobs depends on the way the server is shut down.

The `qterm` command does not exit until the server has completed its shutdown procedure.

If the complex is configured for failover, and the primary server is shut down, the normal behavior for the secondary server is to become active. The `qterm` command provides options to manage the behavior of the secondary server; it can be shut down, forced to remain idle, or shut down in place of the primary server.

2.60.2.1 Required Privilege

In order to run the `qterm` command, you must have PBS Operator or Manager privilege.

2.60.3 Options to qterm

(no options)

The `qterm` command defaults to `qterm -t quick`.

-f

If the complex is configured for failover, shuts down both the primary and secondary servers.

Without the `-f` option, `qterm` shuts down the primary server and makes the secondary server active.

The `-f` option cannot be used with the `-i` or `-F` options.

-F

If the complex is configured for failover, shuts down only the secondary server, leaving the primary server active.

The `-F` option cannot be used with the `-f` or `-i` options.

-i

If the complex is configured for failover, leaves the secondary server idle when the primary server is shut down.

The `-i` option cannot be used with the `-f` or `-F` options.

-m

Shuts down the primary server and all MoMs (`pbs_mom`). This option does not cause jobs or subjobs to be killed. Jobs are left running subject to other options to the `qterm` command.

-s

Shuts down the primary server and the scheduler (`pbs_sched`).

-t <type>

immediate
: Shuts down the primary server. Immediately stops all running jobs. Any running jobs that can be checkpointed are checkpointed, terminated, and requeued. Jobs that cannot be checkpointed are terminated and requeued if they are rerunnable, otherwise they are killed.

  If any job cannot be terminated, for example the server cannot contact the MoM of a running job, the server continues to execute and the job is listed as running. The server can be terminated by a second `qterm -t immediate` command.

  While terminating, the server is in the *Terminating* state.

delay
: Shuts down the primary server. The server waits to terminate until all non-checkpointable, non-rerunnable jobs are finished executing. Any running jobs that can be checkpointed are checkpointed, terminated, and requeued. Jobs that cannot be checkpointed are terminated and requeued if they are rerunnable, otherwise they are allowed to continue to run.

  While terminating, the server is in the *Terminating-Delayed* state.

quick
: Shuts down the primary server. Running jobs and subjobs are left running.

  This is the default behavior when no options are given to the `qterm` command.

  While terminating, the server is in the *Terminating* state.

--version
: The `qterm` command returns its PBS version information and exits. This option can only be used alone.

2.60.4 Operands

You optionally specify the list of servers to shut down using *[<server name>[<server name> ...]]*.

If you do not specify any servers, the `qterm` command shuts down the default server.

2.60.4.1 Standard Error

The `qterm` command writes a diagnostic message to standard error for each error occurrence.

2.60.4.2 Exit Status

Zero
: Upon successful processing of all operands presented to the `qterm` command

Greater than zero
: If the `qterm` command fails to process any operand

2.60.4.3 See Also

The PBS Professional Administrator's Guide, "pbs_server" on page 108, "pbs_sched" on page 105, "pbs_mom" on page 71

2.61 tracejob

Extracts and prints log messages for a PBS job

2.61.1 Synopsis

tracejob [-a] [-c <count>] [-f <filter>] [-l] [-m] [-n <days>] [-p <path>] [-s] [-v] [-w <cols>] [-z] <job ID>

tracejob --version

2.61.2 Description

The `tracejob` command extracts log messages for a given *job ID* and prints them in chronological order.

The `tracejob` command extracts information from server, default scheduler, accounting, and MoM logs. Server logs contain information such as when a job was queued or modified. Scheduler logs contain clues as to why a job is not running. Accounting logs contain accounting records for when a job was queued, started, ended, or deleted. MoM logs contain information about what happened to a job while it was running.

To get MoM log messages for a job, `tracejob` must be run on the machine on which the job ran. If the job ran on multiple hosts, you must run `tracejob` on each of those hosts.

Some log messages appear many times. In order to make the output of `tracejob` more readable, messages that appear over a certain number of times (see option `-c` below) are restricted to only the most recent message.

2.61.3 Using tracejob on Job Arrays

If `tracejob` is run on a job array, the information returned is about the job array itself, and not its subjobs. Job arrays do not have associated MoM log messages. If `tracejob` is run on a subjob, the same types of log messages are available as for a job. Certain log messages that occur for a regular job will not occur for a subjob.

2.61.4 Required Privilege

All users have access to server, scheduler, and MoM information. Only Administrator or root can access accounting information.

2.61.5 Options to `tracejob`

-a

Do not report accounting information.

-c <count>

Set excessive message limit to *count*. If a message is logged at least *count* times, only the most recent message is printed.

The default for *count* is 15.

-f <filter>

Do not include log events of type *filter*. The `-f` option can be used more than once on the command line. The following table shows each filter with its hex value and category:

Table 2-39: tracejob Filters

Filter	Hex Value	Message Category
error	*0x0001*	Internal errors
system	*0x0002*	System errors
admin	*0x0004*	Administrative events
job	*0x0008*	Job-related events
job_usage	*0x0010*	Job accounting info
security	*0x0020*	Security violations
sched	*0x0040*	Scheduler events
debug	*0x0080*	Common debug messages
debug2	*0x0100*	Uncommon debug messages
resv	*0x0200*	Reservation debug messages
debug3	*0x0400*	Less common than *debug2*
debug4	*0x0800*	Less common than *debug3*

-l

Do not report scheduler information.

-m

Do not report MoM information.

-n <days>

Report information from up to *days* days in the past.

Default number of days: *1* = today

-p <path>

Use *path* as path to `PBS_HOME` on machine being queried.

-s

Do not report server information.

-w <cols>

Width of current terminal. If *cols* is not specified, `tracejob` queries OS to get terminal width. If OS doesn't return anything, defaults to *80*.

-v

Verbose. Report more of `tracejob`'s errors than default.

-z

Suppresses printing of duplicate messages.

--version

The `tracejob` command returns its PBS version information and exits. This option can only be used alone.

2.61.6 Operands

The `tracejob` command accepts one *job ID* operand.

For a job, this has the form:

<sequence number>[.<server name>][@<server name>]

For a job array, the form is:

<sequence number>[][.<server name>][@<server name>]

For a subjob, the form is:

<sequence number>[<index>][.<server name>][@<server name>]

Note that some shells require that you enclose a job array identifier in double quotes.

2.61.7 Exit Status

Zero
: Upon successful processing of all options

Greater than zero
: If `tracejob` is unable to process any options

2.61.8 See Also

The PBS Professional Administrator's Guide

2.62 win_postinstall.py

For Windows. Configures PBS MoM or client

2.62.1 Synopsis

<PBS_EXEC>\etc\python win_postinstall.py -u <PBS service account> -p <PBS service account password> -t <installation type> -s <server name> [-c <path to scp command>]

2.62.2 Description

The `win_postinstall.py` command configures the PBS MoM and commands. It performs post-installation steps such as validating the PBS service account username and password, installing the Visual C++ redistributable binary, and creating the `pbs.conf` file in the PBS destination folder.

For an "execution" type of installation, it creates `PBS_HOME`, and registers and starts the PBS_MOM service.

When you use this command during an "execution" type installation of PBS, the command automatically un-registers any old PBS MoM.

Available on Windows only.

2.62.2.1 Required Privilege

You must have Administrator privilege to run this command.

2.62.3 Options to win_postinstall.py

-c, --scp-path <path to scp command>
: Specifies path to `scp` command.

-p, --passwd <PBS service account password>
: Specifies password for PBS service account.

-s, --server <server name>
: Specifies the hostname on which the PBS server will run; required when the installation type is one of "execution" or "client".

-t, --type <installation type>
: Specifies type of installation. Type can be one of "execution" or "client".

-u, --user <PBS service account>
: Specifies PBS service account. When you specify the PBS service account, whether or not you are on a domain machine, include only the username, not the domain. For example, if the full username on a domain machine is *<domain>\<username>*, pass only *<username>* as an argument.

3
MoM Parameters

This chapter describes the configuration files used by MoM and lists the MoM configuration parameters that are found in the Version 1 MoM configuration file, *PBS_HOME/mom_priv/config*.

3.1 Syntax of MoM Configuration File

The Version 1 MoM configuration file contains parameter settings for the MoM on the local host.

Version 1 configuration files list local resources and initialization values for MoM. Local resources are either static, listed by name and value, or externally-provided, listed by name and command path. Local static resources are for use only by the scheduler for MoM's partition. They do not appear in a `pbsnodes -a` query. See the `-c` option to the `pbs_mom` command. Do not change the syntax of the Version 1 configuration file.

Each configuration item is listed on a single line, with its parts separated by white space. Comments begin with a hash-mark ("#").

3.1.1 Externally-provided Resources

Externally-provided resources, for example dynamic resources such as scratch space, use a shell escape to run a command. These resources are described with a name and value, where the first character of the value is an exclamation mark ("!"). The remainder of the value is the path and command to execute.

Parameters in the command beginning with a percent sign ("%") can be replaced when the command is executed. For example, this line in a configuration file describes a resource named "escape":

```
escape    !echo %xxx %yyy
```

If a query for the "escape" resource is sent with no parameter replacements, the command executed is "echo %xxx %yyy". If one parameter replacement is sent, "escape[xxx=hi there]", the command executed is "echo hi there %yyy". If two parameter replacements are sent, "escape[xxx=hi][yyy=there]", the command executed is "echo hi there". If a parameter replacement is sent with no matching token in the command line, "escape[zzz=snafu]", an error is reported.

3.1.2 Windows Notes

If the argument to a MoM option is a pathname containing a space, enclose it in double quotes as in the following:

```
hostn !"\Program Files\PBS\exec\bin\hostn" host
```

When you edit any PBS configuration file, make sure that you put a newline at the end of the file. The Notepad application does not automatically add a newline at the end of a file; you must explicitly add the newline.

3.2 Contents of MoM Configuration File

3.2.1 Replacing Actions

$action <default action> <timeout> <new action>

Replaces the *default action* for an event with the site-specified *new action*. *timeout* is the time allowed for *new action* to run. *new action* is the site-supplied script that replaces *default action*. This is the complete list of values for *default action*:

Table 3-1: How $action is Used

default action	Result
checkpoint	Run *new action* in place of the periodic job checkpoint, after which the job continues to run.
checkpoint_abort	Run *new action* to checkpoint the job, after which the job must be terminated by the script.
multinodebusy <timeout> requeue	Used with cycle harvesting and multi-vnode jobs. Changes default behavior when a vnode becomes busy. Instead of allowing the job to run, the job is requeued. Timeout is ignored. The only *new action* is *requeue*.
restart	Runs *new action* in place of restart.
terminate	Runs *new action* in place of SIGTERM or SIGKILL when MoM terminates a job.

3.2.2 MoM Parameters

$alps_client <path>

Cray only. Path to the Cray `apbasil` command. Must be full path to command.

Format: *path to command*

Default: None

$alps_release_jitter <maximum jitter>

Cray only. PBS sends requests to ALPS to release a finished job at intervals specified by the sum of $alps_release_wait_time and a randomly generated value between zero and *maximum jitter*, in seconds.

Format: *Float*

Default: 0.12 seconds

$alps_release_timeout <timeout>

Cray only. Specifies the amount of time that PBS tries to release an ALPS reservation before giving up. After this amount of time has passed, PBS stops trying to release the ALPS reservation, the job exits, and the job's resources are released. PBS sends a HUP to the MoM so that she rereads the ALPS inventory to get the current available ALPS resources.

We recommend that the value for this parameter be twice the value for suspectbegin.

Format: *Seconds, specified as positive integer*

Default: *600* (10 minutes)

$alps_release_wait_time <wait time>

Cray only. PBS sends requests to ALPS to release a finished job at intervals specified by the sum of *wait time* and a randomly generated value between zero and the maximum specified in $alps_release_jitter, in seconds.

Format: *Float*

Default: 0.4 seconds

$checkpoint_path <path>

MoM passes this parameter to the checkpoint and restart scripts. This path can be absolute or relative to `PBS_HOME/mom_priv`. Overrides default. Overridden by path specified in the `pbs_mom -C` option and by PBS_CHECKPOINT_PATH environment variable. See "Specifying Checkpoint Path" on page 428 in the PBS Professional Administrator's Guide.

$clienthost <hostname>

hostname is added to the list of hosts which are allowed to connect to MoM as long as they are using a privileged port. For example, this allows the hosts "fred" and "wilma" to connect to MoM:

```
$clienthost fred
$clienthost wilma
```

The following hostnames are added to $clienthost automatically: the server, the localhost, and if configured, the secondary server. The server sends each MoM a list of the hosts in the nodes file, and these are added internally to *$clienthost*. None of these hostnames need to be listed in the configuration file.

Two hostnames are always allowed to connect to `pbs_mom`, "`localhost`" and the name returned to MoM by the system call `gethostname()`. These hostnames do not need to be added to the MoM configuration file.

The hosts listed as "clienthosts" make up a "sisterhood" of machines. Any one of the sisterhood will accept connections from within the sisterhood. The sisterhood must all use the same port number.

$cputmult <factor>

This sets a factor used to adjust CPU time used by each job. This allows adjustment of time charged and limits enforced where jobs run on a system with different CPU performance. If MoM's system is faster than the reference system, set factor to a decimal value greater than 1.0. For example:

```
$cputmult 1.5
```

If MoM's system is slower, set factor to a value between *1.0* and *0.0*. For example:

```
$cputmult 0.75
```

$dce_refresh_delta <delta>

Obsolete (2020.1)

Defines the number of seconds between successive refreshings of a job's DCE login context. For example:

```
$dce_refresh_delta 18000
```

$enforce <limit>

MoM will enforce the given *limit*. Some limits have associated values. Syntax:

$enforce <variable name> <value>

$enforce mem

MoM will enforce each job's memory limit.

$enforce cpuaverage

MoM will enforce ncpus when the average CPU usage over a job's lifetime usage is greater than the job's limit.

$enforce average_trialperiod <seconds>

Modifies cpuaverage. Minimum number of seconds of job walltime before enforcement begins.

Format: *Integer*

Default: *120*

$enforce average_percent_over <percentage>
Modifies cpuaverage. Gives percentage by which a job may exceed its ncpus limit.

Format: *Integer*

Default: *50*

$enforce average_cpufactor <factor>
Modifies cpuaverage. The ncpus limit is multiplied by *factor* to produce actual limit.

Format: *Float*

Default: *1.025*

$enforce cpuburst
MoM will enforce the ncpus limit when CPU burst usage exceeds the job's limit.

$enforce delta_percent_over <percentage>
Modifies cpuburst. Gives percentage over limit to be allowed.

Format: *Integer*

Default: *50*

$enforce delta_cpufactor <factor>
Modifies cpuburst. The ncpus limit is multiplied by *factor* to produce actual limit.

Format: *Float*

Default: *1.5*

$enforce delta_weightup <factor>
Modifies cpuburst. Weighting factor for smoothing burst usage when average is increasing.

Format: *Float*

Default: *0.4*

$enforce delta_weightdown <factor>
Modifies cpuburst. Weighting factor for smoothing burst usage when average is decreasing.

Format: *Float*

Default: *0.4*

$ideal_load <load>
Defines the *load* below which the vnode is not considered to be busy. Used with the $max_load directive.

Example:

```
$ideal_load 1.8
```

Format: *Float*

No default

$jobdir_root <stage directory root>
Directory under which PBS creates job-specific staging and execution directories. PBS creates a job's staging and execution directory when the job's sandbox attribute is set to *PRIVATE*. If $jobdir_root is unset, it defaults to the job owner's home directory. In this case the user's home directory must exist. If *stage directory root* does not exist when MoM starts up, MoM will abort. If *stage directory root* does not exist when MoM tries to run a job, MoM will kill the job. Path must be owned by root, and permissions must be *1777*. On Windows, this directory should have *Full Control Permission* for the local Administrators group.

When you set $jobdir_root to a shared directory, tell MoM it is shared by setting the *shared* directive after the directory name:

$jobdir_root <directory name> shared

Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

Example of a shared directory:

```
$jobdir_root /r/shared shared
```

Example of a non-shared directory:

```
$jobdir_root /scratch/foo
```

$job_launch_delay

When the primary MoM gets a job whose tolerate_node_failures attribute is set to *all* or *job_start*, the primary MoM can wait to start the job (running the job script or executable) for up to a configured number of seconds. During this time, execjob_prologue hooks can finish and the primary MoM can check for communication problems with sister MoMs. You configure the number of seconds for the primary MoM to wait for hooks via the job_launch_delay configuration parameter in MoM's config file:

```
$job_launch_delay <number of seconds to wait>
```

Default: the sum of the values of the alarm attributes of any enabled execjob_prologue hooks. If there are no enabled execjob_prologue hooks, the default value is 30 seconds. For example, if there are two enabled execjob_prologue hooks, one with alarm = 30 and one with alarm = 60, the default value of MoM's job_launch_delay is 90 seconds.

After all the execjob_prologue hooks have finished, or MoM has waited for the value of the job_launch_delay parameter, she starts the job.

$kbd_idle <idle wait> <min use> <poll interval>

Declares that the vnode will be used for batch jobs during periods when the keyboard and mouse are not in use.

idle wait

Time, in seconds, that the workstation keyboard and mouse must be idle before being considered available for batch jobs.

Must be set to non-zero value for cycle harvesting to be enabled.

Format: *Integer*

No default

min use

Time, in seconds, during which the workstation keyboard or mouse must continue to be in use before the workstation is determined to be unavailable for batch jobs.

Format: *Integer*

Default: *10*

poll interval

Interval, in seconds, at which MoM checks for keyboard and mouse activity.

Format: *Integer*

Default: *1*

Example:

```
$kbd_idle 1800 10 5
```

$logevent <mask>

Sets the mask that determines which event types are logged by pbs_mom. To include all debug events, use *0xffffffff*. See "Log Levels" on page 547 in the PBS Professional Administrator's Guide.

Default: 975

$max_check_poll <seconds>

Maximum time between polling cycles, in seconds. See "Configuring MoM Polling Cycle" on page 34 in the PBS Professional Administrator's Guide. Minimum recommended value: 30 seconds.

Minimum value: *1 second*

Default: *120 seconds*

Format: *Integer*

$max_load <load> [suspend]

Defines the load above which the vnode is considered to be *busy*. Used with the $ideal_load directive. No new jobs are started on a *busy* vnode.

The optional *suspend* directive tells PBS to suspend jobs running on the vnode if the load average exceeds the $max_load number, regardless of the source of the load (PBS and/or logged-in users). Without this directive, PBS will not suspend jobs due to load.

We recommend setting *load* to a value that is slightly higher than the number of CPUs, for example *.25 + ncpus*.

Example:

```
$max_load 3.5
```

Format: *Float*

Default: number of CPUs on machine

$max_poll_downtime <downtime>

When mother superior detects that a sister mom has lost connectivity (e.g. MoM went down or the network is having problems) it waits *downtime* seconds for the sister to reconnect before it gives up and kills the job.

Format: *Integer*

Default: *five minutes*

memreserved <megabytes>

Deprecated. The amount of per-vnode memory reserved for system overhead. This much memory is deducted from the value of resources_available.mem for each vnode managed by this MoM.

For example,

```
memreserved 16
```

Default: *0MB*

$min_check_poll <seconds>

Minimum time between polling cycles, in seconds. Must be greater than zero and less than $max_check_poll. See "Configuring MoM Polling Cycle" on page 34 in the PBS Professional Administrator's Guide. Minimum recommended value: *10 seconds*.

Format: *Integer*

Minimum value: *1 second*

Default: *10 seconds*

pbs_accounting_workload_mgmt <value>

Controls whether CSA accounting is enabled. Name does not start with dollar sign. If set to "*1*", "*on*", or "*true*", CSA accounting is enabled. If set to "*0*", "*off*", or "*false*", accounting is disabled. Cray only. Requires CLE 5.2.

Default: "*true*"; enabled

$prologalarm <timeout>

Defines the maximum number of seconds the prologue and epilogue may run before timing out.

Example:

```
$prologalarm 30
```

Format: *Integer*

Default: *30 seconds*

$reject_root_scripts <True|False>

When set to *True*, MoM won't acquire any new hook scripts, and MoM won't run job scripts that would execute as root or Admin. However, MoM will run previously-acquired hooks that run as root.

Format: *Boolean*

Default: *False*

$restart_background <True|False>

Controls how MoM runs a restart script after checkpointing a job. When this option is set to *True*, MoM forks a child which runs the restart script. The child returns when all restarts for all the local tasks of the job are done. MoM does not block on the restart. When this option is set to *False*, MoM runs the restart script and waits for the result.

Format: *Boolean*

Default: *False*

$restart_transmogrify <True | False>

Controls how MoM runs a restart script after checkpointing a job.

When this option is set to *True*, MoM runs the restart script, replacing the session ID of the original task's top process with the session ID of the script.

When this option is set to *False*, MoM runs the restart script and waits for the result. The restart script must restore the original session ID for all the processes of each task so that MoM can continue to track the job.

When this option is set to *False* and the restart uses an external command, the configuration parameter restart_background is ignored and treated as if it were set to *True*, preventing MoM from blocking on the restart.

Format: *Boolean*

Default: *False*

$restrict_user <True | False>

Controls whether users not submitting jobs have access to this machine. If value is *True*, restrictions are applied.

See $restrict_user_exceptions and $restrict_user_maxsysid.

Not supported on Windows.

Format: *Boolean*

Default: *False*

$restrict_user_exceptions <user list>

Comma-separated list of users who are exempt from access restrictions applied by $restrict_user. Leading spaces within each entry are allowed. Maximum of 10 names.

$restrict_user_maxsysid <value>

Any user with a numeric user ID less than or equal to *value* is exempt from restrictions applied by $restrict_user.

If $restrict_user is *True* and no *value* exists for $restrict_user_maxsysid, PBS looks in `/etc/login.defs`, if it exists, for the *value*. Otherwise the default is used.

Format: *Integer*

Default: *999*

$restricted <hostname>

The *hostname* is added to the list of hosts which are allowed to connect to MoM without being required to use a privileged port. Queries from the hosts in the restricted list are only allowed access to information internal to this host, such as load average, memory available, etc. They may not run shell commands.

Hostnames can be wildcarded. For example, to allow queries from any host from the domain "xyz.com":

```
$restricted *.xyz.com
```

$sister_join_job_alarm

When the primary MoM gets a job whose tolerate_node_failures attribute is set to *all* or *job_start*, the primary MoM can wait to start the job for up to a configured number of seconds if the sister MoMs do not immediately acknowledge joining the job. This gives the sister MoMs more time to join the job. You configure the number of seconds for the primary MoM to wait for sister MoMs via the sister_join_job_alarm configuration parameter in MoM's config file:

```
$sister_join_job_alarm <number of seconds to wait>
```

Default: the sum of the values of the alarm attributes of any enabled execjob_begin hooks. If there are no enabled execjob_begin hooks, the default value is 30 seconds. For example, if there are two enabled execjob_begin hooks, one with alarm = 30 and one with alarm = 20, the default value of MoM's sister_join_job_alarm is 50 seconds.

After all the sister MoMs have joined the job, or MoM has waited for the value of the $sister_join_job_alarm parameter, she starts the job.

$suspendsig <suspend signal> [resume signal]

Alternate signal *suspend signal* is used to suspend jobs instead of SIGSTOP. Optional *resume signal* is used to resume jobs instead of SIGCONT.

$tmpdir <directory>

Location where each job's scratch directory will be created.

PBS creates a temporary directory for use by the job, not by PBS. PBS creates the directory before the job is run and removes the directory and its contents when the job is finished. It is scratch space for use by the job. Permission must be 1777 on Linux, writable by *Everyone* on Windows.

Example:

```
$tmpdir /memfs
```

Default on Linux: `/var/tmp`

Default on Windows: value of the TMP environment variable

$usecp <hostname:source directory> <destination directory>

MoM uses `/bin/cp` to deliver output files when the destination is a network mounted file system, or when the source and destination are both on the local host, or when the *source directory* can be replaced with the *destination directory* on *hostname*. Both *source directory* and *destination directory* are absolute pathnames of directories, not files.

Overrides PBS_RCP and PBS_SCP.

Use trailing slashes on both the source and destination. For example:

```
$usecp HostA:/users/work/myproj/ /sharedwork/proj_results/
```

$vnodedef_additive

Specifies whether MoM considers a vnode that appeared previously either in the inventory or in a vnode definition file, but that does not appear now, to be in her list of vnodes.

When $vnodedef_additive is *True*, MoM treats missing vnodes as if they are still present, and continues to report them as if they are present. This means that the server does not mark missing vnodes as *stale*.

When $vnodedef_additive is *False*, MoM does not list missing vnodes, the server's information is brought up to date with the inventory and vnode definition files, and the server marks missing vnodes as *stale*.

PBS automatically sets the value of the $vnodedef_additive MoM configuration option to *False* on any MoM on a login node.

Visible in configuration file on Cray only.

Format: *Boolean*

Default for MoM on Cray login node: *False*

$wallmult <factor>

Each job's walltime usage is multiplied by *factor*. For example:

```
$wallmult 1.5
```

3.2.3 Static MoM Resources

Static resources local to the vnode are described one resource to a line, with a name and value separated by white space. For example, tape drives of different types could be specified by:

```
tape3480 4
tape3420 2
tapedat 1
tape8mm 1
```

4
Scheduler Parameters

This chapter lists scheduler configuration parameters. These parameters are found in each scheduler's configuration file, *PBS_HOME/sched_priv/sched_config*.

4.1 Format of Scheduler Configuration File

4.1.1 Parameters with Separate Primetime and Non-primetime Specification

If a scheduler parameter can be specified separately for primetime and non-primetime, the format for the parameter is the following:

name: value [prime | non_prime | all | none]

- The *name* field cannot contain any whitespace.
- The *value* field may contain whitespace if the string is double-quoted. *value* can be: *True* | *False* | <number> | <string>. "*True*" and "*False*" are not case-sensitive.
- The third field allows you to specify that the setting is to apply during primetime, non-primetime, all the time, or none of the time. A blank third field is equivalent to "*all*" which means that it applies to both primetime and non-primetime.

  Acceptable values: "*all*", "*ALL*", "*none*", "*NONE*", "*prime*", "*PRIME*", "*non_prime*", "*NON_PRIME*"

4.1.2 Parameters without Separate Primetime and Non-primetime Specification

If a scheduler parameter cannot be specified separately for primetime and non-primetime, the format for the parameter is the same as the above, except that there is no third field.

4.1.3 Format Details

- Each entry must be a single, unbroken line.
- Entries must be quoted if they contain whitespace.
- Any line starting with a "#" is a comment, and is ignored.

4.2 Configuration Parameters

backfill

Deprecated. Use the backfill_depth queue/server attribute instead. Toggle that controls whether PBS uses backfilling. If this is set to *True*, this scheduler attempts to schedule smaller jobs around higher-priority jobs when using strict_ordering, as long as running the smaller jobs won't change the start time of the jobs they were scheduled around. This scheduler chooses jobs in the standard order, so other high-priority jobs will be considered first in the set to fit around the highest-priority job.

When this parameter is *True* and help_starving_jobs is *True*, this scheduler backfills around starving jobs.

Can be used with strict_ordering and help_starving_jobs

Format: *Boolean*

Default: *True all*

backfill_prime

This scheduler will not run jobs which would overlap the boundary between primetime and non-primetime. This assures that jobs restricted to running in either primetime or non-primetime can start as soon as the time boundary happens.

See also prime_spill, prime_exempt_anytime_queues.

Format: *Boolean*

Default: *False all*

by_queue

If set to *True*, all jobs that can be run from the highest-priority queue are run, then any jobs that can be run from the next queue are run, and so on. Queues are ordered highest-priority first. If by_queue is set to *False*, all jobs are treated as if they are in one large queue. The by_queue parameter is overridden by the round_robin parameter when round_robin is set to *True*.

See "Examining Jobs Queue by Queue" on page 110 in the PBS Professional Administrator's Guide.

Format: *Boolean*

Default: *True all*

cpus_per_ssinode

Obsolete.

dedicated_prefix

Queue names with this prefix are treated as dedicated queues, meaning jobs in that queue are considered for execution only when the system is in dedicated time as specified in the configuration file `PBS_HOME/sched_priv/dedicated_time`.

See "Dedicated Time" on page 125 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *ded*

fair_share

Enables the fairshare algorithm, and turns on usage collecting. Jobs will be selected based on a function of their recent usage and priority (shares). Not a prime option.

See "Using Fairshare" on page 138 in the PBS Professional Administrator's Guide.

Format: *Boolean*

Default: *False all*

fairshare_decay_factor

Decay multiplier for fairshare usage reduction. Each decay period, the usage is multiplied by this value. Valid values: between 0 and 1, not inclusive. Not a prime option.

Format: *Float*

Default: 0.5

fairshare_decay_time

Time between fairshare usage decay operations. Not a prime option.

Format: *Duration*

Default: *24:00:00*

fairshare_entity

Specifies the entity for which fairshare usage data will be collected. Can be one of "*euser*", "*egroup*", "*Account_Name*", "*queue*", or "*egroup:euser*". Not a prime option.

Format: *String*

Default: *euser*

fairshare_enforce_no_shares

If this option is enabled, jobs whose entity has zero shares will never run. Requires fair_share parameter to be enabled. Not a prime option.

Format: *Boolean*

Default: *False*

fairshare_usage_res

Specifies the mathematical formula to use in fairshare calculations. Is composed of PBS resources as well as mathematical operators that are standard Python operators and/or those in the Python math module. When using a PBS resource, if resources_used.<resource name> exists, that value is used. Otherwise, the value is taken from Resource_List.<resource name>. Not a prime option.

See "Tracking Resource Usage" on page 141 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *"cput"*

half_life

Deprecated (as of 13.0).

The half-life for fairshare usage; after the amount of time specified, the fairshare usage is halved. Requires that fair_share parameter be enabled. Not a prime option.

See "Using Fairshare" on page 138 in the PBS Professional Administrator's Guide.

Format: *Duration*

Default: *24:00:00*

help_starving_jobs

Setting this option enables starving job support. Once jobs have waited for the amount of time given by max_starve they are considered starving. If a job is considered starving, no lower-priority jobs will run until the starving job can be run, unless backfilling is also used. To use this option, the max_starve configuration parameter needs to be set as well. See also max_starve, and the server's backfill_depth and eligible_time_enable attributes.

At each scheduler iteration, PBS calculates estimated.start_time and estimated.exec_vnode for starving jobs being backfilled around.

Format: *Boolean*

Default: *True all*

job_sort_key

Specifies how jobs should be sorted. job_sort_key can be used to sort using either (a) resources or (b) special case sorting routines. Multiple job_sort_key entries can be used, one to a line, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc. This attribute is overridden by the job_sort_formula attribute. If both are set, job_sort_key is ignored and an error message is printed.

Syntax:

job_sort_key: "<resource name> HIGH|LOW"

job_sort_key: "fairshare_perc HIGH|LOW"

job_sort_key: "job_priority HIGH|LOW"

Options: One of the following is required.

HIGH
Specifies descending sort.

LOW
Specifies ascending sort.

There are three special case sorting routines, which can be used instead of *resource name*:

Table 4-1: Special Sorting in job_sort_key

Special Sort	Description
fairshare_perc HIGH	Sort based on how much fairshare percentage the entity deserves, based on the values in the `resource_group` file. If user A has more priority than user B, all of user A's jobs will always be run first. Past history is not used. For calculation, see "Computing Target Usage for Each Vertex (fairshare_perc)" on page 143 in the PBS Professional Administrator's Guide. This should only be used if entity share (strict priority) sorting is needed. See "Sorting Jobs by Entity Shares (Was Strict Priority)" on page 131 in the PBS Professional Administrator's Guide Incompatible with fair_share scheduling parameter being *True*.
job_priority HIGH|LOW	Sort jobs by the job priority attribute regardless of job owner.
sort_priority HIGH|LOW	**Deprecated**. See job_priority above.

The following example illustrates how to sort jobs so that those with high CPU count come first:

```
job_sort_key: "ncpus HIGH" all
```

The following example shows how to sort jobs so that those with lower memory come first:

```
job_sort_key: "mem LOW" prime
```

Format: *Quoted string*

Default: Not enforced

load_balancing

Deprecated (2020.1).

When set to *True*, this scheduler takes into account the load average on vnodes as well as the resources listed in the `resources` line in `sched_config`. Load balancing can result in overloaded CPUs.

See "Using Load Balancing" on page 156 in the PBS Professional Administrator's Guide.

Format: *Boolean*

Default: *False all*

load_balancing_rr

Deprecated. To duplicate this setting, enable load_balancing and set smp_cluster_dist to *round_robin*.

See "Using Load Balancing" on page 156 in the PBS Professional Administrator's Guide.

log_filter

Obsolete. See "log_events" on page 298 of the PBS Professional Reference Guide.

max_starve

The amount of time before a job is considered starving. This variable is used only if help_starving_jobs is set.

Upper limit: None

Format: *Duration*

Default: *24:00:00*

mem_per_ssinode

Obsolete.

mom_resources

Deprecated as of 18.2.1.

This option is used to query the MoMs to set the value of resources_available.<resource name> where *resource name* is a site-defined resource. Each MoM is queried with the resource name and the return value is used to replace resources_available.<resource name> on that vnode. On a multi-vnoded machine with a parent vnode, all vnodes share anything set in mom_resources.

Format: *String*

Default: Unset

node_sort_key

Defines sorting on resource or priority values on vnodes. Resource must be numerical, for example, *long* or *float*. Up to 20 node_sort_key entries can be used, in which case the first entry will be the primary sort key, the second will be used to sort equivalent items from the first sort, etc.

Syntax:

node_sort_key: <resource name> | sort_priority <HIGH | LOW>

node_sort_key: <resource name> <HIGH | LOW> <total | assigned | unused>

where

total

Use the resources_available value. This is the default setting when sorting on a resource.

assigned

Use the resources_assigned value.

unused

Use the value given by resources_available - resources_assigned.

sort_priority

Sort vnodes by the value of the vnode priority attribute.

When sorting on a resource, the default third field is "*total*".

See "Sorting Vnodes on a Key" on page 226 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *node_sort_key: "sort_priority HIGH all"*

nonprimetime_prefix

Queue names which start with this prefix are treated as non-primetime queues. Jobs in these queues run only during non-primetime. Primetime and non-primetime are defined in the `holidays` file.

See "Using Primetime and Holidays" on page 191 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *np_*

peer_queue

Defines the mapping of a pulling queue to a furnishing queue for peer scheduling. Maximum number is 50 peer queues per scheduler.

See "Peer Scheduling" on page 165 in the PBS Professional Administrator's Guide.

Format: *String*

Default: Unset

preemptive_sched

Enables job preemption.

See preempt_order and "Using Preemption" on page 180 in the PBS Professional Administrator's Guide for details.

Format: *String*

Default: *True all*

preempt_order

No longer available. Use the preempt_sort scheduler attribute. See "preempt_order" on page 299..

preempt_prio

No longer available. Use the preempt_sort scheduler attribute. See "preempt_prio" on page 300..

preempt_queue_prio

No longer available. Use the preempt_sort scheduler attribute. See "preempt_queue_prio" on page 300..

preempt_sort

No longer available. Use the preempt_sort scheduler attribute. See "preempt_sort" on page 300..

primetime_prefix

Queue names starting with this prefix are treated as primetime queues. Jobs in these queues run only during primetime. Primetime and non-primetime are defined in the `holidays` file.

See "Using Primetime and Holidays" on page 191 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *p_*

prime_exempt_anytime_queues

Determines whether *anytime* queues are controlled by backfill_prime.

If set to *True*, jobs in an *anytime* queue are not prevented from running across a primetime/non-primetime or non-primetime/primetime boundary.

If set to *False*, the jobs in an *anytime* queue may not cross this boundary, except for the amount specified by their prime_spill setting.

See also backfill_prime, prime_spill.

Format: *Boolean*

Default: *False*

prime_spill

Specifies the amount of time a job can spill over from non-primetime into primetime or from primetime into non-primetime. This option can be separately specified for primetime and non-primetime. This option is only meaningful if backfill_prime is *True*.

See also backfill_prime, prime_exempt_anytime_queues.

For example, non-primetime jobs can spill into primetime by 1 hour:

```
prime_spill: 1:00:00 prime
```

For example, jobs in either prime/non-prime can spill into the other by 1 hour:

```
prime_spill: 1:00:00 all
```

Format: *Duration*

Default: *00:00:00*

provision_policy

Specifies how vnodes are selected for provisioning. Can be set by Manager only; readable by all. Can be set to one of the following:

avoid_provision

PBS first tries to satisfy the job's request from free vnodes that already have the requested AOE instantiated. PBS uses node_sort_key to sort these vnodes.

If PBS cannot satisfy the job's request using vnodes that already have the requested AOE instantiated, PBS uses the server's node_sort_key to select the free vnodes that must be provisioned in order to run the job, choosing from any free vnodes, regardless of which AOE is instantiated on them.

Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

aggressive_provision

PBS selects vnodes to be provisioned without considering which AOE is currently instantiated.

PBS uses the server's node_sort_key to select the vnodes on which to run the job, choosing from any free vnodes, regardless of which AOE is instantiated on them. Of the selected vnodes, PBS provisions any that do not have the requested AOE instantiated on them.

Format: *String*

Default: *aggressive_provision*

resources

Specifies those resources which are not to be over-allocated, or if Boolean are to be honored, when scheduling jobs. Vnode-level Boolean resources are automatically honored and do not need to be listed here. Limits are set by setting resources_available.<resource name> on vnodes, queues, and the server. A scheduler considers numeric (integer or float) items as consumable resources and ensures that no more are assigned than are available (e.g. ncpus or mem). Any string resources are compared using string comparisons. If "host" is not added to the resources line, when the user submits a job requesting a specific vnode in the following syntax:

```
qsub -l select=host=vnodeName
```

the job will run on any host.

Format: *String*

Default: *ncpus, mem, arch, host, vnode*, *aoe*

resource_unset_infinite

Resources in this list are treated as infinite if they are unset. Cannot be set differently for primetime and non-primetime.

Example:

```
resource_unset_infinite: "vmem, foo_licenses"
```

Format: *Comma-delimited list of resources*

Default: Empty list

round_robin

If set to *True*, this scheduler considers one job from the first queue, then one job from the second queue, and so on in a circular fashion. The queues are ordered with the highest-priority queue first. Each scheduling cycle starts with the same highest-priority queue, which will therefore get preferential treatment.

If there are groups of queues with the same priority, and this parameter is set to *True*, this scheduler round-robins through each group of queues before moving to the next group.

If round_robin is set to *False*, this scheduler considers jobs according to the setting of the by_queue parameter.

When *True*, overrides the by_queue parameter.

Format: *Boolean*

Default: *False all*

server_dyn_res

Directs this scheduler to replace the server's resources_available values with new values returned by a site-specific external script or program.

See "Dynamic Server-level Resources" on page 267 in the PBS Professional Administrator's Guide for details of usage.

Default timeout for server dynamic resource scripts is 30 seconds. You can configure this in the scheduler server_dyn_res_alarm attribute.

Format: *String*

Default: Unset

smp_cluster_dist

Deprecated (12.2). Specifies how single-host jobs should be distributed to all hosts of the complex.

Options:

pack
: Keep putting jobs onto one host until it is full and then move on to the next.

round_robin
: Put one job on each vnode in turn before cycling back to the first one.

lowest_load
Put the job on the lowest-loaded host.

See "SMP Cluster Distribution" on page 218 in the PBS Professional Administrator's Guide and "Using Load Balancing" on page 156 in the PBS Professional Administrator's Guide.

Format: *String*

Default: *pack all*

sort_queues

Obsolete

strict_fifo

Deprecated. Use strict_ordering.

strict_ordering

Specifies that jobs must be run in the order determined by whatever sorting parameters are being used. This means that a job cannot be skipped due to resources required not being available. If a job due to run next cannot run, no job will run, unless backfilling is used, in which case jobs can be backfilled around the job that is due to run next.

See "FIFO with Strict Ordering" on page 149 in the PBS Professional Administrator's Guide.

Example line in `PBS_HOME/sched_priv/sched_config`:

```
strict_ordering: True ALL
```

Format: *Boolean*

Default: *False all*

sync_time

Removed. Do not use; this will cause an error.

unknown_shares

The number of shares for the *unknown* group. These shares determine the portion of a resource to be allotted to that group via fairshare. Requires fair_share to be enabled.

See "Using Fairshare" on page 138 in the PBS Professional Administrator's Guide.

Format: *Integer*

Default: The unknown group gets 0 shares

5 List of Built-in Resources

This chapter lists all of the built-in PBS resources. For information on setting, viewing, and using resources, see "Using PBS Resources" on page 229 in the PBS Professional Administrator's Guide.

5.1 Resource Data Types

Data types for resources are described in section 7.2, "Resource Formats", on page 359.

Boolean

Name of Boolean resource is a string.

Values:

TRUE, True, true, T, t, Y, y, 1

FALSE, False, false, F, f, N, n, 0

Duration

A period of time, expressed either as

An integer whose units are seconds

or

[[hours:]minutes:]seconds[.milliseconds]

in the form:

[[HH:]MM:]SS[.milliseconds]

Milliseconds are rounded to the nearest second.

Float

Floating point. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

Long

Long integer. Allowable values: 0-9 [[0-9] ...], and + and -

<queue name>@<server name>

Size

Number of bytes or words. The size of a word is 64 bits.

Format: *<integer>[<suffix>]*

where *suffix* can be one of the following:

Table 5-1: Size in Bytes

Suffix	Meaning	Size
b or w	Bytes or words	1
kb or kw	Kilobytes or kilowords	2 to the 10th, or 1024
mb or mw	Megabytes or megawords	2 to the 20th, or 1,048,576
gb or gw	Gigabytes or gigawords	2 to the 30th, or 1,073,741,824
tb or tw	Terabytes or terawords	2 to the 40th, or 1024 gigabytes
pb or pw	Petabytes or petawords	2 to the 50th, or 1,048,576 gigabytes

Default: *bytes*

Note that a scheduler rounds all resources of type size up to the nearest kb.

String

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~] ...]

String resource values are case-sensitive. No limit on length.

String Array

Comma-separated list of strings.

Strings in string_array may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

5.2 Viewing Resource Information

You can see attribute values of resources for the server, queues, and vnodes using the `qmgr` or `pbsnodes` commands. The value in the server, queue, or vnode resources_assigned attribute is the amount explicitly requested by running and exiting jobs and, at the server and vnodes, started reservations.

You can see job attribute values using the `qstat` command. The value in the job's Resource_List attribute is the amount explicitly requested by the job. See "Resources Requested by Job" on page 245 in the PBS Professional Administrator's Guide.

The following table summarizes how to find resource information:

Table 5-2: How to View Resource Information

Location	Item to View	Command
server	default_chunk, default_qsub_arguments, resources_available, resources_assigned, resources_default	qmgr, qstat, pbsnodes
scheduler	`sched_config file`	Favorite editor or viewer
queues	default_chunk, resources_available, resources_assigned, resources_default	qmgr, qstat
MoM and vnodes	resources_available, sharing, pcpus, resources_assigned	qmgr, pbsnodes
	`mom_config file`	Favorite editor or viewer
job	Resource_List	qstat
reservation	Resource_List	pbs_rstat -f
accounting	`resources_assigned` entry in accounting log	Favorite editor or viewer

Every consumable resource, for example mem, can appear in four PBS attributes. These attributes are used in the following elements of PBS:

Table 5-3: Values Associated with Consumable Resources

Attribute	Vnode	Queue	Server	Accounting Log	Job	Scheduler
resources_available	Yes	Yes	Yes			Yes
resources_assigned	Yes	Yes	Yes	Yes		
resources_used				Yes	Yes	Yes
Resource_List					Yes	Yes

5.3 Resource Flags

Resource flags are described and listed in "Resource Flags" on page 258 in the PBS Professional Administrator's Guide.

5.4 Attributes where Resources Are Tracked

Resources are tracked in the following attributes:

Table 5-4: Attributes Where Resources Are Tracked

Resource Being Tracked	Attribute Name			
	Server and Queue	Vnode	Job	Reservation
Amount of each resource available for use at the object (server, queue, vnode)	resources_available.<resource name>	resources_available.<resource name>		
Amount of each resource allocated to jobs running and exiting at the object (server, queue, vnode)	resources_assigned.<resource name>	resources_assigned.<resource name>		
Amount of each resource used by the job			resources_used.<resource name>	
Amount of each job-wide resource that is assigned to any job that does not explicitly request the resource	resources_default.<resource name>			
Amount of each host-level resource that is assigned to each chunk of any job where that does not explicitly request the resource	default_chunk.<resource name>			

Table 5-4: Attributes Where Resources Are Tracked

Resource Being Tracked	Attribute Name			
	Server and Queue	Vnode	Job	Reservation
List of resources requested by the object (job or reservation)			Resource_List.<resource name>	Resource_List.<resource name>
List of chunks for the job. Each chunk shows the name of the vnode from which it is taken along with the host-level, consumable resources allocated from that vnode.			exec_vnode	
List of vnodes and resources allocated to them to satisfy the chunks requested for this reservation or occurrence				resv_nodes

5.5 Resource Table Format

In the following tables, the columns contain the following information:

Name
: The name of the resource

Description
: A description of the resource's function

Format
: The resource's format

Scope

Some resources are either:

- Job-wide and can be requested only outside of a select statement
- Host-level and can be requested only inside of a select statement

Consumable

A resource is consumable if use of this resource by a job reduces the amount available to other jobs

Val/Opt

If the resource can take only specific values or options, each is listed here

Value/Option Description

If the resource can take only specific values or options, the behavior of each value or option is described here

Default Value

The resource's default value, if any

Python Type

The resource's Python type

Platform

Platform where available

5.6 Resources Built Into PBS

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
accelerator Indicates whether this vnode is associated with an accelerator. Used for requesting accelerators. On Cray XC, this resource exists only when there is at least one associated accelerator.	*Boolean*	Host-level	No	*True*	On Cray XC, this is set to *True* when there is at least one associated accelerator whose state is *UP*.	*False*	bool	
				False	On Cray XC, set to *False* when all associated accelerators are in state *DOWN*.			

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
accelerator_memory Indicates amount of memory for accelerator(s) associated with this vnode. On Cray XC, PBS sets this resource only on vnodes with at least one accelerator with state = *UP*. For Cray XC, PBS sets this resource on the 0th NUMA node (the vnode with PBScrayseg=*0*), and the resource is shared by other vnodes on the compute node. For example, on vnodeA_2_0: resources_available.accelerator_memory=4196mb On vnodeA_2_1: resources_available.accelerator_memory=@vnodeA_2_0 A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Host-level	Yes			No default	pbs.size	
accelerator_model Indicates model of the accelerator(s) associated with this vnode. On Cray XC, PBS sets this resource only on vnodes with at least one accelerator with state = *UP*.	*String*	Host-level	No			No default	str	
aoe List of AOEs (Application Operating Environments) that can be instantiated on this vnode. Case-sensitive. An AOE is the environment that results from provisioning a vnode. Each job can request at most one AOE. Cannot be set on server's host.	*string array*	Host-level	No	Allowable values are site-dependent.		No default	str	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
arch System architecture. One architecture can be defined for a vnode. One architecture can be requested per vnode. Allowable values and effect on job placement are site-dependent. The resources_available.arch resource is the value reported by MoM unless explicitly set by the administrator.	*String*	Host-level	No	*linux*	Linux	No default	str	Linux
				linux_cpuset	Linux with cpusets			Linux with cpusets
				XT	CLE			CLE
				windows	Windows			Windows
cput Amount of CPU time used by the job for all processes on all vnodes. Establishes a job-wide resource limit.	*Duration*	Job-wide	No			No default	pbs.duration	
energy The energy used by a job. Set by PBS.	*Float.* Units: *kWh*		Yes			No default		
eoe Stands for "Energy Operational Environment". When set on a vnode in resources_available.eoe, contains the list of available power profiles. When set for a job in Resource_List.eoe, can contain at most one power profile. (A job can request only one power profile.) Automatically added to `resources:` line in `sched_config`.	*string array*		No			For resources_available.eoe: *unset* For Resource_List.eoe: no default	str	
exec_vnode The vnodes that PBS estimates this job will use. This is not the job's exec_vnode attribute. This appears only in job's estimated attribute. Cannot be requested for a job; used for reporting only. Read-only.	*String*					No default	str	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
file Size of any single file that may be created by the job. A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Job-wide				No default	pbs.size	
hbmem High-bandwidth memory. Available only on some architectures such as Xeon Phi KNL.	*Size*	Host-level	Yes		Values must be greater than or equal to zero.	No default	pbs.size	Xeon Phi KNL
host Name of execution host. Site-dependent.	*String*	Host-level				Automatically set to the short form of the hostname in the Mom attribute. On Cray XC compute node, set to *<mpp_host>_<nid>*.	str	
max_walltime Maximum walltime allowed for a shrink-to-fit job. Job's actual walltime is between max_walltime and min_walltime. PBS sets walltime for a shrink-to-fit job. If max_walltime is specified, min_walltime must also be specified. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations.	*Duration*	Job-wide	No	Must be greater than or equal to min_walltime.		*5 years*	pbs.duration	
mem Amount of physical memory i.e. workingset allocated to the job, either job-wide or host-level. A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Either job-wide or host-level. Can be requested only inside of a select statement.	Yes			No default	pbs.size	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
min_walltime Minimum walltime allowed for a shrink-to-fit job. When min_walltime is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's walltime. Job's actual walltime is between max_walltime and min_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations.	*Duration*	Job-wide	No	Must be less than or equal to max_walltime.		No default	pbs.duration	
mpiprocs Number of MPI processes for this chunk. Cannot use sum from chunks as job-wide limit. The number of lines in PBS_NODEFILE is the sum of the values of mpiprocs for all chunks requested by the job. For each chunk with mpiprocs=P, the host name for that chunk is written to the PBS_NODEFILE *P* times.	*Integer*	Host-level				If ncpus >0: *1* Otherwise: *0*	int	
naccelerators Number of accelerators on the host. PBS sets this resource to the number of accelerators with state = *UP*. On Cray XC, PBS sets this resource only on vnodes whose hosts have at least one accelerator with state = *UP*. For Cray XC, PBS sets this resource on the 0th NUMA node (the vnode with PBScrayseg=*0*), and the resource is shared by other vnodes on the compute node. For example, on vnodeA_2_0: resources_available.naccelerators=1 On vnodeA_2_1: resources_available.naccelerators=@vnodeA_2_0	*Integer*	Host-level	Yes			No default	int	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
nchunk Number of chunks requested between plus symbols in a select statement. For example, if the select statement is `-lselect 4:ncpus=2+12:ncpus=8`, the value of nchunk for the first part is *4*, and for the second part it is *12*. The nchunk resource cannot be named in a select statement; it can only be specified by placing a number before the colon, as in the above example. When the number is omitted, nchunk is 1. This resource can be used to specify the default number of chunks at the server or queue. Example: `set queue myqueue default_chunk.nchunk=2` This resource cannot be used in server and queue resources_min and resources_max.	*Integer*		No			*1*	int	
ncpus Number of processors.	*Integer*	Host-level	Yes			No default	int	
nice Nice value with which the job is to be run. Host-dependent.	*Integer*	Job-wide				No default	int	
nodect **Deprecated**. Number of chunks in resource request from selection directive, or number of hosts requested from node specification. Read-only.	*Integer*	Job-wide				*1*	int	
nodes **Deprecated**. Number of hosts requested.	*Integer*					No default		

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
ompthreads Number of OpenMP threads for this chunk. Cannot use sum from chunks as job-wide limit. For the MPI process with rank 0, the environment variables NCPUS and OMP_NUM_THREADS are set to the value of ompthreads. For other MPI processes, behavior is dependent on MPI implementation.	*Integer*	Host-level	No			Value of ncpus	int	
PBScrayhost Used to differentiate a Cray XC system, containing ALPS, login nodes running PBS MoMs, and compute nodes, from a separate Cray XC system with a separate ALPS.	*String*		No			Value of mpp_host for this system	str	Cray XC only
PBScraylabel_<label name> Tracks labels applied to compute nodes. For each label on a compute node, PBS creates a custom resource whose name is a concatenation of *PBScraylabel_* and the name of the label. Name format: *PBScraylabel_<label name>* For example, if the label name is *Blue*, the name of this resource is *PBScraylabel_Blue*.	*Boolean*					PBS sets the value of the resource to *True* on all vnodes representing the compute node.	bool	Cray XC only
PBScraynid Tracks the node ID of the associated compute node. All vnodes representing a particular compute node share a value for PBScraynid.	*String*		No			The value of PBScraynid is set to the value of node_id for this compute node.	str	Cray XC only

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
PBScrayorder Tracks the order in which compute nodes are listed in the Cray XC inventory. All vnodes associated with a particular compute node share a value for PBScrayorder. Do not use this resource in a resource request.	*Integer*		No			Vnodes for the first compute node listed are assigned a value of 1 for PBScrayorder. The vnodes for each subsequent compute node listed are assigned a value one greater than the previous value.	int	Cray XC only
PBScrayseg Not used.	*String*					No default	str	Cray XC only
pcput Amount of CPU time allocated to any single process in the job. Establishes a per-process resource limit.	*Duration*	Job-wide	No			No default	pbs.duration	
pmem Amount of physical memory (workingset) for use by any single process of the job. Establishes a per-process resource limit. A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Job-wide	No			No default	pbs.size	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
preempt_targets List of resources and/or queues. Jobs requesting those resources or in those queues are preemption targets.	*string array* Syntax: *preempt_targets="Queue=<queue name>[,Queue=<queue name>],Resource_List.<resource>=<value>[,Resource_List.<resource>= <value>]"* or *preempt_targets=None* Keywords "queue" and "none" are case-insensitive. You can list multiple comma-separated targets.	Job-wide	No			No default	str	
pvmem Amount of virtual memory for use by any single process in the job. Establishes a per-process resource limit. A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Job-wide	No			No default	pbs.size	
site Arbitrary string resource.	*String*	Job-wide	No			No default	str	
software Site-specific software specification.	*String*	Job-wide		Allowable values and effect on job placement are site-dependent.		No default	pbs.software	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
soft_walltime Soft limit on walltime. Similar to walltime, but cannot be requested by unprivileged users, and job is not killed if it exceeds its soft_walltime. A job's soft_walltime cannot exceed its walltime. Can be set by Manager only.	*Duration*					No default	pbs.duration	
start_time The estimated start time for this job. Cannot be requested for a job; used for reporting only. Appears only in job's estimated attribute. Read-only.	*Integer*					No default	int	
vmem Amount of virtual memory for use by all concurrent processes in the job. Establishes a per-chunk resource limit. A scheduler rounds all resources of type size up to the nearest kb.	*Size*	Host-level	Yes			No default	pbs.size	
vnode Name of virtual node (vnode) on which to execute. Site-dependent. See "Vnode Attributes" on page 320 of the PBS Professional Reference Guide.	*String*	Host-level				No default	str	

Resources								
Name Description	Format	Scope	Consumable	Val / Opt	Value/Option Description	Default Value	Python Type	Platform
vntype The type of the vnode. Automatically set by PBS to one of two specific values for Cray XC vnodes. Has no meaning for non-Cray XC vnodes. On CLE, automatically added to `resources:` line in `sched_config`. Must be manually added to resources: line when your server/scheduler runs on non-CLE host.	*string array*	Host-level	No	*cray_compute*	This vnode represents part of a compute node.	Automatically set for Cray XC; none for non-Cray XC	str	all
				cray_login	This vnode represents a login node.			
walltime Amount of wall-clock time. Establishes a job-wide resource limit. Actual elapsed time may differ from walltime during Daylight Savings transitions.	*Duration*	Job-wide	No			5 years	pbs.duration	

6 Attributes

This chapter lists all of the supported PBS attributes. Attributes are listed by the PBS object they modify. For example, all supported attributes of jobs are listed in section 6.11, “Job Attributes”, on page 328. Attributes are case-sensitive.

6.1 Attribute Behavior

- When you set the value of most attributes, the change takes place immediately. You do not need to restart any daemons in order to make the change.
- When an attribute is unset, it behaves as if it is at its default value.

6.2 How To Set Attributes

You set most attributes via the qmgr command. You can set vnode attributes during vnode creation (see "Creating Vnodes" on page 38 in the PBS Professional Administrator’s Guide), or afterward (see "Configuring Vnodes" on page 41 in the PBS Professional Administrator’s Guide). Many job attributes can be set at submission via the qsub command.

The following are the instructions for setting most attributes.

To set the value of a non-string_array attribute, use the `qmgr` command, either from the command line or within `qmgr`:

```
qmgr -c "set <object> <attribute> = <value>"
Qmgr: set <object> <attribute> = <value>
```

To set or change the value of a string_array attribute, use the qmgr command, either from the command line or within qmgr:

```
qmgr -c "set <object> <attribute> = <value>"
qmgr -c 'set <object> <attribute> = "<value,value>"'
qmgr -c 'set <object> <attribute> += <value>'
qmgr -c 'set <object> <attribute> -= <value>'
Qmgr: set <object> <attribute> = <value>
Qmgr: set <object> <attribute> = '<value,value>'
Qmgr: set <object> <attribute> += <value>
Qmgr: set <object> <attribute> -= <value>
```

To unset the value of an attribute:

```
qmgr -c "unset <object> <attribute>"
Qmgr: unset <object> <attribute>
```

where <*object*> is one of *server*, *queue*, *hook*, *node*, or *sched*.

For example, to set resources_max.walltime at the server to be 24 hours:

```
Qmgr: set server resources_max.walltime = 24:00:00
```

See "qmgr" on page 149.

6.3 Viewing Attribute Values

If you want to view attribute values, the following commands are helpful:

qstat; see section 2.57, "qstat", on page 197

qmgr; see section 2.47, "qmgr", on page 149

pbs_rstat; see section 2.32, “pbs_rstat”, on page 94

- To see server attributes, use one of the following:
  ```
  qstat -B -f
  Qmgr: list server
  ```
- To see queue attributes, use one of the following:
  ```
  qstat -Q -f <queue name>
  Qmgr: list queue <queue name>
  ```
- To see job attributes:
  ```
  qstat -f <job ID>
  ```
- To see hook attributes:
  ```
  Qmgr: list hook <hook name>
  ```
- To see scheduler attributes:
  ```
  Qmgr: list sched
  ```
- To see vnode attributes:
  ```
  Qmgr: list node <node name>
  ```
- To see reservation attributes:
  ```
  pbs_rstat -F
  ```

6.4 Attribute Table Format

In the following tables, the columns contain the following information:

Name
: The name of the attribute

Description
: A description of the attribute's function

Format
: The attribute's format

Val/Opt
: If the attribute can take only specific values or options, each is listed here

Value/Option Description

If the attribute can take only specific values or options, the behavior of each value or option is described here

Default Value, Def Val

The attribute's default value, if any

Python Type

The attribute's Python type

User, Oper, Mgr

Indicates the actions allowed for unprivileged users, Operators, and Managers

The following table shows the operations allowed and their symbols:

Table 6-1: User, Operator, Manager Actions

Symbol	Explanation
r	Entity can read attribute
w	Entity can directly set or alter attribute
s	Entity can set but not alter attribute
a	Entity can alter but not set attribute
i	Entity can indirectly set attribute
-	Entity cannot set or alter attribute, whether directly or indirectly

6.5 Caveats

- The Python types listed as Python dictionaries support a restricted set of operations. They can reference values by index. Other features, such as has_key(), are not available.
- Do not use `qmgr` to set attributes for reservation queues.

6.6 Server Attributes

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
acl_host_enable Specifies whether the server obeys the host access control list in the acl_hosts server attribute.	*Boolean.*		When this attribute is *True*, the server limits host access according to the access control list.	*False*; all hosts allowed access	bool	r	r	r, w
acl_host_moms_enable Specifies whether all MoMs are automatically allowed to contact the server with the same privilege as hosts listed in the acl_hosts server attribute.	*Boolean*	*True*	All MoMs are automatically allowed to contact the server with the same privilege as hosts listed in the acl_hosts server attribute.	*False*	bool	r	r	r, w
		False	MoMs are not automatically allowed to contact the server with the same privilege as hosts listed in the acl_hosts server attribute.					
acl_hosts List of hosts from which services can be requested of this server. Requests from the server host are always honored whether or not that host is in the list. This list contains the fully qualified domain names of the hosts. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *"[+|-]<host-name>.<domain>[, ...]"*			No default	pbs.acl	r	r	r, w
acl_resv_group_enable Specifies whether the server obeys the group reservation access control list in the acl_resv_groups server attribute.	*Boolean*		When this attribute is *True*, the server limits group access according to the access control list.	*False*; all groups allowed access	bool	r	r	r, w
acl_resv_groups List of groups allowed or denied permission to create reservations in this PBS complex. The groups in the list are groups on the server host, not submission hosts. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *"[+|-]<group name>[, ...]"*				pbs.acl	r	r	r, w
acl_resv_host_enable Specifies whether the server obeys the host reservation access control list in the acl_resv_hosts server attribute.	*Boolean*		When this attribute is *True*, the server limits host access according to the access control list.	*False*; access allowed from all hosts	bool	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
acl_resv_hosts List of hosts from which reservations can be created in this PBS complex. This list is made up of the fully-qualified domain names of the hosts. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *"[+|-]<host-name>.<domain>[, ...]"*			No default	pbs.acl	r	r	r, w
acl_resv_user_enable Specifies whether the server limits which users are allowed to create reservations, according to the access control list in the acl_resv_users server attribute.	*Boolean*		When this attribute is *True*, the server limits user reservation creation according to the access control list.	*False*; all users are allowed to create reservations	bool	r	r	r, w
acl_resv_users List of users allowed or denied permission to create reservations in this PBS complex. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *"[+|-]<user-name>[@<host-name>][, ...]"*			No default	pbs.acl	r	r	r, w
acl_roots List of users with root privilege who can submit and run jobs in this PBS complex. For any job whose owner is root or Administrator, the job owner must be listed in this access control list, or the job is rejected. List is evaluated left-to-right; first match in list is used. Can be set or altered by root only, and only at the server host.	*String.* Syntax: *"[+|-]<user-name>[@<host-name>][, ...]"*			No default; no root jobs allowed	pbs.acl	r	r	r
acl_user_enable Specifies whether the server limits which users are allowed to run commands at the server, according to the control list in the acl_users server attribute.	*Boolean*		When this attribute is *True*, the server limits user access according to the access control list.	*False*; all users have access	bool	r	r	r, w
acl_users List of users allowed or denied permission to run commands at this server. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *"[+|-]<user-name>[@<host-name>][, ...]"*			No default	pbs.acl	r	r	r, w
backfill_depth Specifies backfilling behavior. Sets the number of jobs that are to be backfilled around. Overridden by backfill_depth queue attribute. Recommendation: set this to less than 100.	*Integer.* Must be >=0	*>=0*	PBS backfills around the specified number of jobs.	Unset. When unset, backfill depth is *1*	int	r	r, w	r, w
		Unset	Backfill depth is set to *1.*					

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
comment Informational text. Can be set by a scheduler or other privileged client.	*String* of any form			No default	str	r	r, w	r, w
default_chunk The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.	*String*. Syntax: *default_chunk.<resource name>=<value>,default_chunk.<resource name>=<value>, ...*			No default	pbs.pbs_resource Syntax: *default_chunk["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
default_node No longer used.						-	-	-
default_qdel_arguments Argument to `qdel` command. Automatically added to all `qdel` commands. See `qdel(1B)`. Overrides standard defaults. Overridden by arguments given on the command line.	*String.* Syntax: *"-Wsuppress_mail=<N>"*			No default	pbs.args	r	r, w	r, w
default_qsub_arguments Arguments that are automatically added to the `qsub` command. Any valid arguments to `qsub` command, such as job attributes. Setting a job attribute via default_qsub_arguments sets that attribute for each job which does not explicitly override it. See `qsub(1B)`. Settable by the administrator via the `qmgr` command. Overrides standard defaults. Overridden by arguments given on the command line and in script directives.	*String*. Syntax: *"<option> <value> <option> <value>", e.g. "-r y -N MyJob"* To add to existing: `Qmgr: s s default_qsub_arguments +="<option> <value>"`			No default	pbs.args	r	r, w	r, w
default_queue The name of the default target queue. Used for requests that do not specify a queue name. Must be set to an existing queue.	*Queue name*			*workq*	pbs.queue	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
eligible_time_enable Controls starving behavior. Toggles between using the value of the job's eligible_time attribute and the value of now() - etime to evaluate whether job is starving.	*Boolean*	*True*	The value of the job's eligible_time attribute is used for its starving time.	*False*	bool	r	r	r, w
		False	The value of now() - etime is used for the job's starving time.					
est_start_time_freq **Obsolete**. No longer used.								
flatuid Used for authorization allowing users to submit and alter jobs. Specifies whether user names are treated as being the same across the PBS server and all submission hosts in the PBS complex. Can be used to allow users without accounts at the server host to submit jobs. If UserA has an account at the server host, PBS requires that `UserA@<server host>` is the same as `UserA@<execution host>`.	*Boolean*	*True*	PBS assumes that `UserA@<submithost>` is same user as `UserA@<server name>`. Jobs that run under the name of the job owner do not need authorization. A job submitted under a different username, by using the u option to the `qsub` command, requires authorization. Entries in `.rhosts` or `hosts.equiv` are not checked, so even if UserA@host1 has an entry for UserB@host2, UserB@host2 cannot operate on UserA@host1's jobs. User without account on server can submit jobs.	*False*; authorization is required	bool	r	r	r, w
		False	PBS does not assume that `UserA@<submission host>` is the same user as `UserA@<server host>`. Jobs that run under the name of the job owner need authorization. Users must have accounts on the server host to submit jobs.					
FLicenses The number of licenses currently available for allocation to unlicensed hosts.	*Integer*			No default	int	r	r	r
job_history_duration The length of time PBS will keep each job's history.	*Duration*			*Two weeks*	pbs.duration	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
job_history_enable Enables job history management. Setting this attribute to *True* enables job history management.	*Boolean*			*False*	bool	r	r	r, w
job_requeue_timeout The amount of time that can be taken while requeueing a job. Minimum allowed value: 1 second. Maximum allowed value: 3 hours.	Duration			*45 seconds*	pbs.duration	r	r, w	r, w
job_sort_formula Formula for computing job priorities. Described in the *PBS Professional Administrator's Guide*. If the attribute job_sort_formula is set, all schedulers use the formula in it to compute job priorities. When this scheduler sorts jobs according to the formula, it computes a priority for each job, where that priority is the value produced by the formula. Jobs with a higher value get higher priority.	*String*. Syntax: mathematical formula; can be made up of expressions, where expressions contain terms which are added, subtracted, multiplied, or divided, and which can contain parentheses, exponents, unary plus and minus, the ternary operator, and Python math module functions.			Unset	pbs.job_sort_formula	r	r	r, w
jobscript_max_size Limit on the size of any job script.	*size* Units default to bytes			*100MB*	pbs.size	r	r	r, w
license_count The license_count attribute contains the following elements with their values: Avail_Global, Avail_Local, Used, High_Use.	*String.* Syntax: *Avail_Global:<value> Avail_Local:<value> Used:<value> High_Use:<value>*	Avail_Global	The number of licenses available at ALM license server (checked in.)	*Avail_Global:0 Avail_Local:0 Used:0 High_Use:0*	pbs.license_count	r	r	r
		Avail_Local	The number of licenses kept by PBS (checked out.)					
		Used	The number of licenses currently in use.					
		High_Use	The highest number of licenses ever checked out and used by the current instance of the PBS server.					
log_events The types of events the server logs.	*Integer* representation of bit string			*511*	int	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
mail_from The username from which server-generated mail is sent to users. Mail is sent **to** this address upon failover.	*String*			*"adm"*	str	r	r	r, w
managers List of PBS Managers.	*String*. Syntax: *"<user-name>@<host-name>.<subdomain>.<domain>[,<user name>@<host-name>.<subdo-main>.<domain> ...]"*. The host, sub-domain, or domain name may be wildcarded with an asterisk (*).			*Root on the server host*	pbs.acl	r	r	r, w
max_array_size The maximum number of subjobs allowed in any array job.	*Integer*			*10000*	int	r	r, w	r, w
max_concurrent_provision The max_concurrent_provision attribute is the number of vnodes allowed to be in the process of being provisioned. Cannot be set to zero. When unset, default value is used.	*Integer*	>0		5	int	r	r	r, w
max_group_res Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single group may consume in this PBS complex.	*String*. Syntax: *max_group_res.<resource name>=<value>*	Any PBS resource, e.g. "ncpus", "mem", "pmem"		No default	pbs.pbs_resource Syntax: *max_group_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_group_res_soft Old limit attribute. Incompatible with new limit attributes. The soft limit for the specified resource that any single group may consume in this complex. If a group is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	*String*. Syntax: *max_group_res_soft.<resource name>=<value>*	Any PBS resource, e.g. "ncpus", "mem", "pmem", etc.		None	pbs.pbs_resource Syntax: *max_group_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_group_run Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by the users in one group allowed to be running within this complex at one time.	*Integer*			No default	int	r	r, w	r, w
max_group_run_soft Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by the users in one group allowed to be running in this complex at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	*Integer*			No default	int	r	r, w	r, w
max_job_sequence_id Maximum value of sequence number in a job ID, job array ID, or reservation ID. Minimum allowed is *9999999*. Maximum allowed is *999999999999*. After specified maximum for sequence number has been reached, job IDs start again at *0*.	*Integer*			*9999999*	int	r	r	r, w
max_queued Limit attribute. The maximum number of jobs allowed to be queued or running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.			No default	str	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_queued_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued or running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353. Syntax: *max_queued_res.<resource name>* = "*<.limit>*"			No default	pbs.pbs_resource Syntax: *max_queued_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_run Limit attribute. The maximum number of jobs allowed to be running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.			No default	str	r	r, w	r, w
max_run_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353 Syntax: *max_run_res.<resource name>* = "*<.limit>*"			No default	pbs.pbs_resource Syntax: *max_run_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_run_res_soft Limit attribute. Soft limit on the amount of the specified resource allowed to be allocated to jobs running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353 *max_run_res_soft.<resource name>* = "*<.limit>*"			No default	pbs.pbs_resource Syntax: *max_run_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_run_soft Limit attribute. Soft limit on the number of jobs allowed to be running in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.			No default	str	r	r, w	r, w
max_running Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs in this complex allowed to be running at any given time.	*Integer*			No default	int	r	r, w	r, w
max_user_res Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single user may consume within this complex.	*String.* Syntax: *max_user_res.<resource name>=<value>*	Any PBS resource, e.g. "ncpus", "mem", "pmem", etc.		No default	pbs.pbs_resource Syntax: *max_user_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_user_res_soft Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single user may consume within this complex. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	*String.* Syntax: *max_user_res_soft.<resource name>=<value>*	Any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc		No default	pbs.pbs_resource Syntax: *max_user_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_user_run Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by a single user allowed to be running within this complex at one time.	*Integer*			No default	int	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
max_user_run_soft Old limit attribute. Incompatible with new limit attributes. The soft limit on the number of jobs owned by a single user that are allowed to be running within this complex at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	*Integer*			No default	int	r	r, w	r, w
node_fail_requeue Controls whether running jobs are automatically requeued or are deleted when the primary execution host fails. Number of seconds to wait after losing contact with Mother Superior before requeueing or deleting jobs. Reverts to default value when server is restarted.	*Integer.* Units: *Seconds.*	*<0*	Behaves as if set to *1*.	*310*	int	r	r, w	r, w
		0	Jobs are not requeued; they are left in the *Running* state until the execution host is recovered.					
		>0	When the host has been down for the specified number of seconds, jobs are requeued if they are marked as rerunnable, or are deleted.					
		Unset	Behaves as if set to default value of *310*.					
node_group_enable Specifies whether placement sets (which includes node grouping) are enabled. See node_group_key server attribute.	*Boolean*		When set to *True*, placement sets are enabled.	*False*	bool	r	r, w	r, w
node_group_key Specifies the resources to use for placement sets (node grouping). Overridden by queue's node_group_key attribute. See node_group_enable server attribute.	*String_array* When specifying multiple resources, separate them with commas and enclose the value in double quotes.			Unset	pbs.node_group_key	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
operators List of PBS Operators.	*String.* Syntax: *<user-name>@<host-name>.<subdomain>.<domain name>[,<user-name>@<host-name>.<subdomain>.<domain name> ...]*. The host, subdomain, or domain name may be wildcarded with an asterisk (*).			No default	pbs.acl	r	r	r, w
pbs_license_file_location **Deprecated**. Do not use.	-	-	-	-	-	-	-	-
pbs_license_info Location of license server(s).	*String*. Syntax: One or more port number and hostname combinations: *<port1>@<host1>[:<port2>@<host2>:...:<portN>@<host N>]* where *host1*, *host2*, ... *hostN* can be IP addresses. Delimiter between items is colon (":").			No default	str	r	r	r, w
pbs_license_linger_time The number of seconds to keep an unused license, when the number of licenses is above the value given by pbs_license_min.	*Integer*. Units: seconds.			*31536000 seconds (1 year).*	pbs.duration	r	r	r, w
pbs_license_max Maximum number of licenses to be checked out at any time, i.e maximum number of licenses to keep in the PBS local license pool. Sets a cap on the number of nodes or sockets that can be licensed at one time.	*Integer*			Maximum value for an integer	int	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
pbs_license_min Minimum number of nodes or sockets to permanently keep licensed, i.e. the minimum number of licenses to keep in the PBS local license pool. This is the minimum number of licenses to keep checked out. If unset, PBS automatically sets the value to *0*.	*Integer*			*0*	int	r	r	r, w
pbs_version The version of PBS for this server.	*String*			No default	pbs.version	r	r	r
power_provisioning Reflects use of power profiles via PBS. Set by PBS to *True* when PBS_power hook is enabled.	*Boolean*	*True* *False*	Power provisioning is enabled. Power provisioning is disabled.	*False*	bool	r	r	r, w
python_restart_max_hooks The maximum number of hooks to be serviced before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.	*Integer*			*100*	int	r	r	r, w
python_restart_max_objects The maximum number of objects to be created before the Python interpreter is restarted. If this number is exceeded, and the time limit set in python_restart_min_interval has elapsed, the Python interpreter is restarted.	*Integer*			*1000*	int	r	r	r, w
python_restart_min_interval The minimum time interval before the Python interpreter is restarted. If this interval has elapsed, and either the maximum number of hooks to be serviced (set in python_restart_max_hooks) has been exceeded or the maximum number of objects to be created (set in python_restart_max_objects) has been exceeded, the Python interpreter is restarted.	*Integer*. Units: *Seconds* or *[[HH:]MM:]SS* (duration)			*30*	pbs.duration	r	r	r, w
query_other_jobs Controls whether unprivileged users are allowed to select or query the status of jobs owned by other users.	*Boolean*		When this attribute is *True*, unprivileged users can query or select other users' jobs.	On installation: *True* After being unset: *False*	bool	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
queued_jobs_threshold Limit attribute. The maximum number of jobs allowed to be queued in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.			No default	str	r	r, w	r, w
queued_jobs_threshold_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued in the complex. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353. *queued_jobs_threshold_res.<resource name> = "<.limit>"*			No default	pbs.pbs_resource Syntax: *queued_jobs_threshold_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
reserve_retry_cutoff **Obsolete**. No longer used.								
reserve_retry_init **Deprecated**. The amount of time after a reservation becomes degraded that PBS waits before attempting to reconfirm the reservation. When this value is changed, only reservations that become degraded after the change use the new value. Must be greater than zero.	*Integer.* Units: *Seconds*			*7200 (2 hours)*	int	-	-	r, w
reserve_retry_time The amount of time after a reservation becomes degraded that PBS waits before attempting to reconfirm the reservation, as well as amount of time between attempts to reconfirm degraded reservations. When this value is changed, PBS uses the new value for any subsequent attempts. Must be greater than zero.	*Integer.* Units: *Seconds*			*600 (10 minutes)*	int	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
resources_assigned The total of each type of resource allocated to jobs running and exiting in this complex, plus the total of each type of resource allocated to any reservation. Reservation resources are added when the reservation starts.	*String*. Syntax: *resources_assigned.<resource name>=<value>[,resources_assigned.<resource name>=<value>,...]*			No default	pbs.pbs_resource Syntax: *resources_assigned["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r	r
resources_available The list of available resources and their values defined on the server. Each resource is listed on a separate line.	*String.* Syntax: *resources_available.<resource name>=<value>*			No default	pbs.pbs_resource Syntax: *resources_available["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
resources_cost No longer used.						-	-	-
resources_default The list of default job-wide resource values that are set as limits for jobs in this complex when a) the job does not specify a limit, and b) there is no queue default. The value for a string array, e.g. resources_default.<string array resource>, can contain only one string. For host-level resources, see the default_chunk.<resource name> server attribute.	*String* Syntax: *resources_default.<resource name>=<value>[,...]*			No limit	pbs.pbs_resource Syntax: *resources_default["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
resources_max The maximum amount of each resource that can be requested by any single job in this complex, if there is not a resources_max value defined for the queue at which the job is targeted. This attribute functions as a gating value for jobs entering the PBS complex.	*String* Syntax: *resources_max.<resource name>=<value>[, ...]*			No limit	pbs.pbs_resource Syntax: *resources_max["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
restrict_res_to_release_on_suspend Comma-separated list of consumable resources to be released when jobs are suspended. If unset, all consumable resources are released on suspension.	*String_array* Syntax: comma-separated list			unset	Python list	r	r	r, w
resv_enable Specifies whether or not advance and standing reservations can be created in this complex.	*Boolean*		When set to *True*, new reservations can be created. When changed from *True* to *False*, new reservations cannot be created, but existing reservations are honored.	*True*	bool	r	r	r, w
resv_post_processing_time The amount of time allowed for reservations to clean up after running jobs. Reservation duration and end time are extended by this amount of time. Jobs are not allowed to run during the cleanup period.	*Duration*			Unset; behaves as if zero	int	r	r, w	r, w
rpp_highwater The maximum number of messages.	*Integer*	Greater than or equal to one		*1024*	int	r	r	r, w
rpp_max_pkt_check Maximum number of TPP messages processed by the main server thread per iteration.	Integer			64	int	r	r	r, w
rpp_retry In a fault-tolerant setup (multiple `pbs_comm`s), when the first `pbs_comm` fails partway through a message, this is number of times TPP tries to use the first `pbs_comm`.	*Integer*	Greater than or equal to zero		*10*	int	r	r	r, w

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
scheduler_iteration The time between scheduling iterations.	*Integer.* Units: *Seconds.*			*10 minutes (600 seconds)*	pbs.duration	r	r, w	r, w
scheduling Enables scheduling of jobs. Specified by value of -a option to `pbs_server` command. If -a is not specified, value is taken from previous invocation of `pbs_server`.	*Boolean*		When this attribute is set to *True*, scheduling is enabled.	*False* if never set via `pbs_server` command.	bool	r	r, w	r, w
server_host The name of the host on which the active server is running. If the secondary server takes over, this attribute is set to the name of the secondary server's host. When the primary server takes control again, this attribute shows the name of the primary server's host.	*String*. Syntax: *<host-name>.<domain name>* If the server is listening to a non-standard port, the port number is appended, with a colon, to the hostname: *<host-name>.<domain name>:<port number>*			No default	str	r	r	r

Server Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
server_state The current state of the server:	*String*	*Active*	The server is running. The scheduler is not in a scheduling cycle.	No default	Server state constant pbs.SV_STATE_ACTIVE	r	r	r
		Hot_Start	The server will run first any jobs that were running when it was shut down.		Server state constant pbs.SV_STATE_HOT			
		Idle	The server is running. The default scheduler's scheduling attribute is *False*.		Server state constant pbs.SV_STATE_IDLE			
		Scheduling	The server is running. The scheduler is in a scheduling cycle.		Server state constant pbs.SV_STATE_ACTIVE			
		Terminating	The server is terminating. No additional jobs will be run.		Server state constant pbs.SV_STATE_SHUTIMM or pbs.SV_STATE_SHUTSIG			
		Terminating_Delayed	Server is terminating in delayed mode. No new jobs will be run. server will shut down after all running jobs are finished.		Server state constant pbs.SV_STATE_SHUTDEL			
single_signon_password_enable **Removed**. (2020.1)								
state_count List of the number of jobs in each state in the complex. Suspended jobs are counted as running.	*String.* Syntax: *transiting=<value>, queued=<value>, ...*			No default	pbs.state_count	r	r	r
system_cost No longer used.						-	-	-
total_jobs The total number of jobs in the complex. If the job_history_enable attribute is set to *True*, this includes jobs that are finished, deleted, and moved.	*Integer*			No default	int	r	r	r

6.7 Scheduler Attributes

Scheduler Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
comment For certain scheduler errors, PBS sets the scheduler's comment attribute to specific error messages. You can use the comment attribute to notify another administrator of something, but PBS does overwrite the value of comment under certain circumstances.	*String*			No default	None	r	r	r, w
do_not_span_psets Specifies whether or not this scheduler requires the job to fit within one existing placement set.	*Boolean*	*True*	The job must fit in one existing placement set. All existing placement sets are checked. If the job fits in an occupied placement set, the job waits for the placement set to be available. If the job can't fit within a single placement set, it won't run.	*False*	None	r	r, w	r, w
		False	This scheduler first attempts to place the job in a single placement set. All existing placement sets are checked. If the job fits in an occupied placement set, the job waits for the placement set to be available. If there is no existing placement set, occupied or empty, into which the job could fit, the job runs regardless of placement sets, running on whichever vnodes can satisfy the job's resource request.					
job_sort_formula_threshold Lower bound for calculated priority for job. If job priority is at or below this value, the job is not eligible to run in the current scheduler cycle.	*Float*			No default	None	-	r	r, w
log_events Types of events logged by this scheduler.	*Integer* representation of bit string			767	None	r	r, w	r, w
only_explicit_psets Specifies whether placement sets are created for unset resources.	*Boolean*	*True*	Placement sets are not created from vnodes whose value for a resource is unset.	*False*	None	r	r, w	r, w
		False	Placement sets are created from vnodes whose value for a resource is unset.					

Scheduler Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
opt_backfill_fuzzy Sets the trade-off between scheduling cycle speed and granularity of estimated start time calculation.	*String*	*off*	Finest granularity, no speedup	*unset*; behaves like *low*	None	r	r	r, w
		low	Fairly fine granularity, some speedup					
		medium	Medium granularity, medium speedup					
		high	Coarse granularity, greatest speedup					
partition Name of partition for which this scheduler is to run jobs. Cannot be set on default scheduler.	*String*			"*None*"	None	r	r	r, w
pbs_version The version of PBS for this scheduler.	*String*			No default	None	-	r	r

preempt_order
Defines the order of preemption methods which this scheduler uses on jobs. This order can change depending on the percentage of time remaining on the job. The ordering can be any combination of *S, C, R,* and *D.*
Usage: an ordering (*SCR*) optionally followed by a percentage of time remaining and another ordering.
For example, PBS should first attempt to use suspension to preempt a job, and if that is unsuccessful, requeue the job:
`preempt_order: "SR"`
For example, if the job has between 100% and 81% of requested time remaining, first try to suspend the job, then try checkpoint, then requeue. If the job has between 80% and 51% of requested time remaining, attempt suspend, then checkpoint. Between 50% and 0% time remaining, just attempt to suspend the job:
`preempt_order: "SCR 80 SC 50 S"`
For each job percentage, each method can be used only once. Note that in the example above, the *S* method appears only once per percentage.
	String, as a quoted list	*C*	Checkpoint job	"*SCR*"		r	r	r, w

		D	Delete job					
		R	Requeue job					
		S	Suspend job					

Scheduler Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr

preempt_prio
Specifies the ordering of priority for different preemption levels. Two or more job types may be combined at the same priority level with a plus sign ("+") between them, using no whitespace. Comma-separated preemption levels are evaluated left to right, with higher priority to the left. Any level not specified in the preempt_prio list is ignored.
For example, starving jobs have the highest priority, then normal jobs, and jobs whose entities are over their fairshare limit are third highest:
`preempt_prio: "starving_jobs, normal_jobs, fairshare"`
For example, starving jobs whose entities are also over their fairshare limit are lower priority than normal jobs:
`preempt_prio: "normal_jobs, starving_jobs+fairshare"`
	string_array, as quoted list	express_queue	Jobs in express queues preempt other jobs. See preempt_queue_prio. Does not require by_queue to be *True*.	"*express_queue, normal_jobs*"	None	r	r	r, w

		starving_jobs	When a job becomes starving it can preempt other jobs.					
		fairshare	When the entity owning a job exceeds its fairshare limit.					
		queue_softlimits	Jobs which are over their queue soft limits					
		server_softlimits	Jobs which are over their server soft limits					
		normal_jobs	The preemption level into which a job falls if it does not fit into any other specified level.					
preempt_queue_prio Specifies the minimum queue priority required for a queue to be classified as an express queue. Express queues do not require by_queue to be *True*.	*Integer*			*150*	None	r	r	r, w
preempt_sort Specifies how jobs most eligible for preemption are sorted. See "Sorting Within Preemption Level" on page 187 in the PBS Professional Administrator's Guide.	*String*	*min_time_since_start*	First job preempted will be that with most recent start time	*min_time_since_start*	None	r	r	r, w
scheduler_iteration Time in seconds between scheduling iterations. If you set the server's scheduler_iteration attribute, that value is assigned to the default scheduler's scheduler_iteration attribute, and vice versa.	*Integer*. Units: *Seconds*			*600*	None	r	r	r, w
scheduling Enables scheduling of jobs. If you set the server's scheduling attribute, that value is assigned to the default scheduler's scheduling attribute, and vice versa.	*Boolean*			For default scheduler: *True* For multischeds: *False*	None	r	r	r, w

Scheduler Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
sched_cycle_length This scheduler's maximum cycle length. Overwritten by the -a alarm option to pbs_sched command.	*Duration*			*20:00* (20 minutes)	None	r	r, w	r, w
sched_host The hostname of the machine on which this scheduler runs. Cannot be set on default scheduler; value for default scheduler is server hostname. Must be set by administrator.	*String*			*Server's host*	None	-	r	r
sched_log Directory where this scheduler writes its logs. Permissions should be *755*. Must be owned by root. Cannot be shared with another scheduler. For default scheduler, directory is always PBS_HOME/sched_log. Settable for multischeds.	*String*			*$PBS_HOME/sched_logs_<scheduler name>*	None	r	r	r, w
sched_port **Deprecated**. Port on which this scheduler listens. Cannot be set on default scheduler. Must be set by administrator.	*String*			No default	None	r	r	r, w
sched_preempt_enforce_resumption Controls whether this scheduler treats preempted jobs as top jobs. When *True*, preempted jobs are treated as top jobs.	*Boolean*			False	None	r	r	r, w
sched_priv Directory where this scheduler keeps fairshare usage, resource_group, holidays, and sched_config files. Must be owned by root. For default scheduler, directory is always PBS_HOME/sched_priv. Settable for multischeds.	*String*			*$PBS_HOME/sched_priv_<scheduler name>*	None	r	r	r, w
server_dyn_res_alarm Specifies how long this scheduler allows any server_dyn_res script to run. If the script times out, the script is terminated and the scheduler uses zero as the value that would have been returned by the script.	*Integer*	*0 (zero)*	No time limit is enforced for server_dyn_res scripts	*30 seconds*	None	r	r	r, w
		>0 (greater than zero)	Value is number of seconds any server_dyn_res script is allowed to run					
state State of this scheduler. Set by server.	*String*	*down*	Scheduler is not running	For default scheduler: idle For multi-sched: down	None	r	r	r
		idle	Scheduler is running and is waiting for a scheduling cycle to be triggered					
		scheduling	Scheduler is running and is in a scheduling cycle					

Scheduler Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	User	Oper	Mgr
throughput_mode Allows scheduler to run faster; it doesn't have to wait for each job to be accepted, and doesn't wait for execjob_begin hooks to finish. Also allows jobs that were changed via `qalter`, server_dyn_res scripts, or peering to run in the same scheduling cycle where they were changed.	*Boolean*	*True*	Scheduler runs asynchronously and faster. Only available when PBS complex is in TPP mode.	True	None	r	r, w	r, w
		False	Scheduler does not run asynchronously					

6.8 Reservation Attributes

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
Account_Name No longer used.						-	-	-
Authorized_Groups List of groups who can or cannot submit jobs to this reservation. Group names are interpreted relative to the server, not the submission host. List is evaluated left-to-right; first match in list is used. This list is used to set the reservation queue's acl_groups attribute. See the G option to the pbs_rsub command.	*String.* Syntax: *[+|-]<group name> [, [+|-]<group name> ...]* where '-' means "deny" and '+' means "allow".			No default. (Jobs can be submitted by all groups)	pbs.acl	r, w	r, w	r, w

Authorized_Hosts
The list of hosts from which jobs can and cannot be submitted to this reservation. List is evaluated left-to-right; first match in list is used. This list is used to set the reservation queue's acl_hosts attribute. See the H option to the pbs_rsub command.
	String. Syntax:
	[+|-]<hostname> [, [+|-]<hostname> ...]
	where '-' means "deny" and '+' means "allow".
	Hostnames may be wildcarded using an asterisk, according to the following rules:
	A hostname can contain at most one asterisk
	The asterisk must be the leftmost label
	Examples:
	`*.test.example.com`
	`*.example.com`
	`*.com`
				No default. (Jobs can be submitted from all hosts)	pbs.acl	r, w	r, w	r, w

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr

Authorized_Users
The list of users who can or cannot submit jobs to this reservation. List is evaluated left-to-right; first match in list is used. This list is used to set the reservation queue's acl_users attribute. See the `U` option to the `pbs_rsub` command.
	String. Syntax: *[+|-]<username>[@<hostname>.<domain>] [, [+|-]<username>[@<hostname>.<domain>] ...]* where '-' means "deny" and '+' means "allow".
	Hostnames may be wildcarded using an asterisk, according to the following rules:
	A hostname can contain at most one asterisk
	The asterisk must be the leftmost label in the hostname
	Examples:
	`*.test.example.com`
	`*.example.com`
	`*.com`
				Reservation owner only	pbs.acl	r, w	r, w	r, w

ctime Timestamp; time at which the reservation was created.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r
delete_idle_time Amount of time a reservation can sit idle before it is deleted. Applies to each instance of a standing reservation.	*Duration*. Syntax: either *integer seconds* or *HHHH:MM:SS*			None except for ASAP reservations: *10 minutes*	pbs.duration	r, w	r, w	r, w

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
group_list No longer used.						-	-	-
hashname No longer used.						-	-	-
interactive Number of seconds that the `pbs_rsub` command will block while waiting for confirmation or denial of the reservation. See the `-I block_time` option to the `pbs_rsub` command.	*Integer*	Less than zero	The reservation is automatically deleted if it cannot be confirmed in the time specified.	*Zero*	int	r, w	r, w	r, w
		Zero or greater than zero	The reservation is not automatically deleted if it cannot be confirmed in the time specified.					
Mail_Points Sets the list of events for which mail is sent by the server. Mail is sent to the list of users specified in the Mail_Users attribute. See the `m mail_points` option to the `pbs_rsub` command.	*String* consisting of 1) one or more of the letters "*a*", "*b*", "*c*", "*e*", or *2*) the string "*n*". Cannot use "*n*" with any other letter	*a*	Notify when reservation is terminated	"*ac*"	pbs.mail_points	r, w	r, w	r, w
		b	Notify when reservation period begins					
		c	Notify when reservation is confirmed					
		e	Notify when reservation period ends					
		n	Do not send mail. Cannot be used with other letters.					
Mail_Users The set of users to whom mail is sent for the reservation events specified in the Mail_Points attribute. See the `M mail_list` option to the `pbs_rsub` command.	*String.* Syntax: *<username>@<hostname>[,<username>@<hostname>, ...]*			Reservation owner only	pbs.user_list	r, w	r, w	r, w

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
mtime Timestamp: the time that the reservation was last modified.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.				int	r	r	r
Priority No longer used.						-	-	-
queue Name of the reservation queue. Jobs that are to use resources belonging to this reservation are submitted to this queue.	*String*. Format for an advance or job-specific reservation: *R<sequence number>* Format for a standing reservation: *S<sequence number>*				pbs.queue	r	r	r
reserve_count The count of occurrences in a standing reservation.	*Integer*				int	r, w	r, w	r, w
reserve_duration Reservation duration in seconds. For a standing reservation, this is the duration for one occurrence.	*Integer*				pbs.duration	r, w	r, w	r, w
reserve_end The date and time when an advance reservation or the soonest occurrence of a standing reservation ends.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.				int	r, w	r, w	r, w
reserve_ID The reservation identifier.	*String*. For an advance or job-specific reservation: string of the form *R<sequence number>.<server name>* For a standing reservation: string of the form *S<sequence number>.<server name>*				str	r	r	r

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_index The index of the soonest occurrence of a standing reservation.	*Integer*				int	r	r	r
reserve_job If this reservation is a job-specific start or now reservation, shows the ID of the job from which the reservation was created.	*String*			No default	str	r	r	r
Reserve_Name The name assigned to the reservation during creation, if specified. See the `N` option to the `pbs_rsub` command.	*String*. Syntax: up to 236 characters. First character is alphabetic			No default	str	r, w	r, w	r, w
Reserve_Owner The login name on the submission host of the user who created the reservation.	*String.* Syntax: <*username*>@<*hostname*>			Login name of creator	str	r	r	r
reserve_retry If this reservation becomes degraded, this is the next time that PBS will attempt to reconfirm this reservation.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_rrule The rule that describes the recurrence pattern of a standing reservation. See the `r` option to the `pbs_rsub` command.	*String*. Syntax: either of two forms: *"FREQ= <freq_spec>; COUNT= <count_spec>; <interval_spec>"* or *"FREQ= <freq_spec>; UNTIL= <until_spec>; <interval_spec>"*	*freq_spec*	Frequency with which the standing reservation repeats. Valid values are: *WEEKLY|DAILY|HOURLY*	No default	str	r, s	r, w	r, w
		count_spec	The exact number of occurrences. Number up to 4 digits in length. Format: *integer*.	No default				
		interval_spec	Specifies interval. Format is one or both of: *BYDAY = MO|TU|WE|TH|FR|SA|SU or BYHOUR = 0|1|2|...|23*	No default				
		until_spec	Occurrences will start up to but not after date and time specified. Format: *YYYYMMDD[THHMMSS]* Year-month-day part and hour-minute-second part separated by a capital *T*.	No default				
reserve_start The date and time when the reservation period for the reservation or soonest occurrence begins.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r, w	r, w	r, w

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
reserve_state The state of the reservation.	*String*	*NO RESV_NONE*	No reservation yet.	No default	Reservation state constant: pbs.RESV_STATE_NONE	r	r	r
		UN RESV_UNCONFIRMED	Reservation request is awaiting confirmation.		Reservation state constant: pbs.RESV_STATE_UNCONFIRMED			
		CO RESV_CONFIRMED	Resv. confirmed. All occurrences of standing resv. confirmed.		Reservation state constant: pbs.RESV_STATE_CONFIRMED			
		WT RESV_WAIT	Unused.		Reservation state constant: pbs.RESV_STATE_WAIT			
		TR RESV_TIME_TO_RUN	Start of the reservation period.		Reservation state constant: pbs.RESV_STATE_TIME_TO_RUN			
		RN RESV_RUNNING	Resv. period has started; reservation is running.		Reservation state constant: pbs.RESV_STATE_RUNNING			
		FN RESV_FINISHED	End of the reservation period.		Reservation state constant: pbs.RESV_STATE_FINISHED			
		BD RESV_BEING_DELETED	Reservation is being deleted.		Reservation state constant: pbs.RESV_STATE_BEING_DELETED			
		DE RESV_DELETED	Reservation has been deleted.		Reservation state constant: pbs.RESV_STATE_DELETED			
		DJ RESV_DELETING_JOBS	Jobs belonging to the reservation are being deleted		Reservation state constant: pbs.RESV_STATE_DELETING_JOBS			
		DG DEGRADED	Reservation is degraded.		Reservation state constant: pbs.RESV_STATE_DEGRADED			
reserve_substate The substate of the reservation or occurrence. The substate is used internally by PBS.	*Integer*			No default	int	r	r	r
reserve_type No longer used.						-	-	-

Reservation Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	User	Oper	Mgr
Resource_List The list of resources allocated to the reservation. Jobs running in the reservation cannot use in aggregate more than the specified amount of a resource.	*String*. Syntax: *Resource_List.<resource name>=<value>[, Resource_List.<resource name>=<value>, ...]*			No default	pbs.pbs_resource Syntax: *Resource_List["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r, w	r, w	r, w
resv_nodes The list of each vnode and the resources allocated from it to satisfy the chunks requested for this reservation or occurrence. For a maintenance reservation, value is set by PBS.	*String*. Syntax: *(<vnode name>:<resource name>=<value>[:<resource name>=<value>]...)[+(<vnode name>:<resource name>=<value>[:<resource name>=<value>])+...]*			No default	pbs.exec_vnode	r	r	r
server Name of server.	*String*			No default	pbs.server	r	r	r
User_List No longer used.						-	-	-
Variable_List Not used						-	-	-

6.9 Queue Attributes

In the following table, Queue Type indicates the type of queue to which the attribute applies: R (routing), E (execution):

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
acl_group_enable Controls whether group access to the queue obeys the access control list defined in the acl_groups queue attribute.	*Boolean*	R, E		When set to *True*, group access to the queue is limited according to the group access control list.	*False*; all groups allowed access	bool	r	r, w	r, w
acl_groups List of groups which are allowed or denied access to this queue. The groups in the list are groups on the server host, not submitting hosts. List is evaluated left-to-right; first match in list is used.	*String*. Syntax: *[+|-] <group name>[, ...]*	R, E			No default	pbs.acl	r	r, w	r, w
acl_host_enable Controls whether host access to the queue obeys the access control list defined in the acl_hosts queue attribute.	*Boolean*	R, E		When set to *True*, host access to the queue is limited according to the host access control list.	*False*; all hosts allowed access.	bool	r	r, w	r, w
acl_hosts List of hosts from which jobs may be submitted to this queue. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *[+|-]<hostname>[. ...]*	R, E			No default	pbs.acl	r	r, w	r, w
acl_user_enable Controls whether user access to the queue obeys the access control list defined in the acl_users queue attribute.	*Boolean*	R, E		When set to *True*, user access to the queue is limited according to the user access control list.	*False*; all users allowed access	bool	r	r, w	r, w
acl_users List of users allowed or denied access to this queue. List is evaluated left-to-right; first match in list is used.	*String.* Syntax: *[+|-]<username> [@<hostname>][, ...]*	R, E			No default	pbs.acl	r	r, w	r, w
alt_router No longer used.							-	-	-
backfill_depth Specifies backfilling behavior for this queue. Sets the number of jobs that are to be backfilled around in this queue. Overrides backfill_depth server attribute. Recommendation: set this to less than 100.	*Integer.* Must be >=0.	E	*>=0*	PBS backfills around the specified number of jobs.	Unset. When unset, backfill depth is *1*	int	r, w	r, w	r, w
			Unset	Backfill depth is set to *1*					

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
checkpoint_min Minimum number of minutes of CPU time or walltime allowed between checkpoints of a job. If a user specifies a time less than this value, this value is used instead. The value given in checkpoint_min is used for both CPU minutes and walltime minutes.	*Integer*	E			No default	pbs.duration	r	r, w	r, w
default_chunk The list of resources which will be inserted into each chunk of a job's select specification if the corresponding resource is not specified by the user. This provides a means for a site to be sure a given resource is properly accounted for even if not specified by the user.	*String.* Syntax: *default_chunk.<resource name>=<value>[, default_chunk.<resource name>=<value>, ...]*	E			No default	pbs.pbs_resource Syntax: *default_chunk["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
enabled Specifies whether this queue accepts new jobs.	*Boolean*	R, E	*True*	This queue is *enabled*. This queue accepts new jobs; new jobs can be enqueued.	*False*	bool	r	r, w	r, w
			False	This queue does not accept new jobs.					
from_route_only Specifies whether this queue accepts jobs only from routing queues, or from both execution and routing queues.	*Boolean*	R, E	*True*	This queue accepts jobs only from routing queues.	*False*	bool	r	r	r, w
			False	This queue accepts jobs from both execution and routing queues as well as directly from submitter.					
hasnodes Indicates whether vnodes are associated with this queue. Set by PBS.	*Boolean*	E		This attribute is set to *True* if there are vnodes associated with this queue.	*False*; no vnodes are associated with this queue	bool	r	r	r, i

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
kill_delay The time delay between sending SIGTERM and SIGKILL when a `qdel` command is issued against a running job.	*Integer*. Units: *Seconds*. Must be greater than or equal to *zero*.	E			*10 seconds*	pbs.duration	r	r, w	r, w
max_array_size The maximum number of subjobs that are allowed in an array job.	*Integer*	R, E			No default	int	r	r, w	r, w
max_group_res Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single group may consume in a complex.	*String.* Syntax: *max_group_res.<resource name>=<value>* Example: `set queue workq max_group_res.ncpus=6`	E	Any PBS resource, e.g. "ncpus", "mem", "pmem", etc.		No default	pbs.pbs_resource Syntax: *max_group_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_group_res_soft Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single group may consume in a complex. If a group is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	*String.* Syntax: *max_group_res_soft.<resource name>=<value>* Example: `set queue workq max_group_res_soft.ncpus=3`	E	Any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc.		No default	pbs.pbs_resource Syntax: *max_group_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_group_run Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by users in a single group that are allowed to be running from this queue at one time.	*Integer*	E			No default	int	r	r, w	r, w
max_group_run_soft Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by users in a single group that are allowed to be running from this queue at one time. If a group has more than this number of jobs running, their jobs are eligible to be preempted by jobs from groups who are not over their soft limit.	*Integer*	E			No default	int	r	r, w	r, w

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
max_queuable Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs allowed to reside in this queue at any given time.	*Integer*	R, E			No default (no limit)	int	r	r, w	r, w
max_queued Limit attribute. The maximum number of jobs allowed to be queued in or running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353	R, E			No default	pbs.pbs_resource Syntax: *max_queued["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_queued_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued in or running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353 Syntax: *max_queued_res.<resource name>=<value>*	R, E			No default	pbs.pbs_resource Syntax: *max_queued_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_run Limit attribute. The maximum number of jobs allowed to be running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: *Limit specification*. See Chapter 7, "Formats", on page 353	E			No default	pbs.pbs_resource Syntax: *max_run["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_run_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 353. Syntax: *max_run_res.<resource name>=<value>*	E			No default	pbs.pbs_resource Syntax: *max_run_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
max_run_res_soft Limit attribute. Soft limit on the amount of the specified resource allowed to be allocated to jobs running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Format: Limit specification. See Chapter 7, "Formats", on page 353. Syntax: *max_run_res_soft.<resource name>=<value>*	E			No default	pbs.pbs_resource Syntax: *max_run_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_run_soft Limit attribute. Soft limit on the number of jobs allowed to be running from this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.	E			No default	pbs.pbs_resource Syntax: *max_run_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_running Old limit attribute. Incompatible with new limit attributes.For an execution queue, this is the largest number of jobs allowed to be running at any given time. For a routing queue, this is the largest number of jobs allowed to be transiting from this queue at any given time.	*Integer*	R, E			No default	int	r	r, w	r, w
max_user_res Old limit attribute. Incompatible with new limit attributes. The maximum amount of the specified resource that any single user may consume.	*String.* Syntax: *max_user_res.<resource name>=<value>* Example: `set queue workq max_user_res.ncpus=6`	E	any PBS resource, e.g. "ncpus", "mem", "pmem", etc		No default	pbs.pbs_resource Syntax: *max_user_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
max_user_res_soft Old limit attribute. Incompatible with new limit attributes. The soft limit on the amount of the specified resource that any single user may consume. If a user is consuming more than this amount of the specified resource, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	*String.* Syntax: *max_user_res_soft.<resource name>=<value>* Example: `set queue workq max_user_res_soft.ncpus=3`	E	any valid PBS resource, e.g. "ncpus", "mem", "pmem", etc		No default	pbs.pbs_resource Syntax: *max_user_res_soft["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
max_user_run Old limit attribute. Incompatible with new limit attributes. The maximum number of jobs owned by a single user that are allowed to be running from this queue at one time.	*Integer*	E			No default	int	r	r, w	r, w
max_user_run_soft Old limit attribute. Incompatible with new limit attributes. The soft limit on the number of jobs owned by any single user that are allowed to be running from this queue at one time. If a user has more than this number of jobs running, their jobs are eligible to be preempted by jobs from users who are not over their soft limit.	*Integer*	E			No default	int	r	r, w	r, w
node_group_key Specifies the resources to use for placement sets (node grouping). Overrides server's node_group_key attribute. Specified resources must be of type string_array.	*String_array.* Syntax: Comma-separated list of resource names. When specifying multiple resources, enclose value in double quotes.	R, E			No default	pbs.node_group_key	r	r, w	r, w
partition Name of partition to which this queue is assigned. Cannot be set for routing queue. An execution queue cannot be changed to a routing queue while this attribute is set.	*String*	E			No default	str	r	r	r, w
Priority The priority of this queue compared to other queues of the same type in this PBS complex. Priority can define a queue as an express queue. See preempt_queue_prio in Chapter 4, "Scheduler Parameters", on page 249. Used for execution queues only; the value of Priority has no meaning for routing queues.	*Integer*	E	Valid values: *-1024 to 1023*		No default	int	r	r, w	r, w
queued_jobs_threshold Limit attribute. The maximum number of jobs allowed to be queued in this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353.	R, E			No default	pbs.pbs_resource Syntax: *queued_jobs_threshold["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
queued_jobs_threshold_res Limit attribute. The maximum amount of the specified resource allowed to be allocated to jobs queued in this queue. Can be specified for projects, users, groups, or all. Cannot be used with old limit attributes.	Limit specification. See Chapter 7, "Formats", on page 353. Syntax: *"queued_jobs_threshold_res.<resource name>=<value>"*	R, E			No default	pbs.pbs_resource Syntax: *queued_jobs_threshold_res["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
queue_type The type of this queue. This attribute must be explicitly set at queue creation.	*String*	R, E	*"e", "execution*	Execution queue	No default	PBS queue type constant: pbs.QUEUETYPE_EXECUTION	r	r, w	r, w
			"r", "route"	Routing queue		PBS queue type constant: pbs.QUEUETYPE_ROUTE			
require_cred **Obsolete** (2020.1) Specifies the credential type required. All jobs submitted to the named queue without the specified credential will be rejected.	*String*	R, E	*krb5*		unset	str	r	r	r, w
			dce						
require_cred_enable **Obsolete** (2020.1) Specifies whether the credential authentication method specified in the require_cred queue attribute is required for this queue.	*Boolean*	R, E		When set to *True*, the credential authentication method is required.	*False*	bool	r	r	r, w
resources_assigned The total for each kind of resource allocated to running and exiting jobs in this queue.	*String.* Syntax: *resources_assigned.<resource name>=<value><newline>resources_assigned.<resource name>=<value><newline>...*	E			No default	pbs.pbs_resource Syntax: *resources_assigned["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r	r

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
resources_available The list of resources and amounts available to jobs running in this queue. The sum of the resource of each type used by all jobs running from this queue cannot exceed the total amount listed here.	*String.* Syntax: *resources_available.<resource name>=<value><newline> resources_available.<resource name>=<value><newline>...*	E			No default	pbs.pbs_resource Syntax: *resources_available["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
resources_default The list of default resource values which are set as limits for a job residing in this queue and for which the job did not specify a limit. If not set, the default limit for a job is determined by the first of the following attributes which is set: server's resources_default, queue's resources_max, server's resources_max. If none of these is set, the job gets unlimited resource usage.	*String.* Syntax: *resources_default.<resource name>=<value>, resources_default.<resource_name>=<value>, ...*	R, E			No default	pbs.pbs_resource Syntax: *resources_default["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
resources_max The maximum amount of each resource that can be requested by a single job in this queue. This queue value supersedes any server wide maximum limit.	*String.* Syntax: *resources_max.<resource name>=<value>, resources_max.<resource name>=<value>, ...*	R, E			No default; infinite usage	pbs.pbs_resource Syntax: *resources_max["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w
resources_min The minimum amount of each resource that can be requested by a single job in this queue.	*String.* Syntax: *resources_max.<resource_name>=<value>, resources_max.<resource name>=<value>, ...*	R, E			No default; zero usage	pbs.pbs_resource Syntax: *resources_min["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Queue Attributes									
Name Description	Format	Queue Type	Value or Option	Value or Option Description	Default Value	Python Type	User	Oper	Mgr
route_destinations The list of destinations to which jobs may be routed. Must be set to at least one valid destination.	*String.* Syntax: comma-separated strings: *<queue name> [@<server host> [:port]]* Example: `Q1, Q2@remote, Q3@remote:15501`	R			No default	pbs.route_destinations	r	r	r, w
route_held_jobs Specifies whether jobs in the held state can be routed from this queue.	*Boolean*	R		When *True*, jobs with a hold can be routed from this queue.	*False*	bool	r	r, w	r, w
route_lifetime The maximum time a job is allowed to reside in this routing queue. If a job cannot be routed in this amount of time, the job is aborted.	*Integer.* Units: *Seconds*	R	*>0*	Jobs can reside for specified number of seconds	Unset; infinite	pbs.duration	r	r, w	r, w
			0	Infinite					
			unset	Infinite					
route_retry_time Time delay between routing retries. Typically used when the network between servers is down.	*Integer.* Units: *Seconds*	R			*30 seconds*	pbs.duration	r	r, w	r, w
route_waiting_jobs Specifies whether jobs whose Execution_Time attribute value is in the future can be routed from this queue.	*Boolean*	R		When *True*, jobs with a future Execution_Time attribute can be routed from this queue.	*False*	bool	r	r, w	r, w
started If this is an execution queue, specifies whether jobs in this queue can be scheduled for execution, or if this is a routing queue, whether jobs can be routed.	*Boolean*	R, E		When *True*, jobs in this queue can run or be routed	*False*	bool	r	r, w	r, w
state_count The number of jobs in each state currently residing in this queue.	*String.* Syntax: *transiting=<value>, exiting=<value>, ...*	R, E			No default	pbs.state_count	r	r	r
total_jobs The number of jobs currently residing in this queue.	*Integer*	R, E			No default	int	r	r	r

6.10 Vnode Attributes

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
comment Information about this vnode. This attribute may be set by the manager to any string to inform users of any information relating to the node. If this attribute is not explicitly set, the PBS server will use the attribute to pass information about the node status, specifically why the node is down. If the attribute is explicitly set by the manager, it will not be modified by the server.	*String* Limit: 80 characters			No default	str	r	r	r, w
current_aoe The AOE currently instantiated on this vnode. Case-sensitive. Cannot be set on server's host.	*String*			Unset	str	r	r	r, w
current_eoe Current value of eoe on this vnode. We do not recommend setting this attribute manually.	*String*			Unset	str	r	r	r, w
in_multivnode_host Specifies whether a vnode is part of a multi-vnoded host. Used internally. Do not set.	*Integer*	*Unset*	Not part of a multi-vnode host		int			r, w
		1	Part of a multi-vnode host					
jobs List of jobs running on this vnode.	*String*. Syntax: *<processor number>/<job ID>, ...*				str	r	r	r
last_state_change_time Records the most recent time that this node changed state.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r, w	r, w

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
last_used_time Records the most recent time that this node finished being used for a job or reservation. Set at creation or reboot time. Updated when node is released early from a running job. Reset when node is ramped up.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			Time of vnode creation or node reboot.	int	r	r, w	r, w
license Indicates whether this vnode is licensed. Set by PBS.	*Character*	*l*	This vnode is licensed.	Unset	str	r	r	r
license_info Number of licenses assigned to this vnode. Set by PBS.	*Integer*			Unset	int	r	r	r
lictype No longer used.					none	-	-	-
maintenance_jobs List of jobs that were running on this vnode, but have been suspended via the *admin-suspend* signal to `qsig`. Set by server.	*String_array*			No default	str	-	-	r
Mom Hostname where server queries for MoM host. By default the server queries the canonicalized name of the MoM host, unless you set this attribute when you create the vnode. Can be explicitly set by Manager only via `qmgr`, and only at vnode creation. The server can set this to the FQDN of the host on which MoM runs, if the vnode name is the same as the hostname.	*String*		.	Value of vnode resource (vnode name)	str	r	r	r, w
name The name of this vnode.	*String*			No default	str	r	r	r, w

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
no_multinode_jobs Controls whether jobs which request more than one chunk are allowed to execute on this vnode. Used for cycle harvesting.	*Boolean*		When set to *True*, jobs requesting more than one chunk are not allowed to execute on this vnode	*False*	bool	r	r	r, w
ntype The type of this vnode.	*String*	*PBS*	Normal vnode	*PBS*	pbs.ND_PBS	r	r	r, w
partition Name of partition to which this vnode is assigned. A vnode can be assigned to at most one partition.	*String*			No default	str	r	r, w	r, w
pbs_version The version of PBS for this MoM	*String*			No default	str	r	r	r
pcpus **Deprecated**. The number of physical CPUs on this vnode. This is set to the number of CPUs available when MoM starts. For a multiple-vnode MoM, only the parent vnode has pcpus.	*Integer*			Number of CPUs on start-tup	int	r	r	r
pnames The list of resources being used for placement sets. Not used for scheduling; advisory only.	*String*. Syntax: comma-separated list of resource names.			No default	str	r	r	r, w
Port Port number on which MoM daemon listens. Can be explicitly set only via `qmgr`, and only at vnode creation.	*Integer*			*15002*	int	-	r, w	r, w
poweroff_eligible Enables powering this vnode up and down by PBS.	*Boolean*	*True* *False*	PBS can power this vnode on and off. PBS cannot power this vnode on and off.	*False*	bool	r	r	r, w
power_provisioning Specifies whether this node is eligible to have its power managed by PBS, including whether it can use power profiles.	*Boolean*	*True* *False*	Power provisioning is enabled at this vnode. Power provisioning is disabled at this vnode.	*False*	bool	r	r	r, w

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Priority The priority of this vnode compared with other vnodes.	*Integer*	*[-1024, +1023]* inclusive		No default	int	r	r, w	r, w
provision_enable Controls whether this vnode can be provisioned. Cannot be set on server's host.	*Boolean*	*True*	This vnode may be provisioned.	*False*	bool	r	r	r, w
		False	This vnode may not be provisioned.					
queue **Deprecated**. The queue with which this vnode is associated. Each vnode can be associated with at most 1 queue. Queues can be associated with multiple vnodes. Any jobs in a queue that has associated vnodes can run only on those vnodes. If a vnode has an associated queue, only jobs in that queue can run on that vnode.	*String*	*<name of queue>*	Only jobs in specified queue may run on this vnode.	No default	pbs.queue	r	r	r, w
		Unset	Any job in any queue that does not have associated vnodes can run on this vnode.					
resources_assigned The total amount of each resource allocated to running and exiting jobs and started reservations on this vnode.	*String*. Syntax: *resources_assigned.<resource name>=<value>[,resources_assigned.<resource name>=<value>*			No default	pbs.pbs_resource Syntax: *resources_assigned['<resource name>'] = <val>* where *resource name* is any built-in or custom resource	r	r	r
resources_available The list of resources and the amounts available on this vnode. If not explicitly set, the amount shown is that reported by the `pbs_mom` running on this vnode. If a resource value is explicitly set, that value is retained across restarts.	*String*. Syntax: *resources_available.<resource name>=<value>, resources_available.<resource name> = <value>, ...*			No default	pbs.pbs_resource Syntax: *resources_available['<resource name>'] = <val>* where *resource name* is any built-in or custom resource	r	r, w	r, w

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
resv List of advance and standing reservations pending on this vnode.	*String*. Comma-separated list of reservation IDs. Syntax: *<reservation ID>[, <reservation ID>, ...]*			No default	str	r	r	r
resv_enable Controls whether the vnode can be used for advance and standing reservations. Reservations are incompatible with cycle harvesting.	*Boolean*		When set to *True*, this vnode can be used for reservations. Existing reservations are honored when this attribute is changed from *True* to *False*.	*True*	bool	r	r	r, w
sharing Specifies whether more than one job at a time can use the resources of the vnode or the vnode's host. Either (1) the vnode or host is allocated exclusively to one job, or (2) the vnode's or host's unused resources are available to other jobs. Can be set in the cgroups hook's configuration file or by using `pbs_mom -s insert`. Behavior of a vnode or host is determined by a combination of the sharing attribute and a job's placement directive, defined as follows:	*String*. Example: `vnodename: sharing=force_excl`	*default_shared*	Defaults to *shared*	*default_shared*	pbs.ND_DEFAULT_SHARED	r	r, w	r, w
		default_excl	Defaults to *exclusive*		pbs.ND_DEFAULT_EXCL			
		default_exclhost	Entire host is assigned to the job unless the job's sharing request specifies otherwise		pbs.ND_DEFAULT_EXCLHOST			
		ignore_excl	Overrides any job *place=excl* setting		pbs.ND_IGNORE_EXCL			
		force_excl	Overrides any job *place=shared* setting		pbs.ND_FORCE_EXCL			
		force_exclhost	The entire host is assigned to the job, regardless of the job's sharing request		pbs.ND_FORCE_EXCLHOST			
		Unset	Defaults to *shared*					

Vnode Attributes

Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr

Behavior of vnode:

Value of sharing	Placement Request (`-lplace=`)				
	Vnode			Host	
	not specified	place=*shared*	place=*excl*	place=*exclhost*	place**!=***exclhost*
not set	shared	shared	exclusive	exclusive	depends on place
default_shared	shared	shared	exclusive	exclusive	depends on place
default_excl	exclusive	shared	exclusive	exclusive	depends on place
default_exclhost	exclusive	shared	exclusive	exclusive	depends on place
ignore_excl	shared	shared	shared	shared	not exclusive
force_excl	exclusive	exclusive	exclusive	exclusive	not exclusive
force_exclhost	exclusive	exclusive	exclusive	exclusive	exclusive

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
state Shows or sets the state of the vnode.	*String.* Comma-separated list of one or more states: *<state>[, <state>, ...]*	*busy*	Vnode is reporting load average greater than allowed max. Can combine with *offline*.	No default	int	r	r	r
		down	Node is not responding to queries from the server. Cannot be combined with *free*, *provisioning*			r	r	r
		free	Vnode is up and capable of accepting new job(s). Cannot be combined with other states.			r	r	r
		job-busy	All CPUs on the vnode are allocated to jobs. Can combine with: *offline*, *resv_exclusive*.			r	r	r
		job-exclusive	Entire vnode is exclusively allocated to one job at the job's request. Can combine with *offline*, *resv_exclusive*			r	r	r
		offline	Jobs are not to be assigned to this vnode. Can combine: *busy, job-busy, job-exclusive, resv_exclusive*.			r	r, w	r, w
		provisioning	Vnode is being provisioned. Cannot be combined with any other states.			r	r	r
		resv-exclusive	Running reservation has requested exclusive use of vnode. Can combine with *job-exclusive*, *offline*			r	r	r
		stale	Vnode was previously reported to server, but is no longer reported to server. Cannot combine with *free*, *provisioning*			r	r	r
		state-unknown	The server has never been able to contact the vnode. Either MoM is not running on the vnode, the vnode hardware is down, or there is a network problem.			r	r	r
		unresolvable	The server cannot resolve the name of the vnode.			r	r	r
		wait-provisioning	Vnode needs to be provisioned, but can't: limit reached for concurrent provisioning vnodes. Cannot be combined with other states. See max_concurrent_provision.			r	r	r
topology_info Contains information intended to be used in hooks. Visible in and usable by hooks only.	*XML string*			*Unset*	str	-	-	-

Vnode Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
vnode_pool Cray only. Allows just one MoM, instead of all, to report inventory upon startup, allowing faster startup and less network communication between server and non-reporting MoMs. On each Cray, all MoMs must have same setting for this attribute. Can be set only at vnode creation; valid only on login nodes running a MoM. Not supported on non-Cray machines.	*Integer*	*0*	Unset; each MoM reports inventory separately	*0* (Unset)	int	r	r	r, w
		>0	Only one MoM per Cray reports inventory					

6.11 Job Attributes

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Account_Name String used for accounting purposes. Can be used for fairshare.	*String*. Can contain any character.			No default	str	r, w	r, w	r, w
accounting_id Accounting ID for tracking accounting data not produced by PBS.	*String*			No default	str	r	r	r
accrue_type Indicates what kind of time the job is accruing.	*Integer*	*0 (initial_time)*	Job is accruing initial time. Can occur when job is blocked by a runjob hook.	2 (eligible_time)	int	-	-	r
		1 (ineligible_time)	Job is accruing ineligible time. Occurs when job or owner has hit limit.					
		2 (eligible_time)	Job is accruing eligible time. Occurs when job is blocked on resources.					
		3 (run_time)	Job is accruing run time. Occurs when job is running.					
alt_id For a few systems, the session ID is insufficient to track which processes belong to the job. Where a different identifier is required, it is recorded in this attribute. If set, it is also recorded in the end-of-job accounting record. On Windows, holds PBS home directory.	*String.* May contain white spaces.			No default	str	r	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
argument_list Job executable's argument list. Shown if job is submitted with "`-- <executable> [<argument list>]`"	*JSDL-encoded string*. *<jsdl-hpcpa:Argument> <1st arg> </jsdl-hpcpa:Argument>* *<jsdl-hpcpa:Argument> <2nd arg> </jsdl-hpcpa:Argument>* *<jsdl-hpcpa:Argument> <nth arg> </jsdl-hpcpa:Argument>* Example: if arguments are "A B": `<jsdl-hpcpa:Argument>A</jsdl-hpcpa:Argument> <jsdl-hpcpa:Argument>B</jsdl-hpcpa:Argument>`			No default	str	r, w	r, w	r, w
array Indicates whether this is a job array.	*Boolean*		Set to *True* if this is an array job.	*False*	bool	r, s	r	r
array_id Applies only to subjobs. Array identifier of subjob.	*String*			No default	str	r	r	r
array_index Applies only to subjobs. Index number of subjob.	*String*			No default	int	r	r	r
array_indices_remaining Applies only to job arrays. List of indices of subjobs still queued.	*String.* Range or list of ranges, e.g. `500, 552, 596-1000`.			No default	str	r	r	r
array_indices_submitted Applies only to job arrays. Complete list of indices of subjobs given at submission time.	*String*. Given as range, e.g. `1-100`			No default	pbs.range	r, s	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
array_state_count Applies only to job arrays. Lists number of subjobs in each state.	*String*			No default	pbs.state_count	r	r	r
block Specifies whether `qsub` will wait for the job to complete and return the exit value of the job. For X11 forwarding jobs, and jobs with interactive and/or block attributes set to *True*, the job's exit status is not returned.	*Boolean*			*False*	int	r, s	r	r
Checkpoint Determines when the job will be checkpointed. An $action script is required to checkpoint the job.	*String*	c	Checkpoint at intervals, measured in CPU time, set on job's execution queue. If no interval set at queue, job is not checkpointed.	*u*	pbs.checkpoint	r, w	r, w	r, w
		c = *<minutes of CPU time>*	Checkpoint at intervals of specified number of minutes of job CPU time. This value must be > 0. If interval specified is less than that set on job's execution queue, queue's interval is used. Format: *Integer*					
		w	Checkpoint at intervals, measured in walltime, set on job's execution queue. If no interval set at queue, job is not checkpointed.					
		w = *<minutes of walltime>*	Checkpoint at intervals of the specified number of minutes of job walltime. This value must be greater than zero. If the interval specified is less that that set on job's execution queue, the queue's interval is used. Format: *Integer*					
		n	No checkpointing.					
		s	Checkpoint only when the server is shut down.					
		u	Unset. Defaults to behavior when *interval* argument is set to *s*.					

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
comment Comment about job. Informational only.	*String*			No default	str	r	r, w	r, w
create_resv_from_job When this job is run, immediately creates and confirms a *job-specific start reservation* on the same resources as the job (including resources inherited by the job), and places the job in the job-specific reservation's queue. Sets the job's create_resv_from_job attribute to *True*. Sets the job-specific reservation's reserve_job attribute to the ID of the job from which the reservation was created. The new reservation's duration and start time are the same as the job's walltime and start time. If the job is peer scheduled, the job-specific reservation is created in the pulling complex.	*Boolean*	*False* *True*	Does not create a reservation. Creates the job-specific start reservation.	*False*	bool	r, w	r, w	r, w
ctime Timestamp; time at which the job was created.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
depend Specifies inter-job dependencies. No limit on number of dependencies.	*String*. Syntax: *<type>:<job ID>[,<job ID> ...],[<type>:<job ID>[,<job ID> ...] ...]* Must be enclosed in double quotes if it contains commas. Example: `"before:123,456"`	after:<job ID list>	This job may run at any point after all jobs in *job ID list* have started execution.	No default; no dependencies	pbs.depend	r, w	r, w	r, w
		afterok:<job ID list>	This job may run only after all jobs in *job ID list* have terminated with no errors.					
		afternotok:<job ID list>	This job may run only after all jobs in *job ID list* have terminated with errors.					
		afterany:<job ID list>	This job can run after all jobs in *job ID list* have finished execution, with or without errors. This job will not run if a job in the *job ID list* was deleted without ever having been run.					
		before:<job ID list>	Jobs in *job ID list* may start once this job has started.					
		beforeok:<job ID list>	Jobs in *job ID list* may start once this job terminates without errors.					
		beforenotok:<job ID list>	If this job terminates execution with errors, jobs in *job ID list* may begin.					
		beforeany:<job ID list>	Jobs in *job ID list* may begin execution once this job terminates execution, with or without errors.					
		on:<count>	This job may run after *count* dependencies on other jobs have been satisfied. This type is used with one of the *before* types listed. *Count* is an integer greater than *0*.					
egroup If the job is queued, this attribute is set to the group name under which the job is to be run.	*String*			No default	str	-	-	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
eligible_time The amount of wall clock wait time a job has accrued while the job is blocked waiting for resources. For a job currently accruing *eligible_time*, if we were to add enough of the right type of resources, the job would start immediately. Viewable via `qstat -f`.	*Duration*			*Zero*	pbs.duration	r	r, w	r, w
Error_Path The final path name for the file containing the job's standard error stream. See the qsub and qalter commands.	*String*. Syntax: *[<hostname>:]<path>*	*<relative path>*	Path is relative to the current working directory of command executing on current host.	Default path is current working directory where `qsub` is run. If the output path is specified, but does not include a filename, the default filename is *<job ID>.ER*. If the path name is not specified, the default filename is *<job name>.e<sequence number>*.	str	r, w	r, w	r, w
		<absolute path>	Path is absolute path on current host where command is executing.					
		<hostname>:<relative path>	Path is relative to user's home directory on specified host.					
		<hostname>:<absolute path>	Path is absolute path on named host.					
		No path	Path is current working directory where `qsub` is executed.					

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
estimated List of estimated values for job. Used to report job's exec_vnode, start_time, and soft_walltime. Can be set in a hook or via `qalter`, but PBS will overwrite the values.	Syntax: *estimated.<resource name>=<value>, estimated.<resource name>=<value>.* *exec_vnode* is a *string*. *soft_walltime* is a *duration*. *start_time* is printed by `qstat` in human-readable *Date* format; *start_time* is output in hooks as seconds since epoch.	*exec_vnode*	The estimated vnodes used by this job.	Unset	pbs.pbs_resource Syntax: *estimated.[<resource name>]=<value>*. *exec_vnode* is a *pbs.exec_vnode*. *soft_walltime* is a *duration*. *start_time* is an *int*.	r	r, w	r, w
		soft_walltime	The estimated soft walltime for this job. Calculated when a job exceeds its soft_walltime resource.	Unset		r	r	r, w
		start_time	The estimated start time for this job.	Unset		r	r, w	r, w
etime Timestamp; time when job became eligible to run, i.e. was enqueued in an execution queue and was in the "*Q*" state. Reset when a job moves queues, or is held then released. Not affected by qaltering.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r
euser If the job is queued, this attribute is set to the user name under which the job is to be run.	*String*			No default	str	-	-	r
executable JSDL-encoded listing of job's executable. Shown if job is submitted with "`-- <executable> [<arg list>]`".	*JSDL-encoded string.* *<jsdl-hpcpa:Executable> <name of executable>* Example: if the executable is `ping`: `<jsdl-hpcpa:Executable>ping</jsdl-hpcpa:Executable>`			No default	str	r, w	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Execution_Time Timestamp; time after which the job may execute. Before this time, the job remains queued in the (*W*)ait state. Can be set when stage-in fails and PBS moves job start time out 30 minutes to allow user to fix problem.	*Datetime*. See Chapter 7, "Formats", on page 353.			Unset; no delay	int	r, w	r, w	r, w
exec_host If the job is running, this is set to the name of the host or hosts on which the job is executing.	*String*. Syntax: *<host-name>/N[*C][+...]*, where *N* is task slot number starting at 0, on that host, and *C* is the number of CPUs allocated to the job. **C* does not appear if its value is *1*.			No default	pbs.exec_host	r	r, i	r, i
exec_vnode List of chunks for the job. Each chunk shows the name of the vnode(s) from which it is taken, along with the host-level, consumable resources allocated from that vnode, and any AOE provisioned on this vnode for this job. If a vnode is allocated to the job but no resources from the vnode are used by the job, the vnode name appears alone. If a chunk is split across vnodes, the name of each vnode and its resources appear inside one pair of parentheses, joined with a plus ("+") sign.	Each chunk is enclosed in parentheses. Chunks are connected by plus signs. Example: For a job which requested two chunks satisfied by resources from three vnodes, exec_vnode is: `(vnodeA:ncpus=N:mem=X)+(nodeB:ncpus=P:mem=Y+nodeC:mem=Z)`. For a job which requested one chunk and exclusive use of a 2-vnode host, where the chunk was satisfied by resources from one vnode, exec_vnode is `(vnodeA:ncpus=N:mem=X)+(vnodeB)`.			No default	pbs.exec_vnode	r	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Exit_status Exit status of job. Set to zero for successful execution. If any subjob of an array job has non-zero exit status, the array job has non-zero exit status.	*Integer*			No default	int	r	r	r
forward_x11_cookie Contains the X authorization cookie.	*String*			No default	str	r	r	r
forward_x11_port Contains the number of the port being listened to by the port forwarder on the submission host.	*Integer*			No default	int	r	r	r
group_list A list of group names used to determine the group under which the job runs. When a job runs, the server selects a group name from the list according to the following ordered set of rules: 1. Select the group name for which the associated host name matches the name of the server host. 2. Select the group name which has no associated host name. 3. Use the login group for the user name under which the job will be run.	*String*. Syntax: *<group name>[@<host-name>] [,<group name>[@<host-name>]...]* Must be enclosed in double quotes if it contains commas.			No default	pbs.group_list	r, w	r, w	r, w
hashname No longer used.						-	-	-
Hold_Types The set of holds currently applied to the job. If the set is not null, the job will not be scheduled for execution and is said to be in the *held* state. The *held* state takes precedence over the *wait* state.	*String*, made up of the letters 'n', 'o', 'p', 's', 'u'	*n*	No hold	*n*	pbs.hold_types	r, w	r, w	r, w
		o	Other hold					
		p	Bad password					
		s	System hold					
		u	User hold					

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
interactive Specifies whether the job is interactive. Can be set, but not altered, by unprivileged user. When both this attribute and the block attribute are *True*, no exit status is returned. For X11 forwarding jobs, the job's exit status is not returned. Cannot be set using a PBS directive. Job arrays cannot be interactive.	*Boolean*		Set to *True* if this is an interactive job.	*False*	int	r, w	r	r
jobdir Path of the job's staging and execution directory on the primary execution host. Either user's home, or private sandbox. Depends on value of sandbox attribute. Viewable via `qstat -f`.	*String*			No default	str	r	r	r
Job_Name The job name. See the `qalter` and `qsub` commands.	*String* up to 236 characters, first character must be alphabetic or numeric			Base name of job script, or STDIN	str	r, w	r, w	r, w
Job_Owner The login name on the submitting host of the user who submitted the batch job.	*String.* Syntax: *<Username>@<submission host>*			No default	str	r	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
job_state The state of the job.	*Character*	*B (Begun)*	Job arrays only. Job array has begun execution.	No default	pbs.JOB_STATE_BEGUN	r, i	r, i	r, i
		E (Exiting)	The job has finished, with or without errors, and PBS is cleaning up post-execution.		pbs.JOB_STATE_EXITING			
		F (Finished)	Job is finished. Job has completed execution, job failed during execution, or job was deleted.		pbs.JOB_STATE_FINISHED			
		H (Held)	The job is held.		pbs.JOB_STATE_HELD			
		M (Moved)	The job has been moved to another server.		pbs.JOB_STATE_MOVED			
		Q (Queued)	The job resides in an execution or routing queue pending execution or routing. It is not in held or waiting state.		pbs.JOB_STATE_QUEUED			
		R (Running)	The job is in an execution queue and is running.		pbs.JOB_STATE_RUNNING			
		S (Suspended)	The job was executing and has been suspended. The job does not use CPU cycles or walltime.		pbs.JOB_STATE_SUSPEND			
		T (Transiting)	The job is being routed or moved to a new destination.		pbs.JOB_STATE_TRANSIT			
		U (User suspended)	The job was running on a workstation configured for cycle harvesting and the keyboard/mouse is currently busy. The job is suspended until the workstation has been idle for a configured amount of time.		pbs.JOB_STATE_SUSPEND_USERACTIVE			
		W (Waiting)	The Execution_Time attribute contains a time in the future. Can be set when stage-in fails and PBS moves job start time out 30 minutes to allow user to fix problem.		pbs.JOB_STATE_WAITING			
		X (Expired)	Subjobs only. Subjob is finished (expired.)		pbs.JOB_STATE_EXPIRED			

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Join_Path Specifies whether the job's standard error and standard output streams are to be merged and placed in the file specified in the Output_Path job attribute.	*String* One of "*oe*", "*eo*", or "*n*".	*eo*	Standard output and standard error are merged, intermixed, into a single stream, which becomes standard error.	*n*	pbs.join_path	r, w	r, w	r, w
		oe	Standard output and standard error are merged, intermixed, into a single stream, which becomes standard output.					
		n	Standard output and standard error are not merged.					
Keep_Files Specifies whether the standard output and/or standard error streams are retained on the execution host in the job's staging and execution directory after the job has executed. Otherwise these files are returned to the submission host. Keep_Files overrides the Output_Path and Error_Path attributes.	*String* One of "*o*", "*e*", "*oe*", "*eo*", or "*n*".	*o*	The standard output stream is retained. The filename is: *<job name>.o<sequence number>*	*n*	pbs.keep_files	r, w	r, w	r, w
		e	The standard error stream is retained. The filename is: *<job name>.e<sequence number>*					
		eo, oe	Both standard output and standard error streams are retained.					
		d	Output and error are written directly to their final destination					
		n	Neither stream is retained. Files are returned to submission host.					
Mail_Points Specifies state changes for which the server sends mail about the job.	*String* Can be any of "*a*", "*b*", "*e*", with optional "*j*", or "*n*".	*a*	Mail is sent when job is aborted	*a*	pbs.mail_points	r, w	r, w	r, w
		b	Mail is sent at beginning of job					
		e	Mail is sent at end of job					
		j	Mail is sent for subjobs. Must be combined with one or more of a, b, and e options					
		n	No mail is sent. Cannot be combined with other options.					
Mail_Users The set of users to whom mail is sent when the job makes state changes specified in the Mail_Points job attribute.	*String* Syntax: *"<username>@<hostname>[,<username>@<hostname>]"* Must be enclosed in double quotes if it contains commas.			Job owner only	pbs.email_list	r, w	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
max_run_subjobs Sets a limit on the number of subjobs that can be running at one time. Can be set using `qsub -J <range> [%<max subjobs>]` or `qalter -Wmax_run_subjobs=<new value> <job ID>`. Suspended subjobs are not counted against this limit.	*Integer*			No default	int	r, w	r, w	r, w
mtime Timestamp; the time that the job was last modified, changed state, or changed locations.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r
no_stdio_sockets Not used.						-	-	-

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Output_Path The final path name for the file containing the job's standard output stream. See the `qsub` and `qalter` commands.	*String*. Syntax: *[<hostname>:]<path>*	*<relative path>*	Path is relative to the current working directory of command executing on current host.	Default path is current working directory where `qsub` is run. If the output path is specified, but does not include a filename, the default filename is *<job ID>.OU*. If the path name is not specified, the default filename is *<job name>.o<sequence number>*.	str	r, w	r, w	r, w
		<absolute path>	Path is absolute path on current host where command is executing.					
		<hostname>:<relative path>	Path is relative to user's home directory on specified host.					
		<hostname>:<absolute path>	Path is absolute path on named host.					
		No path	Path is current working directory where `qsub` is executed.					
pcap_accelerator Power attribute. Power cap for an accelerator. Corresponds to Cray `capmc set_power_cap --accel` setting. See `capmc` documentation.	*Integer* Units*: Watts*			Unset	int	r, w	r, w	r, w
pcap_node Power attribute. Power cap for a node. Corresponds to Cray `capmc set_power_cap --node` setting. See `capmc` documentation.	*Integer* Units*: Watts*			Unset	int	r, w	r, w	r, w
pgov Power attribute. Cray ALPS reservation setting for CPU throttling corresponding to p-governor. See BASIL 1.4 documentation. We do not recommend using this attribute.	*String*			Unset	str	r, w	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Priority The scheduling priority for the job. Higher value indicates greater priority.	*Integer*. Syntax: *[+|-]nnnn*	*[-1024, +1023]* inclusive		Unset	int	r, w	r, w	r, w
project The job's project. A project is a way to tag jobs. Each job can belong to at most one project.	*String*. Can contain any characters except for the following: Slash ("/"), left bracket ("["), right bracket ("]"), double quote (""""), semicolon (";"), colon (":"), vertical bar ("|"), left angle bracket ("<"), right angle bracket (">"), plus ("+"), comma (","), question mark ("?"), and asterisk ("*").			*_pbs_project_default*	str	r, w	r, w	r, w
pset **Deprecated**. Name of placement set used by the job.	*String*				str	r	r	r, w
pstate Power attribute. Cray ALPS reservation setting for CPU frequency corresponding to p-state. See BASIL 1.4 documentation.	*String* Units: *Hertz*			Unset	str	r, w	r, w	r, w
qtime Timestamp; the time that the job entered the current queue.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r
queue The name of the queue in which the job currently resides.	*String*			No default	pbs.queue	r	r	r
queue_rank A number indicating the job's position within its queue. Only used internally by PBS.	*Integer*			No default	int	-	-	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
queue_type The type of queue in which the job is currently residing.	*Character*	*E*	Execution queue	No default	pbs.QTYPE_EXECUTION	-	-	r
		R	Routing queue		pbs.QTYPE_ROUTE			
release_nodes_on_stageout Controls whether job vnodes are released when stageout begins. Cannot be used with vnodes tied to Cray X* series systems. When cgroups is enabled and this is used with some but not all vnodes from one MoM, resources on those vnodes that are part of a cgroup are not released until the entire cgroup is released. The job's stageout attribute must be set for the release_nodes_on_stageout attribute to take effect.	*Boolean*	*True*	All of the job's vnodes not on the primary execution host are released when stageout begins	*False*	bool	r, w	r, w	r, w
		False	Job's vnodes are released when the job finishes and MoM cleans up the job					
Remove_Files Specifies whether standard output and/or standard error files are automatically removed upon job completion.	*String*	*e*	Standard error is removed upon job completion	Unset	str	r, w	r, w	r, w
		o	Standard output is removed upon job completion					
		eo	Standard output and standard error are removed upon job completion					
		oe	Standard output and standard error are removed upon job completion					
		unset	Neither is removed					
Rerunable Specifies whether the job can be rerun. Does not affect how a job is treated if the job could not begin execution. See "Allowing Your Job to be Re-run", on page 118 of the PBS Professional User's Guide. Job arrays are required to be rerunnable and are rerunnable by default.	*Character*	*y*	The job can be rerun.	*y*	bool	r, w	r, w	r, w
		n	Once the job starts running, it can never be rerun.					

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Resource_List The list of resources required by the job. List is a set of <*resource name*>=<*value*> strings. The meaning of name and value is dependent upon defined resources. Each value establishes the limit of usage of that resource. If not set, the value for a resource may be determined by a queue or server default established by the administrator. See Chapter 5, "List of Built-in Resources", on page 259.	*String.* Syntax: *Resource_List.<resource name>=<value>], Resource_List.<resource name>=<value>, ...]*			No default	pbs.pbs_resource Syntax: *Resource_List["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r, w	r, w	r, w
resources_released Listed by vnode, consumable resources that were released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server.	*String.* Syntax: *(<vnode>:<resource name>=<value>:<resource name>=<value>:...)+(<vnode>:<resource name>=<value>:...)*			No default	str	r	r	r
resource_released_list Sum of each consumable resource requested by the job that was released when the job was suspended. Populated only when restrict_res_to_release_on_suspend server attribute is set. Set by server.	*String.* Syntax: *resource_released_list.<resource name>=<value>,resource_released_list.<resource name>=<value>, ...*			No default	pbs.pbs_resource	--	r	r
resources_used The amount of each resource used by the job.	*String.* Syntax: List of *resources_used.<resource name>=<value>,resources_used.<resource name>=<value>* pairs. Example: `resources_used.mem=2mb`			No default	pbs.pbs_resource Syntax: *resources_used["<resource name>"]=<value>* where *resource name* is any built-in or custom resource	r	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
run_count The number of times the server thinks the job or subjob has been executed. The run_count attribute starts at zero. Job is held after 21 tries. When a subjob hits the run_count limit, it and its parent job array get a System hold. Can be set via `qsub`, `qalter`, or a hook.	*Integer.* Must be greater than or equal to *zero*.			*Zero*	int	-	r, w	r, w
run_version Used internally by PBS to track the instance of the job.	*Integer*				int	--	--	r
sandbox Specifies type of location PBS uses for job staging and execution. User-settable via `qsub -Wsandbox=<value>` or via a PBS directive. See the $jobdir_root MoM configuration option in `pbs_mom`.8B.	*String*	*PRIVATE*	PBS creates job-specific staging and execution directories under the directory specified in the $jobdir_root MoM configuration option.	Unset	str	r, w	r, w	r, w
		HOME or unset	PBS will use the job owner's home directory for staging and execution.					
schedselect The union of the select specification of the job, and the queue and server defaults for resources in a chunk.	*String*			No default	pbs.select	-	-	r
sched_hint No longer used.						-	-	-
security_context Contains security context of job submitter. Set by PBS to the security context of the job submitter at the time of job submission. If not present when a request is submitted, an error occurs, a server message is logged, and the request is rejected.	*String in SELinux format*			Unset	str	r	r	r

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
server The name of the server which is currently managing the job. When the secondary server is running during failover, shows the name of the primary server. After a job is moved to another server, either via `qmove` or peer scheduling, shows the name of the new server.	*String*			No default	pbs.server	r	r	r
session_id If the job is running, this is set to the session ID of the first executing task.				No default	int	r	r	r
Shell_Path_List One or more absolute paths to the program(s) to process the job's script file.	*String*. Syntax: *"<path>[@<hostname>][,<path>[@<hostname>]...]"* Must be enclosed in double quotes if it contains commas.			User's login shell on execution host	pbs.path_list	r, w	r, w	r, w
stagein The list of files to be staged in prior to job execution.	*String*. Syntax: *"<execution path>@<storage host>:<storage path>[, <execution path>@<storage host>:<storage path>, ...]"* Must be enclosed in double quotes if it contains commas.			No default	pbs.staging_list	r, w	r, w	r, w
stageout The list of files to be staged out after job execution.	*String*. Syntax: *"<execution path>@<storage host>:<storage path>[, <execution path>@<storage host>:<storage path>, ...]"* Must be enclosed in double quotes if it contains commas.			No default	pbs.staging_list	r, w	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
Stageout_status Status of stageout. If stageout succeeded, this is set to *1*. If stageout failed, this is set to *0*. Available only for finished jobs. Displayed only if set. If stageout fails for any subjob of an array job, the value of Stageout_status is zero for the array job. Available only for finished jobs.	*Integer*			No default	int	r	r	r
stime Timestamp; time when the job started execution. Changes when job is restarted.	*Timestamp.* Printed by `qstat` in human-readable *Date* format. Output in hooks as seconds since epoch.			No default	int	r	r	r
Submit_arguments Job submission arguments given on the `qsub` command line. Available for all jobs.	*String*			No default	str	r, w	r, w	r, w
substate The substate of the job. The substate is used internally by PBS.	*Integer*			No default	int	r	r	r
sw_index No longer used.						-	-	-

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
tolerate_node_failures Specifies whether job can have extra vnodes allocated, and whether for startup only or for the life of the job.	*String*	*none, unset*	No extra vnodes are allocated to the job.	None	str	r, s	r, s	r, s
		job_start	Extra vnodes are allocated only long enough to start the job successfully. Tolerate vnode failures that occur only during job start, just before executing the job's top level shell or executable or any execjob_launch hooks. Failures tolerated are those such as an assigned sister MoM failing to join the job and communication errors between MoMs.					
		all	Extra vnodes are allocated for the life of the job. Tolerate all node failures resulting from communication problems, such as polling problems, between the primary MoM and the sister MoMs assigned to the job Tolerate failures due to rejections from execjob_begin or execjob_prologue hooks run at sister MoMs.					
topjob_ineligible Allows administrators to mark this job as ineligible to be a top job.	*Boolean*	*True*	This job is not eligible to be a top job.	*Unset*, behaves like *False*	bool	-	-	r, w
		False	This job is eligible to be a top job.					
umask The initial umask of the job is set to the value of this attribute when the job is created. This may be changed by `umask` commands in the shell initialization files such as `.profile` or `.cshrc`.	*Decimal integer*			*077*	int	r, w	r, w	r, w

Job Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Def Val	Python Type	Usr	Opr	Mgr
User_List The list of users which determines the user name under which the job is run on a given host. No length limit. When a job is to be executed, the server selects a user name from the list according to the following ordered set of rules: 1. Select the user name from the list for which the associated host name matches the name of the server. 2. Select the user name which has no associated host name; the wild card name. 3. Use the value of Job_Owner as the user name.	*String*. Syntax: *"<username>@<hostname> [,<username>@<hostname>...]"* Must be enclosed in double quotes if it contains commas. May be up to 256 characters in length.			Value of Job_Owner job attribute	pbs.user_list	r, w	r, w	r, w
Variable_List List of environment variables set in the job's execution environment. See the `qsub(1B)` command.	*String*. Syntax: *"<variable name>=<value> [,<variable name>=<value>...]"* Must be enclosed in double quotes if it contains commas.			No default	pbs.pbs_resource Syntax: *Variable_List["<variable name>"]=<value>*	r, w	r, w	r, w

6.12 Hook Attributes

An unset hook attribute takes the default value for that attribute.

Hook attributes can be set by root or the Admin at the local server only.

Hook Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	Usr	Opr	Mgr
alarm Specifies the number of seconds to allow a hook to run before the hook times out.	*Integer.* Must be greater than *zero*.			*30*				
debug Specifies whether or not the hook produces debugging files under `PBS_HOME/server_priv/hooks/tmp` or `PBS_HOME/mom_priv/hooks/tmp`. Files are named *hook_<hook event>_<hook name>_<unique ID>.in*, *.data*, and *.out*. See "Producing Files for Debugging" on page 159 in the PBS Professional Hooks Guide.	*Boolean*	*True*	The hook leaves debugging files when it runs.	*False*				
		False	The hook does not leave debugging files when it runs.					
enabled Determines whether or not a hook is run when its triggering event occurs.	*Boolean*	*True*	Hook runs when triggering event occurs.	*True*				
		False	Hook does not run when triggering event occurs.					

Hook Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	Usr	Opr	Mgr
event List of events that trigger the hook. Can be operated on with the "=", "+=", and "-=" operators. The *provision* event cannot be combined with any other events.	*String_array*	*"queuejob"*	Triggered when job is queued	"" meaning hook is not triggered	str			
		"modifyjob"	Triggered when job is modified					
		"movejob"	Triggered when job is moved					
		"resv_end"	Triggered when reservation ends					
		"resvsub"	Triggered when reservation is created					
		"runjob"	Triggered when job is run					
		"periodic"	Triggered periodically at server					
		"provision"	Hook is master provisioning hook					
		"execjob_begin"	Triggered when MoM receives job					
		"execjob_prologue"	Triggered just before first job process					
		"execjob_launch"	Triggered just before executing user's program					
		"execjob_attach"	Triggered before running any execjob_prologue hooks, on each vnode where `pbs_attach()` runs					
		execjob_postsuspend	Triggered just after successfully suspending a job					
		execjob_preresume	Triggered just before resuming a job					
		"execjob_end"	Triggered after job finishes or is killed					
		"execjob_preterm"	Triggered just before job is killed					
		"execjob_epilogue"	Triggered just after job runs successfully					
		"exechost_periodic"	Triggered at periodic interval on execution hosts					
		"exechost_startup"	Triggered when MoM starts up or receives SIGHUP (Linux)					
		""	Hook is not triggered					

Hook Attributes								
Name Description	Format	Val / Opt	Value/Option Description	Default Value	Python Type	Usr	Opr	Mgr
fail_action Specifies the action to be taken when hook fails due to alarm call or unhandled exception, or to an internal error such as not enough disk space or memory. Can also specify a subsequent action to be taken when hook runs successfully. Value can be either "*none*" or one or more of "*offline_vnodes*", "*clear_vnodes_upon_recovery*", and "*scheduler_restart_cycle*". If this attribute is set to multiple values, scheduler restart happens last. See "Offlining and Clearing Vnodes Using the fail_action Hook Attribute" on page 66 in the PBS Professional Hooks Guide and "Restarting Scheduler Cycle After Hook Failure" on page 63 in the PBS Professional Hooks Guide.	*String_array*	*"none"*	No action is taken.	*"none"*				
		"offline_vnodes"	After unsuccessful hook execution, offlines the vnodes managed by the MoM executing the hook. Only available for execjob_prologue, exechost_startup and execjob_begin hooks.					
		"clear_vnodes_upon_recovery"	After successful hook execution, clears vnodes previously offlined via *"offline_vnodes"* fail action. Only available for exechost_startup hooks.					
		"scheduler_restart_cycle"	After unsuccessful hook execution, restarts scheduling cycle. Only available for execjob_begin and execjob_prologue hooks.					
freq Number of seconds between periodic or exechost_periodic triggers.	*Integer*		Number of seconds between triggers	*120*				
order Indicates relative order of hook execution, for hooks of the same type sharing a trigger. Hooks with lower order values execute before those with higher values. Does not apply to periodic or exechost_periodic hooks.	*Integer*	*Range:* *built-in hooks: [-1000, 2000]* *site hooks: [1,1000]*		*1*				
type The type of the hook. Cannot be set for a built-in hook.	*String*	*pbs*	Hook is built in	*site*				
		site	Hook is custom (site-defined)					
user Specifies who executes the hook.	*String*	*pbsadmin*	Hook runs as root	*pbsadmin*				
		pbsuser	Hook runs as owner of job					

7 Formats

This chapter describes the formats used in PBS Professional.

7.1 Non-resource Formats

Accounting Log Entry

logfile-date-time;record-type;id-string;message-text

where

logfile-date-time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss

record-type

A single character indicating the type of record

id-string

The job or reservation identifier

message-text

Format: blank-separated *keyword=value* fields.

Message text is ASCII text.

Content depends on the record type.

Attribute Name

PBS NAME. Cannot be used for a vnode name.

Date

<Day of week> <Name of month> <Day of month> <HH:MM:SS> <YYYY>

Datetime

A datetime is

[[[[CC]YY]MM]DD]hhmm[.SS]

where

Table 7-1: Datetime Symbols

Symbol	Meaning
CC	Century
YY	Year
MM	Month
DD	Day of month
hh	Hour
mm	Minute
SS	Second

When setting the value, each portion of the date defaults to the current date, as long as the next-smaller portion is in the future. For example, if today is the 3rd of the month and the specified day DD is the 5th, the month MM will be set to the current month.

If a specified portion has already passed, the next-larger portion will be set to one after the current date. For example, if the day DD is not specified, but the hour hh is specified to be 10:00 a.m. and the current time is 11:00 a.m., the day DD will be set to tomorrow.

Destination Identifier

String used to specify a particular destination. The identifier may be specified in one of three forms:

<queue name>@<server name>

<queue name>

@<server name>

where *<queue name>* is an ASCII character string of up to 15 characters.

Valid characters are alphanumerics, the hyphen and the underscore. The string must begin with a letter.

Hostname

String of the form

<machine name>.<domain name>

where *domain name* is a hierarchical, dot-separated list of subdomains.

A hostname cannot contain the following:

- A dot ("."), other than as a subdomain separator
- The commercial at sign, "@", as this is often used to separate a file from the host in a remote file name
- To prevent confusion with port numbers, a hostname cannot contain a colon (":")

The maximum length of a hostname supported by PBS is *255*.

Job Array ID, Job Array Identifier

The identifier returned upon success when submitting a job array.

Job array identifiers are a sequence number followed by square brackets:

<sequence number>[][.<server name>][@<server name>]

Example:

```
1234[]
```

Note that some shells require that you enclose a job array ID in double quotes.

The largest value that *sequence number* can be is set in the max_job_sequence_id server attribute. This attribute defaults to *9999999*. Minimum value for this attribute is *9999999*, and maximum is *999999999999*. After maximum for sequence number has been reached, job array IDs start again at *0*.

Job Array Range

<sequence number>[<first>-<last>][.<server name>][@<server name>]

first and *last* are the first and last indices of the subjobs.

Job ID, Job Identifier

When a job is successfully submitted to PBS, PBS returns a unique identifier for the job. Format:

<sequence number>[.<server>][@<new server>]

The <server> portion indicates the name of the original server where the job was submitted.

The @<new server> portion indicates the current location of the job if it is not at the original server.

The largest value that *sequence number* can be is set in the max_job_sequence_id server attribute. This attribute defaults to *9999999*. Minimum value for this attribute is *9999999*, and maximum is *999999999999*. After maximum for sequence number has been reached, job IDs start again at *0*.

Job Name, Job Array Name

A job name or job array name can be at most 230 characters. It must consist only of alphabetic, numeric, plus sign ("+"), dash or minus or hyphen ("-"), underscore ("_"), and dot or period (".") characters.

Default: if a script is used to submit the job, the job's name is the name of the script. If no script is used, the job's name is "*STDIN*".

Limit Specification

<limit specification>=<limit value>[, <limit specification>=<limit value>, ...]

where *limit specification* is:

Table 7-2: Limit Specification Syntax

Limit Specification	Limit
o:PBS_ALL	Overall limit
u:PBS_GENERIC	Generic users
u:<username>	An individual user
g:PBS_GENERIC	Generic groups
g:<group name>	An individual group
p:PBS_GENERIC	Generic projects
p:<project name>	An individual project

- The *limit specification* can contain spaces anywhere except after the colon (":").
- If there are comma-separated *limit specifications*, the entire string must be enclosed in double quotes.
- A username, group name, or project name containing spaces must be enclosed in quotes.
- If a username, group name, or project name is quoted using double quotes, and the entire string requires quotes, the outer enclosing quotes must be single quotes. Similarly, if the inner quotes are single quotes, the outer quotes must be double quotes.
- *PBS_ALL* is a keyword which indicates that this limit applies to the usage total.
- *PBS_GENERIC* is a keyword which indicates that this limit applies to generic users, groups, or projects.
- When removing a limit, the *limit value* does not need to be specified.
- *PBS_ALL* and *PBS_GENERIC* are case-sensitive.

Format for setting a limit attribute:

set server <limit attribute> = "<limit specification>=<limit value>[, <limit specification>=<limit value>], ..."

set queue <queue name> <limit attribute> = "<limit specification>=<limit value>[, <limit specification>=<limit value>], ..."

For example, to set the max_queued limit on QueueA to *5* for total usage, and to limit user bill to *3*:

```
Qmgr: s q QueueA max_queued = "[o:PBS_ALL=5], [u:bill =3]"
```

Examples of setting, adding, and removing:

```
Qmgr: set server max_run="[u:PBS_GENERIC=2], [g:group1=10], [o:PBS_ALL = 100]"
Qmgr: set server max_run+="[u:user1=3], [g:PBS_GENERIC=8]"
Qmgr: set server max_run-="[u:user2], [g:group3]"
Qmgr: set server max_run_res.ncpus="[u:PBS_GENERIC=2], [g:group1=8], [o:PBS_ALL = 64]"
```

See "How to Set Limits at Server and Queues" on page 296 in the PBS Professional Administrator's Guide.

Event logfile-date-time

Date and time stamp in the format:

mm/dd/yyyy hh:mm:ss[.xxxxxx]

If microsecond logging is enabled, microseconds are logged using the *.xxxxxx* portion. Microseconds may be preceded by zeroes. Microsecond logging is controlled per host via the PBS_LOG_HIGHRES_TIMESTAMP configuration parameter or environment variable.

pathname

All printable characters except for ampersand ("&")

PBS NAME

This is a generic term, used to describe various PBS entities. For example, attribute names are PBS NAMEs.

Must start with an alphabetic character, and may contain only the following: alpha-numeric, underscore ("_"), or dash ("-").

Do not use PBS keywords as PBS NAMEs.

PBS Password

The `pbs_ds_password` command generates passwords containing the following characters:

0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&()_+*

When creating a password manually, do not use \ (backslash) or ‘ (backquote). This can prevent certain commands such as `pbs_server`, `pbs_ds_password`, and `printjob` from functioning properly, as they rely on connecting to the database.

Project Name

A project name can contain any characters except for the following: slash ("/"), left bracket ("["), right bracket ("]"), double quote (""""), semicolon (";"), colon (":"), vertical bar ("|"), left angle bracket ("<"), right angle bracket (">"), plus ("+"), comma (","), question mark ("?"), and asterisk ("*").

Default value: "*_pbs_project_default*".

Queue ID, Queue Identifier

To specify a queue at the default server:

<queue name>

To specify all queues at a server:

@<server name>

To specify a queue at a specific server:

<queue name>@<server name>

Queue Name

PBS NAME up to 15 characters in length

Reservation ID, Reservation Identifier

Format for an advance reservation:

R<sequence number>[.<server name>][@<server name>]

Format for a standing reservation:

S<sequence number>[.<server name>][@<server name>]

Format for a maintenance reservation:

M<sequence number>[.<server name>][@<server name>]

The largest value that *sequence number* can be is set in the max_job_sequence_id server attribute. This attribute defaults to *9999999*. Minimum value for this attribute is *9999999*, and maximum is *999999999999*. After maximum for sequence number has been reached, reservation IDs start again at *0*.

Reservation Name

Same as Job Name. See "Job Name, Job Array Name" on page 355.

Resource Name

PBS NAME up to 64 characters in length

Resource names are case-insensitive.

Subjob Identifier

Subjob identifiers are a sequence number followed by square brackets enclosing the subjob's index:

<sequence number>[<index>][.<server name>][@<server name>]

Example:

```
1234[99]
```

Timestamp

Output format varies depending on context:

- Printed by `qstat` in human-readable *Date* format
- Output in hooks as seconds since epoch

Username

Linux username:

String up to 16 characters in length. PBS supports usernames containing any printable, non-whitespace character except the at sign ("@"). Your platform may place additional limitations on usernames.

Windows username:

Must conform to the POSIX-1 standard for portability:

- The username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-".
- The hyphen must not be the first letter of the username.
- If "@" appears in the username, it will assumed to be in the context of a Windows domain account: username@domainname.

An exception to the above rule is the space character, which is allowed. If a space character appears in a username string, it will be displayed quoted and must be specified in a quoted manner.

Vnode Name

Hostname, IP address, or other legal string, according to the following:

- For the parent vnode, the vnode name must conform to legal name for a host; see Hostname
- For other vnodes, the vnode name can be alphanumeric and any of these:

  - (dash)

  _ (underscore)

  @ (at sign)

  [(left bracket)

  ] (right bracket)

  # (hash)

  ^ (caret)

  / (slash)

  \ (backslash)

. (period)

- Cannot be the same as an attribute name
- Vnode names are case-insensitive

7.2 Resource Formats

Boolean

Name of Boolean resource is a string.

Values:

TRUE, True, true, T, t, Y, y, 1

FALSE, False, false, F, f, N, n, 0

Duration

A period of time, expressed either as

An integer whose units are seconds

or

[[hours:]minutes:]seconds[.milliseconds]

in the form:

[[HH:]MM:]SS[.milliseconds]

Milliseconds are rounded to the nearest second.

Float

Floating point. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

Long

Long integer. Allowable values: 0-9 [[0-9] ...], and + and -

<queue name>@<server name>

Size

Number of bytes or words. The size of a word is 64 bits.

Format: *<integer>[<suffix>]*

where *suffix* can be one of the following:

Table 7-3: Size in Bytes

Suffix	Meaning	Size
b or w	Bytes or words	1
kb or kw	Kilobytes or kilowords	2 to the 10th, or 1024
mb or mw	Megabytes or megawords	2 to the 20th, or 1,048,576
gb or gw	Gigabytes or gigawords	2 to the 30th, or 1,073,741,824
tb or tw	Terabytes or terawords	2 to the 40th, or 1024 gigabytes
pb or pw	Petabytes or petawords	2 to the 50th, or 1,048,576 gigabytes

Default: *bytes*

Note that a scheduler rounds all resources of type size up to the nearest kb.

String

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~] ...]

String resource values are case-sensitive. No limit on length.

String Array

Comma-separated list of strings.

Strings in string_array may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

8 States

This chapter lists and describes the states in PBS Professional.

8.1 Job States

Job states are abbreviated to one character.

Table 8-1: Job States

State	Numeric	Description
B	*7*	Job arrays only: job array is begun, meaning that at least one subjob has started
E	*5*	Job is exiting after having run
F	*9*	Job is finished. Job has completed execution, job failed during execution, or job was deleted.
H	*2*	Job is held. A job is put into a held state by the server or by a user or administrator. A job stays in a held state until it is released by a user or administrator.
M	*8*	Job was moved to another server
Q	*1*	Job is queued, eligible to run or be routed
R	*4*	Job is running
S	*None; sub-state of Running*	Job is suspended by scheduler. A job is put into the suspended state when a higher priority job needs the resources.
T	*0*	Job is in transition to or from a server
U	*None; sub-state of Running*	Job is suspended due to workstation becoming busy
W	*3*	Job is waiting for its requested execution time to be reached, or job is delayed due to stagein failure.
X	*6*	Subjobs only; subjob is finished (expired.)

8.1.1 Job Substates

Job substates are numeric:

Table 8-2: Job Substates

Substate Number	Substate Description
00	Transit in, prior to waiting for commit
01	Transit in, waiting for commit
02	transiting job outbound, not ready to commit
03	transiting outbound, ready to commit
10	job queued and ready for selection
11	job queued, has files to stage in
13	Job waiting on sync start ready
14	job staging in files before waiting
15	job staging in files before running
16	job stage in complete
20	job held - user or operator
21	job held waiting on sync regist
22	job held - waiting on dependency
30	job waiting until user-specified execution time
37	job held - file stage in failed
41	job sent to MoM to run
42	Running
43	Suspended by Operator or Manager
45	Suspended by scheduler
50	Server received job obit
51	Staging out stdout/err and other files
52	Deleting stdout/err files and staged-in files
53	Mom releasing resources
54	job is being aborted by server
56	(Set by MoM) Mother Superior telling sisters to kill everything
57	(Set by MoM) job epilogue running
58	(Set by MoM) job obit notice sent
59	Waiting for site "job termination" action script
60	Job to be rerun, MoM sending stdout/stderr back to server
61	Job to be rerun, staging out files

Table 8-2: Job Substates

Substate Number	Substate Description
62	Job to be rerun, deleting files
63	Job to be rerun, freeing resources
69	subjob is gone
70	Array job has begun
71	Job is waiting for vnode(s) to be provisioned with its requested AOE.
91	Job is terminated
92	Job is finished
93	Job failed
94	Job was moved
153	(Set by MoM) Mother Superior waiting for delete ACK from sisters

8.2 Job Array States

Job array states map closely to job states except for the '*B*' state. The '*B*' state applies to job arrays and indicates that at least one subjob has left the queued state and is running or has run, but not all subjobs have run. Job arrays will never be in the '*R*', '*S*' or '*U*' states.

Table 8-3: Job Array States

State	Numeric	Indication
B	*7*	The job array has started
E	*5*	All subjobs are finished and the server is cleaning up the job array
F	*9*	The job array is finished
H	*2*	The job array is held
Q	*1*	The job array is queued, or has been qrerun
T	*0*	The job array is in transit between servers
W	*3*	The job array is waiting for its execution time to be reached, or job array was delayed due to stagein failure

8.3 Subjob States

Subjobs can be in one of six states, listed here.

Table 8-4: Subjob States

State	Numeric	Indication
E	*5*	Ending
F	*9*	Finished
Q	*1*	Queued
R	*4*	Running
S	*None; sub-state of Running*	Suspended
U	*None; sub-state of Running*	Suspended by keyboard activity
X	*6*	Expired or deleted; subjob has completed execution or been deleted

8.4 Server States

The state of the server is shown in the server_state server attribute. Possible values are shown in the following table:

Table 8-5: Server States

State	Description
Hot_Start	The server has been started so that it will run first any jobs that were running when the server was shut down. Python type: pbs.SV_STATE_HOT
Idle	The server is running. The scheduler is between scheduling cycles. Python type: pbs.SV_STATE_IDLE
Scheduling	The server is running. The scheduler is in a scheduling cycle. Python type: pbs.SV_STATE_ACTIVE
Terminating	The server is terminating. Python type: pbs.SV_STATE_SHUTIMM or pbs.SV_STATE_SHUTSIG
Terminating_Delayed	The server is terminating in delayed mode. No new jobs will be run, and the server will shut down when the last running job finishes. Python type: pbs.SV_STATE_SHUTDEL

8.5 Vnode States

If a vnode's state attribute is unset, that is equivalent to the state being *free*. A vnode's state is shown in its state attribute, which can take on zero or more of the values listed here. Some vnode state values can be set simultaneously. Values are:

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
busy	Server	Node is up and has load average greater than max_load, or is showing keyboard or mouse activity. When the loadave is above max_load, that node is marked *busy*. A scheduler won't place jobs on a node marked *busy*. When the loadave drops below ideal_load, or when the mouse and keyboard have not shown any activity for a specified amount of time, the *busy* mark is removed. Consult your OS documentation to determine values that make sense.	*offline* *maintenance*
down	Server	Node is not usable. Existing communication lost between server and MoM.	*maintenance* Cannot be set with *free*
free	Server	Node is up and has available CPU(s). Server will mark a vnode "*free*" on first successful ping after vnode was "*down*".	None
job-busy	Server	Node is up and all CPUs are allocated to jobs.	*offline* *resv-exclusive*
job-exclusive	Server	Node is up and has been allocated exclusively to a single job.	*offline* *resv-exclusive*
maintenance	Server	A vnode enters the *maintenance* state when any of its jobs is suspended with the *admin-suspend* signal. Other jobs running on this vnode continue to run; each job must be *admin-suspend*ed. The vnode leaves the *maintenance* state when the last job is resumed with the *admin-resume* signal. A scheduler does not start or resume jobs on a node in the *maintenance* state. Any reservations on vnodes in the *maintenance* state are marked *degraded*. PBS searches for alternate vnodes for those reservations.	*down* *offline*
offline	Manager Operator	Node is not usable. Jobs running on this vnode will continue to run. Used by Manager/Operator to mark a vnode not to be used for jobs.	*busy* *job-busy* *job-exclusive* *resv-exclusive*
powered-off		Indicates that this vnode was powered off by PBS via power provisioning. This tells the scheduler that it can schedule jobs on this vnode; in that case PBS powers the vnode back up.	

Table 8-6: Vnode States

State	Set By	Description	Can Combine With these States
powering-down		Indicates that this vnode is in the process of being powered down by PBS via power provisioning.	
powering-on		Indicates that this vnode is in the process of being powered up by PBS via power provisioning.	
provisioning	Server	A vnode is in the provisioning state while it is in the process of being provisioned. No jobs are run on vnodes in the provisioning state.	Cannot be set with any other states
resv-exclusive	Server	Reservation has requested exclusive use of vnode, and reservation is running.	*job-exclusive, offline*
sleep	Server	Indicates that this vnode was ramped down or powered off via PBS power management. This tells the scheduler that it can schedule jobs on this vnode; in that case PBS powers the vnode back up.	
stale	Server	MoM managing vnode is not reporting any information about this vnode, but was reporting it previously. Server can still communicate with MoM. A vnode becomes *stale* when: 1. A vnode is defined in the server 2. MoM starts or restarts and reports a set of vnodes according to her configuration 3. A vnode which existed in the server earlier is not in the set being reported now by MoM 4. That vnode is marked "*stale*"	Cannot be set with *free*
state-unknown, down	Server	Node is not usable. Since server's latest start, no communication with this vnode. May be network or hardware problem, or no MoM on vnode.	
unresolvable	Server	Server cannot resolve name of vnode	
wait-provisioning	Server	There is a limit on the maximum number of vnodes that can be in the provisioning state. This limit is specified in the server's max_concurrent_provision attribute. If a vnode is to be provisioned, but cannot because the number of concurrently provisioning vnodes has reached the specified maximum, the vnode goes into the *wait-provisioning state*. No jobs are run on vnodes in the *wait-provisioning* state.	Cannot be set with any other states

8.6 Reservation States

The following table shows the list of possible states for a reservation. The states that you will usually see are *CO*, *UN*, *BD*, and *RN*, although a reservation usually remains unconfirmed for too short a time to see that state.

Table 8-7: Reservation States

Code	Numeric	State	Description
AL	11	*RESV_BEING_ALTERED*	Transitory state; reservation is being altered
BD	7	*RESV_BEING_DELETED*	Transitory state; reservation is being deleted
CO	2	*RESV_CONFIRMED*	Reservation confirmed
DG	10	*RESV_DEGRADED*	Vnode(s) allocated to reservation unavailable
DE	8	*RESV_DELETED*	Transitory state; reservation has been deleted
DJ	9	*RESV_DELETING_JOBS*	Jobs remaining after reservation's end time being deleted
FN	6	*RESV_FINISHED*	Transitory state; reservation's end time has arrived and reservation will be deleted
IC	12	*RESV_IN_CONFLICT*	This reservation conflicts with a maintenance reservation
NO	0	*RESV_NONE*	No reservation yet
RN	5	*RESV_RUNNING*	Time period from reservation's start time to end time is being traversed
TR	4	*RESV_TIME_TO_RUN*	Transitory state; reservation's start time has arrived
UN	1	*RESV_UNCONFIRMED*	Reservation not confirmed
WT	3	*RESV_WAIT*	Unused

8.6.1 Degraded Reservation Substates

The following table shows states and substates for degraded reservations:

Table 8-8: Degraded Reservation States and Substates

Occurrence Type	Reservation Time Is Now		Reservation Time in Future	
	State	Substate	State	Substate
Advance and job-specific reservation: running	*RESV_RUNNING*	*RESV_DEGRADED*	*RESV_DEGRADED*	*RESV_DEGRADED*
Advance and job-specific reservation: conflicts with maintenance reservation	*RESV_DEGRADED*	*RESV_IN_CONFLICT*	*RESV_DEGRADED*	*RESV_IN_CONFLICT*
Standing reservation soonest occurrence: running	*RESV_RUNNING*	*RESV_DEGRADED*	*RESV_DEGRADED*	*RESV_DEGRADED*
Standing reservation soonest occurrence: conflicts with maintenance reservation	*RESV_DEGRADED*	*RESV_IN_CONFLICT*	*RESV_DEGRADED*	*RESV_IN_CONFLICT*
Standing reservation non-soonest occurrence only: conflicts with maintenance reservation	*N/A*	*N/A*	*RESV_CONFIRMED*	*RESV_DEGRADED*
	N/A	*N/A*	*RESV_RUNNING*	*RESV_RUNNING*

9 The PBS Configuration File

9.1 Format of Configuration File

Each line in the /etc/pbs.conf file gives a value for one parameter, or is a comment, or is blank. The order of the elements is not important.

9.1.1 Specifying Parameters

When you specify a parameter value, do not include any spaces in the line. Format for specifying a parameter value:

<parameter>=<value>

For example, to specify a value for PBS_START_MOM on the local host:

```
PBS_START_MOM=1
```

9.1.2 Comment Lines in Configuration File

You can comment out lines you are not using. Precede a comment with the hashmark ("#"). For example:

```
#This is a comment line
```

9.2 Contents of Configuration File

The /etc/pbs.conf file contains configuration parameters for PBS. The following table describes the parameters you can use in the pbs.conf configuration file:

Table 9-1: Parameters in pbs.conf

Parameter	Description
PBS_AUTH_METHOD	Specifies default authentication method and library to be used by PBS. Used only at authenticating client. Case-insensitive. Default value: *resvport* To use MUNGE, set to *munge*
PBS_BATCH_SERVICE_PORT	Port on which server listens. Default: 15001
PBS_BATCH_SERVICE_PORT_DIS	DIS port on which server listens.
PBS_COMM_LOG_EVENTS	Communication daemon log mask. Default: *511*
PBS_COMM_ROUTERS	Tells a pbs_comm the location of the other pbs_comms.
PBS_COMM_THREADS	Number of threads for communication daemon.

Table 9-1: Parameters in `pbs.conf`

Parameter	Description
PBS_REMOTE_VIEWER	Specifies remote viewer client. If not specified, PBS uses native Remote Desktop client for remote viewer. Set on submission host(s). Supported on Windows only.
PBS_CORE_LIMIT	Limit on corefile size for PBS daemons. Can be set to an integer number of bytes or to the string "unlimited". If unset, core file size limit is inherited from the shell environment.
PBS_DATA_SERVICE_PORT	Used to specify non-default port for connecting to data service. Default: 15007
PBS_ENCRYPT_METHOD	Specifies method and library for encrypting and decrypting data in client-server communication. Used only at authentication client side. Case-insensitive. To use TLS encryption in client-server communication, set this parameter to *tls*. No default; if this is not set, PBS does not encrypt or decrypt data.
PBS_ENVIRONMENT	Location of `pbs_environment` file.
PBS_EXEC	Location of PBS `bin` and `sbin` directories.
PBS_HOME	Location of PBS working directories.
PBS_LEAF_NAME	Tells endpoint what hostname to use for network. The value does not include a port, since that is usually set by the daemon. By default, the name of the endpoint's host is the hostname of the machine. You can set the name where an endpoint runs. This is useful when you have multiple networks configured, and you want PBS to use a particular network. The server only queries for the canonicalized address of the MoM host, unless you let it know via the Mom attribute; if you have set PBS_LEAF_NAME in `/etc/pbs.conf` to something else, make sure you set the Mom attribute at vnode creation. TPP internally resolves the name to a set of IP addresses, so you do not affect how `pbs_comm` works.
PBS_LEAF_ROUTERS	Location of endpoint's `pbs_comm` daemon(s).
PBS_LOCALLOG=<value>	Enables logging to local PBS log files. Valid values: 0: no local logging 1: local logging enabled Only available when using `syslog`.

Table 9-1: Parameters in `pbs.conf`

Parameter	Description
PBS_LOG_HIGHRES_TIMESTAMP	Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is *HH:MM:SS*. With microsecond logging, format is *HH:MM:SS:XXXXXX*. Does not affect accounting log. Not applicable when using `syslog`. Overridden by environment variable of the same name. Valid values: *0*, *1*. Default: *0* (no microsecond logging)
PBS_LR_SAVE_PATH	Path where Undo Live Recorder stores daemon execution recordings. Default: `$PBS_HOME/spool`. See "Sending Daemon Execution Recordings to Altair" on page 663 in the PBS Professional Administrator's Guide
PBS_MAIL_HOST_NAME	Used in addressing mail regarding jobs and reservations that is sent to users specified in a job or reservation's Mail_Users attribute. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). For how this is used in email address, see section 2.2.2, "Specifying Mail Delivery Domain", on page 20.
PBS_MANAGER_SERVICE_PORT	Port on which MoM listens. Default: 15003
PBS_MOM_HOME	Location of MoM working directories.
PBS_MOM_NODE_NAME	Name that MoM should use for parent vnode, and if they exist, child vnodes. If this is not set, MoM defaults to using the non-canonicalized hostname returned by `gethostname()`. If you use the IP address for a vnode name, set `PBS_MOM_NODE_NAME=<IP address>` in `pbs.conf` on the execution host. Dots are not allowed in this parameter unless they are part of an IP address.
PBS_MOM_SERVICE_PORT	Port on which MoM listens. Default: 15002
PBS_OUTPUT_HOST_NAME	Host to which all job standard output and standard error are delivered. If specified in `pbs.conf` on a job submission host, the value of PBS_OUTPUT_HOST_NAME is used in the host portion of the job's Output_Path and Error_Path attributes. If the job submitter does not specify paths for standard output and standard error, the current working directory for the `qsub` command is used, and the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies only a file path for standard output and standard error, the value of PBS_OUTPUT_HOST_NAME is appended after an at sign ("@"). If the job submitter specifies paths for standard output and standard error that include host names, the specified paths are used. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Delivering Output and Error Files" on page 60 in the PBS Professional Administrator's Guide.

Table 9-1: Parameters in `pbs.conf`

Parameter	Description
PBS_PRIMARY	Hostname of primary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the primary server host, make sure that PBS_PRIMARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_PRIMARY matches the hostname of the server host.
PBS_RCP	Location of `rcp` command if `rcp` is used.
PBS_SCHEDULER_SERVICE_PORT	Port on which default scheduler listens. Default value: 15004
PBS_SCHED_THREADS	Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable. Overridden by `pbs_sched -t` option and PBS_SCHED_THREADS environment variable. Default: 1
PBS_SCP	Location of `scp` command if `scp` is used; setting this parameter causes PBS to first try `scp` rather than `rcp` for file transport.
PBS_SECONDARY	Hostname of secondary server. Used only for failover configuration. Overrides PBS_SERVER_HOST_NAME. If you set PBS_LEAF_NAME on the secondary server host, make sure that PBS_SECONDARY matches PBS_LEAF_NAME on the corresponding host. If you do not set PBS_LEAF_NAME on the server host, make sure that PBS_SECONDARY matches the hostname of the server host.
PBS_SERVER	Hostname of host running the server. Cannot be longer than 255 characters. If the short name of the server host resolves to the correct IP address, you can use the short name for the value of the PBS_SERVER entry in `pbs.conf`. If only the FQDN of the server host resolves to the correct IP address, you must use the FQDN for the value of PBS_SERVER. Overridden by PBS_SERVER_HOST_NAME and PBS_PRIMARY.
PBS_SERVER_HOST_NAME	The FQDN of the server host. Used by clients to contact server. Overridden by PBS_PRIMARY and PBS_SECONDARY failover parameters. Overrides PBS_SERVER parameter. Optional. If specified, must be a fully qualified domain name. Cannot contain a colon (":"). See "Contacting the Server" on page 60 in the PBS Professional Administrator's Guide.
PBS_START_COMM	Set this to *1* if a communication daemon is to run on this host.
PBS_START_MOM	Default is *0*. Set this to *1* if a MoM is to run on this host.
PBS_START_SCHED	**Deprecated**. Set this to *1* if default scheduler is to run on this host. Overridden by scheduler's scheduling attribute.
PBS_START_SERVER	Set this to *1* if server is to run on this host.

Table 9-1: Parameters in `pbs.conf`

Parameter	Description
PBS_SUPPORTED_AUTH_METHODS	Specifies supported authentication methods for client-server communication. Used by authenticating server (PBS server, scheduler, MoM, or comm); ignored at client. Case-insensitive. If this parameter is set, PBS accepts only the methods listed. Format: comma-separated list of authentication methods. Default value: *resvport* Example: *munge,GSS*
PBS_SYSLOG=<value>	Controls use of `syslog` facility under which the entries are logged. Valid values: 0: no syslogging 1: logged via LOG_DAEMON facility 2: logged via LOG_LOCAL0 facility 3: logged via LOG_LOCAL1 facility ... 9: logged via LOG_LOCAL7 facility
PBS_SYSLOGSEVR=<value>	Filters `syslog` messages by severity. Valid values: 0: only LOG_EMERG messages are logged 1: messages up to LOG_ALERT are logged ... 7: messages up to LOG_DEBUG are logged
PBS_TMPDIR	Location of temporary files/directories used by PBS components.

10 Log Levels

10.1 Log Levels

PBS allows specification of the types of events that are logged for each daemon. Each type of log event has a different log level. All daemons use the same log level for the same type of event.

The following table lists the log level for each type of event.

Table 10-1: PBS Events and Log Levels

Name	Decimal	Hex	Event Description
PBSEVENT_ERROR	1	0x0001	Internal PBS errors
PBSEVENT_SYSTEM	2	0x0002	System (OS) errors, such as malloc failure
PBSEVENT_ADMIN	4	0x0004	Administrator-controlled events, such as changing queue attributes
PBSEVENT_JOB	8	0x0008	Job related events, e.g. submitted, ran, deleted
PBSEVENT_JOB_USAGE	16	0x0010	Job resource usage
PBSEVENT_SECURITY	32	0x0020	Security related events
PBSEVENT_SCHED	64	0x0040	When the scheduler was called and why
PBSEVENT_DEBUG	128	0x0080	Common debug messages
PBSEVENT_DEBUG2	256	0x0100	Debug event class 2
PBSEVENT_RESV	512	0x0200	Reservation-related messages
PBSEVENT_DEBUG3	1024	0x0400	Debug event class 3. Debug messages rarer than event class 2.
PBSEVENT_DEBUG4	2048	0x0800	Debug event class 4. Limit-related messages.

11
Job Exit Status

11.1 Job Exit Status

The exit status of a job may fall in one of three ranges, listed in the following table:

Table 11-1: Job Exit Status Ranges

Exit Status Range	Reason	Description
X < 0	The job could not be executed	See Table 11-2, "Job Exit Codes," on page 377
0 <=X < 128	Exit value of shell or top process	This is the exit value of the top process in the job, typically the shell. This may be the exit value of the last command executed in the shell or the `.logout` script if the user has such a script (`csh`). The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.
X >=128	Job was killed with a signal	This means the job was killed with a signal. The signal is given by X modulo 128 (or 256). For example an exit value of 137 means the job's top process was killed with signal 9 (137 % 128 = 9). The exit status values greater than 128 (or 256) indicate which signal killed the job. Depending on the system, values greater than 128 (or on some systems 256; see `wait(2)` or `waitpid(2)` for more information), are the value of the signal that killed the job. To interpret (or "decode") the signal contained in the exit status value, subtract the base value from the exit status. For example, if a job had an exit status of 143, that indicates the job was killed via a SIGTERM (e.g. 143 - 128 = 15, signal 15 is SIGTERM). See the `kill(1)` manual page for a mapping of signal numbers to signal name on your operating system.

The exit status of jobs is recorded in the PBS server logs and the accounting logs.

Negative exit status indicates that the job could not be executed. Negative exit values are listed in the table below:

Table 11-2: Job Exit Codes

Exit Code	Name	Description
0	*JOB_EXEC_OK*	Job execution was successful
-1	*JOB_EXEC_FAIL1*	Job execution failed, before files, no retry
-2	*JOB_EXEC_FAIL2*	Job execution failed, after files, no retry

Table 11-2: Job Exit Codes

Exit Code	Name	Description
-3	*JOB_EXEC_RETRY*	Job execution failed, do retry
-4	*JOB_EXEC_INITABT*	Job aborted on MoM initialization
-5	*JOB_EXEC_INITRST*	Job aborted on MoM initialization, checkpoint, no migrate
-6	*JOB_EXEC_INITRMG*	Job aborted on MoM initialization, checkpoint, ok migrate
-7	*JOB_EXEC_BADRESRT*	Job restart failed
-10	*JOB_EXEC_FAILUID*	Invalid UID/GID for job
-11	*JOB_EXEC_RERUN*	Job was rerun
-12	*JOB_EXEC_CHKP*	Job was checkpointed and killed
-13	*JOB_EXEC_FAIL_PASSWORD*	Job failed due to a bad password
-14	*JOB_EXEC_RERUN_ ON_SIS_FAIL*	Job was requeued (if rerunnable) or deleted (if not) due to a communication failure between the primary execution host MoM and a Sister
-15	JOB_EXEC_QUERST	Requeue job for restart from checkpoint
-16	JOB_EXEC_FAILHOOK_RERUN	Job execution failed due to hook rejection; requeue for later retry
-17	JOB_EXEC_FAILHOOK_DELETE	Job execution failed due to hook rejection; delete the job at end
-18	JOB_EXEC_HOOK_RERUN	A hook requested for job to be requeued
-19	JOB_EXEC_HOOK_DELETE	A hook requested for job to be deleted
-20	JOB_EXEC_RERUN_MS_FAIL	Job requeued because server couldn't contact the primary execution host MoM

12 Example Configurations

This chapter shows some configuration-specific scenarios which will hopefully clarify any configuration questions. Several configuration models are discussed, followed by several complex examples of specific features.

- Single Vnode System
- Single Vnode System with Separate PBS server
- Multi-vnode complex
- Complex Multi-level Route Queues (including group ACLs)
- Multiple User ACLs

For each of these possible configuration models, the following information is provided:

- General description for the configuration model
- Type of system for which the model is well suited
- Contents of server nodes file
- Any required server configuration
- Any required MoM configuration
- Any required scheduler configuration

12.1 Single Vnode System

Running PBS on a single vnode/host as a standalone system is the least complex configuration. This model is most applicable to sites who have a single large server system. In this model, all PBS components run on the same host, which is the same host on which jobs will be executed. The following illustration shows how communication works when PBS is on a single host in TPP mode. For more on TPP mode, see Chapter 4, "Communication", on page 45.

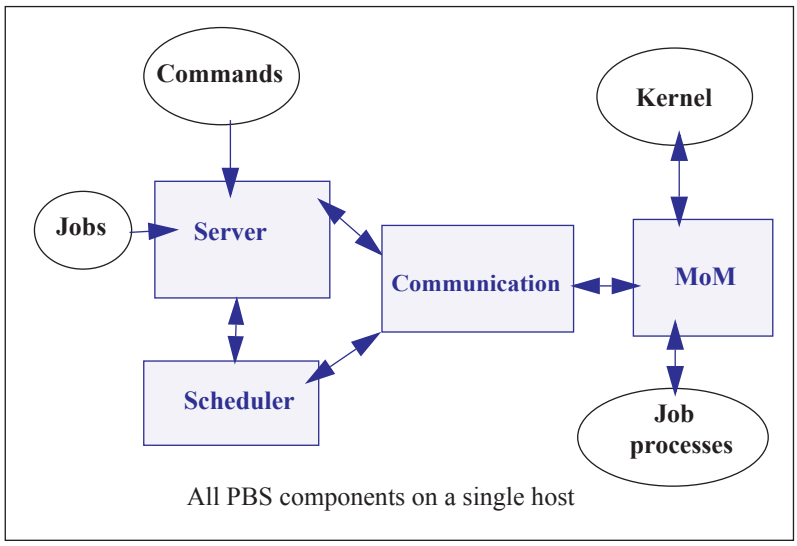


Figure 12-1:PBS daemons on a single execution host

For this example, let's assume we have a 32-CPU server machine named "mars". We want users to log into mars and jobs will be run via PBS on mars.

In this configuration, the server's default `nodes` file (which should contain the name of the host on which the server was installed) is sufficient. Our example `nodes` file would contain only one entry: `mars`

The default MoM and scheduler `config` files, as well as the default queue/Server limits are also sufficient in order to run jobs. No changes are required from the default configuration, however, you may wish to customize PBS to your site.

12.2 Separate Server and Execution Host

A variation on the model presented above would be to provide a "front-end" system that ran the PBS server, scheduler, and communication daemons, and from which users submitted their jobs. Only the MoM would run on our execution server, mars. This model is recommended when the user load would otherwise interfere with the computational load on the server. The following illustration shows how communication works when the PBS server and scheduler are on a front-end system and MoM is on a separate host, in TPP mode. For more on TPP mode, see Chapter 4, "Communication", on page 45.

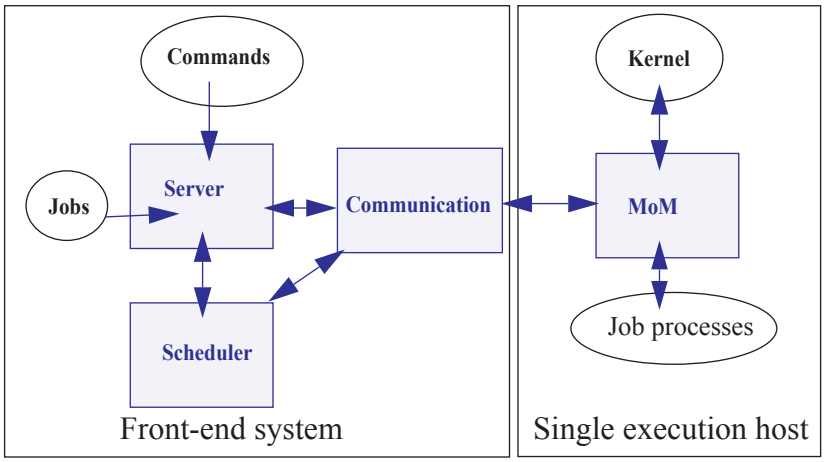


Figure 12-2:PBS daemons on single execution system with front end

In this case, the PBS `server_priv/nodes` file would contain the name of our execution server mars, but this may not be what was written to the file during installation, depending on which options were selected. It is possible the hostname of the machine on which the server was installed was added to the file, in which case you would need to use `qmgr(1B)` to manipulate the contents to contain one vnode: `mars`. If the default scheduling policy, based on available CPUs and memory, meets your requirements, no changes are required in either the MoM or scheduler configuration files.

However, if you wish the execution host (mars) to be scheduled based on load average, the following changes are needed. Edit MoM's `mom_priv/config` file so that it contains the target and maximum load averages:

```
$ideal_load 30
$max_load 32
```

12.3 Multiple Execution Hosts

The multi-vnode complex model is a very common configuration for PBS. In this model, there is typically a front-end system as we saw in the previous example, with a number of back-end execution hosts. The PBS server, scheduler, and communication daemons are typically run on the front-end system, and a MoM is run on each of the execution hosts, as shown in the diagram to the right.

In this model, the server's `nodes` file will need to contain the list of all the vnodes in the complex.

The MoM `config` file on each vnode will need two static resources added, to specify the target load for each vnode. If we assume each of the vnodes in our "planets" cluster is a 32-processor system, the following example shows what might be desirable ideal and maximum load values to add to the MoM `config` files:

```
$ideal_load 30
$max_load 32
```

Furthermore, suppose we want the partition scheduler to load balance the workload across the available vnodes, making sure not to run two jobs in a row on the same vnode . We accomplish this by editing the scheduler configuration file and enabling load balancing:

```
load_balancing: True all
smp_cluster_dist: round_robin
```

The following diagram illustrates this for an eight-host complex in TPP mode.

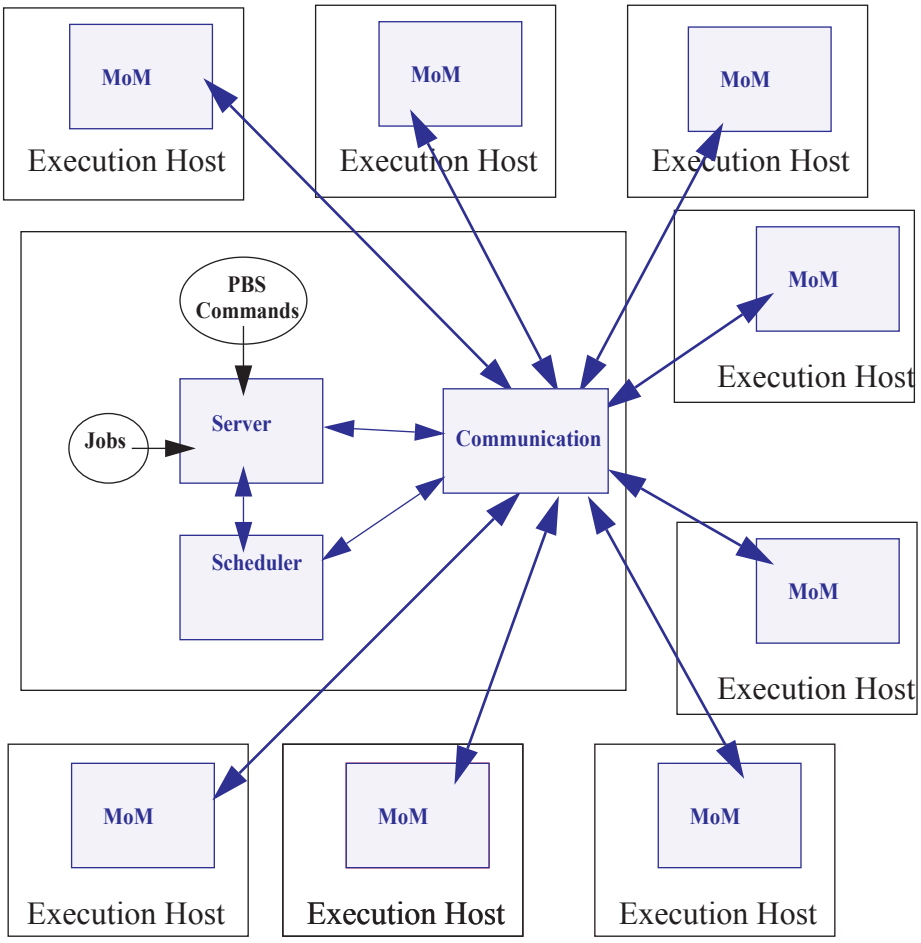


Figure 12-3:Typical PBS daemon locations for multiple execution hosts

This diagram illustrates a multi-vnode complex TPP configuration wherein the server and scheduler daemons communicate with the MoMs on the execution hosts via the communication daemon. Jobs are submitted to the server, scheduled for execution by the partition scheduler, and then transferred to a MoM when it's time to be run. MoM periodically sends status information back to the server, and answers resource requests from the scheduler.

12.4 Complex Multi-level Route Queues

There are times when a site may wish to create a series of route queues in order to filter jobs, based on specific resources, or possibly to different destinations. For this example, consider a site that has two large server systems, and a Linux cluster. The Administrator wants to configure route queues such that everyone submits jobs to a single queue, but the jobs get routed based on (1) requested architecture and (2) individual group IDs. In other words, users request the architecture they want, and PBS finds the right queue for them. Only groups "math", "chemistry", and "physics" are permitted to use either server systems; while anyone can use the cluster. Lastly, the jobs coming into the cluster should be divided into three separate queues for long, short, and normal jobs. But the "long" queue was created for the astronomy department, so only members of that group should be permitted into that queue. Given these requirements, let's look at how we would set up such a collection of route queues. (Note that this is only one way to accomplish this task. There are various other ways too.)

First we create a queue to which everyone will submit their jobs. Let's call it "submit". It will need to be a route queue with three destinations, as shown:

```
Qmgr: create queue submit
Qmgr: set queue submit queue_type = Route
Qmgr: set queue submit route_destinations = server_1
Qmgr: set queue submit route_destinations += server_2
Qmgr: set queue submit route_destinations += cluster
Qmgr: set queue submit enabled = True
Qmgr: set queue submit started = True
```

Now we need to create the destination queues. (Notice in the above example, we have already decided what to call the three destinations: `server_1`, `server_2`, `cluster`.) First we create the `server_1` queue, complete with a group ACL, and a specific architecture limit.

```
Qmgr: create queue server_1
Qmgr: set queue server_1 queue_type = Execution
Qmgr: set queue server_1 from_route_only = True
Qmgr: set queue server_1 resources_max.arch = linux
Qmgr: set queue server_1 resources_min.arch = linux
Qmgr: set queue server_1 acl_group_enable = True
Qmgr: set queue server_1 acl_groups = math
Qmgr: set queue server_1 acl_groups += chemistry
Qmgr: set queue server_1 acl_groups += physics
Qmgr: set queue server_1 enabled = True
Qmgr: set queue server_1 started = True
```

Next we create the queues for server_2 and cluster. Note that the server_2 queue is very similar to the server_1 queue, only the architecture differs. Also notice that the cluster queue is another route queue, with multiple destinations.

```
Qmgr: create queue server_2
Qmgr: set queue server_2 queue_type = Execution
Qmgr: set queue server_2 from_route_only = True
Qmgr: set queue server_2 resources_max.arch = sv2
Qmgr: set queue server_2 resources_min.arch = sv2
Qmgr: set queue server_2 acl_group_enable = True
Qmgr: set queue server_2 acl_groups = math
Qmgr: set queue server_2 acl_groups += chemistry
Qmgr: set queue server_2 acl_groups += physics
Qmgr: set queue server_2 enabled = True
Qmgr: set queue server_2 started = True
Qmgr: create queue cluster
Qmgr: set queue cluster queue_type = Route
Qmgr: set queue cluster from_route_only = True
Qmgr: set queue cluster resources_max.arch = linux
Qmgr: set queue cluster resources_min.arch = linux
Qmgr: set queue cluster route_destinations = long
Qmgr: set queue cluster route_destinations += short
Qmgr: set queue cluster route_destinations += medium
Qmgr: set queue cluster enabled = True
Qmgr: set queue cluster started = True
```

In the cluster queue above, you will notice the particular order of the three destination queues (long, short, medium). PBS will attempt to route a job into the destination queues in the order specified. Thus, we want PBS to first try the long queue (which will have an ACL on it), then the short queue (with its short time limits). Thus any jobs that had not been routed into any other queues (server or cluster) will end up in the medium cluster queue. Now to create the remaining queues.

```
Qmgr: create queue long
Qmgr: set queue long queue_type = Execution
Qmgr: set queue long from_route_only = True
Qmgr: set queue long resources_max.cput = 20:00:00
Qmgr: set queue long resources_max.walltime = 20:00:00
Qmgr: set queue long resources_min.cput = 02:00:00
Qmgr: set queue long resources_min.walltime = 03:00:00
Qmgr: set queue long acl_group_enable = True
Qmgr: set queue long acl_groups = astronomy
Qmgr: set queue long enabled = True
Qmgr: set queue long started = True
```

```
Qmgr: create queue short
Qmgr: set queue short queue_type = Execution
Qmgr: set queue short from_route_only = True
Qmgr: set queue short resources_max.cput = 01:00:00
Qmgr: set queue short resources_max.walltime = 01:00:00
Qmgr: set queue short enabled = True
Qmgr: set queue short started = True
Qmgr: create queue medium
Qmgr: set queue medium queue_type = Execution
Qmgr: set queue medium from_route_only = True
Qmgr: set queue medium enabled = True
Qmgr: set queue medium started = True
Qmgr: set server default_queue = submit
```

Notice that the `long` and `short` queues have time limits specified. This will ensure that jobs of certain sizes will enter (or be prevented from entering) these queues. The last queue, `medium`, has no limits, thus it will be able to accept any job that is not routed into any other queue.

Lastly, note the last line in the example above, which specified that the default queue is the new `submit` queue. This way users will simply submit their jobs with the resource and architecture requests, without specifying a queue, and PBS will route the job into the correct location. For example, if a user submitted a job with the following syntax, the job would be routed into the `server_2` queue:

```
qsub -l select=arch=sv2:ncpus=4 testjob
```

12.5 External Software License Management

PBS Professional can be configured to schedule jobs based on externally-controlled licensed software. A detailed example is provided in "Example of Floating, Externally-managed License with Features" on page 276 in the PBS Professional Administrator's Guide.

12.6 Multiple User ACL Example

A site may have a need to restrict individual users to particular queues. In the previous example we set up queues with group-based ACLs, in this example we show user-based ACLs. Say a site has two different groups of users, and wants to limit them to two separate queues (perhaps with different resource limits). The following example illustrates this.

```
Qmgr: create queue structure
Qmgr: set queue structure queue_type = Execution
Qmgr: set queue structure acl_user_enable = True
Qmgr: set queue structure acl_users = curly
Qmgr: set queue structure acl_users += jerry
Qmgr: set queue structure acl_users += larry
Qmgr: set queue structure acl_users += moe
Qmgr: set queue structure acl_users += tom
Qmgr: set queue structure resources_max.nodes = 48
Qmgr: set queue structure enabled = True
Qmgr: set queue structure started = True

Qmgr: create queue engine
Qmgr: set queue engine queue_type = Execution
Qmgr: set queue engine acl_user_enable = True
Qmgr: set queue engine acl_users = bill
Qmgr: set queue engine acl_users += bobby
Qmgr: set queue engine acl_users += chris
Qmgr: set queue engine acl_users += jim
Qmgr: set queue engine acl_users += mike
Qmgr: set queue engine acl_users += rob
Qmgr: set queue engine acl_users += scott
Qmgr: set queue engine resources_max.nodes = 12
Qmgr: set queue engine resources_max.walltime=04:00:00
Qmgr: set queue engine enabled = True
Qmgr: set queue engine started = True
```

13
Run Limit Error Messages

This chapter lists the error messages generated when limits are exceeded. See "Managing Resource Usage By Users, Groups, and Projects, at Server & Queues" on page 287 in the PBS Professional Administrator's Guide.

13.1 Run Limit Error Messages

When a job would exceed a limit by running, the job's comment field is set to one of the following messages. The following table shows the limit attribute, where the limit is applied, to whom the limit is applied, and the message.

Table 13-1: Job Run Limit Error Messages

Attribute	Where Applied	To What Applied	Message
max_run	queue	o: PBS_ALL	Not Running: Queue <queue name> job limit has been reached.
max_run	server	o: PBS_ALL	Not Running: Server job limit has been reached.
max_run	server	p:PBS_GENERIC	Not Running: Project has reached server running limit.
max_run	queue	p:PBS_GENERIC	Not Running: Project has reached queue<queue-name>'s running limit.
max_run	server	p:<project name>	Not Running: Server job limit reached for project <project name>
max_run	queue	p:<project name>	Not Running: Queue <queue-name> job limit reached for project <project name>
max_run	queue	g: PBS_GENERIC	Not Running: Group has reached queue <queue name> running limit.
max_run	server	g: PBS_GENERIC	Not Running: Group has reached server running limit.
max_run	queue	u: PBS_GENERIC	Not Running: User has reached queue <queue name> running job limit.
max_run	server	u: PBS_GENERIC	Not Running: User has reached server running job limit.
max_run	queue	g:<group name>	Queue <queue name> job limit reached for group <G>
max_run	server	g:<group name>	Server job limit reached for group <G>
max_run	queue	u:<user name>	Queue <queue name> job limit reached for user <U>
max_run	server	u:<user name>	Server job limit reached for user <U>
max_run_res	queue	o: PBS_ALL	Queue <queue name> job limit reached on resource <resource name>
max_run_res	server	o: PBS_ALL	Server job limit reached on resource <resource name>

Table 13-1: Job Run Limit Error Messages

Attribute	Where Applied	To What Applied	Message
max_run_res	queue	p:PBS_GENERIC	Not Running: Queue <queue name> per-project limit reached on resource <resource name>
max_run_res	server	p:PBS_GENERIC	Not Running: Server per-project limit reached on resource <resource name>
max_run_res	server	p:<project name>	Not Running: would exceed project <project_name>'s limit on resource <resource name> in complex
max_run_res	queue	p:<project name>	Not Running: would exceed project <project_name>'s limit on resource <resource name> in queue <queue-name>
max_run_res	queue	g: PBS_GENERIC	Queue <queue name> per-group limit reached on resource <resource name>
max_run_res	server	g: PBS_GENERIC	Server per-group limit reached on resource <resource name>
max_run_res	queue	u: PBS_GENERIC	Queue <queue name> per-user limit reached on resource <resource name>
max_run_res	server	u: PBS_GENERIC	Server per-user limit reached on resource <resource name>
max_run_res	queue	g:<group name>	would exceed group <G>'s limit on resource <resource name> in queue <queue name>
max_run_res	server	g:<group name>	would exceed group <G>'s limit on resource <resource name> in complex
max_run_res	queue	u:<user name>	would exceed user <U>'s limit on resource <resource name> in queue <queue name>
max_run_res	server	u:<user name>	would exceed user <U>'s limit on resource <resource name> in complex

14
Error Codes

The following table lists all the PBS error codes, their textual names, and a description of each.

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_NONE	0	No error
PBSE_UNKJOBID	15001	Unknown Job Identifier
PBSE_NOATTR	15002	Undefined Attribute
PBSE_ATTRRO	15003	Attempt to set READ ONLY attribute
PBSE_IVALREQ	15004	Invalid request
PBSE_UNKREQ	15005	Unknown batch request
PBSE_TOOMANY	15006	Too many submit retries
PBSE_PERM	15007	No permission
PBSE_BADHOST	15008	Access from host not allowed
PBSE_JOBEXIST	15009	Job already exists
PBSE_SYSTEM	15010	System error occurred
PBSE_INTERNAL	15011	Internal server error occurred
PBSE_REGROUTE	15012	Parent job of dependent in route queue
PBSE_UNKSIG	15013	Unknown signal name
PBSE_BADATVAL	15014	Bad attribute value
PBSE_MODATRRUN	15015	Cannot modify attribute in run state
PBSE_BADSTATE	15016	Request invalid for job state
PBSE_UNKQUE	15018	Unknown queue name
PBSE_BADCRED	15019	Invalid Credential in request
PBSE_EXPIRED	15020	Expired Credential in request
PBSE_QUNOENB	15021	Queue not enabled
PBSE_QACESS	15022	No access permission for queue

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_BADUSER	15023	Missing userID, username, or GID. Returned under following conditions: 1. User does not have a password entry (getpwnam() returns null). 2. User's UID is zero and root isn't allowed to run jobs (acl_roots). 3. PBS_O_HOST is not set in the job.
PBSE_HOPCOUNT	15024	Max hop count exceeded
PBSE_QUEEXIST	15025	Queue already exists
PBSE_ATTRTYPE	15026	Incompatible queue attribute type
PBSE_OBJBUSY	15027	Object Busy
PBSE_QUENBIG	15028	Queue name too long
PBSE_NOSUP	15029	Feature/function not supported
PBSE_QUENOEN	15030	Can't enable queue, lacking definition
PBSE_PROTOCOL	15031	Protocol (ASN.1) error. Message is distorted or truncated.
PBSE_BADATLST	15032	Bad attribute list structure
PBSE_NOCONNECTS	15033	No free connections
PBSE_NOSERVER	15034	No server to connect to
PBSE_UNKRESC	15035	Unknown resource
PBSE_EXCQRESC	15036	Job exceeds Queue resource limits
PBSE_QUENODFLT	15037	No Default Queue Defined
PBSE_NORERUN	15038	Job Not Rerunnable
PBSE_ROUTEREJ	15039	Route rejected by all destinations
PBSE_ROUTEEXPD	15040	Time in Route Queue Expired
PBSE_MOMREJECT	15041	Request to MoM failed
PBSE_BADSCRIPT	15042	(qsub) Cannot access script file
PBSE_STAGEIN	15043	Stage In of files failed
PBSE_RESCUNAV	15044	Resources temporarily unavailable
PBSE_BADGRP	15045	Bad Group specified
PBSE_MAXQUED	15046	Max number of jobs in queue
PBSE_CKPBSY	15047	Checkpoint Busy, may be retries
PBSE_EXLIMIT	15048	Limit exceeds allowable
PBSE_BADACCT	15049	Bad Account attribute value
PBSE_ALRDYEXIT	15050	Job already in exit state

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_NOCOPYFILE	15051	Job files not copied
PBSE_CLEANEDOUT	15052	Unknown job id after clean init
PBSE_NOSYNCMSTR	15053	No Master in Sync Set
PBSE_BADDEPEND	15054	Invalid dependency
PBSE_DUPLIST	15055	Duplicate entry in List
PBSE_DISPROTO	15056	Bad DIS based Request Protocol
PBSE_EXECTHERE (**Obsolete**)	15057	Cannot execute there (Obsolete; no longer used.)
PBSE_SISREJECT	15058	Sister rejected
PBSE_SISCOMM	15059	Sister could not communicate
PBSE_SVRDOWN	15060	Request rejected -server shutting down
PBSE_CKPSHORT	15061	Not all tasks could checkpoint
PBSE_UNKNODE	15062	Named vnode is not in the list
PBSE_UNKNODEATR	15063	Vnode attribute not recognized
PBSE_NONODES	15064	Server has no vnode list
PBSE_NODENBIG	15065	Node name is too big
PBSE_NODEEXIST	15066	Node name already exists
PBSE_BADNDATVAL	15067	Bad vnode attribute value
PBSE_MUTUALEX	15068	State values are mutually exclusive
PBSE_GMODERR	15069	Error(s) during global mod of vnodes
PBSE_NORELYMOM	15070	Could not contact MoM
PBSE_REV_NO_WALLTIME	15075	Reservation lacks walltime
Reserved	15076	Not used.
PBSE_TOOLATE	15077	Reservation submitted with a start time that has already passed
PBSE_IRESVE	15078	Internal reservation system error
PBSE_UNKRESVTYPE	15079	Unknown reservation type
PBSE_RESVEXIST	15080	Reservation already exists
PBSE_resvFail	15081	Reservation failed
PBSE_genBatchReq	15082	Batch request generation failed
PBSE_mgrBatchReq	15083	`qmgr` batch request failed
PBSE_UNKRESVID	15084	Unknown reservation ID
PBSE_delProgress	15085	Delete already in progress

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_BADTSPEC	15086	Bad time specification(s)
PBSE_RESVMSG	15087	So reply_text can return a msg
PBSE_BADNODESPEC	15089	Node(s) specification error
PBSE_LICENSEINV	15091	License is invalid
PBSE_RESVAUTH_H	15092	Host not authorized to make AR
PBSE_RESVAUTH_G	15093	Group not authorized to make AR
PBSE_RESVAUTH_U	15094	User not authorized to make AR
PBSE_R_UID	15095	Bad effective UID for reservation
PBSE_R_GID	15096	Bad effective GID for reservation
PBSE_IBMSPSWITCH	15097	IBM SP Switch error
PBSE_NOSCHEDULER	15099	Unable to contact scheduler
PBSE_RESCNOTSTR	15100	Resource is not of type string
PBSE_SSIGNON_UNSET_REJECT	15101	rejected if SVR_ssignon_enable not set
PBSE_SSIGNON_SET_REJECT	15102	rejected if SVR_ssignon_enable set
PBSE_SSIGNON_BAD_TRANSITION1	15103	bad attempt: true to false
PBSE_SSIGNON_NOCONNECT_DEST	15105	couldn't connect to destination host during a user migration request
PBSE_SSIGNON_NO_PASSWORD	15106	no per-user/per-server password
PBSE_MaxArraySize	15107	max array size exceeded
PBSE_INVALSELECTRESC	15108	resource invalid in select spec
PBSE_INVALJOBRESC	15109	invalid job resource
PBSE_INVALNODEPLACE	15110	node invalid w/place|select
PBSE_PLACENOSELECT	15111	cannot have place w/o select
PBSE_INDIRECTHOP	15112	too many indirect resource levels
PBSE_INDIRECTBT	15113	target resource undefined
PBSE_NGBLUEGENE	15114	No node_group_enable on BlueGene
PBSE_NODESTALE	15115	Cannot change state of stale vnode
PBSE_DUPRESC	15116	cannot dupe resource within a chunk
PBSE_CONNFULL	15117	server connection table full
PBSE_LICENSE_MIN_BADVAL	15118	bad value for pbs_license_min
PBSE_LICENSE_MAX_BADVAL	15119	bad value for pbs_license_max
PBSE_LICENSE_LINGER_BADVAL	15120	bad value for pbs_license_linger_time
PBSE_LICENSE_BAD_ACTION	15122	Not allowed action with licensing

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_BAD_FORMULA	15123	invalid sort formula
PBSE_BAD_FORMULA_KW	15124	invalid keyword in formula
PBSE_BAD_FORMULA_TYPE	15125	invalid resource type in formula
PBSE_BAD_RRULE_YEARLY	15126	reservation duration exceeds 1 year
PBSE_BAD_RRULE_MONTHLY	15127	reservation duration exceeds 1 month
PBSE_BAD_RRULE_WEEKLY	15128	reservation duration exceeds 1 week
PBSE_BAD_RRULE_DAILY	15129	reservation duration exceeds 1 day
PBSE_BAD_RRULE_HOURLY	15130	reservation duration exceeds 1 hour
PBSE_BAD_RRULE_MINUTELY	15131	reservation duration exceeds 1 minute
PBSE_BAD_RRULE_SECONDLY	15132	reservation duration exceeds 1 second
PBSE_BAD_RRULE_SYNTAX	15133	invalid recurrence rule syntax
PBSE_BAD_RRULE_SYNTAX2	15134	invalid recurrence rule syntax
PBSE_BAD_ICAL_TZ	15135	Undefined timezone info directory
PBSE_HOOKERROR	15136	error encountered related to hooks
PBSE_NEEDQUET	15137	need queue type set
PBSE_ETEERROR	15138	not allowed to alter attribute when eligible_time_enable is off
PBSE_HISTJOBID	15139	History job ID
PBSE_JOBHISTNOTSET	15140	job_history_enable not SET
PBSE_MIXENTLIMS	15141	mixing old and new limit enforcement
PBSE_HEADERROR	15145	Server host not allowed to be provisioned
PBSE_NODEPROV_NOACTION	15146	While provisioning, provisioning attributes can't be modified
PBSE_NODEPROV	15147	State of provisioning vnode can't be changed
PBSE_NODEPROV_NODEL	15148	Vnode can't be deleted while provisioning
PBSE_NODE_BAD_CURRENT_AOE	15149	Attempt to set an AOE that is not in resources_available.aoe
PBSE_NOTLOCALNODE	15150	Non-local node not allowed in Personal Mode (not used)
PBSE_MOM_INCOMPLETE_HOOK	15167	Execution hooks not fully transferred to a particular MoM
PBSE_MOM_REJECT_ROOT_SCRIPTS	15168	A MoM has rejected a request to copy a hook-related file, or a job script to be executed by root
PBSE_HOOK_REJECT	15169	A MoM received a reject result from a mom hook
PBSE_HOOK_REJECT_RERUNJOB	15170	Hook rejection requiring a job to be rerun
PBSE_HOOK_REJECT_DELETEJOB	15171	Hook rejection requiring a job to be deleted

Table 14-1: Error Codes

Error Name	Error Code	Description
PBSE_JOBNBIG	15173	Submitted job or reservation name is too long
	15178	Cannot alter start time of running, non-empty reservation
	15179	Cannot alter current or next occurrence of a standing reservation so that it interferes with a later occurrence
Resource monitor specific error codes		
PBSE_RMUNKNOWN	15201	Resource unknown
PBSE_RMBADPARAM	15202	Parameter could not be used
PBSE_RMNOPARAM	15203	A needed parameter did not exist
PBSE_RMEXIST	15204	Something specified didn't exist
PBSE_RMSYSTEM	15205	A system error occurred
PBSE_RMPART	15206	Only part of reservation made
PBSE_SSIGNON_BAD_TRANSITION2	15207	bad attempt: false to true
PBSE_TRYAGAIN	15208	Try the request again later
PBSE_ALPSRELERR	15209	PBS is unable to release the ALPS reservation
PBSE_NOTARRAY_ATTR	15231	Attempt to set max_run_subjobs for a non-array job

15
Request Codes

When reading the PBS event logfiles, you may see messages of the form "Type 19 request received from PBS_Server...". These "type codes" correspond to different PBS batch requests. The following table lists all the PBS type codes and the corresponding request of each.

Table 15-1: Request Codes

Numeric Value	Name
0	PBS_BATCH_Connect
1	PBS_BATCH_QueueJob
2	UNUSED
3	PBS_BATCH_jobscript
4	PBS_BATCH_RdytoCommit
5	PBS_BATCH_Commit
6	PBS_BATCH_DeleteJob
7	PBS_BATCH_HoldJob
8	PBS_BATCH_LocateJob
9	PBS_BATCH_Manager
10	PBS_BATCH_MessJob
11	PBS_BATCH_ModifyJob
12	PBS_BATCH_MoveJob
13	PBS_BATCH_ReleaseJob
14	PBS_BATCH_Rerun
15	PBS_BATCH_RunJob
16	PBS_BATCH_SelectJobs
17	PBS_BATCH_Shutdown
18	PBS_BATCH_SignalJob
19	PBS_BATCH_StatusJob
20	PBS_BATCH_StatusQue
21	PBS_BATCH_StatusSvr
22	PBS_BATCH_TrackJob
23	PBS_BATCH_AsyrunJob
24	PBS_BATCH_Rescq
25	PBS_BATCH_ReserveResc

Table 15-1: Request Codes

Numeric Value	Name
26	PBS_BATCH_ReleaseResc
27	PBS_BATCH_FailOver
48	PBS_BATCH_StageIn
49	PBS_BATCH_AuthenUser
50	PBS_BATCH_OrderJob
51	PBS_BATCH_SelStat
52	PBS_BATCH_RegistDep
54	PBS_BATCH_CopyFiles
55	PBS_BATCH_DelFiles
56	PBS_BATCH_JobObit
57	PBS_BATCH_MvJobFile
58	PBS_BATCH_StatusNode
59	PBS_BATCH_Disconnect
60	UNUSED
61	UNUSED
62	PBS_BATCH_JobCred
63	PBS_BATCH_CopyFiles_Cred
64	PBS_BATCH_DelFiles_Cred
65	PBS_BATCH_GSS_Context
66	UNUSED
67	UNUSED
68	UNUSED
69	UNUSED
70	PBS_BATCH_SubmitResv
71	PBS_BATCH_StatusResv
72	PBS_BATCH_DeleteResv
73	PBS_BATCH_UserCred
74	PBS_BATCH_UserMigrate
75	PBS_BATCH_ConfirmResv
80	PBS_BATCH_DefSchReply
81	PBS_BATCH_StatusSched
82	PBS_BATCH_StatusRsc
83	PBS_BATCH_StatusHook

Table 15-1: Request Codes

Numeric Value	Name
84	PBS_BATCH_PySpawn
85	PBS_BATCH_CopyHookFile
86	PBS_BATCH_DelHookFile
87	PBS_BATCH_MomRestart
88	PBS_BATCH_AuthExternal
89	PBS_BATCH_HookPeriodic
90	PBS_BATCH_RelnodesJob
91	PBS_BATCH_ModifyResv

16
PBS Environment Variables

The following table lists the PBS environment variables:

Table 16-1: PBS Environment Variables

Variable	Origin	Meaning
CONTAINER_IMAGE	Job submitter	Name of container image in which job is to run
NCPUS		Number of threads, defaulting to number of CPUs, on the vnode
NEC_PROCESS_DIST	Job submitter	For NEC SX-Aurora TSUBASA. Specifies process distribution. See "Specifying Process Distribution", on page 204 of the PBS Professional User's Guide.
OMP_NUM_THREADS		Same as NCPUS.
PBS_ARRAY_ID	Server	Identifier for job arrays. Consists of sequence number.
PBS_ARRAY_INDEX	Server	Index number of subjob in job array.
PBS_CONF_FILE		Path to `pbs.conf`
PBS_CONTAINER_ARGS	Job submitter	Arguments to pass to container engine. Multiple arguments are separated with a semicolon. When using this environment variable, the `-env` and `--entrypoint` arguments to `docker run` are not supported. To pass environment variables directly to PBS, use `qsub -v`.
PBS_DEFAULT		Name of default PBS server
PBS_DATA_SERVICE_USER	Admin, during installation	Account used by data service.
PBS_ENVIRONMENT		Indicates job type: *PBS_BATCH* or *PBS_INTERACTIVE*
PBS_JOBCOOKIE		Unique identifier for inter-MoM job-based communication.
PBS_JOBDIR		Pathname of job-specific staging and execution directory
PBS_JOBID	Server	The job identifier assigned to the job or job array by the batch system.
PBS_JOBNAME	User	The job name supplied by the user.
PBS_LICENSE_INFO	Admin	Location of license server

Table 16-1: PBS Environment Variables

Variable	Origin	Meaning
PBS_LOG_HIGHRES_TIMESTAMP		Controls whether daemons on this host log timestamps in microseconds. Default timestamp log format is *HH:MM:SS*. With microsecond logging, format is *HH:MM:SS:XXXXXX*. Does not affect accounting log. Not applicable when using `syslog`. Overrides configuration parameter in `pbs.conf` of the same name. Valid values: *0*, *1*. Default: *0* (no microsecond logging)
PBS_MOMPORT		Port number on which this job's MoMs will communicate.
PBS_NODEFILE		The filename containing a list of vnodes assigned to the job.
PBS_NODENUM		Index into $PBS_NODEFILE. Starts at zero.
PBS_O_HOME	Submission environment	Value of HOME from submission environment.
PBS_O_HOST	Submission environment; set by PBS	The host name on which the `qsub` command was executed.
PBS_O_LANG	Submission environment	Value of LANG from submission environment
PBS_O_LOGNAME	Submission environment	Value of LOGNAME from submission environment
PBS_O_MAIL	Submission environment	Value of MAIL from submission environment
PBS_O_PATH	Submission environment	Value of PATH from submission environment
PBS_O_QUEUE	Submission environment	The original queue name to which the job was submitted.
PBS_O_SHELL	Submission environment	Value of SHELL from submission environment
PBS_O_SYSTEM	Submission environment	The operating system name where `qsub` was executed.
PBS_O_TZ	Submission environment	Value of TZ from submission environment
PBS_O_WORKDIR	Submission environment	The absolute path of directory where `qsub` was executed.
PBS_QUEUE		The name of the queue from which the job is executed.
PBS_SCHED_THREADS		Maximum number of scheduler threads. Scheduler automatically caps number of threads at the number of cores (or hyperthreads if applicable), regardless of value of this variable. Overridden by `pbs_sched -t` option. Overrides PBS_SCHED_THREADS parameter in `pbs.conf`. Default: 1
PBS_SERVER	Submission environment	The name of the default PBS server.
PBS_SID		Session ID

Table 16-1: PBS Environment Variables

Variable	Origin	Meaning
PBS_TASKNUM		The task (process) number for the job on this vnode.
PBS_TMPDIR		Root of temporary directories/files for PBS components.
TMPDIR		The job-specific temporary directory for this job.

17
File Listing

The following table lists all the PBS files and directories; owner and permissions are specific to Linux systems.

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
/opt/pbs/default/etc/pbs_bootcheck.py	root	-rw-r--r--	4111
/var/tmp/pbs_bootcheck.py	root	-rw-r--r--	4111
/var/tmp/pbs_boot_check See "Discovering Last Reboot Time of Server" on page 556 in the PBS Professional Administrator's Guide.	root	-rw-r--r--	188
PBS_EXEC/	root	drwxr-xr-x	4096
PBS_EXEC/bin	root	drwxr-xr-x	4096
PBS_EXEC/bin/pbsdsh	root	-rwxr-xr-x	111837
PBS_EXEC/bin/pbsnodes	root	-rwxr-xr-x	153004
PBS_EXEC/bin/pbs_dataservice	root	-rwx------	
PBS_EXEC/bin/pbs_hostn	root	-rwxr-xr-x	35493
PBS_EXEC/bin/pbs_rdel	root	-rwxr-xr-x	151973
PBS_EXEC/bin/pbs_rstat	root	-rwxr-xr-x	156884
PBS_EXEC/bin/pbs_rsub	root	-rwxr-xr-x	167446
PBS_EXEC/bin/pbs_tclsh	root	-rwxr-xr-x	857552
PBS_EXEC/bin/pbs_wish	root	-rwxr-xr-x	1592236
PBS_EXEC/bin/printjob	root	-rwxr-xr-x	42667
PBS_EXEC/bin/qalter	root	-rwxr-xr-x	210723
PBS_EXEC/bin/qdel	root	-rwxr-xr-x	164949
PBS_EXEC/bin/qdisable	root	-rwxr-xr-x	139559
PBS_EXEC/bin/qenable	root	-rwxr-xr-x	139558
PBS_EXEC/bin/qhold	root	-rwxr-xr-x	165368
PBS_EXEC/bin/qmgr	root	-rwxr-xr-x	202526
PBS_EXEC/bin/qmove	root	-rwxr-xr-x	160932
PBS_EXEC/bin/qmsg	root	-rwxr-xr-x	160408
PBS_EXEC/bin/qorder	root	-rwxr-xr-x	146393
PBS_EXEC/bin/qrerun	root	-rwxr-xr-x	157228

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/bin/qrls	root	-rwxr-xr-x	165361
PBS_EXEC/bin/qrun	root	-rwxr-xr-x	160978
PBS_EXEC/bin/qselect	root	-rwxr-xr-x	163266
PBS_EXEC/bin/qsig	root	-rwxr-xr-x	160083
PBS_EXEC/bin/qstart	root	-rwxr-xr-x	139589
PBS_EXEC/bin/qstat	root	-rwxr-xr-x	207532
PBS_EXEC/bin/qstop	root	-rwxr-xr-x	139584
PBS_EXEC/bin/qsub	root	-rwxr-xr-x	275460
PBS_EXEC/bin/qterm	root	-rwxr-xr-x	132188
PBS_EXEC/bin/tracejob	root	-rwxr-xr-x	64730
PBS_EXEC/etc	root	drwxr-xr-x	4096
PBS_EXEC/etc/modulefile	root	-rw-r--r--	749
PBS_EXEC/etc/pbs_db_schema.sql	root	-rw-r--r--	10522
PBS_EXEC/etc/pbs_dedicated	root	-rw-r--r--	557
PBS_EXEC/etc/pbs_holidays	root	-rw-r--r--	2612
PBS_EXEC/etc/pbs_holidays.<year>	root	-rw-r--r--	2643
PBS_EXEC/etc/pbs_resource_group	root	-rw-r--r--	657
PBS_EXEC/etc/pbs_sched_config	root	-r--r--r--	9791
PBS_EXEC/include	root	drwxr-xr-x	4096
PBS_EXEC/include/pbs_error.h	root	-r--r--r--	7543
PBS_EXEC/include/pbs_ifl.h	root	-r--r--r--	17424
PBS_EXEC/include/rm.h	root	-r--r--r--	740
PBS_EXEC/include/tm.h	root	-r--r--r--	2518
PBS_EXEC/include/tm_.h	root	-r--r--r--	2236
PBS_EXEC/lib	root	drwxr-xr-x	4096
PBS_EXEC/lib/libattr.a	root	-rw-r--r--	390274
PBS_EXEC/lib/liblog.a	root	-rw-r--r--	101230
PBS_EXEC/lib/libnet.a	root	-rw-r--r--	145968
PBS_EXEC/lib/libpbs.a	root	-rw-r--r--	1815486
PBS_EXEC/lib/libsite.a	root	-rw-r--r--	132906
PBS_EXEC/lib/MPI	root	drwxr-xr-x	4096
PBS_EXEC/lib/MPI/pbsrun.ch_gm.init.in	root	-rw-r--r--	9924

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/lib/MPI/pbsrun.ch_mx.init.in	root	-rw-r--r--	9731
PBS_EXEC/lib/MPI/pbsrun.gm_mpd.init.in	root	-rw-r--r--	10767
PBS_EXEC/lib/MPI/pbsrun.intelmpi.init.in	root	-rw-r--r--	10634
PBS_EXEC/lib/MPI/pbsrun.mpich2.init.in	root	-rw-r--r--	10694
PBS_EXEC/lib/MPI/pbsrun.mx_mpd.init.in	root	-rw-r--r--	10770
PBS_EXEC/lib/MPI/sgiMPI.awk	root	-rw-r--r--	6564
PBS_EXEC/lib/pbs_sched.a	root	-rw-r--r--	822026
PBS_EXEC/lib/pm	root	drwxr--r--	4096
PBS_EXEC/lib/pm/PBS.pm	root	-rw-r--r--	3908
PBS_EXEC/libexec/au-nodeupdate.pl	root	-rw-r--r--	
PBS_EXEC/libexec/install_db	root	-rwx------	10506
PBS_EXEC/libexec/pbs_habitat	root	-rwx------	10059
PBS_EXEC/libexec/pbs_init.d	root	-rwx------	25568
PBS_EXEC/libexec/pbs_postinstall	root	-rwx------	29104
PBS_EXEC/share/man	root	drwxr-xr-x	4096
PBS_EXEC/share/man/man1	root	drwxr-xr-x	4096
PBS_EXEC/share/man/man1/pbs.1B	root	-rw-r--r--	5376
PBS_EXEC/share/man/man1/pbsdsh.1B	root	-rw-r--r--	2978
PBS_EXEC/share/man/man1/pbs_ralter.1B	root	-rw-r--r--	
PBS_EXEC/share/man/man1/pbs_rdel.1B	root	-rw-r--r--	2342
PBS_EXEC/share/man/man1/pbs_rstat.1B	root	-rw-r--r--	2682
PBS_EXEC/share/man/man1/pbs_rsub.1B	root	-rw-r--r--	9143
PBS_EXEC/share/man/man1/qalter.1B	root	-rw-r--r--	21569
PBS_EXEC/share/man/man1/qdel.1B	root	-rw-r--r--	3363
PBS_EXEC/share/man/man1/qhold.1B	root	-rw-r--r--	4323
PBS_EXEC/share/man/man1/qmove.1B	root	-rw-r--r--	3343
PBS_EXEC/share/man/man1/qmsg.1B	root	-rw-r--r--	3244
PBS_EXEC/share/man/man1/qorder.1B	root	-rw-r--r--	3028
PBS_EXEC/share/man/man1/qrerun.1B	root	-rw-r--r--	2965
PBS_EXEC/share/man/man1/qrls.1B	root	-rw-r--r--	3927
PBS_EXEC/share/man/man1/qselect.1B	root	-rw-r--r--	12690
PBS_EXEC/share/man/man1/qsig.1B	root	-rw-r--r--	3817

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/share/man/man1/qstat.1B	root	-rw-r--r--	15274
PBS_EXEC/share/man/man1/qsub.1B	root	-rw-r--r--	36435
PBS_EXEC/share/man/man3	root	drwxr-xr-x	4096
PBS_EXEC/share/man/man3/pbs_alterjob.3B	root	-rw-r--r--	5475
PBS_EXEC/share/man/man3/pbs_connect.3B	root	-rw-r--r--	3493
PBS_EXEC/share/man/man3/pbs_default.3B	root	-rw-r--r--	2150
PBS_EXEC/share/man/man3/pbs_deljob.3B	root	-rw-r--r--	3081
PBS_EXEC/share/man/man3/pbs_disconnect.3B	root	-rw-r--r--	1985
PBS_EXEC/share/man/man3/pbs_geterrmsg.3B	root	-rw-r--r--	2473
PBS_EXEC/share/man/man3/pbs_holdjob.3B	root	-rw-r--r--	3006
PBS_EXEC/share/man/man3/pbs_manager.3B	root	-rw-r--r--	4337
PBS_EXEC/share/man/man3/pbs_movejob.3B	root	-rw-r--r--	3220
PBS_EXEC/share/man/man3/pbs_msgjob.3B	root	-rw-r--r--	2912
PBS_EXEC/share/man/man3/pbs_orderjob.3B	root	-rw-r--r--	2526
PBS_EXEC/share/man/man3/pbs_rerunjob.3B	root	-rw-r--r--	2531
PBS_EXEC/share/man/man3/pbs_rlsjob.3B	root	-rw-r--r--	3043
PBS_EXEC/share/man/man3/pbs_runjob.3B	root	-rw-r--r--	3484
PBS_EXEC/share/man/man3/pbs_selectjob.3B	root	-rw-r--r--	7717
PBS_EXEC/share/man/man3/pbs_sigjob.3B	root	-rw-r--r--	3108
PBS_EXEC/share/man/man3/pbs_statjob.3B	root	-rw-r--r--	4618
PBS_EXEC/share/man/man3/pbs_statnode.3B	root	-rw-r--r--	3925
PBS_EXEC/share/man/man3/pbs_statque.3B	root	-rw-r--r--	4009
PBS_EXEC/share/man/man3/pbs_statserver.3B	root	-rw-r--r--	3674
PBS_EXEC/share/man/man3/pbs_submit.3B	root	-rw-r--r--	6320
PBS_EXEC/share/man/man3/pbs_submitresv.3B	root	-rw-r--r--	3878
PBS_EXEC/share/man/man3/pbs_terminate.3B	root	-rw-r--r--	3322
PBS_EXEC/share/man/man3/tm.3B	root	-rw-r--r--	11062
PBS_EXEC/share/man/man7	root	drwxr-xr-x	4096
PBS_EXEC/share/man/man7/pbs_job_attributes.7B	root	-rw-r--r--	15920
PBS_EXEC/share/man/man7/pbs_node_attributes.7B	root	-rw-r--r--	7973
PBS_EXEC/share/man/man7/pbs_queue_attributes.7B	root	-rw-r--r--	11062
PBS_EXEC/share/man/man7/pbs_resources.7B	root	-rw-r--r--	22124

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/share/man/man7/pbs_resv_attributes.7B	root	-rw-r--r--	11662
PBS_EXEC/share/man/man7/pbs_server_attributes.7B	root	-rw-r--r--	14327
PBS_EXEC/share/man/man8	root	drwxr-xr-x	4096
PBS_EXEC/share/man/man8/mpiexec.8B	root	-rw-r--r--	4701
PBS_EXEC/share/man/man8/pbs-report.8B	root	-rw-r--r--	19221
PBS_EXEC/share/man/man8/pbsfs.8B	root	-rw-r--r--	3703
PBS_EXEC/share/man/man8/pbsnodes.8B	root	-rw-r--r--	3441
PBS_EXEC/share/man/man8/pbsrun.8B	root	-rw-r--r--	20937
PBS_EXEC/share/man/man8/pbsrun_unwrap.8B	root	-rw-r--r--	2554
PBS_EXEC/share/man/man8/pbsrun_wrap.8B	root	-rw-r--r--	3855
PBS_EXEC/share/man/man8/pbs_attach.8B	root	-rw-r--r--	3790
PBS_EXEC/share/man/man8/pbs_hostn.8B	root	-rw-r--r--	2781
PBS_EXEC/share/man/man8/pbs_idled.8B	root	-rw-r--r--	2628
PBS_EXEC/share/man/man8/pbs_lamboot.8B	root	-rw-r--r--	2739
PBS_EXEC/share/man/man8/pbs_mom.8B	root	-rw-r--r--	23496
PBS_EXEC/share/man/man8/pbs_mpihp.8B	root	-rw-r--r--	4120
PBS_EXEC/share/man/man8/pbs_mpilam.8B	root	-rw-r--r--	2647
PBS_EXEC/share/man/man8/pbs_mpirun.8B	root	-rw-r--r--	3130
PBS_EXEC/share/man/man8/pbs_probe.8B	root	-rw-r--r--	3344
PBS_EXEC/share/man/man8/pbs_sched_cc.8B	root	-rw-r--r--	6731
PBS_EXEC/share/man/man8/pbs_server.8B	root	-rw-r--r--	7914
PBS_EXEC/share/man/man8/pbs_tclsh.8B	root	-rw-r--r--	2475
PBS_EXEC/share/man/man8/pbs_tmrsh.8B	root	-rw-r--r--	3556
PBS_EXEC/share/man/man8/pbs_wish.8B	root	-rw-r--r--	2123
PBS_EXEC/share/man/man8/printjob.8B	root	-rw-r--r--	2823
PBS_EXEC/share/man/man8/qdisable.8B	root	-rw-r--r--	3104
PBS_EXEC/share/man/man8/qenable.8B	root	-rw-r--r--	2937
PBS_EXEC/share/man/man8/qmgr.8B	root	-rw-r--r--	7282
PBS_EXEC/share/man/man8/qrun.8B	root	-rw-r--r--	2850
PBS_EXEC/share/man/man8/qstart.8B	root	-rw-r--r--	2966
PBS_EXEC/share/man/man8/qstop.8B	root	-rw-r--r--	2963
PBS_EXEC/share/man/man8/qterm.8B	root	-rw-r--r--	4839

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_EXEC/share/man/man8/tracejob.8B	root	-rw-r--r--	4664
PBS_EXEC/pgsql	root	-rwxr-xr-x	
PBS_EXEC/sbin	root	drwxr-xr-x	4096
PBS_EXEC/sbin/pbs-report	root	-rwxr-xr-x	68296
PBS_EXEC/sbin/pbsfs	root	-rwxr-xr-x	663707
PBS_EXEC/sbin/pbs_demux	root	-rwxr-xr-x	38688
PBS_EXEC/sbin/pbs_idled	root	-rwxr-xr-x	99373
PBS_EXEC/sbin/pbs_iff	root	-rwsr-xr-x	133142
PBS_EXEC/sbin/pbs_mom	root	-rwx------	839326
PBS_EXEC/sbin/pbs_probe	root	-rwsr-xr-x	83108
PBS_EXEC/sbin/pbs_rcp	root	-rwsr-xr-x	75274
PBS_EXEC/sbin/pbs_sched	root	-rwx------	705478
PBS_EXEC/sbin/pbs_server	root	-rwx------	1133650
PBS_EXEC/tcltk	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/bin/tclsh8.3	root	-rw-r--r--	552763
PBS_EXEC/tcltk/bin/wish8.3	root	-rw-r--r--	1262257
PBS_EXEC/tcltk/include	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/include/tcl.h	root	-rw-r--r--	57222
PBS_EXEC/tcltk/include/tclDecls.h	root	-rw-r--r--	123947
PBS_EXEC/tcltk/include/tk.h	root	-rw-r--r--	47420
PBS_EXEC/tcltk/include/tkDecls.h	root	-rw-r--r--	80181
PBS_EXEC/tcltk/lib	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/libtcl8.3.a	root	-rw-r--r--	777558
PBS_EXEC/tcltk/lib/libtclstub8.3.a	root	-rw-r--r--	1832
PBS_EXEC/tcltk/lib/libtk8.3.a	root	-rw-r--r--	1021024
PBS_EXEC/tcltk/lib/libtkstub8.3.a	root	-rw-r--r--	3302
PBS_EXEC/tcltk/lib/tcl8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tclConfig.sh	root	-rw-r--r--	7076
PBS_EXEC/tcltk/lib/tk8.3	root	drwxr-xr-x	4096
PBS_EXEC/tcltk/lib/tkConfig.sh	root	-rw-r--r--	3822
PBS_EXEC/tcltk/license.terms	root	-rw-r--r--	2233

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_HOME	root	drwxr-xr-x	4096
PBS_HOME/aux	root	drwxr-xr-x	4096
PBS_HOME/checkpoint	root	drwx------	4096
PBS_HOME/datastore	data service account	-rwx------	
PBS_HOME/mom_logs	root	drwxr-xr-x	4096
PBS_HOME/mom_priv	root	drwxr-x--x	4096
PBS_HOME/mom_priv/config	root	-rw-r--r--	18
PBS_HOME/mom_priv/jobs	root	drwxr-x--x	4096
PBS_HOME/mom_priv/mom.lock	root	-rw-r--r--	4
PBS_HOME/pbs_environment	root	-rw-r--r--	0
PBS_HOME/sched_log	root	drwxr-xr-x	4096
PBS_HOME/sched_priv	root	drwxr-x---	4096
PBS_HOME/sched_priv/dedicated_time	root	-rw-r--r--	557
PBS_HOME/sched_priv/holidays	root	-rw-r--r--	1228
PBS_HOME/sched_priv/resource_group	root	-rw-r--r--	0
PBS_HOME/sched_priv/sched.lock	root	-rw-r--r--	4
PBS_HOME/sched_priv/sched_config	root	-rw-r--r--	6370
PBS_HOME/sched_priv/sched_out	root	-rw-r--r--	0
PBS_HOME/server_logs	root	drwxr-xr-x	4096
PBS_HOME/server_priv	root	drwxr-x---	4096
PBS_HOME/server_priv/accounting	root	drwxr-xr-x	4096
PBS_HOME/server_priv/acl_groups	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_hosts	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr	root	drwxr-x---	4096
PBS_HOME/server_priv/acl_svr/managers	root	-rw-------	13
PBS_HOME/server_priv/acl_users	root	drwxr-x---	4096
PBS_HOME/server_priv/config			
PBS_HOME/server_priv/db_user			
PBS_HOME/server_priv/db_password			
PBS_HOME/server_priv/hooks			
PBS_HOME/server_priv/jobs	root	drwxr-x---	4096

Table 17-1: File Listing

Directory / File	Owner	Permission	Average Size
PBS_HOME/server_priv/license_file	root	-rw-r--r--	34
PBS_HOME/server_priv/nodes			
PBS_HOME/server_priv/queues/newqueue	root	-rw-------	303
PBS_HOME/server_priv/queues/workq	root	-rw-------	303
PBS_HOME/server_priv/resourcedef	root		
PBS_HOME/server_priv/server.lock	root	-rw-------	4
PBS_HOME/server_priv/svrlive	root	-rw-------	
PBS_HOME/server_priv/tracking	root	-rw-------	0
PBS_HOME/spool	root	drwxrwxrwt	4096
PBS_HOME/undelivered	root	drwxrwxrwt	4096

18 Introduction to PBS

18.1 Acknowledgements

PBS Professional is the enhanced commercial version of the PBS software originally developed for NASA. The NASA version had a number of corporate and individual contributors over the years, for which the PBS developers and PBS community is most grateful. Below we provide formal legal acknowledgements to corporate and government entities, then special thanks to individuals.

The NASA version of PBS contained software developed by NASA Ames Research Center, Lawrence Livermore National Laboratory, and MRJ Technology Solutions. In addition, it included software developed by the NetBSD Foundation, Inc., and its contributors as well as software developed by the University of California, Berkeley and its contributors.

Other contributors to the NASA version of PBS include Bruce Kelly and Clark Streeter of NERSC; Kent Crispin and Terry Heidelberg of LLNL; John Kochmar and Rob Pennington of Pittsburgh Supercomputing Center; and Dirk Grunwald of University of Colorado, Boulder. The ports of PBS to the Cray T3e and the IBM SP SMP were funded by DoD USAERDC; the port of PBS to the Cray SV1 was funded by DoD MSIC.

No list of acknowledgements for PBS would possibly be complete without special recognition of the first two beta test sites. Thomas Milliman of the Space Sciences Center of the University of New Hampshire was the first beta tester. Wendy Lin of Purdue University was the second beta tester and holds the honor of submitting more problem reports than anyone else outside of NASA.

Index

$action RG-240
$alps_client RG-240
$alps_release_jitter RG-240
$alps_release_timeout RG-240
$alps_release_wait_time RG-241
$checkpoint_path RG-241
$clienthost RG-241
$cputmult RG-241
$dce_refresh_delta RG-241
$enforce RG-241
$job_launch_delay RG-243
$jobdir_root RG-242
$logevent RG-244
$max_check_poll RG-244
$max_load RG-244
$max_poll_downtime RG-244
$min_check_poll RG-244
$pbs_accounting_workload_mgmt RG-245
$prologalarm RG-245
$reject_root_scripts RG-245
$restart_background RG-245
$restart_transmogrify RG-245
$restrict_user RG-245
$restrict_user_exceptions RG-245
$restrict_user_maxsysid RG-246
$restricted RG-246
$sister_join_job_alarm RG-246
$suspendsig RG-246
$tmpdir RG-246
$usecp RG-246
$vnodedef_additive RG-247
$wallmult RG-247

A

accelerator RG-265
accelerator_memory RG-266
accelerator_model RG-266
accept an action RG-1
access
    by group RG-8
    by user RG-20
    from host RG-8
    to a queue RG-1
    to a reservation RG-1
    to the server RG-1
access control list RG-1
account string RG-1
Account_Name
    job attribute RG-328
accounting
    account string RG-1
accounting log entry
    format RG-353
accounting_id
    job attribute RG-328
accrue_type
    job attribute RG-328
ACL RG-1, RG-379, RG-382, RG-383, RG-385
acl_group_enable
    queue attribute RG-311
acl_groups
    queue attribute RG-311
acl_host_enable RG-281
    queue attribute RG-311
acl_host_moms_enable RG-281
acl_hosts
    queue attribute RG-311
    server attribute RG-281
acl_resv_group_enable
    server attribute RG-281
acl_resv_groups
    server attribute RG-281
acl_resv_host_enable
    server attribute RG-281
acl_resv_hosts
    server attribute RG-282
acl_resv_user_enable
    server attribute RG-282
acl_resv_users
    server attribute RG-282
acl_roots
    server attribute RG-282
acl_user_enable
    queue attribute RG-311
    server attribute RG-282
acl_users
    queue attribute RG-311
    server attribute RG-282
action RG-1
    accept RG-1
    reject RG-16
active (failover) RG-1
Active Directory RG-1

Admin RG-1
administrator RG-2
Administrators RG-2
advance reservation RG-2, RG-392
aggressive_provision RG-255
alarm
    hook attribute RG-350
ALM license server RG-2
alt_id
    job attribute RG-328
Ames Research Center RG-411
AOE RG-2
aoe RG-266
API RG-2
application checkpoint RG-2
application operating environment RG-2
arch RG-267
argument_list
    job attribute RG-329
array
    job attribute RG-329
array job RG-2, RG-9
array_id
    job attribute RG-329
array_index
    job attribute RG-329
array_indices_remaining
    job attribute RG-329
array_indices_submitted
    job attribute RG-329
array_state_count
    job attribute RG-330
ASAP reservation RG-2, RG-10
attribute
    definition RG-2
    log_events RG-298
    rerunnable RG-16
attribute name
    format RG-353
Authorized_Groups
    reservation attribute RG-303
Authorized_Hosts
    reservation attribute RG-303
Authorized_Users
    reservation attribute RG-304
avoid_provision RG-255

B

backfill RG-250
backfill_depth
    queue attribute RG-311
    server attribute RG-282
backfill_prime RG-250
backfilling RG-2
batch job RG-9
batch processing RG-3
block
    job attribute RG-330
Boolean
    format RG-259, RG-359
borrowing vnode RG-3
built-in hook RG-3
built-in resource RG-3
busy RG-365
by_queue RG-250

C

Checkpoint
    job attribute RG-330
checkpoint RG-240, RG-390, RG-409
    restart RG-17
    restart file RG-17
    restart script RG-17
checkpoint and abort RG-3
checkpoint and restart RG-3
checkpoint/restart RG-3
checkpoint_abort RG-3, RG-240
checkpoint_min
    queue attribute RG-312
child vnode RG-3
chunk RG-3
chunk set RG-3
chunk-level resource RG-3
cluster RG-3
comm RG-4
commands RG-4
comment
    job attribute RG-331
    scheduler attribute RG-298
    server attribute RG-283
    vnode attribute RG-320
communication daemon RG-4
complex RG-4
    Linux-Windows RG-11
    mixed-mode RG-12
    Windows-Linux RG-20
configuration file
    version 1 RG-20
    version 2 RG-20
consumable resource RG-4
CPU RG-4
cpus_per_ssinode RG-250
cput RG-267
creating a hook RG-4
ctime
    job attribute RG-331

reservation attribute RG-304
current_aoe
vnode attribute RG-320
current_eoe RG-320
custom resource RG-4

D

data service account RG-4
data service management account RG-4
date
format RG-353
datetime
format RG-354
debug
hook attribute RG-350
dedicated_prefix RG-250
default server RG-5
default_chunk
queue attribute RG-312
server attribute RG-283
default_qdel_arguments
server attribute RG-283
default_qsub_arguments
server attribute RG-283
default_queue
server attribute RG-283
degraded reservation RG-16
delegation RG-5
delete_idle_time RG-304
depend
job attribute RG-332
destination
definition RG-5
destination identifier RG-5
format RG-354
destination queue RG-5
destination server RG-5
directive RG-6
directory
staging and execution RG-19
DIS RG-369
do_not_span_psets
scheduler attribute RG-298
Domain Admin Account RG-6
Domain Admins RG-6
Domain User Account RG-6
Domain Users RG-6
down RG-365

E

egroup
job attribute RG-332
eligible_time
job attribute RG-333
eligible_time_enable
server attribute RG-284
enabled
hook attribute RG-350
queue attribute RG-312
endpoint RG-6
energy RG-267
Enterprise Admins RG-6
entity RG-6
entity share RG-6
environment variables RG-399
eoe RG-267
error codes RG-389
Error_Path
job attribute RG-333
est_start_time_freq
server attribute RG-284
estimated
job attribute RG-334
etime
job attribute RG-334
euser
job attribute RG-334
event RG-6
hook attribute RG-351
exec_host
job attribute RG-335
exec_vnode RG-267
job attribute RG-335
executable
job attribute RG-334
execution event hooks RG-6
execution host RG-6
execution queue RG-6
Execution_Time
job attribute RG-335
Exit_status
job attribute RG-336
express_queue RG-300
externally-provided resources RG-239

F

fail_action
hook attribute RG-352
failover RG-6
idle RG-8
primary scheduler RG-15
primary server RG-15
secondary scheduler RG-17
secondary server RG-17
failure action RG-7
fair_share RG-250

fairshare RG-7, RG-300
fairshare_decay_factor RG-251
fairshare_decay_time RG-251
fairshare_enforce_no_shares RG-251
fairshare_entity RG-251
fairshare_perc RG-252
fairshare_usage_res RG-251
file RG-268
  stage in RG-18
  stage out RG-18
  vnodedefs RG-20
file staging RG-7
files
  MoM
    config RG-381
  nodes RG-380
finished jobs RG-7
flatuid
  server attribute RG-284
FLicenses
  server attribute RG-284
float
  format RG-259, RG-359
floating license RG-7
format
  accounting log entry RG-353
  attribute name RG-353
  Boolean RG-259, RG-359
  date RG-353
  datetime RG-354
  destination identifier RG-354
  float RG-259, RG-359
  host name RG-354
  job array identifier RG-354
  job array name RG-355
  job array range RG-355
  job identifier RG-355, RG-357
  job name RG-355
  limit specification RG-356
  logfile-date-time RG-356
  pathname RG-357
  PBS NAME RG-357
  PBS password RG-357
  project name RG-357
  queue identifier RG-357
  queue name RG-357
  reservation name RG-358
  size RG-260, RG-360
  string resource value RG-260, RG-360
  string_array RG-260, RG-360
  subjob identifier RG-358
  username RG-358
    Windows RG-358
  vnode name RG-358
forward_x11_cookie
  job attribute RG-336
forward_x11_port
  job attribute RG-336
free RG-365
freq
  hook attribute RG-352
from_route_only
  queue attribute RG-312
furnishing queue RG-7

G

global resource RG-7
group RG-7
  access RG-8
  ID (GID) RG-8
group limit RG-8
group_list
  job attribute RG-336

H

half_life RG-251
hasnodes
  queue attribute RG-312
hbmem RG-268
help_starving_jobs RG-251
history jobs RG-8
hold RG-8
Hold_Types
  job attribute RG-336
hook RG-8
  creating RG-4
  importing RG-8
  provisioning RG-16
hooks
  accept RG-1
  action RG-1
  execution event RG-6
  non-job event RG-12
  pre-execution event RG-15
  reject action RG-16
host RG-8, RG-268
  access RG-8
host name
  format RG-354
hostname RG-8
Hot_Start
  server state RG-364
HTT RG-8

I

Idle
  server state RG-364

idle (failover) RG-8
importing a hook RG-8
in_multivnode_host
    vnode attribute RG-320
index
    subjob RG-19
indirect resource RG-9
InfiniBand RG-48, RG-49
installation account RG-9
instance RG-13
interactive
    job attribute RG-337
    reservation attribute RG-305
interactive job RG-9

J

job
    attribute RG-16
    batch RG-9
    identifier RG-10
    interactive RG-9
    kill RG-11
    owner RG-13
    rerunnable RG-16
    route RG-17
    shrink-to-fit RG-18
    state RG-10
    states RG-361
    substates RG-362
job array RG-9
    identifier RG-9
    range RG-9
    subjob RG-19
    subjob index RG-19
job array identifier
    format RG-354
job array name RG-10
    format RG-355
job array range
    format RG-355
job ID RG-10
job identifier
    format RG-355, RG-357
job name RG-10
    format RG-355
Job Submission Description Language RG-10
job_history_duration
    server attribute RG-284
job_history_enable
    server attribute RG-285
Job_Name
    job attribute RG-337
Job_Owner
    job attribute RG-337
job_priority RG-252
job_requeue_timeout
    server attribute RG-285
job_sort_formula
    server attribute RG-285
job_sort_formula_threshold
    scheduler attribute RG-298
job_sort_key RG-252
job_state
    job attribute RG-338
job-busy RG-365
jobdir
    job attribute RG-337
job-exclusive RG-365
jobs
    moved RG-12
    vnode attribute RG-320
jobscript_max_size
    server attribute RG-285
job-specific ASAP reservation RG-2, RG-10
job-specific now reservation RG-10, RG-13
job-specific reservation RG-10
Job-specific start reservation RG-10
job-specific start reservation RG-19
job-wide resource RG-11
Join_Path
    job attribute RG-339
JSDL RG-10

K

Keep_Files
    job attribute RG-339
kill job RG-11
kill_delay
    queue attribute RG-313

L

last_state_change_time RG-320
last_used_time RG-321
leaf RG-11
license
    external RG-384
    vnode attribute RG-321
license server RG-11
    ALM RG-2
license server configuration
    redundant RG-16
License Server List Configuration RG-11
license_info
    vnode attribute RG-321
limit RG-11
    generic group limit RG-7

generic project limit RG-7
generic user limit RG-7
group limit RG-8
individual group limit RG-9
individual project limit RG-9
individual user limit RG-9
overall RG-13
project RG-15
user limit RG-20
limit specification
format RG-356
Linux-Windows complex RG-11
load balance RG-11
load_balancing RG-253
load_balancing_rr RG-253
local resource RG-12
log_events
scheduler attribute RG-298
server attribute RG-285
log_filter RG-253
logfile-date-time
format RG-356

M

mail_from
server attribute RG-286
Mail_Points
job attribute RG-339
reservation attribute RG-305
Mail_Users
job attribute RG-339
reservation attribute RG-305
maintenance RG-365
maintenance_jobs RG-321
Manager RG-12
managers
server attribute RG-286
managing vnode RG-12
master provisioning script RG-12
master script RG-12
max_array_size
queue attribute RG-313
server attribute RG-286
max_concurrent_provision
server attribute RG-286
max_group_res
queue attribute RG-313
max_group_res_soft
queue attribute RG-313
max_group_run
queue attribute RG-313
max_group_run_soft
queue attribute RG-313
max_job_sequence_id RG-287
max_queuable
queue attribute RG-314
max_queued
queue attribute RG-314
max_queued_res
queue attribute RG-314
max_run
queue attribute RG-314
max_run_res
queue attribute RG-314
max_run_res_soft
queue attribute RG-315
max_run_soft
queue attribute RG-315
max_run_subjobs RG-340
max_running
queue attribute RG-315
max_starve RG-253
max_user_res
queue attribute RG-315
max_user_res_soft
queue attribute RG-315
max_user_run
queue attribute RG-316
max_user_run_soft
queue attribute RG-316
max_walltime RG-268
mem RG-268
mem_per_ssinode RG-253
memory-only vnode RG-12
memreserved RG-244
min_walltime RG-269
mixed-mode complex RG-12
MoM RG-12
subordinate RG-19
Mom
vnode attribute RG-321
mom_resources RG-253
monitoring RG-12
Mother Superior RG-12
moved jobs RG-12
mpiprocs RG-269
MRJ Technology Solutions RG-411
mtime
job attribute RG-340
reservation attribute RG-306
multinodebusy RG-240
multi-vnode complex RG-380

N

naccelerators RG-269
name

vnode attribute RG-321
NASA
and PBS RG-411
nchunk RG-270
NCPUS RG-399
ncpus RG-270
nice RG-270
no_multinode_jobs
vnode attribute RG-322
no_stdio_sockets
job attribute RG-340
node
definition RG-14
node_group_key
queue attribute RG-316
server attribute RG-290
node_sort_key RG-253
nodect RG-270
nodes RG-270
non-consumable resource RG-12
non-job event hooks RG-12
non-primetime RG-15
nonprimetime_prefix RG-254
normal_jobs RG-300
now reservation RG-10, RG-13
nppcu RG-13
ntype
vnode attribute RG-322

O

object RG-13
occurrence of a standing reservation RG-13
offline RG-365
OMP_NUM_THREADS RG-399
ompthreads RG-271
only_explicit_psets
scheduler attribute RG-298
Operator RG-13
operators
server attribute RG-291
opt_backfill_fuzzy
scheduler attribute RG-299
order
hook attribute RG-352
Output_Path
job attribute RG-341
overall limit RG-13
owner RG-13

P

parameter RG-13
parent vnode RG-13
partition RG-316, RG-322
scheduler attribute RG-299
pathname
format RG-357
PBS RG-400
pbs RG-28
PBS Administrator RG-14
PBS entity RG-6, RG-14
pbs module RG-14
PBS NAME
format RG-357
PBS object RG-13, RG-14
PBS password
format RG-357
PBS Professional RG-14
PBS_ARRAY_ID RG-399
PBS_ARRAY_INDEX RG-399
pbs_attach RG-55
PBS_AUTH_METHOD RG-369
PBS_BATCH_SERVICE_PORT RG-369
PBS_BATCH_SERVICE_PORT_DIS RG-369
pbs_comm RG-4, RG-57
PBS_COMM_LOG_EVENTS RG-369
PBS_COMM_ROUTERS RG-369
PBS_COMM_THREADS RG-369
PBS_CONF_FILE RG-399
PBS_CONF_SYSLOG RG-373
PBS_CONF_SYSLOGSEVR RG-373
PBS_CORE_LIMIT RG-370
PBS_DATA_SERVICE_PORT RG-370
pbs_dataservice RG-60
pbs_ds_password RG-61
PBS_ENCRYPT_METHOD RG-370
PBS_ENVIRONMENT RG-370, RG-399
PBS_EXEC RG-14, RG-370
PBS_HOME RG-14, RG-370
pbs_hostn RG-63
pbs_idled RG-64
pbs_interactive RG-67
PBS_JOBCOOKIE RG-399
PBS_JOBID RG-399
PBS_JOBNAME RG-399
pbs_lamboot RG-68
PBS_LEAF_NAME RG-370
PBS_LEAF_ROUTERS RG-370
pbs_license_info
server attribute RG-291
pbs_license_linger_time
server attribute RG-291
pbs_license_max
server attribute RG-291
pbs_license_min
server attribute RG-292
PBS_LOCALLOG RG-370
PBS_LOG_HIGHRES_TIMESTAMP RG-371, RG-400

PBS_LR_SAVE_PATH RG-371
PBS_MAIL_HOST_NAME RG-371
PBS_MANAGER_SERVICE_PORT RG-371
pbs_mkdirs RG-70
pbs_mom RG-71
PBS_MOM_HOME RG-371
PBS_MOM_NODE_NAME RG-371
PBS_MOM_SERVICE_PORT RG-371
PBS_MOMPORT RG-400
pbs_mpihp RG-76
pbs_mpilam RG-78
pbs_mpirun RG-79
PBS_NODENUM RG-400
PBS_O_HOME RG-400
PBS_O_HOST RG-400
PBS_O_LANG RG-400
PBS_O_LOGNAME RG-400
PBS_O_MAIL RG-400
PBS_O_PATH RG-400
PBS_O_QUEUE RG-400
PBS_O_SHELL RG-400
PBS_O_SYSTEM RG-400
PBS_O_TZ RG-400
PBS_O_WORKDIR RG-400
PBS_OUTPUT_HOST_NAME RG-371
PBS_PRIMARY RG-372
pbs_probe RG-81
pbs_python RG-83
PBS_QUEUE RG-400
PBS_RCP RG-372
pbs_rdel RG-90
PBS_REMOTE_VIEWER RG-370
pbs_rstat RG-94
pbs_rsub RG-96
pbs_sched RG-105
PBS_SCHED_THREADS RG-372
PBS_SCHEDULER_SERVICE_PORT RG-372
PBS_SCP RG-372
PBS_SECONDARY RG-372
PBS_SERVER RG-372, RG-400
pbs_server RG-108
PBS_SERVER_HOST_NAME RG-372
PBS_SID RG-400
PBS_START_COMM RG-372
PBS_START_MOM RG-372
PBS_START_SCHED RG-372
PBS_START_SERVER RG-372
PBS_SUPPORTED_AUTH_METHODS RG-373
PBS_TASKNUM RG-401
pbs_tclsh RG-119
PBS_TMPDIR RG-373, RG-401
pbs_tmrsh RG-120
pbs_version
  scheduler attribute RG-299
  server attribute RG-292
  vnode attribute RG-322
pbs_wish RG-122, RG-124
pbsadmin RG-14
PBScrayhost RG-271
PBScraylabel_ RG-271
PBScraynid RG-271
PBScrayorder RG-272
PBScrayseg RG-272
pbsdsh RG-29
pbsfs RG-31
pbshook RG-14
pbsnodes RG-35
pbsrun RG-40
pbsrun_unwrap RG-50
pbsrun_wrap RG-51
pcap_accelerator RG-341
pcap_node RG-341
pcpus
  vnode attribute RG-322
pcput RG-272
peer scheduling RG-14
pgov RG-341
p-governor RG-341
placement pool RG-15
placement set RG-14
placement set series RG-14
pmem RG-272
pnames
  vnode attribute RG-322
policy RG-15
  scheduling RG-17
Port
  vnode attribute RG-322
POSIX RG-15
power_provisioning
  server attribute RG-292
  vnode attribute RG-322
poweroff_eligible
  vnode attribute RG-322
preempt RG-15
preempt_order RG-254
preempt_prio RG-254
preempt_queue_prio RG-254
preempt_sort RG-254
preempt_targets RG-273
preemption
  level RG-15
  method RG-15
  target RG-15
preemptive_sched RG-254
pre-execution event hooks RG-15
primary execution host RG-15
primary scheduler RG-15

primary server RG-15, RG-372
prime_spill RG-255
primetime RG-15
primetime_prefix RG-254
printjob RG-125
Priority
    job attribute RG-342
    queue attribute RG-316
    vnode attribute RG-323
project RG-15
    job attribute RG-342
project limit RG-15
project name
    format RG-357
provision RG-16
provision_enable
    vnode attribute RG-323
provision_policy RG-255
provisioned vnode RG-16
provisioning RG-366
    hook RG-16
provisioning tool RG-16
pset
    job attribute RG-342
pstate RG-342
pulling queue RG-16
pvmem RG-273
python_restart_max_hooks
    server attribute RG-292
python_restart_max_objects
    server attribute RG-292
python_restart_min_interval
    server attribute RG-292

Q

qalter RG-127
qdel RG-140
qdisable RG-143
qenable RG-145
qhold RG-147
qmgr RG-149, RG-380
qmove RG-172
qmsg RG-174
qorder RG-176
qrerun RG-178
qrls RG-180
qrun RG-182
qselect RG-186
qsig RG-192
qstart RG-195
qstat RG-197
qstop RG-211
qsub RG-213
qterm RG-233
qtime
    job attribute RG-342
query_other_jobs
    server attribute RG-292
queue
    access to a RG-1
    definition RG-16
    execution RG-6
    furnishing RG-7
    job attribute RG-342
    pulling RG-16
    reservation attribute RG-306
    routing RG-17
    vnode attribute RG-323
queue identifier
    format RG-357
queue name
    format RG-357
queue_rank
    job attribute RG-342
queue_softlimits RG-300
queue_type
    job attribute RG-343
    queue attribute RG-317
queued_jobs_threshold
    queue attribute RG-316
queued_jobs_threshold_res
    queue attribute RG-317
    server attribute RG-293
queuing RG-16

R

rcp RG-372
redundant license server configuration RG-16
reject an action RG-16
release_nodes_on_stageout RG-343
requeue RG-16
require_cred
    queue attribute RG-317
require_cred_enable
    queue attribute RG-317
Rerunable
    job attribute RG-343
reservation
    access to a RG-1
    advance RG-2
    ASAP RG-2, RG-10
    degradation RG-16
    degraded RG-5
    instance RG-13
    job-specific RG-10
        ASAP RG-2, RG-10

now RG-10, RG-13
start RG-10, RG-19
now RG-10, RG-13
occurrence RG-13
soonest occurrence RG-18
standing RG-19
instance RG-13
soonest occurrence RG-18
start RG-10
reservation degradation RG-16
reservation ID RG-16
reservation identifier RG-16
reservation name
format RG-358
reserve_count
reservation attribute RG-306
reserve_duration
reservation attribute RG-306
reserve_end
reservation attribute RG-306
reserve_ID
reservation attribute RG-306
reserve_index
reservation attribute RG-307
reserve_job RG-307
Reserve_Name
reservation attribute RG-307
Reserve_Owner
reservation attribute RG-307
reserve_retry
reservation attribute RG-307
reserve_retry_cutoff
server attribute RG-293
reserve_retry_init
server attribute RG-293
reserve_retry_time
server attribute RG-293
reserve_rrule
reservation attribute RG-308
reserve_start
reservation attribute RG-308
reserve_state
reservation attribute RG-309
reserve_substate
reservation attribute RG-309
resource RG-17
built-in RG-3
consumable RG-4
custom RG-4
indirect RG-9
job-wide RG-11
non-consumable RG-12
shared RG-18
Resource_List
job attribute RG-344
reservation attribute RG-310
Resource_List.eoe RG-267
resource_unset_infinite RG-256
resources RG-256
resources_assigned
queue attribute RG-317
server attribute RG-294
vnode attribute RG-323
resources_available
queue attribute RG-318
server attribute RG-294
vnode attribute RG-323
resources_available.eoe RG-267
resources_default
queue attribute RG-318
server attribute RG-294
resources_max
queue attribute RG-318
server attribute RG-295
resources_min
queue attribute RG-318
resources_released RG-344
resources_released_list RG-344
resources_used
job attribute RG-344
restart RG-17, RG-240
restart file RG-17
restart script RG-17
restrict_res_to_release_on_suspend RG-295
resv
vnode attribute RG-324
RESV_BEING_DELETED RG-367
RESV_CONFIRMED RG-367
RESV_DEGRADED RG-367
RESV_DELETED RG-367
RESV_DELETING_JOBS RG-367
resv_enable
vnode attribute RG-324
RESV_FINISHED RG-367
RESV_IN_CONFLICT RG-367
resv_nodes
reservation attribute RG-310
RESV_NONE RG-367
resv_post_processing_time
server attribute RG-295
RESV_RUNNING RG-367
RESV_TIME_TO_RUN RG-367
RESV_UNCONFIRMED RG-367
RESV_WAIT RG-367
resv-exclusive RG-366
round_robin RG-256
route RG-17
route queue RG-379, RG-382

route_destinations
queue attribute RG-319
route_held_jobs
queue attribute RG-319
route_lifetime
queue attribute RG-319
route_retry_time
queue attribute RG-319
route_waiting_jobs
queue attribute RG-319
routing queue RG-17
rpp_highwater
server attribute RG-295
rpp_max_pkt_check RG-295
rpp_retry
server attribute RG-295
run_count RG-137, RG-228
job attribute RG-345
run_version
job attribute RG-345

S

sandbox RG-228
job attribute RG-345
sched_cycle_length
scheduler attribute RG-301
sched_host
scheduler attribute RG-301
sched_log
scheduler attribute RG-301
sched_port
scheduler attribute RG-301
sched_preempt_enforce_resumption
scheduler attribute RG-301
sched_priv
scheduler attribute RG-301
schedselect
job attribute RG-345
scheduler RG-17
scheduler_iteration
scheduler attribute RG-300
server attribute RG-296
Scheduling
server state RG-364
scheduling
policy RG-15, RG-17
scheduler attribute RG-300
server attribute RG-296
scheduling jobs RG-17
Schema Admins RG-17
scp RG-372
secondary scheduler RG-17
secondary server RG-17, RG-372
sequence number RG-17
server RG-18
access to the RG-1
default RG-5
job attribute RG-346
name RG-18
primary RG-372
reservation attribute RG-310
secondary RG-372
server_dyn_res RG-256
server_dyn_res_alarm RG-301
server_softlimits RG-300
server_state
server attribute RG-297
session_id
job attribute RG-346
set_power_cap RG-341
shared resource RG-18
sharing
vnode attribute RG-324
Shell_Path_List
job attribute RG-346
shrink-to-fit job RG-18
single_signon_password_enable
server attribute RG-297
sister RG-18
sisterhood RG-18
site RG-273
definition RG-18
size
format RG-260, RG-360
smp_cluster_dist RG-256
snapshot checkpoint RG-18
soft_walltime RG-274
software RG-273
soonest occurrence RG-18
sort_priority RG-252
sort_queues RG-257
stage
in RG-18
out RG-18
stagein
job attribute RG-346
stageout
job attribute RG-346
Stageout_status
job attribute RG-347
staging and execution directory RG-19
stale RG-366
standing reservation RG-19
start reservation RG-10, RG-19
start_time RG-274
started
queue attribute RG-319

starving_jobs RG-251, RG-300
state RG-19
    scheduler attribute RG-301
    server
        Hot_Start RG-364
        Idle RG-364
        Scheduling RG-364
        Terminating RG-364
        Terminating_Delayed RG-364
    vnode attribute RG-326
state_count
    queue attribute RG-319
    server attribute RG-297
state-unknown, down RG-366
stime
    job attribute RG-347
strict ordering RG-19
strict_fifo RG-257
strict_ordering RG-257
string resource value
    format RG-260, RG-360
string_array
    format RG-260, RG-360
subject RG-19
subjob RG-19
subjob identifier
    format RG-358
subjob index RG-19
Submit_arguments
    job attribute RG-347
subordinate MoM RG-19
substate
    job attribute RG-347
sw_index
    job attribute RG-347
sync_time RG-257

T

task RG-19
task placement RG-19
terminate RG-240
Terminating
    server state RG-364
Terminating_Delayed
    server state RG-364
three-server configuration RG-19
throughput_mode
    scheduler attribute RG-302
time-sharing RG-379, RG-380
TMPDIR RG-401
tolerate_node_failures RG-348
topjob_ineligible
    job attribute RG-348
topology_info
    vnode attribute RG-326
total_jobs
    queue attribute RG-319
    server attribute RG-297
TPP RG-20
tracejob RG-235
type
    hook attribute RG-352

U

UID RG-20
umask
    job attribute RG-348
unknown_shares RG-257
user
    access RG-20
    definition RG-20
    hook attribute RG-352
    ID RG-20
user limit RG-20
User_List
    job attribute RG-349
username
    format RG-358
    Windows
        format RG-358

V

Variable_List
    job attribute RG-349
vchunk RG-20
version 1 configuration file RG-20
version 2 configuration file RG-20
vmem RG-274
vnode RG-20, RG-274
    borrowing RG-3
    managing RG-12
    memory-only RG-12
vnode name
    format RG-358
vnode_pool RG-327
vnodedefs file RG-20
vntype RG-275
vp RG-20

W

wait-provisioning RG-366
walltime RG-275
Windows
    mixed-mode complex RG-12
Windows-Linux complex RG-20



Altair PBS Professional 2021.1

User's Guide

You are reading the Altair PBS Professional 2021.1

User's Guide (UG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation ix

1 Getting Started with PBS 1

1.1 Why Use PBS? . . . 1
1.2 PBS Tasks and Components . . . 1
1.3 Interfaces to PBS . . . 3
1.4 Setting Up Your Environment . . . 5

2 Submitting a PBS Job 11

2.1 Introduction to the PBS Job . . . 11
2.2 The PBS Job Script . . . 14
2.3 Submitting a PBS Job . . . 18
2.4 Job Submission Recommendations and Advice . . . 23
2.5 Job Submission Options . . . 24
2.6 Job Submission Caveats . . . 30

3 Job Input & Output Files 31

3.1 Introduction to Job File I/O in PBS . . . 31
3.2 Input/Output File Staging . . . 31
3.3 Managing Output and Error Files . . . 39

4 Allocating Resources & Placing Jobs 49

4.1 What is a Vnode? . . . 49
4.2 PBS Resources . . . 49
4.3 Requesting Resources . . . 51
4.4 How Resources are Allocated to Jobs . . . 59
4.5 Limits on Resource Usage . . . 61
4.6 Viewing Resources . . . 63
4.7 Specifying Job Placement . . . 64
4.8 Backward Compatibility . . . 70

5 Multiprocessor Jobs 77

5.1 Submitting Multiprocessor Jobs . . . 77
5.2 Using MPI with PBS . . . 81
5.3 Using PVM with PBS . . . 102
5.4 Using OpenMP with PBS . . . 103
5.5 Hybrid MPI-OpenMP Jobs . . . 105

6 Controlling How Your Job Runs 107

6.1 Using Job Exit Status . . . 107
6.2 Using Job Dependencies . . . 107
6.3 Adjusting Job Running Time . . . 110
6.4 Using Checkpointing . . . 113
6.5 Holding and Releasing Jobs . . . 115
6.6 Allowing Your Job to be Re-run . . . 118
6.7 Controlling Number of Times Job is Re-run . . . 119
6.8 Deferring Execution . . . 119
6.9 Setting Priority for Your Job . . . 120
6.10 Making `qsub` Wait Until Job Ends . . . 120
6.11 Running Your Job Interactively . . . 121
6.12 Using Environment Variables . . . 126
6.13 Specifying Which Jobs to Preempt . . . 127
6.14 Releasing Unneeded Vnodes from Your Job . . . 128
6.15 Running Your Job in a Container . . . 130
6.16 Running Your Job in the Cloud . . . 133
6.17 Allowing Your Job to Tolerate Vnode Failures . . . 134

7 Reserving Resources 135

7.1 Glossary . . . 135
7.2 Quick Explanation of Reservations for Jobs . . . 136
7.3 Prerequisites for Reserving Resources . . . 136
7.4 Advance and Standing Reservations . . . 136
7.5 Job-specific Reservations . . . 140
7.6 Getting Confirmation of a Reservation . . . 142
7.7 Modifying Reservations . . . 142
7.8 Deleting Reservations . . . 143
7.9 Viewing the Status of a Reservation . . . 143
7.10 Submitting a Job to a Reservation . . . 146
7.11 Reservation Caveats and Errors . . . 147

8 Job Arrays 149

8.1 Advantages of Job Arrays . . . 149
8.2 Glossary . . . 149
8.3 Description of Job Arrays . . . 149
8.4 Submitting a Job Array . . . 152
8.5 Viewing Status of a Job Array . . . 157
8.6 Using PBS Commands with Job Arrays . . . 160
8.7 Job Array Caveats . . . 162

9 Working with PBS Jobs 163

9.1 Using Job History . . . 163
9.2 Modifying Job Attributes . . . 164
9.3 Deleting Jobs . . . 166
9.4 Sending Messages to Jobs . . . 167
9.5 Sending Signals to Jobs . . . 168
9.6 Changing Order of Jobs . . . 168
9.7 Moving Jobs Between Queues . . . 169

10 Checking Job & System Status 171
10.1 Checking Job Status . . . 171
10.2 Checking Server Status. . . . 183
10.3 Checking Queue Status . . . 184
10.4 Full Display Options for Job, Queue, and Server Status. . . . 186
10.5 Selecting a List of Jobs . . . 187
10.6 Checking License Availability . . . 189

11 Submitting Cray Jobs 191
11.1 PBS Jobs on Cray Shasta. . . . 191
11.2 PBS Jobs on the Cray XC. . . . 191
11.3 Resources for Cray XC . . . 191
11.4 Rules for Submitting Jobs on the Cray XC . . . 192
11.5 Techniques for Submitting Cray XC Jobs . . . 193
11.6 Using Xeon Phi Vnodes on Cray XC. . . . 195
11.7 Using Hyperthreads on Cray XC. . . . 195
11.8 Viewing Cray XC Job Information . . . 196
11.9 Caveats and Advice for Cray XC. . . . 197
11.10 Errors and Logging on Cray XC . . . 199

12 Submitting Jobs to NEC SX-Aurora TSUBASA 201
12.1 Vnodes for NEC SX-Aurora TSUBASA. . . . 201
12.2 Terminology. . . . 201
12.3 Resources for SX-Aurora TSUBASA . . . 202
12.4 Running Your Job on NEC SX-Aurora TSUBASA . . . 202
12.5 Job Accounting on NEC SX-Aurora TSUBASA. . . . 209
12.6 Environment Variables for NEC MPI. . . . 209

13 Using MLS with PBS Professional 211
13.1 About SELinux PBS Professional . . . 211
13.2 Requirement for Submitting Jobs . . . 211
13.3 Viewing and Operating on Jobs . . . 211
13.4 Credentials of Deleted Jobs . . . 211
13.5 Caveats . . . 212
13.6 Errors and Logging . . . 212
13.7 SELinux Documentation . . . 213

14 Using Provisioning 215
14.1 Definitions . . . 215
14.2 How Provisioning Works . . . 215
14.3 Requirements and Restrictions . . . 216
14.4 Using Provisioning. . . . 218
14.5 Caveats and Errors . . . 219

15 Using Accounting 221
15.1 Using Accounting . . . 221

Index 223

1 Getting Started with PBS

1.1 Why Use PBS?

PBS frees you from the mechanics of getting your work done; you don't need to shepherd each job to the right machine, get input and output copied back and forth, or wait until a particular machine is available. You need only specify requirements for the tasks you want executed, and hand the tasks off to PBS. PBS holds each task until a slot opens up, then takes care of copying input files to the execution directory, executing the task, and returning the output to you.

PBS keeps track of which hardware is available, and all waiting and running tasks. PBS matches the requirements of each of your tasks to the right hardware and time slot, and makes sure that tasks are run according to the site's policy. PBS also maximizes usage and throughput.

1.2 PBS Tasks and Components

1.2.1 PBS Tasks

PBS is a distributed workload management system. PBS manages and monitors the computational workload for one or more computers. PBS does the following:

Queuing jobs

PBS collects jobs (work or tasks) to be run on one or more computers. Users submit jobs to PBS, where they are queued up until PBS is ready to run them.

Scheduling jobs

PBS selects which jobs to run, and when and where to run them, according to the resources requested by the job, and the policy specified by the site administrator. PBS allows the administrator to prioritize jobs and allocate resources in a wide variety of ways, to maximize efficiency and/or throughput.

Monitoring jobs

PBS tracks system resources, enforces usage policy, and reports usage. PBS tracks job completion, ensuring that jobs run despite system outages.

1.2.2 PBS Components

PBS consists of a set of commands and system daemons/services, shown here:

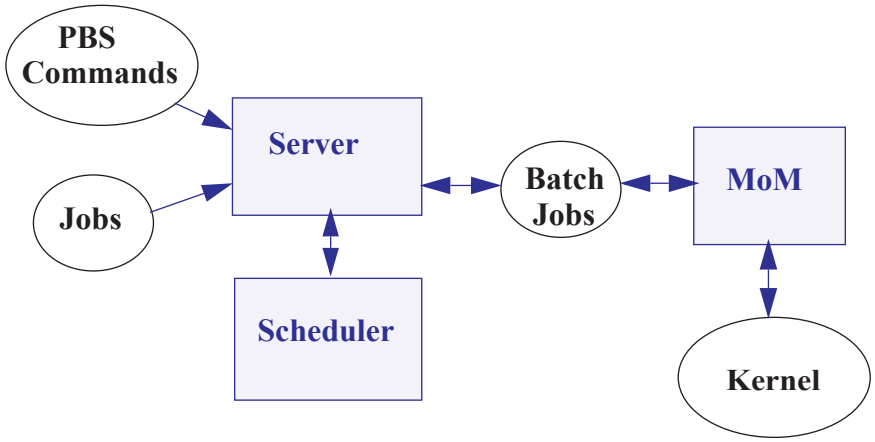


Figure 1-1: Jobs are submitted to the PBS server. The scheduler chooses where and when to run the jobs, and the server sends the jobs to MoM. PBS commands communicate with the server.

The server, scheduler, and communication daemons run on the server host. A machine that executes jobs is called an execution host. Each execution host runs a MoM daemon. The server host can run a MoM daemon. One server manages any number of MoM daemons. Commands can be run from the server host, execution hosts, and command-only client hosts. The server/scheduler/communication host, the execution hosts, and the client hosts are called a PBS complex.

Commands

PBS provides a set of commands that you can use to submit, monitor, alter, and delete jobs. The PBS commands can be installed on any supported platform, with or without the other PBS components.

Some PBS commands can be run by any PBS user, while some require administrator or operator privilege. Some commands provide extended features for administrators and operators.

Job

A PBS job is a task, in the form of a shell script, `cmd` batch file, Python script, etc. describing the commands and/or applications you want to run. You hand your task off to PBS, where it becomes a PBS job.

Server

The PBS server manages jobs for the PBS complex. PBS commands talk to the PBS server, jobs are submitted to the server, and the server queues the jobs and sends them to execution hosts.

Scheduler

The scheduler runs jobs according to the policy specified by the site administrator. The scheduler matches each job's requirements with available resources, and prioritizes jobs and allocates resources according to policy.

MoM

MoM manages jobs once they are sent to the execution host. One MoM manages the jobs on each execution host. MoM stages files in, runs any prologue, starts each job, monitors the job, stages files out and returns output to the job submitter, runs any epilogue, and cleans up after the job. MoM can also run any execution host hooks.

MoM creates a new session that is as identical to your login session as is possible. For example, under Linux, if the job submitter's login shell is `csh`, then MoM creates a session in which `.login` is run as well as `.cshrc`.

MoM is a reverse-engineered acronym that stands for Machine-oriented Mini-server.

1.3 Interfaces to PBS

PBS provides a command-line interface, and Altair offers a web-based front end to PBS called Access, which is a separate product. This document describes the PBS command-line interface. For information on Access, see www.pbsworks.com.

1.3.1 PBS Commands

PBS provides a set of commands that allow you to submit, monitor, and manage your jobs. Some PBS commands can be used by any PBS user; some can be used only by administrators, and some have different behavior depending on the role of the person invoking them. In this document, we describe the commands that can be used by any PBS user. For a complete description of all commands and their requirements, see "Requirements for Commands" on page 21 of the PBS Professional Reference Guide.

Table 1-1: PBS Professional User Commands

PBS User Commands	
Command	**Purpose**
pbs_login	Cache your password
pbs_rdel	Delete a reservation
pbs_rstat	Status a reservation
pbs_python	Python interpreter
pbs_rsub	Submit a reservation
pbsdsh	PBS distributed shell
qalter	Alter job
qdel	Delete job
qhold	Hold a job
qmove	Move job
qmsg	Send message to job
qorder	Reorder jobs
qrls	Release hold on job
qselect	Select jobs by criteria
qsig	Send signal to job
qstat	Status job, queue, server
qsub	Submit a job
tracejob	Report job history

We also list the PBS administrator commands here:

Table 1-2: PBS Administrator Commands

PBS Administrator Commands	
Command	**Purpose**
`pbs-report`	Report job statistics
`pbs_hostn`	Report host name(s)
`pbs_probe`	PBS diagnostic tool
`pbs_tclsh`	TCL with PBS API
`pbsfs`	Show fairshare usage
`pbsnodes`	Manage vnodes
`printjob`	Report job details
`qdisable`	Disable a queue
`qenable`	Enable a queue
`qmgr`	Manager interface
`qrerun`	Requeue running job
`qrun`	Manually start a job
`qstart`	Start a queue
`qstop`	Stop a queue
`qterm`	Shutdown PBS

1.4 Setting Up Your Environment

1.4.1 Prerequisites for Account

Your account must have the following characteristics for PBS to work correctly:

- Account must have access to all PBS hosts
- Account must have valid username and group on all execution hosts and on the server
- Account must be able to transfer files between hosts using the file transfer mechanism chosen by the administrator. This is described in section 15.6, "Setting File Transfer Mechanism", on page 559 of the PBS Professional Administrator's Guide.
- The time zone environment variable must be set correctly in order to use advance and standing reservations. See section 1.4.4, "Setting Time Zone for Submission Host", on page 9.
- Username must be 256 characters or less in length.
- Your environment must be correctly configured:
  - For Linux, see section 1.4.2, "Setting Up Your Linux Environment", on page 5.
  - For Windows, see section 1.4.3, "Setting Up Your Windows Environment", on page 7.
- Account must have correct user authorization to run jobs.
  - For Linux, see section 1.4.2.7, "User Authorization Under Linux", on page 7.
  - For Windows, see section 1.4.3.4, "User Authorization under Windows", on page 8

1.4.2 Setting Up Your Linux Environment

1.4.2.1 Set Paths to PBS Commands

PBS commands reside in a directory pointed to by `$PBS_EXEC/bin`. This path may change from one installation of PBS to the next, so use the variable instead of the absolute path. The location of `$PBS_EXEC` is given in `/etc/pbs.conf`. Make it easy to use PBS commands by doing the following:

1. In your `.login` file, source `/etc/pbs.conf`:

   If you are using `bash` or `sh`, do the following:

   ```
   % . /etc/pbs.conf
   ```

   If you are using `csh`, do the following:

   ```
   %source /etc/pbs.conf
   ```

2. Add the path to PBS commands to your PATH environment variable. Use `$PBS_EXEC`, not the absolute path. For example, where `MY_PATH` is your existing set of paths:

   ```
   setenv PATH ${MY_PATH}:$PBS_EXEC/bin/
   ```

1.4.2.2 Set Paths to PBS Man Pages

Add the path to the PBS man pages to your MANPATH environment variable:

```
setenv MANPATH /usr/man:/usr/local/man:$PBS_EXEC/share/man/
```

1.4.2.3 Make Login and Logout Files Behave Properly for Jobs

By default, PBS runs your jobs under your login, meaning that your login and logout files are sourced for each job. If your `.cshrc`, `.login`, `.profile`, or `.logout` contains commands that attempt to set terminal characteristics or produce output, such as by writing to `stdout`, jobs may not run. Make sure that any such command in these files is skipped when the file is run inside a PBS job. PBS sets the PBS_ENVIRONMENT environment variable inside jobs. Test for the PBS_ENVIRONMENT environment variable and run commands only when it is not set. For example, in a `.login` file:

```
if ( ! $?PBS_ENVIRONMENT ) then
    do terminal settings here
    run command with output here
endif
```

1.4.2.4 Capture Correct Job Exit Status

When a PBS job runs, the exit status of the last command executed in the job is reported by the job's shell to PBS as the exit status of the job. The exit status of the job is important for job dependencies and job chaining. Under Linux, the last command executed might not be the last command in your job, if you have a `.logout` on the execution host. In that case, the last command executed is from the `.logout` and not from your job. To prevent this, preserve the job's exit status in your `.logout` file by saving it at the top, then doing an explicit `exit` at the end, as shown below:

```
set EXITVAL = $status
previous contents of .logout here
exit $EXITVAL
```

Under Windows, you do not need to take special steps to preserve the job's exit status.

1.4.2.5 Avoid Background Processes Inside Jobs

Make sure that your login file doesn't run processes in the background when invoked inside a PBS job. If your login file contains a command that runs in the background inside a PBS job, persistent processes can cause trouble.

1.4.2.6 Provide `bash` Functions to Jobs

If your jobs need to have exported bash functions available to them, you can put these functions in your `.profile` or `.login` on the execution host(s). You can also use `qsub -V` or `qsub -v <function name>` to forward the function at job submission. Just make sure that you don't have a function with the same name as an environment variable if you use `-v` or `-V`. See section 6.12.4, “Forwarding Exported Shell Functions”, on page 127.

1.4.2.7 User Authorization Under Linux

The server's flatuid attribute determines whether it assumes that identical user names mean identical users. If *True*, it assumes that if UserS exists on both the submission host and the server host, then UserS can run jobs on that server. If not *True*, the server calls `ruserok()` which uses `/etc/hosts.equiv` or `.rhosts` to authorize UserS to run as UserS. In this case, the username you specify with the `-u` option must have a `.rhosts` file on the server host listing the job owner, meaning that UserS at the server must have a `.rhosts` file listing UserS.

Example 1-1: Our user is UserA on the submission host, but is userB at the server. In order to submit jobs as UserA and run jobs as UserB, UserB must have a `.rhosts` file on the server host that lists UserA.

Table 1-3: Linux User ID and flatuid

Value of flatuid	Submission Host Username vs. Server Host Username	
	UserS Same as UserS	UserS Different from UserA
True	Server assumes user has permission to run job	Server checks whether UserS can run job as UserA
False/*unset*	Server checks whether UserS can run job as UserS	Server checks whether UserS can run job as UserA

Note that if different names are listed via the `-u` option, then they are checked regardless of the value of `flatuid`.

Using `hosts.equiv` is not recommended.

1.4.2.8 Submitting Linux Jobs from Linux Clients

If the authentication method at a Linux client host has been set to pwd, set it to munge before you submit a Linux job. For example:

```
export PBS_AUTH_METHOD=munge; qsub -lselect=1:arch=linux -- sleep 100
```

1.4.3 Setting Up Your Windows Environment

1.4.3.1 HOMEDIR for Windows Users

PBS starts jobs in the job owner's home directory, which is pointed to by `HOMEDIR`.

If you have not been explicitly assigned a home directory, PBS uses a Windows-assigned default as the base location for your default home directory, and starts jobs there. Windows assigns the following default home path:

```
[PROFILE_PATH]\My Documents\PBS Pro
```

For example, if userA has not been assigned a home directory, the default home directory is the following:

```
\Documents and Settings\userA\My Documents\PBS Pro
```

Windows can return one `PROFILE_PATH` in one of the following forms:

```
\Documents and Settings\username
\Documents and Settings\username.local-hostname
\Documents and Settings\username.local-hostname.00N
```

where *N* is a number

```
\Documents and Settings\username.domain-name
```

1.4.3.2 Requirements for Windows Username

- The username must contain only alphanumeric characters, dot (.), underscore (_), and/or hyphen "-".
- The hyphen must not be the first letter of the username.
- If "@" appears in the username, then it is assumed to be in the context of a Windows domain account: `username@domainname`.
- The space character is allowed. If a space character appears in a username string, then the string is displayed in quotes, and must be specified in quotes.

1.4.3.3 Requirements for Windows User Account

Your Windows user account must be a normal user account. You cannot submit jobs from a SYSTEM account.

1.4.3.4 User Authorization under Windows

PBS runs your jobs under your account. When your job runs on a remote execution host, it needs to be able to log in and transfer files using your account. If your system administrator has not set up access using `hosts.equiv`, you can set up access using `.rhosts` files. A `.rhosts` file on the server allows you to submit jobs from a remote machine to the server.

Set up the `.rhosts` file in your `PROFILE_PATH`, in your home directory, on the PBS server host and on each execution host. For example:

```
\Documents and Settings\username\.rhosts
```

Format of `.rhosts` file:

hostname username

Be sure the `.rhosts` file is owned by you or an administrator-type group, and has write access granted only to you or an administrator or group.

Add all PBS hosts to your `.rhosts` file:

```
Host1 user1
Host2 user1
Host3 user1
```

Make sure that you list all the names by which a host may be known. For instance, if Host4 is known as "Host4", "Host4.<subdomain>", or "Host4.<subdomain>.<domain>" you should list all three in the `.rhosts` file:

```
Host4 user1
Host4.subdomain user1
Host4.subdomain.domain user1
```

If your username contains white space, quote it in the `.rhosts` file:

```
Host4.subdomain.domain "Bob Jones"
```

Example 1-2: The following entry in user user1's `.rhosts` file on the server permits user user1 to run jobs submitted from the workstation wks031:

```
wks031 user1
```

To allow user1's output files from a job that runs on execution host Host1 to be returned to user1 automatically by PBS, user1 adds an entry to the `.rhosts` file on the workstation naming the execution host Host1:

```
Host1 user1
```

1.4.3.5 Set up Paths

If you will use a mapped drive for submitting jobs, staging files in and out, or for output and error files, you must map that drive with a local system account. We recommend using UNC paths. If you do not use a local system account, file transfer behavior is undefined. To map a drive with global access using a local system account, use the psExec utility from SysInternals:

<path to psexec binary> -s net use <mapped drive letter>: <UNC path to map>

For example:

```
psexec -s net use Z: \\examplehost\mapping_directory\mydirectory
```

To unmap a mapped drive:

<path to psexec binary> -s net use /delete <mapped drive letter>

For example:

```
psexec -s net use /delete Z:
```

PBS requires that your username be consistent across a server and its execution hosts, but not across a submission host and a server. You may have access to more than one server, and may have a different username on each server. You can change the user ID for a job; see section 2.5.4, “Specifying Job Username”, on page 28.

1.4.3.6 Password for Job Submission Authentication

Run the `pbs_login` command whenever your password changes. The new password is used for any job that is not already running.

1.4.3.6.i Setting Password at Windows Clients

Run the pbs_login command once for each Windows submission host, so that you can submit jobs and run PBS client commands.

```
echo <password>| pbs_login -p
```

Test whether you can run client commands:

```
qstat -Bf
```

The new password is used for any job that is not already running.

1.4.3.6.ii Setting Password at Linux Clients

Run the pbs_login command at any Linux client host where you want to submit a Windows job. Set PBS_AUTH_METHOD to *pwd*:

```
export PBS_AUTH_METHOD=pwd; pbs_login
```

1.4.3.7 Authentication for Client Commands

You can run all client commands except `qsub` using either pwd or munge as the authentication method, so you don't need to make any changes for commands such as `qstat`, etc.

1.4.4 Setting Time Zone for Submission Host

Make sure that the environment variable PBS_TZID is set correctly at your submission host. Set this environment variable to a timezone location known to PBS Professional. You can get the appropriate zone location from the PBS server host.

On Linux, use the `tzselect` command if it is available, or get the zone location from `/usr/share/zoneinfo/zone.tab`.

On all other platforms, use the list of `libical` supported zoneinfo locations available under `$PBS_EXEC/lib/ical/zoneinfo/zones.tab`.

The format for PBS_TZID is a timezone location, rather than a timezone POSIX abbreviation. Examples of values for PBS_TZID are:

```
America/Los_Angeles
America/Detroit
Europe/Berlin
Asia/Calcutta
```

2
Submitting a PBS Job

2.1 Introduction to the PBS Job

To use PBS, you create a *batch job*, usually just called a *job,* which you then hand off, or *submit*, to PBS. A batch job is a set of commands and/or applications you want to run on one or more execution machines, contained in a file or typed at the command line. You can include instructions which specify the characteristics such as job name, and resource requirements such as memory, CPU time, etc., that your job needs. The job file can be a shell script under Linux, a `cmd` batch file under Windows, a Python script, a Perl script, etc.

For example, here is a simple PBS batch job file which requests one hour of time, 400MB of memory, 4 CPUs, and runs `my_application`:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l mem=400mb,ncpus=4
./my_application
```

To submit the job to PBS, you use the `qsub` command, and give the job script as an argument to `qsub`. For example, to submit the script named "`my_script`":

```
qsub my_script
```

We will go into the details of job script creation in section 2.2, “The PBS Job Script”, on page 14, and job submission in section 2.3, “Submitting a PBS Job”, on page 18.

2.1.1 Lifecycle of a PBS Job, Briefly

Your PBS job has the following lifecycle:

1. You write a job script
2. You submit the job to PBS
3. PBS accepts the job and returns a job ID to you
4. The PBS scheduler finds the right place and time to run your job, and sends your job to the selected execution host(s)
5. Application licenses are checked out
6. On each execution host, PBS creates a job-specific staging and execution directory
7. PBS sets PBS_JOBDIR and the job's jobdir attribute to the path of the job's staging and execution directory.
8. On each execution host allocated to the job, PBS creates a job-specific temporary directory.
9. PBS sets the TMPDIR environment variable to the pathname of the temporary directory.
10. If any errors occur during directory creation or the setting of variables, the job is requeued.
11. Input files or directories are copied to the primary execution host
12. If needed, cpusets are created

- If it exists, the prologue runs on the primary execution host, with its current working directory set to `PBS_HOME/mom_priv`, and with PBS_JOBDIR and TMPDIR set in its environment.

13. The job runs under your login
14. If it exists, the epilogue runs on the primary execution host, with its current working directory set to the path of the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in its environment.
15. Output files or directories are copied to specified locations
16. Temporary files and directories are cleaned up
17. Application licenses are returned to pool
18. Any cpusets are deleted

For more detail about the lifecycle of a job, see section 3.2.7, “Summary of the Job Lifecycle”, on page 37 and section 3.2.8, “Detailed Description of Job Lifecycle”, on page 37.

2.1.2 Where and How Your PBS Job Runs

Your PBS jobs run on hosts that the administrator has designated to PBS as execution hosts. The PBS scheduler chooses one or more execution hosts that have the resources that your job requires.

PBS runs your jobs under your user account. This means that your login and logout files are executed for each job, and some of your environment goes with the job. It's important to make sure that your login and logout files don't interfere with your jobs; see section 1.4.2, “Setting Up Your Linux Environment”, on page 5.

2.1.3 The Job Identifier

After you submit a job, PBS returns a *job identifier*. Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>.<domain>

You'll need the job identifier for any actions involving the job, such as checking job status, modifying the job, tracking the job, or deleting the job.

The largest possible job ID is the 7-digit number 9,999,999. After this has been reached, job IDs start again at zero.

2.1.4 Shell Script(s) for Your Job

When PBS runs your job, PBS starts the top shell that you specify for the job. The top shell defaults to your login shell on the execution host, but you can set another using the job's Shell_Path_List attribute. See section 2.3.3.1, "Specifying the Top Shell for Your Job", on page 19.

Under Linux, if you do not specify a shell inside the job script, PBS defaults to using `/bin/sh`. If you specify a different shell inside the job script, the top shell spawns that shell to run the script; see section 2.3.3.2, "Specifying Job Script Shell or Interpreter", on page 20.

Under Windows, the job shell is the same as the top shell.

2.1.5 Scratch Space for Jobs

When PBS runs your job, it creates a temporary scratch directory for the job on each execution host. If your administrator has not specified a temporary directory, the root of the temporary directory is `/var/tmp`. Your administrator can specify a root for the temporary directory on each execution host using the $tmpdir MoM parameter. PBS sets the TMPDIR environment variable to the full path to the temporary scratch directory.

Under Windows, PBS creates the temporary directory and sets TMP to the value of the Windows %TMPDIR% environment variable. If your administrator has not specified a temporary directory, PBS creates the temporary directory under either `\winnt\temp` or `\windows\temp`.

PBS removes the directory when the job is finished. The location of the temporary directory is set by PBS; you should not set TMPDIR.

Your job script can access the scratch space. For example:

Linux:

```
cd $TMPDIR
```

Windows:

```
cd %TMPDIR%
```

For scratch space for MPI jobs, see section 5.2.3, "Caveats for Using MPIs", on page 84.

2.1.6 Types of Jobs

PBS allows you to submit standard batch jobs or interactive jobs. The difference is that while the interactive job runs, you have an interactive session running, giving you interactive access to job processes. There is no interactive access to a standard batch job. We cover interactive jobs in section 6.11, "Running Your Job Interactively", on page 121.

2.1.7 Job Input and Output Files

You can tell PBS to copy files or directories from any accessible location to the execution host, and to copy output files and directories from the execution host wherever you want. We describe how to do this in Chapter 3, "Job Input & Output Files", on page 31.

2.2 The PBS Job Script

2.2.1 Overview of a Job Script

A PBS job script consists of:

- An optional shell specification
- PBS directives
- Job tasks (programs or commands)

2.2.2 Types of Job Scripts

PBS allows you to use any of the following for job scripts:

- A Python, Perl, or other script that can run under Windows or Linux
- A shell script that runs under Linux
- Windows command or PowerShell batch script under Windows

2.2.2.1 Linux Shell Scripts

Since the job file can be a shell script, the first line of a shell script job file specifies which shell to use to execute the script. Your login shell is the default, but you can change this. This first line can be omitted if it is acceptable for the job file to be interpreted using the login shell. We recommend that you always specify the shell.

2.2.2.2 Python Job Scripts

PBS allows you to submit jobs using Python scripts under Windows or Linux. PBS includes a Python package, allowing Python job scripts to run; you do not need to install Python. To run a Python job script:

Linux:

```
qsub <script name>
```

Windows:

```
qsub -S %PBS_EXEC%\bin\pbs_python.exe <script name>
```

If the path contains any spaces, it must be quoted, for example:

```
qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <python job script>
```

You can include PBS directives in a Python job script as you would in a Linux shell script. For example:

```
% cat myjob.py
#!/usr/bin/python
#PBS -l select=1:ncpus=3:mem=1gb
#PBS -N HelloJob
print "Hello"
```

Python job scripts can access Win32 APIs, including the following modules:

- Win32api
- Win32con
- Pywintypes

2.2.2.2.i Debugging Python Job Scripts

You can run Python interactively, outside of PBS, to debug a Python job script. You use the Python interpreter to test parts of your script.

Under Linux, use the -i option to the pbs_python command, for example:

```
/opt/pbs/bin/pbs_python -i <return>
```

Under Windows, the -i option is not necessary, but can be used. For example, either of the following will work:

```
C:\Program Files\PBS\exec\bin\pbs_python.exe <return>
C:\Program Files\PBS\exec\bin\pbs_python.exe -i <return>
```

When the Python interpreter runs, it presents you with its own prompt. For example:

```
% /opt/pbs/bin/pbs_python -i <return>
>> print "hello"
hello
```

2.2.2.2.ii Python Windows Caveat

If you have Python natively installed, and you need to use the win32api, make sure that you import pywintypes before win32api, otherwise you will get an error. Do the following:

```
cmd> pbs_python
>> import pywintypes
>> import win32api
```

2.2.2.3 Windows Job Scripts

The Windows script can be a .exe or .bat file, or a Python or Perl script.

2.2.2.3.i Requirements for Windows Command Scripts

- Under Windows, comments in the job script must be in ASCII characters.
- Any .bat files that are to be executed within a PBS job script have to be prefixed with "call" as in:

  ```
  @echo off
  call E:\step1.bat
  call E:\step2.bat
  ```

  Without the "call", only the first .bat file gets executed and it doesn't return control to the calling interpreter.

  For example, an old job script that contains:

  ```
  @echo off
  E:\step1.bat
  E:\step2.bat
  ```

  should now be:

  ```
  @echo off
  call E:\step1.bat
  call E:\step2.bat
  ```

2.2.2.3.ii Windows Advice and Caveats

- In Windows, if you use notepad to create a job script, the last line is not automatically newline-terminated. Be sure to add one explicitly, otherwise, PBS job will get the following error message:

  ```
  More?
  ```

when the Windows command interpreter tries to execute that last line.

- Drive mapping commands are typically put in the job script.
- Do not use `xcopy` inside a job script. Use `copy`, `robocopy`, or `pbs_rcp` instead. The `xcopy` command sometimes expects input from the user. Because of this, it must be assigned an input handle. Since PBS does not create the job process with an input handle assigned, `xcopy` can fail or behave abnormally if used inside a PBS job script.
- PBS jobs submitted from `cygwin` execute under the native `cmd` environment, and not under `cygwin`.

2.2.3 Setting Job Characteristics

2.2.3.1 Job Attributes

PBS represents the characteristics of a job as *attributes*. For example, the name of a job is an attribute of that job, stored in the value of the job's Job_Name attribute. Some job attributes can be set by you, some can be set only by administrators, and some are set only by PBS. For a complete list of PBS job attributes, see "Job Attributes" on page 328 of the PBS Professional Reference Guide. Job attributes are case-insensitive.

2.2.3.2 Job Resources

PBS represents the things that a job might use as *resources*. For example, the number of CPUs and the amount of memory on an execution host are resources. PBS comes with a set of built-in resources, and your PBS administrator can define resources. You can see a list of all built-in PBS resources in Chapter 5, "List of Built-in Resources", on page 259. Resources are case-insensitive.

2.2.3.3 Setting Job Attributes

You can set job attributes and request resources using the following equivalent methods:

- Using specific options to the `qsub` command at the command line; for example, `-e <path>` to set the error path.
- Using PBS directives in the job script; for example, `#PBS Error_Path=<path>` to set the error path.

These methods have the same functionality. If you give conflicting options to `qsub`, the last option specified overrides any others. Options to the `qsub` command override PBS directives, which override defaults. Some job attributes and resources have default values; your administrator can set default values for some attributes and resources.

After the job is submitted, you can use the `qalter` command to change the job's characteristics.

2.2.3.4 Using PBS Directives

You can use PBS directives to set the values of job attributes. A directive has the directive prefix as the first non-whitespace characters. The default for the prefix is *#PBS*.

Put all your PBS directives at the top of the script file, above any commands. Any directive after an executable line in the script is ignored. For example, if your script contains "`@echo`", put that line below all PBS directives.

2.2.3.4.i Changing the Directive Prefix

By default, the text string "*#PBS*" is used by PBS to determine which lines in the job file are PBS directives. The leading "`#`" symbol was chosen because it is a comment delimiter to all shell scripting languages in common use on Linux systems. Because directives look like comments, the scripting language ignores them.

Under Windows, however, the command interpreter does not recognize the '#' symbol as a comment, and will generate a benign, non-fatal warning when it encounters each "*#PBS*" string. While it does not cause a problem for the batch job, it can be annoying or disconcerting to you. If you use Windows, you may wish to specify a different PBS directive, via either the PBS_DPREFIX environment variable, or the "-C" option to qsub. The qsub option overrides the environment variable. For example, we can direct PBS to use the string "*REM PBS*" instead of "*#PBS*" and use this directive string in our job script:

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb
REM PBS -j oe
date /t
.\my_application
date /t
```

Given the above job script, we can submit it to PBS in one of two ways:

```
set PBS_DPREFIX=REM PBS
qsub my_job_script
```

or

```
qsub -C "REM PBS" my_job_script
```

2.2.3.4.ii Caveats and Restrictions for PBS Directives

- You cannot use *PBS_DPREFIX* as the directive prefix.
- The limit on the length of a directive string is 4096 characters.

2.2.4 Job Tasks

These can be programs or commands. This is where you can specify an application to be run.

2.2.5 Job Script Names

We recommended that you avoid using special characters in job script names. If you must use them, on Linux you must escape them using the backslash ("\") character.

2.2.5.1 How PBS Parses a Job Script

PBS parses a job script in two parts. First, the qsub command scans the script looking for directives, and stops at the first executable line it finds. This means that if you want qsub to use a directive, it must be above any executable lines. Any directive below the first executable line is ignored. The first executable line is the first line that is not a directive, whose first non-whitespace character is not "#", and is not blank. For information on directives, see section 2.2.3.4, "Using PBS Directives", on page 16.

Second, lines in the script are processed by the job shell. PBS pipes the name of the job script file as input to the top shell, and the top shell executes the job shell, which runs the script. You can specify which shell is the top shell; see section 2.3.3.1, "Specifying the Top Shell for Your Job", on page 19, and, under Linux, which shell you want to run the script in the first executable line of the script; see section 2.3.3.2, "Specifying Job Script Shell or Interpreter", on page 20.

2.2.5.1.i Comparison Between Equivalent Linux and Windows Job Scripts

The following Linux and Windows job scripts produce the same results.

Linux:

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l select=mem=400mb
#PBS -j oe

date
./my_application
date
```

Windows:

```
REM PBS -l walltime=1:00:00
REM PBS -l select=mem=400mb
REM PBS -j oe

date /t
my_application
date /t
```

The first line in the Windows script does not contain a path to a shell because you cannot specify the path to the shell or interpreter inside a Windows job script. See section 2.3.3.2, “Specifying Job Script Shell or Interpreter”, on page 20.

The remaining lines of both files are almost identical. The primary differences are in file and directory path specifications, such as the use of drive letters, and slash vs. backslash as the path separator.

The lines beginning with "`#PBS`" and "`REM PBS`" are PBS directives. PBS reads down the job script until it finds the first line that is not a valid PBS directive, then stops. From there on, the lines in the script are read by the job shell or interpreter. In this case, PBS sees lines 6-8 as commands to be run by the job shell.

In our examples above, the "`-l <resource>=<value>`" lines request specific resources. Here, we request 1 hour of wall-clock time as a job-wide request, and 400 megabytes (MB) of memory in a chunk. We will cover requesting resources in Chapter 4, "Allocating Resources & Placing Jobs", on page 49.

The "`-j oe`" line requests that PBS *join* the `stdout` and `stderr` output streams of the job into a single stream. We will cover merging output in "Merging Output and Error Files” on page 43.

The last three lines are the command lines for executing the programs we wish to run. You can specify as many programs, tasks, or job steps as you need.

2.3 Submitting a PBS Job

2.3.1 Prerequisites for Submitting Jobs

Before you submit any jobs, set your environment appropriately. Follow the instructions in section 1.4, “Setting Up Your Environment”, on page 5.

2.3.2 Ways to Submit a PBS Job

You can use the qsub command to submit a normal or interactive job to PBS:

- You can call qsub with a job script; see section 2.3.3, "Submitting a Job Using a Script", on page 19
- You can call qsub with an executable and its arguments; see section 2.3.4, "Submitting Jobs by Specifying Executable on Command Line", on page 22
- You can call qsub and give keyboard input; see section 2.3.5, "Submitting Jobs Using Keyboard Input", on page 22

You can use an Altair front-end product to submit and monitor jobs; go to www.pbsworks.com.

2.3.3 Submitting a Job Using a Script

You submit a job to PBS using the qsub command. For details on qsub, see "qsub" on page 213 of the PBS Professional Reference Guide. To submit a PBS job, type the following:

- Linux shell script:
  qsub <name of shell script>
- Linux Python or Perl script:
  qsub <name of Python or Perl job script>
- Windows command script:
  qsub <name of job script>
- Windows Python script:
  qsub -S %PBS_EXEC%\bin\pbs_python.exe <name of python job script>

  If the path contains any spaces, it must be quoted, for example:

  qsub -S "%PBS_EXEC%\bin\pbs_python.exe" <name of python job script>

2.3.3.1 Specifying the Top Shell for Your Job

You can can specify the path and name of the shell to use as the top shell for your job. The rules for specifying the top shell are different for Linux and Windows; do not skip the following subsections numbered 2.3.3.1.i and 2.3.3.1.ii.

The Shell_Path_List job attribute specifies the top shell; the default is your login shell on the execution host. You can set this attribute using the the following:

- The "-S <path list>" option to qsub
- The #PBS Shell_Path_List=<path list> PBS directive

The option argument *path list* has this form:

<path>[@<hostname>][,<path>[@<hostname>],...]

You must supply a *path list* if you attempt to set Shell_Path_List, otherwise, you will get an error. You can specify only one path for any host you name. You can specify only one path that doesn't have a corresponding host name.

PBS chooses the path whose host name matches the name of the execution host. If no matching host is found, then PBS chooses the path specified without a host, if one exists.

2.3.3.1.i Specifying Job Top Shell Under Linux

On Linux, the job's top shell is the one MoM starts when she starts your job, and the job shell is the shell or interpreter that runs your job script commands.

Under Linux, you can use any shell such as csh or sh, by specifying qsub -S <path>. You cannot use Perl or Python as your top shell.

Example 2-1: Using bash:

```
qsub -S /bin/bash <script name>
```

2.3.3.1.ii Specifying Job Top Shell Under Windows

On Windows, the job shell is the same as the top shell.

Under Windows, you can specify a shell or an interpreter such as Perl or Python, and if your job script is Perl or Python, you must specify the language using an option to qsub; you cannot specify it in the job script.

Example 2-2: Running a Python script on Windows:

```
qsub -S "C:\Program Files\PBS\exec\bin\pbs_python.exe" <script name>
```

2.3.3.1.iii Caveats for Specifying Job Top Shell

If you specify a relative path for the top shell, the full path must be available in your PATH environment variable on the execution host(s). We recommend specifying the full path.

2.3.3.2 Specifying Job Script Shell or Interpreter

2.3.3.2.i Specifying Job Script Shell or Interpreter Under Linux

If you don't specify a shell for the job script, it defaults to /bin/sh. You can use any shell, and you can use an interpreter such as Perl or Python.

You specify the shell or interpreter in the first line of your job script. The top shell spawns the specified process, and this process runs the job script. For example, to use /bin/sh to run the script, use the following as the first line in your job script:

```
#!/bin/sh
```

To use Perl or Python to run your script, use the path to Perl or Python as the first line in your script:

```
#!/usr/bin/perl
```

or

```
#!/usr/bin/python
```

2.3.3.2.ii Specifying Job Script Shell or Interpreter Under Windows

Under Windows, the job shell or interpreter is the same as the top shell or interpreter. You can specify the top/job shell or interpreter, but not a separate job shell or interpreter. To use a non-default shell or interpreter, you must specify it using an option to qsub:

qsub -S <path to shell or interpreter> <script name>

2.3.3.3 Examples of Submitting Jobs Using Scripts

Example 2-3: Our job script is named "myjob". We can submit it by typing:

```
qsub myjob
```

and then PBS returns the job ID:

```
16387.exampleserver.exampledomain
```

Example 2-4: The following is the contents of the script named "myjob". In it, we name the job "testjob", and run a program called "myprogram":

```
#!/bin/sh
#PBS -N testjob
./myprogram
```

Example 2-5: The simplest way to submit a job is to give the script name as the argument to qsub, and hit return:

```
qsub <job script> <return>
```

If the script contains the following:

```
#!/bin/sh
./myapplication
```

you have simply told PBS to run myapplication.

2.3.3.4 Passing Arguments to Jobs

If you need to pass arguments to a job script, you can do the following:

- Use environment variables in your script, and pass values for the environment variables using -v or -V.

  For example, to use myinfile as the input to a.out, your job script contains the following:

  ```
  #PBS -N myjobname
  a.out < $INFILE
  ```

  You can then use the -V option:

  ```
  qsub -v INFILE=/tmp/myinfile <job script>
  ```

  For example, to use myinfile and mydata as the input to a.out, your job script contains the following:

  ```
  #PBS -N myjobname
  cat $INFILE $INDATA | a.out
  ```

  You can then use the -V option:

  ```
  qsub -v INFILE=/tmp/myinfile, INDATA=/tmp/mydata <job script>
  ```

  You can export the environment variable first:

  ```
  export INFILE=/tmp/myinfile
  qsub -V <job script>
  ```

- Use a here document. For example:

  ```
  qsub [option] [option] ... <return>
  #PBS <directive>
  ./jobscript.sh arg1      <^d>
  152.examplehost
  ```

If you need to pass arguments to a job, you can do any of the following:

- Pipe a shell command to qsub.

  For example, to directly pass myinfile and mydata as the input to a.out, type the following, or make them into a shell script:

  ```
  echo "a.out myinfile mydata" | qsub -l select=...
  ```

For example:

```
echo "jobscript.sh -a arg1 -b arg2" | qsub -l select=...
```

For example, to use an environment variable to pass `myinfile` as the input to `a.out`, type the following, or make them into a shell script:

```
export INFILE=/tmp/myinfile
export INDATA=/tmp/mydata
echo "a.out $INFILE $INDATA" | qsub
```

- Use `qsub --<executable> <arguments to executable>`. See section 2.3.4, "Submitting Jobs by Specifying Executable on Command Line", on page 22.

2.3.4 Submitting Jobs by Specifying Executable on Command Line

You can run a PBS job by specifying an executable and its arguments instead of a job script. When you run `qsub` this way, it runs the *executable* directly. It does not start a shell, so no shell initialization scripts are run, and execution paths and other environment variables are not set. There is not an easy way to run your command in a different directory. You should make sure that environment variables are set correctly, and you will usually have to specify the full path to the command.

To submit a job directly, you specify the executable on the command line:

qsub [<options>] -- <executable> [<arguments to executable>] <return>

For example, to run `myprog` with the arguments `a` and `b`:

```
qsub -- myprog a b <return>
```

To run `myprog` with the arguments `a` and `b`, naming the job *JobA*,

```
qsub -N JobA -- myprog a b <return>
```

To use environment variables you define earlier:

```
export INFILE=/tmp/myinfile
export INDATA=/tmp/mydata
qsub -- a.out $INFILE $INDATA
```

2.3.5 Submitting Jobs Using Keyboard Input

You can specify that `qsub` read input from the keyboard. If you run the `qsub` command, with the resource requests on the command line, and then press "enter" without naming a job file, PBS will read input from the keyboard. (This is often referred to as a "here document".) You can direct `qsub` to stop reading input and submit the job by typing on a line by itself a `control-d` (Linux) or `control-z`, then "enter" (Windows). You get the same behavior with and without a dash operand.

Note that, under Linux, if you enter a `control-c` while `qsub` is reading input, `qsub` will terminate the process and the job will not be submitted. Under Windows, however, often the `control-c` sequence will, depending on the command prompt used, cause `qsub` to submit the job to PBS. In such case, a `control-break` sequence will usually terminate the `qsub` command.

```
qsub [<options>] [-] <return>
    [<directives>]
    [<tasks>]
    ctrl-D
```

2.3.6 Submitting Windows Jobs

Your PBS complex may have all Windows execution and client (submission) hosts, or it may have some Linux and some Windows execution and client hosts. If your complex has some of each execution host, make sure that Windows jobs land on Windows execution hosts, whether you are submitting from Linux or Windows clients.

2.3.6.1 Submitting Windows Jobs from Windows Clients

If you have not already, run the pbs_login command at each submission host, initially and once for each password change:

```
echo <password>| pbs_login -p
```

When you submit a Windows job from a Windows client, make sure you request a Windows execution host. Request the arch resource set to "windows":

```
qsub -lselect=1:arch=windows
```

2.3.6.2 Submitting Windows Jobs from Linux Clients

If you have not already, run the `pbs_login` command at any Linux client host where you want to submit a Windows job. Set PBS_AUTH_METHOD to *pwd*:

```
export PBS_AUTH_METHOD=pwd; pbs_login
```

In order to submit a Windows job from a Linux client, specify that the architecture is Windows. For example:

```
export PBS_AUTH_METHOD=pwd; qsub -lselect=1:arch=windows -- pbs-sleep 100
```

2.4 Job Submission Recommendations and Advice

2.4.1 Trapping Signals in Script

You can trap signals in your job script. For example, you can trap preemption and suspension signals.

If you want to trap the signal in your job script, the signal may need to be trapped by all of the job's shells, depending on the signal.

The signal TERM is useful, because it is ignored by shells, but you can trap it and do useful things such as write out status.

Example 2-6: Ignore the listed signals:

```
trap "" 1 2 3 15
```

Example 2-7: Call the function "goodbye" for the listed signals:

```
trap goodbye 1 2 3 15
```

2.5 Job Submission Options

The table below lists the options to the qsub command, and points to an explanation of each:

Table 2-1: Options to the qsub Command

Option	Function and Page Reference
-A <account_string>	"Specifying Accounting String" on page 29
-a <date_time>	"Deferring Execution" on page 119
-C "<directive prefix>"	"Changing the Directive Prefix" on page 16
-c <interval>	"Using Checkpointing" on page 113
-e <path>	"Paths for Output and Error Files" on page 42
-G	"Submitting Interactive GUI Jobs on Windows" on page 125
-h	"Holding and Releasing Jobs" on page 115
-I	"Running Your Job Interactively" on page 121
-J X-Y[:Z]	"Submitting a Job Array" on page 152
-j <join>	"Merging Output and Error Files" on page 43
-k <keep>	"Keeping Output and Error Files on Execution Host" on page 44
-l <resource list>	"Requesting Resources" on page 51
-M <user list>	"Setting Email Recipient List" on page 26
-m <mail options>	"Specifying Email Notification" on page 25
-N <name>	"Specifying Job Name" on page 27
-o <path>	"Paths for Output and Error Files" on page 42
-p <priority>	"Setting Priority for Your Job" on page 120
-P <project>	"Specifying a Project for a Job" on page 27
-q <destination>	"Specifying Server and/or Queue" on page 29
-r <value>	"Allowing Your Job to be Re-run" on page 118
-R <remove options>	"Avoiding Creation of stdout and/or stderr" on page 43
-S <path list	"Specifying the Top Shell for Your Job" on page 19
-u <user list>	"Specifying Job Username" on page 28
-V	"Exporting All Environment Variables" on page 126
-v <variable list>	"Exporting Specific Environment Variables" on page 126
-W <attribute>=<value>	"Setting Job Attributes" on page 16
-W block=true	"Making qsub Wait Until Job Ends" on page 120
-W depend=<list>	"Using Job Dependencies" on page 107
-W group_list=<list>	"Specifying Job Group ID" on page 28

Table 2-1: Options to the `qsub` Command

Option	Function and Page Reference
`-W release_nodes_on_stageout=<value>`	"Releasing Unneeded Vnodes from Your Job" on page 128
`-W run_count=<value>`	"Controlling Number of Times Job is Re-run" on page 119
`-W sandbox=<value>`	"Staging and Execution Directory: User Home vs. Job-specific" on page 31
`-W stagein=<list>`	"Input/Output File Staging" on page 31
`-W stageout=<list>`	"Input/Output File Staging" on page 31
`-W umask=<value>`	"Changing Linux Job umask" on page 45
`-X`	"Receiving X Output from Interactive Linux Jobs" on page 124
`-z`	"Suppressing Printing Job Identifier to stdout" on page 30

2.5.1 Specifying Email Notification

For each job, PBS can send mail to designated recipients when that job or subjob reaches specific points in its lifecycle. There are points in the life of the job where PBS always sends email, and there are points where you can choose to receive email; see the table below for a list.

Table 2-2: Points in Job/Reservation Lifecycle when PBS Sends Mail

Point in Lifecycle	Always Sent or Optional?
Job cannot be routed, either because the job makes too many routing hops or because all destinations reject it	Optional. Mail is sent when `-m a` is specified. For subjobs, mail is sent when `-m aj` is specified.
Job is deleted by job owner	Optional; depends on `qdel -Wsuppress_email`
Job is deleted by someone other than job owner	Always
Job or subjob is aborted by PBS: Job or subjob cannot be executed because of bad user/group account, bad checkpoint/restart file, system error, bad resource request, or bad dependency	Optional. Mail is sent when `-m a` is specified. For subjobs, mail is sent when `-m aj` is specified.
Job is held by PBS with bad password hold	Always
Job begins execution	Optional
Job ends execution	Optional
Stagein fails	Always
All file stageout attempts fail	Always
Reservation is confirmed or denied	Always

PBS always sends you mail when your job or subjob is deleted. For job arrays, PBS sends one email per subjob.

You can restrict the number of job-related emails PBS sends when you delete jobs or subjobs; see section 2.5.1.3, "Restricting Number of Job Deletion Emails", on page 27.

2.5.1.1 Specifying Job Lifecycle Email Points

The set of points where PBS sends mail is specified in the Mail_Points job attribute. When you use the `-j` suboption with one or more of the other sub-options, PBS sends mail for each subjob; without this suboption, PBS sends mail only for jobs and parent array jobs. You can set the Mail_Points attribute using the following methods:

- The `-m <mail points>` option to `qsub`
- The `-m <mail points>` option to `qalter`
- The `#PBS -WMail_Points=<mail points>` PBS directive

The *mail points* argument is a string which consists of either:

- The single character "*n*"
- One or more of the characters "*a*", "*b*", and "*e*" with optional "*j*".

The following table lists the sub-options to the -m option:

Table 2-3: Sub-options to m Option

Suboption	Meaning
n	Do not send mail
a	Send mail when job or subjob is aborted by batch system. This is the default
b	Send mail when job or subjob begins execution Example: `Begun execution`
e	Send mail when job or subjob ends execution
j	Send mail for subjobs. Must be combined with one or more of a, b, or e sub-options.

Example 2-8: PBS sends mail when the job is aborted or ends:

```
qsub -m ae my_job
#PBS -m ae
```

2.5.1.2 Setting Email Recipient List

The list of recipients to whom PBS sends mail is specified in the Mail_Users job attribute. You can set the Mail_Users attribute using the following methods:

- The `-M <mail recipients>` option to `qsub`
- The `#PBS -WMail_Users=<mail recipients>` PBS directive

The mail recipients argument is a list of user names with optional hostnames in this format:

<username>[@<hostname>][,<username>[@<hostname>],...]

For example:

```
qsub -M user1@mydomain.com my_job
```

When you set this option for a job array, PBS sets the option for each subjob, and sends mail for each subjob.

2.5.1.3 Restricting Number of Job Deletion Emails

By default, when you delete a job or subjob, PBS sends you email. You can use `qdel -Wsuppress_email=<limit>` to restrict the number of emails sent to you each time you use `qdel`. This option behaves as follows:

limit >=*1*

You receive at most *limit* emails.

limit = *0*

PBS ignores this option.

limit =*-1*

You receive no emails.

2.5.2 Specifying Job Name

If you submit a job using a script without specifying a name for the job, the name of the job defaults to the name of the script. If you submit a job without using a script and without specifying a name for the job, the job name is STDIN.

You can specify the name of a job using the following methods:

- Using `qsub -N <job name>`
- Using `#PBS -N <job name>`
- Using `#PBS -WJob_Name=<job name>`

For example:

```
qsub -N myName my_job
#PBS -N myName
#PBS -WJob_Name=my_job
```

The job name can be up to 236 characters in length, and must consist of printable, non-whitespace characters. The first character must be alphabetic, numeric, hyphen, underscore, or plus sign.

2.5.3 Specifying a Project for a Job

In PBS, a project is a way to organize jobs independently of users and groups. You can use a project as a tag to group a set of jobs. Each job can be a member of up to one project.

Projects are not tied to users or groups. One user or group may run jobs in more than one project. For example, user Bob runs JobA in ProjectA and JobB in ProjectB. User Bill runs JobC in ProjectA. User Tom runs JobD in ProjectB. Bob and Tom are in Group1, and Bill is in Group2.

A job's project attribute specifies the job's project. See "project" on page 342 of the PBS Professional Reference Guide. You can set the job's project attribute in the following ways:

- At submission:
  - Using `qsub -P <project name>`
  - Via `#PBS project=<project name>`
- After submission, via `qalter -P <project name>`; see "qalter" on page 127 of the PBS Professional Reference Guide

2.5.4 Specifying Job Username

By default PBS runs your job under the username with which you log in. You may need to run your job under a different username depending on which PBS server runs the job. You can specify a list of user names under which the job can run. All but one of the entries in the list must specify the PBS server hostname as well, so that PBS can choose which username to use by looking at the hostname. You can include one entry in the list that does not specify a hostname; PBS uses this in the case where the job was sent to a server that is not in your list.

The list of user names is stored in the User_List job attribute. The value of this attribute defaults to the user name under which you logged in. There is no limit to the length of the attribute.

List entries are in the following format:

<username>@<hostname>[,<username>@<hostname> ...][,<username>]

You can set the value of User_List in the following ways:

- You can use `qsub -u <username>`
- You can use a directive: `#PBS User_List=<username list>`

Example 2-9: Our user is UserS on the submission host HostS, UserA on server ServerA, and UserB on server ServerB, and is UserC everywhere else. Note that this user must be UserA on all ExecutionA and UserB on all ExecutionB machines. Then our user can use "`qsub -u UserA@ServerA,UserB@ServerB,UserC`" for the job. The job owner will always be UserS. On Linux, UserA, UserB, and UserC must each have `.rhosts` files at their servers that list UserS.

2.5.4.1 Caveats for Changing Job Username

- Wherever your job runs, you must have permission to run the job under the specified username:
  - For Linux, see section 1.4.2.7, “User Authorization Under Linux”, on page 7.
  - For Windows, see section 1.4.3.4, “User Authorization under Windows”, on page 8.
- Usernames are limited to 256 characters.

2.5.5 Specifying Job Group ID

Your username can belong to more than one group, but each PBS job is only associated with one of those groups. By default, the job runs under the primary group. The job's group is specified in the group_list job attribute. You can change the group under which your job runs on the execution host either on the command line or by using a PBS directive:

```
qsub -W group_list=<group list>
#PBS group_list=<group list>
```

For example:

```
qsub -W group_list=grpA,grpB@jupiter my_job
```

The *<group list>* argument has the following form:

<group>[@<hostname>][,<group>[@<hostname>],...]

You can specify only one group name per host.

You can specify only one group without a corresponding host; that group name is used for execution on any host not named in the argument list.

The *group_list* defaults to the primary group of the username under which the job runs.

2.5.5.1 Group Names Under Windows

Under Windows, the primary group is the first group found for the username by PBS when querying the accounts database.

Under Windows, the default group assigned is determined by what the Windows API NetUserGetLocalGroup() and NetUserGetGroup() return as first entry. PBS checks the former output (the local groups) and returns the first group it finds. If the former call does not return any value, then it proceeds to the latter call (the Global groups). If PBS does not find any output on the latter call, it uses the default "Everyone".

We do not recommend depending on always getting "Users" in this case. Sometimes you may submit a job without the `-Wgroup_list` option, and get a default group of "None" assigned to your job.

2.5.6 Specifying Accounting String

You can associate an accounting string with your job by setting the value of the Account_Name job attribute. This attribute has no default value. You can set the value of Account_Name at the command line or in a PBS directive:

```
qsub -A <accounting string>
#PBS Account_Name=<accounting string>
```

The <*accounting string*> can be any string of characters; PBS does not attempt to interpret it.

2.5.7 Specifying Server and/or Queue

By default, PBS provides a default server and a default queue, so that jobs submitted without a server or queue specification end up in the default queue at the default server.

If your administrator has configured the PBS server with more than one queue, and has configured those queues to accept jobs from you, you can submit your job to a non-default queue.

- If you will submit jobs mainly to one non-default server, set the PBS_SERVER environment variable to the name of your preferred server. Once this environment variable is set to your preferred server, you don't need to specify that server when you submit a job to it.
- If you will submit jobs mostly to the default server, and just want to submit this one to a specific queue at a non-default server:
  - Use `qsub -q <queue name>@<server name>`
  - Use `#PBS -q <queue name>@<server name>`
- If you will submit jobs mostly to the default server, and just want to submit this one to the default queue at a non-default server:
  - Use `qsub -q @<server name>`
  - Use `#PBS -q @<server name>`
- You can submit your job to a non-default queue at the default server, or the server given in the PBS_SERVER environment variable if it is defined:
  - Use `qsub -q <queue name>`
  - Use `#PBS -q <queue name>`

If the PBS server has no default queue and you submit a job without specifying a queue, the `qsub` command will complain.

PBS or your administrator may move your job from one queue to another. You can see which queue has your job using `qstat [job ID]`. The job's queue attribute contains the name of the queue where the job resides.

Examples:

```
qsub -q queue my_job
qsub -q @server my_job
#PBS -q queue1
qsub -q queue1@myserver my_job
qsub -q queue1@myserver.mydomain.com my_job
```

2.5.7.1 Using or Avoiding Dedicated Time

Dedicated time is one or more specific time periods defined by the administrator. These are not repeating time periods. Each one is individually defined.

During dedicated time, the only jobs PBS starts are those in special dedicated time queues. PBS schedules non-dedicated jobs so that they will not run over into dedicated time. Jobs in dedicated time queues are also scheduled so that they will not run over into non-dedicated time. PBS will attempt to backfill around the dedicated-non-dedicated time borders.

PBS uses walltime to schedule within and around dedicated time. If a job is submitted without a walltime to a non-dedicated-time queue, it will not be started until all dedicated time periods are over. If a job is submitted to a dedicated-time queue without a walltime, it will never run.

To submit a job to be run during dedicated time, use the *-q <queue name>* option to *qsub* and give the name of the dedicated-time queue you wish to use as the queue name. Queues are created by the administrator; see your administrator for queue name(s).

2.5.8 Suppressing Printing Job Identifier to `stdout`

To suppress printing the job identifier to standard output, use the **-z** option to `qsub`. You can use it at the command line or in a PBS directive:

```
qsub -z my_job
#PBS -z
```

There is no associated job attribute for this option.

2.5.9 Running qsub in the Foreground

Normally, `qsub` runs in the background. You can run it in the foreground by using the `-f` option. By default, `qsub` attempts to communicate with a background `qsub` daemon that may have been instantiated from an earlier invocation. This background daemon can be holding onto an authenticated server connection, speeding up performance.

This option can be helpful when you are submitting a very short job which submits another job, or when you are running codes written in-house for Windows.

2.6 Job Submission Caveats

2.6.1 Caveats for Mixed Linux-Windows Operation

- You cannot submit a Linux job from a Windows client

3 Job Input & Output Files

3.1 Introduction to Job File I/O in PBS

PBS allows you to manage input files, output files, standard output, and standard error. PBS has two mechanisms for handling job files; you use staging for input and output files, and you select whether *stdout* and/or *stderr* are copied back using the Keep_Files job attribute.

3.2 Input/Output File Staging

File staging is a way to specify which input files should be copied onto the execution host before the job starts, and which output files should be copied off the execution host when it finishes.

3.2.1 Staging and Execution Directory: User Home vs. Job-specific

The job's staging and execution directory is the directory to which files are copied before the job runs, and from which output files are copied after the job has finished. This directory is either your home directory or a job-specific directory created by PBS just for this job. If you use job-specific staging and execution directories, you don't need to have a home directory on each execution host, as long as those hosts are configured properly. In addition, each job gets its own staging and execution directory, so you can more easily avoid filename collisions.

This table lists the differences between using your home directory for staging and execution and using a job-specific staging and execution directory created by PBS.

Table 3-1: Differences Between User Home and Job-specific Directory for Staging and Execution

Question Regarding Action, Requirement, or Setting	User Home Directory	Job-specific Directory
Does PBS create a job-specific staging and execution directory?	No	Yes
User's home directory must exist on execution host(s)?	Yes	No
Standard out and standard error automatically deleted when `qsub -k` option is used?	No	Yes
When are staged-out files are deleted?	Successfully staged-out files are deleted; others go to "undelivered"	Only after all are successfully staged out
Staging and execution directory deleted after job finishes?	No	Yes
What is job's sandbox attribute set to?	*HOME* or not set	*PRIVATE*

3.2.2 Using Job-specific Staging and Execution Directories

3.2.2.1 Setting the Job Staging and Execution Directory

The job's sandbox attribute controls whether PBS creates a unique job-specific staging and execution directory for this job. If the job's sandbox attribute is set to *PRIVATE*, PBS creates a unique staging and execution directory for the job. If sandbox is unset, or is set to *HOME*, PBS uses your home directory as the job's staging and execution directory. By default, the sandbox attribute is not set.

You can set the sandbox attribute via `qsub`, or through a PBS directive. For example:

```
qsub -Wsandbox=PRIVATE
```

The job's sandbox attribute cannot be altered while the job is executing.

Table 3-2: Effect of Job sandbox Attribute on Location of Staging and Execution Directory

Job's sandbox attribute	Effect
not set	Job's staging and execution directory is your home directory
HOME	Job's staging and execution directory is your home directory
PRIVATE	Job's staging and execution directory is a job-specific directory created by PBS. If the `qsub -k` option is used, output and error files are retained on the primary execution host in the staging and execution directory. This directory is removed, along with all of its contents, when the job finishes.

3.2.2.2 The jobdir Job Attribute and the PBS_JOBDIR Environment Variable

The job's jobdir attribute is a read-only attribute, set to the pathname of the job's staging and execution directory on the primary host. You can view this attribute by using `qstat -f`, only while the job is executing. The value of jobdir is not retained if a job is rerun; it is undefined whether jobdir is visible or not when the job is not executing.

The environment variable PBS_JOBDIR is set to the pathname of the staging and execution directory on the primary execution host. PBS_JOBDIR is added to the job script process, any job tasks, and the prologue and epilogue.

3.2.3 Attributes and Environment Variables Affecting Staging

The following attributes and environment variables affect staging and execution.

Table 3-3: Attributes and Environment Variables Affecting Staging

Job's Attribute or Environment Variable	Effect
sandbox attribute	Determines whether PBS uses user's home directory or creates job-specific directory for staging and execution. User-settable per job via `qsub -W` or through a PBS directive.
stagein attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W` or through a PBS directive.
stageout attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W` or through a PBS directive.
Keep_Files attribute	Determines whether output and/or error files remain on execution host. User-settable per job via `qsub -k` or through a PBS directive. If the Keep_Files attribute is set to `o` and/or `e` (output and/or error files remain in the staging and execution directory) and the job's sandbox attribute is set to *PRIVATE*, standard out and/or error files are removed, when the staging directory is removed at job end along with its contents.
jobdir attribute	Set to pathname of staging and execution directory on primary execution host. Read-only; viewable via `qstat -f`.
Remove_Files attribute	Specifies whether standard output and/or standard error files are automatically removed (deleted) upon job completion.
PBS_JOBDIR environment variable	Set to pathname of staging and execution directory on primary execution host. Added to environments of job script process, job tasks, and prologue and epilogue.
TMPDIR environment variable	Location of job-specific scratch directory.

3.2.4 Specifying Files To Be Staged In or Staged Out

You can specify files to be staged in before the job runs and staged out after the job runs by setting the job's stagein and stageout attributes. You can use options to `qsub`, or directives in the job script:

qsub -Wstagein=<execution path>@<input file storage host>:<input file storage path>[,...] -Wstageout=<execution path>@<output file storage host>:<output file storage path>[,...]

#PBS -W stagein=<execution path>@<input file storage host>:<input file storage path>[,...]

#PBS -W stageout=<execution path>@<output file storage host>:<output file storage path>[,...]

The name *execution path* is the name of the file in the job's staging and execution directory (on the execution host). The *execution path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage path* is the file name on the host specified by *storage host*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a hostname. The name can be absolute, or it can be relative to your home directory on the machine named *storage host*.

For stagein, the direction of travel is **from** *storage path* **to** *execution path*.

For stageout, the direction of travel is **from** *execution path* **to** *storage path*.

The following example shows how to use a directive to stagein a file named `grid.dat` located in the directory `/u/user1` on the host called serverA. The staged-in file is copied to the staging and execution directory and given the name `dat1`. Since *execution path* is evaluated relative to the staging and execution directory, it is not necessary to specify a full pathname for `data1`.

```
#PBS -W stagein=data1@serverA:/u/user1/grid.dat ...
```

To use the `qsub` option to stage in the file residing on myhost, in `/Users/myhome/mydata/data1`, calling it `input_data1` in the staging and execution directory:

```
qsub -W stagein=input_data1@myhost:/Users/myhome/mydata/data1
```

To stage more than one file or directory, use a comma-separated list of paths, and enclose the list in double quotes. For example, to stage two files `data1` and `data2` in:

```
qsub -W stagein="input1@hostA:/myhome/data1,input2@hostA:/myhome/data1"
```

3.2.5 Caveats and Requirements for Staging

3.2.5.1 Linux: Staging and Special Characters

If you need to use special characters, such as parentheses, in your file or directory names, enclose that part of the path in an extra layer of quotes. Syntax:

-W stageout="<execution path> @<storage host>:'<storage path>'"

Example:

```
-W stageout="myoutfile@myhost:'/home/user1/outfile(1234)'"
```

3.2.5.2 Windows: Staging and Special Characters or Paths

3.2.5.2.i Special Characters

Under Windows, if your path contains special characters such as spaces, backslashes (\), colons (:), or drive letter specifications, enclose the staging specification in double quotes. For example, to stage the grid.dat file on drive D at hostB to the execution file named "`dat1`" on drive `C`:

```
qsub -W stagein="dat1@hostB:D\Documents and Settings\grid.dat"
```

3.2.5.2.ii Using UNC Paths

If you use a UNC path to stage in or out, the hostname is optional. If you use a non-UNC path, the hostname is required.

3.2.5.3 Path Names for Staging

- It is advisable to use an absolute pathname for the *storage path*. Remember that the path to your home directory may be different on each machine, and that when using sandbox = *PRIVATE*, you may or may not have a home directory on all execution machines.
- Always use a relative pathname for *execution path* when the job's staging and execution directory is created by PBS, meaning when using a job-specific staging and execution directory, do not use an absolute path in `execution path`.

3.2.5.4 Required Permissions

You must have read permission for any files or directories that you will stage in, and write permission for any files or directories that you will stage out.

3.2.5.5 Warning About Ampersand

You cannot use the ampersand ("&") in any staging path. Staging will fail.

3.2.5.6 Interactive Jobs and File I/O

When an interactive job finishes, staged files may not have been copied back yet.

3.2.5.7 Copying Directories Into and Out Of the Staging and Execution Directory

You can stage directories into and out of the staging and execution directory the same way you stage files. The *storage path* and *execution path* for both stagein and stageout can be a directory. If you stagein or stageout a directory, PBS copies that directory along with all of its files and subdirectories. At the end of the job, the directory, including all files and subdirectories, is deleted. This can create a problem if multiple jobs are using the same directory.

3.2.5.8 Wildcards In File Staging

You can use wildcards when staging files and directories, according to the following rules.

- The asterisk "*" matches one or more characters.
- The question mark "?" matches a single character.
- All other characters match only themselves.
- Wildcards inside of quote marks are expanded.
- Wildcards cannot be used to match Linux files that begin with period "." or Windows files that have the "SYSTEM" or "HIDDEN" attributes.
- When using the `qsub` command line on Linux, you must prevent the shell from expanding wildcards. For some shells, you can enclose the pathnames in double quotes. For some shells, you can use a backslash before the wildcard.
- Wildcards can only be used in the source side of a staging specification. This means they can be used in the *storage path* specification for stagein, and in the *execution path* specification for stageout.
- When staging using wildcards, the destination must be a directory. If the destination is not a directory, the result is undefined. So for example, when staging out all `.out` files, you must specify a directory for *storage path*.
- Wildcards can only be used in the final path component, i.e. the basename.
- When wildcards are used during stagein, PBS will not automatically delete staged files at job end. Note that if PBS created the staging and execution directory, that directory and all its contents are deleted at job end.

3.2.6 Examples of File Staging

Example 3-1: Stage out all files from the execution directory to a specific directory:

Linux

```
-W stageout=*@myworkstation:/user/project1/case1
```

Windows

```
-W stageout=*@mypc:E:\project1\case1
```

Example 3-2: Stage out specific types of result files and disregard the scratch and other temporary files after the job terminates. The result files that are interesting for this example end in '.dat':

Linux

```
-W stageout=*.dat@myworkstation:project3/data
```

Windows

```
-W stageout=*.dat@mypc:C:\project\data
```

Example 3-3: Stage in all files from an application data directory to a subdirectory:

Linux

```
-W stagein=jobarea@myworkstation:crashtest1/*
```

Windows

```
-W stagein=jobarea@mypc:E:\crashtest1\*
```

Example 3-4: Stage in data from files and directories matching "wing*":

Linux

```
-W stagein=.@myworkstation:848/wing*
```

Windows

```
-W stagein=.@mypc:E:\flowcalc\wing*
```

Example 3-5: Stage in .bat and .dat files to job area:

Linux:

```
-W stagein=jobarea@myworkstation:/users/me/crash1.?at
```

Windows:

```
-W stagein=jobarea@myworkstation:C:\me\crash1.?at
```

3.2.6.1 Example of Using Job-specific Staging and Execution Directories

In this example, you want the file "jay.fem" to be delivered to the job-specific staging and execution directory given in PBS_JOBDIR, by being copied from the host "submithost". The job script is executed in PBS_JOBDIR and "jay.out" is staged out from PBS_JOBDIR to your home directory on the submission host (i.e., "storage host"):

```
qsub -Wsandbox=PRIVATE -Wstagein=jay.fem@submithost:jay.fem -Wstageout=jay.out@submithost:jay.out
```

3.2.7 Summary of the Job Lifecycle

This is a summary of the steps performed by PBS. The steps are not necessarily performed in this order.

- On each execution host, if specified, PBS creates a job-specific staging and execution directory.
- PBS sets PBS_JOBDIR and the job's jobdir attribute to the path of the job's staging and execution directory.
- On each execution host allocated to the job, PBS creates a job-specific temporary directory.
- PBS sets the TMPDIR environment variable to the pathname of the temporary directory.
- If any errors occur during directory creation or the setting of variables, the job is requeued.
- PBS stages in any files or directories.
- The prologue is run on the primary execution host, with its current working directory set to `PBS_HOME/mom_priv`, and with PBS_JOBDIR and TMPDIR set in its environment.
- The job is run as you on the primary execution host.
- The job's associated tasks are run as you on the execution host(s).
- The epilogue is run on the primary execution host, with its current working directory set to the path of the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in its environment.
- PBS stages out any files or directories.
- PBS removes standard error and/or standard output according to the value of the job's Remove_Files attribute.
- PBS removes any staged files or directories.
- PBS removes any job-specific staging and execution directories and their contents, and all TMPDIRs and their contents.
- PBS writes the final job accounting record and purges any job information from the server's database.

3.2.8 Detailed Description of Job Lifecycle

3.2.8.1 Creation of TMPDIR

For each host allocated to the job, PBS creates a job-specific temporary scratch directory for the job. If the temporary scratch directory cannot be created, the job is aborted.

3.2.8.2 Choice of Staging and Execution Directories

If the job's sandbox attribute is set to `PRIVATE`, PBS creates job-specific staging and execution directories for the job. If the job's sandbox attribute is set to `HOME`, or is unset, PBS uses your home directory for staging and execution.

3.2.8.2.i Job-specific Staging and Execution Directories

If the staging and execution directory cannot be created the job is aborted. If PBS fails to create a staging and execution directory, see the system administrator.

You should not depend on any particular naming scheme for the new directories that PBS creates for staging and execution.

3.2.8.2.ii User Home Directory as Staging and Execution Directory

You must have a home directory on each execution host. The absence of your home directory is an error and causes the job to be aborted.

3.2.8.3 Setting Environment Variables and Attributes

PBS sets PBS_JOBDIR and the job's jobdir attribute to the pathname of the staging and execution directory. The TMPDIR environment variable is set to the pathname of the job-specific temporary scratch directory.

3.2.8.4 Staging Files Into Staging and Execution Directories

PBS evaluates `execution path` and `storage path` relative to the staging and execution directory given in PBS_JOBDIR, whether this directory is your home directory or a job-specific directory created by PBS. PBS copies the specified files and/or directories to the job's staging and execution directory.

3.2.8.5 Running the Prologue

The MoM's prologue is run on the primary host as root, with the current working directory set to `PBS_HOME/mom_priv`, and with PBS_JOBDIR and TMPDIR set in its environment.

3.2.8.6 Job Execution

PBS runs the job script on the primary host as you. PBS also runs any tasks created by the job as you. The job script and tasks are executed with their current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in their environment.

3.2.8.7 Standard Out and Standard Error

The job's `stdout` and `stderr` files are created directly in the job's staging and execution directory on the primary execution host.

3.2.8.7.i Job-specific Staging and Execution Directories

If the `qsub -k` option is used, the `stdout` and `stderr` files will **not** be automatically copied out of the staging and execution directory at job end - they will be deleted when the directory is automatically removed.

3.2.8.7.ii User Home Directory as Staging and Execution Directory

If the `-k` option to `qsub` is used, standard out and/or standard error files are retained on the primary execution host instead of being returned to the submission host, and are not deleted after job end.

3.2.8.8 Running the Epilogue

PBS runs the epilogue on the primary host as root. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in its environment.

3.2.8.9 Staging Files Out and Removing Execution Directory

When PBS stages files out, it evaluates `execution path` and `storage path` relative to PBS_JOBDIR. Files that cannot be staged out are saved in `PBS_HOME/undelivered`.

3.2.8.9.i Job-specific Staging and Execution Directories

If PBS created job-specific staging and execution directories for the job, it cleans up at the end of the job. The staging and execution directory and all of its contents are removed, on all execution hosts.

3.2.8.10 Removing TMPDIRs and Files

PBS removes all TMPDIRs, along with their contents. If Remove_Files specifies output and/or error files, these files are removed.

3.2.9 Staging with Job Arrays

File staging is supported for job arrays. See "File Staging for Job Arrays" on page 153.

3.2.10 Stagein and Stageout Failure

3.2.10.1 File Stagein Failure

When stagein fails, the job is placed in a 30-minute wait to allow you time to fix the problem. Typically this is a missing file or a network outage. Email is sent to the job owner when the problem is detected. Once the problem has been resolved, the job owner or a PBS Operator may remove the wait by resetting the time after which the job is eligible to be run via the `-a` option to `qalter`. The server will update the job's comment with information about why the job was put in the wait state. When the job is eligible to run, it may run on different vnodes.

3.2.10.2 File Stageout Failure

When stageout encounters an error, there are three retries. PBS waits 1 second and tries again, then waits 11 seconds and tries a third time, then finally waits another 21 seconds and tries a fourth time. Email is sent to the job owner if all attempts fail. Files that cannot be staged out are saved in `PBS_HOME/undelivered`. See section 3.3.8.1, "Non-delivery of Output", on page 46.

3.3 Managing Output and Error Files

3.3.1 Default Behavior For Output and Error Files

By default, PBS copies the standard output (`stdout`) and standard error (`stderr`) files back to $PBS_O_WORKDIR on the submission host when a job finishes. When `qsub` is run, it sets $PBS_O_WORKDIR to the current working directory where the `qsub` command is executed. This means that if you want your job's `stdout` and `stderr` files to be delivered to your submission directory, you do not need to do anything.

The following options to the `qsub` command control where `stdout` and `stderr` are created and whether and where they are copied when the job is finished:

sandbox

By default, PBS runs the job script in the owner's home directory. If sandbox is set to *PRIVATE*, PBS creates a job-specific execution directory, and runs the job script there. See section 3.2.2.1, "Setting the Job Staging and Execution Directory", on page 32.

k

Specifies whether and which of `stdout` and `stderr` is retained in the job's execution directory. When set, this option overrides `o` and `e`. See section 3.3.5, "Keeping Output and Error Files on Execution Host", on page 44.

You can also specify that output and/or error files are written directly to the final destination. See section 3.3.6, "Writing Files Directly to Final Destination", on page 45.

o

Specifies destination for `stdout`. Overridden by `k` when `k` is set. See section 3.3.2, “Paths for Output and Error Files”, on page 42.

e

Specifies destination for `stderr`. Overridden by `k` when `k` is set. See section 3.3.2, “Paths for Output and Error Files”, on page 42.

R

Specifies whether standard output and/or standard error are deleted upon job completion. See section 3.3.3, “Avoiding Creation of stdout and/or stderr”, on page 43.

The following table shows how these options control creation and copying of `stdout` and `stderr`:

Table 3-4: How `k`, sandbox, `o`, and `e` Options to `qsub` Affect `stdout` and `stderr`

sandbox	`-k` (o, e, eo, oe)	-e, -o	-R	-k d	Where `stdout`, `stderr` Are Created	Where `stdout`, `stderr` Are Copied
HOME or unset	unset	unset	unset	unset	`PBS_HOME/spool`	PBS_O_WORKDIR, which is job submission directory
HOME or unset	unset	*<path>*	unset	unset	`PBS_HOME/spool`	Destination specified in `-o <path>` and/or `-e <path>`
HOME or unset	*e, o, eo, oe*	unset	unset	unset	Job submitter's home directory on execution host	Not copied; left in submitter's home directory on execution host, and not deleted
HOME or unset	*e, o, eo, oe*	*<path>*	unset	unset	Job submitter's home directory on execution host	Not copied; left in submitter's home directory on execution host, and not deleted
PRIVATE	unset	unset	unset	unset	Job-specific execution directory created by PBS	PBS_O_WORKDIR, which is job submission directory
PRIVATE	unset	*<path>*	unset	unset	Job-specific execution directory created by PBS	Destination specified in `-o <path>` and/or `-e <path>`
PRIVATE	*e, o, eo, oe*	unset	unset	unset	Job-specific execution directory created by PBS	Not copied; left in job-specific execution directory; deleted when job-specific execution directory is deleted
PRIVATE	*e, o, eo, oe*	*<path>*	unset	unset	Job-specific execution directory created by PBS	Not copied; left in job-specific execution directory; deleted when job-specific execution directory is deleted
any	any	any	*-R e/o*	any	Deleted regardless of where created	Does not exist, so not copied
any	any	any	unset	*-k d*	Final destination, if MoM can reach it	Does not exist, so not copied

- You can specify a path for `stdout` and/or `stderr`: see section 3.3.2, “Paths for Output and Error Files”, on page 42.
- You can merge `stdout` and `stderr`: see section 3.3.4, “Merging Output and Error Files”, on page 43.
- You can prevent creation of `stdout` and/or `stderr`: see section 3.3.3, “Avoiding Creation of stdout and/or stderr”, on page 43.
- You can choose whether to retain `stdout` and/or `stderr` on the execution host: see section 3.3.5, “Keeping Output and Error Files on Execution Host”, on page 44.
- You can specify that output and/or error files are written directly to the final destination. See section 3.3.6, “Writing Files Directly to Final Destination”, on page 45.
- You can specify that output and/or error files are deleted when the job finishes. See section 3.3.3, “Avoiding Creation of stdout and/or stderr”, on page 43.

3.3.2 Paths for Output and Error Files

3.3.2.1 Default Paths for Output and Error Files

By default, PBS names the output and error files for your job using the job name and the job's sequence number. The output file name is specified in the Output_Path job attribute, and the error file name is specified in the Error_Path job attribute.

The default output filename has this format:

<job name>.o<sequence number>

The default error filename has this format:

<job name>.e<sequence number>

The *job name*, if not specified, defaults to the script name. For example, if the job ID is `1234.exampleserver` and the script name is "`myscript`", the error file is named `myscript.e1234`. If you specify a name for your job, the script name is replaced with the job name. For example, if you name your job "fixgamma", the output file is named `fixgamma.o1234`.

For details on naming your job, see section 2.5.2, “Specifying Job Name”, on page 27.

3.3.2.2 Specifying Paths

You can specify the path and name for the output and error files for each job, by setting the value for the Output_Path and Error_Path job attributes. You can set these attributes using the following methods:

- Use the `-o <output path>` and `-e <error path>` options to `qsub`
- Use `#PBS Output_Path=<path>` and `#PBS Error_Path=<path>` directives in the job script

The `path` argument has the following form:

[<hostname>:]<pathname>

where *hostname* is the name of a host and *pathname* is the path name on that host.

You can specify relative or absolute paths. If you specify only a file name, it is assumed to be relative to your home directory. Do not use variables in the path.

The following examples show how you can specify paths:

```
#PBS -o /u/user1/myOutputFile
#PBS -e /u/user1/myErrorFile

qsub -o myOutputFile my_job
qsub -o /u/user1/myOutputFile my_job
qsub -o myWorkstation:/u/user1/myOutputFile my_job
qsub -e myErrorFile my_job
qsub -e /u/user1/myErrorFile my_job
qsub -e myWorkstation:/u/user1/myErrorFile my_job
```

3.3.2.3 Specifying Paths from Windows Hosts

3.3.2.3.i Using Special Characters in Paths

If you submit your job from a Windows host, you may end up using special characters such as spaces, backslashes ("\"), and colons (":") for specifying pathnames, and you may need drive letter specifications. The following examples are allowed:

```
qsub -o \temp\my_out job.scr
qsub -e "myhost:e:\Documents and Settings\user\Desktop\output"
```

The error output of the example job is to be copied onto the `e:` drive on `myhost` using the path "`\Documents and Settings\user\Desktop\output`".

3.3.2.3.ii Using UNC Paths

If you use a UNC path for output or error files, the hostname is optional. If you use a non-UNC path, the hostname is required.

3.3.2.4 Caveats for Paths

Enclose arguments to `qsub` in quotes if the arguments contain spaces.

3.3.3 Avoiding Creation of `stdout` and/or `stderr`

For each job, PBS always creates the job's output and error files. The location where files are created is listed in Table 3-4, “How k, sandbox, o, and e Options to qsub Affect stdout and stderr,” on page 41.

If you do not want `stdout` and/or `stderr`, you can do either of the following:

- Specify that PBS deletes the file(s) when the job finishes, using the `-R` option to `qsub` or `qalter`. The `-R` option takes *o*, *e*, *eo*, or *oe* as sub-options. For example, to have PBS delete the error file:

  ```
  qsub -R e job.sh
  ```

- Redirect them to `/dev/null` within the job script. For example, to redirect `stdout` and `stderr` to `/dev/null`:

  ```
  exec >&/dev/null 1>&2
  ```

- Standard output and standard error are normally written to a location such as `/var/spool`, then copied to their final location. To avoid creating these files at all, and to avoid copying them, use direct write to send them to `/dev/null`:

  ```
  qsub -koed -o /dev/null -e /dev/null
  ```

  Your administrator must also set up the MoM's configuration file to support this.

3.3.4 Merging Output and Error Files

By default, PBS creates separate standard output and standard error files for each job. You can specify that `stdout` and `stderr` are to be joined by setting the job's Join_Path attribute. The default for the attribute is *n*, meaning that no joining takes place. You can set the attribute using the following methods:

- Use `qsub -j <joining option>`
- Use `#PBS Join_Path=<joining option>`

You can specify one of the following *joining options*:

oe

Standard output and standard error are merged, intermixed, into a single stream, which becomes standard output.

eo
: Standard output and standard error are merged, intermixed, into a single stream, which becomes standard error.

n
: Standard output and standard error are not merged.

For example, to merge standard output and standard error for my_job into standard output:

```
qsub -j oe my_job
#PBS -j oe
```

3.3.5 Keeping Output and Error Files on Execution Host

By default, PBS copies `stdout` and `stderr` to the job's submission directory. You can specify that PBS keeps `stdout`, `stderr`, or both in the job's execution directory on the execution host. This behavior is controlled by the job's Keep_Files attribute. You can set this attribute to one of the following values:

e
: PBS keeps `stderr` in the job's staging and execution directory on the primary execution host.

o
: PBS keeps `stdout` in the job's staging and execution directory on the primary execution host.

eo, oe
: PBS keeps both standard output and standard error on the primary execution host, in the job's staging and execution directory.

n
: PBS does not keep either file on the execution host.

d
: PBS writes both `stdout` and `stderr` to their final destinations. Overrides `o` and `e` options. See section 3.3.6, "Writing Files Directly to Final Destination", on page 45.

The default value for Keep_Files is "*n*".

You can set the value of the Keep_Files job attribute using the following methods:

- Use `qsub -k <keep option>`
- Use `#PBS Keep_Files=<keep option>`

For example, you can use either of the following to keep both standard output and standard error on the execution host:

```
qsub -k oe my_job
#PBS -k oe
```

3.3.5.1 Caveats for Keeping Files on Execution Host

- When a job finishes, its job-specific execution directory, and all files in that directory, are deleted. If you specified that `stdout` and/or `stderr` should be kept on the execution host, any files you specified are deleted as well.
- The `qsub -k` option overrides the `-o` and `-e` options. For example, if you specify `qsub -k o -o <path>`, `stdout` is kept on the execution host, and is not copied to the path you specified.

3.3.6 Writing Files Directly to Final Destination

If the MoM on the primary execution host can reach the final destination, she can write the job's standard output and standard error files to that destination. To be reachable, the final destination host and path must either be on the execution host, or be mapped from the primary execution host via the $usecp directive in the MoM configuration file. To specify that standard output and/or standard error should be written directly to their final destinations, use the `d` sub-option to the `-k` option to `qsub` or `qalter`.

For example, to directly write both output and error to their final destinations:

```
qsub -koed job.sh
```

To directly write output to its final destination, and let error go through normal spooling and staging:

```
qsub -kod job.sh
```

3.3.7 Changing Linux Job `umask`

On Linux, whenever your job copies or creates a file or directory on the execution host, MoM uses `umask` to determine the permissions for the file or directory. If you do not specify a value for `umask`, MoM uses the system default. You can specify a value using the following methods:

- Use `qsub -W umask=<value>`
- Use `#PBS umask=<value>`

This applies when staging or copying files or directories to the execution host, or writing `stdout` or `stderr` on the execution host.

In the following example, we set `umask` to *022*, to have files created with write permission for owner only. The desired permissions are `-rw-r--r--`.

```
qsub -W umask=022 my_job
#PBS -W umask=022
```

3.3.7.1 Caveats

This feature does not apply to Windows.

3.3.8 Troubleshooting File Delivery

File delivery is handled by MoM on the execution host. For a description of how file delivery works, see "Setting File Transfer Mechanism" on page 559 in the PBS Professional Administrator's Guide.

For troubleshooting file delivery, see "Troubleshooting File Transfer" on page 564 in the PBS Professional Administrator's Guide.

3.3.8.1 Non-delivery of Output

If the output of a job cannot be delivered to you, it is saved in a special directory named `PBS_HOME/undelivered` and mail is sent to you. The typical causes of non-delivery are:

1. The destination host is not trusted and you do not have a `.rhosts` file.
2. An improper path was specified.
3. A directory in the specified destination path is not writable.
4. Your `.cshrc` on the destination host generates output when executed.
5. The path specified by PBS_SCP in `pbs.conf` is incorrect.
6. The `PBS_HOME/spool` directory on the execution host does not have the correct permissions. This directory must have mode *1777 drwxrwxrwxt* (on Linux) or "Full Control" for "Everyone" (on Windows).

3.3.9 Caveats for Output and Error Files

3.3.9.1 Retaining Files on Execution Host

When PBS creates a job-specific staging and execution directory and you use the -k option to `qsub` or you specify *o* and/or *e* in the Keep_Files attribute, the files you requested kept on the execution host are deleted when the job-specific staging and execution directory is deleted at the end of the job.

3.3.9.2 Standard Output and Error Appended When Job is Rerun

If your job runs and writes to `stdout` or `stderr`, and then is rerun, meaning that another job with the same name is run, PBS appends the `stdout` of the second run to that of the first, and appends the `stderr` of the second run to that of the first.

3.3.9.3 Windows Mapped Drives and PBS

In Windows, when you map a drive, it is mapped locally to your session. The mapped drive cannot be seen by other processes outside of your session. A drive mapped on one session cannot be un-mapped in another session even if the user is the same. This has implications for running jobs under PBS. Specifically if you map a drive, `chdir` to it, and submit a job from that location, the vnode that executes the job may not be able to deliver the files back to the same location from which you issued `qsub`. The workaround is to tell PBS to deliver the files to a local, non-mapped, directory. Use the "`-o`" or "`-e`" options to `qsub` to specify the directory location for the job output and error files. For details see section 3.3.2, “Paths for Output and Error Files”, on page 42.

3.3.9.4 Harmless `csh` Error Message

If your login shell is `csh` the following message may appear in the standard output of a job:

```
Warning: no access to tty, thus no job control in this shell
```

This message is produced by many `csh` versions when the shell determines that its input is not a terminal. Short of modifying `csh`, there is no way to eliminate the message. Fortunately, it is just an informative message and has no effect on the job.

3.3.9.5 Interactive Jobs and File I/O

When an interactive job finishes, `stdout` and/or `stderr` may not have been copied back yet.

3.3.9.6 Write Permissions Required

- You must have write permission for any directory where you will copy `stdout` or `stderr`.
- Root must be able to write in `PBS_HOME/spool`.

4 Allocating Resources & Placing Jobs

4.1 What is a Vnode?

A virtual node, or vnode, is an abstract object representing a set of resources which form a usable part of a machine. This could be an entire host, or a nodeboard or a blade. A single host can be made up of multiple vnodes.

A host is any computer. Execution hosts used to be called nodes, and are still often called nodes outside of the PBS documentation. PBS views hosts as being composed of one or more vnodes.

PBS manages and schedules each vnode independently. Jobs run on one or more vnodes. Each vnode has its own set of attributes; see "Vnode Attributes" on page 320 of the PBS Professional Reference Guide.

4.1.1 Deprecated Vnode Types

All vnodes are treated alike, and are treated the same as what were once called "time-shared nodes". The types "time-shared" and "cluster" are deprecated. The `:ts` suffix is deprecated. It is silently ignored, and not preserved during rewrite.

The vnode attribute ntype was only used to distinguish between PBS and Globus vnodes. Globus can still send jobs to PBS, but PBS no longer supports sending jobs to Globus. The ntype attribute is read-only.

4.2 PBS Resources

4.2.1 Introduction to PBS Resources

In this section, "Introduction to PBS Resources", we will briefly cover the basics of PBS resources. For a thorough discussion, see "Using PBS Resources" on page 229 in the PBS Professional Administrator's Guide, especially sections 5.4 and 5.5. For a complete description of each PBS resource, see Chapter 5, "List of Built-in Resources", on page 259.

PBS resources represent things such as CPUs, memory, application licenses, switches, scratch space, and time. They can also represent whether or not something is true, for example, whether a machine is dedicated to a particular project.

PBS provides a set of built-in resources, and allows the administrator to define additional custom resources. Custom resources are used for application licenses, scratch space, etc., and are defined by the administrator. Custom resources are used the same way built-in resources are used. PBS supplies the following types of resources:

Boolean

Name of Boolean resource is a string.

Values:

TRUE, True, true, T, t, Y, y, 1

FALSE, False, false, F, f, N, n, 0

Duration

A period of time, expressed either as

An integer whose units are seconds

or

[[hours:]minutes:]seconds[.milliseconds]
in the form:
[[HH:]MM:]SS[.milliseconds]

Milliseconds are rounded to the nearest second.

Float

Floating point. Allowable values: [+-] 0-9 [[0-9] ...][.][[0-9] ...]

Long

Long integer. Allowable values: 0-9 [[0-9] ...], and + and -

<queue name>@<server name>

Size

Number of bytes or words. The size of a word is 64 bits.

Format: *<integer>[<suffix>]*

where *suffix* can be one of the following:

Table 4-1: Size in Bytes

Suffix	Meaning	Size
b or w	Bytes or words	1
kb or kw	Kilobytes or kilowords	2 to the 10th, or 1024
mb or mw	Megabytes or megawords	2 to the 20th, or 1,048,576
gb or gw	Gigabytes or gigawords	2 to the 30th, or 1,073,741,824
tb or tw	Terabytes or terawords	2 to the 40th, or 1024 gigabytes
pb or pw	Petabytes or petawords	2 to the 50th, or 1,048,576 gigabytes

Default: *bytes*

Note that a scheduler rounds all resources of type size up to the nearest kb.

String

Any character, including the space character.

Only one of the two types of quote characters, " or ', may appear in any given value.

Values:[_a-zA-Z0-9][[-_a-zA-Z0-9 ! " # $ % ´ ( ) * + , - . / : ; < = > ? @ [\ ] ^ _ ' { | } ~] ...]

String resource values are case-sensitive. No limit on length.

String Array

Comma-separated list of strings.

Strings in string_array may not contain commas. No limit on length.

Python type is *str*.

A string array resource with one value works exactly like a string resource.

See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide for a listing of built-in resources.

For some systems, PBS creates specific custom resources.

The administrator can specify which resources are available at the server, each queue, and each vnode. Resources defined at the queue or server level apply to an entire job. Resources defined at the vnode level apply only to the part of the job running on that vnode.

Jobs can request resources. The scheduler matches requested resources with available resources, according to rules defined by the administrator. PBS always places jobs where it finds the resources requested by the job. PBS will not place a job where that job would use more resources than PBS thinks are available. For example, if you have two jobs, each requesting 1 CPU, and you have one vnode with 1 CPU, PBS will run only one job at a time on the vnode.

PBS can enforce limits on resource usage by jobs; see section 4.5, "Limits on Resource Usage", on page 61.

4.2.2 Glossary

Chunk

A set of resources allocated as a unit to a job. Specified inside a selection directive. All parts of a chunk come from the same host. In a typical MPI (Message-Passing Interface) job, there is one chunk per MPI process.

Chunk-level resource, host-level resource

A resource that is available at the host level, for example, CPUs or memory. Chunk resources are requested inside of a selection statement. The resources of a chunk are to be applied to the portion of the job running in that chunk.

Chunk resources are requested inside a select statement.

Job-wide resource, server resource, queue resource

A job-wide resource, also called a server-level or queue-level resource, is a resource that is available to the entire job at the server or queue.

A job-wide resource is available to be consumed or matched at the server or queue if you set the server or queue resources_available.<resource name> attribute to the available or matching value. For example, you can define a custom resource called *FloatingLicenses* and set the server's resources_available.FloatingLicenses attribute to the number of available floating licenses.

Examples of job-wide resources are shared scratch space, application licenses, or walltime.

A job can request a job-wide resource for the entire job, but not for individual chunks.

4.3 Requesting Resources

Your job can request resources that apply to the entire job, or resources that apply to job chunks. For example, if your entire job needs an application license, your job can request one job-wide license. However, if one job process needs two CPUs and another needs 8 CPUs, your job can request two chunks, one with two CPUs and one with eight CPUs. Your job cannot request the same resource in a job-wide request and a chunk-level request.

PBS supplies resources such as walltime that can be used only as job-wide resources, and other resources, such as ncpus and mem, that can be used only as chunk resources. A resource is either job-wide or chunk-level, but not both. The description of each resource tells you which way to use the resource; see "List of Built-in Resources" on page 259 of the PBS Professional Reference Guide.

We will cover the details of requesting resources in section 4.3.2, "Requesting Job-wide Resources", on page 52 and section 4.3.3, "Requesting Resources in Chunks", on page 53.

4.3.1 Quick Summary of Requesting Resources

Job-wide resources are requested in *<resource neme>=<value>* pairs. You can request job-wide resources using any of the following:

- The `qsub -l <resource name>=<value>` option

  You can request multiple resources, using either format:

  `-l <resource>=<value>,<resource>=<value>`

  `-l <resource>=<value> -l <resource>=<value>`

- One or more `#PBS -l <resource name>=<value>` directives

Chunk resources are requested in *chunk specifications* in a *select statement*. You can request chunk resources using any of the following:

- The `qsub -l select=[N:][<chunk specification>][+[N:]<chunk specification>]` option
- A `#PBS -l select=[N:][<chunk specification>][+[N:]<chunk specification>]` directive

Format for requesting both job-wide and chunk resources:

`qsub ...` (non-resource portion of job)
`-l <resource>=<value>` (this is the job-wide request)
`-l select=<chunk>[+<chunk>]` (this is the selection statement)

PBS supplies several commands that you can use to request resources or alter resource requests:

- The `qsub` command (both via command-line and in PBS directives)
- The `pbs_rsub` command (via command-line only)
- The `qalter` command (via command-line only)

4.3.2 Requesting Job-wide Resources

Your job can request resources that apply to the entire job in *job-wide* resource requests. A job-wide resource is designed to be used by the entire job, and is available at the server or a queue, but not at the host level. Job-wide resources are used for requesting floating application licenses or other resources not tied to specific vnodes, such as cput and walltime.

Job-wide resources are requested outside of a selection statement, in this form:

-l <resource name>=<value>[,<resource name>=<value> ...]

A resource request "outside of a selection statement" means that the resource request comes after "`-l`", but not after "`-lselect=`". In other words, you cannot request a job-wide resource in chunks.

For example, to request one hour of walltime for a job:

-l walltime=1:00:00

You can request job-wide resources using any of the following:

- The `qsub -l <resource name>=<value>` option

You can request multiple resources, using either format:

```
-l <resource>=<value>,<resource>=<value>
-l <resource>=<value> -l <resource>=<value>
```

- One or more `#PBS -l <resource name>=<value>` directives

4.3.3 Requesting Resources in Chunks

A *chunk* specifies the value of each resource in a set of resources which are to be allocated as a unit to a job. It is the smallest set of resources to be allocated to a job. All of a chunk is taken from a single host. One chunk may be broken across vnodes, but all participating vnodes must be from the same host.

Your job can request chunk resources, which are resources that apply to the host-level parts of the job. Host-level resources can only be requested as part of a chunk. Server or queue resources cannot be requested as part of a chunk. A chunk resource is used by the part of the job running on that chunk, and is available at the host level. Chunks are used for requesting host-related resources such as CPUs, memory, and architecture.

Chunk resources are requested inside a select statement. A select statement has this form:

-l select=[N:]<chunk>[+[N:]<chunk> ...]

Now, we'll explain the details. A single chunk is requested using this form:

-l select=<resource name>=<value>[:<resource name>=<value>...]

For example, one chunk might have 2 CPUs and 4GB of memory:

```
-l select=ncpus=2:mem=4gb
```

To request multiples of a chunk, prefix the chunk specification by the number of chunks:

-l select=[<number of chunks>]<chunk specification>

For example, to request six of the previous chunk:

```
-l select=6:ncpus=2:mem=4gb
```

If you don't specify *N*, the number of chunks, it is taken to be *1*.

To request different chunks, concatenate the chunks using the plus sign ("+"):

-l select=[<number of chunks>]<chunk specification>+[<number of chunks>]<chunk specification>

For example, to request two sets of chunks where one set of 6 chunks has 2 CPUs per chunk, and one set of 3 chunks has 8 CPUs per chunk, and both sets have 4GB of memory per chunk:

```
-l select=6:ncpus=2:mem=4gb+3:ncpus=8:mem=4GB
```

No spaces are allowed between chunks.

You must specify all your chunks in a single select statement.

You can request chunk resources using any of the following:

- The `qsub -l select=[N:][<chunk specification>][+[N:]<chunk specification>]` option
- A `#PBS -l select=[N:][<chunk specification>][+[N:]<chunk specification>]` directive

4.3.4 Requesting Boolean Resources

A resource request can specify whether a Boolean resource should be *True* or *False*.

Example 4-1: Some vnodes have green=*True* and some have red=*True*, and you want to request two vnodes, each with one CPU, all green and no red:

```
-l select=2:green=true:red=false:ncpus=1
```

Example 4-2: This job script snippet has a job-wide request for walltime and a chunk request for CPUs and memory where the Boolean resource HasMyApp is *True*:

```
#PBS -l walltime=1:00:00
#PBS -l select=ncpus=4:mem=400mb:HasMyApp=true
```

Keep in mind the difference between requesting a vnode-level boolean and a job-wide boolean:

```
qsub -l select=1:green=True
```

requests a vnode with green set to *True*. However,

```
qsub -l green=True
```

requests green set to *True* on the server and/or queue.

4.3.5 Requesting Application Licenses

Application licenses are managed as resources defined by your PBS administrator. PBS doesn't actually check out the licenses; the application being run inside the job's session does that.

4.3.5.1 Requesting Floating Application Licenses

A site-wide floating license is typically configured as a server-level, job-wide resource.

To request a job-wide application license called AppF, use:

```
qsub -l AppF=<number of licenses> <other qsub arguments>
```

If only certain hosts can run the application, they will typically have a host-level Boolean resource set to *True*.

The job-wide resource AppF is a numerical resource indicating the number of licenses available at the site. The host-level Boolean resource haveAppF indicates whether a given host can run the application. To request the application license and the vnodes on which to run the application:

```
qsub  -l AppF=<number of licenses> <other qsub arguments>
      -l select=haveAppF=True
```

PBS queries the license server to find out how many floating licenses are available at the beginning of each scheduling cycle. PBS doesn't actually check out the licenses, the application being run inside the job's session does that.

4.3.5.2 Requesting Node-locked Application Licenses

Node-locked application licenses are available at the vnode(s) that are licensed for the application. These are host-level (chunk) resources that are requested inside of a select statement.

4.3.5.2.i Requesting Per-host Node-locked Application Licenses

Per-host node-locked application licenses are typically configured as a Boolean resource that indicates whether or not the required license is available at that host.

When requesting Boolean-valued per-host node-locked licenses, request one per host. Format:

qsub -l select=<Boolean resource name>=true:<rest of chunk specification>

Example 4-3: The Boolean resource runsAppA specifies whether this vnode has the necessary license. To request a host with a per-host node-locked license for AppA in one chunk:

```
qsub -l select=1:runsAppA=1 <job script>
```

4.3.5.2.ii Requesting Per-use Node-locked Application Licenses

Per-use node-locked application licenses are typically configured as a consumable numeric resource so that the host(s) that run the application have the number of licenses that can be used at one time.

When requesting numerical per-use node-locked licenses, request the required number of licenses for each host:

qsub -l select=<consumable resource name>=<required amount>:<rest of chunk specification>

Example 4-4: The consumable resource named AppB indicates the number of available per-use application licenses on a host. To request a host with a per-use node-locked license for AppB, where you'll run one instance of AppB on two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppB=1
```

4.3.5.2.iii Requesting Per-CPU Node-locked Application Licenses

Per-CPU node-locked licenses are typically arranged so that the host has one license for each licensed CPU. The PBS administrator configures a consumable numerical resource indicating the number of available licenses.

You must request one license for each CPU. When requesting numerical per-use node-locked licenses, request the required number of licenses for each host:

qsub -l select=<per-CPU resource name>=<required amount>:<rest of chunk specification>

Example 4-5: The numerical consumable resource named AppC indicates the number of available per-CPU licenses. To request a host with two per-CPU node-locked licenses for AppC, where you'll run a job using two CPUs in one chunk:

```
qsub -l select=1:ncpus=2:AppC=2
```

4.3.6 Requesting Scratch Space

Scratch space on a machine is configured as a host-level dynamic resource. Ask your administrator for the name of the scratch space resource.

When requesting scratch space, include the resource in your chunk request:

-l select=<scratch resource name>=<amount of scratch needed>:<rest of chunk specification>

Example 4-6: Your administrator has named the scratch resource "dynscratch". To request 10MB of scratch space in one chunk:

```
-l select=1:ncpus=N:dynscratch=10MB
```

4.3.7 Requesting GPUs

Your PBS job can request GPUs. How you request GPUs depends on whether PBS uses cgroups to manage GPUs; check with your administrator.

4.3.7.1 Requesting GPUs Managed via Cgroups

Recommended: On Linux only, PBS can be configured to use cgroups to fence GPUs off, so that when your job requests GPUs it automatically gets exclusive use of its GPUs. You don't have to request exclusivity. When PBS uses cgroups to manage GPUs, you request the number of GPUs you want via the ngpus resource:

qsub -l select=ngpus=<value>:<rest of chunk specification>

When GPUs are managed via cgroups, jobs requesting memory will use that amount both for physical memory and for swap. For example, a job that requests 20GB and uses 16GB but reads a 50GB file can only swap 4GB at a time. So if a job requires 32GB of application memory but also requires 5GB of private file cache to perform adequately, then it needs to request 37GB.

4.3.7.2 Requesting GPUs Not Managed via Cgroups

On Windows or Linux, when PBS is not using cgroups to manage GPUs, your administrator can configure PBS to support any of the following:

- ("Basic GPU scheduling") Job uses non-specific GPUs and exclusive use of a node
- ("Advanced GPU scheduling") Job uses non-specific GPUs and shared use of a node
- ("Advanced GPU scheduling") Job uses specific GPUs and either shared or exclusive use of a node

4.3.7.2.i Binding to GPUs

PBS Professional allocates GPUs, but does not bind jobs to any particular GPU; the application itself, or the CUDA library, is responsible for the actual binding.

4.3.7.2.ii Requesting Non-specific GPUs and Exclusive Use of Node

When your site uses "basic GPU scheduling", if your job needs GPUs, but does not require specific GPUs, and can request exclusive use of GPU nodes, you can request GPUs the same way you request CPUs.

Your administrator can set up a resource to represent the GPUs on a node. We recommend that the GPU resource is called *ngpus*.

When requesting GPUs in this manner, your job should request exclusive use of the node to prevent other jobs being scheduled on its GPUs.

qsub -l select=ngpus=<value>:<rest of chunk specification> -lplace=excl

Example 4-7: To submit the job named "*my_gpu_job*", requesting one node with two GPUs and one CPU, and exclusive use of the node:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=excl my_gpu_job
```

It is up to the application or CUDA to bind the GPUs to the application processes.

4.3.7.2.iii Requesting Non-specific GPUs and Shared Use of Node

When your site uses "advanced GPU scheduling", your administrator can configure PBS to allow your job to use non-specific GPUs on a node while sharing GPU nodes. In this case, your administrator puts each GPU in its own vnode.

Your administrator can configure a resource to represent GPUs. We recommend that the GPU resource is called *ngpus*.

Your administrator can configure each GPU vnode so it has a resource containing the device number of the GPU. We recommend that this resource is called *gpu_id*.

Example 4-8: To submit the job named "*my_gpu_job*", requesting two GPUs and one CPU, and shared use of the node:

```
qsub -lselect=1:ncpus=1:ngpus=2 -lplace=shared my_gpu_job
```

When a job is submitted requesting any GPU, the PBS scheduler looks for a vnode with an available GPU and assigns that vnode to the job. Since there is a one-to-one correspondence between GPUs and vnodes, the job can determine the gpu_id of that vnode. Finally, the application can use the appropriate CUDA call to bind the process to the allocated GPU.

4.3.7.2.iv Requesting Specific GPUs

When your site uses "advanced GPU scheduling", your job can request one or more specific GPUs. This allows you to run applications on the GPUs for which the applications are written.

Your administrator can set up a resource to allow jobs to request specific GPUs. We recommend that the GPU resource is called *gpu_id*.

When you request specific GPUs, specify the GPU that you want for each chunk:

qsub -l select=gpu_id=<GPU ID>:<rest of chunk specification>

Example 4-9: To request 4 vnodes, each with GPU with ID *0*:

```
qsub -lselect=4:ncpus=1:ngpus=1:gpu_id=gpu0 my_gpu_job
```

When a job is submitted requesting specific GPUs, the PBS scheduler assigns the vnode with the resource containing that gpu_id to the job. The application can use the appropriate CUDA call to bind the process to the allocated GPU.

4.3.7.3 Viewing GPU Information for Nodes

You can find the number of GPUs available and assigned on execution hosts via the `pbsnodes` command. See section 4.6, "Viewing Resources", on page 63.

4.3.8 Caveats and Restrictions on Requesting Resources

4.3.8.1 Caveats and Restrictions for Specifying Resource Values

- Resource values which contain commas, quotes, plus signs, equal signs, colons, or parentheses must be quoted to PBS. The string must be enclosed in quotes so that the command (e.g. `qsub`, `qalter`) will parse it correctly.
- When specifying resources via the command line, any quoted strings must be escaped or enclosed in another set of quotes. This second set of quotes must be different from the first set, meaning that double quotes must be enclosed in single quotes, and vice versa.
- If a string resource value contains spaces or shell metacharacters, enclose the string in quotes, or otherwise escape the space and metacharacters. Be sure to use the correct quotes for your shell and the behavior you want.

4.3.8.2 Warning About NOT Requesting walltime

If your job does not request a walltime, and there is no default for walltime, your job is treated as if it had requested a very, very long walltime. Translation: the scheduler will have a hard time finding a time slot for your job. Remember, the administrator may schedule dedicated time for the entire PBS complex once a year, for upgrading, etc. In this case, your job will never run. We recommend requesting a reasonable walltime for your job.

4.3.8.3 Caveats for Jobs Requesting Undefined Resources

If you submit a job that requests a job-wide or host-level resource that is undefined, the job is not rejected at submission; instead, it is aborted upon being enqueued in an execution queue, if the resources are still undefined. This preserves backward compatibility.

4.3.8.4 Matching Resource Requests with Unset Resources

When job resource requests are being matched with available resources, a numerical resource that is unset on a host is treated as if it were zero, and an unset string cannot satisfy a request. An unset Boolean resource is treated as if it were set to "*False*". An unset resource at the server or queue is treated as if it were infinite.

4.3.8.5 Caveat for Invisible or Unrequestable Resources

Your administrator may define custom resources which restricted, so that they are invisible, or are visible but unrequestable. Custom resources which were created to be invisible or unrequestable cannot be requested or altered. The following is a list of the commands normally used to view or request resources or modify resource requests, and their limitations for restricted resources:

`pbsnodes`

Job submitters cannot view restricted host-level custom resources.

`pbs_rstat`

Job submitters cannot view restricted reservation resources.

`pbs_rsub`

Job submitters cannot request restricted custom resources for reservations.

`qalter`

Job submitters cannot alter a restricted resource.

`qmgr`

Job submitters cannot print or list a restricted resource.

`qselect`

Job submitters cannot specify restricted resources via `-l Resource_List.`

`qsub`

Job submitters cannot request a restricted resource.

`qstat`

Job submitters cannot view a restricted resource.

4.3.8.6 Warning About Requesting Tiny Amounts of Memory

The smallest unit of memory you can request is 1KB. If you request 400 bytes, you get 1KB. If you request 1400 bytes, you get 2KB.

4.3.8.7 Maximum Length of Job Submission Command Line

The maximum length of a command line in PBS is 4095 characters. When you submit a job using the command line, your select and place statements, and the rest of your command line, must fit within 4095 characters.

4.3.8.8 Only One select Statement Per Job

You can include at most one select statement per job submission.

4.3.8.9 The software Resource is Job-wide

The built-in resource "software" is not a vnode-level resource. See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide.

4.3.8.10 Do Not Mix Old and New Syntax

Do not mix old and new syntax when requesting resources. See section 4.8, “Backward Compatibility”, on page 70 for a description of old syntax.

4.4 How Resources are Allocated to Jobs

Resources are allocated to your job when the job explicitly requests them, and when PBS applies defaults.

Jobs explicitly request resources either at the vnode level in chunks defined in a selection statement, or in job-wide resource requests. We will cover requesting resources in section 4.3.3, “Requesting Resources in Chunks”, on page 53 and section 4.3.2, “Requesting Job-wide Resources”, on page 52.

The administrator can set default resources at the server and at queues, so that a job that does not request a resource at submission time ends up being allocated the default value for that resource. We will cover default resources in section 4.4.1, “Applying Default Resources”, on page 59.

The administrator can also specify default arguments for `qsub` so that jobs automatically request certain resources. Resource values explicitly requested by your job override any `qsub` defaults. See “qsub” on page 213 of the PBS Professional Reference Guide.

4.4.1 Applying Default Resources

PBS applies resource defaults only where the job has not explicitly requested a value for a resource.

Job-wide and per-chunk resources are applied, with the following order of precedence, via the following:

1. Resources that are explicitly requested via `-l <resource>=<value>` and `-l select=<chunk>`
2. Default `qsub` arguments
3. The queue's default_chunk.<resource>
4. The server's default_chunk.<resource>
5. The queue's resources_default.<resource>
6. The server's resources_default.<resource>
7. The queue's resources_max.<resource>
8. The server's resources_max.<resource>

4.4.1.1 Applying Job-wide Default Resources

The explicit job-wide resource request is checked first against default `qsub` arguments, then against queue resource defaults, then against server resource defaults. Any default job-wide resources not already in the job's resource request are added. PBS applies job-wide default resources defined in the following places, in this order:

- Via `qsub`: The server's default_qsub_arguments attribute can include any requestable job-wide resources.
- Via the queue: Each queue's resources_default attribute defines each queue-level job-wide resource default in resources_default.<resource>.
- Via the server: The server's resources_default attribute defines each server-level job-wide resource default in resources_default.<resource>.

4.4.1.2 Applying Per-chunk Default Resources

For each chunk in the job's selection statement, first qsub defaults are applied, then queue chunk defaults are applied, then server chunk defaults are applied. If the chunk request does not include a resource listed in the defaults, the default is added. PBS applies default chunk resources in the following order:

- Via qsub: The server's default_qsub_arguments attribute can include any requestable chunk resources.
- Via the queue: Each queue's default_chunk attribute defines each queue-level chunk resource default in default_chunk.<resource>.
- Via the server: The server's default_chunk attribute defines each server-level chunk resource default in default_chunk.<resource>.

Example 4-10: Applying chunk defaults: if the queue in which the job is enqueued has the following defaults defined:

```
default_chunk.ncpus=1
default_chunk.mem=2gb
```

A job submitted with this selection statement:

```
select=2:ncpus=4+1:mem=9gb
```

The job has this specification after the default_chunk elements are applied:

select=2:ncpus=4:*mem=2gb*+1:*ncpus=1*:mem=9gb.

In this example, *mem=2gb* and *ncpus=1* are inherited from default_chunk.

4.4.1.3 Caveat for Moving Jobs From One Queue to Another

If the job is moved from the current queue to a new queue, any default resources in the job's resource list that were contributed by the current queue are removed. This includes a select specification and place directive generated by the rules for conversion from the old syntax. If a job's resource is unset (undefined) and there exists a default value at the new queue or server, that default value is applied to the job's resource list. If either select or place is missing from the job's new resource list, it will be automatically generated, using any newly inherited default values.

Given the following set of queue and server default values:

Server

```
resources_default.ncpus=1
```

Queue QA

```
resources_default.ncpus=2
default_chunk.mem=2gb
```

Queue QB

```
default_chunk.mem=1gb
```

no default for ncpus

The following examples illustrate the equivalent select specification for jobs submitted into queue QA and then moved to (or submitted directly to) queue QB:

qsub -l ncpus=1 -lmem=4gb

In QA: select=1:ncpus=1:mem=4gb

No defaults need be applied

In QB: select=1:ncpus=1:mem=4gb

No defaults need be applied

qsub -l ncpus=1

In QA: select=1:ncpus=1:mem=2gb

Picks up 2gb from queue default chunk and 1 ncpus from qsub

In QB: `select=1:ncpus=1:mem=1gb`

Picks up 1gb from queue default chunk and 1 ncpus from qsub

`qsub -lmem=4gb`

In QA: select=1:ncpus=2:mem=4gb

Picks up 2 ncpus from queue level job-wide resource default and 4gb mem from qsub

In QB: `select=1:ncpus=1:mem=4gb`

Picks up 1 ncpus from server level job-wide default and 4gb mem from qsub

`qsub -lnodes=4`

In QA: `select=4:ncpus=1:mem=2gb`

Picks up a queue level default memory chunk of 2gb. (This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=1gb`

(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

`qsub -l mem=16gb -lnodes=4`

In QA: `select=4:ncpus=1:mem=4gb`

(This is not 4:ncpus=2 because in prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated.)

In QB: `select=4:ncpus=1:mem=4gb`

(In prior versions, "nodes=x" implied 1 CPU per node unless otherwise explicitly stated, so the ncpus=1 is not inherited from the server default.)

4.5 Limits on Resource Usage

Jobs are assigned limits on the amount of resources they can use. These limits apply to how much the whole job can use (job-wide limit) and to how much the job can use at each host (host limit). Limits are applied only to resources the job requests or inherits.

Your administrator can configure PBS to enforce limits on mem and ncpus, but the other limits are always enforced.

If you want to make sure that your job does not exceed a given amount of some resource, request that amount of the resource.

4.5.1 Enforceable Resource Limits

Limits can be enforced on the following resources:

Table 4-2: Enforceable Resource Limits

Resource Name	Where Specified	Where Enforced	Always Enforced?
cput	Host	Host	Always
mem	Host	Host	Optional
ncpus	Host	Host	Optional

Table 4-2: Enforceable Resource Limits

Resource Name	Where Specified	Where Enforced	Always Enforced?
pcput	Job-wide	Per-process	Always
pmem	Job-wide	Per-process	Always
pvmem	Job-wide	Per-process	Always
vmem	Host	Host	Always
walltime	Job-wide	Job-wide	Always

4.5.2 Origins of Resource Limits

Limits are derived from both requested resources and applied default resources. Resource limits are derived in the order shown in section 4.4.1, “Applying Default Resources”, on page 59.

4.5.3 Job-wide Resource Limits

Job-wide resource limits set a limit for per-job resource usage. Job resource limits are derived from job-wide resources and from totals of per-chunk consumable resources. Limits are derived from explicitly requested resources and default resources.

Job-wide resource limits that are derived from from sums of all chunks override those that are derived from job-wide resources.

Example 4-11: Job-wide limits are derived from sums of chunks. With the following chunk request:

```
qsub -lselect=2:ncpus=3:mem=4gb:arch=linux
```

The following job-wide limits are derived:

ncpus=6

mem=8gb

4.5.4 Per-chunk Resource Limits

Each chunk's per-chunk limits determine how much of any resource can be used at that host. PBS sums the chunk limits at each host, and uses that sum as the limit at that resource. Per-chunk resource usage limits are the amount of per-chunk resources allocated to the job, both from explicit requests and from defaults.

4.5.4.1 Effects of Limits

If a running job exceeds its limit for walltime, the job is terminated.

If any of the job's processes exceed the limit for pcput, pmem, or pvmem, the job is terminated.

If any of the host limits for mem, ncpus, cput, or vmem is exceeded, the job is terminated. These are host-level limits, so if for example your job has two chunks on one host, and the processes on one chunk exceed one of these limits, but the processes on the other are under the chunk limit, the job can continue to run as long as the total used for both chunks is less than the host limit.

4.5.5 Examples of Memory Limits

Your administrator may choose to enforce memory limits. If this is the case, the memory used by the entire job cannot exceed the amount in Resource_List.mem, and the memory used at any host cannot exceed the sum of the chunks on that host. For the following examples, assume the following:

The queue has these settings:

```
resources_default.mem=200mb
default_chunk.mem=100mb
```

Example 4-12: A job requesting `-l select=2:ncpus=1:mem=345mb` uses 345mb from each of two vnodes and has a job-wide limit of 690mb (2 * 345). The job's Resource_List.mem shows *690mb*.

Example 4-13: A job requesting `-l select=2:ncpus=2` takes 100mb via default_chunk from each vnode and has a job-wide limit of 200mb (2 * 100mb). The job's Resource_List.mem shows *200mb*.

Example 4-14: A job requesting `-l ncpus=2` takes 200mb (inherited from resources_default and used to create the select specification) from one vnode and has a job-wide limit of 200mb. The job's Resource_List.mem shows *200mb*.

Example 4-15: A job requesting `-lnodes=2` inherits 200mb from resources_default.mem which becomes the job-wide limit. The memory is taken from the two vnodes, half (100mb) from each. The generated select specification is `2:ncpus=1:mem=100mb`. The job's Resource_List.mem shows *200mb*.

4.6 Viewing Resources

You can look at the resources on the server, queue, and vnodes. You can also see what resources are allocated to and used by your job.

4.6.1 Viewing Server, Queue, and Vnode Resources

To see server resources:

```
qstat - Bf
```

To see queue resources:

```
qstat -Qf
```

To see vnode resources, use any of the following:

```
qmgr -c "list node <vnode name> <attribute name>"
pbsnodes -av
pbsnodes [<host list>]
```

Look at the following attributes:

resources_available.<resource name>

(Server, queue, vnode) Total amount of the resource available at the server, queue, or vnode; does not take into account how much of the resource is in use.

resources_default.<resource name>

(Server, queue) Default value for job-wide resource. This amount is allocated to job if job does not request this resource. Queue setting overrides server setting.

resources_max.<resource name>

(Server, queue) Maximum amount that a single job can request. Queue setting overrides server setting.

resources_min.<resource name>

(Queue) Minimum amount that a single job can request.

resources_assigned.<resource name>

(Server, queue, vnode) Total amount of the resource that has been allocated to running and exiting jobs and reservations at the server, queue, or vnode.

4.6.2 Viewing Job Resources

To see the resources allocated to or used by your job:

```
qstat -f
```

Look at the following job attributes:

Resource_List.<resource name>

The amount of the resource that has been allocated to the job, including defaults.

resources_used.<resource name>

The amount of the resource used by the job.

4.6.2.1 Resources Shown in Resource_List Job Attribute

When your job requests a job-wide resource or any of certain built-in host-level resources, the value requested is stored in the job's Resource_List attribute, as Resource_List.<resource name>=<value>. When you request a built-in host-level resource inside multiple chunks, the value in Resource_List is the sum over all of the chunks for that resource. For a list of the resources that can appear in Resource_List, see section 5.9.2, "Resources Requested by Job", on page 245 of the PBS Professional Administrator's Guide.

If your administrator has defined default values for any of those resources, and your job has inherited any defaults, those defaults control the value shown in the Resource_List attribute.

4.7 Specifying Job Placement

You can specify how your job should be placed on vnodes. You can choose to place each chunk on a different host, or a different vnode, or your job can use chunks that are all on one host. You can specify that all of the job's chunks should share a value for some resource.

Your job can request exclusive use of each vnode, or shared use with other jobs. Your job can request exclusive use of its hosts.

We will cover the basics of specifying job placement in the following sections. For details on placing chunks for an MPI job, see "Submitting Multiprocessor Jobs".

4.7.1 Using the place Statement

You use the *place* statement to specify how the job's chunks are placed.

The place statement can contain the following elements in any order:

-l place=[<arrangement>][: <sharing>][: <grouping>]

where

arrangement

Whether this chunk is willing to share this vnode or host with other chunks from the same job. One of *free* | *pack* | *scatter* | *vscatter*

sharing

Whether this this chunk is willing to share this vnode or host with other jobs. One of *excl* | *shared* | *exclhost*

grouping

Whether the chunks from this job should be placed on vnodes that all have the same value for a resource. Can have only one instance of *group=<resource name>*

and where

Table 4-3: Placement Modifiers

Modifier	Meaning
free	Place job on any vnode(s)
pack	All chunks will be taken from one host
scatter	Only one chunk is taken from any host
vscatter	Only one chunk is taken from any vnode. Each chunk must fit on a vnode.
excl	Only this job uses the vnodes chosen
exclhost	The entire host is allocated to this job
shared	This job can share the vnodes chosen
group=<resource>	Chunks will be placed on vnodes according to a resource shared by those vnodes. This resource must be a string or string array. All vnodes in the group must have a common value for the resource.

The place statement may be not be used without the select statement.

The place statement may not begin with a colon.

4.7.1.1 Specifying Arrangement of Chunks

To place your job's chunks wherever they fit:

```
-l place=free
```

To place all of the job's chunks on a single host:

```
-l place=pack
```

To place each chunk on its own host:

```
-l place=scatter
```

To place each chunk on its own vnode:

```
-l place=vscatter
```

4.7.1.1.i Caveats and Restrictions for Arrangement

- For all arrangements except *vscatter*, chunks cannot be split across hosts, but they can be split across vnodes on the same host. If a job does not request *vscatter* for its arrangement, any chunk can be broken across vnodes. This means that one chunk could be taken from more than one vnode.
- If the job requests *vscatter* for its arrangement, no chunk can be larger than a vnode, and no chunk can be split across vnodes. This behavior is different from other values for arrangement, where chunks can be split across vnodes.

4.7.1.2 Specifying Shared or Exclusive Use of Vnodes

Each vnode can be allocated exclusively to one job, or its resources can be shared among jobs. Hosts can also be allocated exclusively to one job, or shared among jobs.

How vnodes are allocated to jobs is determined by a combination of the vnode's sharing attribute and the job's resource request. The possible values for the vnode sharing attribute, and how they interact with a job's placement request, are described in "sharing" on page 324 of the PBS Professional Reference Guide. The following table expands on this:

Table 4-4: How Vnode sharing Attribute Affects Vnode Allocation

Value of Vnode sharing Attribute	Effect on Allocation
not set	The job's arrangement request determines how vnodes are allocated to the job. If there is no specification, vnodes are shared.
default_share	Vnodes are shared unless the job explicitly requests exclusive use of the vnodes.
default_excl	Vnodes are allocated exclusively to the job unless the job explicitly requests shared allocation.
default_exclhost	All vnodes from this host are allocated exclusively to the job, unless the job explicitly requests shared allocation.
ignore_excl	Vnodes are shared, regardless of the job's request.
force_excl	Vnodes are allocated exclusively to the job, regardless of the job's request.
force_exclhost	All vnodes from this host are allocated exclusively to the job, regardless of the job's request.

If a vnode is allocated exclusively to a job, all of its resources are assigned to the job. The state of the vnode becomes *job-exclusive*. No other job can use the vnode.

If a host is to be allocated exclusively to one job, all of the host must be used: if any vnode from a host has its sharing attribute set to either *default_exclhost* or *force_exclhost*, all vnodes on that host must have the same value for the sharing attribute.

If your job requests exclusive placement, and it is in a reservation, the reservation must also request exclusive placement via `-l place=excl`.

To see the value for a vnode's sharing attribute, you can do either of the following:

- Use `qmgr`:
  `Qmgr: list node <vnode name> sharing`
- Use `pbsnodes`:
  `pbsnodes -av`

4.7.1.3 Grouping on a Resource

You can specify that all of the chunks for your job should run on vnodes that have the same value for a selected resource.

To group your job's chunks this way, use the following format:

-l place=group=<resource name>

where *resource name* is a string or string array.

The value of the resource can be one or more strings at each vnode, but there must be one string that is the same for each vnode. For example, if the resource is *router*, the value can be "*r1i0,r1*" at one vnode, and "*r1i1,r1*" at another vnode, and these vnodes can be grouped because they share the string "*r1*".

Using the method of grouping on a resource, you cannot specify what the value of the resource should be, only that all vnodes have the same value. If you need the resource to have a specific value, specify that value in the description of the chunks.

4.7.1.3.i Grouping vs. Placement Sets

Your administrator may define placement sets for your site. A placement set is a group of vnodes that share a value for a resource. By default, placement sets attempt to group vnodes that are "close to" each other. If your job doesn't request a specific placement, and placement sets are defined, your job may automatically run in a placement set. See "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

If your job requests grouping by a resource, using *place=group=resource*, the chunks are placed as requested and placement sets are ignored.

If your job requests grouping but no group contains the required number of vnodes, grouping is ignored.

4.7.2 How the Job Gets its Place Statement

If the administrator has defined default values for arrangement, sharing, and grouping, each job inherits these unless it explicitly requests at least one. That means that if your job requests arrangement, but not sharing or grouping, it will not inherit values for sharing or grouping. For example, the administrator sets a default of `place=pack:exclhost:group=host`. Job A requests `place=free`, but doesn't specify sharing or grouping, so Job A does not inherit sharing or grouping. Job B does not request any placement, so it inherits all three.

The place statement can be specified, in order of precedence, via:

1. Explicit placement request in `qalter`
2. Explicit placement request in `qsub`
3. Explicit placement request in PBS job script directives
4. Default `qsub` place statement
5. Queue default placement rules
6. Server default placement rules
7. Built-in default conversion and placement rules

4.7.3 Caveats and Restrictions for Specifying Placement

- The place specification cannot be used without the select specification. In other words, you can only specify placement when you have specified chunks.
- A select specification cannot be used with a nodes specification.
- A select specification cannot be used with old-style resource requests such as -lncpus, -lmem, -lvmem, -larch, -lhost.
- When using *place=group=<resource>*, the resource must be a string or string array.
- Do not mix old and new syntax when requesting placement. See section 4.8, "Backward Compatibility", on page 70 for a description of old syntax.
- If your job requests exclusive placement, and it is in a reservation, the reservation must also request exclusive placement via `-l place=excl`.

4.7.4 Examples of Specifying Placement

Unless otherwise specified, the vnodes allocated to the job will be allocated as shared or exclusive based on the setting of the vnode's sharing attribute. Each of the following shows how you would use -l select= and -l place=.

1. A job that will fit in a single host but not in any of the vnodes, packed into the fewest vnodes:

   ```
   -l select=1:ncpus=10:mem=20gb
   -l place=pack
   ```

   In earlier versions, this would have been:

   ```
   -lncpus=10,mem=20gb
   ```

2. Request four chunks, each with 1 CPU and 4GB of memory taken from anywhere.

   ```
   -l select=4:ncpus=1:mem=4GB
   -l place=free
   ```

3. Allocate 4 chunks, each with 1 CPU and 2GB of memory from between

one and four vnodes which have an arch of "linux".

```
-l select=4:ncpus=1:mem=2GB:arch=linux -l place=free
```

4. Allocate four chunks on 1 to 4 vnodes where each vnode must have 1 CPU, 3GB of memory and 1 node-locked dyna license available for each chunk.

```
-l select=4:dyna=1:ncpus=1:mem=3GB -l place=free
```

5. Allocate four chunks on 1 to 4 vnodes, and 4 floating dyna licenses. This assumes "dyna" is specified as a server dynamic resource.

```
-l dyna=4 -l select=4:ncpus=1:mem=3GB -l place=free
```

6. This selects exactly 4 vnodes where the arch is linux, and each vnode will be on a separate host. Each vnode will have 1 CPU and 2GB of memory allocated to the job.

```
-lselect=4:mem=2GB:ncpus=1:arch=linux -lplace=scatter
```

7. This will allocate 3 chunks, each with 1 CPU and 10GB of memory. This will also reserve 100mb of scratch space if scratch is to be accounted . Scratch is assumed to be on a file system common to all hosts. The value of "place" depends on the default which is "place=free".

```
-l scratch=100mb -l select=3:ncpus=1:mem=10GB
```

8. This will allocate 2 CPUs and 50GB of memory on a host named zooland. The value of "place" depends on the default which defaults to "place=free":

```
-l select=1:ncpus=2:mem=50gb:host=zooland
```

9. This will allocate 1 CPU and 6GB of memory and one host-locked swlicense from each of two hosts:

```
-l select=2:ncpus=1:mem=6gb:swlicense=1
-lplace=scatter
```

10. Request free placement of 10 CPUs across hosts:

```
-l select=10:ncpus=1
-l place=free
```

11. Here is an odd-sized job that will fit on a single HPE system, but not on any one node-board. We request an odd number of CPUs that are not shared, so they must be "rounded up":

```
-l select=1:ncpus=3:mem=6gb
-l place=pack:excl
```

12. Here is an odd-sized job that will fit on a single HPE system, but not on any one node-board. We are asking for small number of CPUs but a large amount of memory:

```
-l select=1:ncpus=1:mem=25gb
-l place=pack:excl
```

13. Here is a job that may be run across multiple HPE systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
-l place=free
```

14. Submit a job that must be run across multiple HPE systems, packed into the fewest vnodes:

```
-l select=2:ncpus=10:mem=12gb
-l place=scatter
```

15. Request free placement across nodeboards within a single host:

```
-l select=1:ncpus=10:mem=10gb
```

```
-l place=group=host
```

16. Request free placement across vnodes on multiple HPE systems:

```
-l select=10:ncpus=1:mem=1gb
-l place=free
```

17. Here is a small job that uses a shared cpuset:

```
-l select=1:ncpus=1:mem=512kb
-l place=pack:shared
```

18. Request a special resource available on a limited set of nodeboards, such as a graphics card:

```
-l select= 1:ncpus=2:mem=2gb:graphics=True + 1:ncpus=20:mem=20gb:graphics=False
-l place=pack:excl
```

19. Align SMP jobs on c-brick boundaries:

```
-l select=1:ncpus=4:mem=6gb
-l place=pack:group=cbrick
```

20. Align a large job within one router, if it fits within a router:

```
-l select=1:ncpus=100:mem=200gb
-l place=pack:group=router
```

21. Fit large jobs that do not fit within a single router into as few available routers as possible. Here, RES is the resource used for node grouping:

```
-l select=1:ncpus=300:mem=300gb
-l place=pack:group=<RES>
```

22. To submit an MPI job, specify one chunk per MPI task. For a 10-way MPI job with 2gb of memory per MPI task:

```
-l select=10:ncpus=1:mem=2gb
```

23. To submit a non-MPI job (including a 1-CPU job or an OpenMP or shared memory) job, use a single chunk. For a 2-CPU job requiring 10gb of memory:

```
-l  select=1:ncpus=2:mem=10gb
```

4.8 Backward Compatibility

4.8.1 Old-style Resource Specifications

Old versions of PBS allowed job submitters to ask for resources outside of a select statement, using "`-lre-source=value`", where those resources must now be requested in chunks, inside a select statement. This old style of resource request was called a "resource specification". Resource specification syntax is **deprecated**.

For backward compatibility, any resource specification is converted to select and place statements, and any defaults are applied.

4.8.2 Old-style Node Specifications

In early versions of PBS, job submitters used "`-l nodes=...`" in what was called a "node specification" to specify where the job should run. The syntax for a "node specification" is **deprecated**.

For backward compatibility, a legal node specification or resource specification is converted into select and place directives; we show how in following sections.

4.8.3 Conversion of Old Style to New

4.8.3.1 Conversion of Resource Specifications

If your job has an old-style resource specification, PBS creates a select specification requesting 1 chunk containing the resources specified by the job and server and/or queue default resources. Resource specification format:

-l<resource>=<value>[:<resource>=<value> ...]

The resource specification is converted to:

-lselect=1[:<*resource*>=<*value*> ...]

-lplace=pack

with one instance of *resource=value* for each of the following vnode-level resources in the resource request:

built-in resources: ncpus | mem | vmem | arch | host

site-defined vnode-level resources

For example, a job submitted with

qsub -l ncpus=4:mem=123mb:arch=linux

gets the following select statement:

select=1:ncpus=4:mem=123mb:arch=linux

4.8.3.2 Conversion of Node Specifications

If your job requests a node specification, PBS creates a select and place specification, according to the following rules.

Old node specification format:

-lnodes=[N:<*spec list*> | <*spec list*>]

[[+N:<*spec list*> | +<*spec list*>] ...]

[#<s*uffix*> ...][-lncpus=Z]

where:

spec list has syntax: <*spec*>[:<*spec*> ...]

spec is any of: hostname | property | ncpus=X | cpp=X | ppn=P

suffix is any of: property | excl | shared

N and P are positive integers

X and Z are non-negative integers

The node specification is converted into select and place statements as follows:

Each *spec list* is converted into one chunk, so that N:<*spec list*> is converted into N chunks.

If *spec* is hostname :

The chunk will include *host=hostname*

If *spec* matches any vnode's resources_available.<hostname> value:

The chunk will include *host=hostname*

If *spec* is property :

The chunk will include <*property*>=*true*

Property must be a site-defined vnode-level boolean resource.

If *spec* is ncpus=X or cpp=X :

The chunk will include ncpus=X

If no *spec* is ncpus=X and no spec is cpp=X :

The chunk will include ncpus=P

If *spec* is ppn=P :

The chunk will include mpiprocs=P

If the *nodespec* is

```
-lnodes=N:ppn=P
```

It is converted to

```
-lselect=N:ncpus=P:mpiprocs=P
```

Example:

```
-lnodes=4:ppn=2
```

is converted into

```
-lselect=4:ncpus=2:mpiprocs=2
```

If -lncpus=Z is specified and no spec contains ncpus=X and no spec is cpp=X :

Every chunk will include ncpus=W, where W is Z divided by the total number of chunks. (Note: W must be an integer; Z must be evenly divisible by the number of chunks.)

If property is a suffix :

All chunks will include property=true

If excl is a suffix :

The placement directive will be -lplace=scatter:excl

If shared is a suffix :

The placement directive will be -lplace=scatter:shared

If neither excl nor shared is a suffix :

The placement directive will be -lplace=scatter

Example:

```
-lnodes=3:green:ncpus=2:ppn=2+2:red
```

is converted to:

```
-l select=3:green=true:ncpus=4:mpiprocs=2+ 2:red=true:ncpus=1
-l place=scatter
```

4.8.3.3 Examples of Converting Old Syntax to New

1. Request CPUs and memory on a single host using old syntax:
   ```
   -l ncpus=5,mem=10gb
   ```

is converted into the equivalent:

```
-l select=1:ncpus=5:mem=10gb
-l place=pack
```

2. Request CPUs and memory on a named host along with custom resources including a floating license using old syntax:

```
-l ncpus=1,mem=5mb,host=sunny,opti=1,arch=arch1
```

is converted to the equivalent:

```
-l select=1:ncpus=1:mem=5gb:host=sunny:arch=arch1
-l place=pack
-l opti=1
```

3. Request one host with a certain property using old syntax:

```
-lnodes=1:property
```

is converted to the equivalent:

```
-l select=1:ncpus=1:property=True
-l place=scatter
```

4. Request 2 CPUs on each of four hosts with a given property using old syntax:

```
-lnodes=4:property:ncpus=2
```

is converted to the equivalent:

```
-l select=4: ncpus=2:property=True -l place=scatter
```

5. Request 1 CPU on each of 14 hosts asking for certain software, licenses and a job limit amount of memory using old syntax:

```
-lnodes=14:mpi-fluent:ncpus=1 -lfluent=1,fluent-all=1, fluent-par=13
-l mem=280mb
```

is converted to the equivalent:

```
-l select=14:ncpus=1:mem=20mb:mpi_fluent=True
-l place=scatter
-l fluent=1,fluent-all=1,fluent-par=13
```

6. Requesting licenses using old syntax:

```
-lnodes=3:dyna-mpi-Linux:ncpus=2 -ldyna=6,mem=100mb, software=dyna
```

is converted to the equivalent:

```
-l select=3:ncpus=2:mem=33mb: dyna-mpi-Linux=True
-l place=scatter
-l software=dyna
-l dyna=6
```

7. Requesting licenses using old syntax:

```
-l ncpus=2,app_lic=6,mem=200mb -l software=app
```

is converted to the equivalent:

```
-l select=1:ncpus=2:mem=200mb
-l place=pack
-l software=app
-l app_lic=6
```

8. Additional example using old syntax:

   ```
   -lnodes=1:fserver+15:noserver
   ```

   is converted to the equivalent:

   ```
   -l select=1:ncpus=1:fserver=True + 15:ncpus=1:noserver=True
   -l place=scatter
   ```

   but could also be more easily specified with something like:

   ```
   -l select=1:ncpus=1:fserver=True + 15:ncpus=1:fserver=False
   -l place=scatter
   ```

9. Allocate 4 vnodes, each with 6 CPUs with 3 MPI processes per vnode, with each vnode on a separate host. The memory allocated would be one-fourth of the memory specified by the queue or server default if one existed. This results in a different placement of the job from version 5.4:

   ```
   -lnodes=4:ppn=3:ncpus=2
   ```

   is converted to:

   ```
   -l select=4:ncpus=6:mpiprocs=3 -l place=scatter
   ```

10. Allocate 4 vnodes, from 4 separate hosts, with the property blue. The amount of memory allocated from each vnode is 2560MB (= 10GB / 4) rather than 10GB from each vnode.

    ```
    -lnodes=4:blue:ncpus=2 -l mem=10GB
    ```

    is converted to:

    ```
    -l select=4:blue=True:ncpus=2:mem=2560mb -lplace=scatter
    ```

4.8.4 Caveats for Using Old Syntax

4.8.4.1 Changes in Behavior

Most jobs submitted with "`-lnodes`" will continue to work as expected. These jobs will be automatically converted to the new syntax. However, job tasks may execute in an unexpected order, because vnodes may be assigned in a different order. Jobs submitted with old syntax that ran successfully on versions of PBS Professional prior to 8.0 can fail because a limit that was per-chunk is now job-wide.

Example 4-16: A job submitted using `-lnodes=X -lmem=M` that fails because the mem limit is now job-wide. If the following conditions are true:

- PBS Professional 9.0 or later using standard MPICH
- The job is submitted with `qsub -lnodes=5 -lmem=10GB`
- The master process of this job tries to use more than 2GB

The job is killed, where in <= 7.0 the master process could use 10GB before being killed. 10GB is now a job-wide limit, divided up into a 2GB limit per chunk.

4.8.4.2 Do Not Mix Old and New Styles

Do not mix old style resource or node specifications ("`-l<resource>=<value>`" or "`-lnodes`") with select and place statements ("`-lselect=`" or "`-lplace=`"). Do not use both in the command line. Do not use both in the job script. Do not use one in a job script and the other on the command line. This will result in an error.

4.8.4.3 Resource Request Conversion Dependent on Where Resources are Defined

A job's resource request is converted from old-style to new according to various rules, one of which is that the conversion is dependent upon where resources are defined. For example: The boolean resource "Red" is defined on the server, and the boolean resource "Blue" is defined at the host level. A job requests "`qsub -l Blue=true`". This looks like an old-style resource request, and PBS checks to see where Blue is defined. Since Blue is defined at the host level, the request is converted into "`-l select=1:Blue=true`". However, if a job requests "`qsub -l Red=true`", while this looks like an old-style resource request, PBS does not convert it to a chunk request because Red is defined at the server.

4.8.4.4 Properties are Deprecated

The syntax for requesting properties is **deprecated**. Your administrator has replaced properties with Booleans.

4.8.4.5 Replace cpp with ncpus

Specifying "cpp" is part of the old syntax, and should be replaced with "ncpus".

4.8.4.6 Environment Variables Set During Conversion

1. When a node specification is converted into a select statement, the job has the environment variables NCPUS and OMP_NUM_THREADS set to the old value of ncpus in the first piece of the old node specification. This may produce incompatibilities with prior versions when a complex node specification using different values of ncpus and ppn in different pieces is converted.

5 Multiprocessor Jobs

5.1 Submitting Multiprocessor Jobs

Before you read this chapter, please read Chapter 4, "Allocating Resources & Placing Jobs", on page 49.

5.1.1 Assigning the Chunks You Want

PBS assigns chunks to job processes in the order in which the chunks appear in the select statement. PBS takes the first chunk from the primary execution host; this is where the top task of the job runs.

Example 5-1: You want three chunks, where the first has two CPUs and 20 GB of memory, the second has four CPUs and 100 GB of memory, and the third has one CPU and five GB of memory:

```
-lselect=1:ncpus=2:mem=20gb+ncpus=4:mem=100gb+mem=5gb
```

5.1.1.1 Specifying Primary Execution Host

The job's primary execution host is the host that supplies the vnode to satisfy the first chunk requested by the job.

5.1.1.2 Request Most Specific Chunks First

Chunk requests are interpreted from left to right. The more specific the chunk, the earlier it should be in the order. For example, if you require a specific host for chunk A, but chunk B is not host-specific, request Chunk A first.

5.1.2 The Job Node File

For each job, PBS creates a job-specific "host file" or "node file", which is a text file containing the name(s) of the host(s) containing the vnode(s) allocated to that job. The file is set on all execution hosts assigned to the job.

5.1.2.1 Node File Format and Contents

The node file contains a list of host names, one per line. The name of the host is the value in resources_available.host of the allocated vnode(s). The order in which hosts appear in the PBS node file is the order in which chunks are specified in the selection directive.

The node file contains one line per MPI process with the name of the host on which that process should execute. The number of MPI processes for a job, and the contents of the node file, are controlled by the value of the resource mpiprocs. mpiprocs is the number of MPI processes per chunk, and defaults to *1* where the chunk contains CPUs, *0* otherwise.

For each chunk requesting mpiprocs=*M*, the name of the host from which that chunk is allocated is written in the node file *M* times. Therefore the number of lines in the node file is the sum of requested mpiprocs for all chunks requested by the job.

Example 5-2: Two MPI processes run on HostA and one MPI process runs on HostB. The node file looks like this:

```
HostA
HostA
HostB
```

5.1.2.2 Name and Location of Node File

The file is created each execution host assigned to the job, in `PBS_HOME/aux/JOB_ID`, where *JOB_ID* is the job identifier for that job.

The full path and name for the node file is set in the job's environment, in the environment variable PBS_NODEFILE.

5.1.2.3 Node File for Old-style Requests

For jobs which request resources using the old *-lnodes=nodespec* format, the host for each vnode allocated to the job is listed *N* times, where *N* is the number of MPI ranks on the vnode. The number of MPI ranks is specified via the ppn resource.

Example 5-3: Request four vnodes, each with two MPI processes, where each process has three threads, and each thread has a CPU:

```
qsub -lnodes=4:ncpus=3:ppn=2
```

This results in each of the four hosts being written twice, in the order in which the vnodes are assigned to the job.

5.1.2.4 Using and Modifying the Node File

You can use $PBS_NODEFILE in your job script.

You can modify the node file. You can remove entries or sort the entries.

5.1.2.5 Node File Caveats

Do not add entries for new hosts; PBS may terminate processes on those hosts because PBS does not expect the processes to be running there. Adding entries on the same host may cause the job to be terminated because it is using more CPUs than it requested.

5.1.2.6 Viewing Execution Hosts

You can see which host is the primary execution host: the primary execution host is the first host listed in the job's node file.

5.1.3 Specifying Number of MPI Processes Per Chunk

How you request chunks matters. First, the number of MPI processes per chunk defaults to *1* for chunks with CPUs, and *0* for chunks without CPUs, unless you specify this value using the mpiprocs resource. Second, you can specify whether MPI processes share CPUs. For example, requesting one chunk with four CPUs and four MPI processes is not the same as requesting four chunks each with one CPU and one MPI process. In the first case, all four MPI processes are sharing all four CPUs. In the second case, each process gets its own CPU.

You request the number of MPI processes you want for each chunk using the mpiprocs resource. For example, to request two MPI processes for each of four chunks, where each chunk has two CPUs:

```
-lselect=4:ncpus=2:mpiprocs=2
```

If you don't explicitly request a value for the mpiprocs resource, it defaults to *1* for each chunk requesting CPUs, and *0* for chunks not requesting CPUs.

Example 5-4: To request one chunk with two MPI processes and one chunk with one MPI process, where both chunks have two CPUs:

```
-lselect=ncpus=2:mpiprocs=2+ncpus=2
```

Example 5-5: A request for three vnodes, each with one MPI process:

```
qsub -l select=3:ncpus=2
```

This results in the following node file:

```
<hostname for 1st vnode>
<hostname for 2nd vnode>
<hostname for 3rd vnode>
```

Example 5-6: If you want to run two MPI processes on each of three hosts and have the MPI processes share a single processor on each host, request the following:

```
-lselect=3:ncpus=1:mpiprocs=2
```

The node file then contains the following list:

```
hostname for VnodeA
hostname for VnodeA
hostname for VnodeB
hostname for VnodeB
hostname for VnodeC
hostname for VnodeC
```

Example 5-7: If you want three chunks, each with two CPUs and running two MPI processes, use:

```
-l select=3:ncpus=2:mpiprocs=2...
```

The node file then contains the following list:

```
hostname for VnodeA
hostname for VnodeA
hostname for VnodeB
hostname for VnodeB
hostname for VnodeC
hostname for VnodeC
```

Notice that the node file is the same as the previous example, even though the number of CPUs used is different.

Example 5-8: If you want four MPI processes, where each process has its own CPU:

```
-lselect=4:ncpus=1
```

See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide for a definitions of the mpiprocs resource.

5.1.3.1 Chunks With No MPI Processes

If you request a chunk that has no MPI processes, PBS may take that chunk from a vnode which has already supplied another chunk. You request a chunk that has no MPI processes using either of the following:

```
-lselect=1:ncpus=0
-lselect=1:ncpus=2:mpiprocs=0
```

5.1.4 Caveats and Advice for Multiprocessor Jobs

5.1.4.1 Requesting Uniform Processors

Some MPI jobs require the work on all vnodes to be at the same stage before moving to the next stage. For these applications, the work can proceed only at the pace of the slowest vnode, because faster vnodes must wait while it catches up. In this case, you may find it useful to ensure that the job's vnodes are homogeneous.

If there is a resource that identifies the architecture, type, or speed of the vnodes, you can use it to ensure that all chunks are taken from vnodes with the same value. You can either request a specific value for this resource for all chunks, or you can group vnodes according to the value of the resource. See section 4.7.1.3, “Grouping on a Resource”, on page 67.

Example 5-9: The resource that identifies the speed is named *speed*, and your job requests 16 chunks, each with two CPUs, two MPI processes, all with speed equal to *fast*:

```
-lselect=16:ncpus=2:mpiprocs=2:speed=fast
```

Example 5-10: Request 16 chunks where each chunk has two CPUs, using grouping to ensure that all chunks share the same speed. The resource that identifies the speed is named *speed*:

```
-lselect=16:ncpus=2:mpiprocs=2:place=group=speed
```

5.1.4.2 Requesting Storage on NFS Server

One of the vnodes in your complex may act as an NFS server to the rest of the vnodes, so that all vnodes have access to the storage on the NFS server.

Example 5-11: The scratch resource is shared among all the vnodes in the complex, and is requested from a central location, called the "*nfs_server*" vnode. To request two vnodes, each with two CPUs to do calculations, and one vnode with 10gb of memory and no MPI processes:

```
-l select=2:ncpus=2+1:host=nfs_server:scratch=10gb:ncpus=0
```

With this request, your job has one MPI process on each chunk containing CPUs, and no MPI processes on the memory-only chunk. The job shows up as having a chunk on the "nfs_server" host.

5.1.5 File Staging for Multiprocessor Jobs

PBS stages files to and from the primary execution host only.

5.1.6 Prologue and Epilogue

The prologue is run as root on the primary host, with the current working directory set to `PBS_HOME/mom_priv`, and with PBS_JOBDIR and TMPDIR set in its environment.

PBS runs the epilogue as root on the primary host. The epilogue is executed with its current working directory set to the job's staging and execution directory, and with PBS_JOBDIR and TMPDIR set in its environment.

5.1.7 MPI Environment Variables

NCPUS

PBS sets the NCPUS environment variable in the job's environment on the primary execution host. PBS sets NCPUS to the value of ncpus requested for the first chunk.

OMP_NUM_THREADS

PBS sets the OMP_NUM_THREADS environment variable in the job's environment on the primary execution host. PBS sets this variable to the value of ompthreads requested for the first chunk, which defaults to the value of ncpus requested for the first chunk.

5.1.8 Examples of Multiprocessor Jobs

Example 5-12: For a 10-way MPI job with 2gb of memory per MPI task:

```
qsub -l select=10:ncpus=1:mem=2gb
```

Example 5-13: If you have a cluster of small systems with for example two CPUs each, and you wish to submit an MPI job that will run on four separate hosts:

```
qsub -l select=4:ncpus=1 -l place=scatter
```

In this example, the node file contains one entry for each of the hosts allocated to the job, which is four entries.

The variables NCPUS and OMP_NUM_THREADS are set to one.

Example 5-14: If you do not care where the four MPI processes are run:

```
qsub -l select=4:ncpus=1 -l place=free
```

Here, the job runs on two, three, or four hosts depending on what is available.

For this example, the node file contains four entries. These are either four separate hosts, or three hosts, one of which is repeated once, or two hosts, etc.

NCPUS and OMP_NUM_THREADS are set to *1*, the number of CPUs allocated from the first chunk.

5.1.9 Submitting SMP Jobs

To submit an SMP job, simply request a single chunk containing all of the required CPUs and memory, and if necessary, specify the hostname. For example:

```
qsub -l select=ncpus=8:mem=20gb:host=host1
```

When the job is run, the node file will contain one entry, the name of the selected execution host.

The job will have two environment variables, NCPUS and OMP_NUM_THREADS, set to the number of CPUs allocated.

5.2 Using MPI with PBS

5.2.1 Using an Integrated MPI

Many MPIs are integrated with PBS. PBS provides tools to integrate most of them; a few MPIs supply the integration. When a job is run under an integrated MPI, PBS can track resource usage, signal job processes, and perform accounting for all processes of the job.

When a job is run under an MPI that is not integrated with PBS, PBS is limited to managing the job only on the primary vnode, so resource tracking, job signaling, and accounting happen only for the processes on the primary vnode.

The instructions that follow are for integrated MPIs. Check with your administrator to find out which MPIs are integrated at your site. If an MPI is not integrated with PBS, you use it as you would outside of PBS.

Some of the integrated MPIs have slightly different command lines. See the instructions for each MPI.

The following table lists the supported MPIs and gives links to instructions for using each MPI:

Table 5-1: List of Supported MPIs

MPI Name	Versions	Instructions for Use
HP MPI	1.08.03 2.0.0	See section 5.2.4, "HP MPI with PBS", on page 84
Intel MPI	2.0.022 3 4	See section 5.2.7, "Intel MPI 2.0.022, 3, and 4 with PBS", on page 85
Intel MPI	4.0.3 on Linux	See section 5.2.5, "Intel MPI 4.0.3 On Linux with PBS", on page 85
Intel MPI	4.0.3 on Windows	See section 5.2.6, "Intel MPI 4.0.3 On Windows with PBS", on page 85
LAM MPI	6.5.9	**Deprecated**. See section 5.2.8.2, "Using LAM 6.5.9 with PBS", on page 89
LAM MPI	7.0.6 7.1.1	**Deprecated**. See section 5.2.8.1, "Using LAM 7.x with PBS", on page 88
MPICH-P4 **Deprecated**.	1.2.5 1.2.6 1.2.7	See section 5.2.9, "MPICH-P4 with PBS", on page 89
MPICH-GM **Deprecated**.		See section 5.2.10, "MPICH-GM with PBS", on page 90
MPICH-MX **Deprecated**.		See section 5.2.11, "MPICH-MX with PBS", on page 93
MPICH2 **Deprecated**.	1.0.3 1.0.5 1.0.7 On Linux	See section 5.2.12, "MPICH2 with PBS on Linux", on page 95
MPICH2	1.4.1p1 on Windows	See section 5.2.13, "MPICH2 1.4.1p1 On Windows with PBS", on page 98
MVAPICH **Deprecated**.	1.2	See section 5.2.14, "MVAPICH with PBS", on page 98
MVAPICH2	1.8	See section 5.2.15, "MVAPICH2 with PBS", on page 99
Open MPI	1.4.x	See section 5.2.16, "Open MPI with PBS", on page 101
Platform MPI	8.0	See section 5.2.17, "Platform MPI with PBS", on page 101
HPE MPI	Any	See section 5.2.18, "HPE MPI with PBS", on page 102

5.2.1.1 Integration Caveats

- Some MPI command lines are slightly different; the differences for each are described.

5.2.1.2 Integrating an MPI on the Fly

The PBS administrator can perform the steps to integrate the supported MPIs. For non-integrated MPIs, you can integrate them on the fly. You integrate Intel MPI 4.0.3 using environment variables; see section 5.2.5, "Intel MPI 4.0.3 On Linux with PBS", on page 85. For the rest, you integrate them using the `pbs_tmrsh` command.

5.2.1.2.i Integrating an MPI on the Fly using the `pbs_tmrsh` Command

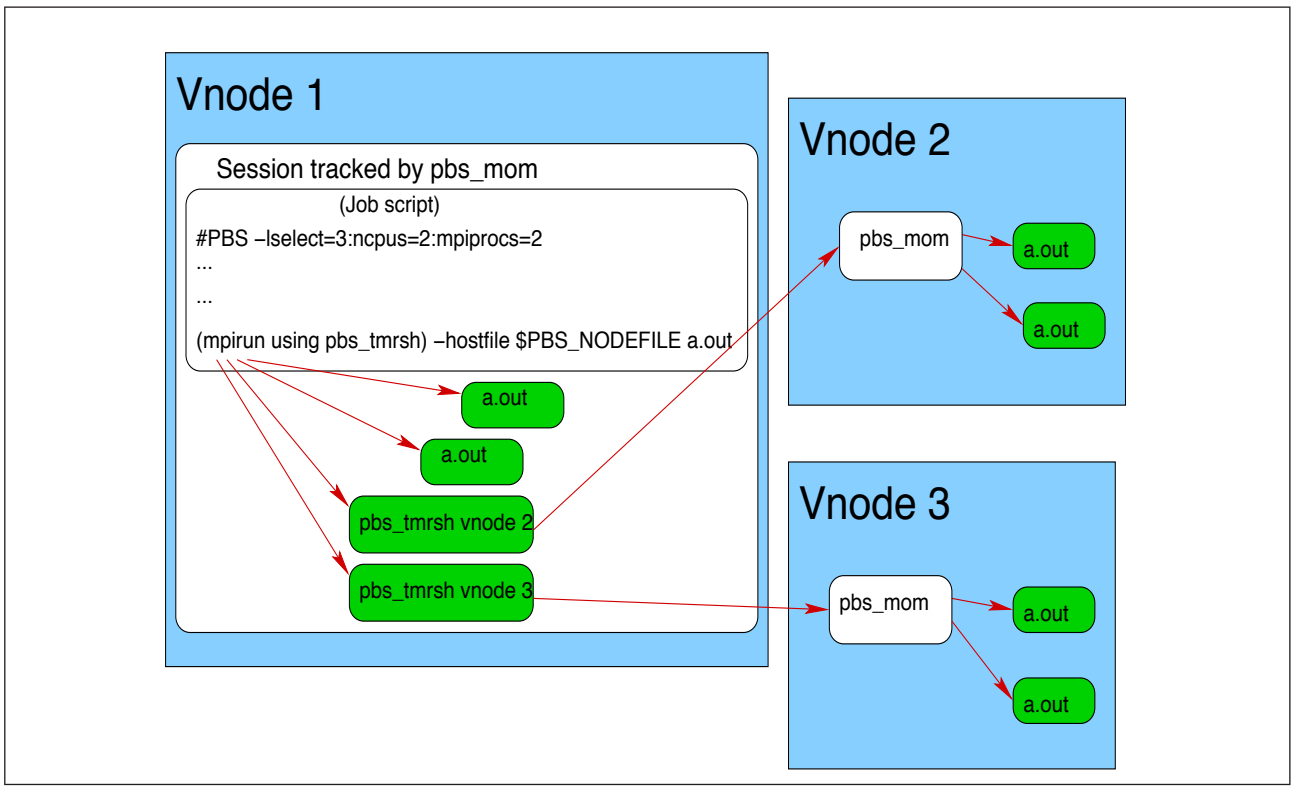
You should not use `pbs_tmrsh` with an integrated MPI or with Intel MPI 4.0.3.

This command emulates `rsh`, but uses the PBS TM interface to talk directly to `pbs_mom` on sister vnodes. The `pbs_tmrsh` command informs the primary and sister MoMs about job processes on sister vnodes. When the job uses `pbs_tmrsh`, PBS can track resource usage for all job processes.

You use `pbs_tmrsh` as your `rsh` or `ssh` command. To use `pbs_tmrsh`, set the appropriate environment variable to *pbs_tmrsh*. For example, to integrate MPICH, set the P4_RSHCOMMAND environment variable to *pbs_tmrsh*, and to integrate HP MPI, set MPI_REMSH to *pbs_tmrsh*.

The following figure illustrates how the `pbs_tmrsh` command can be used to integrate an MPI on the fly:

Figure 5-1: PBS knows about processes on vnodes 2 and 3, because `pbs_tmrsh` talks directly to `pbs_mom`, and `pbs_mom` starts the processes on vnodes 2 and 3



5.2.1.2.ii Caveats for the `pbs_tmrsh` Command

- This command cannot be used outside of a PBS job; if used outside a PBS job, this command will fail.
- The `pbs_tmrsh` command does not perform exactly like `rsh`. For example, you cannot pipe output from `pbs_tmrsh`; this will fail.

5.2.2 Prerequisites to Using MPI with PBS

The MPI that you intend to use with PBS must be working before you try to use it with PBS. You must be able to run an MPI job outside of PBS.

5.2.3 Caveats for Using MPIs

Some applications write scratch files to a temporary location. PBS makes a temporary directory available for this, and puts the path in the TMPDIR environment variable. The location of the temporary directory is host-dependent. If you are using an MPI other than LAM MPI or Open MPI, and your application needs scratch space, the temporary directory for the job should be consistent across execution hosts. Your PBS administrator can specify a root for the temporary directory on each host using the $tmpdir MoM parameter. In this case, the TMPDIR environment variable is set to the full path of the resulting temporary directory. Do not attempt to set TMPDIR.

5.2.4 HP MPI with PBS

HP MPI can be integrated with PBS on Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate HP MPI with PBS.

5.2.4.1 Setting up Your Environment for HP MPI

In order to override the default `rsh`, set PBS_RSHCOMMAND in your job script:

```
export PBS_RSHCOMMAND=<rsh choice>
```

5.2.4.2 Using HP MPI with PBS

You can run jobs under PBS using HP MPI without making any changes to your MPI command line.

5.2.4.3 Options

When running a PBS HP MPI job, you can use the same arguments to the `mpirun` command as you would outside of PBS. The following options are treated differently under PBS:

-h <host>
: Ignored

-l <user>
: Ignored

-np <number>
: Modified to fit the available resources

5.2.4.4 Caveats for HP MPI with PBS

Under the integrated HP MPI, the job's working directory is changed to your home directory.

5.2.5 Intel MPI 4.0.3 On Linux with PBS

If your PBS administrator has integrated Intel MPI 4.0.3 on Linux with PBS, you can use its `mpirun` exactly the same way inside and outside of a PBS job.

The default process manager for Intel MPI 4.0.3 on Linux is Hydra.

5.2.6 Intel MPI 4.0.3 On Windows with PBS

On Windows PBS supplies a wrapper script for Intel MPI called `pbs_intelmpi_mpirun.bat`, located in `$PBS_EXEC\bin`. You call this script instead of Intel `mpirun`. All options are passed through the script to `mpirun`.

5.2.6.1 Integrating Intel MPI 4.0.3 on the Fly

If you are using Intel MPI 4.0.3 but it has not been integrated with PBS, you can integrate it on the fly by setting environment variables:

1. Specify `rsh`:

   ```
   I_MPI_HYDRA_BOOTSTRAP=rsh
   ```

2. Specify `pbs_tmrsh`.

   a. If you are running your job entirely on hosts which have PBS_EXEC/bin in the default PATH, set this:

      ```
      I_MPI_HYDRA_BOOTSTRAP_EXEC=pbs_tmrsh
      ```

   b. If you are running your job entirely on hosts which do not have PBS_EXEC/bin in the default PATH, include the full path in the environment variable. For example:

      ```
      I_MPI_HYDRA_BOOTSTRAP_EXEC=/opt/pbs/bin/pbs_tmrsh
      ```

5.2.7 Intel MPI 2.0.022, 3, and 4 with PBS

PBS provides an interface to Intel MPI `mpirun` for these versions. If executed inside a PBS job, this allows PBS to track all Intel MPI processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard Intel MPI `mpirun` was used.

5.2.7.1 Using Intel MPI 2.0.022, 3, or 4 Integrated with PBS

You use the same `mpirun` command as you would use outside of PBS.

When submitting PBS jobs that invoke the PBS-supplied interface to `mpirun` for Intel MPI, be sure to explicitly specify the actual number of ranks or MPI tasks in the `qsub select` specification. Otherwise, jobs will fail to run with "`too few entries in the machinefile`".

For an example of this problem, specification of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

results in the following node file:

```
hostA
hostB
```

which conflicts with the "`-np 3`" specification in `mpirun` since only two MPD daemons are started.

The correct way is to specify either of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

which causes the node file to contain:

```
hostA
hostB
hostB
```

and is consistent with "`mpirun -np 3`".

5.2.7.2 Options to Integrated Intel MPI 2.0.022, 3, or 4

If executed inside a PBS job script, all of the options to the PBS interface are the same as for Intel MPI's `mpirun` except for the following:

-host, -ghost

For specifying the execution host to run on. Ignored.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

mpdboot option --totalnum=*

Ignored and replaced by the number of unique entries in $PBS_NODEFILE.

mpdboot option --file=*

Ignored and replaced by the name of $PBS_NODEFILE. The argument to this option is replaced by $PBS_NODEFILE.

Argument to `mpdboot` option `-f <mpd_hosts_file>` replaced by $PBS_NODEFILE.

-s

If the PBS interface to Intel MPI's `mpirun` is called inside a PBS job, Intel MPI's `mpirun -s` argument to `mpdboot` is not supported as this closely matches the `mpirun` option "`-s <spec>`". You can simply run a separate `mpdboot -s` before calling `mpirun`. A warning message is issued by the PBS interface upon encountering a `-s` option describing the supported form.

-np

If you do not specify a `-np` option, then no default value is provided by the PBS interface. It is up to the standard `mpirun` to decide what the reasonable default value should be, which is usually *1*. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

5.2.7.3 MPD Startup and Shutdown

Intel MPI's `mpirun` takes care of starting and stopping the MPD daemons. The PBS interface to Intel MPI's `mpirun` always passes the arguments `-totalnum=<number of mpds to start>` and `-file=<mpd_hosts_file>` to the actual `mpirun`, taking its input from unique entries in $PBS_NODEFILE.

5.2.7.4 Examples

Example 5-15: Run a single-executable Intel MPI job with six processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

Node file:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job script:

```
# mpirun takes care of starting the MPD
# daemons on unique hosts listed in
# $PBS_NODEFILE, and also runs the 6 processes
# on the 6 hosts listed in
# $PBS_NODEFILE; mpirun takes care of
# shutting down MPDs.
mpirun /path/myprog.x 1200
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

Example 5-16: Run an Intel MPI job with multiple executables on multiple hosts using $PBS_NODEFILE and `mpiexec` arguments to `mpirun`:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
# mpirun runs MPD daemons on hosts listed in $PBS_NODEFILE
# mpirun runs 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on hostC.
# mpirun takes care of shutting down the
# MPDs at the end of MPI job run.
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np 2 /tmp/mpitest3
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
 <job-id>
```

Example 5-17: Run an Intel MPI job with multiple executables on multiple hosts via the -configfile option and $PBS_NODEFILE:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
echo "-np 2 /tmp/mpitest1" >> my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file

# mpirun takes care of starting the MPD daemons
# config file says run 2 instances of mpitest1
# on hostA; 2 instances of mpitest2 on
# hostB; 2 instances of mpitest3 on hostC.
# mpirun takes care of shutting down the MPD daemons.
mpirun -configfile my_config_file

# cleanup
rm -f my_config_file
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

5.2.7.5 Restrictions

The maximum number of ranks that can be launched under integrated Intel MPI is the number of entries in $PBS_NODEFILE.

5.2.8 LAM MPI with PBS

LAM MPI can be integrated with PBS on Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate LAM MPI with PBS.

Support for LAM MPI is **deprecated**.

5.2.8.1 Using LAM 7.x with PBS

You can run jobs under PBS using LAM 7.x without making any changes to your mpirun call.

5.2.8.2 Using LAM 6.5.9 with PBS

You can run jobs under PBS using LAM 6.5.9.

5.2.8.2.i Caveats for LAM 6.5.9 with PBS

- If you specify the `bhost` argument, PBS will print a warning saying that the `bhost` argument is ignored by PBS.
- If you do not specify the `where` argument, `pbs_mpilam` will try to run the your program on all available CPUs using the *C* keyword.

5.2.8.3 Example Job Submission Script

The following is a simple PBS job script for use with LAM MPI:

```
#!/bin/bash

# Job Name
#PBS -N LamSubTest
# Merge output and error files
#PBS -j oe
# Select 2 nodes with 1 CPU each
#PBS -l select=2:ncpus=1
# Export Users Environmental Variables to Execution Host
#PBS -V
# Send email on abort, begin and end
#PBS -m abe
# Specify mail recipient
#PBS -M username@example.com

cd $PBS_O_WORKDIR

date
lamboot -v $PBS_NODEFILE
mpirun -np $(cat $PBS_NODEFILE|wc -l) ./ANY_C_MPI_CODE HERE
date
```

When using the integrated `lamboot` in a job script, `lamboot` takes input from $PBS_NODEFILE automatically, so the argument is not necessary.

5.2.9 MPICH-P4 with PBS

The wrapper is **deprecated**. MPICH-P4 can be integrated with PBS on Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate MPICH-P4 with PBS.

5.2.9.1 Options for MPICH-P4 with PBS

Under PBS, the syntax and arguments for the MPICH-P4 `mpirun` command on Linux are the same except for one option, which you should not set:

-machinefile <file>

PBS supplies the machinefile. If you try to specify it, PBS prints a warning that it is replacing the machinefile.

5.2.9.2 Example of Using MPICH-P4 with PBS

Example of using `mpirun`:

```
#PBS -l select=arch=linux
#
mpirun a.out
```

5.2.9.3 MPICH Under Windows

Under Windows, you may need to use the `-localroot` option to MPICH's `mpirun` command in order to allow the job's processes to run more efficiently, or to get around the error "`failed to communicate with the barrier command`". Here is an example job script:

```
C:\DOCUME~1\user1>type job.scr
echo begin
type %PBS_NODEFILE%
"\Program Files\MPICH\mpd\bin\mpirun" -localroot -np 2 -machinefile %PBS_NODEFILE%
    \winnt\temp\netpipe -reps 3
echo done
```

5.2.9.3.i Caveats for MPICH Under Windows

Under Windows, MPICH is not integrated with PBS. Therefore, PBS is limited to tracking and controlling processes and performing accounting only for job processes on the primary vnode.

5.2.10 MPICH-GM with PBS

5.2.10.1 Using MPICH-GM and MPD with PBS

The wrapper is **deprecated**. PBS provides an interface to MPICH-GM's `mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-GM processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun` with MPD had been used.

You use the same `mpirun` command as you would use outside of PBS. If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

5.2.10.1.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` with MPD except for the following:

-m <file>

The `file` argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

5.2.10.1.ii MPD Startup and Shutdown

The script starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either the `rsh` or `ssh` method based on the value of the environment variable RSHCOMMAND. The default is `rsh`. The script also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun` will start GM's MPD daemons as you on the allocated PBS hosts. The MPD daemons may have been started already by the administrator or by you. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun` that you executed (GM or MX), which would determine the path to the MPD binary.

5.2.10.1.iii Examples

Example 5-18: Run a single-executable MPICH-GM job with 3 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
```

```
qsub -l select=3:ncpus=1
[MPICH-GM-HOME]/bin/mpirun -np 3 /path/myprog.x 1200
^D
<job-id>
```

If the GM MPD daemons are not running, the PBS interface to `mpirun` will start them as you on the allocated PBS hosts. The daemons may have been previously started by the administrator or by you.

Example 5-19: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file `procgrp`:

Job script:

```
qsub -l select=2:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp

[MPICH-GM-HOME]/bin/mpirun -pg procgrp /path/mypro.x 1200
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun` gives the warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control.
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

5.2.10.2 Using MPICH-GM and `rsh`/`ssh` with PBS

PBS provides an interface to MPICH-GM's `mpirun` using `rsh/ssh`. If executed inside a PBS job, this lets PBS track all MPICH-GM processes started via `rsh/ssh` so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard `mpirun` had been used.

You use the same `mpirun` command as you would use outside of PBS.

5.2.10.2.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` except for the following:

-machinefile <file>

The `file` argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

5.2.10.2.ii Examples

Example 5-20: Run a single-executable MPICH-GM job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1 -l place=scatter
mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 5-21: Run an MPICH-GM job with multiple executables on multiple hosts listed in the process group file `procgrp`:

```
qsub -l select=2:ncpus=1
echo  "host1 1 user1 /x/y/a.exe arg1 arg2" > procgrp
echo  "host2 1 user1 /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/mypro.x
rm -f procgrp
^D
<job-id>
```

When the job runs, `mpirun` gives this warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under the control of PBS.
```

The warning is issued because if any of the hosts listed in `procgrp` are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

5.2.10.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-GM is the number of entries in $PBS_NODEFILE.

5.2.11 MPICH-MX with PBS

5.2.11.1 Using MPICH-MX and MPD with PBS

The wrapper is **deprecated**. PBS provides an interface to MPICH-MX's `mpirun` using MPD. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by the MPD daemons so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MPICH-MX `mpirun` with MPD was used.

You use the same `mpirun` command as you would use outside of PBS. If the MPD daemons are not already running, the PBS interface will take care of starting them for you.

5.2.11.1.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as `mpirun` with MPD except for the following:

-m <file>

The `file` argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the `-pg` option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

5.2.11.1.ii MPD Startup and Shutdown

The PBS `mpirun` interface starts MPD daemons on each of the unique hosts listed in $PBS_NODEFILE, using either the `rsh` or `ssh` method, based on value of environment variable RSHCOMMAND. The default is `rsh`. The interface also takes care of shutting down the MPD daemons at the end of a run.

If the MPD daemons are not running, the PBS interface to `mpirun` starts MX's MPD daemons as you on the allocated PBS hosts. The MPD daemons may already have been started by the administrator or by you. MPD daemons are not started inside a PBS prologue script since it won't have the path of `mpirun` that you executed (GM or MX), which would determine the path to the MPD binary.

5.2.11.1.iii Examples

Example 5-22: Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1 -lplace=scatter
[MPICH-MX-HOME]/bin/mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

If the MPD daemons are not running, the PBS interface to mpirun starts MX's MPD daemons as you on the allocated PBS hosts. The MPD daemons may be already started by the administrator or by you.

Example 5-23: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file procgrp:

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
[MPICH-MX-HOME]/bin/mpirun -pg procgrp /path/myprog.x 1200
rm -f procgrp
^D
<job-id>
```

mpirun prints a warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

5.2.11.2 Using MPICH-MX and rsh/ssh with PBS

Deprecated. PBS provides an interface to MPICH-MX's mpirun using rsh/ssh. If executed inside a PBS job, this allows for PBS to track all MPICH-MX processes started by rsh/ssh so that PBS can perform accounting and has complete job control. If executed outside of a PBS job, it behaves exactly as if standard mpirun had been used.

You use the same mpirun command as you would use outside of PBS.

5.2.11.2.i Options

Inside a PBS job script, all of the options to the PBS interface are the same as standard mpirun except for the following:

-machinefile <file>

The file argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-np

If not specified, the number of entries found in the $PBS_NODEFILE is used. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

-pg

The use of the -pg option, for having multiple executables on multiple hosts, is allowed but it is up to you to make sure only PBS hosts are specified in the process group file; MPI processes spawned on non-PBS hosts are not guaranteed to be under the control of PBS.

5.2.11.2.ii Examples

Example 5-24: Run a single-executable MPICH-MX job with 64 processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
...
pbs-host64
```

```
qsub -l select=64:ncpus=1
mpirun -np 64 /path/myprog.x 1200
^D
<job-id>
```

Example 5-25: Run an MPICH-MX job with multiple executables on multiple hosts listed in the process group file procgrp:

```
qsub -l select=2:ncpus=1
echo  "pbs-host1 1 username /x/y/a.exe arg1 arg2" > procgrp
echo  "pbs-host2 1 username /x/x/b.exe arg1 arg2" >> procgrp
mpirun -pg procgrp /path/myprog.x
rm -f procgrp
^D
<job-id>
```

mpirun prints the warning message:

```
warning: "-pg" is allowed but it is up to user to make sure only PBS hosts are specified; MPI
    processes spawned are not guaranteed to be under PBS-control
```

The warning is issued because if any of the hosts listed in procgrp are not under the control of PBS, then the processes on those hosts will not be under the control of PBS.

5.2.11.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-MX is the number of entries in $PBS_NODEFILE.

5.2.12 MPICH2 with PBS on Linux

On Linux, PBS provides an interface to MPICH2's mpirun. If executed inside a PBS job, this allows for PBS to track all MPICH2 processes so that PBS can perform accounting and have complete job control. If executed outside of a PBS job, it behaves exactly as if standard MPICH2's mpirun had been used.

You use the same mpirun command as you would use outside of PBS.

When submitting PBS jobs under the PBS interface to MPICH2's mpirun, be sure to explicitly specify the actual number of ranks or MPI tasks in the qsub select specification. Otherwise, jobs will fail to run with "too few entries in the machinefile".

For instance, the following erroneous specification:

```
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB
mpirun -np 3 /tmp/mytask
```

results in this $PBS_NODEFILE listing:

```
hostA
hostB
```

which conflicts with the "`-np 3`" specification in `mpirun` as only two MPD daemons are started.

The correct way is to specify either of the following:

```
#PBS -l select=1:ncpus=1:host=hostA+2:ncpus=1:host=hostB
#PBS -l select=1:ncpus=1:host=hostA+1:ncpus=2:host=hostB:mpiprocs=2
```

which causes $PBS_NODEFILE to contain:

```
hostA
hostB
hostB
```

and this is consistent with "`mpirun -np 3`".

5.2.12.1 Options

If executed inside a PBS job script, all of the options to the PBS interface are the same as MPICH2's `mpirun` except for the following:

-host, -ghost

For specifying the execution host to run on. Ignored.

-machinefile <file>

The file argument contents are ignored and replaced by the contents of $PBS_NODEFILE.

-localonly <number of processes>

For specifying the *number of processes* to run locally. Not supported. You are advised instead to use the equivalent arguments:

"`-np <x> -localonly`".

-np

If you do not specify a `-np` option, then no default value is provided by the PBS interface to MPICH2. It is up to the standard `mpirun` to decide what the reasonable default value should be, which is usually 1. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

5.2.12.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in $PBS_NODEFILE. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

5.2.12.3 Examples

Example 5-26: Run a single-executable MPICH2 job with six processes spread out across the PBS-allocated hosts listed in $PBS_NODEFILE. Only three hosts are available:

$PBS_NODEFILE:

```
pbs-host1
pbs-host2
pbs-host3
pbs-host1
pbs-host2
pbs-host3
```

Job.script:

```
# mpirun runs 6 processes, scattered over 3 hosts
# listed in $PBS_NODEFILE
mpirun -np 6 /path/myprog.x 1200
```

Run job script:

```
qsub -l select=6:ncpus=1 -lplace = scatter job.script
<job-id>
```

Example 5-27: Run an MPICH2 job with multiple executables on multiple hosts using $PBS_NODEFILE and `mpiexec` arguments in `mpirun`:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2:mpiprocs=2
mpirun -np 2 /tmp/mpitest1 : -np 2 /tmp/mpitest2 : -np 2 /tmp/mpitest3
```

Run job:

```
qsub job.script
```

Example 5-28: Run an `MPICH2` job with multiple executables on multiple hosts using `mpirun -configfile` option and $PBS_NODEFILE:

$PBS_NODEFILE:

```
hostA
hostA
hostB
hostB
hostC
hostC
```

Job script:

```
#PBS -l select=3:ncpus=2:mpiprocs=2
echo "-np 2 /tmp/mpitest1" > my_config_file
echo "-np 2 /tmp/mpitest2" >> my_config_file
echo "-np 2 /tmp/mpitest3" >> my_config_file
mpirun -configfile my_config_file
rm -f my_config_file
```

Run job:

```
qsub job.script
```

5.2.12.4 Restrictions

The maximum number of ranks that can be launched under integrated MPICH2 is the number of entries in $PBS_NODEFILE.

5.2.13 MPICH2 1.4.1p1 On Windows with PBS

On Windows PBS supplies a wrapper script for MPICH2 1.4.1p1 called `pbs_mpich2_mpirun.bat`, located in `$PBS_EXEC\bin`. You call this script instead of MPICH2 `mpirun`. All options are passed through the script to `mpirun`.

5.2.14 MVAPICH with PBS

The wrapper is **deprecated**. PBS provides an `mpirun` interface to the MVAPICH `mpirun`. When you use the PBS-supplied `mpirun`, PBS can track all MVAPICH processes, perform accounting, and have complete job control. Your PBS administrator can integrate MVAPICH with PBS so that you can use the PBS-supplied `mpirun` in place of the MVAPICH `mpirun` in your job scripts.

MVAPICH allows your jobs to use InfiniBand.

5.2.14.1 Interface to MVAPICH `mpirun` Command

If executed outside of a PBS job, the PBS-supplied interface to `mpirun` behaves exactly as if standard MVAPICH `mpirun` had been used.

If executed inside a PBS job script, all of the options to the PBS interface are the same as MVAPICH's `mpirun` except for the following:

-map
: The `map` option is ignored.

-machinefile <file>
: The `machinefile` option is ignored.

-exclude
: The `exclude` option is ignored.

-np
: If you do not specify a `-np` option, then PBS uses the number of entries found in $PBS_NODEFILE. The maximum number of ranks that can be launched is the number of entries in $PBS_NODEFILE.

5.2.14.2 Examples

Example 5-29: Run a single-executable MVAPICH job with six ranks spread out across the PBS-allocated hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Contents of job.script:

```
# mpirun runs 6 processes mapped one to each line in $PBS_NODEFILE
mpirun -np 6 /path/myprog
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

5.2.14.3 Restrictions

The maximum number of ranks that can be launched under integrated MVAPICH is the number of entries in $PBS_NODEFILE.

5.2.15 MVAPICH2 with PBS

PBS provides an `mpiexec` interface to MVAPICH2's `mpiexec`. When you use the PBS-supplied `mpiexec`, PBS can track all MVAPICH2 processes, perform accounting, and have complete job control. Your PBS administrator can integrate MVAPICH2 with PBS so that you can use the PBS-supplied `mpirun` in place of the MVAPICH2 `mpirun` in your job scripts.

MVAPICH2 allows your jobs to use InfiniBand.

5.2.15.1 Interface to MVAPICH2 `mpiexec` Command

If executed outside of a PBS job, it behaves exactly as if standard MVAPICH2's `mpiexec` had been used.

If executed inside a PBS job script, all of the options to the PBS interface are the same as MVAPICH2's `mpiexec` except for the following:

-host
: The `host` option is ignored.

-machinefile <file>
: The `file` option is ignored.

-mpdboot
: If `mpdboot` is not called before `mpiexec`, it is called automatically before `mpiexec` runs so that an MPD daemon is started on each host assigned by PBS.

5.2.15.2 MPD Startup and Shutdown

The interface ensures that the MPD daemons are started on each of the hosts listed in $PBS_NODEFILE. It also ensures that the MPD daemons are shut down at the end of MPI job execution.

5.2.15.3 Examples

Example 5-30: Run a single-executable MVAPICH2 job with six ranks on hosts listed in $PBS_NODEFILE:

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
mpiexec -np 6 /path/mpiprog
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

Example 5-31: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "`mpd.hosts`". Here, run executables prog1 and prog2 with two ranks of prog1 on host1, two ranks of prog2 on host2 and two ranks of prog2 on host3, all specified on the command line:

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
mpiexec -n 2 prog1 : -n 2  prog2 : -n 2  prog2
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

Example 5-32: Launch an MVAPICH2 MPI job with multiple executables on multiple hosts listed in the default file "`mpd.hosts`". Run executables prog1 and prog2 with two ranks of prog1 on host1, two ranks of prog2 on host2 and two ranks of prog2 on host3, all specified using the `-configfile` option:

$PBS_NODEFILE:

```
pbs-host1
pbs-host1
pbs-host2
pbs-host2
pbs-host3
pbs-host3
```

Job.script:

```
echo "-n 2 -host host1 prog1" > /tmp/jobconf
echo "-n 2 -host host2 prog2" >> /tmp/jobconf
echo "-n 2 -host host3 prog2" >> /tmp/jobconf
mpiexec -configfile /tmp/jobconf
rm /tmp/jobconf
```

Run job script:

```
qsub -l select=3:ncpus=2:mpiprocs=2 job.script
<job-id>
```

5.2.15.4 Restrictions

The maximum number of ranks that can be launched under MVAPICH2 is the number of entries in $PBS_NODEFILE.

5.2.16 Open MPI with PBS

Open MPI can be integrated with PBS on Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate Open MPI with PBS.

5.2.16.1 Using Open MPI with PBS

You can run jobs under PBS using Open MPI without making any changes to your MPI command line.

5.2.17 Platform MPI with PBS

Platform MPI can be integrated with PBS on Linux so that PBS can track resource usage, signal processes, and perform accounting, for all job processes. Your PBS administrator can integrate Platform MPI with PBS.

5.2.17.1 Using Platform MPI with PBS

You can run jobs under PBS using Platform MPI without making any changes to your MPI command line.

5.2.17.2 Setting up Your Environment

In order to override the default `rsh`, set PBS_RSHCOMMAND in your job script:

```
export PBS_RSHCOMMAND=<rsh command>
```

5.2.18 HPE MPI with PBS

PBS supplies its own `mpiexec` to use with HPE MPI on a multi-vnode machine running supported versions of HPE MPI. When you use the PBS-supplied `mpiexec`, PBS can track resource usage, signal processes, and perform accounting, for all job processes. The PBS `mpiexec` provides the standard `mpiexec` interface.

See your PBS administrator to find out whether your system is configured for the PBS `mpiexec`.

5.2.18.1 Using HPE MPI with PBS

You can launch an MPI job on a single HPE system, or across multiple HPE systems. For MPI jobs across multiple HPE systems, PBS will manage the multi-host jobs. For example, if you have two HPE systems named host1 and host2, and want to run two applications called `mympi1` and `mympi2` on them, you can put this in your job script:

```
mpiexec -host host1 -n 4 mympi1 : -host host2 -n 8 mympi2
```

PBS will manage and track the job's processes. When the job is finished, PBS will clean up after it.

You can run MPI jobs in the placement sets chosen by PBS.

5.2.18.2 Prerequisites

In order to use MPI within a PBS job with HPE MPI, you may need to add the following in your job script before you call MPI:

```
module load mpt
```

5.2.18.3 Fitting Jobs onto Nodeboards

PBS will try to put a job that fits in a single nodeboard on just one nodeboard. However, if the only CPUs available are on separate nodeboards, and those vnodes are not allocated exclusively to existing jobs, and the job can share a vnode, then the job is run on the separate nodeboards.

5.2.18.4 Checkpointing and Suspending Jobs

Jobs are suspended on the HPE systems using the PBS suspend feature.

Jobs are checkpointed on HPE systems using application-level checkpointing. There is no OS-level checkpoint.

Suspended or checkpointed jobs will resume on the original nodeboards.

5.2.18.5 Using CSA

PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

5.3 Using PVM with PBS

You use the `pvmexec` command to execute a Parallel Virtual Machine (PVM) program. PVM is not integrated with PBS; PBS is limited to monitoring, controlling, and accounting for job processes only on the primary vnode.

5.3.1 Arguments to `pvmexec` Command

The `pvmexec` command expects a `hostfile` argument for the list of hosts on which to spawn the parallel job.

5.3.2 Using PVM Daemons

To start the PVM daemons on the hosts listed in $PBS_NODEFILE:

1. Start the PVM console on the first host in the list
2. Print the hosts to the standard output file named `jobname.o<PBS job ID>`:

```
echo conf | pvm $PBS_NODEFILE
```

To quit the PVM console but leave the PVM daemons running:

```
quit
```

To stop the PVM daemons, restart the PVM console, and quit:

```
echo halt | pvm
```

5.3.3 Submitting a PVM Job

To submit a PVM job to PBS, use the following:

qsub <job script>

5.3.4 Examples

Example 5-33: To submit a PVM job to PBS, use the following:

```
qsub  your_pvm_job
```

Here is an example script for your_pvm_job:

```
#PBS -N pvmjob
#PBS -V
cd $PBS_O_WORKDIR
echo conf   | pvm $PBS_NODEFILE
echo quit | pvm
./my_pvm_program
echo halt | pvm
```

Example 5-34: Sample PBS script for a PVM job:

```
#PBS -N pvmjob
#
pvmexec a.out -inputfile data_in
```

5.4 Using OpenMP with PBS

PBS Professional supports OpenMP applications by setting the OMP_NUM_THREADS variable in the job's environment, based on the resource request of the job. The OpenMP run-time picks up the value of OMP_NUM_THREADS and creates threads appropriately.

MoM sets the value of OMP_NUM_THREADS based on the first chunk of the `select` statement. If you request ompthreads in the first chunk, MoM sets the environment variable to the value of ompthreads. If you do not request ompthreads in the first chunk, then OMP_NUM_THREADS is set to the value of the ncpus resource of that chunk. If you do not request either ncpus or ompthreads for the first chunk of the select statement, then OMP_NUM_THREADS is set to *1*.

You cannot directly set the value of the OMP_NUM_THREADS environment variable; MoM will override any setting you attempt.

See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide for a definition of the ompthreads resource.

Example 5-35: Submit an OpenMP job as a single chunk, for a two-CPU, two-thread job requiring 10gb of memory:

```
qsub -l select=1:ncpus=2:mem=10gb
```

Example 5-36: Run an MPI application with 64 MPI processes, and one thread per process:

```
#PBS -l select=64:ncpus=1
mpiexec -n 64 ./a.out
```

Example 5-37: Run an MPI application with 64 MPI processes, and four OpenMP threads per process:

```
#PBS -l select=64:ncpus=4
mpiexec -n 64 omplace -nt 4 ./a.out
```

or

```
#PBS -l select=64:ncpus=4:ompthreads=4
mpiexec -n 64 omplace -nt 4 ./a.out
```

5.4.1 Running Fewer Threads than CPUs

You might be running an OpenMP application on a host and wish to run fewer threads than the number of CPUs requested. This might be because the threads need exclusive access to shared resources in a multi-core processor system, such as to a cache shared between cores, or to the memory shared between cores.

Example 5-38: You want one chunk, with 16 CPUs and eight threads:

```
qsub -l select=1:ncpus=16:ompthreads=8
```

5.4.2 Running More Threads than CPUs

You might be running an OpenMP application on a host and wish to run more threads than the number of CPUs requested, perhaps because each thread is I/O bound.

Example 5-39: You want one chunk, with eight CPUs and 16 threads:

```
qsub -l select=1:ncpus=8:ompthreads=16
```

5.4.3 Caveats for Using OpenMP with PBS

Make sure that you request the correct number of MPI ranks for your job, so that the PBS node file contains the correct number of entries. See section 5.1.3, "Specifying Number of MPI Processes Per Chunk", on page 78.

5.5 Hybrid MPI-OpenMP Jobs

For jobs that are both MPI and multi-threaded, the number of threads per chunk, for all chunks, is set to the number of threads requested (explicitly or implicitly) in the first chunk, except for MPIs that have been integrated with the PBS TM API.

For MPIs that are integrated with the PBS TM interface, (LAM MPI and Open MPI), you can specify the number of threads separately for each chunk, by specifying the ompthreads resource separately for each chunk.

For most MPIs, the OMP_NUM_THREADS and NCPUS environment variables default to the number of ncpus requested for the first chunk.

Should you have a job that is both MPI and multi-threaded, you can request one chunk for each MPI process, or set mpiprocs to the number of MPI processes you want on each chunk. See section 5.1.3, "Specifying Number of MPI Processes Per Chunk", on page 78.

5.5.1 Examples

Example 5-40: To request four chunks, each with one MPI process, two CPUs and two threads:

```
qsub -l select=4:ncpus=2
```

or

```
qsub -l select=4:ncpus=2:ompthreads=2
```

Example 5-41: To request four chunks, each with two CPUs and four threads:

```
qsub -l select=4:ncpus=2:ompthreads=4
```

Example 5-42: To request 16 MPI processes each with two threads on machines with two processors:

```
qsub -l select=16:ncpus=2
```

Example 5-43: To request two chunks, each with eight CPUs and eight MPI tasks and four threads:

```
qsub -l select=2:ncpus=8:mpiprocs=8:ompthreads=4
```

Example 5-44: For the following:

```
qsub -l select=4:ncpus=2
```

This request is satisfied by four CPUs from VnodeA, two from VnodeB and two from VnodeC, so the following is written to $PBS_NODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeC
```

The OpenMP environment variables are set, for the four PBS tasks corresponding to the four MPI processes, as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=2 NCPUS=2
- For PBS task #4 on VnodeC: OMP_NUM_THREADS=2 NCPUS=2

Example 5-45: For the following:

```
qsub -l select=3:ncpus=2:mpiprocs=2:ompthreads=1
```

This is satisfied by two CPUs from each of three vnodes (VnodeA, VnodeB, and VnodeC), so the following is written to $PBS_NODEFILE:

```
VnodeA
VnodeA
VnodeB
VnodeB
VnodeC
VnodeC
```

The OpenMP environment variables are set, for the six PBS tasks corresponding to the six MPI processes, as follows:

- For PBS task #1 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #2 on VnodeA: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #3 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #4 on VnodeB: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #5 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1
- For PBS task #6 on VnodeC: OMP_NUM_THREADS=1 NCPUS=1

Example 5-46: To run two threads on each of N chunks, each running a process, all on the same HPE system:

```
qsub -l select=N:ncpus=2 -l place=pack
```

This starts *N* processes on a single host, with two OpenMP threads per process, because OMP_NUM_THREADS=2.

6 Controlling How Your Job Runs

6.1 Using Job Exit Status

PBS can use the exit status of your job as input to the epilogue, and to determine whether to run a dependent job. If you are running under Linux, make sure that your job's exit status is captured correctly; see section 1.4.2.4, “Capture Correct Job Exit Status”, on page 6.

Job exit codes are listed in section 14.9, "Job Exit Status Codes", on page 519 of the PBS Professional Administrator’s Guide.

The exit status of a job array is determined by the status of each of its completed subjobs, and is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob. See “Job Array Exit Status” on page 156.

6.1.1 Caveats for Exit Status

- Normally, `qsub` exits with the exit status for a blocking job, but if you submit a job that is both blocking and interactive, PBS does not return the job's exit status. See section 6.10, “Making qsub Wait Until Job Ends”, on page 120.
- For a blocking job, the exit status is returned before staging finishes. See section 6.10.2, “Caveats for Blocking Jobs”, on page 121.
- The exit status of an interactive job is always recorded as 0 (zero), regardless of the actual exit status.

6.2 Using Job Dependencies

PBS allows you to specify dependencies between two or more jobs. Dependencies are useful for a variety of tasks, such as:

- Specifying the order in which jobs in a set should execute
- Requesting a job run only if an error occurs in another job
- Holding jobs until a particular job starts or completes execution

There is no limit on the number of dependencies per job.

If you have one or more jobs j2... jN that are dependent on a job j1 so that they can run only after j1 runs, and you delete j1, PBS deletes jobs j2... jN. If you have jobs j2... jN that can run only after j1 has not run successfully, and you delete j1, PBS releases the dependencies for jobs j2... jN so that they can run.

6.2.1 Syntax for Job Dependencies

Use the "`-W depend=<dependency list>`" option to `qsub` to define dependencies between jobs. The *dependency list* has the format:

<type>:<arg list>[,<type>:<arg list> ...]

where except for the on type, the *arg list* is one or more PBS job IDs in the form:

<job ID>[:<job ID> ...]

These are the available dependency types:

after:<arg list>
: This job may start only after all jobs in *arg list* have started execution.

afterok:<arg list>
: This job may start only after all jobs in *arg list* have terminated with no errors.

afternotok:<arg list>
: This job may start only after all jobs in *arg list* have terminated with errors.

afterany:<arg list>
: This job may start after all jobs in *arg list* have finished execution, with or without errors. This job will not run if a job in the *arg list* was deleted without ever having been run.

before:<arg list>
: Jobs in *arg list* may start only after specified jobs have begun execution. You must submit jobs that will run before other jobs with a type of on.

beforeok:<arg list>
: Jobs in *arg list* may start only after this job terminates without errors.

beforenotok:<arg list>
: If this job terminates execution with errors, the jobs in *arg list* may begin.

beforeany:<arg list>
: Jobs in *arg list* may start only after specified jobs terminate execution, with or without errors. Requires use of on dependency for jobs that will run before other jobs.

on:count
: This job may start only after *count* dependencies on other jobs have been satisfied. This type is used in conjunction with one of the before types. *count* is an integer greater than *0*.

runone:<job ID>
: (2020.1 Beta) Puts the current job and the job with *job ID* in a set of jobs out of which PBS will eventually run just one. To add a job to a set, specify the job ID of another job already in the set.

The depend job attribute controls job dependencies. You can set it using the `qsub` command line or a PBS directive:

```
qsub -W depend=...
#PBS depend=...
```

6.2.1.1 Running Your Job on First Available Resources (2020.1 Beta)

2020.1 Beta. This feature is subject to drastic change or removal without notice in future releases.

You may want to run a job on whichever resources become available first, even if the job could run on other sets of resources. You may want to start a flexible job as soon as possible on a smaller set of resources rather than waiting longer for a larger set of resources, or you may prefer certain resources but be able to use others (for example, you might prefer a specific processor, but still be able to run on another if that is all that's available).

If you submit a set of jobs where each job has a "runone" dependency on the others, PBS runs only one of the jobs in the "runone set". PBS automatically groups the jobs into a runone set. The jobs in a runone set can run different scripts.

When any of the jobs in the set starts, PBS applies a system hold to the others. The hold on the other jobs is released when the running job is requeued:

- Via `qrerun`
- When node fail requeue is triggered

The other jobs in the set are deleted:

- When a job ends, regardless of its exit status
- When the running job is deleted

To identify a job as a member of the set, give it a "runone" dependency on the previously-submitted member of the set. For example, we have three jobs, each of which runs on different resources. To submit these three jobs as a runone set:

```
qsub -lselect=200:ncpus=16 -lwalltime=1:00:00 myscript.sh
10.myserver
qsub -lselect=100:ncpus=16 -lwalltime=2:00:00 -Wdepend=runone:10 myscript.sh
11.myserver
qsub -lselect=50:ncpus=16 -lwalltime=4:00:00 -Wdepend=runone:10 myscript.sh
12.myserver
```

6.2.2 Job Dependency Examples

Example 6-1: You have three jobs, job1, job2, and job3, and you want job3 to start *after* job1 and job2 have *ended*:

```
qsub job1
16394.jupiter
qsub job2
16395.jupiter
qsub -W depend=afterany:16394:16395 job3
16396.jupiter
```

Example 6-2: You want job2 to start *only if* job1 ends with no errors:

```
qsub job1
16397.jupiter
qsub -W depend=afterok:16397 job2
16396.jupiter
```

Example 6-3: job1 should run before job2 and job3. To use the beforeany dependency, you must use the on dependency:

```
qsub -W depend=on:2 job1
16397.jupiter
qsub -W depend=beforeany:16397 job2
16398.jupiter
qsub -W depend=beforeany:16397 job3
16399.jupiter
```

6.2.3 Job Array Dependencies

Job dependencies are supported:

- Between jobs and jobs
- Between job arrays and job arrays
- Between job arrays and jobs
- Between jobs and job arrays

Job dependencies are not supported for subjobs or ranges of subjobs.

6.2.4 Caveats and Advice for Job Dependencies

6.2.4.1 Correct Exit Status Required

Under Linux, make sure that job exit status is captured correctly; see section 6.1, “Using Job Exit Status”, on page 107.

6.2.4.2 Permission Required for Dependencies

To use the before types, you must have permission to alter the jobs in *arg list*. Otherwise, the dependency is rejected and the new job is aborted.

6.2.4.3 Warning About Job History

Enabling job history changes the behavior of dependent jobs. If a job j1 depends on a finished job j2 for which PBS is maintaining history, PBS releases j1's dependency, and takes appropriate action. If job j1 depends on a finished job j3 that has been purged from job history, j1 is rejected just as in previous versions of PBS where the job was no longer in the system.

6.2.4.4 Error Reporting

PBS checks for errors in the existence, state, or condition of the job after accepting the job. If there is an error, PBS sends you mail about the error and deletes the job.

6.3 Adjusting Job Running Time

This feature was added in PBS Professional 12.0.

6.3.1 Shrink-to-fit Jobs

PBS allows you to submit a job whose running time can be adjusted to fit into an available scheduling slot. The job's minimum and maximum running time are specified in the min_walltime and max_walltime resources. PBS chooses the actual walltime. Any job that requests min_walltime is a **shrink-to-fit** job.

6.3.1.1 Requirements for a Shrink-to-fit Job

A job must have a value for min_walltime to be a shrink-to-fit job. Shrink-to-fit jobs are not required to request max_walltime, but it is an error to request max_walltime and not min_walltime.

Jobs that do not have values for min_walltime are not shrink-to-fit jobs, and you can specify their walltime.

6.3.1.2 Comparison Between Shrink-to-fit and Non-shrink-to-fit Jobs

The only difference between a shrink-to-fit and a non-shrink-to-fit job is how the job's walltime is treated. PBS sets the walltime when it runs the job. Any walltime value that exists before the job runs is ignored.

6.3.2 Using Shrink-to-fit Jobs

If you have jobs that can run for less than the expected time needed and still make useful progress, you can make them shrink-to-fit jobs in order to maximize utilization.

You can use shrink-to-fit jobs for the following:

- Jobs that are internally checkpointed. This includes jobs which are part of a larger effort, where a job does as much work as it can before it is killed, and the next job in that effort takes up where the previous job left off.
- Jobs using periodic PBS checkpointing
- Jobs whose real running time might be much less than the expected time
- When you have dedicated time for system maintenance, and you want to take advantage of time slots right up until shutdown, you can run speculative shrink-to-fit jobs if you can risk having a job killed before it finishes. Similarly, speculative jobs can take advantage of the time just before a reservation starts
- Any job where you do not mind running the job as a speculative attempt to finish some work

6.3.3 Running Time of a Shrink-to-fit Job

6.3.3.1 Setting Running Time Range for Shrink-to-fit Jobs

It is only required that the job request min_walltime to be a shrink-to-fit job. Requesting max_walltime without requesting min_walltime is an error.

You can set the job's running time range by requesting min_walltime and max_walltime, for example:

qsub -l min_walltime=<min walltime>, max_walltime=<max walltime> <job script>

6.3.3.2 Setting walltime for Shrink-to-fit Jobs

For a shrink-to-fit job, PBS sets the walltime resource based on the values of min_walltime and max_walltime, regardless of whether walltime is specified for the job.

PBS examines each shrink-to-fit job when it gets to it, and looks for a time slot whose length is between the job's min_walltime and max_walltime. If the job can fit somewhere, PBS sets the job's walltime to a duration that fits the time slot, and runs the job. The chosen value for walltime is visible in the job's Resource_List.walltime attribute. Any existing walltime value, regardless of where it comes from, e.g. previous execution, is reset to the new calculated running time.

If a shrink-to-fit job is run more than once, PBS recalculates the job's running time to fit an available time slot that is between min_walltime and max_walltime, and resets the job's walltime, each time the job is run.

For a multi-vnode job, PBS chooses a walltime that works for all of the chunks required by the job, and places job chunks according to the placement specification.

6.3.4 Modifying Shrink-to-fit and Non-shrink-to-fit Jobs

6.3.4.1 Modifying min_walltime and max_walltime

You can change min_walltime and/or max_walltime for a shrink-to-fit job by using the `qalter` command. Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

6.3.4.1.i Making Non-shrink-to-fit Jobs into Shrink-to-fit Jobs

You can convert a normal non-shrink-to-fit job into a shrink-to-fit job using the `qalter` command to set values for min_walltime and max_walltime.

Any changes take effect after the current scheduling cycle. Changes affect only queued jobs; running jobs are unaffected unless they are rerun.

6.3.4.1.ii Making Shrink-to-fit Jobs into Non-shrink-to-fit Jobs

To make a shrink-to-fit job into a normal, non-shrink-to-fit job, use the `qalter` command to do the following:

- Set the job's walltime to the value for max_walltime
- Unset min_walltime
- Unset max_walltime

6.3.5 Viewing Running Time for a Job

6.3.5.1 Viewing min_walltime and max_walltime

You can use `qstat -f` to view the values of min_walltime and max_walltime. For example:

```
% qsub -lmin_walltime=01:00:15, max_walltime=03:30:00 job.sh
<job-id>
% qstat -f <job-id>
...
Resource_List.min_walltime=01:00:15
Resource_List.max_walltime=03:30:00
```

You can use `tracejob` to display max_walltime and min_walltime as part of the job's resource list. For example:

```
12/16/2011 14:28:55  A    user=pbsadmin group=Users project=_pbs_project_default
…
    Resource_List.max_walltime=10:00:00
    Resource_List.min_walltime=00:00:10
```

6.3.5.2 Viewing walltime for a Shrink-to-fit Job

PBS sets a job's walltime only when the job runs. While the job is running, you can see its walltime via `qstat -f`. While the job is not running, you cannot see its real walltime; it may have a value set for walltime, but this value is ignored.

You can see the walltime value for a finished shrink-to-fit job if you are preserving job history. See section 14.15, “Managing Job History”, on page 530.

6.3.6 Lifecycle of a Shrink-to-fit Job

6.3.6.1 Execution of Shrink-to-fit Jobs

Shrink-to-fit jobs are started just like non-shrink-to-fit jobs.

6.3.6.2 Termination of Shrink-to-fit Jobs

When a shrink-to-fit job exceeds the walltime PBS has set for it, it is killed by PBS exactly as a non-shrink-to-fit job is killed when it exceeds its walltime.

6.3.7 The min_walltime and max_walltime Resources

max_walltime

Maximum walltime allowed for a shrink-to-fit job. Job's actual walltime is between max_walltime and min_walltime. PBS sets walltime for a shrink-to-fit job. If this resource is specified, min_walltime must also be specified. Must be greater than or equal to min_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. If not specified, PBS uses 5 years as the maximum time slot. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

min_walltime

Minimum walltime allowed for a shrink-to-fit job. When this resource is specified, job is a shrink-to-fit job. If this attribute is set, PBS sets the job's walltime. Job's actual walltime is between max_walltime and min_walltime. Must be less than or equal to max_walltime. Cannot be used for resources_min or resources_max. Cannot be set on job arrays or reservations. Can be requested only outside of a select statement. Non-consumable. Default: None. Type: duration. Python type: pbs.duration

6.3.8 Caveats and Restrictions for Shrink-to-fit Jobs

It is erroneous to specify max_walltime for a job without specifying min_walltime. If attempted via `qsub` or `qalter`, the following error is printed:

```
'Can not have "max_walltime" without "min_walltime"'
```

It is erroneous to specify a min_walltime that is greater than max_walltime. If attempted via `qsub` or `qalter`, the following error is printed:

```
'"min_walltime" can not be greater than "max_walltime"'
```

Job arrays cannot be shrink-to-fit. You cannot have a shrink-to-fit job array. It is erroneous to specify a min_walltime or max_walltime for a job array. If attempted via `qsub` or `qalter`, the following error is printed:

```
'"min_walltime" and "max_walltime" are not valid resources for a job array'
```

Reservations cannot be shrink-to-fit. You cannot have a shrink-to-fit reservation. It is erroneous to set min_walltime or max_walltime for a reservation. If attempted via `pbs_rsub`, the following error is printed:

```
'"min_walltime" and "max_walltime" are not valid resources for reservation.'
```

It is erroneous to set resources_max or resources_min for min_walltime and max_walltime. If attempted, the following error message is displayed, whichever is appropriate:

```
"Resource limits can not be set for min_walltime"
"Resource limits can not be set for max_walltime"
```

6.4 Using Checkpointing

6.4.1 Prerequisites for Checkpointing

A job is checkpointable if it has not been marked as non-checkpointable and any of the following is true:

- Its application supports checkpointing and your administrator has set up checkpoint scripts
- There is a third-party checkpointing application available
- The OS supports checkpointing

6.4.2 Minimum Checkpoint Interval

The execution queue in which the job resides controls the minimum interval at which a job can be checkpointed. The interval is specified in CPU minutes or walltime minutes. The same value is used for both, so for example if the minimum interval is specified as 12, then a job using the queue's interval for CPU time is checkpointed every 12 minutes of CPU time, and a job using the queue's interval for walltime is checkpointed every 12 minutes of walltime.

6.4.3 Syntax for Specifying Checkpoint Interval

Use the "`-c checkpoint-spec`" option to `qsub` to specify the interval, in CPU minutes, or in walltime minutes, at which the job will be checkpointed.

The *checkpoint-spec* argument is specified as:

c
: Job is checkpointed at the interval, measured in CPU time, set on the execution queue in which the job resides.

c=<minutes of CPU time>
: Job is checkpointed at intervals of the specified number of minutes of CPU time used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

  Format: *Integer*

w
: Job is checkpointed at the interval, measured in walltime, set on the execution queue in which the job resides.

w=<minutes of walltime>
: Checkpointing is to be performed at intervals of the specified number of minutes of walltime used by the job. This value must be greater than zero. If the interval specified is less than that set on the execution queue in which the job resides, the queue's interval is used.

  Format: *Integer*

n
: Job is not checkpointed.

s
: Job is checkpointed only when the PBS server is shut down.

u
: Checkpointing is unspecified, and defaults to the same behavior as "`s`".

The Checkpoint job attribute controls the job's checkpoint interval. You can set it using the `qsub` command line or a PBS directive:

Use `qsub` to specify that the job should use the execution queue's checkpoint interval:

```
qsub -c c my_job
```

Use a directive to checkpoint the job every 10 minutes of CPU time:

```
#PBS -c c=10
```

6.4.4 Using Checkpointing for Preempting or Holding Jobs

Your site may need to preempt jobs while they are running, or you may want to be able to place a hold your job while it runs. To allow either of these, make your job checkpointable. This means that you should not mark it as non-checkpointable (do not use `qsub -c n`), your application must be checkpointable or there is a third-party checkpointing application, and your administrator must supply a checkpoint script to be run by the MoM where the job runs.

You can use application-level checkpointing when your job is preempted or you place a hold on it to save the partial results. When your checkpointed job is restarted, your job script can find that the job was checkpointed, and can start from the checkpoint file instead of starting from scratch.

If you try to hold a running job that is not checkpointable (either it is marked as non-checkpointable or the script is missing or returns failure), the job continues to run with its Hold_Types attribute set to *h*. See section 6.5, "Holding and Releasing Jobs", on page 115.

6.4.5 Caveats and Restrictions for Checkpointing

- Checkpointing is not supported for job arrays.
- If you do not specify `qsub -c checkpoint-spec`, it is unspecified, and defaults to the same as "`s`".
- PBS limits the number of times it tries to run a job to 21, and tracks this count in the job's run_count attribute. If your job is checkpointed and requeued enough times, it will be held.

6.5 Holding and Releasing Jobs

You can place a hold on your job to do the following:

- A queued job remains queued until you release the hold; see section 6.5.3, "Holding a Job Before Execution", on page 116
- A running job stops running but can resume where it left off; see section 6.5.4.1, "Checkpointing and Requeueing the Job", on page 116
- A running job continues to run but is held if it is requeued; see section 6.5.4.2, "Setting Hold Type for a Running Job", on page 116

You hold a job using the qhold command; see "qhold" on page 147 of the PBS Professional Reference Guide.

You can release a held queued job to make it eligible to be scheduled to run, and you can release a hold on a running job. You release a hold on your job using the qrls command; see "qrls" on page 180 of the PBS Professional Reference Guide.

The qhold command uses the following syntax:

qhold [-h <hold list>] <job ID> [<job ID> ...]

The qrls command uses the following syntax:

qrls [-h <hold list>] <job ID> [<job ID> ...]

For a job array the *job ID* must be enclosed in double quotes.

6.5.1 Types of Holds

The *hold list* specifies the types of holds to be placed on the job. The *hold list* argument is a string consisting of one or more of the letters `u`, `p`, `o`, or `s` in any combination, or the letter `n`. The following table shows the hold type associated with each letter:

Table 6-1: Hold Types

Hold Type	Meaning	Who Can Set or Release
u	*User*	*Job owner, Operator, Manager, administrator, root*
o	*Other*	*Operator, Manager, administrator, root*

Table 6-1: Hold Types

Hold Type	Meaning	Who Can Set or Release
s	*System*	*Manager, administrator, root, PBS (dependency)*
n	*No hold*	*Job owner, Operator, Manager, administrator, root*
p	*Bad password*	*Administrator, root*

If no -h option is specified, PBS applies a *user* hold to the jobs listed in the *job ID* list.

If a job in the *job ID* list is in the queued, held, or waiting states, the only change is that the hold type is added to the job's other holds. If the job is queued or waiting in an execution queue, the job is also put in the held state.

6.5.2 Requirements for Holding or Releasing a Job

The user executing the `qhold` or `qrls` command must have the necessary privilege to apply a hold or release a hold. The same rules apply for releasing a hold and for for setting a hold.

6.5.3 Holding a Job Before Execution

Normally, PBS runs your job as soon as an appropriate slot opens up. However, you can tell PBS that the job is ineligible to run and should remain queued. Use the "`-h`" option to `qsub` to apply a *user hold* to the job when you submit it. PBS accepts the job and places it in the *held* state. The job remains held and ineligible to run until the hold is released.

The Hold_Types job attribute controls the job's holding behavior; set it via `qsub` or a directive:

```
qsub -h my_job
#PBS -h
```

6.5.4 Holding a Job During Execution

6.5.4.1 Checkpointing and Requeueing the Job

If your job is checkpointable, you can stop its execution by holding it. In this case the following happens:

- The job is checkpointed
- The resources assigned to the job are released
- The job is put back in the execution queue in the Held state

See section 6.4.1, “Prerequisites for Checkpointing”, on page 113.

To hold your job, use the `qhold` command:

```
qsub -h my_job
```

6.5.4.2 Setting Hold Type for a Running Job

If your job is not checkpointable, `qhold` merely sets the job's Hold_Types attribute. This has no effect unless the job is requeued with the `qrerun` command. In that case the job remains queued and ineligible to run until you release the hold.

6.5.5 Releasing a Job

You can release one or more holds on a job by using the `qrls` command.

The `qrls` command uses the following syntax:

qrls [-h <hold list>] <job ID> ...

For job arrays, the *job ID* must be enclosed in double quotes.

If you try to release a hold on a job which is not held, the `qrls` command is ignored. If you use the `qrls` command to release a hold on a job that had been previously running and was checkpointed, the hold is released and the job is returned to the queued (Q) state, and the job becomes eligible to be scheduled to run when resources come available.

The `qrls` command does not run the job; it simply releases the hold and makes the job eligible to be run the next time the scheduler selects it.

6.5.6 Caveats and Restrictions for Holding and Releasing Jobs

- The `qhold` command can be used on job arrays, but not on subjobs or ranges of subjobs. On job arrays, the `qhold` command can be applied only in the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. Job arrays cannot be moved in the 'H' state if any subjobs are running.
- Checkpointing is not supported for job arrays. Even on systems that support checkpointing, no subjobs will be checkpointed; they will run to completion.
- To hold a running job and stop its execution, the job must be checkpointable. See section 6.4.1, “Prerequisites for Checkpointing”, on page 113.
- The `qrls` command can only be used with job array objects, not with subjobs or ranges. The job array will be returned to its pre-hold state, which can be either 'Q', 'B', or 'W'.
- The `qhold` command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the 'Q', 'B' or 'W' states. This will put the job array in the 'H', held, state. If any subjobs are running, they will run to completion. No queued subjobs will be started while in the 'H' state.
- PBS limits the number of times it tries to run a job to 21, and tracks this count in the job's run_count attribute. If your job is checkpointed and requeued enough times, it will be held.

6.5.7 Why is Your Job Held?

Your job may be held for any of the following reasons:

- Provisioning fails due to invalid provisioning request or to internal system error ("s")
- After provisioning, the AOE reported by the vnode does not match the AOE requested by the job ("s")
- The job was held by a PBS Manager or Operator ("o")
- The job was checkpointed and requeued ("s")
- Your job depends on a finished job for which PBS is maintaining history ('s")
- The job's password is invalid ("p")
- The job's run_count attribute has a value greater than 20.

6.5.8 Examples of Holding and Releasing Jobs

The following examples illustrate how to use both the `qhold` and `qrls` commands. Notice that the state ("S") column shows how the state of the job changes with the use of these two commands.

```
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --

qhold 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 H  --




qrls -h u 54
qstat -a 54

                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq  engine  --   --   1  -- 0:20 Q  --
```

6.6 Allowing Your Job to be Re-run

You can specify whether or not your job is eligible to be re-run if for some reason the job is terminated before it finishes. Use the "`-r`" option to `qsub` to specify whether the job is rerunnable. The argument to this option is "*y*", meaning that the job can be re-run, or "*n*", meaning that it cannot. If you do not specify whether or not your job is rerunnable, it is rerunnable.

If running your job more than once would cause a problem, mark your job as non-rerunnable. Otherwise, leave it as rerunnable . The purpose of marking a job as non-rerunnable is to prevent it from starting more than once.

If a job that is marked non-rerunnable has an error during startup, before it begins execution, that job is requeued for another attempt.

The Rerunable job attribute controls whether the job is rerunnable; you can set it via `qsub` or a PBS directive:

```
qsub -r n my_job
#PBS -r n
```

The following table lists the circumstances where the job's Rerunable attribute makes a difference or does not:

Table 6-2: When does Rerunable Attribute Matter?

Circumstance	Rerunnable	Not Rerunnable
Job fails upon startup, before running	Job is requeued	Job is requeued
Job is running on multiple hosts, and one host goes down	Job is requeued	Job is deleted
Job is scheduled to run on multiple hosts, and did not start on at least one host	Job is requeued	Job is requeued
Server is shut down with a delay	Job is requeued	Job finishes
Server is shut down immediately	Job is requeued	Job is deleted
Job requests provisioning and provisioning script fails	Job is requeued	Job is requeued
Job is running on multiple hosts and one host becomes busy due to console activity	Job is requeued	Job is deleted
Higher-priority job needs resources	Job may be requeued	Job may be deleted

6.6.1 Caveats and Restrictions for Marking Jobs as Rerunnable

- Interactive jobs are not rerunnable.
- Job arrays are required to be rerunnable. PBS will not accept a job array that is marked as not rerunnable. You can submit a job array without specifying whether it is rerunnable, and PBS will automatically mark it as rerunnable.
- Mark your job as not rerunnable only if running it more than once would cause a problem. If your job is marked as not rerunnable, and a higher-priority job needs resources, your job could be deleted.

6.7 Controlling Number of Times Job is Re-run

PBS has a built-in limit of 21 on the number of times it will try to run your job. The number of attempts is tracked in the job's run_count attribute. By default, the value of run_count is zero at job submission. The job is held when the value of run_count goes above 20.

You can reduce the number of times PBS attempts to run your job. You can specify a non-negative value for run_count at job submission, and you can use `qalter` to raise the value of run_count while the job is running. You cannot give a job more retries than the limit, and you cannot lower the value of run_count while the job is running.

6.7.1 Caveats for Raising Value of run_count Attribute

If your job is checkpointed and requeued enough times, it will be held.

6.8 Deferring Execution

Normally, PBS runs your job as soon as an appropriate slot opens up. Instead, you can specify a time after which the job is eligible to run. The job is in the wait (W) state from the time it is submitted until the time it is eligible for execution.

6.8.1 Syntax for Deferring Execution

Use the "`-a <datetime>`" option to `qsub` to specify the time after which the job is eligible for execution. The *datetime* argument is in the form:

[[[[CC]YY]MM]DD]hhmm[.SS]

where

CC is the first two digits of the year (the century): optional

YY is the second two digits of the year: optional

MM is the two digits for the month: optional

DD is the day of the month: optional

hh is the hour

mm is the minute

SS is the seconds: optional

If the day *DD* is in the future, and the month *MM* is not specified, the month defaults to the current month. If the day *DD* is in the past, and the month *MM* is not specified, the month is set to next month. For example, if today is the 10th, and you specify the 12th but no month, your job is eligible to run two days from today, on the 12th.

Similarly, if the time *hhmm* is in the future, and the day *DD* is not specified, the day defaults to the current day. If the time *hhmm* is in the past, and the day *DD* is not specified, the day is set to tomorrow. For example, if you submit a job at 11:15am with a time of "`1110`", the job will be eligible to run at 11:10am tomorrow.

The job's Execution_Time attribute controls deferred execution. You can set it using either of the following:

```
qsub -a 0700 my_job
#PBS -a 10220700
```

6.9 Setting Priority for Your Job

PBS includes a place in each job where you can specify the job's priority. Your administrator may or may not choose to use this priority value when scheduling jobs. Use the "`-p <priority>`" to specify the priority of the job. The *priority* argument must be an integer between -1024 (lowest priority) and +1023 (highest priority) inclusive. The default is unset, which is equivalent to zero.

The Priority job attribute contains the value you specify. Set it via `qsub` or a directive:

```
qsub -p 120 my_job
#PBS -p -300
```

If you need an absolute ordering of your own jobs, see section 6.2, “Using Job Dependencies”, on page 107.

6.10 Making `qsub` Wait Until Job Ends

Normally, when you submit a job, the `qsub` command exits after returning the ID of the new job. You can use the "`-W block=true`" option to `qsub` to specify that you want `qsub` to "block", meaning wait for the job to complete and report the exit value of the job.

If your job is successfully submitted, `qsub` blocks until the job terminates or an error occurs. If job submission fails, no special processing takes place.

If the job runs to completion, qsub exits with the exit status of the job. For job arrays, blocking qsub waits until the entire job array is complete, then returns the exit status of the job array.

The block job attribute controls blocking. Set it either via qsub or a PBS directive:

```
qsub -W block=true
#PBS -W block=true
```

6.10.1 Signal Handling and Error Processing for Blocking Jobs

Signals SIGQUIT and SIGKILL are not trapped, and immediately terminate the qsub process, leaving the associated job either running or queued.

If qsub receives one of the signals SIGHUP, SIGINT, or SIGTERM, it prints a message and then exits with an exit status of *2*.

If the job is deleted before running to completion, or an internal PBS error occurs, qsub prints an an error message describing the situation to this error stream and qsub exits with an exit status of *3*.

6.10.2 Caveats for Blocking Jobs

- If you submit a job that is both blocking and interactive, the job's exit status is not returned at the end of the job.
- PBS returns the exit status of a blocking job before staging finishes for the job. To see whether the job is still staging, use qstat -f, and look at the job's substate attribute. This attribute has value *51* when files are staging out.

6.11 Running Your Job Interactively

PBS provides a special kind of batch job called an *interactive-batch job* or *interactive job*. An interactive job is treated just like a regular batch job in that it is queued up, and has to wait for resources to become available before it can run. However, once it starts, your terminal input and output are connected to the job similarly to a login session. It appears that you are logged into one of the available execution machines, and the resources requested by the job are reserved for that job. This is useful for debugging applications or for computational steering.

You can use GUI applications in interactive jobs on remote hosts. The PBS interface is slightly different on Linux and Windows. For Linux, see section 6.11.9, “Receiving X Output from Interactive Linux Jobs”, on page 124. For Windows, see section 6.11.10, “Submitting Interactive GUI Jobs on Windows”, on page 125.

Interactive jobs can use provisioning.

6.11.1 Input and Output for Interactive Jobs

An interactive job comes complete with a pseudotty suitable for running commands that set terminal characteristics. Once the interactive job has started execution, input to and output from the job pass through qsub. You provide all input to your interactive job through the terminal session in which the job runs.

For interactive jobs, you can specify PBS directives in a job script. You cannot provide commands to the job by using a job script. For interactive jobs, PBS ignores executable commands in job scripts.

6.11.2 Running Your Interactive Job

To run your job interactively, you can do either of the following:

- Use `qsub -I` at the command line
- Use `#PBS interactive=true (deprecated)` in a PBS directive

When your interactive job is running, you can run commands, executables, shell scripts, DOS commands, etc. These commands behave normally; for example, if the path to a command is not in your PATH environment variable, you must provide the full path.

6.11.3 Lifecycle of an Interactive Job

1. You start the interactive job using `qsub` `#PBS interactive=true (deprecated)` or `-I`
2. If there is a script, PBS processes any directives in the script
3. The scheduler runs the job
4. Output is connected to the submission window
5. You run commands, executables, shell scripts, etc. interactively
6. The job is terminated

6.11.3.1 Terminating Interactive Jobs

When you run an interactive job, the `qsub` command does not terminate when the job is submitted. `qsub` remains running until one of the following:

- You `qdel` the job
- Someone else deletes the job
- You exit the shell
- The job is aborted
- You interrupt `qsub` with a SIGINT (the control-C key) before the scheduler starts the job.

Once the scheduler starts the job, SIGINT is ignored.

Under Linux, if you interrupt `qsub` before the job starts, `qsub` queries whether you want it to exit. If you respond "*yes*", `qsub` exits and the job is aborted. Under Windows, if you interrupt the job before it starts, the job is deleted, and the following messages are printed:

```
qsub: wait for job <job ID> interrupted by signal 2
<job ID> is being deleted
```

6.11.4 Interactive Jobs and Exit Codes

Under Windows, if you specify an exit code when you exit the interactive session, via "exit <exit code>", that exit code is used as the job's exit code. This exit code is visible in the output of the `tracejob` command.

Under Linux, you cannot provide an exit code for the interactive session.

6.11.5 Tracking Progress for Interactive Jobs

After you have submitted an interactive job, PBS prints the following message to the window where you submitted the job:

```
qsub: waiting for job <job ID> to start
```

When the job is started by the scheduler, PBS prints the following message to the submission window:

```
qsub: job <job ID> ready
```

When the interactive job finishes, PBS prints the following message to the submission window:

```
qsub: job <job ID> completed
```

6.11.6 Special Sequences for Interactive Jobs

Keyboard-generated interrupts are passed to the job. Lines entered that begin with the tilde ('~') character and contain special sequences are interpreted by `qsub` itself. The recognized special sequences are:

~.
: `qsub` terminates execution. The batch job is also terminated.

~susp
: Suspends the `qsub` program. "*susp*" is the suspend character, usually CTRL-Z.

~asusp
: Suspends the input half of `qsub` (terminal to job), but allows output to continue to be displayed. "*asusp*" is the auxiliary suspend character, usually control-Y.

6.11.7 Caveats and Restrictions for Interactive Jobs

- Make sure that your login file does not run processes in the background. See section 1.4.2.5, “Avoid Background Processes Inside Jobs”, on page 6.
- You cannot run an array job interactively.
- Interactive jobs are not rerunnable.
- An interactive job on a Cray must run on a login node. See section 11.4.6, “Specify Host for Interactive Jobs on Cray XC”, on page 193.
- You cannot use the `CLS` command in an interactive job. It will not clear the screen.
- After the scheduler has started the interactive job, SIGINT (Ctrl-C) is ignored.
- Under Linux, you cannot provide an exit code for the interactive session.
- When an interactive job finishes, staged files and *`stdout`* and/or *`stderr`* may not have been copied back yet.
- The submission host must accept incoming ephemeral ports

6.11.8 Errors and Logging

- If PBS cannot open a remote interactive shell to run an interactive job, PBS prints the following error message:
  "qsub: failed to run remote interactive shell"
- If IPC$ on the remote host cannot be connected, PBS prints the following message:
  "Couldn't connect to host <hostname>"
- If PBS is successful in connecting to the IPC$ at the execution host, but fails to execute the remote shell, PBS prints the following error message:
  "Couldn't execute remote shell at host <hostname>"

6.11.9 Receiving X Output from Interactive Linux Jobs

Under Linux, you can receive X output from an interactive job via the `qsub -X` option.

6.11.9.1 How to Receive X Output Under Linux

To receive X output, use `qsub -X -I`. For example:

```
qsub -I -X <return>
xterm <return>
```

Control is returned here when your X process terminates. You can background the process here, if you want to.

6.11.9.1.i Receiving X Output on Non-submission Host

You can view your X output on a host that is not the job submission host. For example, you submit a job from SubHost, and want to see the output on ViewHost. If you want to receive X output on a host that is not the submission host, for example ViewHost, do the following:

- Run an X server on ViewHost
- On ViewHost, log into SubHost using `ssh -X`
- In window logged into SubHost, run `qsub -I -X`

6.11.9.2 Requirements for Receiving X Output

- You must be running Linux.
- The job must be interactive: you must also specify `-I`.
- An X server must be running on the system where you want to see the X output.
- The DISPLAY variable in the job's submission environment must be set to the display where the X output is desired.
- Your administrator must configure MoM's PATH to include the `xauth` utility.

6.11.9.3 Viewing X Output Job Attributes

Each job has two read-only attributes containing X forwarding information. These are the following:

forward_x11_cookie
: This attribute contains the X authorization cookie.

forward_x11_port
: This attribute contains the number of the port being listened to by the port forwarder on the submission host.

You can view these attributes using `qstat -f <job ID>`.

6.11.9.4 Caveats and Advice for Receiving X Output

- This option is not available under Windows.
- If you use the `qsub -V` option, PBS will handle the DISPLAY variable correctly.
- If you use the `qsub -v DISPLAY` option, you will get an error.
- At most 25 concurrent X applications can run using the same job session.
- If you experience a problem with X when using `qsub –X –I`, use the following to create the correct `~/.Xauthority` file for `qsub` to use when establishing the X session:
  `ssh –X <hostname> server <-> <exec host(s)>`

6.11.9.5 X Forwarding Errors

- If the DISPLAY environment variable is pointing to a display number that is correctly formatted but incorrect, submitting an interactive X forwarding job results in the following error message:
  `"cannot read data from 'xauth list <display number>', errno=<errno>"`
- If the DISPLAY environment variable is pointing to an incorrectly formatted value, submitting an interactive X forwarding job results in the following error message:
  `"qsub: Failed to get xauth data (check $DISPLAY variable)"`
- If the X authority utility (`xauth`) is not found on the submission host, the following error message is displayed:
  `"execution of xauth failed: sh: xauth: command not found"`
- When the execution of the `xauth` utility results in an error, the error message displayed by the `xauth` utility is preceded by the following:
  `"execution of xauth failed: "`
- When the `qsub -X` option is used without `-I`, the following error message is displayed:
  `"qsub: X11 forwarding possible only for Interactive Jobs"`

6.11.10 Submitting Interactive GUI Jobs on Windows

You can run an interactive job that uses a GUI application. If the job executes on a host other than the one from which you submit the job, PBS uses a remote viewer or interactive shell to connect the GUI application to the remote host. Under Windows, PBS supports any GUI application, including Remote Viewer and X. If your job requires a GUI application or interactive shell, you must run it as an interactive job.

To run an interactive PBS job that launches a GUI application:

`qsub -I -G -- <GUI application>`

When the same host is used for submission and execution, the application is launched on the local console. No remote viewer client is launched.

When the submission and execution hosts are different, the GUI application is launched in the remote session using the specified remote viewer. The remote viewer client is launched.

To run X under Windows, do not use the `-X` option. This option is not available under Windows. Use `-G`.

To launch an interactive shell in a PBS job:

`qsub -I -G`

When the submission and execution host are the same, the interactive shell is launched on the local console. No remote viewer client is launched.

When the submission and execution hosts are different, the interactive shell is launched, and any GUI application launched through this shell is visible in the remote session using the configured remote viewer. The remote viewer client is launched.

Your interactive GUI job is finished or no longer running under the following circumstances:

- When the GUI application launched via `qsub -I -G <GUI application>` is closed
- When the interactive shell launched via `qsub -I -G` exits
- When the remote viewer is terminated, closed, or logged off, all applications started by the remote viewer are closed.
- When a GUI job is deleted via `qdel`, all the applications and tasks associated with the job are killed

See "-G [<path to GUI application or script>]" on page 220 of the PBS Professional Reference Guide.

6.12 Using Environment Variables

PBS provides your job with environment variables where the job runs. PBS takes some from your submission environment, and creates others. You can create environment variables for your job. The environment variables created by PBS begin with "*PBS_*". The environment variables that PBS takes from your submission (originating) environment begin with "*PBS_O_*".

For example, here are a few of the environment variables that accompany a job, with typical values:

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/bin:/usr/local/bin:/bin
PBS_O_SHELL=/bin/tcsh
PBS_O_HOST=host1
PBS_O_WORKDIR=/u/user1
PBS_JOBID=16386.server1
```

For a complete list of PBS environment variables, see "PBS Environment Variables" on page 399 of the PBS Professional Reference Guide.

6.12.1 Exporting All Environment Variables

The "`-V`" option declares that all environment variables in the `qsub` command's environment are to be exported to the batch job.

```
qsub -V my_job
#PBS -V
```

6.12.2 Exporting Specific Environment Variables

The "`-v <variable list>`" option to `qsub` allows you to specify additional environment variables to be exported to the job. *variable list* names environment variables from the `qsub` command environment which are made available to the job when it executes. These variables and their values are passed to the job. These variables are added to those already automatically exported. Format: comma-separated list of strings in the form:

-v <variable>

or

-v <variable>=<value>

If a <*variable*>=<*value*> pair contains any commas, the value must be enclosed in single or double quotes, and the <*variable*>=<*value*> pair must be enclosed in the kind of quotes not used to enclose the value. For example:

```
qsub -v DISPLAY,myvariable=32 my_job
qsub -v "var1='A,B,C,D'" job.sh
qsub -v a=10, "var2='A,B'", c=20, HOME=/home/zzz job.sh
```

6.12.3 Caveat for Environment Variables and Shell Functions

Make sure that no exported shell function you want to forward has the same name as an environment variable. The shell function will not be visible in the environment.

6.12.4 Forwarding Exported Shell Functions

You can forward exported shell functions using either `qsub -V` or `qsub -v <function name>`. You can also put these functions in your `.profile` or `.login` on the execution host(s).

If you use `-v` or `-V`, make sure that there is no environment variable with the same name as any exported shell functions you want to forward; otherwise, the shell function will not be visible in the environment.

6.13 Specifying Which Jobs to Preempt

You can specify which groups of jobs your job is allowed to preempt in order to run. You can specify all the jobs in one or more queues, and all jobs that request particular resources, by listing them in the preempt_targets resource.

Syntax:

...-l preempt_targets="queue=<queue name>[,queue=<queue name>],
Resource_List.<resource>=<value>[,Resource_List.<resource>=<value>]"

For example, to specify that your job can preempt jobs in the queue named QueueA and/or jobs that requested arch=linux:

```
...-l preempt_targets="queue=QueueA,Resource_List.arch=linux"
```

You can prevent a job from preempting any other job in the complex by setting its preemption_targets to the keyword "None" (case-insensitive).

Make the preempt_targets resource specification last or use another -l specification for subsequent resource specifications. Otherwise, subsequent resource specifications will look to PBS like additions to preempt_targets.

6.14 Releasing Unneeded Vnodes from Your Job

If you want to prevent unnecessary resource usage, you can release unneeded sister hosts or vnodes (not the primary execution host or its vnodes) from your job. You can use the `pbs_release_nodes` command or the release_nodes_on_stageout job attribute:

- You can use the `pbs_release_nodes` command either at the command line or in your job script to release sister hosts or vnodes when the command is issued. You can use this command to release specific vnodes that are not on the primary execution host, or all vnodes that are not on the primary execution host. You can also use it to release all hosts or vnodes except for what you specify, which can be either a count of hosts to keep, or a select specification describing the vnodes to keep. You cannot use the command to release vnodes on the primary execution host. See "pbs_release_nodes" on page 92 of the PBS Professional Reference Guide.
- You can set the job's release_nodes_on_stageout attribute to *True* so that PBS releases all of the job's vnodes that are not on the primary execution host when stageout begins. You must set the job's stageout attribute as well. See "Job Attributes" on page 328 of the PBS Professional Reference Guide.

6.14.1 Caveats and Restrictions for Releasing Vnodes

- You must specify a stageout parameter in order to be able to release vnodes on stageout. If you do not specify stageout, release_nodes_on_stageout has no effect.
- You can release only vnodes that are not on the primary execution host. You cannot release vnodes on the primary execution host.
- The job must be running (in the *R* state).
- The `pbs_release_nodes` command is not supported on vnodes tied to Cray X* series systems (vnodes whose vntype has the "cray_" prefix).
- If cgroups support is enabled, and `pbs_release_nodes` is called to release some but not all the vnodes managed by a MoM, resources on those vnodes are released.
- If a vnode on a multi-vnode host is assigned exclusively to a job, and the vnode is released, the job will show that the vnode is released, but the vnode will still show as assigned to the job in `pbsnodes -av` until the other vnodes on that host have been released. If a vnode on a multi-vnode machine is not assigned exclusively to a job, and the vnode is released, it shows as released whether or not the other vnodes on that host are released.
- If you specify release of a vnode on which a job process is running, that process is terminated when the vnode is released.

6.14.2 What Happens When You Release Vnodes

After you release a job's vnode:

- The job's $PBS_NODEFILE no longer lists the released vnode
- The server continues to hold on to the job until receiving confirmation that the job has been cleaned up from the vnode
- The vnode reports to the primary execution host MoM its resources_used* values for the job as the final action. The released vnode no longer updates the resources_used values for the job since it's no longer part of the job. But the primary execution host holds onto the data, and adds the data during final aggregation of resources_used values when job exits
- After every successful call to `pbs_release_nodes`, `qstat` shows updated values for the job's exec_host, exec_vnode, and Resource_List attributes

When releasing vnodes, if all vnodes assigned to a job managed by the same MoM have been released, the job is completely removed from that MoM's host. This results in the following:

- The execjob_epilogue hook script (if it exists) runs
- Job processes are killed on that host
- Any job-specific specific files including job temporary directories are removed

If one or more, but not all, the vnodes from an execution host assigned to a job are released, the job is not removed from that host yet. If those released vnodes have been configured to be shared, they can be reassigned to other jobs.

6.14.3 Examples of Releasing Unneeded Vnodes From Job

Example 6-4: Submit a job that will release its non-primary-execution-host vnodes on stageout:

```
% qsub -W stageout=my_stageout@executionhost2:my_stageout.out -W release_nodes_on_stageout=true
  job.scr
```

Example 6-5: Release particular vnodes from a job:

Syntax: *pbs_release_nodes [-j <job ID>] <vnode name> [<vnode name>] ...]*

```
% qsub job.scr
241.myserverhost
% qstat  241 | grep "exec|Resource_List|select"
exec_host = executionhost1[0]/0*0+executionhost2/0*0+executionhost3/0*2
exec_vnode =
    (executionhost1[0]:mem=1048576kb:ncpus=1+executionhost1[1]:mem=1048576kb:ncpus=1+executionho
    st1[2]:ncpus=1)+(executionhost2:mem=104
8576kb:ncpus=1+executionhost2[0]:mem=1048576k:ncpus=1+executionhost2[1]:ncpus=1)+(executionhost3
    :ncpus=2:mem=2097152kb)
Resource_List.mem = 6gb
Resource_List.ncpus = 8
Resource_List.nodect = 3
Resource_List.place = scatter
Resource_List.select = ncpus=3:mem=2gb+ncpus=3:mem=2gb+ncpus=2:mem=2gb
schedselect = 1:ncpus=3:mem=2gb+1:ncpus=3:mem=2gb+1:ncpus=2:mem=2gb
% pbs_release_nodes -j 241 executionhost2[1] executionhost3
% qstat  241 | grep "exec|Resource_List|select"
exec_host = executionhost1[0]/0*0+executionhost2/0*0  (no executionhost3; all assigned vnodes in
    executionhost3 have been released)
exec_vnode =
    (executionhost1[0]:mem=1048576kb:ncpus=1+executionhost1[1]:mem=1048576kb:ncpus=1+executionho
    st1[2]:ncpus=1)+(executionhost2:mem=1048576kb:ncpus=1+executionhost2[0]:mem=1048576kb:ncpus=
    1)  (executionhost2[1] and executionhost3 no longer appear)
Resource_List.mem = 4194304kb  (reduced by 2gb from executionhost3)
Resource_List.ncpus = 5     (reduced by 3 CPUs, 1 from executionhost2[1] and 2 from executionhost3)
Resource_List.nodect = 2     (reduced by 1 chunk; when executionhost3 was released, its entire chunk assignment
    disappeared)
Resource_List.place = scatter
schedselect = 1:mem=2097152kb:ncpus=3+1:mem=2097152kb:ncpus=2
```

Example 6-6: Release all vnodes not on the primary execution host:

Syntax: *pbs_release_nodes [-j <job ID>] -a*

```
% pbs_release_nodes -j 241 -a
% qstat -f 241
exec_host = executionhost1[0]/0*0
exec_vnode =
    (executionhost1[0]:mem=1048576kb:ncpus=1)+executionhost1[1]:mem=1048576kb:ncpus=1+executionh
    ost1[2]:ncpus=1)
Resource_List.mem = 2097152kb
Resource_List.ncpus =  3
Resource_List.nodect = 1
Resource_List.place = scatter
schedselect = 1:mem=2097152kb:ncpus=3
```

Example 6-7: Release all sister hosts except for 4:

```
% pbs_release_nodes -k 4
```

Example 6-8: Release all sister vnodes except for 8 of those marked with "bigmem":

```
% pbs_release_nodes -k select=8:bigmem=true
```

Example 6-9: Sister vnodes are no longer listed in $PBS_NODEFILE after they are released:

```
% qsub -l select=2:ncpus=1:mem=1gb -l place=scatter -I
qsub: waiting for job 247.executionhost1.example.com to start
qsub: job 247.executionhost1.example.com ready
% cat $PBS_NODEFILE
executionhost1.example.com
executionhost2.example.com
% pbs_release_nodes -j 247 executionhost2
% cat $PBS_NODEFILE
executionhost1.example.com
```

6.15 Running Your Job in a Container

PBS supports running multi-vnode, multi-host, and interactive jobs in Docker and Singularity containers.

You can pull from a public registry, or you can pull from a private registry as long as you can log into it.

If you do not specify a script, for example "`qsub -l container_image=hello-world`", `qsub` asks you interactively for a script.

If you supply a script to `qsub`, PBS runs the script inside the specified container.

For a multi-host job, you can use any version of OpenMPI with containers.

PBS runs an infinite-duration sleep command in the container to keep the container alive.

6.15.1 Requesting a Container Engine

You can specify a container engine by requesting a resource whose value is set to that engine, or you can use the default by not requesting one. You can request only one container engine per job, even though this resource is requested at the host level. You must request the same container engine for all chunks. Ask your administrator for the name of the resource listing available container engines, or find it using `pbsnodes` (look for container engine names). We recommend that this resource is named "container engine".

qsub ... -l select=ncpus=...:<container engine resource>=<container engine>

6.15.2 Requesting a Container Image

You request a container image for your job via `-l container_image=<container image>` or by setting the CONTAINER_IMAGE environment variable to the name of the image and passing the environment variable with the job:

qsub ... -l container_image=<container image> ...

or

qsub ... -v CONTAINER_IMAGE=<name of container image> ...

6.15.2.1 Specifying a Registry

If you don't specify a registry, PBS uses a default, set by your administrator. You can specify the registry in the container image.

Example 6-10: Specifying the registry (and namespace) in the container image:

```
qsub -v CONTAINER_IMAGE=pbsprohub.local/pbsuser/test-image
```

6.15.2.2 Pulling from a Private Registry

To pull from a private registry, PBS uses a credential file to log into the registry. This file is in JSON format.

6.15.2.2.i Registry Credential Filename

The credential filename has this format:

<job owner>/.container/tokens.json

6.15.2.2.ii Registry Credential File Format

The file contents have this format:

```
{
   "registry1 <URL>/<endpoint>": {
      "user_id" : "<user ID>" , "passwd" : "<generated OAUTH token/password>"
   },

   "registry2 <URL>/<endpoint>": {
      "user_id" : "<user ID>" , "passwd" : "<generated OAUTH token/password>"
   }
}
```

6.15.2.2.iii Registry Credential File Default Values

registry: default registry (first element in the allowed_registries parameter)

user_id: job owner; if this is empty, PBS tries instead with the job owner ID

passwd: no password

6.15.2.2.iv Registry Credential File Location

The registry credential file *base path* is the path to where registry credential files are stored, up to but not including `<job owner>/.container/tokens.json`. The default base path to registry credential files is `/home`. Your administrator can configure the base path to where registry credential files are stored.

Example 6-11: The base path is "/container/creds/", and your job owner is User1. The full path to the JSON file is:

```
/container/creds/User1/.container/tokens.json
```

6.15.2.3 Specifying Image Namespace

PBS uses the registry's default namespace for container images unless you specify otherwise. If the image you want is in a non-default namespace, specify the namespace with the image name.

Example 6-12: To request the Docker container engine and an image named "centos", using the "MyImages" namespace:

```
qsub -l select=1:ncpus=1:container_engine=docker -lcontainer_image="MyImages/centos"  --
   /bin/sleep 500
```

Example 6-13: To request the Docker container engine and an image named "centos", using the default namespace:

```
qsub -l select=1:ncpus=1:container_engine=docker -lcontainer_image="centos"  -- /bin/sleep 500
```

6.15.3 Specifying Ports with Docker Containers

For single-vnode jobs in Docker containers, you can request ports for applications. PBS maps requested ports to available ports on the host and returns the mapping. You request ports by listing comma-separated port numbers in the container_ports job resource. Lists of port numbers must be enclosed in single quotes. PBS sets the job's resources_used.container_ports value to comma-separated *<container port>:<host port>* pairs. For example, your job can request specific ports:

```
qsub -l container_ports="'2324,8989'" ...
```

PBS returns the port mapping in the job's resources_used.container_ports resource:

```
resources_used.container_ports = 2324:8080,8989:32771
```

6.15.4 Specifying Additional Arguments to Container Engine

You can specify additional arguments to the container engine via the PBS_CONTAINER_ARGS environment variable, which is a semicolon-separated list. For example, to specify `--shm-size` to be 1GB and `--tmpfs` to be "/run:rw,noexec,nosuid,size=65536k":

```
export PBS_CONTAINER_ARGS="--shm-size=1GB";"--tmpfs /run:rw,noexec,nosuid,size=65536k"
```

Your PBS administrator must whitelist any additional arguments before you use them in a job.

The `--env` and `--entrypoint` arguments to `docker run` are not supported.

6.15.5 Passing Environment Variables Into Containers

To pass environment variables directly to PBS, use `qsub -v <environment variable list>`. The `--env` argument is not supported for passing environment variables into containers.

6.15.6 Adding Job Owner to Secondary Groups in Docker Containers

Your administrator can configure PBS to add the job owner to secondary groups inside the container. These are the groups on the execution host where the job owner is already a member. This feature applies only to Docker containers, since Singularity automatically adds the job owner to all groups.

6.15.7 Running Single-vnode Single-host Jobs in Singularity Containers

In addition to using PBS to launch your containers, you can always run a single-vnode job in a single Singularity container by prepending your scripts, executables, or commands with the Singularity binary.

6.15.8 Specifying Shell in Container

You can run your default shell inside a container without taking any extra steps. To run a shell in a container using anything besides the default, you must specify the shell using the `-S` option to `qsub`. Make sure the selected shell is available inside the container.

6.15.9 Caveats and Restrictions

- You cannot use old-style resource requests such as `-lncpus` with containers.
- Any entry point in a container is disabled. If you want to run the equivalent of an entry point command, you must include the complete command with its arguments on the command line.
- Mounting some directories or files in your container may be restricted. Ask your administrator for details.

6.16 Running Your Job in the Cloud

To submit a job that can run in the cloud, submit it to the configured cloud queue. Syntax:

qsub -q <name of cloud queue> -l <resource request> <job script>

For example:

```
qsub -q cloudq -- /bin/sleep 100
```

6.16.1 Restrictions and Caveats for Cloud Bursting with PBS

- Cloud bursting is supported only on Linux.
- Reservations are not supported on cloud nodes.

6.17 Allowing Your Job to Tolerate Vnode Failures

You can allow your job to tolerate vnode failures if your administrator has configured PBS to do so. PBS lets you allocate extra vnodes to a job so that the job can successfully start and run even if some vnodes fail. PBS can allocate the extra vnodes only for startup, or for the life of the job. Later, for jobs where the extra vnodes are needed only for reliable startup, PBS can trim the allocated vnodes back to just what the job will use to run, releasing the unneeded vnodes for other jobs.

To allow your job to tolerate vnode failures during startup only, set the job's tolerate_node_failures attribute to "*start*".

To allow your job to tolerate vnode failures during the life of the job, set the job's tolerate_node_failures attribute to "*all*".

Examples of setting this attribute:

- Via qsub:
  ```
  qsub -W tolerate_node_failures="all" <job script>
  ```
- Via qalter:
  ```
  qalter -W tolerate_node_failures="job_start" <job ID>
  ```

7 Reserving Resources

In this chapter we go over job reservations only (advance, standing, and job-specific reservations); maintenance reservations are covered in "Reservations" on page 197 in the PBS Professional Administrator's Guide.

7.1 Glossary

Advance reservation

A reservation for a set of resources for a specified time. The reservation is available only to the creator of the reservation and any users or groups specified by the creator.

Degraded reservation

A job-specific or advance reservation for which one or more associated vnodes are unavailable.

A standing reservation for which one or more vnodes associated with any occurrence are unavailable.

Job-specific reservation

A reservation created for a specific job, for the same resources that the job requested.

Job-specific ASAP reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS schedules the reservation to run as soon as possible, and PBS moves the job into the reservation. Created when you use `pbs_rsub -Wqmove=<job ID>` on a queued job.

Job-specific now reservation

Reservation created for a specific running job. PBS immediately creates a job-specific now reservation on the same resources as the job is using, and moves the job into the reservation. The reservation is created and starts running immediately when you use `pbs_rsub --job <job ID>` on a running job.

Job-specific start reservation

Reservation created for a specific queued job, for the same resources the job requests. PBS starts the job according to scheduling policy. When the job starts, PBS creates and starts the reservation, and PBS moves the job into the reservation. Created when you use `qsub -Wcreate_resv_from_job=true` on a queued job.

Occurrence of a standing reservation

An instance of the standing reservation.

An occurrence of a standing reservation behaves like an advance reservation, with the following exceptions:

- while a job can be submitted to a specific advance reservation, it can only be submitted to the standing reservation as a whole, not to a specific occurrence. You can only specify *when* the job is eligible to run. See"qsub" on page 213 of the PBS Professional Reference Guide.
- when an advance reservation ends, it and all of its jobs, running or queued, are deleted, but when an occurrence ends, only its running jobs are deleted.

Each occurrence of a standing reservation has reserved resources which satisfy the resource request, but each occurrence may have its resources drawn from a different source. A query for the resources assigned to a standing reservation will return the resources assigned to the soonest occurrence, shown in the resv_nodes attribute reported by `pbs_rstat`.

Soonest occurrence of a standing reservation

The occurrence which is currently active, or if none is active, then it is the next occurrence.

Standing reservation

An advance reservation which recurs at specified times. For example, you can reserve 8 CPUs and 10GB every Wednesday and Thursday from 5pm to 8pm, for the next three months.

7.2 Quick Explanation of Reservations for Jobs

You can reserve resources to be used later by jobs, or you can create a reservation using the resources requested by a specific job, and move the job into that reservation.

You create an advance or standing reservation, then submit jobs to the reservation. An **advance reservation** reserves specific resources for a specific time period, and a **standing reservation** does the same thing, but for a repeating sequence of time periods.

PBS creates **job-specific reservations** by reserving the same resources that a queued job requests, or a running job is using, then moving the job into the reservation's queue.

- PBS creates Job-specific Start Reservations for specific queued jobs whose create_resv_from_job attribute is *True*. When the job runs, PBS creates and starts the reservation, and PBS moves the job into the reservation. This reservation allows you to re-run the job later without having to wait for it to be scheduled again. You can set this attribute at submission using `qsub -Wcreate_resv_from_job=true`.
- PBS creates Job-specific ASAP Reservations for specific queued jobs when you use `pbs_rsub -Wqmove=<job ID>` on those jobs. PBS creates the reservation and moves the job into the reservation, and the reservation is scheduled to run as soon as possible.
- PBS creates Job-specific Now Reservations for specific running jobs when you use `pbs_rsub --job <job ID>` on them. PBS immediately creates a reservation, starts it, and moves the job into the reservation. This reservation allows you to re-run the job without having to wait for it to be scheduled again.

7.3 Prerequisites for Reserving Resources

The time for which a reservation is requested is in the time zone at the submission host.

You must set the submission host's `PBS_TZID` environment variable. The format for `PBS_TZID` is a timezone location. Example: `America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Kolkata`. See section 1.4.4, "Setting Time Zone for Submission Host", on page 9.

7.4 Advance and Standing Reservations

7.4.1 Introduction to Creating and Using Advance and Standing Reservations

You can create both advance and standing reservations using the `pbs_rsub` command. PBS either confirms that the reservation can be made, or rejects the request. Once the reservation is confirmed, PBS creates a queue for the reservation's jobs. Jobs are then submitted to this queue.

When a reservation is confirmed, it means that the reservation will not conflict with currently running jobs, other confirmed reservations, or dedicated time, and that the requested resources are available for the reservation. A reservation request that fails these tests is rejected. All occurrences of a standing reservation must be acceptable in order for the standing reservation to be confirmed.

The pbs_rsub command returns a *reservation ID*, which is the reservation name. For an advance reservation, this reservation ID has the format:

R<sequence number>.<server name>

For a standing reservation, this reservation ID refers to the entire series, and has the format:

S<sequence number>.<server name>

You specify the resources for a reservation using the same syntax as for a job. Jobs in reservations are placed the same way non-reservation jobs are placed in placement sets.

The time for which a reservation is requested is in the time zone at the submission host.

The pbs_rsub command returns a reservation ID string, and the current status of the reservation.

You can create an advance or standing reservation so that if the reservation sits idle, it is automatically deleted after the amount of time you specify. For a standing reservation, this applies to each occurrence separately. If one occurrence of a standing reservation is deleted, the next occurrence still starts at its designated time. To have your reservation be deleted automatically, use pbs_rsub -Wdelete_idle_time=<allowed idle time> and specify the number of seconds as an integer, or the duration as *HH:MM:SS*.

For the options to the pbs_rsub command, see "pbs_rsub" on page 96 of the PBS Professional Reference Guide.

7.4.2 Creating Advance Reservations

You create an advance reservation using the pbs_rsub command. PBS must be able to calculate the start and end times of the reservation, so you must specify two of the following three options:

D Duration

E End time

R Start time

7.4.2.1 Setting Time Zone for Advance Reservations

If you need the time zone for your advance reservation to be UTC, set this when you create the reservation:

```
TZ=UTC pbs_rsub -R...
```

7.4.2.2 Examples of Creating Advance Reservations

The following example shows the creation of an advance reservation asking for 1 vnode, 30 minutes of wall-clock time, and a start time of 11:30. Since an end time is not specified, PBS will calculate the end time based on the reservation start time and duration.

```
pbs_rsub -R 1130 -D 00:30:00
```

PBS returns the reservation ID:

```
R226.south UNCONFIRMED
```

The following example shows an advance reservation for 2 CPUs from 8:00 p.m. to 10:00 p.m.:

```
pbs_rsub -R 2000.00 -E 2200.00 -l select=1:ncpus=2
```

PBS returns the reservation ID:

```
R332.south UNCONFIRMED
```

7.4.3 Creating Standing Reservations

You create standing reservations using the pbs_rsub command. You **must** specify a start and end date when creating a standing reservation. The recurring nature of the reservation is specified using the -r option to pbs_rsub. The -r option takes the recurrence_rule argument, which specifies the standing reservation's occurrences. The recurrence rule uses iCalendar syntax, and uses a subset of the parameters described in RFC 2445.

The recurrence rule can take two forms:

"FREQ=<freq spec>;COUNT=<count spec>;<interval spec>"

In this form, you specify how often there will be occurrences, how many there will be, and which days and/or hours apply.

"FREQ=<freq spec>;UNTIL=<until spec>;<interval spec>"

Do not include any spaces in your recurrence rule.

In this form, you specify how often there will be occurrences, when the occurrences will end, and which days and/or hours apply.

freq spec

This is the frequency with which the reservation repeats. Valid values are *WEEKLY|DAILY|HOURLY*

When using a *freq spec* of *WEEKLY*, you may use an *interval spec* of *BYDAY* and/or *BYHOUR*. When using a *freq spec* of *DAILY*, you may use an *interval spec* of *BYHOUR*. When using a *freq spec* of HOURLY, do not use an *interval spec*.

count spec

The exact number of occurrences. Number up to 4 digits in length. Format: integer.

interval spec

Specifies the interval at which there will be occurrences. Can be one or both of *BYDAY=<days>* or *BYHOUR=<hours>*. Valid values are *BYDAY = MO|TU|WE|TH|FR|SA|SU* and *BYHOUR = 0|1|2|...|23*. When using both, separate them with a semicolon. Separate days or hours with a comma.

For example, to specify that there will be recurrences on Tuesdays and Wednesdays, at 9 a.m. and 11 a.m., use BYDAY=TU,WE;BYHOUR=9,11

BYDAY should be used with *FREQ=WEEKLY*. *BYHOUR* should be used with *FREQ=DAILY* or *FREQ=WEEKLY*.

until spec

Occurrences will start up to but not after this date and time. This means that if occurrences last for an hour, and normally start at 9 a.m., then a time of 9:05 a.m on the day specified in the *until spec* means that an occurrence will start on that day.

Format: *YYYYMMDD[THHMMSS]*

Note that the year-month-day section is separated from the hour-minute-second section by a capital *T*.

Default: 3 years from time of reservation creation.

7.4.3.1 Setting Reservation Start Time and Duration

In a standing reservation, the arguments to the -R and -E options to pbs_rsub can provide more information than they do in an advance reservation. In an advance reservation, they provide the start and end time of the reservation. In a standing reservation, they can provide the start and end time, but they can also be used to compute the duration and the offset from the interval start.

The difference between the values of the arguments for -R and -E is the duration of the reservation. For example, if you specify

```
-R 0930 -E 1145
```

the duration of your reservation will be two hours and fifteen minutes. If you specify

```
-R 150800 -E 170830
```

the duration of your reservation will be two days plus 30 minutes.

The *interval spec* can be used to specify the day or the hour at which the interval starts. If you specify

```
-R 0915 -E 0945 ... BYHOUR=9,10
```

the duration is 30 minutes, and the offset is 15 minutes from the start of the interval. The interval start is at 9 and again at 10. Your reservation will run from 9:15 to 9:45, and again at 10:15 and 10:45. Similarly, if you specify

```
-R 0800 -E -1000 ... BYDAY=WE,TH
```

the duration is two hours and the offset is 8 hours from the start of the interval. Your reservation will run Wednesday from 8 to 10, and again on Thursday from 8 to 10.

Elements specified in the recurrence rule override those specified in the arguments to the -R and -E options. Therefore if you specify

```
-R 0730 -E 0830 ... BYHOUR=9
```

the duration is one hour, but the hour element (9:00) in the recurrence rule has overridden the hour element specified in the argument to -R (7:00). The offset is still 30 minutes after the interval start. Your reservation will run from 9:30 to 10:30. Similarly, if the 16th is a Monday, and you specify

```
-R 160800 -E 170900 ... BYDAY=TU;BYHOUR=11
```

the duration 25 hours, but both the day and the hour elements have been overridden. Your reservation will run on Tuesday at 11, for 25 hours, ending Wednesday at 12. However, if you specify

```
-R 160810 -E 170910 ... BYDAY=TU;BYHOUR=11
```

the duration is 25 hours, and the offset from the interval start is 10 minutes. Your reservation will run on Tuesday at 11:10, for 25 hours, ending Wednesday at 12:10. The minutes in the offset weren't overridden by anything in the recurrence rule.

The values specified for the arguments to the -R and -E options can be used to set the start and end times in a standing reservation, just as they are in an advance reservation. To do this, don't override their elements inside the recurrence rule. If you specify

```
-R 0930 -E 1030 ... BYDAY=MO,TU
```

you haven't overridden the hour or minute elements. Your reservation will run Monday and Tuesday, from 9:30 to 10:30.

7.4.3.2 Requirements for Creating Standing Reservations

- You must specify a start and end date.
- You must set the submission host's PBS_TZID environment variable. The format for PBS_TZID is a timezone location. Example: America/Los_Angeles, America/Detroit, Europe/Berlin, Asia/Calcutta. See section 1.4.4, “Setting Time Zone for Submission Host”, on page 9.
- The recurrence rule must be one unbroken line.
- The recurrence rule must be enclosed in double quotes.
- Vnodes that have been configured to accept jobs only from a specific queue (vnode-queue restrictions) cannot be used for advance or standing reservations. See your PBS administrator to determine whether some vnodes have been configured to accept jobs only from specific queues.
- Make sure that there are no spaces in your recurrence rule.

7.4.3.3 Examples of Creating Standing Reservations

For a reservation that runs every day from 8am to 10am, for a total of 10 occurrences:

```
pbs_rsub -R 0800 -E 1000 - r"FREQ=DAILY;COUNT=10"
```

Every weekday from 6am to 6pm until December 10, 2008:

```
pbs_rsub -R 0600 -E 1800 -r "FREQ=WEEKLY;BYDAY=MO,TU,WE,TH,FR;UNTIL=20081210"
```

Every week from 3pm to 5pm on Monday, Wednesday, and Friday, for 9 occurrences, i.e., for three weeks:

```
pbs_rsub -R 1500 -E 1700 -r "FREQ=WEEKLY;BYDAY=MO,WE,FR;COUNT=9"
```

7.5 Job-specific Reservations

7.5.1 Job-specific Start Reservations

PBS runs the job normally, and when the job starts, PBS creates and starts a *job-specific start reservation* and moves the job into the reservation. PBS creates the reservation using the same resources that are being used by the job. The reservation holds the resources needed for the job in case the job fails and needs to be re-submitted, allowing it to run again without having to wait to be scheduled. The reservation starts when the job starts and has the same end time as the job.

If you have a queued job that you think is likely to fail and need to be corrected and re-submitted, you can create a job-specific *start reservation*. When you submit the job, set its create_resv_from_job attribute to *True* using the `-W` option to qsub:

qsub ... -Wcreate_resv_from_job=true

For example, to create a job-specific start reservation for the job whose script is named myscript.sh:

```
qsub myscript.sh -Wcreate_resv_from_job=true
```

You can also `qalter` a queued job to set this attribute:

qalter -Wcreate_resv_from_job=true <job ID>

For example, to create a start reservation when job 1234.myserver starts:

```
qalter -Wcreate_resv_from_job=true 1234.myserver
```

A job-specific start reservation ID has the format:

R<sequence number>.<server name>

PBS sets the start reservation's reserve_job attribute to the ID of the job from which the reservation was created, sets the reservation's Reserve_Owner attribute to the value of the job's Job_Owner attribute, sets the reservation's resv_nodes attribute to the jobs's exec_vnode attribute, sets the reservation's resources to the job's schedselect attribute, and sets the reservation's Resource_List attribute to the job's Resource_List attribute.

The start reservation's duration and start time are the same as the job's walltime and start time. If the job is peer scheduled, the now reservation is created in the pulling complex.

The start reservation is created when the job begins execution. You can set the create_resv_from_job attribute to *True* at any time, but this is only effective if you do it before the job starts. If your job has started running and you want to create a job-specific reservation for it, create a job-specific now reservation; see section 7.5.3, “Job-specific Now Reservations”, on page 141.

Can be used only with queued jobs.

Cannot be used with job arrays, jobs being submitted to other reservations, or other users' jobs.

7.5.2 Job-specific ASAP Reservations

PBS schedules a *job-specific ASAP reservation* to start as soon as possible. PBS creates a *job-specific ASAP reservation* using the resources requested by a specific queued job, and moves the job into the reservation.

Other jobs can also be moved into that queue via `qmove` or submitted to that queue via `qsub`.

To create an ASAP reservation:

pbs_rsub -W qmove=<job ID>

For example, to create an ASAP reservation for job 1234.myserver:

```
pbs_rsub -W qmove="1234.myserver"
```

A job-specific ASAP reservation ID has the format:

R<sequence number>.<server name>

The `-R` and `-E` options to `pbs_rsub` are disabled when using the `-W qmove` option.

Cannot be used on job arrays.

For more information, see "pbs_rsub" on page 96 of the PBS Professional Reference Guide.

We recommend using ASAP reservations only for sites that set job walltime. A job's default walltime is 5 years. Therefore an ASAP reservation's start time can be 5 years later, or more, if all the jobs in the system have the default walltime.

The delete_idle_time attribute for an ASAP reservation has a default value of 10 minutes.

7.5.3 Job-specific Now Reservations

PBS creates and starts a *job-specific now reservation* on the same resources used by a running job, and moves the running job into the reservation. The reservation holds the resources needed for the job in case the job fails and needs to be re-submitted, allowing it to run again without having to wait to be scheduled.

If you realize that a running job needs modification and re-submitting, and you don't want to have to wait until the scheduler finds a slot, you can create a now reservation. Later, you can submit a modified version of the job into the reservation:

pbs_rsub --job <job ID>

For example, to create a now reservation for job 1234.myserver while it's running:

```
pbs_rsub --job 1234.myserver
```

A job-specific now reservation ID has the format:

R<sequence number>.<server name>

PBS sets the job's create_resv_from_job attribute to *True*, sets the now reservation's reserve_job attribute to the ID of the job from which the reservation was created, sets the reservation's Reserve_Owner attribute to the value of the job's Job_Owner attribute, sets the reservation's resv_nodes attribute to the jobs's exec_vnode attribute, sets the reservation's resources to the job's schedselect attribute, and sets the reservation's Resource_List attribute to the job's Resource_List attribute.

The now reservation's duration and start time are the same as the job's walltime and start time. If the job is peer scheduled, the now reservation is created in the pulling complex.

Can be used on running jobs only (jobs in the *R* state, with substate *42*).

Cannot be used with job arrays, jobs already in reservations, or other users' jobs.

7.6 Getting Confirmation of a Reservation

By default the pbs_rsub command does not immediately notify you whether the reservation is confirmed or denied. Instead you receive email with this information. You can specify that the pbs_rsub command should wait for confirmation by using the -I *<block time>* option. The pbs_rsub command will wait up to *block time* seconds for the reservation to be confirmed or denied and then notify you of the outcome. If *block time* is negative and the reservation is not confirmed in that time, the reservation is automatically deleted.

To find out whether the reservation has been confirmed, use the pbs_rstat command. It will display the state of the reservation. CO and RESV_CONFIRMED indicate that it is confirmed. If the reservation does not appear in the output from pbs_rstat, that means that the reservation was denied.

To ensure that you receive mail about your reservations, set the reservation's Mail_Users attribute via the -M *<email address>* option to pbs_rsub. By default, you will get email when the reservation is terminated or confirmed. If you want to receive email about events other than those, set the reservation's Mail_Points attribute via the -m *<mail events>* option. For more information, see "pbs_rsub" on page 96 of the PBS Professional Reference Guide and "Reservation Attributes" on page 303 of the PBS Professional Reference Guide.

7.7 Modifying Reservations

You can use the pbs_ralter command to alter an existing reservation. You can use this command to modify a job-specific reservation or an advance reservation or the next or current instance of a standing reservation. Syntax:

pbs_ralter [-D <duration>] [-E <end time>] [-G <auth group list>] [-I <block time>] [-l select=<select spec>] [-m <mail points>] [-M <mail list>] [-N <reservation name>] [-R <start time>] [-U <auth user list>] <reservation ID>

When modifying the reservation select specification, the chunks for the new specification must be a subset of the previous set of chunks.

After the change is requested, the change is either confirmed or denied. On denial of the change, the reservation is not deleted and is left as is, and the following message appears in the server's log:

```
Unable to alter reservation <reservation ID>
```

When a reservation is confirmed, the following message appears in the server's log:

```
Reservation alter successful for <reservation ID>
```

To find out whether or not the change was allowed:

- Use the pbs_rstat command: see whether you altered reservation attribute(s)
- Use the interactive option: check for confirmation after the blocking time has run out

If the reservation has not started and it cannot be confirmed on the same vnodes, PBS searches for another set of vnodes. See section 9.4, "Reservation Fault Tolerance", on page 432 of the PBS Professional Administrator's Guide.

You must be the reservation owner or the PBS Administrator to run this command.

For details, see "pbs_ralter" on page 86 of the PBS Professional Reference Guide.

7.8 Deleting Reservations

You can delete a reservation by using the `pbs_rdel` command. For a standing reservation, you can only delete the entire reservation, including all occurrences. When you delete a reservation, all of the jobs that have been submitted to the reservation are also deleted. A reservation can be deleted by its owner or by a PBS Operator or Manager. For example, to delete `S304.south`:

```
pbs_rdel S304.south
```

or

```
pbs_rdel S304
```

You can create a reservation so that if the reservation sits idle, it is automatically deleted after the amount of time you specify. For a standing reservation, this applies to each occurrence separately. If one occurrence of a standing reservation is deleted, the next occurrence still starts at its designated time. To have your reservation be deleted automatically, use `pbs_rsub -Wdelete_idle_time=<allowed idle time>` and specify the number of seconds as an integer, or the duration as *HH:MM:SS*.

7.9 Viewing the Status of a Reservation

The following table shows the list of possible states for a reservation. The states that you will usually see are `CO, UN, BD`, and `RN`, although a reservation usually remains unconfirmed for too short a time to see that state. See "Reservation States" on page 367 of the PBS Professional Reference Guide.

To view the status of a reservation, use the `pbs_rstat` command. It will display the status of all reservations at the PBS server. For a standing reservation, the `pbs_rstat` command will display the status of the soonest occurrence. Duration is shown in seconds. The `pbs_rstat` command will not display a custom resource which has been created to be invisible. See section 4.3.8, "Caveats and Restrictions on Requesting Resources", on page 57. This command has three options:

Table 7-1: Options to `pbs_rstat` Command

Option	Meaning	Description
`B`	Brief	Lists only the names of the reservations
`S`	Short	Lists in table format the name, queue name, owner, state, and start, duration and end times of each reservation
`F`	Full	Lists the name and all non-default-value attributes for each reservation.
<none>	Default	Default is `S` option

The full listing for a standing reservation is identical to the listing for an advance reservation, with the following additions:

- A line that specifies the recurrence rule:
  `reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5`
- An entry for the vnodes reserved for the soonest occurrence of the standing reservation. This entry also appears for an advance reservation, but will be different for each occurrence:
  `resv_nodes=(<vnode name>:...)`
- A line that specifies the total number of occurrences of the standing reservation:
  `reserve_count = 5`
- The index of the soonest occurrence:
  `reserve_index = 1`
- The timezone at the site of submission of the reservation is appended to the reservation's Variable_List attribute. For example, in California:
  `Variable_List=<other variables>PBS_TZID=America/Los_Angeles`

To get the status of a reservation at a server other than the default server, set the `PBS_SERVER` environment variable to the name of the server you wish to query, then use the `pbs_rstat` command. Your PBS commands will treat the new server as the default server, so you may wish to unset this environment variable when you are finished.

You can also get information about the reservation's queue by using the `qstat` command. See "qstat" on page 197 of the PBS Professional Reference Guide .

7.9.1 Examples of Viewing Reservation Status Using `pbs_rstat`

In our example, we have one advance reservation and one standing reservation. The advance reservation is for today, for two hours, starting at noon. The standing reservation is for every Thursday, for one hour, starting at 3:00 p.m. Today is Monday, April 28th, and the time is 1:00, so the advance reservation is running, and the soonest occurrence of the standing reservation is Thursday, May 1, at 3:00 p.m.

Example brief output:

```
pbs_rstat -B
Name: R302.south
Name: S304.south
```

Example short output:

```
pbs_rstat -S

Name      Queue User State Start / Duration / End
---------------------------------------------------------
R302.south R302 user1 RN Today 12:00 / 7200/  Today 14:00
S304.south S304 user1 CO May 1 2008 15:00/3600/May 1 2008 16:00
```

Example full output:

```
pbs_rstat -F
Name: R302.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_RUNNING
reserve_substate = 5
reserve_start = Mon Apr 28 12:00:00 2008
reserve_end = Mon Apr 28 14:00:00 2008
reserve_duration = 7200
queue = R302
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 02:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:00:00 2008
Mail_Users = user1@mydomain.com
mtime = Mon Apr 28 11:00:00 2008
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com

Name: S304.south
Reserve_Name = NULL
Reserve_Owner = user1@south.mydomain.com
reserve_state = RESV_CONFIRMED
reserve_substate = 2
reserve_start = Thu May 1 15:00:00 2008
reserve_end = Thu May 1 16:00:00 2008
reserve_duration = 3600
queue = S304
Resource_List.ncpus = 2
Resource_List.nodect = 1
Resource_List.walltime = 01:00:00
Resource_List.select = 1:ncpus=2
Resource_List.place = free
resv_nodes = (south:ncpus=2)
reserve_rrule = FREQ=WEEKLY;BYDAY=MO;COUNT=5
reserve_count = 5
reserve_index = 2
Authorized_Users = user1@south.mydomain.com
server = south
ctime = Mon Apr 28 11:01:00 2008
Mail_Users = user1@mydomain.com
```

```
mtime = Mon Apr 28 11:01:00 2008
Variable_List = PBS_O_LOGNAME=user1,PBS_O_HOST=south.mydomain.com,PBS_TZID=America/Los_Angeles
```

7.10 Submitting a Job to a Reservation

Jobs can be submitted to the queue associated with a reservation, or they can be moved from another queue into the reservation queue. You submit a job to a reservation by using the -q <*queue*> option to the qsub command to specify the reservation queue. For example, to submit a job to the soonest occurrence of a standing reservation named S123.south, submit to its queue S123:

```
qsub -q S123 <script>
```

You move a job into a reservation queue by using the qmove command. For more information, see "qsub" on page 213 of the PBS Professional Reference Guide and "qmove" on page 172 of the PBS Professional Reference Guide. For example, to qmove job 22.myhost from workq to S123, the queue for the reservation named S123.south:

```
qmove S123 22.myhost
```

or

```
qmove S123 22
```

A job submitted to a standing reservation without a restriction on when it can run will be run, if possible, during the soonest occurrence. In order to submit a job to a specific occurrence, use the -a <*start time*> option to the qsub command, setting the start time to the time of the occurrence that you want. You can also use a cron job to submit a job at a specific time. See "qsub" on page 213 of the PBS Professional Reference Guide and the cron(8) man page.

7.10.1 Who Can Use Your Reservation

By default, the reservation accepts jobs only from the user who created the reservation, and accepts jobs submitted from any group or host. You can specify a list of users and groups whose jobs will and will not be accepted by the reservation by setting the reservation's Authorized_Users and Authorized_Groups attributes using the -U <*authorized user list*> and -G <*authorized group list*> options to pbs_rsub and pbs_ralter. You can specify the hosts from which jobs can and cannot be submitted by setting the reservation's Authorized_Hosts attribute using the -H <*authorized host list*> option to pbs_rsub.

The administrator can also specify which users and groups can and cannot submit jobs to a reservation, and the list of hosts from which jobs can and cannot be submitted.

For more information, see "pbs_rsub" on page 96 of the PBS Professional Reference Guide and "Reservation Attributes" on page 303 of the PBS Professional Reference Guide.

7.10.2 Viewing Status of a Job Submitted to a Reservation

You can view the status of a job that has been submitted to a reservation or to an occurrence of a standing reservation by using the qstat command. See "qstat" on page 197 of the PBS Professional Reference Guide.

For example, if a job named MyJob has been submitted to the soonest occurrence of the standing reservation named S304.south, it is listed under S304, the name of the queue:

```
qstat

Job id      Name       User          Time Use  S  Queue
----------  ---------  ------------  --------  --  -----
139.south   MyJob      user1                0  Q  S304
```

7.10.3 How Reservations Treat Jobs

A confirmed reservation will accept jobs into its queue at any time. Jobs are only scheduled to run from the reservation once the reservation period arrives.

The jobs in a reservation are not allowed to use, in aggregate, more resources than the reservation requested. A reservation job is accepted in the reservation regardless of whether its requested walltime will fit within the reservation period. So for example if the reservation runs from 10:00 to 11:00, and the job's walltime is 4 hours, the job will be started.

When an advance reservation ends, any running or queued jobs in that reservation are deleted.

When an occurrence of a standing reservation ends, any running jobs in that reservation are killed. Any jobs still queued for that reservation are kept in the queued state. They are allowed to run in future occurrences. When the last occurrence of a standing reservation ends, all jobs remaining in the reservation are deleted, whether queued or running.

A job in a reservation cannot be preempted.

A job in a reservation runs with the normal job environment variables; see section 6.12, “Using Environment Variables”, on page 126.

7.10.3.1 Caveats for How Reservations Treat Jobs

If you submit a job to a reservation, and the job's walltime fits within the reservation period, but the time between when you submit the job and when the reservation ends is less than the job's walltime, PBS will start the job, and then kill it if it is still running when the reservation ends.

7.11 Reservation Caveats and Errors

7.11.1 Time Zone Must be Correct

The environment variable `PBS_TZID` must be set at the submission host. The time for which a reservation is requested is the time defined at the submission host. See section 1.4.4, “Setting Time Zone for Submission Host”, on page 9.

7.11.2 Time Required Between Reservations

Leave enough time between reservations for the reservations and jobs in them to clean up. A job consumes resources even while it is in the `E` or exiting state. This can take longer when large files are being staged. If the job is still running when the reservation ends, it may take up to two minutes to be cleaned up. The reservation itself cannot finish cleaning up until its jobs are cleaned up. This will delay the start time of jobs in the next reservation unless there is enough time between the reservations for cleanup.

7.11.3 Reservation Information in the Accounting Log

The PBS server writes an accounting record for each reservation in the job accounting file. The accounting record for a reservation is similar to that for a job. The accounting record for any job belonging to a reservation will include the reservation ID. See "Accounting" on page 627 in the PBS Professional Administrator’s Guide.

7.11.4 Reservation Fault Tolerance

If one or more vnodes allocated to a job-specific reservation, an advance reservation, or to the soonest occurrence of a standing reservation become unavailable, the reservation's state becomes *DG* or *RESV_DEGRADED*. A degraded reservation does not have all the reserved resources to run its jobs.

PBS attempts to reconfirm degraded reservations. This means that it looks for alternate available vnodes on which to run the reservation. The reservation's retry_time attribute lists the next time when PBS will try to reconfirm the reservation.

If PBS is able to reconfirm a degraded reservation, the reservation's state becomes *CO*, or *RESV_CONFIRMED*, and the reservation's resv_nodes attribute shows the new vnodes.

7.11.5 Job and Reservation Exclusivity Must Match

If your job requests exclusive placement, and it is in a reservation, the reservation must also request exclusive placement via `-l place=excl`.

8
Job Arrays

8.1 Advantages of Job Arrays

PBS provides job arrays, which are useful for collections of almost-identical jobs. Each job in a job array is called a "subjob". Subjobs are scheduled and treated just like normal jobs, with the exceptions noted in this chapter. You can group closely related work into a set so that you can submit, query, modify, and display the set as a unit. Job arrays are useful where you want to run the same program over and over on different input files. PBS can process a job array more efficiently than it can the same number of individual normal jobs. Job arrays are suited for SIMD operations, for example, parameter sweep applications, rendering in media and entertainment, EDA simulations, and forex (historical data).

8.2 Glossary

Job array identifier

The identifier returned upon success when submitting a job array. Format:

<sequence number>[]

Job array range

A set of subjobs within a job array. When specifying a range, indices used must be valid members of the job array's indices.

Sequence number

The numeric part of a job or job array identifier, e.g. *1234*.

Subjob

Individual entity within a job array (e.g. *1234[7]*, where *1234[]* is the job array itself, and *7* is the index) which has many properties of a job as well as additional semantics (defined below.)

Subjob index

The unique index which differentiates one subjob from another. This must be a non-negative integer.

8.3 Description of Job Arrays

A job array is a compact representation of two or more jobs. A job that is part of a job array is called a "subjob". Each subjob in a job array is treated exactly like a normal job, except for any differences noted in this chapter.

8.3.1 Job Script for Job Arrays

All subjobs in a job array share a single job script, including the PBS directives and the shell script portion. The job script is run once for each subjob.

The job script may invoke different commands based on the subjob index. The commands of course may be scripts themselves. You can do this by naming different commands with the subjob index or via "if" statements in the script.

8.3.2 Attributes and Resources for Job Arrays

All subjobs in one job array have the same attributes, including resource requirements and limits.

The same job script runs for each subjob in the job array. If the job script calls other scripts or commands, those scripts or commands cannot change the attributes and resources for individual subjobs, because PBS stops processing directives when it starts processing commands.

8.3.3 Scheduling Job Arrays and Subjobs

The scheduler handles each subjob in a job array as a separate job. All subjobs within a job array have the same scheduling priority.

8.3.3.1 Starving

A job array's starving status is based on the queued portion of the array. This means that if there is a queued subjob which is starving, the job array is starving. A running subjob retains the starving status it had when it was started.

8.3.4 Identifier Syntax

The sequence number (1234 in 1234[].<server>) is unique, so that jobs and job arrays cannot share a sequence number. The job identifiers of the subjobs in the same job array are the same except for their indices. Each subjob has a unique index. You can refer to job arrays or parts of job arrays using the following syntax forms:

- The job array object itself: The format is *<sequence number>[]* or *<sequence number>[].<server>.<domain>.com*

  Example: 1234[].myserver or 1234[]
- A single subjob with index *M*: The format is *<sequence number>[M]* or *<sequence number>[M].<server>.<domain>.com*

  Example where *M*=17: 1234[17].myserver or 1234[17]
- A range of subjobs of a job array: The format is *<sequence number>[start-end[:step]]* or *<sequence number>[start-end[:step]].<server>.<domain>.com*

  Example where we start at 2, end at 8, and the step is 3: 1234[2-8:3].myserver or 1234[2-8:3]

8.3.4.1 Examples of Using Identifier Syntax

1234[]	Short job array identifier
1234[].myserver.domain.com	Full job array identifier
1234[73]	Short subjob identifier of the 73rd index of job array 1234[]
1234[73].myserver.domain.com	Full subjob identifier of the 73rd index of job array 1234[]

8.3.4.2 Shells and Array Identifiers

Since some shells, for example `csh` and `tcsh`, read "[" and "]" as shell metacharacters, job array names and subjob names must be enclosed in double quotes for all PBS commands.

Example:

```
qdel "1234[5].myhost"
qdel "1234[].myhost"
```

Single quotes will work, except where you are using shell variable substitution.

8.3.5 Special Attributes for Job Arrays

Job arrays and subjobs have all of the attributes of a job. In addition, they have the following when appropriate. These attributes are read-only.

Table 8-1: Job Array Attributes

Name	Type	Applies To	Value
array	Boolean	Job array	*True* if item is job array
array_id	String	Subjob	Subjob's job array identifier
array_index	String	Subjob	Subjob's index number
array_indices_remaining	String	Job array	List of indices of subjobs still queued. Range or list of ranges, e.g. *500*, *552*, *596-1000*
array_indices_submitted	String	Job array	Complete list of indices of subjobs given at submission time. Given as range, e.g. *1-100*
array_state_count	String	Job array	Similar to state_count attribute for server and queue objects. Lists number of subjobs in each state.
max_run_subjobs	Integer	Job array	Limit on number of subjobs that can be running at one time.

8.3.6 Job Array States

The state of subjobs in the same job array can be different. See "Job Array States" on page 363 of the PBS Professional Reference Guide and "Subjob States" on page 364 of the PBS Professional Reference Guide.

8.3.7 PBS Environmental Variables for Job Arrays

Table 8-2: PBS Environmental Variables for Job Arrays

Environment Variable Name	Used For	Description
PBS_ARRAY_INDEX	Subjobs	Subjob index in job array, e.g. 7
PBS_ARRAY_ID	Subjobs	Identifier for a job array. Sequence number of job array, e.g. 1234[].myserver
PBS_JOBID	Jobs, subjobs	Identifier for a job or a subjob. For subjob, sequence number and subjob index in brackets, e.g. 1234[7].myserver

8.3.8 Accounting

Job accounting records for job arrays and subjobs are the same as for jobs. When a job array has been moved from one server to another, the subjob accounting records are split between the two servers.

Subjobs do not have "Q" records.

8.3.9 Prologues and Epilogues

If defined, prologues and epilogues run at the beginning and end of each subjob, but not for the array object.

8.3.10 The "Rerunnable" Flag and Job Arrays

Job arrays are required to be rerunnable. PBS will not accept a job array that is marked as not rerunnable. You can submit a job array without specifying whether it is rerunnable, and PBS will automatically mark it as rerunnable.

8.4 Submitting a Job Array

8.4.1 Job Array Submission Syntax

You submit a job array through a single command. You specify subjob indices, and optionally a limit on the number of subjobs that can be running at one time, at submission.

For the range, you can specify any of the following:

- A contiguous range, e.g. 1 through 100
- A range with a stepping factor, e.g. every second entry in 1 through 100 (1, 3, 5, ... 99)

The limit is an optional percent sign followed by an integer.

Syntax for submitting a job array:

qsub -J <index start>-<index end>[:<stepping factor>] [%<max subjobs>]

where

index start is the lowest index number in the range

index end is the highest index number in the range

stepping factor is the optional difference between index numbers

max subjobs is the limit on the number of subjobs that can be running at one time

The index start and end must be whole numbers, and the stepping factor must be a positive integer. The index end must be greater than the index start. If the index end is not a multiple of the stepping factor above the index start, it will not be used as an index value, and the highest index value used will be lower than the index end. For example, if index start is 1, index end is 8, and the stepping factor is 3, the index values are 1, 4, and 7.

8.4.1.1 Limiting Number of Simultaneously Running Subjobs

By default PBS simultaneously runs as many subjobs from a job array as possible. You can limit the number of subjobs that are running at one time by setting the value of the max_run_subjobs job attribute. This is helpful if for example every subjob needs access to the same shared data file and you want to prevent slowdowns due to an access bottleneck. You can set the limit at submission by appending %<max subjobs> to your `-J` option:

qsub -J <index start>-<index end>[:<stepping factor>] [%<max subjobs>]

For example:

```
qsub -J 1-20000 %500 myscript.sh
```

Or you can use `qalter` to set or change the max_run_subjobs attribute:

qalter -Wmax_run_subjobs=<new value> <job ID>

For example:

```
qalter -Wmax_run_subjobs=1000 123[].myserver
```

Suspended subjobs do not count against the limit set in max_run_subjobs.

8.4.2 Examples of Submitting Job Arrays

Example 8-1: To submit a job array of 10,000 subjobs, with indices 1, 2, 3, ... 10000:

```
$ qsub -J 1-10000 job.scr
1234[].server.domain.com
```

Example 8-2: To submit a job array of 500 subjobs, with indices 500, 501, 502, ... 1000:

```
$ qsub -J 500-1000 job.scr
1235[].server.domain.com
```

Example 8-3: To submit a job array with indices 1, 3, 5 ... 999:

```
$ qsub -J 1-1000:2 job.scr
1236[].server.domain.com
```

Example 8-4: To submit a job array of 10,000 subjobs with indices 1, 2, 3, ... 10000, and a limit of 500 simultaneously running subjobs:

```
$ qsub -J 1-10000 %500 job.scr
1237[].server.domain.com
```

8.4.3 File Staging for Job Arrays

When preparing files to be staged for a job array, plan on naming the files so that they match the index numbers of the subjobs. For example, `inputfile3` is meant to be used by the subjob with index value *3*.

To stage files for job arrays, you use the same mechanism as for normal jobs, but include a variable to specify the subjob index. This variable is named *array_index*.

8.4.3.1 File Staging Syntax for Job Arrays

You can specify files to be staged in before the job runs and staged out after the job runs. Format:

qsub -W stagein=<stagein file list> -W stageout=<stageout file list>

You can use these as options to `qsub`, or as directives in the job script.

For both stagein and stageout, the *file list* has the form:

<execution path>^array_index^@<storage host>:<storage path>^array_index^[,...]

The name *<execution path><index number>* is the name of the file in the job's staging and execution directory (on the execution host). The *execution path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *<storage path><index number>* is the file name on the host specified by *storage host*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. You must specify a *storage host*. The name can be absolute, or it can be relative to your home directory on the storage machine.

For stagein, the direction of travel is **from** *storage path* **to** *execution path*.

For stageout, the direction of travel is **from** *execution path* **to** *storage path*.

When staging more than one set of filenames, separate the filenames with a comma and enclose the entire list in double quotes.

8.4.3.2 Job Array Staging Syntax on Windows

In Windows the stagein and stageout string must be contained in double quotes when using ^array_index^.

Example of a stagein:

```
qsub -W stagein="foo.^array_index^@host-1:C:\WINNT\Temp\foo.^array_index^" -J 1-5 stage_script
```

Example of a stageout:

```
qsub -W stageut="C:\WINNT\Temp\foo.^array_index^@host-1:Q:\my_username\foo. ^array_index^_out" -J
    1-5 stage_script
```

8.4.3.3 Job Array File Staging Caveats

We recommend using an absolute pathname for the *storage path*. Remember that the path to your home directory may be different on each machine, and that when using sandbox = *PRIVATE*, you may or may not have a home directory on all execution machines.

8.4.3.4 Examples of Staging for Job Arrays

Example 8-5: Simple example:

Storage path: `store:/film`

Data files used as input: `frame1`, `frame2`, `frame3`

execution path: `pix`

Executable: `a.out`

For this example, `a.out` produces `frame2.out` from `frame2`.

```
#PBS -W stagein=pix/in/frame^array_index^@store:/film/frame^array_index^
#PBS- W stageout=pix/out/frame^array_index^.out @store:/film/frame^array_index^.out
#PBS -J 1-3 a.out frame$PBS_ARRAY_INDEX ./in ./out
```

Note that the stageout statement is all one line.

The result is that your directory named "`film`" contains the original files `frame1`, `frame2`, `frame3`, plus the new files `frame1.out`, `frame2.out`, and `frame3.out`.

Example 8-6: In this example, we have a script named ArrayScript which calls `scriptlet1` and `scriptlet2`.

All three scripts are located in `/homedir/testdir`.

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-2
echo "Main script: index " $PBS_ARRAY_INDEX
/homedir/testdir/scriptlet$PBS_ARRAY_INDEX
```

In our example, scriptlet1 and scriptlet2 simply echo their names. We run ArrayScript using the qsub command:

```
qsub ArrayScript
```

Example 8-7: In this example, we have a script called StageScript. It takes two input files, dataX and extraX, and makes an output file, newdataX, as well as echoing which iteration it is on. The dataX and extraX files will be staged from inputs to work, then newdataX will be staged from work to outputs.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="/homedir/work/data^array_index^@host1:/homedir/inputs/data^array_index^, \
    /homedir/work/extra^array_index^ @host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=/homedir/work/newdata^array_index^@host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cd /homedir/work
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX >> newdata$PBS_ARRAY_INDEX
```

Execution path:

```
/homedir/work
```

Storage host:

```
host1
```

Storage path for inputs (original data files dataX and extraX):

```
/homedir/inputs
```

Storage path for results (output of computation newdataX):

```
/homedir/outputs
```

StageScript resides in /homedir/testdir. In that directory, we can run it by typing:

```
qsub StageScript
```

It will run in /homedir, our home directory, which is why the line

```
"cd /homedir/work"
```

is in the script.

Example 8-8: In this example, we have the same script as before, but we will run it in a staging and execution directory created by PBS. StageScript takes two input files, dataX and extraX, and makes an output file, newdataX, as well as echoing which iteration it is on. The dataX and extraX files will be staged from inputs to the staging and execution directory, then newdataX will be staged from the staging and execution directory to outputs.

```
#!/bin/sh
#PBS -N StagingExample
#PBS -J 1-2
#PBS -W stagein="data^array_index^@host1:/homedir/inputs/data^array_index^, \
    extra^array_index^@host1:/homedir/inputs/extra^array_index^"
#PBS -W stageout=newdata^array_index^@host1:/homedir/outputs/newdata^array_index^
echo "Main script: index " $PBS_ARRAY_INDEX
cat data$PBS_ARRAY_INDEX extra$PBS_ARRAY_INDEX >> newdata$PBS_ARRAY_INDEX
```

Execution path (directory): created by PBS; we don't know the name

Storage host:

```
host1
```

Storage path for inputs (original data files dataX and extraX):

`/homedir/inputs`

Storage path for results (output of computation newdataX):

`/homedir/outputs`

StageScript resides in /homedir/testdir. In that directory, we can run it by typing:

`qsub StageScript`

It will run in the staging and execution directory created by PBS. See section 3.2, “Input/Output File Staging”, on page 31.

8.4.4 Filenames for Standard Output and Standard Error

The name for `stdout` for a subjob defaults to *<job name>.o<sequence number>.<index>*, and the name for `stderr` for a subjob defaults to *<job name>.e<sequence number>.<index>*.

Example 8-9: The job is named "fixgamma" and the sequence number is "1234".

The subjob with index 7 is 1234[7].<server name>. For this subjob, `stdout` and `stderr` are named `fixgamma.o1234.7` and `fixgamma.e1234.7`.

8.4.5 Job Array Dependencies

Job dependencies are supported for the following relationships:

- Between job arrays and job arrays
- Between job arrays and jobs
- Between jobs and job arrays

8.4.5.1 Caveats for Job Array Dependencies

Job dependencies are not supported for subjobs or ranges of subjobs.

8.4.6 Job Array Exit Status

The exit status of a job array is determined by the status of each of the completed subjobs. It is only available when all valid subjobs have completed. The individual exit status of a completed subjob is passed to the epilogue, and is available in the 'E' accounting log record of that subjob.

Table 8-3: Job Array Exit Status

Exit Status	Meaning
0	All subjobs of the job array returned an exit status of 0. No PBS error occurred. Deleted subjobs are not considered
1	At least 1 subjob returned a non-zero exit status. No PBS error occurred.
2	A PBS error occurred.

8.4.6.1 Making `qsub` Wait Until Job Array Finishes

Blocking `qsub` waits until the entire job array is complete, then returns the exit status of the job array.

8.4.6.2 Caveats for Job Array Exit Status

Subjob exit status is available only as long as the subjob is in job history. When a subjob is not in job history, a failed or terminated subjob will show an exit status of *Finished*, instead of failed or terminated.

8.4.7 Caveats for Submitting Job Arrays

8.4.7.1 No Interactive Job Submission of Job Arrays

Interactive submission of job arrays is not allowed.

8.5 Viewing Status of a Job Array

You can use the `qstat` command to query the status of a job array. The default output is to list the job array in a single line, showing the job array identifier. You can combine options.

You can use the `-f` option to the `qstat` command to see all of a subjob's attributes.

To show the state of all running subjobs, use `-t -r`. To show the state of subjobs only, not job arrays, use `-t -J`.

Table 8-4: Job Array and Subjob Options to `qstat`

Option	Result
-t	Shows state of job array object and subjobs. Also shows state of jobs.
-J	Shows state only of job arrays.
-p	Prints the default display, with column for Percentage Completed. For a job array, this is the number of subjobs completed or deleted divided by the total number of subjobs. For a job, it is time used divided by time requested.

8.5.1 Example of Viewing Job Array Status

We run an example job and an example job array, on a machine with 2 processors:

`demoscript`:

```
#!/bin/sh
#PBS -N JobExample
sleep 60
```

`arrayscript`:

```
#!/bin/sh
#PBS -N ArrayExample
#PBS -J 1-5
sleep 60
```

We run these scripts using `qsub`:

```
qsub arrayscript
1235[].host
qsub demoscript
1236.host
```

We query using various options to `qstat`:

```
qstat
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq
1236.host   JobExample   user1              0 Q workq

qstat -J
Job id      Name         User        Time Use S Queue
----------- ------------ ----------  -------- - -----
1235[].host ArrayExample user1              0 B workq

qstat -p
Job id      Name         User        % done  S Queue
----------- ------------ ----------  ------- - -----
1235[].host ArrayExample user1            0  B workq
1236.host   JobExample   user1           --  Q workq

qstat -t
Job id       Name         User        Time Use S Queue
-----------  ------------ ----------  -------- - -----
1235[].host  ArrayExample user1              0 B workq
1235[1].host ArrayExample user1       00:00:00 R workq
1235[2].host ArrayExample user1       00:00:00 R workq
1235[3].host ArrayExample user1              0 Q workq
1235[4].host ArrayExample user1              0 Q workq
1235[5].host ArrayExample user1              0 Q workq
1236.host    JobExample   user1              0 Q workq

qstat -Jt
Job id       Name         User   Time Use S Queue
------------ ------------ -----  -------- - -----
1235[1].host ArrayExample user1  00:00:00 R workq
1235[2].host ArrayExample user1  00:00:00 R workq
1235[3].host ArrayExample user1         0 Q workq
1235[4].host ArrayExample user1         0 Q workq
1235[5].host ArrayExample user1         0 Q workq
```

After the first two subjobs finish:

```
qstat -Jtp
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq

qstat -pt
Job id       Name         User   % done S Queue
------------ ------------ -----  ------ - -----
1235[].host  ArrayExample user1      40 B workq
1235[1].host ArrayExample user1     100 X workq
1235[2].host ArrayExample user1     100 X workq
1235[3].host ArrayExample user1      -- R workq
1235[4].host ArrayExample user1      -- R workq
1235[5].host ArrayExample user1      -- Q workq
1236.host    JobExample   user1      -- Q workq
```

Now if we wait until only the last subjob is still running:

```
qstat -rt
                                                  Req'd  Req'd  Elap
Job ID     Username Queue Jobname   SessID NDS TSK Memory Time S Time
---------- ------  ----- ---------  ---- --- --- ------ ---- - -----
1235[5].host user1  workq ArrayExamp 3048  --   1    --    --  R 00:00
1236.host    user1  workq JobExample 3042  --   1    --    --  R 00:00

qstat -Jrt
                                                  Req'd  Req'd  Elap
Job ID      Username Queue Jobname  SessID NDS TSK Memory Time S Time
----------- -------- ----- --------- ---- --- --- ------ ---- - -----
1235[5].host   user1 workq ArrayExamp 048 --   1    --    --  R 00:01
```

8.6 Using PBS Commands with Job Arrays

The following table shows how you can or cannot use PBS commands with job arrays, subjobs or ranges:

Table 8-5: Using PBS Commands with Job Arrays

	Argument to Command		
Command	**Array[]: Array Object**	**Array[Range]: Specified Range of Subjobs**	**Array[Index]: Specified Subjob**
`qalter`	Array object	erroneous	erroneous
`qdel`	Array object & Running subjobs	Running subjobs in specified range	Specified subjob
`qhold`	Array object & Queued subjobs	erroneous	erroneous
`qmove`	Array object & Queued subjobs	erroneous	erroneous
`qmsg`	erroneous	erroneous	erroneous
`qorder`	Array object	erroneous	erroneous
`qrerun`	Running and finished subjobs	Running subjobs in specified range	Specified subjob
`qrls`	Array object & Queued subjobs	erroneous	erroneous
`qsig`	Running subjobs	Running subjobs in specified range	Specified subjob
`qstat`	Array object	Specified range of subjobs	Specified subjob
`tracejob`	erroneous	erroneous	Specified subjob

8.6.1 Deleting a Job Array

The `qdel` command will take a job array identifier, subjob identifier or job array range. The indicated object(s) are deleted, including any currently running subjobs. Running subjobs are treated like running jobs. Subjobs not running are deleted and never run.

By default, one email is sent per deleted subjob, so deleting a job array of 5000 subjobs results in 5000 emails being sent, unless you are suppressing the number of emails sent. See "-Wsuppress_email=<N>" on page 141 of the PBS Professional Reference Guide.

8.6.2 Altering a Job Array

The `qalter` command can only be used on a job array object, not on subjobs or ranges. Job array attributes are the same as for jobs.

To modify the max_run_subjobs attribute, use `qalter -Wmax_run_subjobs=<new value> <job ID>`.

8.6.3 Moving a Job Array

The `qmove` command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the '*Q*', '*H*', or '*W*' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a `qstat` on the server from which the job array was moved does not show the job array. A `qstat` on the job array object is redirected to the new server.

8.6.4 Holding a Job Array

The `qhold` command can only be used with job array objects, not with subjobs or ranges. A hold can be applied to a job array only from the '*Q*', '*B*' or '*W*' states. This puts the job array in the '*H*', held, state. If any subjobs are running, they will run to completion. No queued subjobs are started while in the '*H*' state.

If a job array has subjobs that have a System hold, the job array also gets a System hold.

8.6.5 Releasing a Job Array

The `qrls` command can be used directly only with job array objects, not with subjobs or ranges. If the job array was in the '*Q*' or '*B*' state, it is returned to that state. If it was in the '*W*' state, it is returned to that state, unless its waiting time was reached, in which case it goes to the '*Q*' state.

You can use `qrls` indirectly on subjobs. If you use `qrls` on a job array, and that job array has a System hold because it has subjobs(s) with a System hold, the subjobs that were held with a System hold are released, then the System hold on the job array is released (you'll need Manager, root, or PBS Administrator privilege for this).

8.6.6 Selecting Job Arrays

The default behavior of `qselect` is to return the job array identifier, without returning subjob identifiers.

The `qselect` command does not return any job arrays when the state selection (`-s`) option restricts the set to '*R*', '*S*', '*T*' or '*U*', because a job array will never be in any of these states. However, you can use `qselect` to return a list of subjobs by using the `-t` option.

You can combine options to `qselect`. For example, to restrict the selection to subjobs, use both the `-J` and the `-T` options. To select only running subjobs, use `-J -T -sR`.

Table 8-6: Options to qselect for Job Arrays

Option	Selects	Result
(none)	jobs, job arrays	Shows job and job array identifiers
-J	job arrays	Shows only job array identifiers
-T	jobs, subjobs	Shows job and subjob identifiers

8.6.7 Ordering Job Arrays in the Queue

The `qorder` command can only be used with job array objects, not on subjobs or ranges. This changes the queue order of the job array in association with other jobs or job arrays in the queue.

8.6.8 Requeueing a Job Array

The `qrerun` command will take a job array identifier, subjob identifier or job array range. If a job array identifier is given as an argument, it is returned to its initial state at submission time, or to its altered state if it has been qaltered. All of that job array's subjobs are requeued, which includes those that are currently running, and those that are completed and deleted. If a subjob or range is given, those subjobs are requeued as jobs would be.

8.6.9 Signaling a Job Array

If a job array object, subjob or job array range is given to `qsig`, all currently running subjobs within the specified set are sent the signal.

8.6.10 Sending Messages to Job Arrays

The `qmsg` command is not supported for job arrays.

8.6.11 Getting Log Data on Job Arrays

The `tracejob` command can be run on job arrays and individual subjobs. When `tracejob` is run on a job array or a subjob, the same information is displayed as for a job, with additional information for a job array. Note that subjobs do not exist until they are running, so `tracejob` will not show any information until they are. When `tracejob` is run on a job array, the information displayed is only that for the job array object, not the subjobs. Job arrays themselves do not produce any MoM log information. Running `tracejob` on a job array gives information about why a subjob did not start.

8.6.12 Caveats for Using PBS Commands with Job Arrays

8.6.12.1 Shells and PBS Commands with Job Arrays

Some shells such as `csh` and `tcsh` use the square bracket ("[", "]") as a metacharacter. When using one of these shells, and a PBS command taking subjobs, job arrays or job array ranges as arguments, the subjob, job array or job array range must be enclosed in double quotes.

8.7 Job Array Caveats

8.7.1 Job Arrays Required to be Rerunnable

Job arrays are required to be rerunnable, and are rerunnable by default.

8.7.2 Resources Same for All Subjobs

You cannot combine jobs into an array that have different hardware requirements, i.e. different select statements.

8.7.3 Checkpointing Not Supported for Job Arrays

Checkpointing is not supported for job arrays. On systems that support checkpointing, subjobs are not checkpointed, instead they run to completion.

8.7.4 Caveats for Job Array Exit Status

Subjob exit status is available only as long as the subjob is in job history. When a subjob is not in job history, a failed or terminated subjob will show an exit status of *Finished*, instead of failed or terminated.

9 Working with PBS Jobs

9.1 Using Job History

PBS Professional can provide job history information, including what the submission parameters were, whether the job started execution, whether execution succeeded, whether staging out of results succeeded, and which resources were used.

PBS can keep job history for jobs which have finished execution, were deleted, or were moved to another server.

9.1.1 Definitions

Moved jobs

Jobs which were moved to another server

Finished jobs

Jobs whose execution is done, for any reason:

- Jobs which finished execution successfully and exited
- Jobs terminated by PBS while running
- Jobs whose execution failed because of system or network failure
- Jobs which were deleted before they could start execution

9.1.2 Job History Information

PBS can keep all job attribute information, including the following:

- Submission parameters
- Whether the job started execution
- Whether execution succeeded
- Whether staging out of results succeeded
- Which resources were used

PBS keeps job history for the following jobs:

- Jobs that are running at another server
- Jobs that have finished execution
- Jobs that were deleted
- Jobs that were moved to another server

While a job is running, you can see information about it. After a job has finished or been deleted, its history information is preserved for the specified duration. The administrator chooses a duration for preservation of job history information after each job has finished or been deleted. PBS periodically checks each finished job, and deletes job history for those whose history has been preserved for longer than the specified duration.

Subjobs are not considered finished jobs until the parent array job is finished, which happens when all of its subjobs have terminated execution.

9.1.2.1 Working With Moved Jobs

You can use the following commands with moved jobs. They will function as they do with normal jobs.

```
qalter
qhold
qmove
qmsg
qorder
qrerun
qrls
qrun
qsig
```

While a moved job is running, its state is *M*. When a moved job finishes, its substate becomes *92*. See "Job States" on page 361 of the PBS Professional Reference Guide.

9.1.2.2 PBS Commands and Finished Jobs

The commands listed above cannot be used with finished jobs, whether they finished at the local server or a remote server. These jobs are no longer running; PBS is storing their information, and this information cannot be altered. Trying to use one of the above commands with a finished job results in the following error message:

```
<command name>: Job <job ID> has finished
```

9.2 Modifying Job Attributes

Most attributes can be changed by the owner of the job (or a manager or operator) while the job is still queued. However, once a job begins execution, the only values that can be modified are cputime, walltime, and run_count. You can decrease walltime, and you can increase run_count.

When the `qalter -l` option is used to alter the resource list of a queued job, it is important to understand the interactions between altering the select directive and job limits.

If the job was submitted with an explicit "`-l select=`", then vnode-level resources must be qaltered using the "`-l select=`" form. In this case a vnode level resource RES cannot be qaltered with the "`-l <resource>`" form.

For example:

Submit the job:

```
% qsub -l select=1:ncpus=2:mem=512mb jobscript
```

Job's ID is 230

`qalter` the job using "-l RES" form:

```
% qalter -l ncpus=4 230
```

Error reported by `qalter`:

```
qalter: Resource must only appear in "select"
```

specification when select is used: ncpus 230

qalter the job using the "-l select=" form:

```
% qalter -l select=1:ncpus=4:mem=512mb 230
```

No error reported by qalter:

```
%
```

9.2.1 Changing the Selection Directive

If the selection directive is altered, the job limits for any consumable resource in the directive are also modified.

For example, if a job is queued with the following resource list:

```
select=2:ncpus=1:mem=5gb
```

job limits are set to ncpus=2, mem=10gb.

If the select statement is altered to request:

```
select=3:ncpus=2:mem=6gb
```

then the job limits are reset to ncpus=6 and mem=18gb

9.2.2 Changing the Job-wide Limit

If the job-wide limit is modified, the corresponding resources in the selection directive are not modified. It would be impossible to determine where to apply the changes in a compound directive.

Reducing a job-wide limit to a new value less than the sum of the resource in the directive is strongly discouraged. This may produce a situation where the job is aborted during execution for exceeding its limits. The actual effect of such a modification is not specified.

A job's walltime may be altered at any time, except when the job is in the *Exiting* state, regardless of the initial value.

If a job is queued, requested modifications must still fit within the queue's and server's job resource limits. If a requested modification to a resource would exceed the queue's or server's job resource limits, the resource request will be rejected.

Resources are modified by using the -l option, either in chunks inside of selection statements, or in job-wide modifications using resource_name=value pairs. The selection statement is of the form:

-l select=[N:]chunk[+[N:]chunk ...]

where N specifies how many of that chunk, and a chunk is of the form:

<resource name>=<value>[:<resource name>=<value> ...]

Job-wide *<resource name>=<value>* modifications are of the form:

-l <resource name>=<value>[,<resource name>=<value> ...]

Placement of jobs on vnodes is changed using the place statement:

-l place=<modifier>[:<modifier>]

where *modifier* is any combination of *group*, *excl*, *exclhost*, and/or one of *free*|*pack*|*scatter*|*vscatter*.

The usage syntax for qalter is:

qalter <job resources> <job list>

The following examples illustrate how to use the `qalter` command. First we list all the jobs of a particular user. Then we modify two attributes as shown (increasing the wall-clock time from 20 to 25 minutes, and changing the job name from "airfoil" to "engine"):

```
qstat -u barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:16 R 0:01
54.south barry  workq airfoil  --   --   1  -- 0:20 Q  --

qalter -l walltime=20:00 -N engine 54
qstat -a 54
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south barry  workq engine   --   --   1  -- 0:25 Q  --
```

The `qalter` command can be used on job arrays, but not on subjobs or ranges of subjobs. When used with job arrays, any job array identifiers must be enclosed in double quotes, e.g.:

```
qalter -l walltime=25:00 "1234[].south"
```

You cannot use the `qalter` command (or any other command) to alter a custom resource which has been created to be invisible or unrequestable. See section 4.3.8, “Caveats and Restrictions on Requesting Resources”, on page 57.

For more information, see “qalter” on page 127 of the PBS Professional Reference Guide.

9.2.2.1 Caveats

Be careful when using a Boolean resource as a job-wide limit.

9.3 Deleting Jobs

PBS provides the `qdel` command for deleting jobs. The `qdel` command deletes jobs in the order in which their job identifiers are presented to the command. A batch job may be deleted by its owner, a PBS operator, or a PBS administrator. Unless you are an administrator or an operator, you can delete only your own jobs.

To delete a queued, held, running, or suspended job:

qdel <job ID>

Example:

```
qdel 51
qdel 1234[].server
```

Job array identifiers must be enclosed in double quotes.

9.3.1 Deleting Jobs with Force

You can delete a job whether or not its execution host is reachable, and whether or not it is in the process of provisioning:

qdel -W force <job ID>

9.3.2 Deleting Finished Jobs

By default, the qdel command does not affect finished jobs. You can use the qdel -x option to delete job histories. This option also deletes any specified jobs that are queued, running, held, suspended, finished, or moved. When you use this, you are deleting the job and its history in one step. If you use the qdel command without the -x option, you delete the job, but not the job history, and you cannot delete a finished job.

To delete a finished job, whether or not it was moved:

qdel -x <job ID>

If you try to delete a finished job without the -x option, you will get the following error:

```
qdel: Job <job ID> has finished
```

9.3.3 Deleting Moved Jobs

You can use the qdel -x option to delete jobs that are queued, running, held, suspended, finished, or moved.

To delete a job that was moved:

qdel <job ID sequence number>.<original server>

To delete a job that was moved, and then finished:

qdel -x <job ID>

9.3.4 Restricting Number of Emails

By default, mail is sent for each job or subjob you delete. Use the following option to qdel to specify a limit on emails sent:

qdel -Wsuppress_email=<N>

See section 2.5.1.3, “Restricting Number of Job Deletion Emails”, on page 27.

9.4 Sending Messages to Jobs

To send a message to a job is to write a message string into one or more output files of the job. Typically this is done to leave an informative message in the output of the job. Such messages can be written using the qmsg command.

You can send messages to running jobs only.

The usage syntax of the qmsg command is:

qmsg [-E][-O] <message string> <job ID>

Example:

```
qmsg -O "output file message" 54
qmsg -O "output file message" "1234[].server"
```

Job array identifiers must be enclosed in double quotes.

The -E option writes the message into the error file of the specified job(s). The -O option writes the message into the output file of the specified job(s). If neither option is specified, the message will be written to the error file of the job.

The first operand, *message_string*, is the message to be written. If the string contains blanks, the string must be quoted. If the final character of the string is not a newline, a newline character will be added when written to the job's file. All remaining operands are job IDs which specify the jobs to receive the message string. For example:

```
qmsg -E "hello to my error (.e) file" 55
qmsg -O "hello to my output (.o) file" 55
qmsg "this too will go to my error (.e) file" 55
```

9.5 Sending Signals to Jobs

You can use the `qsig` command to send a signal to your job. The signal is sent to all of the job's processes.

Usage syntax of the `qsig` command is:

qsig [-s <signal>] <job ID>

Job array *job IDs* must be enclosed in double quotes.

If the `-s` option is not specified, SIGTERM is sent. If the `-s` option is specified, it declares which *signal* is sent to the job. The *signal* argument is either a signal name, e.g. SIGKILL, the signal name without the *SIG* prefix, e.g. *KILL*, or an unsigned signal number, e.g. *9*. The signal name SIGNULL is allowed; the server will send the signal 0 to the job which will have no effect. Not all signal names will be recognized by `qsig`. If it doesn't recognize the signal name, try issuing the signal number instead. The request to signal a batch job will be rejected if:

- You are not authorized to signal the job
- The job is not in the running state
- The requested signal is not supported by the execution host
- The job is exiting
- The job is provisioning

Two special signal names, "suspend" and "resume", (note, all lower case), are used to suspend and resume jobs. When suspended, a job continues to occupy system resources but is not executing and is not charged for walltime. Manager or operator privilege is required to suspend or resume a job.

The signal TERM is useful, because it is ignored by shells, but you can trap it and do useful things such as write out status.

The three examples below all send a signal 9 (SIGKILL) to job 34:

```
qsig -s SIGKILL 34
qsig -s KILL 34
```

If you want to trap the signal in your job script, the signal must be trapped by all of the job's shells.

On most Linux systems the command "`kill -l`" (that's 'minus ell') will list all the available signals.

9.6 Changing Order of Jobs

PBS provides the `qorder` command to change the order of two jobs, within or across queues. To order two jobs is to exchange the jobs' positions in the queue or queues in which the jobs reside. If job1 is at position 3 in queue A and job2 is at position 4 in queue B, qordering them will result in job1 being in position 4 in queue B and job2 being in position 3 in queue A.

No attribute of the job (such as Priority) is changed. The impact of changing the order within the queue(s) is dependent on local job scheduling policy; contact your systems administrator for details.

Usage of the `qorder` command is:

qorder <job ID>1 <job ID2>

Job array identifiers must be enclosed in double quotes.

Both operands are *job IDs* which specify the jobs to be exchanged.

```
qstat -u bob
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --

qorder 54 "63[]"
qstat -u bob
                                               Req'd     Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
63[].south bob  workq airfoil  --   --   1  -- 0:13 Q  --
54.south bob    workq twinkie  --   --   1  -- 0:20 Q  --
```

9.6.1 Restrictions

- The two jobs must be located at the same server, and both jobs must be owned by you. However, a PBS Manager or Operator can exchange any jobs.
- A job in the running state cannot be reordered.
- The `qorder` command can be used with entire job arrays, but not on subjobs or ranges. Reordering a job array changes the queue order of the job array in relation to other jobs or job arrays in the queue.

9.7 Moving Jobs Between Queues

PBS provides the `qmove` command to move jobs between different queues (even queues on different servers). To move a job is to remove the job from the queue in which it resides and instantiate the job in another queue.

A job in the running state cannot be moved.

The usage syntax of the `qmove` command is:

qmove <destination> <job ID(s)>

Job array *<job ID>s* must be enclosed in double quotes.

The first operand is the new destination for

```
<queue>
@<server>
<queue>@<server>
```

If the *destination* operand describes only a queue, then `qmove` will move jobs into the queue of the specified name at the job's current server. If the *destination* operand describes only a server, then `qmove` will move jobs into the default queue at that server. If the *destination* operand describes both a queue and a server, then `qmove` will move the jobs into the specified queue at the specified server. All following operands are *job IDs* which specify the jobs to be moved to the new *destination*.

The `qmove` command can only be used with job array objects, not with subjobs or ranges. Job arrays can only be moved from one server to another if they are in the '*Q*', '*H*', or '*W*' states, and only if there are no running subjobs. The state of the job array object is preserved in the move. The job array will run to completion on the new server.

As with jobs, a `qstat` on the server from which the job array was moved will not show the job array. A `qstat` on the job array object will be redirected to the new server.

The subjob accounting records will be split between the two servers.

10 Checking Job & System Status

10.1 Checking Job Status

You can use the `qstat` command to view job information in the following formats:

- Basic format: minimal summary of jobs
- Alternate format: intermediate listing of job information
- Long format: shows all information about jobs

You can see only the information for which you have the required privilege.

We discuss each format in the following sections. See "qstat" on page 197 of the PBS Professional Reference Guide.

By default, `qstat` displays information for queued or running jobs only. However, you can tell `qstat` to display information for all jobs, whether they are running, queued, finished, or moved; we cover this in this chapter. Job history is kept for a period defined by your administrator.

Summary of usage:

Displaying Job Status

Default format:

qstat [-E] [-J] [-p] [-t] [-x] [[<job ID> | <destination>] ...]

Long format:

qstat -f [-F json|dsv [-D <delimiter>]] [-E] [-J] [-p] [-t] [-w] [-x] [[<job ID> | <destination>] ...]

Alternate format:

qstat [-a [-w]| -H | -i | -r] [-E] [-G | -M] [-J] [-n [-1] [-w]] [-s [-1] [-w]] [-t] [-T [-w]] [-u <user list>] [[<job ID> | <destination>] ...]

Displaying Queue Status

Default format:

qstat -Q [<destination> ...]

Long format:

qstat -Q -f [-F json|dsv [-D <delimiter>]] [-w] [<destination> ...]

Alternate format:

qstat -q [-G | -M] [<destination> ...]

Displaying Server Status

Default format:

qstat -B [<server> ...]

Long format:

qstat -B -f [-F json|dsv [-D <delimiter>]] [-w] [<server> ...]

qstat --version

10.1.1 Specifying Jobs to View

You can specify that you want information for a job identifier, a list of job identifiers, or all of the jobs at a destination.

To specify a job identifier, it must be in the following form:

<sequence number>[.<server>][@<server>]

where *<sequence number>.<server>* is the job identifier assigned at submission. If you do not specify.*<server>*, the default server is used. If *@<server>* is supplied, the request will be for the job identifier currently at that server.

If you specify a destination identifier, it takes one of the following three forms:

<queue>

@<server>

<queue>@<server>

If you specify *<queue>*, the request is for status of all jobs in that queue at the default server.

If you use the *@<server>* form, the request is for status of all jobs at that server.

If you specify a full destination identifier, *<queue>@<server>*, the request is for status of all jobs in the named *queue* at the named *server*.

10.1.2 Viewing Default Job Status

You can use the `qstat` command to view basic job status, in the default format.

Syntax for simple form and with options:

qstat

qstat [-E] [-J] [-p] [-t] [-x] [[<job ID> | <destination>] ...]

The default display shows the following information:

- The job identifier assigned by PBS
- The job name given by the submitter
- The job owner
- The CPU time used
- The job state; see "Job States" on page 361 of the PBS Professional Reference Guide.
- The queue in which the job resides

The following example illustrates the default display of `qstat`.

```
qstat
Job id    Name         User         Time Use S Queue
--------- ---------- ---------- -------- - -----
16.south  aims14      user1              0 H workq
18.south  aims14      user1              0 W workq
26.south  airfoil     barry       00:21:03 R workq
27.south  airfoil     barry       21:09:12 R workq
28.south  myjob       user1              0 Q workq
29.south  tns3d       susan              0 Q workq
30.south  airfoil     barry              0 Q workq
31.south  seq_35_3    donald             0 Q workq
```

10.1.3 Viewing Job Status in Alternate Format

You can use the `qstat` command to view more detail than the basic job information, in the alternate format.

Syntax for simple form and with options:

qstat -a

qstat [-a [-w]| -H | -i | -r] [-E] [-G | -M] [-J] [-n [-1] [-w]] [-s [-1] [-w]] [-t] [-T [-w]] [-u <user list>] [[<job ID> | <destination>] ...]

The alternate display shows the following fields:

- Job ID
- Job owner
- Queue in which job resides
- Job name
- Session ID (only appears when job is running)
- Number of chunks or vnodes requested
- Number of CPUs requested
- Amount of memory requested
- Amount of CPU time requested, if CPU time requested; if not, amount of wall clock time requested
- State of job
- Amount of CPU time elapsed, if CPU time requested; if not, amount of wall clock time elapsed

```
qstat -a

                                                  Req'd  Elap
Job ID   User   Queue Jobname Ses NDS TSK Mem Time S Time
-------- ------ ----- ------- --- --- --- --- ---- - ----
16.south user1  workq aims14  --  --   1  --  0:01 H  --
18.south user1  workq aims14  --  --   1  --  0:01 W  --
51.south barry  workq airfoil 930 --   1  --  0:13 R 0:01
52.south user1  workq myjob   --  --   1  --  0:10 Q  --
53.south susan  workq tns3d   --  --   1  --  0:20 Q  --
54.south barry  workq airfoil --  --   1  --  0:13 Q  --
55.south donald workq seq_35_ --  --   1  --  2:00 Q  --
```

You can use the `-1` option to reformat `qstat` output to a single line. This option can only be used in conjunction with the `-n` and/or `-s` options.

10.1.3.1 Display Size in Gigabytes

The "-G" option to qstat displays all jobs at the requested or default server using the alternative display, showing all size information in gigabytes (GB) rather than the default of smallest displayable units. Note that if the size specified is less than 1 GB, then the amount if rounded up to 1 GB. For example:

```
qstat -G
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1    1gb     --  Q     --
```

10.1.3.2 Display Size in Megawords

The "-M" option to qstat displays all jobs at the requested (or default) server using the alternative display, showing all size information in megawords (MW) rather than the default of smallest displayable units. A word is considered to be 8 bytes. For example:

```
qstat -M
host1:
                                                  Req'd  Req'd     Elap
Job ID     User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
---------- ----- ----- ------- ---- --- ---  -----  -----  -  -----
43.host1   user1 workq STDIN   4693   1   1     --     --  R  00:05
44[].host1 user1 workq STDIN     --   1   1     --     --  Q     --
45.host1   user1 workq STDIN     --   1   1   25mw     --  Q     --
```

10.1.4 Viewing Job Status in Long Format

You can use the qstat command to view all of the information about a job, including values for its attributes and resources, in the long format.

Syntax for simple form and with options:

qstat -f

qstat -f [-F json|dsv [-D <delimiter>]] [-E] [-J] [-p] [-t] [-w] [-x] [[<job ID> | <destination>] ...]

You can see job status in JSON or delimiter-separated value formats; see section 10.4, "Full Display Options for Job, Queue, and Server Status", on page 186. The long format shows the following fields, including job attributes. See "Job Attributes" on page 328 of the PBS Professional Reference Guide for a description of attributes:

```
qstat -f 13
Job Id: 13.host1
    Job_Name = STDIN
    Job_Owner = user1@host2
    resources_used.cpupercent = 0
    resources_used.cput = 00:00:00
    resources_used.mem = 2408kb
    resources_used.ncpus = 1
    resources_used.vmem = 12392kb
    resources_used.walltime = 00:01:31
    job_state = R
    queue = workq
    server = host1
    Checkpoint = u
    ctime = Thu Apr  2 12:07:05 2010
    Error_Path = host2:/home/user1/STDIN.e13
    exec_host = host2/0
    exec_vnode = (host3:ncpus=1)
    Hold_Types = n
    Join_Path = n
    Keep_Files = n
    Mail_Points = a
    mtime = Thu Apr  2 12:07:07 2010
    Output_Path = host2:/home/user1/STDIN.o13
    Priority = 0
    qtime = Thu Apr  2 12:07:05 2010
    Rerunable = True
    Resource_List.ncpus = 1
    Resource_List.nodect = 1
    Resource_List.place = free
    Resource_List.select = host=host3
    stime = Thu Apr  2 12:07:08 2010
    session_id = 32704
    jobdir = /home/user1
    substate = 42
    Variable_List = PBS_O_HOME=/home/user1,PBS_O_LANG=en_US.UTF-8,
        PBS_O_LOGNAME=user1,
        PBS_O_PATH=/opt/gnome/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/X11R
        6/bin:/bin:/usr/games:/opt/gnome/bin:/opt/kde3/bin:/usr/lib/mit/bin:/us
        r/lib/mit/sbin,PBS_O_MAIL=/var/mail/root,PBS_O_SHELL=/bin/bash,
        PBS_O_HOST=host2,PBS_O_WORKDIR=/home/user1,PBS_O_SYSTEM=Linux,
        PBS_O_QUEUE=workq
    comment = Job run at Thu Apr 02 at 12:07 on (host3:ncpus=1)
```

```
alt_id = <dom0:job ID xmlns:dom0="http://schemas.microsoft.com/HPCS2008/hpcb
    p">149</dom0:Job ID>
etime = Thu Apr  2 12:07:05 2010
Submit_arguments = -lselect=host=host3 -- ping -n 100 127.0.0.1
executable = <jsdl-hpcpa:Executable>ping</jsdl-hpcpa:Executable>
argument_list = <jsdl-hpcpa:Argument>-n</jsdl-hpcpa:Argument><jsdl-hpcpa:Ar
    gument>100</jsdl-hpcpa:Argument><jsdl-hpcpa:Argument>127.0.0.1</jsdl-hp
    cpa:Argument>
```

10.1.4.1 Path Display under Windows

When you view a job in long format that was submitted from a mapped drive, PBS displays the UNC path for the job's Output_Path, Error_Path attributes, and the value for PBS_O_WORKDIR in the job's Variable_List attribute.

When you view a job in long format that was submitted using UNC paths for output and error files, PBS displays the UNC path for the job's Output_Path and Error_Path attributes.

10.1.5 Listing Jobs by User

The "-u" option to `qstat` displays jobs owned by any of a list of user names you specify.

Syntax:

qstat -u <user name>[@<host>][,<user name>[@<host>],...]

Host names are not required, and may be "wild carded" on the left end, e.g. "**.mydomain.com*". *<user name>* without a "*@<host>*" is equivalent to "*<user name>@**", that is at any host.

```
qstat -u user1
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --

qstat -u user1,barry
                                                  Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
51.south barry  workq airfoil  930  --   1  -- 0:13 R 0:01
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

10.1.6 Listing Running Jobs

The "-r" option to qstat displays the status of all running jobs at the (optionally specified) PBS server. Running jobs include those that are running and suspended. One line of output is generated for each job reported, and the information is presented in the alternative display. For example:

```
qstat -r
host1:
                                                  Req'd  Req'd     Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem    Time  S  Time
-------- ----- ----- ------- ---- --- --- -----  ----- -  -----
43.host1 user1 workq STDIN   4693   1   1   --     --  R  00:00
```

10.1.7 Listing Non-Running Jobs

The "-i" option to qstat displays the status of all non-running jobs at the (optionally specified) PBS server. Non-running jobs include those that are queued, held, and waiting. One line of output is generated for each job reported, and the information is presented in the alternative display (see description above). For example:

```
qstat -i
host1:
                                                  Req'd  Req'd     Elap
Job ID      User  Queue Jobname Sess NDS TSK  Mem    Time   S  Time
----------  ----- ----- ------- ---- --- ---  -----  -----  -  -----
44[].host1  user1 workq STDIN     --   1   1     --     --  Q     --
```

10.1.8 Listing Hosts Assigned to Jobs

The "-n" option to qstat displays the hosts allocated to any running job at the (optionally specified) PBS server, in addition to the other information presented in the alternative display. The exec_host information is printed immediately below the job (see job 51 in the example below), and includes the host name and number of virtual processors assigned to the job (i.e. "south/0", where "south" is the host name, followed by the virtual processor(s) assigned.). A text string of "--" is printed for non-running jobs. Notice the differences between the queued and running jobs in the example below:

```
qstat -n
                                                    Req'd  Elap
Job ID   User   Queue Jobname Sess NDS TSK Mem Time S Time
-------- ------ ----- ------- ---- --- --- --- ---- - ----
16.south user1  workq aims14   --   --   1  -- 0:01 H  --
 --
18.south user1  workq aims14   --   --   1  -- 0:01 W  --
 --
51.south barry  workq airfoil  930  --   1  -- 0:13 R  0:01
 south/0
52.south user1  workq my_job   --   --   1  -- 0:10 Q  --
 --
```

10.1.9 Displaying Job Comments

The "-s" option to qstat displays the job comments, in addition to the other information presented in the alternative display. The job comment is printed immediately below the job. By default the job comment is updated by the scheduler with the reason why a given job is not running, or when the job began executing. A text string of "--" is printed for jobs whose comment has not yet been set. The example below illustrates the different type of messages that may be displayed:

```
qstat -s
                                                  Req'd  Elap
Job ID   User  Queue Jobname Sess NDS TSK Mem Time S Time
-------- ----- ----- ------- ---- --- --- --- ---- - ----
16.south user1 workq aims14   --   --   1  -- 0:01 H  --
    Job held by user1 on Wed Aug 22 13:06:11 2004
18.south user1 workq aims14   --   --   1  -- 0:01 W  --
    Waiting on user requested start time
51.south barry workq airfoil  930  --   1  -- 0:13 R 0:01
    Job run on host south - started Thu Aug 23 at 10:56
52.south user1 workq my_job   --   --   1  -- 0:10 Q  --
    Not Running: No available resources on nodes
57.south susan workq solver   --   --   2  -- 0:20 Q  --
    --
```

10.1.10 Showing State of Job, Job Array or Subjob

The "-t" option to qstat will show the state of a job, a job array object, and all non-X subjobs.

The "-J" option to qstat will show only the state of job arrays.

The combination of "-J" and "-t" options to qstat will show only the state of subjobs.

For example:

```
qstat -t
Job ID          Name      User          Time Use  S  Queue
----------      --------  -----------   --------  -  -----
44[].host1      STDIN     user1                0  B  workq
44[1].host1     STDIN     user1         00:00:00  R  workq
44[2].host1     STDIN     user1                0  Q  workq
44[3].host1     STDIN     user1                0  Q  workq

qstat -J
Job ID          Name      User          Time Use  S  Queue
-----------     --------  -----------   --------  -  -----
44[].host1      STDIN     user1                0  B  workq

$ qstat -Jt
Job ID           Name      User          Time Use  S  Queue
-----------      --------  -----------   --------  -  -----
44[1].host1     STDIN     user1         00:00:00  R  workq
44[2].host1     STDIN     user1                0  Q  workq
44[3].host1     STDIN     user1                0  Q  workq
```

10.1.11 Printing Job Array Percentage Completed

The "-p" option to qstat prints the default display, with a column for Percentage Completed. For a job array, this is the number of subjobs completed and deleted, divided by the total number of subjobs. For example:

```
qstat -p
Job ID            Name      User            % done  S  Queue
-------------     --------  -----------   --------  -  -----
44[].host1        STDIN     user1               40  B  workq
```

10.1.12 Viewing Job Start Time

There are two ways you can find the job's start time. If the job is still running, you can do a qstat -f and look for the stime attribute. If the job has finished, you look in the accounting log for the *S* record for the job. For an array job, only the *S* record is available; array jobs do not have a value for the stime attribute.

10.1.13 Viewing Estimated Start Times For Jobs

You can view the estimated start times and vnodes of jobs using the qstat command. If you use the -T option to qstat when viewing job information, the *Elap Time* field is replaced with the *Est Start Time* field. Running jobs are shown above queued jobs. Running jobs are sorted by their stime attribute (start time).

Queued jobs whose estimated start times are unset (estimated.start_time = *unset*) are displayed after those with estimated start times, with estimated start time shown as a double dash ("--"). Queued jobs with estimated start times in the past are treated as if their estimated start times are unset.

Time displayed is local to the qstat command. Current week begins on Sunday.

If the estimated start time or vnode information is invisible to unprivileged users, no estimated start time or vnode information is available via `qstat`.

Example output:

```
qstat -T
                                                                  Est
                                                  Req'd  Req'd   Start
Job ID   Username Queue Jobname SessID NDS TSK Memory Time  S Time
-------  -------- ----- -------- ----- --- --- ------ ----- - -----
5.host1   user1   workq  foojob  12345  1   1  128mb 00:10  R   --
9.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  11:30
10.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  Tu 15
7.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  Jul
8.host1   user1   workq  foojob   --    1   1  128mb 00:10  Q  2010
11.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q  >5yrs
13.host1  user1   workq  foojob   --    1   1  128mb 00:10  Q   --
```

If the start time for a job cannot be estimated, the start time is shown as a question mark ("?").

10.1.13.1 Why Does Estimated Start Time Change?

The estimated start time for your job may change for the following reasons:

- Changes to the system, such as vnodes going down, or the administrator offlining vnodes
- A higher priority job coming into the system, or a shift in priority of the existing jobs

10.1.14 Viewing Job Status in Wide Format

The `-w qstat` option displays job status in wide format. The total width of the display is extended from 80 characters to 120 characters. The Job ID column can be up to 30 characters wide, while the Username, Queue, and Jobname column can be up to 15 characters wide. The SessID column can be up to eight characters wide, and the NDS column can be up to four characters wide.

You can use this option only with the `-a`, `-n`, or `-s qstat` options.

This option is different from the `-w` option used with `-f`.

10.1.15 Viewing Information for Finished and Moved Jobs

You can view information for finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being stored by PBS.

The `-x` option to the `qstat` command allows you to see information for all jobs, whether they are running, queued, finished or moved. This information is presented in standard format.

You can view the history for selected sets of jobs:

Linux:

```
qstat -fx `qselect -x -s "MF"`
```

Windows:

```
for /F "usebackq" %%j in (`"\Program Files\ PBSPro\ exec\ bin\qselect" -x -s MF`)
    do ("\Program Files\PBS\exec\bin\qstat" -fx %%j)
```

10.1.15.1 Getting Information on Jobs Moved to Another Server

If your job is running at another server, you can use the `qstat` command to see its status. If your site is using peer scheduling, your job may be moved to a server that is not your default server. In this case, to see information on your job, you can use any of the following methods:

- Use `qstat -x` to see information about all jobs, whether running, queued, finished, or moved; you can specify job IDs
- Give the job ID as an argument to `qstat`. If you use only "`qstat`", your job will not appear to exist. For example: you submit a job to ServerA, and it returns the job ID as "123.ServerA". Then 123.ServerA is moved to ServerB. In this case, use

  `qstat 123`

  or

  `qstat 123.ServerA`

  to get information about your job. ServerA will query ServerB for the information. To list all jobs at ServerB, you can use:

  `qstat @<server>B`

  If you use "`qstat`" without the job ID, the job will not appear to exist.

10.1.15.2 Job History In Standard Format

You can use the `-x` option to the `qstat` command to see information for finished, moved, queued, and running jobs, in standard format.

Usage:

qstat -x

Displays information for queued, running, finished, and moved jobs, in standard format.

qstat -x <job ID>

Displays information for a job, regardless of its state, in standard format.

Example 10-1: Showing finished and moved jobs with queued and running jobs:

```
qstat -x
Job id            Name         User    Time Use  S   Queue
------------      ----------  ------  -------  ---  ------
101.server1       STDIN        user1   00:00:00  F   workq
102.server1       STDIN        user1   00:00:00  M   destq@server2
103.server1       STDIN        user1   00:00:00  R   workq
104.server1       STDIN        user1   00:00:00  Q   workq
```

To see status for jobs, job arrays and subjobs that are queued, running, finished, and moved, use `qstat -xt`.

To see status for job arrays that are queued, running, finished, or moved, use `qstat -xJ`.

When information for a moved job is displayed, the destination queue and server are shown as <queue>@<server>.

Example 10-2: `qstat -x` output for moved job: destination queue is `destq`, and destination server is `server2`.

```
Job id            Name         User    Time Use  S   Queue
----------------  -----------  -----   -------  ---  ------
101.sequoia       STDIN        user1   00:00:00  F   workq
102.sequoia       STDIN        user1   00:00:00  M   destq@server2
103.sequoia       STDIN        user1   00:00:00  R   workq
```

Example 10-3: Viewing moved job:

- There are three servers with hostnames ServerA, ServerB, and ServerC
- User1 submits job 123 to ServerA.
- After some time, User1 moves the job to ServerB.
- After more time, the administrator moves the job to QueueC at ServerC.

This means:

- The `qstat` command will show QueueC@ServerC for job 123.

10.1.15.3 Job History In Alternate Format

You can use the -H option to the `qstat` command to see job history for finished or moved jobs in alternate format. This does not display running or queued jobs.

Usage:

qstat -H

Displays information for finished or moved jobs, in alternate format

qstat -H <job ID>

Displays information for that job in alternate format, whether or not it is finished or moved

qstat -H <destination>

Displays information for finished or moved jobs at that destination

Example 10-4: Job history in alternate format:

```
qstat -H
                                                  Req'd  Req'd   Elap
Job ID  Username  Queue Jobname  SessID  NDS TSK Memory Time  S Time
------  --------  ----  -------  ------  --- --- ------ ----  -- -----
101.S1  user1     workq STDIN    5168     1  1   --     --    F  00:00
102.S1  user1     Q1@S2 STDIN     --      1  2   --     --    M   --
```

To see alternate-format status for jobs, job arrays and subjobs that are finished and moved, use `qstat -Ht`.

To see alternate-format status for job arrays that are finished or moved, use `qstat -HJ`.

The -H option is incompatible with the -a, -i and -r options.

10.1.16 Grouping Jobs and Sorting by ID

You can use the -E option to sort and group jobs in the output of qstat. The -E option groups jobs by server and displays each group by ascending ID. This option also improves qstat performance. The following table shows how the -E option affects the behavior of qstat:

Table 10-1: How -E Option Affects qstat Output

How qstat is Used	Result Without -E	Result With -E
qstat (no job ID specified)	Queries the default server and displays result	No change in behavior; same as without -E option
qstat <list of job IDs from single server>	Displays results in the order specified	Displays results in ascending ID order
qstat <job IDs at multiple servers>	Displays results in the order they are specified	Groups jobs by server. Displays each group in ascending order

10.1.17 Caveats for Job Information

- MoM periodically polls jobs for usage by the jobs running on her host, collects the results, and reports this to the server. When a job exits, she polls again to get the final tally of usage for that job.

  For example, MoM polls the running jobs at times T1, T2, T4, T8, T16, T24, and so on.

  The output shown by a qstat during the window of time between T8 and T16 shows the resource usage up to T8.

  If the qstat is done at T17, the output shows usage up through T16. If the job ends at T20, the accounting log (and the final log message, and the email to you if "qsub -me" was used in job submission) contains usage through T20.
- The final report does not include the epilogue. The time required for the epilogue is treated as system overhead.
- The order in which jobs are displayed is undefined.

10.2 Checking Server Status

To see server information in default format:

qstat -B [<server> ...]

To see server information in long format:

qstat -B -f [-F json|dsv [-D <delimiter>]] [-w] [<server> ...]

10.2.1 Viewing Server Information in Default Format

The "-B" option to qstat displays the status of the specified PBS server. One line of output is generated for each server queried. The three letter abbreviations correspond to various job limits and counts as follows: Maximum, Total, Queued, Running, Held, Waiting, Transiting, and Exiting. The last column gives the status of the server itself: active, idle, or scheduling.

```
qstat -B
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
fast.domain   0   14   13    1    0    0    0    0 Active
```

10.2.2 Viewing Server Information in Long Format

You can see server status in JSON or delimiter-separated value formats; see section 10.4, “Full Display Options for Job, Queue, and Server Status”, on page 186.

When querying jobs, servers, or queues, you can add the "`-f`" option to `qstat` to change the display to the *full* or *long* display. For example, the server status shown above can be expanded using "`-f`" as shown below:

```
qstat -Bf
Server: fast.mydomain.com
    server_state = Active
    scheduling = True
    total_jobs = 14
    state_count = Transit:0 Queued:13 Held:0 Waiting:0
        Running:1 Exiting:0
    managers = user1@fast.mydomain.com
    default_queue = workq
    log_events = 511
    mail_from = adm
    query_other_jobs = True
    resources_available.mem = 64mb
    resources_available.ncpus = 2
    resources_default.ncpus = 1
    resources_assigned.ncpus = 1
    resources_assigned.nodect = 1
    scheduler_iteration = 600
    pbs_version = PBSPro_2021.1.41640
```

10.3 Checking Queue Status

To view queue information in default format:

qstat -Q [<destination> ...]

To view queue information in alternate format:

qstat -q [-G | -M] [<destination> ...]

To view queue information in long format:

qstat -Q -f [-F json|dsv [-D <delimiter>]] [-w] [<destination> ...]

If you specify a destination identifier, it takes one of the following three forms:

<queue>

@<server>

<queue>@<server>

If you specify *<queue>*, the request is for status of that queue at the default server.

If you use the *@<server>* form, the request is for status of all queues at that server.

If you specify a full destination identifier, *<queue>@<server>*, the request is for status of the named *queue* at the named *server*.

10.3.1 Viewing Queue Information in Default Format

The "-Q" option to qstat displays the status of specified queues at the (optionally specified) PBS server. One line of output is generated for each queue queried.

```
qstat -Q
Queue Max Tot Ena Str Que Run Hld Wat Trn Ext Type
----- --- --- --- --- --- --- --- --- --- --- ---------
workq   0  10 yes yes   7   1   1   1   0   0 Execution
```

The columns show the following for each queue:

- Queue Queue name
- Max Maximum number of jobs allowed to run concurrently in the queue
- Tot Total number of jobs in the queue
- Ena Whether the queue is enabled or disabled
- Str Whether the queue is started or stopped
- Que Number of queued jobs
- Run Number of running jobs
- Hld Number of held jobs
- Wat Number of waiting jobs
- Trn Number of jobs being moved (transiting)
- Ext Number of exiting jobs
- Type Type of queue: execution or routing

10.3.2 Viewing Queue Information in Long Format

You can see queue information in JSON or delimiter-separated value formats; see section 10.4, “Full Display Options for Job, Queue, and Server Status”, on page 186.

Use the long format to see the value for each queue attribute:

```
qstat -Qf
Queue: workq
    queue_type = Execution
    total_jobs = 10
    state_count = Transit:0 Queued:7 Held:1 Waiting:1
        Running:1 Exiting:0
    resources_assigned.ncpus = 1
    hasnodes = False
    enabled = True
    started = True
```

10.3.3 Displaying Queue Limits in Alternate Format

The "-q" option to qstat displays any limits set on the requested (or default) queues. Since PBS is shipped with no queue limits set, any visible limits will be site-specific. The limits are listed in the format shown below.

```
qstat -q
server: south

Queue  Memory CPU Time Walltime Node Run Que Lm  State
------ ------ -------- -------- ---- --- --- --  -----
workq    --      --       --     --    1   8 --   E R
```

10.4 Full Display Options for Job, Queue, and Server Status

-f [-w]

Full display. Job, queue or server attributes displayed one to a line.

Optional -w prints each attribute on one unbroken line. Feed characters are converted:

- Newline is converted to backslash concatenated with "n": "\n"
- Form feed is converted to backslash concatenated with "f": "\f"

This -w is independent of the -w job alternate output option.

-F dsv [-D <delimiter>]

Prints output in delimiter-separated value format. The default delimiter is a pipe ("|"). You can specify a character or a string delimiter using the -D argument to the -F dsv option. For example, to use a comma as the delimiter:

```
qstat -f -F dsv -D,
```

If the delimiter itself appears in a value, it is escaped:

- On UNIX/Linux, the delimiter is escaped with a backslash ("\").
- On Windows, the delimiter is escaped with a caret ("^").

Feed characters are converted:

- Newline is converted to backslash concatenated with "n": "\n"
- Form feed is converted to backslash concatenated with "f": "\f"

A newline separates each job from the next. Using newline as the delimiter leads to undefined behavior.

Example of getting output in delimiter-separated value format:

```
qstat -f -Fdsv


Job Id: 1.vbox|Job_Name = STDIN|Job_Owner = root@vbox|job_state = Q|queue = workq|server =
    vbox|Checkpoint = u|ctime = Fri Nov 11 17:57:05 2016|Error_Path = ...
```

-F json

Prints output in JSON format (http://www.json.org/).

Attribute output is preceded by timestamp, PBS version, and PBS server hostname.

Example:

```
qstat -f -F json


{
    "timestamp":1479277336,
    "pbs_version":"14.1",
    "pbs_server":"vbox",
    "Jobs":{
        "1.vbox":{
            "Job_Name":"STDIN",
            "Job_Owner":"root@vbox",
            "job_state":"Q",
...
```

-G

Show size in gigabytes. Alternate format is used.

-M

Show size in megawords. A word is considered to be 8 bytes. Alternate format is used.

10.4.1 Caveats for the `qstat` Command

When you use the `-f` option to `qstat` to display attributes of jobs, queues, or servers, attributes that are unset may not be displayed. If you do not see an attribute, it is unset.

10.5 Selecting a List of Jobs

The `qselect` command provides a method to list the job identifier of those jobs, job arrays or subjobs which meet a list of selection criteria. Jobs are selected from those owned by a single server. The `qselect` command writes to standard output a list of zero or more job identifiers which meet the criteria specified by the options. Each option acts as a filter restricting the number of jobs which might be listed. With no options, the `qselect` command will list all jobs at the server which you are authorized to list (query status of). The `-u` option may be used to limit the selection to jobs owned by you or other specified users.

For a description of the `qselect` command, see "qselect" on page 186 of the PBS Professional Reference Guide.

For example, say you want to list all jobs owned by user "barry" that requested more than 16 CPUs. You could use the following `qselect` command syntax:

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

Notice that what is returned is the job identifiers of jobs that match the selection criteria. This may or may not be enough information for your purposes. Many users will use shell syntax to pass the list of job identifiers directly into `qstat` for viewing purposes, as shown in the next example (necessarily different between Linux and Windows).

Linux:

```
qstat -a  `qselect -u barry -l ncpus.gt.16 `
                                                  Req'd Req'd  Elap
Job ID    User  Queue Jobname Sess NDS TSK Mem   Time S Time
--------  ----- ----- ------- ---- --- --- ---   ---- - ----
121.south barry workq airfoil  --   --  32  --   0:01 H   --
133.south barry workq trialx   --   --  20  --   0:01 W   --
154.south barry workq airfoil 930   --  32  --   1:30 R 0:32
```

Windows (type the following at the cmd prompt, all on one line):

```
for /F "usebackq" %j in (`qselect -u barry -l ncpus.gt.16`)  do
( qstat -a %j )
121.south
133.south
154.south
```

Note: This technique of using the output of the qselect command as input to qstat can also be used to supply input to other PBS commands as well.

10.5.1 Listing Job Identifiers of Finished and Moved Jobs

You can list identifiers of finished and moved jobs in the same way as for queued and running jobs, as long as the job history is still being preserved.

The -x option to the qselect command allows you to list job identifiers for all jobs, whether they are running, queued, finished or moved. The -H option to the qselect command allows you to list job identifiers for finished or moved jobs only.

10.5.2 Listing Jobs by Time Attributes

You can use the qselect command to list queued, running, finished and moved jobs, job arrays, and subjobs according to their time attributes. The -t option to the qselect command allows you to specify how you want to select based on time attributes. You can also use the -t option twice to bracket a time period.

Example 10-5: Select jobs with end time between noon and 3PM.

```
qselect -te.gt.09251200 -te.lt.09251500
```

Example 10-6: Select finished and moved jobs with start time between noon and 3PM.

```
qselect -x -s "MF" -ts.gt.09251200 -ts.lt.09251500
```

Example 10-7: Select all jobs with creation time between noon and 3PM

```
qselect -x -tc.gt.09251200 -tc.lt.09251500
```

Example 10-8: Select all jobs including finished and moved jobs with qtime of 2.30PM (default relation is .eq.)

```
qselect -x -tq09251430
```

10.6 Checking License Availability

You can check to see where licenses are available. You can do either of the following:

- Display license information for the current host:

  ```
  qstat -Bf
  ```

- Display resources available (including licenses) on all hosts:

  ```
  qmgr
  Qmgr: print node @default
  ```

If your site is using floating licenses, when looking at the server's license_count attribute, use the sum of the *Avail_Global* and *Avail_Local* values.

11 Submitting Cray Jobs

11.1 PBS Jobs on Cray Shasta

Submitting a PBS job on a Shasta system is exactly like submitting a job on a standard Linux machine. The rest of this chapter does NOT apply to Shasta systems.

11.2 PBS Jobs on the Cray XC

You can submit jobs that are designed to run on the Cray XC, using the PBS select and place syntax.

When you submit a job that is designed to run on the Cray XC, you create a job script that contains the same `aprun` command as a non-PBS job, but submit the job using the PBS select and place syntax.

If a job does not request a login node, one is chosen for it. A login node is assigned to each PBS job that runs on the Cray XC. The job script runs on this login node.

Jobs requesting a vntype of *cray_compute* are expected to have an `aprun` in the job script to launch the job on the compute nodes. PBS does not verify that the job script contains an `aprun` statement.

You may find it helpful to run `qsub` in the foreground by using the `-f` option. This can avoid stale ALPS reservations not being released.

11.3 Resources for Cray XC

accelerator
accelerator_memory
accelerator_model
energy
eoe
naccelerators
PBScrayhost
PBScraylabel_<label name>
PBScraynid
PBScrayorder
vntype

See "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide.

11.3.1 Resource Accounting for Cray XC

Jobs that request only compute nodes are not assigned resources from login nodes. PBS accounting logs do not show any login node resources being used by these jobs.

Jobs that request login nodes are assigned resources from login nodes, and those resources appear in the PBS accounting logs for these jobs.

PBS performs resource accounting on the login nodes, under the control of their MoMs.

Comprehensive System Accounting (CSA) runs on the compute nodes, under the control of the Cray XC system. CSA requires CLE 5.2.

11.4 Rules for Submitting Jobs on the Cray XC

11.4.1 Always Specify Node Type on Cray XC

If you want your job to run on Cray XC nodes, you must specify a Cray XC node type for your job. You do this by requesting a value for the vntype vnode resource. On each vnode on a Cray XC, the vntype resource includes one of the following values:

cray_login, for a login node

cray_compute, for a compute node

Each chunk of a Cray XC job that must run on a login node must request a vntype of *cray_login*.

Each chunk of a Cray XC job that must run on a compute node must request a vntype of *cray_compute*.

Example 11-1: Request any login node, and two compute-node vnodes. The job is run on the login node selected by the scheduler:

```
qsub -lselect=1:ncpus=2:vntype=cray_login +2:ncpus=2:vntype=cray_compute
```

Example 11-2: Launch a job on a particular login node by specifying the login node vnode name first in the select line. The job script runs on the specified login node:

```
qsub -lselect=1:ncpus=2:vnode=login1 +2:ncpus=2:vntype=cray_compute
```

For a description of the vntype resource, see "Resources Built Into PBS" on page 265 of the PBS Professional Reference Guide.

11.4.2 Always Reserve Required Vnodes on Cray XC

Always reserve at least as many PEs as you request in your `aprun` statement.

11.4.3 Requesting Cray XC Login Node Where Job Script Runs

If you request a login node as part of your resource request, the login node resource request must be the first element of the select statement. The job script is run on the login node. If you request more than one login node, the job script runs on the first login node requested.

11.4.4 Cray XC Login Nodes in PBS Reservations

If the jobs that are to run in a PBS reservation require a particular login node, you must do the following:

- The reservation must request the specific login node
- Each job that will run in the reservation must request the same login node that the reservation requested

11.4.5 Specifying Number of Chunks on Cray XC

You specify the number of chunks by prefixing each chunk request with an integer. If not specified, this integer defaults to *1*. For example, to specify 4 chunks with 2 CPUs each, and 8 chunks with 1 CPU each:

```
qsub -lselect=4:ncpus=2+8:ncpus=1
```

You cannot request the nchunk resource.

If you request fewer chunks, the scheduling cycle is faster. See section 11.9.5, "Request Fewer Chunks on Cray XC", on page 198.

11.4.6 Specify Host for Interactive Jobs on Cray XC

Interactive jobs on a Cray XC must run on a login node. When you run an interactive job, specify the login node as the host for the job. You can do so using a PBS directive, or the command line. For example:

```
qsub -l select=host=<name of login node> -I job.sh
#PBS -l select=host=<name of login node>
```

See section 6.11, "Running Your Job Interactively", on page 121.

11.5 Techniques for Submitting Cray XC Jobs

11.5.1 Requesting Groups of Cray XC Login Nodes

If you want to use groups of esLogin nodes and internal login nodes, your administrator can set the vntype resource on these nodes to a special value, for example *cray_compile*.

To submit a job requesting any combination of esLogin nodes and internal login nodes, you specify the special value for the vntype resource in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_compile job
```

11.5.2 Using Internal Cray XC Login Nodes Only

Compiling, preprocessing, and postprocessing jobs can run on internal login nodes. Internal login nodes have a vntype value of *cray_login*. If you want to run a job that needs to use the resources on internal login nodes only, you can specify `vntype=cray_login` in your select statement. For example:

```
qsub -lselect=4:ncpus=1:vntype=cray_login job
```

11.5.3 Using Cray XC Compute Nodes

If your job script contains an `aprun` launch, you must run your job on compute nodes. To run your job on compute nodes, specify a vntype of *cray_compute*. For example:

```
lselect=2:ncpus=2:vntype=cray_compute
```

11.5.4 Requesting Specific Groups of Cray XC Nodes

You can use select and place to request the groups of vnodes you want.

You may need to group nodes by some criteria, for example:

- Certain nodes are fast nodes
- Certain nodes share a required or useful characteristic
- Some combination of nodes gives the best performance for an application

Your administrator can set up either of the following:

- Custom Boolean resources on each vnode, which reflect how the nodes are labeled, and allow you to request the vnodes that represent the group of nodes you want. These resources are named *PBScraylabel_<label name>,* and set to *True* on the vnodes that represent the labeled nodes.

  Your administrator must label the groups of nodes. For example, if a node is both fast and best for App1, it can have two labels, *fast*, and *BestForApp1*.

  To request the fast nodes in this example, add the following to each chunk request:

  :PBScraylabel_fast=True
- Other custom resources on each vnode, which are set to reflect the vnode's characteristics. For example, if a vnode is fast, it can have a custom string resource called "*speed*", with a value of *fast* on that vnode. You must ask your administrator for the name and possible values for the resource.

11.5.5 Requesting Cray XC Nodes in Specific Order

Your application may perform better when the ranks are laid out on specific nodes in a specific order. If you want to request vnodes so that the nodes are in a specific order, you can specify the host for each chunk of the job. For example, if you need nodes ordered "nid0, nid2, nid4", you can request the following:

```
qsub -l select=2:ncpus=2:host=nid0 +2:ncpus=2:host=nid2 +2:ncpus=2:host=nid4
```

11.5.6 Requesting Specific Hardware on Cray XC

PBS allows you to specifically request (or avoid) specific hardware. Your administrator must create a Boolean resource on each vnode, and set it to *True* where the vnode has that specific hardware. We recommend that the Boolean is called "*PBScraylabel_<hardware name>*".

You request or avoid this resource using `PBScraylabel_<hardware name>=True` or `PBScraylabel_<hardware name>=False`. For example:

```
qsub -lselect=3:ncpus=2:PBScraylabel_<new_and_fast>=true myjob
```

11.5.7 Requesting Accelerators on Cray XC

Accelerators are associated with vnodes when those vnodes represent a host that has at least one accelerator in state *UP*. PBS allows you to request vnodes with associated accelerators. PBS sets the value of the naccelerators host-level resource to the number of accelerators on the host. Note that this value is set for all vnodes on that host, so if you have a host with one accelerator and four vnodes, each of the four vnodes has naccelerators set to 1.

To request accelerators for your job, use the integer naccelerators resource to request a specific number of accelerators, or the Boolean accelerator resource if you do not care how many accelerators you get.

To request a vnode on a host with a specific number of associated accelerators, include the following in the job's select statement:

```
naccelerators = <number of accelerators>
```

To request a vnode on a host with any number of associated accelerators, you can include the following in the job's select statement:

```
accelerator = True
```

11.5.7.1 Advice on Requesting Accelerators on Cray XC

- When requesting accelerators, put them in the same chunks as CPUs. Otherwise the accelerators could end up in a chunk taken from a different host from the CPUs, and in that case your CPUs could be on a host without an accelerator.
- Use `accelerator=True` in a chunk only when you don't care how many accelerators are in the chunk.

11.5.7.2 Examples of Requesting Accelerators on Cray XC

Example 11-3: You want a total of 40 PEs with 4 PEs per compute node and one accelerator per compute node:

```
-lselect=10:ncpus=4:naccelerators=1
```

Example 11-4: You want 30PEs and a Tesla_x2090 accelerator on each host, and the accelerator should have at least 4000MB, and you don't care how many hosts the job uses:

```
-lselect=30:ncpus=1:nacclerators=1:accelerator_model="Tesla_x2090" :accelerator_memory=4000MB
    myjob
```

Example 11-5: You want a total of 40 PEs with 4 PEs per compute node and at least one accelerator per compute node:

```
-lselect=10:ncpus=4:accelerator=True
```

Example 11-6: Your system has some compute nodes with one type of accelerator (GPU1), and another type of compute node with a different type of accelerator (GPU2), and you want to request 10 PEs and 1 accelerator of model "GPU1" per compute node and 4 PEs and 1 accelerator of model "GPU2" per compute node. Your job request would look like this:

```
-lselect=10:ncpus=1:naccelerators=1:accelerator_model="GPU1"
    +4:ncpus=1:naccelerators=1:accelerator_model="GPU2" myjob
```

11.6 Using Xeon Phi Vnodes on Cray XC

PBS automatically detects a Xeon Phi in the ALPS inventory. The Xeon Phi attributes are configured automatically in the PBS vnode attributes. A job can make an explicit request for a Xeon Phi on a compute node and be dispatched to that node.

When using Xeon Phi vnodes, it's generally useful to request an AOE that gives you a Xeon Phi memory model with your desired values for numa_cfg and hbm_cache_pct. Run the `pbsnodes` command to find the available AOEs (look at resources_available.aoe) and choose the AOE you want.

To request Xeon Phi vnodes for the compute nodes used by your job, specify the cray_compute resource. For example, to request 10 Xeon Phi vnodes:

```
qsub -l select=10:vntype=cray_compute:aoe=<desired AOE>
```

11.7 Using Hyperthreads on Cray XC

PBS supports hyperthreading on Cray X* series systems using ALPS. On each vnode, PBS sets resources_available.ncpus to the number of compute unit elements returned in the XML inventory. This allows PBS to make ALPS reservations for the compute units of a node, and get all of the hyperthreads associated with those compute units.

You can use `aprun -j` to control how you use hyperthreads on XC series machines.

Your administrator may label hyperthreaded vnodes with a custom resource.

We recommend that when you submit jobs to PBS, you use the size of a compute node as the size of a chunk.

To use hyperthreading in your job, request hyperthreading in your `aprun` call in your job script. Please refer to Cray's man page for `aprun`.

Check the *Cray Programming Environment User Guide* for Cray's advice on using CPUs and compute units.

When hyperthreading is enabled on a system, each core is paired with one associated hyperthread, and you should request ncpus in multiples of *2* so that each associated hyperthread is assigned to the job's cgroup.

When hyperthreading is not enabled, you should request just the cores.

Let's say you request the following:

```
select=1:ncpus=2
```

If hyperthreading is enabled, this nets one core and one hyperthread.

If hyperthreading is not enabled, this nets two cores.

11.8 Viewing Cray XC Job Information

11.8.1 Finding Out Where Job Was Launched on Cray XC

To determine the internal login node where the job was launched, use the `qstat -f` command:

```
qstat -f <job ID>
```

Look at the exec_host line of the output. The first vnode is the login node where the job was launched.

11.8.2 Listing Jobs Running on Vnode on Cray XC

To see which jobs are running on a vnode, use the `pbsnodes` command:

```
pbsnodes -av
```

The jobs attribute of each vnode lists the jobs running on that vnode.

Jobs launched from an internal login node, requesting a vntype of *cray_compute* only, are **not** listed in the internal login node's vnode's jobs attribute.

Jobs that are actually running on a login node, which requested a vntype of *cray_login*, do appear in the login node's vnode's jobs attribute.

11.8.3 How ALPS Request Is Constructed on Cray XC

The reservation request that is sent to the Cray XC is constructed from the contents of the exec_vnode and Resource_List.select job attributes. If the exec_vnode attribute contains chunks asking for the same ncpus and mem, these are grouped into one section of an ALPS request. Cray XC requires one CPU per thread. The ALPS request is constructed using the following rules:

Table 11-1: How Cray XC Elements Are Derived From exec_vnode Terms

Cray XC Element	exec_vnode Term
Processing Element (PE)	mpiprocs
Requested number of PEs / compute node in this section of job request (width)	Total mpiprocs on vnodes representing compute node involved in this section of job request
Number of threads per PE (depth)	(total assigned ncpus on vnodes representing a compute node) / (total mpiprocs on vnodes representing a compute node)
Memory per PE (mem)	(total memory in chunk request)/total mpiprocs in chunk
Number of PEs per compute node (nppn)	Sum of mpiprocs on vnodes representing a compute node
Number of PEs per segment (npps)	Not used.
Number of segments per node (nspn)	Not used.
NUMA node (segments)	Not used.

11.8.4 Viewing Accelerator Information on Cray XC

There is no `aprun` interface for requesting accelerator memory or model, so this information is not translated into Cray XC elements. To see this information, look in the MoM logs for the job's login node.

11.9 Caveats and Advice for Cray XC

11.9.1 Using Combination or Number Resources on Cray XC

When requesting a resource that is set up by the administrator, you **must** use the same resource string values as the ones set up by the administrator. "*012*" is **not** the same as "*102*" or "*201*".

11.9.2 Avoid Invalid Cray XC Requests

It is possible to create a select and place statement that meets the requirements of PBS but not of the Cray XC.

Example 11-7: The Cray XC width and depth values cannot be calculated from ncpus and mpiprocs values. For example, if ncpus is 2 and mpiprocs is 4, the depth value is calculated by dividing ncpus by mpiprocs, and is one-half. This is not a valid depth value for Cray XC.

Example 11-8: ALPS cannot run jobs with some complex select statements. In particular, a multiple program, multiple data (MPMD) ALPS reservation where two groups span a compute node will produce an ALPS error, because the nid shows up in two ReserveParam sections.

11.9.3 Resource Restrictions and Deprecations on Cray XC

11.9.3.1 mpp* Resources Removed

The mpp* resources are no longer used.

11.9.4 Do Not Request PBScrayorder on Cray XC

Do not use PBScrayorder in a resource request.

11.9.5 Request Fewer Chunks on Cray XC

The more chunks in each translated job request, the longer the scheduling cycle takes. Jobs that request a value for ncpus effectively direct PBS to use the size of ncpus as the value for ncpus for each chunk, thus dividing the number of chunks by ncpus.

If you are on a homogeneous system, we recommend that chunks use the value for ncpus for a vnode or for a compute node.

Example 11-9: Comparison of larger vs. smaller chunk size and the effect on scheduling time:

Submit job with chunk size 1 and 8544 chunks:

```
qsub -lselect=8544:ncpus=1 job
```

Job's Resource_List:

```
Resource_List.ncpus = 8544
Resource_List.place = free
Resource_List.select = 8544:vntype=cray_compute
Submit_arguments = -lselect=8544:ncpus=1 job
```

Scheduling took 6 seconds:

```
12/05/2011 16:46:10;0080;pbs_sched;Job;23.example;considering job to run
12/05/2011 16:46:16;0040;pbs_sched;Job;23.example;Job run
```

If you are on a heterogeneous system, with varying sizes for vnodes or compute nodes, you can request chunk sizes that fit available hardware, but this may not be feasible.

11.9.6 Improving Performance on Cray XC

- You can speed performance via the -E option to qstat; see section 10.1.16, “Grouping Jobs and Sorting by ID”, on page 183.
- We give several recommendations for improving file transfer performance in section 15.6.6, "Advice on Improving File Transfer Performance", on page 564 of the PBS Professional Administrator’s Guide.
- You can avoid using the -f option to qsub where possible; see section 2.5.9, “Running qsub in the Foreground”, on page 30 and “Options to qsub” on page 218 of the PBS Professional Reference Guide.

11.10 Errors and Logging on Cray XC

11.10.1 Invalid Cray XC Requests

When a select statement does not meet Cray XC requirements, and the Cray XC reservation fails, the following error message is printed in MoM's log, at log event class 0x080:

```
"Fatal MPP reservation error preparing request"
```

11.10.2 Job Requests More Than Available on Cray XC

If do_not_span_psets is set to *True*, and a job requests more resources than are available in one placement set, the following happens:

- The job's comment is set to the following:
  ```
  "Not Running: can't fit in the largest placement set, and can't span psets"
  ```
- The following message is printed to the scheduler's log:
  ```
  "Can't fit in the largest placement set, and can't span placement sets"
  ```

12 Submitting Jobs to NEC SX-Aurora TSUBASA

12.1 Vnodes for NEC SX-Aurora TSUBASA

The basic hardware unit for NEC SX-Aurora TSUBASA is a *vector host* (a standard x86 server) connected to a set of accelerators called *vector engines* (VEs) via optional PCIe. The unit can consist of one or more NUMA nodes. Each unit uses one or more host channel adapters to communicate with other units and with the rest of the world.

The increasing order of communication overhead is first within a vector engine, then between vector engines via a shared PCIe, then between vector engines via PCIes on a common vector host, and finally between vector engines on separate vector hosts.

PBS creates topology-aware vnodes by grouping each PCIe with its associated vector host and vector engines together into one vnode. If there is no PCIe, PBS groups the vector host and its VEs into a vnode. A NUMA node without its own PCIe is in its own vnode. The topology-based vnode creation is handled by an exechost_startup event in the built-in hook named PBS_sx_aurora.

PBS tries to do topology-aware scheduling by grouping job processes on vector engines in a way that produces the lowest communication overhead. When a job requests vector engines, PBS tries to assign vector engines from a single vnode to minimize communication overhead between vector engines.

12.2 Terminology

HCA

Host channel adapter. Network interconnect used by vnode. Each vector host can have one or more HCAs.

Vector engine, VE

Accelerator associated with vector host. Executes parallel and/or vectorized numeric operations.

Vector host, VH

Standard x86 server. Performs tasks such as I/O.

VE offloading

Main operations that take place on the vector host offload parallel and/or vectorized numeric operations to vector engines. In offloading, NEC MPI launches processes on the VH, and those processes then launch other job processes on VEs assigned to the job. See section 12.4.3.4, “Using VE Offloading”, on page 206.

12.3 Resources for SX-Aurora TSUBASA

nves

Host-level consumable integer. Allows you to specify the number of vector engines per chunk. PBS sets the available VEs on a vnode in resources_available.nves. The default for resources_available.nves is number of VEs attached to the PCIe. The out-of-the-box default value for a job request is zero; PBS assigns a value of zero unless the administrator has set the value otherwise.

nhcas

Chunk-level non-consumable integer. When requested in a job chunk, PBS sets _NEC_HCA_LIST_IO and _NEC_HCA_LIST_MPI environment variables accordingly for that chunk. When not requested for a chunk, PBS sets _NEC_HCA_LIST_IO and _NEC_HCA_LIST_MPI to include all HCAs on a host.

ve_mem

Job-wide string. Used for reporting the maximum memory on vector engines used by job.

ve_cput

Job-wide string. Used for reporting the total CPU time, in seconds, on vector engines used by job.

ncpus

PBS sets the value of resources_available.ncpus on each vnode to (#CPUs on whole host / #vnodes on host) - #VEs on vnode. One CPU per VE is reserved for the VEOS daemon.

mem

PBS sets the value of resources_available.mem on each vnode by dividing the memory of the whole host equally among the vnodes on the host.

12.4 Running Your Job on NEC SX-Aurora TSUBASA

12.4.1 Requesting Resources on NEC SX-Aurora TSUBASA

You can request CPUs on the VH using the ncpus resource. If you do not request ncpus for the processes that will run on the VH, PBS will assign your job the default number of CPUs, which is generally 1 (one).

You request VEs for each chunk using the nves resource.

You request HCAs for each chunk using the nhcas resource. PBS will always try to assign the HCA closest to the job's VEs to the job.

12.4.1.1 Restrictions for Requesting HCAs

If you request chunks with different numbers of HCAs, for example a one-HCA chunk and a two-HCA chunk, add `-l place=scatter` to your request, otherwise performance may be reduced. If you do not specify `-l place=scatter`, the value of nhcas for all chunks on each vector host is set to the value given for the first chunk.

12.4.2 Default Process Distribution

On NEC SX-Aurora TSUBASA, use the mpirun command and optionally specify process distribution via the NEC_PROCESS_DIST environment variable. NEC MPI launches processes directly on VHs and/or VEs and orders process ranks according to the mpirun command line. Here are some example mpirun commands that use default process distribution:

Example 12-1: To run 32 ve.out processes on VEs:

```
mpirun -np 32 ve.out
```

Example 12-2: To run 1 vh.out process on the VH and 12 ve.out processes on VEs:

```
mpirun -vh -np 1 vh.out : -np 12 ve.out
```

The process with rank 0 executes vh.out on a VH, and processes with ranks 1 through 12 execute ve.out on VEs.

Example 12-3: We have 1 chunk with 4 VEs running 4 processes, and 1 chunk with 4 VEs running 13 processes:

```
qsub -lselect=1:nves=4:mpiprocs=4+1:nves=4:mpiprocs=13
```

In job script:

```
mpirun -np 17 ve.out
```

12.4.2.1 Letting PBS Distribute VE Processes in a Chunk

Without NEC_PROCESS_DIST, PBS distributes processes in a chunk as described in this section, depending on whether or not the number of processes in the chunk is an integer multiple of the number of VEs in the chunk. Processes are directly launched by NEC MPI. When it is an integer multiple, we call it "perfect distribution".

12.4.2.1.i Perfect Distribution

Within a chunk, if the number of processes is an integer multiple of the number of VEs, PBS puts the same number of processes on each VE. For example if you have 12 processes and 4 VEs, PBS puts 3 processes on each VE.

To let PBS distribute VE processes as evenly as possible across the VEs in a chunk, request the chunk, and specify the number of VEs and the number of processes via the mpiprocs resource.

Example 12-4: We have 1 chunk with 6 processes and 2 VEs, and we want 3 VE processes to run on each VE:

In the job script:

```
mpirun -np 6 ve.out
```

Submit the job:

```
qsub -l select=ncpus=2:nves=2:mpiprocs=6 ...
```

12.4.2.1.ii Imperfect Distribution

If the number of processes is not an integer multiple of the number of VEs, PBS gives more processes to the VEs earlier in topological order as recognized by PBS. PBS gives those VEs the smallest integer greater than (#processes / #VEs), and puts the remainder on the next VE. For example if you have 10 processes and 4 VEs, PBS may put 3 processes on the first 3 VEs, and one process on the last VE. However, if you have 5 processes and 4 VEs, PBS puts 2 processes on the first 2 VEs, 1 process on the next VE, and no processes on the last VE.

Example 12-5: We request 2 CPUs, 3 VEs, and 8 mpiprocs. In this example, the first VE gets 3 VE processes, the second VE gets 3 VE processes, and the third VE gets 2 processes. In the job script:

```
mpirun -np 8 ve.out
```

Submit the job:

```
qsub -l select=ncpus=2:mpiprocs=8:nves=3 ...
```

Distribution of processes on VEs:

```
ve=0
ve=0
ve=0
ve=1
ve=1
ve=1
ve=2
ve=2
```

12.4.3 Specifying Process Distribution

You can optionally use the NEC_PROCESS_DIST environment variable to specify where processes should run. You specify process distribution chunk by chunk. You can either launch processes directly via NEC MPI, or you can have the process(es) that run on the VH launch any processes that run on VEs.

In each chunk, you can do the following:

- Specify the number of processes for each VE in the chunk, using one of these methods:
  - Specify different numbers of processes for direct launch on each VE, as in Section 12.4.3.1, "Specifying Process Placement for All VEs in a Chunk"
  - Distribute processes for direct launch evenly by specifying the number of processes for just the first VE and implying the rest, as in Section 12.4.3.2, "Replicating Process Distribution Across VEs in a Chunk"
  - Let PBS distribute the processes for direct launch as evenly as possible, as in Section 12.4.2.1, "Letting PBS Distribute VE Processes in a Chunk"
  - Use VE offloading to indirectly launch processes on VEs, as in Section 12.4.3.4, "Using VE Offloading"
- Specify which processes should run on the vector host, as in Section 12.4.3.3, "Placing Processes on VHs"

Additionally, if you have a group of identical chunks, you can specify the process placement for the first of these chunks, and PBS can replicate the placement for the remaining identical chunks. If you have more than one such group, where each group is different, you can specify the placement for just the first chunk for each group. See Section 12.4.3.5, "Replicating the Same Process Distribution Across Multiple Chunks".

We describe the syntax for process distribution in detail in each of the following sections, but here is a summary. Syntax for process distribution for multiple chunks:

NEC_PROCESS_DIST = <chunk1 distribution>+<chunk2 distribution> +... + <chunkN distribution>

where *<chunk distribution>* is

[<number of processes on vector host>]:[<number of processes on first VE>][:<numbers of processes on subsequent VEs>]

Separate each chunk specification using a plus sign ("+").

12.4.3.1 Specifying Process Placement for All VEs in a Chunk

Within each chunk, you can optionally specify how many VE processes should be directly launched by NEC MPI on each VE. Separate the number of VE processes you want on each VE with a colon.

Syntax for distribution of processes on VEs in a single chunk:

NEC_PROCESS_DIST=<process count for first VE>:<process count for second VE> :...: <process count for nth VE>

Example 12-6: We have 6 processes, and 3 VEs, and we want 1 process to run on the first VE, 3 on the second, and 2 on the third:

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

12.4.3.1.i Restrictions and Caveats for Process Placement for All VEs in Chunk

For a chunk, when you specify the process count for more than one VE, you must specify the process count for all VEs.

12.4.3.2 Replicating Process Distribution Across VEs in a Chunk

If you specify how many VE processes should be directly launched by NEC MPI on just the first VE in a chunk, PBS can replicate that distribution for the rest of the VEs in the chunk. Distribution depends on whether or not the number of processes in the chunk is an integer multiple of the number of VEs in the chunk. When it is an integer multiple, we call it "perfect distribution". We say that when you specify only for the first VE in a chunk, you are *implying* what should happen for the others in that chunk.

Syntax for replicating process distribution across VEs in a single chunk:

NEC_PROCESS_DIST=<process count for first VE>

12.4.3.2.i Implying Perfect Distribution

You can use implied specification for perfect distribution via direct launch by NEC MPI over VEs.

Example 12-7: We have 1 chunk with 6 processes and 2 VEs, and we want 3 VE processes to run on each VE:

```
-l select=ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=3
```

12.4.3.2.ii Implying Imperfect Distribution

You can imply imperfect distribution (when the number of processes is not an integer multiple of the number of VEs) via direct launch by NEC MPI, by following the same formula PBS uses. Specify the larger number of processes (the smallest integer greater than #processes/#VEs) for the first VE. For example, if you have 7 processes and 3 VEs, specify 3 processes for the first VE, not 1 or 2.

Example 12-8: We have 1 chunk with 11 processes and 3 VEs, and we want 4 VE processes to run on the first and second VEs, and 3 processes on the third VE:

```
-l select=ncpus=2:nves=3:mpiprocs=11 -v NEC_PROCESS_DIST=4
```

12.4.3.3 Placing Processes on VHs

For each chunk in a job, you can place one or more of the processes on the VH by specifying the number of VH processes. Use the letter "S" before the number of VH processes. You can combine this with the methods for launching processes on VEs (direct launch or VE offloading).

Syntax for distribution of processes on VH:

*NEC_PROCESS_DIST=**S**<VH process count>*

Syntax for distribution of processes for a single chunk on VH and on VEs (the specification for VH process count can appear in any position in the chunk distribution; we show it here in the first position):

*NEC_PROCESS_DIST=**S**<VH process count>:<process count for first VE>[:<process count for second VE> :...: <process count for nth VE>]*

Separate chunks with a plus sign:

*NEC_PROCESS_DIST=**S**<VH process count>:<process count for first VE>[:<process count for second VE> :...: <process count for nth VE>]+**S**<VH process count>:<process count for first VE>[:<process count for second VE> :...: <process count for nth VE>]*

Example 12-9: We have 5 processes and 1 VE. The first 2 processes will run on the VH, and the last 3 will run on the VE:

```
qsub -lselect=1:ncpus=2:nves=1:mpiprocs=5 -v NEC_PROCESS_DIST=S2:3
```

12.4.3.3.i Restrictions and Caveats for Placing Processes on VHs

- When you specify process placement on a VH, you must request the ncpus resource to reserve CPUs for those processes.
- You can specify VH process distribution only once per chunk.
- If you specify VH processes, you must specify at least one VH process; you cannot specify zero VH processes.

12.4.3.4 Using VE Offloading

In offloading, NEC MPI launches main processes on the vector host, and those processes then launch (offload) parallel and/or vectorized numeric operations on vector engines that are attached to the vector host and assigned to the job. A process that has been offloaded to a VE is not directly started by NEC MPI.

A VE process that has not been offloaded is directly started by NEC MPI on the VE.

VE offloading is applied chunk by chunk; you specify which chunks should use it. In any chunk, you can either use VE offloading or not; you cannot mix offloading and direct launch on VEs by NEC MPI in the same chunk.

To offload processes to VEs:

- Use the nves resource to specify the number of VEs you need for offloading
- Use the NEC_PROCESS_DIST environment variable to specify that one or more processes should start on the VH
- Specify that no processes should be started directly by NEC MPI on VEs
- Make sure that the number of processes in the `mpirun` command and the total number of mpiprocs is the same, because NEC MPI launches only the processes on the VH; it does not launch processes that are offloaded.

We use a specific syntax to indicate offloading, regardless of the number of VEs. While it may seem that logically equivalent syntax should also work, it will not. Specify zero for the first VE only, and nothing else for other VEs. Syntax for VE offloading:

NEC_PROCESS_DIST=S<number of VH processes>:0

Example 12-10: We will run 2 processes on the VH, and offload 3 processes to the VEs. Process `vh.out` launches `ve.out` on attached VEs:

```
qsub -l select=ncpus=2:mpiprocs=2:nves=2 -v NEC_PROCESS_DIST=S2:0
```

In job script:

```
mpirun -vh -np 2 vh.out
```

12.4.3.4.i Restrictions and Caveats for VE Offloading

The syntax for VE offloading is ***S<VH process count>:0***, not S<VH process count>:0:0:0 or S<VH process count>, despite the fact that the last two look logically equivalent.

12.4.3.5 Replicating the Same Process Distribution Across Multiple Chunks

If you have a group of identical chunks, you can specify the process placement for the first of these chunks, and PBS can replicate the placement for the remaining identical chunks. (You can repeat the specification for each identical chunk if you want, but you don't need to.) Syntax for process distribution for one group of identical chunks:

NEC_PROCESS_DIST = <distribution for all chunks in group>

Example 12-11: We have 2 identical chunks, each with 6 processes and 3 VEs, and for each chunk we want 1 process to run on the first VE, 3 on the second, and 2 on the third. We specify the distribution for the first chunk, and PBS replicates it for the remaining identical chunk:

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

If you have more than one group of identical chunks, where each group is different, you can specify just the first chunk for each group. Syntax for process distribution for multiple groups of identical chunks:

NEC_PROCESS_DIST = <group1 chunk distribution>+<group2 chunk distribution> +... + <group N chunk distribution>

Example 12-12: We have 2 groups of chunks:

The first group is 2 chunks that have 6 processes and 3 VEs and we want 1 process to run on the first VE, 3 on the second, and 2 on the third.

For the second group of 2 chunks, we have 6 processes evenly distributed across 2 VEs.

We specify the distribution for the first chunk in each group, and PBS replicates it for the remaining identical chunk in each group:

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6+2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2+3
```

12.4.3.6 Examples of Specifying Process Distribution

In the following examples, we'll use `vh.out` as the name of a process that runs on VHs, and `ve.out` as the name of the process that runs on VEs.

Example 12-13: We have 6 processes, and 3 VEs, and we want 1 process to run on the first VE, 3 on the second, and 2 on the third:

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2
```

In job script:

```
mpirun -np 6 ve.out
```

Example 12-14: We have 1 chunk with 6 processes and 2 VEs, and we want 3 VE processes to run on each VE:

```
qsub -l select=ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=3
```

In job script:

```
mpirun -np 6 ve.out
```

Example 12-15: We have 2 chunks:

In the first chunk, we have 6 processes, and 3 VEs, and we want 1 process to run on the first VE, 3 on the second, and 2 on the third.

In the second chunk, we have 6 processes and 2 VEs, and we want 3 VE processes to run on each VE (combining examples 12-13 and 12-14):

```
qsub -lselect=1:ncpus=2:nves=3:mpiprocs=6+1:ncpus=2:nves=2:mpiprocs=6 -v NEC_PROCESS_DIST=1:3:2+3
```

In job script:

```
mpirun -np 12 ve.out
```

Example 12-16: We have 3 chunks:

In the first chunk, we have 5 processes and 1 VE. The first 2 processes will run on the VH, and the last 3 will run on the VE.

In the second chunk, we have 6 processes and 3 VEs; 1 process will run on the VH and 5 will run on the VEs.

In the third chunk, we'll run 4 processes on a VE:

```
qsub -l
    select=1:ncpus=2:nves=1:mpiprocs=5+1:ncpus=2:nves=3:mpiprocs=6+1:ncpus=2:nves=1:mpiprocs=4 -v
    NEC_PROCESS_DIST=S2:3+2:1:S1:2+4
```

In job script:

```
mpirun -vh -np 2 vh.out : -np 6 ve.out : -vh -np 1 vh.out : -np 6 ve.out
```

Example 12-17: We have 2 chunks:

In the first chunk, process `vh.out` launches 3 processes `ve.out` on attached VEs.

In the second chunk, we run 4 processes on 2 VEs:

```
qsub -l select=ncpus=2:mpiprocs=2:nves=2+1:ncpus=2:nves=2:mpiprocs=4 -v NEC_PROCESS_DIST=S2:0+2
```

In job script:

```
mpirun -vh -np 2 vh.out : -np 4 ve.out
```

Example 12-18: We have 3 groups of chunks:

The first group is 2 chunks that have 6 processes and 3 VEs and we want 1 process to run on the first VE, 3 on the second, and 2 on the third.

For the second group of 2 chunks, we have 6 processes evenly distributed across 2 VEs.

For the third group, we have 3 chunks, and we run 1 process on the VH and 2 processes on a VE.

We specify the distribution for the first chunk in each group, and PBS replicates it for the remaining identical chunk in each group:

```
qsub -lselect=2:ncpus=2:nves=3:mpiprocs=6+2:nves=2:mpiprocs=6+3:ncpus=2:nves=1:mpiprocs=3  -v
    NEC_PROCESS_DIST=1:3:2+3+S1:2
```

In the job script:

```
mpirun -np 24 ve.out : -vh -np 1 vh.out : -np 2 ve.out : -vh -np 1 vh.out : -np 2 ve.out : -vh -np
    1 vh.out : -np 2 ve.out
```

Example 12-19: We have 3 chunks:

In the first chunk, we have 5 processes and 1 VE. The first 2 processes will run on the VH, and the last 3 will run on the VE.

In the second chunk, we have 3 processes, all of which will run on the VH.

In the third chunk, we'll run 4 processes on a VE:

```
qsub -l select=1:ncpus=2:nves=1:mpiprocs=5+1:ncpus=3:mpiprocs=3+1:ncpus=2:nves=1:mpiprocs=4 -v
    NEC_PROCESS_DIST=S2:3+S3+4
```

In job script:

```
mpirun -vh -np 2 vh.out : -np 3 ve.out : -vh -np 3 vh.out : -np 4 ve.out
```

12.4.3.7 Restrictions and Caveats for Specifying Process Distribution

- If you use the NEC_PROCESS_DIST environment variable for a job, you must request the mpiprocs resource for all chunks in the job.
- For each job, make sure that the number of chunks in your select specification is the same as the number in NEC_PROCESS_DIST.
- For each chunk, the sum of the specified VE processes and VH processes must be equal to the sum of mpiprocs. You can imply even distribution, as in section 12.4.3.2.i, “Implying Perfect Distribution”, on page 205.
- If you specify distribution for any VEs in a job, you must specify distribution for all VEs in all chunks of the job. You can imply even distribution, as in section 12.4.3.2.i, “Implying Perfect Distribution”, on page 205.
- When you are not using VE offloading, you cannot specify a count of zero processes on any VEs.

12.5 Job Accounting on NEC SX-Aurora TSUBASA

When PBS writes accounting records, PBS records nves in both Resource_List.nves and resources_assigned.nves. PBS also writes ve_mem and ve_cput as part of the value of the job's resources_used attribute.

12.6 Environment Variables for NEC MPI

Before launching a job requesting vector engines, PBS sets the following environment variables on each execution host:

_VENODELIST

List of VE numbers assigned to this job on a VH.

For example, PBS assigns VE numbers 0-3 to a job on a VH via `export _VENODELIST="0 1 2 3";` if no VEs are assigned to a job on a VH, you get `export _VENODELIST=""`

VE_NODE_NUMBER

The lowest VE number assigned to this job on a VH.

For example, PBS assigns VE numbers 0-3 on a VH via `export VE_NODE_NUMBER="0";` if no VEs are assigned to a job on a VH, you get `export VE_NODE_NUMBER=-1`

_NECMPI_VE_NUM_NODES

Number of vector engines assigned to this job which are directly used by NEC MPI and not used for VE off-loading.

For example, PBS assigns VE numbers 0-3 to a job on a VH via `export _NECMPI_VE_NUM_NODES=4`

_NECMPI_VE_NODELIST

Space-separated list of VE numbers assigned to this job which are directly used by NEC MPI and not used for VE offloading.

For example, PBS assigns VE numbers 0-3 to a job on a VH via `export _NECMPI_VE_NODELIST="0 1 2 3"`

_NEC_HCA_LIST_IO

Space-separated list of HCA-lists assigned to this job on this vector host.

An HCA-list is a comma-separated list of HCAs assigned to this job on each VE.

When a job requests nhcas for a job chunk, PBS sets _NEC_HCA_LIST_IO accordingly for that chunk. When nhcas is not requested for a chunk, PBS sets _NEC_HCA_LIST_IO to include all HCAs on a host.

For example, if there are two VEs and two HCAs (labeled mlx5_0:1 and mlx5_1:1) attached to the VH, and each VE uses both HCAs, _NEC_HCA_LIST_IO=mlx5_0:1,mlx5_1:1 mlx5_0:1,mlx5_1:1

_NEC_HCA_LIST_MPI

Space-separated list of HCA-lists for each VE used by this job, and not used for offloading. See the environment variable _NEC_HCA_LIST_IO

When a job requests nhcas for a job chunk, PBS sets _NEC_HCA_LIST_MPI accordingly for that chunk. When nhcas is not requested for a chunk, PBS sets _NEC_HCA_LIST_MPI to include all HCAs on a host.

For example, if there are two VEs and two HCAs (labeled mlx5_0:1 and mlx5_1:1) attached to the VH, and each VE uses both HCAs, but one of the VEs is used for offloading,
_NEC_HCA_LIST_IO=mlx5_0:1,mlx5_1:1

13 Using MLS with PBS Professional

13.1 About SELinux PBS Professional

With this version of PBS Professional, we offer a separate software package that supports SELinux enforcement mode used with one or more MLS policies on RHEL 7. In this chapter, we describe how to use SELinux PBS only.

13.2 Requirement for Submitting Jobs

When you submit a job to PBS, you must do so from a machine running this version of the PBS commands. This version of PBS includes the security_context job attribute, and if it is missing, an error occurs, a server message is logged, and the job is rejected.

13.3 Viewing and Operating on Jobs

When a user queries or operates on his or her own job, PBS requires that the security_context attribute of the job match the security context of the requester. It is not sufficient that the credential of the requester dominate that of the job.

13.3.1 Checking Security Context

PBS writes the security context for a job in its security_context attribute. For example, if you need to compare the security context of jobs and users:

Find job security context:

```
bash-4.2$ qstat -f | grep security
security_context = user_u:user_r:user_t:s3:c1,c2
```

Find user security context:

```
bash-4.2$ id -Z
user_u:user_r:user_t:s3:c1,c2
```

13.4 Credentials of Deleted Jobs

When a job is deleted via `qdel` or `qdel -x`, the job's credentials are handled the same way they would be if PBS were not being used.

13.5 Caveats

If your site is using polyinstantiation and MoM (the execution service process) dies or is told to exit while there are running jobs, you may need to manually clean up directories and/or files created for those jobs after the jobs have exited.

13.6 Errors and Logging

13.6.1 Logging

PBS may log the following messages in the server and/or MoM logs:

Table 13-1: Log Messages

Message	Log Level
no security context found	Error (0x0001)
failed to read credential file	Error (0x0001)
job credential <context>	Debug (0x0080)
unable to create the job directory	Error (0x0001)
the staging and execution directory <dir> already exists	Debug3 (0x0400)
could not set security context for <dir>	Error (0x0001)
unable to change permissions on staging and execution directory <dir>	Debug (0x0080)
unable to open user session	Debug4 (0x0800)
unable to start job, no security context found or not able to set security context	Error (0x0001)
cannot get current socket context	Error (0x0001)
cannot set socket context for <jobid>	Error (0x0001)
cannot restore socket context	Error (0x0001)
failed to set file context for <dir>	Error (0x0001)
could not save security context	Debug (0x0080)
saved security context	Debug (0x0080)
client connection no security context information present	Security (0x0020)
client connection security context information present, but no job security context information present	Security (0x0020)
client connection security context information not present, but job security context information present	Security (0x0020)
client connection security context information job security context information do not match	Security (0x0020)

13.6.2 Errors

PBS may write the following error messages to the job submitter's standard error:

```
"pbs_iff:  fgetfilecon(1) returned -1"
"pbs_iff:  filecon overflow"
"pbs_iff:  malloc extendarg failed"
```

13.7 SELinux Documentation

- DIRECTOR OF CENTRAL INTELLIGENCE DIRECTIVE 6/3 PROTECTING SENSITIVE COMPARTMENTED INFORMATION WITHIN INFORMATION SYSTEMS
  http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm
- Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide
  https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/index
- Getting Started with Reference Policy
  http://oss.tresys.com/projects/refpolicy/wiki/GettingStarted

14 Using Provisioning

PBS provides automatic provisioning of an OS or application on vnodes that are configured to be provisioned. When a job requires an OS that is available but not running, or an application that is not installed, PBS provisions the vnode with that OS or application.

14.1 Definitions

AOE

The environment on a vnode. This may be one that results from provisioning that vnode, or one that is already in place

Provision

To install an OS or application, or to run a script which performs installation and/or setup

Provisioned Vnode

A vnode which, through the process of provisioning, has an OS or application that was installed, or which has had a script run on it

14.2 How Provisioning Works

Provisioning can be performed only on vnodes that have provisioning enabled, shown in the vnode's provision_enable attribute.

Provisioning can be the following:

- Directly installing an OS or application
- Running a script which may perform setup or installation

Each vnode is individually configured for provisioning with a list of available AOEs, in the vnode's resources_available.aoe attribute.

Each vnode's current_aoe attribute shows that vnode's current AOE. The scheduler queries each vnode's aoe resource and current_aoe attribute in order to determine which vnodes to provision for each job.

Provisioning can be used for interactive jobs.

A job's walltime clock starts when provisioning for the job has finished.

14.2.1 Causing Vnodes To Be Provisioned

An AOE can be requested for a job or a reservation. When a job requests an AOE, that means that the job will be run on vnodes running that AOE. When a reservation requests an AOE, that means that the reservation reserves vnodes that have that AOE available. The AOE is instantiated on reserved vnodes only when a job requesting that AOE runs.

When the scheduler runs each job that requests an AOE, it either finds the vnodes that satisfy the job's requirements, or provisions the required vnodes. For example, if SLES is available on a set of vnodes that otherwise suit your job, you can request SLES for your job, and regardless of the OS running on those vnodes before your job starts, SLES will be running at the time the job begins execution.

14.2.2 Using an AOE

When you request an AOE for a job, the requested AOE must be one of the AOEs that has been configured at your site. For example, if the AOEs available on vnodes are "*rhel*" and "*sles*", you can request only those; you cannot request "*suse*".

Your job can run where its requested AOE can be supplied both by provisioning and where the AOE already matches the request. Some of your job chunks can run on the non-provisionable vnodes that already match the requested AOE, and some chunks can run on vnodes that can be provisioned to match the requested AOE.

You can request a reservation for vnodes that have a specific AOE available. This way, jobs needing that AOE can be submitted to that reservation. This means that jobs needing that AOE are guaranteed to be running on vnodes that have that AOE available.

Each reservation can have at most one AOE specified for it. Any jobs that run in that reservation must not request a different AOE from the one requested for the reservation. That is, the job can run in the reservation if it either requests no AOE, or requests the same AOE as the reservation.

14.2.3 Job Substates and Provisioning

When a job is in the process of provisioning, its substate is *provisioning*. This is the description of the substate:

provisioning

The job is waiting for vnode(s) to be provisioned with its requested AOE. Integer value is *71*. See "Job Substates" on page 362 of the PBS Professional Reference Guide for a list of job substates.

The following table shows how provisioning events affect job states and substates:

Table 14-1: Provisioning Events and Job States/Substates

Event	Initial Job State, Substate	Resulting Job State, Substate
Job submitted		*Queued and ready for selection*
Provisioning starts	*Queued, Queued*	*Running, Provisioning*
Provisioning fails to start	*Queued, Queued*	*Held, Held*
Provisioning fails	*Running, Provisioning*	*Queued, Queued*
Provisioning succeeds and job runs	*Running, Provisioning*	*Running, Running*
Internal error occurs	*Running, Provisioning*	*Held, Held*

14.3 Requirements and Restrictions

14.3.1 Host Restrictions

14.3.1.1 Single-vnode Hosts Only

PBS will provision only single-vnode hosts. Do not attempt to use provisioning on hosts that have more than one vnode.

14.3.1.2 Server Host Cannot Be Provisioned

The server host cannot be provisioned: a MoM can run on the server host, but that MoM's vnode cannot be provisioned. The provision_enable vnode attribute, resources_available.aoe, and current_aoe cannot be set on the server host.

14.3.2 AOE Restrictions

Only one AOE can be instantiated at a time on a vnode.

Only one kind of aoe resource can be requested in a job. For example, an acceptable job could make the following request:

```
-l select=1:ncpus=1:aoe=suse+1:ncpus=2:aoe=suse
```

14.3.2.1 Vnode Job Restrictions

A vnode with any of the following jobs will not be selected for provisioning:

- One or more running jobs
- A suspended job
- A job being backfilled around

14.3.2.2 Provisioning Job Restrictions

A job that requests an AOE will not be backfilled around.

14.3.2.3 Vnode Reservation Restrictions

A vnode will not be selected for provisioning for job MyJob if the vnode has a confirmed reservation, and the start time of the reservation is before job MyJob will end.

A vnode will not be selected for provisioning for a job in reservation R1 if the vnode has a confirmed reservation R2, and an occurrence of R1 and an occurrence of R2 overlap in time and share a vnode for which different AOEs are requested by the two occurrences.

14.3.3 Requirements for Jobs

14.3.3.1 If AOE is Requested, All Chunks Must Use Same AOE

If any chunk of a job requests an AOE, all chunks must use that AOE, even if they do not explicitly request an AOE. For example, if your job requests

```
-l select=2:ncpus=1:aoe=suse+4:ncpus=2
```

all chunks must use the *suse* AOE.

If a job requesting an AOE is submitted to a reservation, that reservation must also request the same AOE.

14.4 Using Provisioning

14.4.1 Requesting Provisioning

You request a reservation with an AOE in order to reserve the resources and AOE required to run a job. You request an AOE for a job if that job requires that AOE. You request provisioning for a job or reservation using the same syntax.

You can request an AOE for the entire job/reservation:

-l aoe = <AOE>

Example:

```
-l aoe = suse
```

The *-l <AOE>* form cannot be used with *-l select.*

You can request an AOE for a single-chunk job/reservation:

-l select=<chunk request>:aoe=<AOE>

Example:

```
-ls select=1:ncpus=2:aoe=rhel
```

You can request the same AOE for each chunk of a job/reservation:

-l select=<chunk request>:aoe=<AOE> + <chunk request>:aoe=<AOE>

Example:

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2:aoe=suse
```

You can request the an AOE for some, but not all, chunks of a job/reservation:

-l select=<chunk request>:aoe=<AOE> + <chunk request>

Example:

```
-l select=1:ncpus=1:aoe=suse + 2:ncpus=2
```

14.4.2 Commands and Provisioning

If you try to use PBS commands on a job that is in the *provisioning* substate, the commands behave differently. The provisioning of vnodes is not affected by the commands; if provisioning has already started, it will continue. The following table lists the affected commands:

Table 14-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
`qdel`	(Without force) Job is not deleted
	(With force) Job is deleted
`qsig -s suspend`	Job is not suspended
`qhold`	Job is not held
`qrerun`	Job is not requeued

Table 14-2: Effect of Commands on Jobs in Provisioning Substate

Command	Behavior While in Provisioning Substate
qmove	Cannot be used on a job that is provisioning
qalter	Cannot be used on a job that is provisioning
qrun	Cannot be used on a job that is provisioning

14.4.3 How Provisioning Affects Jobs

A job that has requested an AOE will not preempt another job. Therefore no job will be terminated in order to run a job with a requested AOE.

A job that has requested an AOE will not be backfilled around.

14.5 Caveats and Errors

14.5.1 Requested Job AOE and Reservation AOE Should Match

Do not submit jobs that request an AOE to a reservation that does not request the same AOE. Reserved vnodes may not supply that AOE; your job will not run.

14.5.2 Allow Enough Time in Reservations

If a job is submitted to a reservation with a duration close to the walltime of the job, provisioning could cause the job to be terminated before it finishes running, or to be prevented from starting. If a reservation is designed to take jobs requesting an AOE, leave enough extra time in the reservation for provisioning.

14.5.3 Requesting Multiple AOEs For a Job or Reservation

Do not request more than one AOE per job or reservation. The job will not run, or the reservation will remain unconfirmed.

14.5.4 Held and Requeued Jobs

The job is held with a system hold for the following reasons:

- Provisioning fails due to invalid provisioning request or to internal system error
- After provisioning, the AOE reported by the vnode does not match the AOE requested by the job

The hold can be released by the PBS Administrator after investigating what went wrong and correcting the mistake.

The job is requeued for the following reasons:

- The provisioning hook fails due to timeout
- The vnode is not reported back up

14.5.5 Conflicting Resource Requests

The values of the resources arch and vnode may be changed by provisioning. Do not request an AOE and either arch or vnode for the same job.

14.5.6 Job Submission and Alteration Have Same Requirements

Whether you use the `qsub` command to submit a job, or the `qalter` command to alter a job, the job must eventually meet the same requirements. You cannot submit a job that meets the requirements, then alter it so that it does not.

15 Using Accounting

15.1 Using Accounting

15.1.1 Specifying Accounting String

You can associate an accounting string with your job by setting the value of the Account_Name job attribute. This attribute has no default value. You can set the value of Account_Name at the command line or in a PBS directive:

```
qsub -A <accounting string>
#PBS Account_Name=<accounting string>
```

The <*accounting string*> can be any string of characters; PBS does not attempt to interpret it.

You can use the `qalter` command to change the value of the Account_Name job attribute while the job is queued, but not while the job is running.

15.1.2 Using Comprehensive System Accounting

You can use CSA on Cray systems running CLE 5.2. PBS support for CSA on HPE systems is no longer available. The CSA functionality for HPE systems has been **removed** from PBS.

CSA provides accounting information about user jobs, called user job accounting.

CSA works the same with and without PBS. To run user job accounting, either you must specify the file to which raw accounting information will be written, or an environment variable must be set. The environment variable is ACCT_TMPDIR. This is the directory where a temporary file of raw accounting data is written.

To run user job accounting, you issue the CSA command "`ja <filename>`" or, if the environment variable ACCT_TMPDIR is set, "`ja`". In order to have an accounting report produced, you issue the command "`ja -<options>`" where the options specify that a report should be written and what kind to write. To end user job accounting, you issue the command "`ja -t`"; the `-t` option can be included in the previous set of options. See the man page on `ja` for details.

The starting and ending `ja` commands must be used before and after any other commands you wish to monitor. Here are examples of a command line and a script:

On the command line:

```
qsub -N myjobname -l ncpus=1
    ja myrawfile
    sleep 50
    ja -c > myreport
    ja -t myrawfile
ctrl-D
```

Accounting data for your job (sleep 50) is written to `myreport`.

If you create a job script foo with these commands:

```
#PBS -N myjobname
#PBS -l ncpus=1
ja myrawfile
sleep 50
ja -c > myreport
ja -t myrawfile
```

Then you can run your job script via qsub, to do the same thing as in the previous example:

```
qsub foo
```

15.1.3 Using Dependencies with Accounting

If you need to run end-of-day accounting, you can use dependencies; see section 6.2, "Using Job Dependencies", on page 107

15.1.4 Advice and Caveats for Using Accounting

15.1.4.1 Use an Integrated MPI

Many MPIs are integrated with PBS. PBS provides tools to integrate most of them; a few MPIs supply the integration. When a job is run under an integrated MPI, PBS can track resource usage, signal job processes, and perform accounting for all processes of the job.

When a job is run under an MPI that is not integrated with PBS, PBS is limited to managing the job only on the primary vnode, so resource tracking, job signaling, and accounting happen only for the processes on the primary vnode.

Under Windows, some MPIs such as MPICH are not integrated with PBS.

See section 5.2.1, "Using an Integrated MPI", on page 81.

Index

_NEC_HCA_LIST_IO UG-209
_NEC_HCA_LIST_MPI UG-210
_NECMPI_VE_NODELIST UG-209
_NECMPI_VE_NUM_NODES UG-209
_VENODELIST UG-209

A
accelerator UG-191
accelerator_memory UG-191
accelerator_model UG-191
accounting UG-221
ACCT_TMPDIR UG-221
advance reservation UG-135
    creation UG-137
AOE UG-215
    using UG-216
application licenses
    floating UG-54
    node-locked
        per-CPU UG-55

B
blocking jobs UG-120
Boolean
    format UG-49

C
changing order of jobs UG-168
chunk UG-51, UG-53
chunk-level resource UG-51
commands UG-2
    and provisioning UG-218
comment UG-178
count_spec UG-138
CSA UG-221
cygwin UG-16

D
deleting jobs UG-166
display
    non-running jobs UG-177
documentation
    PBS Professional UG-213
    SELinux UG-213

E
errors
    fgetfilecon UG-213
    filecon UG-213
    malloc UG-213
exclhost UG-65
exclusive UG-65
exit status
    job arrays UG-156

F
fgetfilecon error UG-213
file
    staging UG-31
filecon error UG-213
float
    format UG-50
floating licenses UG-54
format
    Boolean UG-49
    float UG-50
    size UG-50
    string resource value UG-50
    string_array UG-51
free UG-65
freq_spec UG-138

G
group=resource UG-65

H
HCA UG-201
here document UG-22
host channel adapter UG-201

I
identifier UG-12
InfiniBand UG-98, UG-99
instance UG-135
instance of a standing reservation UG-135
Intel MPI
    examples UG-86
interval_spec UG-138

J

ja
- CSA command UG-221

job
- comment UG-178
- definition UG-2
- dependencies UG-107
- identifier UG-12
- identifier syntax UG-150
- submission options UG-24

job array
- identifier UG-149
- range UG-149
- states UG-151

job arrays UG-149
- exit status UG-156
- prologues and epilogues UG-152

job attributes
- setting UG-16

jobs
- changing order UG-168
- deleting UG-166
- moving between queues UG-169
- sending messages to UG-167
- sending signals to UG-168

job-specific ASAP reservation UG-135
job-specific now reservation UG-135
job-specific reservation UG-135
job-specific start reservation UG-135
job-wide resource UG-51, UG-52

L

limits
- resource usage UG-61

M

malloc error UG-213
max_walltime UG-113
min_walltime UG-113
MoM UG-2
monitoring UG-1
moving jobs between queues UG-169
MPI
- Intel MPI
  - examples UG-86
- MPICH2
  - examples UG-100
- MPICH-MX
  - MPD
    - examples UG-93
  - rsh/ssh
    - examples UG-94
- MVAPICH1 UG-98
  - examples UG-98

MPICH UG-89
MPICH2
- examples UG-100

MPICH-MX
- MPD
  - examples UG-93
- rsh/ssh
  - examples UG-94

MPI-OpenMP UG-105
MVAPICH1 UG-98
- examples UG-98

N

naccelerators UG-191
NEC SX-Aurora TSUBASA UG-201
NEC_PROCESS_DIST UG-204
nhcas UG-202
nves UG-202

O

OpenMP UG-103

P

pack UG-65
Parallel Virtual Machine (PVM) UG-102
PBS environmental variables UG-151
PBS_ARRAY_ID UG-151
PBS_ARRAY_INDEX UG-151
pbs_hostn UG-4
pbs_iff UG-213
PBS_JOBID UG-151
pbs_login UG-3
pbs_probe UG-4
pbs_python UG-3
pbs_rdel UG-3
pbs_rstat UG-3
pbs_rsub UG-3
pbs_tclsh UG-4
PBScrayhost UG-191
PBScraylabel UG-191
PBScraynid UG-191
PBScrayorder UG-191
pbsdsh UG-3
pbsfs UG-4
pbsnodes UG-4
pbs-report UG-4
PCIe UG-201
per-CPU node-locked licenses UG-55
printjob UG-4
prologues and epilogues
- job arrays UG-152

provision UG-215

provisioned vnode UG-215
provisioning UG-216
    allowing time UG-219
    and commands UG-218
    AOE restrictions UG-217
    host restrictions UG-216
    requesting UG-218
    using AOE UG-216
    vnodes UG-215
PVM (Parallel Virtual Machine) UG-102

Q

qalter UG-3
qdel UG-3
qdisable UG-4
qenable UG-4
qhold UG-3, UG-118
qmgr UG-4
qmove UG-3, UG-169
qmsg UG-3, UG-167
qorder UG-3, UG-168, UG-169
qrerun UG-4
qrls UG-3, UG-118
qrun UG-4
qselect UG-3, UG-187
qsig UG-3
qstart UG-4
qstat UG-3, UG-118, UG-166, UG-169, UG-172, UG-174, UG-176
qstop UG-4
qsub UG-3
qterm UG-4
queuing UG-1

R

recurrence rule UG-138
report UG-221
requesting provisioning UG-218
reservation
    advance UG-135, UG-137
    degraded UG-135
    deleting UG-143
    instance UG-135
    job-specific UG-135
        ASAP UG-135
        now UG-135
        start UG-135
    setting start time & duration UG-138
    soonest occurrence UG-136
    standing UG-136
        instance UG-135
        soonest occurrence UG-136
    standing reservation UG-138
    submitting jobs UG-146
reservations
    time for provisioning UG-219
resource
    job-wide UG-51, UG-52
Resource_List UG-24
restrictions
    AOE UG-217
    provisioning hosts UG-216
resv_nodes UG-135
run_count UG-25, UG-119

S

scatter UG-65
scheduler UG-2
scheduling UG-1
sequence number UG-149
server UG-2
setting job attributes UG-16
share UG-65
SIGKILL UG-168
SIGNULL UG-168
SIGTERM UG-168
size
    format UG-50
soonest occurrence UG-136
stagein UG-25
stageout UG-25
standing reservation UG-136, UG-138
start reservation UG-135
states
    job array UG-151
string resource value
    format UG-50
string_array
    format UG-51
subjob UG-149
subjob index UG-149
submitting a PBS job UG-11
SX-Aurora UG-201
syntax
    identifier UG-150

T

time between reservations UG-147
tracejob UG-3
TSUBASA UG-201

U

until_spec UG-138
user job accounting UG-221

V

VE UG-201
VE offloading UG-201
ve_cput UG-202, UG-209
ve_mem UG-202, UG-209
VE_NODE_NUMBER UG-209
vector engine UG-201
vector host UG-201
VH UG-201
vnode types UG-49
vnodes
    provisioning UG-215
vscatter UG-65

W

waiting for job completion UG-120



Altair PBS Professional 2021.1

Programmer's Guide

You are reading the Altair PBS Professional 2021.1

Programmer's Guide (PG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation ix

1 PBS Architecture 1

1.1 PBS Components . . . 1

2 Server Functions 5

2.1 Roles and Required Privilege . . . 5
2.2 Batch Server Functions . . . 5
2.3 Server Management . . . 5
2.4 Queue Management . . . 6
2.5 Job Management . . . 6
2.6 Server to Server Requests . . . 11
2.7 Deferred Services . . . 12
2.8 Resource Management . . . 16
2.9 Network Protocol . . . 16

3 Developer Headers and Libraries 19

3.1 Location of API Libraries . . . 19
3.2 Location of Header Files . . . 19
3.3 Developer Package . . . 19
3.4 Batch Interface Library . . . 20
3.5 Example Compilation Line . . . 20

4 Batch Interface Library (IFL) 21

4.1 Interface Library Overview . . . 21
4.2 Batch Library Routines . . . 22

5 TM Library 93

5.1 TM Library Routines . . . 93

6 RM Library 99

6.1 RM Library Routines . . . 99

7 TCL/tk Interface 103

7.1 TCL/tk API Functions . . . 103

8 Hooks 109

8.1 Introduction . . . 109
8.2 How Hooks Work . . . 109
8.3 Interface to Hooks . . . 110

9 Custom Authentication and Encryption Library APIs 121

Index 133

List of APIs

4.3	pbs_alterjob	24
4.4	pbs_asyrunjob	26
4.5	pbs_confirmresv	28
4.6	pbs_connect	30
4.7	pbs_default	32
4.8	pbs_deljob	33
4.9	pbs_delresv	35
4.10	pbs_disconnect	36
4.11	pbs_geterrmsg	37
4.12	pbs_holdjob	38
4.13	pbs_locjob	39
4.14	pbs_manager	41
4.15	pbs_modify_resv	45
4.16	pbs_movejob	47
4.17	pbs_msgjob	49
4.18	pbs_orderjob	51
4.19	pbs_preempt_jobs	52
4.20	pbs_relnodesjob	54
4.21	pbs_rerunjob	55
4.22	pbs_rlsjob	56
4.23	pbs_runjob	57
4.24	pbs_selectjob	59
4.25	pbs_selstat	62
4.26	pbs_sigjob	66
4.27	pbs_statfree	68
4.28	pbs_stathost	69
4.29	pbs_statjob	71
4.30	pbs_statnode	74
4.31	pbs_statque	76
4.32	pbs_statresv	78
4.33	pbs_statrsc	80
4.34	pbs_statsched	82
4.35	pbs_statserver	84
4.36	pbs_statvnode	86
4.37	pbs_submit	88
4.38	pbs_submit_resv	90
4.39	pbs_terminate	92
5.2	tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit, tm_taskinfo, tm_atnode, tm_rescinfo, tm_publish, tm_subscribe, tm_finalize, tm_attach	94
6.2	openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq,	

	fullresp	100
7.2	pbs_tclapi	104
8.4	pbs_module	111
8.5	pbs_stathook(3B)	117
9.1	pbs_auth_set_config	122
9.2	pbs_auth_create_ctx	123
9.3	pbs_auth_destroy_ctx	125
9.4	pbs_auth_get_userinfo	126
9.5	pbs_auth_process_handshake_data	128
9.6	pbs_auth_encrypt_data	130
9.7	pbs_auth_decrypt_data	131

20 PBS Architecture

PBS is a distributed workload management system which manages and monitors the computational workload on a set of one or more computers.

20.1 PBS Components

You can manage one or more machines using PBS. PBS consists of commands, a data service, and the following daemons:

- Server daemon for central management; this daemon runs on Linux only.
- One or more scheduler daemons to schedule jobs; schedulers run on Linux only.
- Communication daemon to manage communication; this daemon also runs only on Linux.
- Job management daemon called MoM to manage each execution host; this daemon can run on Linux or Windows.

The data service runs on Linux only. Commands can run on Linux or Windows.

20.1.1 Single Execution System

If PBS is to manage a single system, all components are installed on that same system. For installation instructions, see the *PBS Professional Installation & Upgrade Guide*.

The following illustration shows how communication works when PBS is on a single host in TPP mode. For more on TPP mode, see "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide.

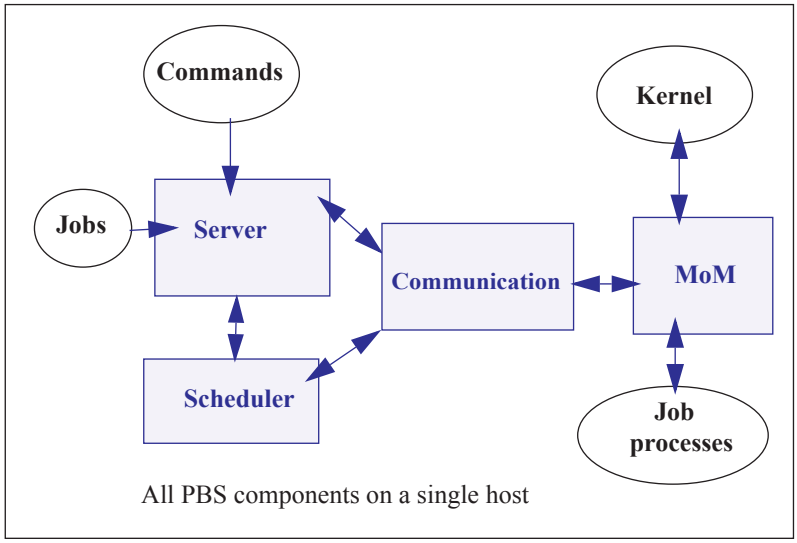


Figure 20-1:PBS daemons on a single execution host

20.1.2 Single Execution System with Front End

The PBS server and default scheduler (`pbs_server` and `pbs_sched`) can run on one system and jobs can execute on another. Job execution is managed by the MoM daemon. The following illustration shows how communication works when the PBS server and scheduler are on a front-end system and MoM is on a separate host, in TPP mode. For more on TPP mode, see "Communication" on page 45 in the PBS Professional Installation & Upgrade Guide.

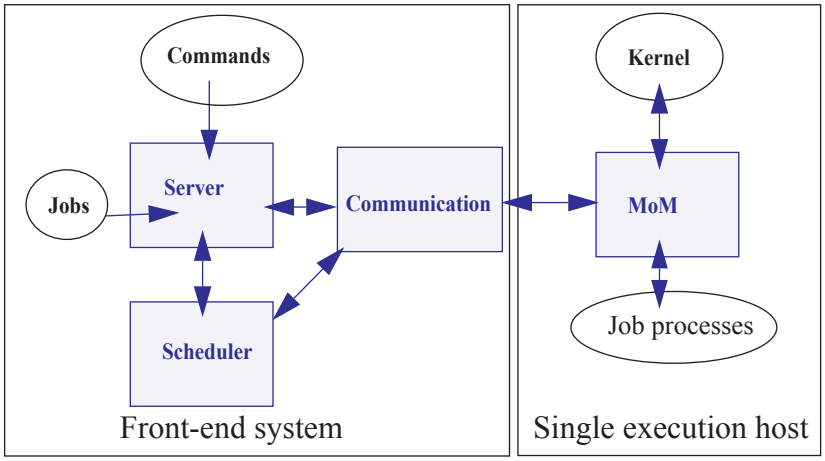


Figure 20-2:PBS daemons on single execution system with front end

20.1.3 Multiple Execution Systems

When you run PBS on several systems, the server (`pbs_server`), the scheduler (`pbs_sched`), and the communication daemon (`pbs_comm`) are installed on a "front end" system, and a MoM (`pbs_mom`) is installed and run on each execution host. The following diagram illustrates this for an eight-host complex in TPP mode.

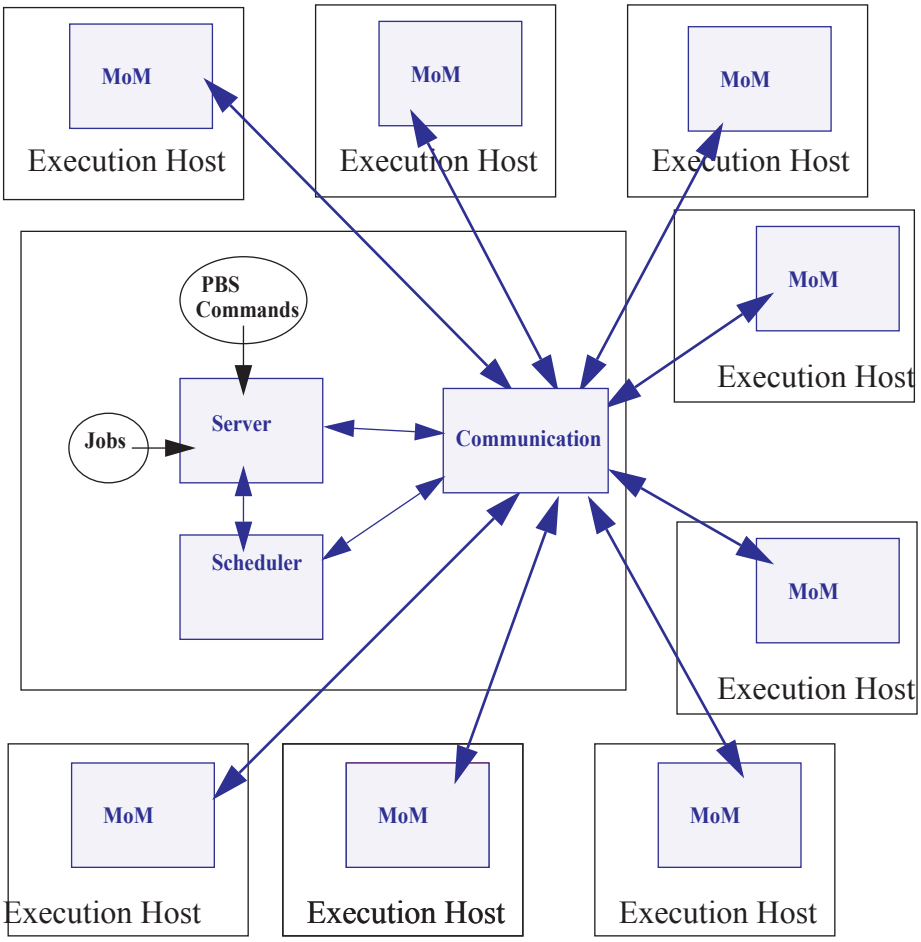


Figure 20-3:Typical PBS daemon locations for multiple execution hosts

20.1.4 Server

The *server* process is the central focus for PBS. In our documentation, it is generally referred to as *the server, the PBS server,* or by the execution name *pbs_server.* All commands and communication with the server are via an *Internet Protocol* (IP) network. The server provides core batch services such as receiving batch job requests, creating batch jobs, modifying jobs, protecting jobs against system crashes, and sending jobs to MoM for execution. One server manages each PBS complex.

20.1.5 Job Executor (MoM)

The *Job Executor* is the component that actually places the job into execution. This daemon, *pbs_mom,* is informally called *MoM* as it is the mother of all executing jobs on that host. MoM places a job into execution when it receives a copy of the job from the server. MoM creates a new session that is as identical to a user login session as is possible. For example, if the user's login shell is `csh`, then MoM creates a session in which `.login` is run as well as `.cshrc`. MoM also returns the job's output to the user. One MoM runs on each execution host (a host where PBS jobs execute).

20.1.6 Scheduler

The *scheduler*, *pbs_sched*, implements the site's policy controlling when each job is run and on which resources. The scheduler queries the MoMs to get the state of system resources, and queries the server to for the list of waiting jobs. See "About Schedulers" on page 89 in the PBS Professional Administrator's Guide.

20.1.7 Communication Daemon

The *communication daemon*, `pbs_comm`, handles communication between the other PBS daemons. For a complete description, see section 4.5, "Inter-daemon Communication Using TPP", on page 49.

20.1.8 Privilege

PBS recognizes separate roles and levels of privilege: Manager role is required for sensitive operations, Operator role can perform various less-sensitive functions, and User role allows access to only the user's own jobs. Root privilege is required for some of the Manager operations; the combination of root privilege and Manager role is called PBS Administrator. See "Security" on page 359 in the PBS Professional Administrator's Guide.

20.1.9 Commands

PBS provides a set of commands for submitting and managing jobs, and for managing PBS. PBS commands are described in "PBS Commands" on page 21 of the PBS Professional Reference Guide.

21 Server Functions

21.1 Roles and Required Privilege

PBS recognizes specific roles and levels of privilege, and these are required for some operations on PBS. For details, see "User Roles and Required Privilege" on page 359 in the PBS Professional Administrator's Guide.

21.2 Batch Server Functions

A batch server provides services in the following ways:

- The server provides a service at the request of a client. Clients are processes that make requests of a batch server. The requests may ask for an action to be performed on one or more jobs, one or more queues, the server itself, etc. Any requests that cannot be successfully completed are rejected. The reason for the rejection is returned in the reply to the client.
- The server provides a deferred service when it detects a change in conditions that it monitors. The server may, depending on conditions being monitored, defer a client service request until a later time. Deferred services include file staging, sending jobs for execution, etc. See section 21.7, "Deferred Services", on page 12.

The server also performs a number of internal bookkeeping functions.

21.3 Server Management

The following sections describe the services provided by a batch server in response to a request from a client. The requests are grouped in the following subsections by the type of object affected by the request: server, queue, job, reservation, vnode, hook, or resource. The batch requests described in this section control the functioning of the batch server. The control is either direct as in the *Shut Down* request, or indirect as when server attributes are modified. For a list of batch request codes, see "Request Codes" on page 395 of the PBS Professional Reference Guide.

21.3.1 Manage Request

The *Manage* request supports `qmgr` and other commands. For more information, see "qmgr" on page 149 of the PBS Professional Reference Guide.

21.3.2 Server Status Request

The status of the server may be requested with a server *Status* request. The batch server will reject the request if the user of the client is not authorized to query the status of the server. If the request is accepted, the server will return a server *Status Reply*. See "qstat" on page 197 of the PBS Professional Reference Guide details of which server attributes are returned to the client.

21.3.3 Starting the PBS Server

A batch request to start a server cannot be sent to a server since the server is not running. Therefore a batch server must be started by a process local to the host on which the server is to run. For how to start the server, see "Server: Starting, Stopping, Restarting" on page 163 in the PBS Professional Installation & Upgrade Guide.

The server recovers the state of managed objects, such as queues and jobs, from the information last recorded by the server. The.treatment of jobs which were in the running state when the server previously shut down is dictated by the start up mode; see "pbs_server" on page 108 of the PBS Professional Reference Guide.

21.3.4 Stopping the PBS Server

The batch server is "shut down" when it no longer responds to requests from clients and does not perform deferred services. The batch server is requested to shut down by sending it a server *Shutdown* request. The server will reject the request from a client not authorized to shut down the server. When the server accepts a shut down request, it will terminate in the manner described in "qterm" on page 233 of the PBS Professional Reference Guide. When shutting down, the server must record the state of all managed objects (jobs, queues, etc.) in non-volatile memory. Jobs which were running will be marked in the secondary state field for possible special treatment when the server is restarted. If checkpoint is supported, any job running at the time of the shut down request whose Checkpoint attribute is not *n*, will be checkpointed. This includes jobs whose Checkpoint attribute value is "unspecified", a value of *u*. If the server receives either a SIGTERM or a SIGSHUTDN signal, the server will act as if it had received a shut down immediate request.

21.4 Queue Management

The following client requests operate on the queues managed by the server.

21.4.1 Queue Status Request

The status of a queue at the server may be requested with a *Queue Status* request. The batch server requires that the following conditions are true:

- The user of the client is authorized to query the status of the designated queue
- The specified queue exists on the server

If the request does not specify a queue, status of all the queues at the server will be returned. When the request is accepted, the server will return a *Queue Status Reply*. See"qstat" on page 197 of the PBS Professional Reference Guide for details of which queue attributes are returned to the client.

21.5 Job Management

The following client requests operate on jobs managed by the server. These requests do not require any special privilege except when the job for which the request is issued is not owned by the user making the request.

21.5.1 Queue Job Request

A *Queue Job* request consists of several subrequests: Initiate *Job Transfer*, *Job Data*, *Job Script*, and *Commit*. The end result of a successful *Queue Job* request is an additional job being managed by the server. The job may have been created by the request or it may have been moved from another server. After the successful request, the job resides in a queue managed by the server. When a queue is not specified in the request, the job is placed in the default queue. The administrator can specify the default queue. We call the queue where the job ends up the *target queue*. The batch server requires that the following conditions are true:

- The client is authorized to create a job in the target queue
- The target queue exists at the server.
- The target queue is enabled.
- If the target queue is an execution queue, no resource requirement of the job exceeds the limits set for the queue
- If the target queue is an execution queue, all resources requested by the job are recognized
- The job requires access to a user identifier that the client is authorized to access

When a job is placed in an execution queue, it is put in the queued state unless one of the following conditions applies:

- The job has an Execution_Time attribute that specifies a time in the future and the Hold_Types attribute has value of *no hold*, in which case the job is placed in the waiting state
- The job has a Hold_Types attribute with a value other than *no hold*, in which case the job is placed in the held state.

When a job is placed in a routing queue, its state may change based on the conditions described in section 21.7.4, “Job Routing”, on page 15.

A server that accepts a *Queue Job* request for a new job will do the following:

- Add the PBS_O_QUEUE variable to the Variable_List attribute of the job and set the value to the name of the target queue
- Add the PBS_JOBID variable to the Variable_List attribute of the job and set the value to the job identifier assigned to the job
- Add the PBS_JOBNAME variable to the Variable_List attribute of the job and set the value to the value of the Job_Name attribute of the job

When the server accepts a *Queue Job* request for an existing job, the server will send a *Track Job* request to the server which created the job.

21.5.2 Job Credential Request

The *Job Credential* sub-request is part of the *Queue Job* request. This sub-request transfers a copy of the credential provided by the authentication facility explained below.

21.5.3 Job Script Request

The *Job Script* sub-request is part of the *Queue Job* request. This sub-request passes a block of the job script file to the receiving server. The script is broken into blocks to prevent having to hold the entire script in memory. Multiple *Job Script* sub-requests may be required to transfer the script file.

21.5.4 Commit Request

The *Commit* sub-request is part of the *Queue Job* request. The *Commit* notifies the receiving server that all parts of the job have been transferred and the receiving server should now assume ownership of the job. Prior to sending the *Commit*, the sending client, command, or another server, is the owner.

21.5.5 Message Job Request

A batch server can be requested to write a string of characters to one or both output streams of an executing job. This request is primarily used by an operator to record a message for the user. The batch server will accept a *Message Job* request if all of the following conditions are true:

- The specified job is in the running state
- The user of the client is authorized to post a message to the designated job
- The specified job is owned by the server

When the server accepts the *Message Job* request, it will forward the request to the primary MoM for the job. Upon receiving the *Message Job* request from the server, the MoM will append the message string, followed by a newline character, to the file or files indicated. If no file is indicated, the message will be written to the standard error of the job.

21.5.6 Locate Job Request

A client may ask a server to return the location of a job that was created by or is owned by the server. When the server accepts the *Locate Job* request, it returns a *Locate Reply*. The request will be accepted if all of the following conditions are true:

- The server owns (manages) the job
- The server created the job
- The server is maintaining a record of the current location of the job

21.5.7 Delete Job Request

A *Delete Job* request asks a server to remove a job from the queue in which it exists and not place it elsewhere. The batch server will accept a *Delete Job* request if all of the following conditions are true:

- The user of the client is authorized to delete the designated job.
- The designated job is owned by the server.
- The designated job is in an eligible state. Eligible states are *queued*, *held*, *waiting*, *running*, and *transiting*.

If the job is in the running state, the server will forward the *Delete Job* request to the primary MoM responsible for the job. The MoM daemon will first send a SIGTERM signal to the job process group. After a delay specified by the delete request, or if not specified, the kill_delay queue attribute, the MoM will send a SIGKILL signal to the job process group. The job is then placed into the *exiting* state. Option arguments exist to specify the delay in seconds between the SIGTERM and SIGKILL signals, as well as to force the deletion of the job even if the node it is running on is not responding.

21.5.8 Modify Job Request

A batch client makes a *Modify Job* request to the server to alter the attributes of a job. The batch server will accept a *Modify Job* request if all of the following conditions are true:

- The user of the client is authorized to make the requested modification to the job.
- The designated job is owned by the server.
- The requested modification is consistent with the state of the job.
- A requested resource change would not exceed the limits of the queue or server.
- An recognized resource is requested for a job in an execution queue.

When the batch server accepts a *Modify Job* request, it will modify all the specified attributes of the job. When the batch server rejects a *Modify Job* request, it will modify none of the attributes of the job.

21.5.9 Run Job Request

The *Run Job* request directs the server to place the specified job into immediate execution. The request is issued by a `qrun` command or by the PBS job scheduler.

21.5.10 Rerun Job Request

To rerun a job is to kill the members of the session (process) group of the job and leave the job in the execution queue. If the Hold_Types attribute is not *NONE* , the job is eligible to be re-scheduled for execution. The server will accept the *Rerun Job* request if all of the following conditions are true:

- The user of the client is authorized to rerun the designated job
- The Rerunable attribute of the job is set to *True*
- The job is in the running state
- The server owns the job

When the server accepts the *Rerun Job* request, the request will be forwarded to the primary MoM responsible for the job, who will then perform the following actions:

1. Send a SIGKILL signal to the session (process) group of the job
2. Send an OBIT notice to the server with resource usage information
3. The server will then requeue the job in the execution queue in which it was executing

If the Hold_Types attribute is not *NONE*, the job will be placed in the *held* state. If the execution_time attribute is a future time, the job will be placed in the *waiting* state. Otherwise, the job is.placed in the *queued* state.

21.5.11 Hold Job Request

A client can request that one or more holds be applied to a job. The batch server will accept a *Hold Job* request if all of the following conditions are true:

- The user of the client is authorized to add any of the specified holds
- The batch server manages the specified job

When the server accepts the *Hold Job* request, it will add each specified hold which is not already present to the value of the Hold_Types attribute of the job. If the job is in the *queued* or *waiting* state, it is placed in the *held* state.

If the job is in *running* state:

If checkpoint / restart is supported by the host system, placing a hold on a running job will cause:

a. The job is checkpointed

b. The resources assigned to the job will be released

c. The job is placed in the held state in the execution queue.

If checkpoint / restart is not supported, the server will only set the requested hold type(s). This will have no effect unless the job is rerun or restarted.

21.5.12 Release Job Request

A client can request that one or more holds be removed from a job. A batch server accepts a *Release Job* request if all of the following conditions are true:

- The user of the client is authorized to add (remove) any of the specified holds.
- The batch server manages the specified job.

When the server accepts the *Release Job* request, it will remove each specified type of hold from the value of the Hold_Types attribute of the job. Normally, the job will then be placed in the queued state, unless another hold type is remaining on the job. However, if all holds have been removed, but the Execution_Time attribute specifies a time in the future, the job is placed in the *waiting* state.

21.5.13 Move Job Request

A client can request a server to move a job to a new destination. The batch server will accept a *Move Job* request if all of the following conditions are true:

- The user of the client is authorized to remove the designated job from the queue in which the job resides
- The user of the client is authorized to submit a job to the new destination
- The designated job is owned by the server
- The designated job is in the *queued*, *held*, or *waiting* state
- The new destination is enabled
- The new destination is accessible. When the server accepts a *Move Job* request, it will
  - Queue the designated job at the new destination.
  - Remove the job from the current queue.

If the destination exists at a different server, the current server will transfer the job to the new server by sending a *Queue Job* request sequence to the target server. The server will ensure that a job is neither lost nor duplicated.

21.5.14 Select Jobs Request

A client is able to request from the server a list of jobs owned by that server that match a list of selection criteria. The request is a *Select Jobs* request. All the jobs owned by the server and which the user is authorized to query are initially eligible for selection. Job attribute and resource relationships listed in the request restrict the selection of jobs. Only jobs which have attributes and resources that meet the specified criteria will be selected. The server will reject the request if the queue portion of a specified destination does not exist on the server. When the request is accepted, the server will return a *Select Reply* containing a list of zero or more jobs that met the selection criteria.

21.5.15 Signal Job Request

A batch client is able to request that the server signal the session (process) group of a job. Such a request is called a *Signal Job* request. The batch server will accept a *Signal Job* request if all of the following conditions are true:

- The user of the client is authorized to signal the job
- The job is in the *running* state, except for the special signal "resume" when the job must be in the *Suspended* state
- The server owns the designated job
- The requested signal is supported by the host operating system. (The kill system call returns [EINVAL].)

When the server accepts a request to signal a job, it will forward the request to the primary MoM daemon responsible for the job, who will then send the signal requested by the client to the all processes in the job's session.

21.5.16 Status Job Request

The status of a job or set of jobs at a destination may be requested with a *Status Job* request. The batch server will accept a *Status Job* request if all of the following conditions are true:

- The user of the client is authorized to query the status of the designated job
- The designated job is owned by the server

When the server accepts the request, it will return a *Job Status* message to the client. See the `qstat` command for details of which job attributes are returned to the client. If the request specifies a job identifier, status will be returned only for that job. If the request specifies a destination identifier, status will be returned for all jobs residing within the specified queue that the user is authorized to query.

21.6 Server to Server Requests

Server to server requests are a special category of client requests. They are only issued to a server by another server.

21.6.1 Track Job Request

A client that wishes to request an action be performed on a job must send a batch request to the server that currently manages the job.

As jobs are routed or moved through the batch network, finding the location of the job can be difficult without a tracking service. The Track Job request forms the basis for this service.

A server that queues a job sends a track job request to the server which created the job.

Additional backup location servers may be defined.

A server that receives a track job request records the information contained therein.

This information is made available in response to a Locate Job request.

21.6.2 Job Dependency

PBS supports job dependencies. A job, the "child", can be declared to be dependent on one or more jobs, the "parents". A parent may have any number of children. The dependency is specified as an attribute via the `qsub` command with the `-W depend=<dependency list>` option.

See "qsub" on page 213 of the PBS Professional Reference Guide for the complete specification of the dependency list, and "Using Job Dependencies", on page 107 of the PBS Professional User's Guide for how to use them.

When a server queues a job with a dependency type of *after*, *afterok*, *afternotok*, or *afterany* in an execution queue, the server will send a *Register Dependent Job* request to the server managing the job specified by the associated job identifier. The request will specify that the server is to register the dependency. This actually creates a corresponding *before* type dependency attribute entry on the parent (e.g. run job X *before* job Y). If the request is rejected because the parent job does not exist, the child job is aborted. If the request is accepted, a *system* hold is placed on the child job. When a parent job with any of the *before...* types of dependency reaches the required state, starts, or terminates, the server executing the parent job sends a *Register Dependent Job* request to the server managing the child job directing it to release the child job. If there are no other dependencies on other jobs, the system hold on the child job is removed. When a child job is submitted with an *on* dependency and the parent is submitted with any of the *before...* types of dependencies, the parent will register with the child. This causes the *on* dependency count to be reduced and a corre-

sponding *after...* dependency to be created for the child job. The result is a pairing between corresponding *before...* and *after...* dependency types. If the parent job terminates so that the child is not released, it is up to the user to correct the situation by either deleting the child job or by correcting the problem with the parent job and resubmitting it. If the parent job is resubmitted, it must have a dependency type of *before*, *beforeok*, *beforenotok*, or *beforeany* specified to connect it to the waiting child job.

21.7 Deferred Services

The PBS server uses an internal mechanism of deferred services to handle some work asynchronously.

Servers use deferred services for these job-related tasks:

- File staging
- Job selection
- Job initiation
- Job routing
- Job exit
- Job abort
- Rerunning jobs after a server restart

The following rules apply to deferred services used for jobs:

- If the server cannot complete a deferred service for a reason which is permanent, the job is aborted
- If the service cannot be completed at the current time but may be completed later, the service is retried a finite number of times

21.7.1 Job Scheduling

If a scheduler's scheduling attribute is *True*, the server requests scheduling cycles for that scheduler.

A scheduler runs in a loop. Inside each loop, it starts up, performs all of its work, and then stops. The scheduling cycle is triggered by a timer and by several possible events.

When there are no events to trigger the scheduling cycle, it is started by a timer. The time between starts is set in each scheduler's scheduler_iteration server attribute. The default value is 10 minutes.

The maximum duration of the cycle is set in each scheduler's sched_cycle_length attribute. A scheduler will terminate its cycle if the duration of the cycle exceeds the value of the attribute. The default value for the length of the scheduling cycle is 20 minutes. A scheduler does not include the time it takes to query dynamic resources in its cycle measurement.

21.7.1.1 Triggers for Scheduling Cycle

A scheduler starts when the following happen:

- The specified amount of time has passed since the previous start
- A job is submitted
- A job finishes execution.
- A new reservation is created
- A reservation starts
- Scheduling is enabled
- The server comes up
- A job is qrun
- A queue is started
- A job is moved to a local queue
- Eligible wait time for jobs is enabled
- A reservation is re-confirmed after being degraded
- A hook restarts the scheduling cycle

While a request for a scheduling cycle is outstanding, the connection to the scheduler is open, and the server will not make another request of the scheduler. If the scheduler scheduling attribute is *False*, the server will not contact the scheduler.

21.7.2 File Staging

PBS provides staging in before execution and staging out after execution. These services are requested via the `-W` option, which sets the stagein and stageout job attributes. The attributes specify the files to be staged:

-W stagein=<execution path>@<input file storage host>:<input file storage path>[,...]

-W stageout=<execution path>@<output file storage host>:<output file storage path>[,...]

The name *execution path* is the name of the file in the job's staging and execution directory (on the execution host). The *execution path* can be relative to the job's staging and execution directory, or it can be an absolute path.

The '@' character separates the execution specification from the storage specification.

The name *storage path* is the file name on the host specified by *storage host*. For stagein, this is the location where the input files come from. For stageout, this is where the output files end up when the job is done. The user must specify a hostname. The name can be absolute, or it can be relative to your home directory on the machine named *storage host*.

For stagein, the direction of travel is **from** *storage path* **to** *execution path*.

For stageout, the direction of travel is **from** *execution path* **to** *storage path*.

A request to stage in a file tells the server to direct MoM to copy a file from the storage location to the execution location. The user must have authority to access the file under the same username under which the job will be run. The storage file is not modified or destroyed. The file will be available before the job is initiated. If a file cannot be staged in for any reason, any files which were staged in are deleted and the job is placed in the wait state and mail is sent to the job owner.

A request to stage out a file tells the server to direct MoM to move a file from the execution location to the storage location. This service is performed after the job has completed execution and regardless of job exit status. If a file cannot be moved, mail is sent to the job owner. If a file is successfully staged out, the local file is deleted.

For file copy mechanism information, see "Setting File Transfer Mechanism" on page 559 in the PBS Professional Administrator's Guide.

21.7.3 Job Start

The server receives *Run Job* requests from a PBS scheduler and the `qrun` command. If a request is authenticated, the server forwards the *Run Job* request to the primary MoM for the job; the primary MoM is chosen by the scheduler or specified in the *Run Job* request.

See the sequence of events in "Sequence of Events for Start of Job" on page 527 in the PBS Professional Administrator's Guide.

The primary MoM creates a session leader that runs the shell program specified in the job's Shell_Path_List attribute.

The pathname of the script and any script arguments are passed as parameters to the shell. If the pathname of the shell is relative, the MoM searches its execution path, $PATH, for the shell. If the pathname of the shell is omitted or is the null string, the MoM uses the login shell for the job owner.

The MoM determines the job owner using the following rules:

1. Choose the username in the User_List job attribute whose hostname matches the execution host.
2. Choose the username in the User_List job attribute which has no associated hostname.
3. Use the username from the Job_Owner job attribute.

The MoM creates and sets the following environment variables in the environment of the session leader of the job:

- PBS_ENVIRONMENT; value set to the string "PBS_BATCH"
- PBS_QUEUE; value set to the name of the execution queue

PBS provides each job with environment variables where the job runs. PBS takes some from the submission environment, and creates others. Job submitters can create environment variables for their jobs. The environment variables created by PBS begin with "PBS_". The environment variables that PBS takes from the job submission environment begin with "PBS_O_".

For example, here are a few of the environment variables that accompany a job submitted by user1, with typical values:

```
PBS_O_HOME=/u/user1
PBS_O_LOGNAME=user1
PBS_O_PATH=/usr/bin:/usr/local/bin:/bin
PBS_O_SHELL=/bin/tcsh
PBS_O_HOST=host1
PBS_O_WORKDIR=/u/user1
PBS_JOBID=16386.server1
```

For a complete list of PBS environment variables, "PBS Environment Variables" on page 399 of the PBS Professional Reference Guide.

The MoM puts all of the variables found in the job's Variable_List attribute, with their corresponding values, into the environment of the job's session leader.

The MoM places the specified limits on host-level resources.

If the job has been run before and is now being rerun, the MoM will ensure that the standard output and standard error streams of the job are appended to the prior streams, if any.

If the MoM and host system support accounting, the MoM will use the value of the Account_Name job attribute as required by the host system.

If the MoM and host system support checkpoint, the MoM will set up checkpointing of the job according to the value of the Checkpoint job attribute. If checkpoint is supported and the Checkpoint attribute requests checkpointing at the minimum interval or at an interval less than the minimum interval for the queue, then checkpoint will be set for an interval given by the queue attribute checkpoint_min.

The MoM will set up the standard output stream and the standard error stream of the job according to the table labeled "How k, sandbox, o, and e Options to qsub Affect stdout and stderr", on page 41 of the PBS Professional User's Guide.

21.7.4 Job Routing

The PBS server performs all job routing tasks. Job routing is described in "Routing Queues" on page 25 in the PBS Professional Administrator's Guide.

If the routing destination is at another server, the current server uses a *Queue Job* request to move the job to the new destination.

21.7.5 Job Exit

When the session leader of a batch job exits, the MoM will perform the following actions in the order listed:

- Place the job in the exiting state.
- Manage the output and error streams of the job, according to "How k, sandbox, o, and e Options to qsub Affect stdout and stderr", on page 41 of the PBS Professional User's Guide.
- If the Mail_Points job attribute contains the value e (*EXIT)*, the server will send mail to the users listed in the Mail_Users job attribute.
- Files are staged out
- Frees the resources allocated to the job. The actual releasing of resources assigned to the processes of the job is performed by the kernel. PBS will free the resources which it reserved for the job by decrementing the resources_used generic data item for the queue and server.
- The job will be removed from the execution queue.

21.7.6 Aborting Job

If the server aborts a job and the Mail_Points job attribute contains the value *a (ABORT)*, the server will send mail to the users listed in the Mail_Users job attribute. The mail message will contain the reason the job was aborted.

21.7.7 Timed Events

The server performs certain events at a specified time or after a specified time delay. Examples:

- A job may have its Execution_Time attribute set to a time in the future. When that time is reached, the job state is updated.
- If the server is unable to make connection with another server, it is to retry after a time specified by the routing queue attribute route_retry_time.

21.7.8 Event Logging

The PBS server maintains an event logfile, the format and contents of which are documented in "Event Logging" on page 546 in the PBS Professional Administrator's Guide.

21.7.9 Accounting

The PBS server maintains an accounting file, the format and contents of which are documented in "Accounting" on page 627 in the PBS Professional Administrator's Guide.

21.8 Resource Management

PBS performs resource allocation at job initiation in two ways depending on the support provided by the host system. Resources are either reservable or non reservable.

21.8.1 Resource Limits

A job submitter can specify limits for resources used by their job, by requesting those amounts. If the job exceeds those limits, it is aborted. The administrator can specify default limits for resource use by jobs. Defaults are specified at the server and at queues. Defaults are applied when limits are not specified by the submitter. The administrator can also use hooks to set resource requests, and thereby limits, in whatever way is useful. See "Allocating Default Resources to Jobs" on page 247 in the PBS Professional Administrator's Guide and the *PBS Professional Plugins (Hooks) Guide*.

If the submitter does not specify a limit for a resource and there is no default, the job can use an unlimited amount of the resource.

21.8.2 Resource Names

For additional information, see "List of Built-in Resources" on page 259 of the PBS Professional Reference Guide where all resource names are documented.

21.9 Network Protocol

The PBS system fits into a client - server model, with a batch client making a request of a batch server and the server replying. This client - server communication necessitates an interprocess communication method and a data exchange (data encoding) format. Since the client and server may reside on different systems, the interprocess communication must be supportable over a network.

While the basic PBS system fits nicely into the client - server model, it also has aspects of a transaction system. When jobs are being moved between servers, it is critical that the jobs are not lost or replicated. Updates to a batch job must be applied once and only once. Thus the operation must be atomic. Most of the client to server requests consist of a single message. Treating these requests as an atomic operation is simple. One request, "Queue Job", is more complex and involves several messages, or subrequests, between the client and the server. Any of these subrequests might be rejected by the server. It is important that either side of the connection be able to abort the request (transaction) without losing or replicating the job. The network connection also might be lost during the request. Recovery from a partially transmitted request sequence is critical. The sequence of recovery from lost connections is discussed in the Queue Job Request description.

The batch system data exchange protocol must be built on top of a reliable stream connection protocol. PBS uses TCP/IP and the socket interface to the network. Either the Simple Network Interface, SNI, or the Detailed Network Interface, DNI, as specified by POSIX.12, Protocol Independent Interfaces, could be used as a replacement.

21.9.1 General DIS Data Encoding

The purpose of the "Data is Strings" encoding is to provide a simple, fast, small, machine-independent form for encoding data to a character string and back again. Because data can be decoded directly into the final internal data structures, the number of data copy operations are reduced. Data items are represented as people think of them, but preceded with a count of the length of each data item.

For small positive integers, it is impossible to tell from the encoded data whether they came from signed or unsigned chars, shorts, ints, or longs. Similarly, for small negative numbers, the only thing that can be determined from the encoded data is that the source datum was not unsigned. It is impossible to tell the word size of the encoding machine, or whether it uses 2's complement, one's complement or sign - magnitude representation, or.even if it uses binary arithmetic. All of the basic C data types are handled. Signed and unsigned chars, shorts, ints, longs produce integers. NULL-terminated and counted strings produce counted strings (with the terminating NULL removed). Floats, doubles, and long doubles produce real numbers. Complex data must be built up from the basic types. Note that there is no type tagging, so the type and sequence of data to be decoded must be known in advance.

22 Developer Headers and Libraries

22.1 Location of API Libraries

All of the libraries containing the PBS API are installed by default in *$PBS_EXEC/lib/*.

22.2 Location of Header Files

Header files used by your code are found in `$PBS_EXEC/include`.

22.3 Developer Package

We provide a development package as an RPM package. The files in this package are useful only for developing and compiling software that interfaces with PBS. They are not required to run PBS.

The development package is named `pbspro-devel` and contains the following headers and libraries:

```
/opt/pbs/include/pbs_error.h
/opt/pbs/include/pbs_ifl.h
/opt/pbs/include/rm.h
/opt/pbs/include/tm.h
/opt/pbs/include/tm_.h
/opt/pbs/lib/libattr.a
/opt/pbs/lib/liblog.a
/opt/pbs/lib/libnet.a
/opt/pbs/lib/libpbs.a
/opt/pbs/lib/libpbs_sched.a
/opt/pbs/lib/libsite.a
```

These files were previously in the `pbspro-server`, `pbspro-client` and `pbspro-execution` packages.

The `pbspro-devel` package also contains the `README` file, like the other PBS Professional RPM packages:

```
/usr/share/doc/pbspro-devel-19.0.0/README.md
```

You can install the `pbspro-devel` package separately from the other PBS packages. This package does not conflict with other PBS packages.

22.4 Batch Interface Library

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides all of the batch service requests used for PBS. The IFL provides a user-callable function corresponding to each batch client command in PBS Professional. Each command generates its own batch service request. You request service from a batch server by calling the appropriate library routine and passing it the required arguments.

The user-callable routines are declared in the header file `PBS_ifl.h`.

We describe the Batch Interface Library in section , “Batch Interface Library (IFL)”, on page 21.

22.4.1 Error Codes

Error codes are available in the header file `PBS_error.h`.

22.4.2 Windows Requirement

To use `pbs_connect()` with Windows, initialize the network library and link with `winsock2`. Call `winsock_init()` before calling `pbs_connect()`, and link against the `ws2_32.lib` library.

22.5 Example Compilation Line

A compile command might look like the following:

```
cc mycode.c -I/usr/pbs/include -L/usr/pbs/lib -lpbs
```

23 Batch Interface Library (IFL)

You can use the commands in this library to build your new batch clients. For example, you can customize your job status display instead of using `qstat`, build new control commands, or use these commands to build jobs that can get their own status or spawn new jobs.

23.1 Interface Library Overview

The primary external application programming interface to PBS is the Batch Interface Library, or IFL. This library provides all of the batch service requests used for PBS. The IFL provides a user-callable function corresponding to each batch client command in PBS Professional. Each command generates its own batch service request. You request service from a batch server by calling the appropriate library routine and passing it the required arguments.

The user-callable routines are declared in the header file `PBS_ifl.h`.

Error codes are available in the header file `PBS_error.h`.

23.1.1 Connection to Server

We provide network connection management routines to be used with our API commands.

You open a connection with a batch server via a call to `pbs_connect()`, which returns a connection handle to the desired server. You can open multiple connections, and you can use each connection for multiple service requests.

When you are finished using a connection to the server, close it via a call to `pbs_disconnect()`.

23.1.2 Authentication

Before it establishes a connection, `pbs_connect()` `fork()`s and `exec()`s a `pbs_iff` process. The `pbs_iff` process provides a credential which validates the user's identity, and prevents a user from spoofing another user's identity. This credential is included in each batch request sent to the server, and consists of the following:

- The user's name from the password file based on running `pbs_iff's` "real uid" value
- The unprivileged, client-side port value associated with the original `pbs_connect()` request message to the server.

The server checks the entries in its connection table for a matching entry which is not yet marked authenticated. The server requires that the matching entry came from a privileged, remote-end, port value.

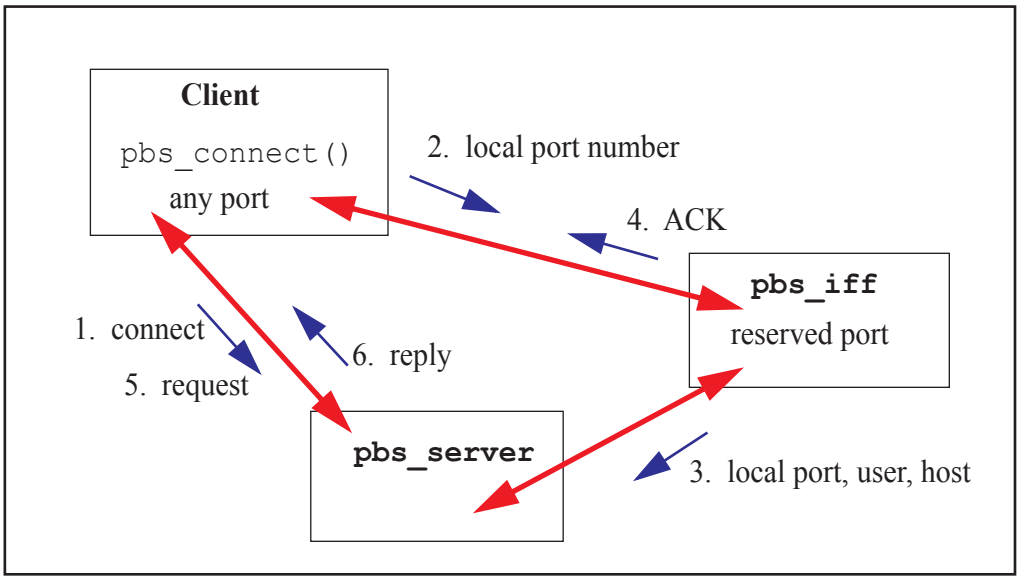


Figure 4-1: Interface Between Client, IFF, and Server

23.1.3 Windows Requirement

To use `pbs_connect()` with Windows, initialize the network library and link with `winsock2`. Call `winsock_init()` before calling `pbs_connect()`, and link against the `ws2_32.lib` library.

23.2 Batch Library Routines

4.3 pbs_alterjob . . . 24
4.4 pbs_asyrunjob . . . 26
4.5 pbs_confirmresv . . . 28
4.6 pbs_connect . . . 30
4.7 pbs_default . . . 32
4.8 pbs_deljob . . . 33
4.9 pbs_delresv . . . 35
4.10 pbs_disconnect . . . 36
4.11 pbs_geterrmsg . . . 37
4.12 pbs_holdjob . . . 38
4.13 pbs_locjob . . . 39
4.14 pbs_manager . . . 41
4.15 pbs_modify_resv . . . 45
4.16 pbs_movejob . . . 47
4.17 pbs_msgjob . . . 49
4.18 pbs_orderjob . . . 51

4.19 pbs_preempt_jobs . . . 52
4.20 pbs_relnodesjob . . . 54
4.21 pbs_rerunjob . . . 55
4.22 pbs_rlsjob . . . 56
4.23 pbs_runjob . . . 57
4.24 pbs_selectjob . . . 59
4.25 pbs_selstat . . . 62
4.26 pbs_sigjob . . . 66
4.27 pbs_statfree . . . 68
4.28 pbs_stathost . . . 69
4.29 pbs_statjob . . . 71
4.30 pbs_statnode . . . 74
4.31 pbs_statque . . . 76
4.32 pbs_statresv . . . 78
4.33 pbs_statrsc . . . 80
4.34 pbs_statsched . . . 82
4.35 pbs_statserver . . . 84
4.36 pbs_statvnode . . . 86
4.37 pbs_submit . . . 88
4.38 pbs_submit_resv . . . 90
4.39 pbs_terminate . . . 92

23.3 pbs_alterjob

alter a PBS batch job

23.3.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_alterjob(int connect, char *jobID, struct attropl *change_list, char *extend)*

23.3.2 Description

Issues a batch request to alter a batch job.

This command generates a *Modify Job* (11) batch request and sends it to the server over the connection specified by connect.

Job state may affect which attributes can be altered. See "qalter" on page 127 of the PBS Professional Reference Guide.

23.3.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job or job array to be altered. Format for a job:

<sequence number>.<server name>

Format for an array job:

<sequence number>[].<server name>

change_list

Pointer to a list of attributes to change. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

extend

Character string for extensions to command. Not currently used.

23.3.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a pointer to a null string.

If the resource is already present in the job's Resource_List attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the value of the attribute or resource.

op

Defines the operation to perform on the attribute or resource. For this command, operators are *SET*, *UNSET*, *INCR*, *DECR.*

23.3.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.3.5 See Also

qalter, qhold, qrls, qsub, pbs_connect, pbs_holdjob, pbs_rlsjob

23.4 pbs_asyrunjob

run an asynchronous PBS batch job

23.4.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_asyrunjob(int connect, char *jobID, char *location, char *extend)*

23.4.2 Description

Issues a batch request to run a batch job.

Generates an *Asynchronous Run Job* (23) request and sends it to the server over the connection specified by connect.

The server validates the request and replies before initiating the execution of the job.

You can use this version of the call to reduce latency in scheduling, especially when the scheduler must start a large number of jobs.

23.4.3 Required Privilege

You must have Manager or Operator privilege to use this command.

23.4.4 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be run.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

location

Location where job should run, and optionally resources to use. Same as `qrun -H`:

-H <vnode specification without resources>

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS applies one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H <vnode specification with resources>

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode specification with resources*, using it instead of the original specification from the user. Any single resource specification results in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

extend

Character string for extensions to command. Not currently used.

23.4.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.4.6 See Also

qrun, pbs_connect, pbs_runjob

23.5 pbs_confirmresv

confirm a PBS reservation

23.5.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_confirmresv(int connect, char *reservationID, char *location, unsigned long start_time, char *extend)*

23.5.2 Description

Issues a batch request to confirm a PBS advance, standing, or maintenance reservation.

This function generates a *Confirm Reservation* (75) batch request and sends it to the server over the connection specified by connect.

23.5.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be confirmed.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

location

String describing vnodes and resources to be used for reservation. Format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

start_time

Unsigned long containing start time in seconds since epoch. Used only for ASAP reservations (reservations created by using `pbs_rsub -W qmove=<jobID>` on an existing job).

extend

Character string for specifying confirmation/non-confirmation action:

- To confirm a normal reservation, pass in *PBS_RESV_CONFIRM_SUCCESS*.
- To have an unconfirmed reservation deleted, pass in *PBS_RESV_CONFIRM_FAIL*.
- To have the scheduler set the time when it will try to reconfirm a degraded reservation, pass in *PBS_RESV_CONFIRM_FAIL*.

23.5.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.5.5 See Also

pbs_rsub, pbs_connect

23.6 pbs_connect

return a connection handle from a PBS batch server

23.6.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_connect(char *server)*

23.6.2 Description

This function establishes a virtual stream (TCP/IP) connection with the specified batch server.

Returns a connection handle.

`pbs_connect()` determines whether or not the complex has a failover server configured. It also determines which server is the primary and which is the secondary.

23.6.3 Arguments

server

Specifies name of server to connect to. Format:

<hostname>[:<port>]

If you do not specify a port, PBS uses the default.

If *server* is a null pointer or a null string, this function opens a connection to the default server. The default server is specified in the PBS_DEFAULT environment variable or the PBS_SERVER parameter in `/etc/pbs.conf`.

23.6.4 Usage

Use this function to establish a connection handle to the desired server before calling any of the other `pbs_*` API functions. They will send their batch requests over the connection established by this function. You can send multiple requests over one connection.

23.6.5 Cleanup

After you are done using the connection handle, close the connection via a call to `pbs_disconnect()`.

23.6.6 Side Effects

The global variable *pbs_server* is declared in `pbs_ifl.h`. This variable is set on return to point to the server name to which `pbs_connect()` connected or attempted to connect.

23.6.7 Windows Requirement

In order to use `pbs_connect()` with Windows, initialize the network library and link with `winsock2`. To do this, call `winsock_init()` before calling `pbs_connect()`, and link against the `ws2_32.lib` library.

23.6.8 Return Value

On success, the routine returns a connection handle which is a non-negative integer.

If an error occurred, the routine returns -1, and the error number is available in the global integer pbs_errno.

23.6.9 See Also

qsub, pbs_alterjob, pbs_deljob, pbs_disconnect, pbs_geterrmsg, pbs_holdjob, pbs_locjob, pbs_manager, pbs_modify_resv, pbs_movejob, pbs_msgjob, pbs_rerunjob, pbs_rlsjob, pbs_runjob, pbs_selectjob, pbs_selstat, pbs_sigjob, pbs_statjob, pbs_statque, pbs_statresv, pbs_statsched, pbs_statserver, pbs_submit, pbs_submit_resv, pbs_terminate, pbs_server

23.7 pbs_default

return the name of the default PBS server

23.7.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_default()*

23.7.2 Description

Returns a pointer to a character string containing the name of the default PBS server.

The default server is specified in the PBS_DEFAULT environment variable or the PBS_SERVER parameter in `/etc/pbs.conf`.

23.7.3 Return Value

On success, returns a pointer to a character string containing the name of the default PBS server. You do not need to free the character string.

Returns null if it cannot determine the name of the default server.

23.8 pbs_deljob

delete a PBS batch job

23.8.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_deljob(int connect, char *jobID, char *extend)*

23.8.2 Description

Issues a batch request to delete a batch job.

This function generates a *Delete Job* (6) batch request and sends it to the server over the connection specified by connect.

If the batch job is running, the MoM sends the SIGTERM signal followed by SIGKILL.

If the batch job is deleted by a user other than the job owner, PBS sends mail to the job owner.

23.8.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job, job array, subjob, or range of subjobs to be deleted.

Format for a job:

<sequence number>.<server name>

Format for an array job:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>][.<server name>]

Format for a range of subjobs:

<sequence number>[<first>-<last>][.<server name>]

extend

Character string for extensions to command. If the string is not null, it is appended to the message mailed to the job owner.

23.8.4 Return Value

The routine returns *0* (*zero*) on success.

On error, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.8.5 See Also

qdel, pbs_connect

23.9 pbs_delresv

delete a reservation

23.9.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_delresv(int connect, char *reservationID, char *extend)*

23.9.2 Description

Issues a batch request to delete a reservation.

This function generates a *Delete Reservation* (72) batch request and sends it to the server over the connection specified by connect.

If the reservation is in state *RESV_RUNNING*, and there are jobs in the reservation queue, those jobs are deleted before the reservation is deleted.

23.9.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be deleted.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

extend

Character string for extensions to command. Not currently used.

23.9.4 Return Value

The routine returns *0* (*zero*) on success.

On error, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.9.5 See Also

pbs_rdel, pbs_connect

23.10 pbs_disconnect

disconnect from a PBS batch server

23.10.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

int pbs_disconnect(int connect)

23.10.2 Description

Closes the virtual stream connection to a PBS batch server. Connection was previously returned from a call to `pbs_connect()`.

23.10.3 Arguments

connect

Connection handle to be closed. Return value of `pbs_connect()`. Specifies connection used earlier to send batch requests to server.

23.10.4 Return Value

The routine returns *0* (*zero*) after successfully closing the connection.

If an error occurred, the routine returns -1, and the error number is available in the global integer pbs_errno.

23.10.5 See Also

pbs_connect

23.11 pbs_geterrmsg

get error message for most recent PBS batch operation

23.11.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_geterrmsg(int connect)*

23.11.2 Description

Returns most recent error message text associated with a batch server request.

If a preceding batch interface library call over the connection specified by connect returned an error from the server, the server may have created an associated text message. If there is a text message, this function returns a pointer to the text message.

23.11.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to request error message from server.

23.11.4 Return Value

If the server returned an error and created an error text string in reply to a previous batch request, this function returns a pointer to the text string. The text string is null-terminated.

If the server does not have an error text string, this function returns a null pointer.

The text string is a global variable; you do not need to free it.

23.11.5 See Also

pbs_connect

23.12 pbs_holdjob

place a hold on a PBS batch job

23.12.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_holdjob(int connect, char *jobID, char *hold_type, char *extend)*

23.12.2 Description

Issues a batch request to place a hold on a job or job array.

This function generates a *Hold Job* (7) batch request sends it to the server over the connection specified by connect.

23.12.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job which is to be held.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

hold_type

Type of hold to apply to job or job array. Valid values are defined in `pbs_ifl.h`. If *hold_type* is a null pointer or points to a null string, PBS applies a *User* hold to the job or job array.

extend

Character string for extensions to command. Not currently used.

23.12.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.12.5 See Also

qhold, pbs_connect, pbs_alterjob, pbs_rlsjob

23.13 pbs_locjob

return current location of a PBS batch job

23.13.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_locjob(int connect, char *jobID, char *extend)*

23.13.2 Description

Issues a batch request to locate a batch job or job array.

This function generates a *Locate Job* (8) batch request and sends it to the server over the connection specified by connect.

If the server currently manages the batch job, or knows which server does currently manage the job, the server returns the location of the job.

23.13.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID
: ID of job to be located.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

extend
: Character string for extensions to command. Not currently used.

23.13.4 Cleanup

The character string returned by `pbs_locjob()` is allocated by `pbs_locjob()`. You must free it via a call to `free()`.

23.13.5 Return Value

On success, returns a pointer to a character string containing current location. Format:

<server name>

On failure, returns a null pointer, and the error number is available in the global integer pbs_errno.

23.13.6 See Also

pbs_connect

23.14 pbs_manager

modify a PBS batch object

23.14.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_manager(int connect, int command, int object_type, char *object_name, struct attropl *attrib_list, char *extend)*

23.14.2 Description

Issues a batch request to perform administrative functions at a server.

Generates a *Manager* (9) batch request and sends it to the server over the connection specified by connect.

You can use this to create, delete, and set attributes of objects such as queues.

23.14.3 Required Privilege

This function requires Manager or Operator privilege depending on the operation, and root privilege when used with hooks.

When not used with hooks:

- Functions MGR_CMD_CREATE and MGR_CMD_DELETE require PBS Manager privilege.
- Functions MGR_CMD_SET and MGR_CMD_UNSET require PBS Manager or Operator privilege.

When used with hooks:

- All commands require root privilege on the server host.
- Functions MGR_CMD_IMPORT, MGR_CMD_EXPORT, and MGR_OBJ_HOOK are used only with hooks, and therefore require root privilege on the server host.
- Hook commands are run at the server host.

23.14.4 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

command
: Operation to be performed. Valid values are specified in `pbs_ifl.h`.

object_type
: Specifies type of object on which command is to operate. Valid values are specified in `pbs_ifl.h`.

object_name
: Name of object on which to operate.

attrib_list

Pointer to a list of attributes to be operated on. Each attribute is described in an attropl structure, defined in pbs_ifl.h as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

extend

Character string for extensions to command. Not currently used.

23.14.4.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

If the resource is already present in the object's attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the new value of the attribute or resource. For parameterized limit attributes, this string contains all parameters for the attribute.

op

Defines the manner in which the new value is assigned to the attribute or resource. The operators used for this function are *SET*, *UNSET*, *INCR*, *DECR*.

23.14.5 Usage for Hooks

When importing a hook or hook configuration file:

- Set *command* to *MGR_CMD_IMPORT*
- Set *object_type* to *SITE_HOOK* (or *PBS_HOOK* if you are importing a configuration file for a built-in hook; you cannot import a built-in hook)
- Set *object_name* to the name of the hook
- In one attropl structure:
  - Set *name* to "*content-type*"
  - Set *value* to "*application/x-python*" for a hook, or "*application/x-config*" for a configuration file
- In another attropl structure:
  - Set *name* to "*content-encoding*"
  - Set *value* to "*default*" or "*base64*"
- In a third attropl structure:
  - Set *name* to "*input-file*"
  - Set *value* to the name of the input file
- Set *op* to *SET*

When exporting a hook or hook configuration file:

- Set *command* to *MGR_CMD_EXPORT*
- Set *object_type* to *SITE_HOOK* (or *PBS_HOOK* if you are exporting a configuration file for a built-in hook; you cannot export a built-in hook)
- Set *object_name* to the name of the hook
- In one attropl structure:
  - Set *name* to "*content-type*"
  - Set *value* to "*application/x-python*" for a hook, or "*application/x-config*" for a configuration file
- In another attropl structure:
  - Set *name* to "*content-encoding*"
  - Set *value* to "*default*" or "*base64*"
- In a third attropl structure:
  - Set *name* to "*output-file*"
  - Set *value* to the name of the output file
- Set *op* to *SET*

See the *PBS Professional Hooks Guide*.

23.14.6 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.14.7 See Also

qmgr, pbs_connect

23.15 pbs_modify_resv

modify a PBS reservation

23.15.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_modify_resv(int connect, char *reservationID, struct attropl *attrib_list, char *extend)*

23.15.2 Description

Issues a batch request to modify a reservation.

Generates a *Modify Reservation* (91) batch request and sends it to the server over the connection specified by connect.

23.15.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

reservationID

Reservation to be modified.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

attrib_list

Pointer to a list of attributes to modify. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the reservation; the action depends on the attribute.

extend

Character string for extensions to command. Not currently used.

23.15.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

If the resource is already present in the reservation's Resource_List attribute, the value is altered as specified. Otherwise the resource is added.

value

Points to a string containing the value of the attribute or resource.

op

Operator. The only allowed operator for this function is *SET*.

23.15.4 Return Value

On success, returns a character string containing the reservation ID assigned by the server.

On failure, returns a null pointer, and the error number is available in the global integer pbs_errno.

23.15.5 Cleanup

The space for the reservation ID string is allocated by `pbs_modify_resv()`.

Release the reservation ID via a call to `free()` when no longer needed.

23.15.6 See Also

pbs_rsub, pbs_connect

23.16 pbs_movejob

move a PBS batch job to a new destination

23.16.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_movejob(int connect, char *jobID, char *destination, char *extend)*

23.16.2 Description

Issues a batch request to move a job or job array to a new destination.

Generates a *Move Job* (12) batch request and sends it to the server over the connection specified by connect.

Moves specified job or job array from its current queue and server to the specified queue and server.

You cannot move a job in the Running, Transiting, or Exiting states.

23.16.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job to be moved.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

destination

New location for job or job array. Formats:

<queue name>@<server name>

Specified queue at specified server

<queue name>

Specified queue at default server

@<server name>

Default queue at specified server

@default

Default queue at default server

(null pointer or null string)

Default queue at default server

extend

Character string for extensions to command. Not currently used.

23.16.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.16.5 See Also

qmove, pbs_connect

23.17 pbs_msgjob

record a message for a running PBS batch job

23.17.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_msgjob(int connect, char *jobID, int file, char *message, char *extend)*

23.17.2 Description

Issues a batch request to write a message in one or more output files of a batch job.

Generates a *Message Job* (10) batch request and sends it to the server over the connection specified by connect.

You can write a message into a job's `stdout` and/or `stderr` files. Can be used on jobs or subjobs, but not job arrays or ranges of subjobs.

23.17.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID
: ID of job into whose output file(s) to write.

Format for a job:

<sequence number>.<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

file
: Indicates whether to write to `stdout`, `stderr`, or both:

1
: Writes to `stdout`

2
: Writes to `stderr`

3
: Writes to `stdout` and `stderr`

message
: Character string to be written to output file(s).

extend
: Character string for extensions to command. Not currently used.

23.17.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.17.5 See Also

qmsg, pbs_connect

23.18 pbs_orderjob

swap positions of two PBS batch jobs

23.18.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_orderjob(int connect, char *jobID1, char *jobID2, char *extend)*

23.18.2 Description

Issues a batch request to swap the positions of two jobs.

Generates an *Order Job* (50) batch request and sends it to the server over the connection specified by connect.

Can be used on jobs and job arrays. Can be used on jobs in different queues. Both jobs must be at the same server.

You cannot swap positions of jobs that are running.

23.18.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID1, jobID2

IDs of jobs to be swapped.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

extend

Character string for extensions to command. Not currently used.

23.18.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.18.5 See Also

qmove, qorder, qsub, pbs_connect

23.19 pbs_preempt_jobs

preempt a list of jobs

23.19.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*preempt_job_info *pbs_preempt_jobs(int connect, char **jobID_list)*

23.19.2 Description

Sends the server a list of jobs to be preempted.

Sends a *Preempt Jobs* (93) batch request to the server over the connection specified by connect.

Returns a list of preempted jobs along with the method used to preempt each one.

23.19.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID_list

List of job IDs to be preempted, as a null-terminated array of pointers to strings.

Format for a job ID:

<sequence number>.<server name>

Format for a job array ID:

<sequence number>[].<server name>

For example:

```
const char *joblist[3];
joblist[0]="123@myserver";
joblist[1]="456@myserver";
joblist[2]=NULL;
```

23.19.4 Return Value

Returns a list of preempted jobs. Each job is represented in a preempt_job_info structure, which has the following fields:

job_id

The job ID, in a char*

preempt_method

How the job was preempted, in a char:

S

The job was preempted using suspension.

C

The job was preempted using checkpointing.

R
: The job was preempted by being requeued.

D
: The job was preempted by being deleted.

0 (zero)
: The job could not be preempted.

23.19.5 Cleanup

You must free the list of preempted jobs by passing it directly to `free()`.

23.20 pbs_relnodesjob

release some or all of the non-primary-execution-host vnodes assigned to a PBS job

23.20.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_relnodesjob (int connect, char *jobID, char *vnode_list, char *extend)*

23.20.2 Description

Issues a batch request to release some or all of the vnodes of a batch job.

Generates a *RelnodesJob* (90) batch request and sends it to the server over the connection specified by connect.

You cannot release vnodes on the primary execution host.

You can use this on jobs and subjobs, but not on job arrays or ranges of subjobs.

23.20.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job or subjob whose vnodes are to be released.

Format for a job:

<sequence number>.<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

vnode_list

List of vnode names separated by plus signs ("+").

If *vnode_list* is a null pointer, this specifies that all the vnodes assigned to the job that are not on the primary execution host are to be released.

extend

Character string for extensions to command. Not currently used.

23.20.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.20.5 See Also

pbs_connect

23.21 pbs_rerunjob

requeue a PBS batch job

23.21.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_rerunjob(int connect, char *jobID, char *extend)*

23.21.2 Description

Issues a batch request to requeue a batch job, job array, subjob, or range of subjobs.

Generates a *Rerun Job* (14) batch request and sends it to the server over the connection specified by connect.

You cannot requeue a job that is marked as not rerunnable (the Rerunable attribute is *False*).

23.21.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be requeued.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

extend

Character string for extensions to command. Not currently used.

23.21.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.21.5 See Also

qrerun, pbs_connect

23.22 pbs_rlsjob

release a hold on a PBS batch job

23.22.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_rlsjob(int connect, char *jobID, char *hold_type, char *extend)*

23.22.2 Description

Issues a batch request to release a hold on a job or job array.

Generates a *Release Job* (13) batch request and sends it to the server over the connection specified by connect.

23.22.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

jobID

ID of job which is to have a hold released.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

hold_type

Type of hold to remove from job or job array. Valid values are defined in `pbs_ifl.h`. If *hold_type* is a null pointer or points to a null string, PBS removes a *User* hold from the job or job array.

extend

Character string for extensions to command. Not currently used.

23.22.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.22.5 See Also

qhold, qrls, pbs_connect, pbs_holdjob

23.23 pbs_runjob

run a PBS batch job

23.23.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_runjob(int connect, char *jobID, char *location, char *extend)*

23.23.2 Description

Issues a batch request to run a batch job.

Generates a *Run Job* (15) batch request and sends it to the server over the connection specified by connect.

If no file stagein is required, the server replies when the job has started execution. If file stagein is required, the server replies when staging is started.

23.23.3 Required Privilege

You must have Operator or Administrator privilege to use this command.

23.23.4 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to run.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

location

Location where job should run, and optionally resources to use. Same as `qrun -H`:

-H <vnode specification without resources>

The *vnode specification without resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>[+<vnode name> ...]

Example:

```
-H (VnodeA+VnodeB)+(VnodeC)
```

PBS applies one requested chunk from the job's selection directive in round-robin fashion to each *vchunk* in the list. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

-H <vnode specification with resources>

The *vnode specification with resources* has this format:

(<vchunk>)[+(<vchunk>) ...]

where *vchunk* has the format

<vnode name>:<vnode resources>[+<vnode name>:<vnode resources> ...]

and where *vnode resources* has the format

<resource name>=<value>[:<resource name>=<value> ...]

Example:

```
-H (VnodeA:mem=100kb:ncpus=1) +(VnodeB:mem=100kb:ncpus=2+VnodeC:mem=100kb)
```

PBS creates a new selection directive from the *vnode specification with resources*, using it instead of the original specification from the user. Any single resource specification results in the job's original selection directive being ignored. Each *vchunk* must be sufficient to run the job's corresponding chunk, otherwise the job may not execute correctly.

If the job being run requests `-l place=exclhost`, take extra care to satisfy the `exclhost` request. Make sure that if any vnodes are from a multi-vnoded host, all vnodes from that host are allocated. Otherwise those vnodes can be allocated to other jobs.

extend

Character string for extensions to command. Not currently used.

23.23.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.23.6 See Also

qrun, pbs_asyrunjob, pbs_connect

23.24 pbs_selectjob

select PBS batch jobs according to specified criteria

23.24.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char **pbs_selectjob(int connect, struct attropl *criteria_list, char *extend)*

23.24.2 Description

`pbs_selectjob()` issues a batch request to select jobs that meet specified criteria, and returns an array of job IDs that meet the specified criteria.

This command generates a *Select Jobs* (16) batch request and sends it to the server over the connection handle specified by connect.

By default, `pbs_selectjob()` returns all batch jobs for which the user is authorized to query status. You filter the jobs by specifying values for job attributes and resources. You send a linked list of attributes with associated values and operators. Job attributes are listed in "Job Attributes" on page 328 of the PBS Professional Reference Guide.

Returns a list of jobs that meet all specified criteria.

23.24.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

criteria_list

Pointer to a list of attributes to use as selection criteria. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

If criteria_list itself is null, you are not using attributes or resources as selection criteria.

extend

Character string where you can specify limits or extensions of your search.

23.24.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

op

Defines the operator in the logical expression:

<existing value> <operator> <specified limit>

Jobs for which the logical expression evaluates to *True* are selected.

For this command, op can be *EQ, NE, GE, GT, LE, LT.*

23.24.4 Querying States

You can select jobs in more than one state using a single request, by listing all states you want returned. For example, to get jobs in *Held* and *Waiting* states:

- Fill in criteria_list->name with "*job_state*"
- Fill in criteria_list->value with "*HW*" for *Held* and *Waiting*

23.24.5 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

23.24.5.1 Querying Finished and Moved Jobs

To get information on finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_selectjob ( ..., ..., <extend characters>) ...
```

To get information on finished jobs only:

- Add the '*x*' character to the extend parameter
- Fill in criteria_list->name with "*ATTR_state*"
- Fill in criteria_list->value with "*FM*" for *Finished* and *Moved*

Subjobs are not considered finished until the parent array job is finished.

23.24.5.2 Querying Job Arrays and Subjobs

To query only job arrays (not jobs or subjobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"

To query only job arrays and subjobs (not jobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"
- Add the '*T*' or '*t*' character to the extend parameter

To query only jobs and subjobs (not job arrays), add the '*T*' or '*t*' character to the extend parameter.

23.24.6 Return Value

The return value is a pointer to a null-terminated array of character pointers. Each character pointer in the array points to a character string which is a job ID in the form:

<sequence number>.<server>@<server>

If no jobs met the criteria, the first pointer in the array is null.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.24.7 Cleanup Required

The returned array of character pointers is malloc()'ed by `pbs_selectjob()`. When the array is no longer needed, you must free it via a call to `free()`.

23.24.8 See Also

pbs_alterjob, pbs_connect, qselect

23.25 pbs_selstat

get status of selected PBS batch jobs

23.25.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_selstat(int connect, struct attropl *criteria_list, struct attrl *output_attribs, char *extend)*

23.25.2 Description

Issues a batch request to get the status of jobs which meet the specified criteria.

Generates a *Select Status* (51) batch request and sends it to the server over the connection specified by connect.

Returns a list of batch_status structures for jobs that meet the selection criteria.

This function is a combination of `pbs_selectjob()` and `pbs_statjob()`.

By default this gives status for all jobs for which you are authorized to query status. You can filter the results by specifying selection criteria.

23.25.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

criteria_list

Pointer to a list of selection criteria, which are attributes and resources with required values. If this list is null, you are not filtering your results via selection criteria. Each attribute or resource is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

If criteria_list itself is null, you are not using attributes or resources as selection criteria.

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string where you can specify limits or extensions of your selection.

23.25.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. For parameterized limit attributes, this string contains all parameters for the attribute.

op

Specifies the test to be applied to the attribute or resource. The operators are EQ, NE, GE, GT, LE, LT.

23.25.3.2 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be null.

next

Points to next attribute in list. A null pointer terminates the list.

23.25.4 Querying States

You can select jobs in more than one state using a single request, by listing all states you want returned. For example, to get jobs in *Held* and *Waiting* states:

- Fill in criteria_list->name with "*job_state*"
- Fill in criteria_list->value with "*HW*" for *Held* and *Waiting*

23.25.5 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

23.25.5.1 Querying Finished and Moved Jobs

To get information on finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_selstat ( ..., ..., <extend characters>) ...
```

To get information on finished jobs only:

- Add the '*x* character to the extend parameter
- Fill in criteria_list->name with "*ATTR_state*"
- Fill in criteria_list->value with "*FM*" for *Finished* and *Moved*

For example:

```
criteria_list->name = ATTR_state;
criteria_list->value = "FM";
criteria_list->op = EQ;
pbs_selstat ( ..., criteria_list, ..., extend) ...
```

Subjobs are not considered finished until the parent array job is finished.

23.25.5.2 Querying Job Arrays and Subjobs

To query only job arrays (not jobs or subjobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"

To query only job arrays and subjobs (not jobs):

- Fill in criteria_list->name with "*array*"
- Fill in criteria_list->value with "*True*"
- Add the '*T*' or '*t*' character to the extend parameter

To query only jobs and subjobs (not job arrays), add the '*T*' or '*t*' character to the extend parameter.

23.25.6 Return Value

Returns a pointer to a list of batch_status structures for jobs that meet the selection criteria. If no jobs meet the criteria or can be queried for status, returns the null pointer.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.25.6.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.25.7 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

23.25.8 See Also

qselect, qstat, pbs_connect, pbs_selectjob, pbs_statfree, pbs_statjob

23.26 pbs_sigjob

send a signal to a PBS batch job

23.26.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_sigjob(int connect, char *jobID, char *signal, char *extend)*

23.26.2 Description

Issues a batch request to send a signal to a batch job.

Generates a *Signal Job* (18) batch request and sends it to the server over the connection specified by connect.

You can send a signal to a job, job array, subjob, or range of subjobs.

The batch server sends the job the specified signal.

The job must be in the running or suspended state.

23.26.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

jobID

ID of job to be signaled.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

signal

Name of signal to send to job. Can be alphabetic, with or without SIG prefix. Can be signal number.

The following special signals are all lower-case, and have no associated signal number:

admin-suspend

Suspends a job and puts its vnodes into the *maintenance* state. The job is put into the *S* state and its processes are suspended.

admin-resume

Resumes a job that was suspended using the *admin-suspend* signal, without waiting for scheduler. Cannot be used on jobs that were suspended with the *suspend* signal. When the last admin-suspended job has been admin-resumed, the vnode leaves the maintenance state.

suspend

Suspends specified job(s). Job goes into *suspended (S)* state.

resume
Marks specified job(s) for resumption by scheduler when there are sufficient resources. Cannot be used on jobs that were suspended with the *admin_suspend* signal.

If the signal is not recognized on the execution host, no signal is sent and an error is returned.

extend
Character string for extensions to command. Not currently used.

23.26.4 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.26.5 See Also

qsig, pbs_connect

23.27 pbs_statfree

free a PBS status object

23.27.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

23.27.2 Description

Frees the specified PBS status object returned by PBS API routines such as `pbs_statque()`, `pbs_statserver()`, `pbs_stathook()`, etc.

23.27.3 Arguments

psj

Pointer to the batch_status structure to be freed.

23.27.3.1 The batch_status Structure

The batch_status structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.27.4 Return Value

No return value.

23.28 pbs_stathost

get status of PBS execution host(s)

23.28.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_stathost(int connect, char *target, struct attrl *output_attribs, char *extend)*

23.28.2 Description

Issues a batch request to get the status of PBS execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host or all execution hosts. If an execution host has multiple vnodes, this command reports aggregated information from the vnodes for that host.

23.28.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target

Name of execution host whose attributes are to be reported. If this argument is a null pointer or points to a null string, returns attributes of all execution hosts known to the server.

output_attribs

Pointer to a list of attributes to return. If this argument is null, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string for extensions to command. Not currently used.

23.28.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

23.28.4 Return Value

Returns a pointer to a list of batch_status structures describing the execution host(s).

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.28.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.28.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.28.6 See Also

qstat, pbs_connect, pbs_statfree

23.29 pbs_statjob

get status of PBS batch jobs

23.29.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statjob(int connect, char *ID, struct attrl *output_attribs, char *extend)*

23.29.2 Description

Issues a batch request to get the status of a specified batch job, a list of batch jobs, or the batch jobs at a queue or server.

Generates a *Status Job* (19) batch request and sends it to the server over the connection specified by connect.

You can query status of jobs, job arrays, subjobs, and ranges of subjobs.

Queries all specified jobs that the user is authorized to query.

23.29.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

ID

Job ID, list of job IDs, queue, server, or null.

If *ID* is a null pointer or points to a null string, gets status of jobs at connected server.

Format for a job:

<sequence number>.<server name>

Format for a job array:

<sequence number>[].<server name>

Format for a subjob:

<sequence number>[<index>].<server name>

Format for a range of subjobs:

<sequence number>[<index start>-<index end>].<server name>

Format for a list of jobs: comma-separated list of job IDs in a single string. White space is ignored. No limit on length:

"<job ID>,<job ID>,<job ID>, ..."

Format for a queue:

<queue name>@<server name>

Format for a server:

<server name>

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in pbs_ifl.h as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string where you can specify limits or extensions of your search. Order of characters is not important.

23.29.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

next

Points to next attribute in list. A null pointer terminates the list.

23.29.4 Querying Job Arrays and Subjobs

You can query status of job arrays and their subjobs, or just the parent job arrays only.

To query status of job arrays and their subjobs, include the job array IDs in the ID argument, and include the '*t*' character in the extend argument. The function returns the status of each parent job array followed by status of each subjob in that job array.

To query status of one or more parent job arrays only, but not their subjobs, include their job IDs in the ID argument, but do not include anything in the extend argument.

23.29.5 Querying the Jobs at a Queue or Server

To query status of all jobs at a queue, give the queue name in the ID argument.

To query status of all jobs at a server, give the server name in the ID argument. If you give a null ID argument, the function queries the default server.

23.29.6 Extending Your Query

You can use the following characters in the extend parameter:

T, t

Extends query to include subjobs. Job arrays are not included.

x

Extends query to include finished and moved jobs.

23.29.6.1 Querying Finished and Moved Jobs

To get status for finished or moved jobs, as well as current jobs, add an '*x*' character to the extend parameter (set one character to be the '*x*' character). For example:

```
pbs_statjob ( ..., ..., <extend characters>) ...
```

Subjobs are not considered finished until the parent array job is finished.

23.29.7 Return Values

For a single job, if the job can be queried, returns a pointer to a batch_status structure containing the status of the specified job. If the job cannot be queried, returns a NULL pointer, and pbs_errno is set to an error number indicating the reason the job could not be queried.

For a list of jobs, if any of the specified jobs can be queried, returns a pointer to a batch_status structure containing the status of all the queryable jobs. If none of the jobs can be queried, returns a NULL pointer, and pbs_errno is set to the error number that indicates the reason that the last job in the list could not be queried.

For a queue, if the queue exists, returns a pointer to a batch_status structure containing the status of all the queryable jobs in the queue. If the queue does not exist, returns a NULL pointer, and pbs_errno is set to *PBSE_UNKQUE* (*15018*). If the queue exists but contains no queryable jobs, returns a NULL pointer, and pbs_errno is set to *PBSE_NONE* (*0*).

When querying a server, the connection to the server is already established by `pbs_connect()`. If there are jobs at the server, returns a pointer to a batch_status structure containing the status of all the queryable jobs at the server. If the server does not contain any queryable jobs, returns a NULL pointer, and pbs_errno is set to *PBSE_NONE* (*0*).

23.29.7.1 The batch_status Structure

The batch_status structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.29.8 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

23.29.9 See Also

qstat, pbs_connect

23.30 pbs_statnode

get status of PBS execution host(s)

23.30.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statnode(int connect, char *target, struct attrl *output_attribs, char *extend)*

23.30.2 Description

Issues a batch request to get the status of PBS execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host or all execution hosts. If an execution host has multiple vnodes, this command reports aggregated information from the vnodes for that host.

Identical to `pbs_stathost()`; retained for backward compatibility.

23.30.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of execution host whose attributes are to be reported. If this argument is null, returns attributes of all execution hosts known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend
: Character string for extensions to command. Not currently used.

23.30.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

23.30.4 Return Value

Returns a pointer to a list of batch_status structures describing the host(s).

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.30.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.30.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.30.6 See Also

qstat, pbs_connect

23.31 pbs_statque

get status of PBS queue(s)

23.31.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_statque(int connect, char *target, struct attrl *output_attribs, char *extend)*

23.31.2 Description

Issues a batch request to get the status of PBS queues.

Generates a *Status Queue* (20) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified queue or all queues.

23.31.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of queue whose attributes are to be reported. If this argument is null, returns attributes of all queues known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend
: Character string for extensions to command. Not currently used.

23.31.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next

Points to next attribute in list. A null pointer terminates the list.

23.31.4 Return Value

Returns a pointer to a list of batch_status structures describing the queue(s).

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.31.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.31.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.31.6 See Also

qstat, pbs_connect, pbs_statfree

23.32 pbs_statresv

get status of PBS reservation(s)

23.32.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statresv(int connect, char *target, struct attrl *output_attribs, char *extend)*

23.32.2 Description

Issues a batch request to get the status of PBS reservation(s).

Generates a *Status Reservation* (71) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified reservation or all reservations.

23.32.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target

ID of reservation whose attributes are to be reported. If this argument is a null pointer or points to a null string, returns attributes of all reservations the user is authorized to query.

Format for advance reservation:

R<sequence number>.<server name>

Format for standing reservation:

S<sequence number>.<server name>

Format for maintenance reservation:

M<sequence number>.<server name>

output_attribs

Pointer to a list of attributes to return. If this argument is null, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string for extensions to command. Not currently used.

23.32.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

23.32.4 Return Value

Returns a pointer to a list of batch_status structures describing the reservation(s).

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.32.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.32.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.32.6 See Also

qstat, pbs_connect, pbs_statfree

23.33 pbs_statrsc

get status of PBS resources

23.33.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statrsc(int connect, char *rescname, struct attrl *output_attribs, char *extend)*

23.33.2 Description

Issues a batch request to query and return the status of a specified resource, or a set of resources at a server.

Generates a *Status Resource* (82) batch request and sends it to the server over the connection specified by connect.

23.33.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

rescname

Name of resource to be queried. If this is null, queries all resources at the server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string for extensions to command. Not currently used.

23.33.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

23.33.4 Querying Resources at Server

Use the `pbs_connect()` command to get a connection handle at the server.

To query all resources at the server, pass a null pointer as the name of the resource.

23.33.5 Return Value

For a single resource, if the resource can be queried, returns a pointer to a batch_status structure containing the status of the specified resource.

If the resource cannot be queried, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

When querying a server, the connection to the server is already established by `pbs_connect()`. If there are resources at the server, returns a pointer to a batch_status structure describing the queryable resource(s) at the server.

In the unlikely event that the server does not contain any queryable resources because the user is unprivileged and all resources are marked as invisible (the i flag is set), returns a NULL pointer, and pbs_errno is set to *PBSE_NONE* (*0*).

23.33.5.1 The batch_status Structure

The batch_status structure is defined in `pbs_ifl.h` as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.33.6 Cleanup

You must free the list of batch_status structures when no longer needed, by calling `pbs_statfree()`.

23.33.7 See Also

qstat, pbs_connect

23.34 pbs_statsched

get status of PBS schedulers

23.34.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statsched(int connect, struct attrl *output_attribs, char *extend)*

23.34.2 Description

Issues a batch request to get the status of the PBS schedulers.

Generates a *Status Scheduler (81)* batch request and sends it to the server over the connection specified by connect.

This command returns status of the default scheduler and all multischeds.

23.34.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string for extensions to command. Not currently used.

23.34.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

23.34.4 Return Value

Returns a pointer to a list of batch_status structures describing the default scheduler and all multischeds.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.34.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.34.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.34.6 See Also

qstat, pbs_connect

23.35 pbs_statserver

get status of a PBS batch server

23.35.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statserver(int connect, struct attrl *output_attribs, char *extend)*

23.35.2 Description

Issues a batch request to get the status of a batch server.

Generates a *Status Server* (21) batch request and sends it to the server over the connection specified by connect.

23.35.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

output_attribs

Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string for extensions to command. Not currently used.

23.35.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource. Should always be a pointer to a null string.

next

Points to next attribute in list. A null pointer terminates the list.

23.35.4 Return Value

Returns a pointer to a batch_status structure describing the server.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.35.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

23.35.5 Cleanup

You must free the batch_status structure when no longer needed, by calling pbs_statfree().

23.35.6 See Also

qstat, pbs_connect, pbs_statfree

23.36 pbs_statvnode

get status of PBS vnode(s) on execution hosts

23.36.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*void pbs_statfree(struct batch_status *psj)*

*struct batch_status *pbs_statvnode(int connect, char *target, struct attrl *output_attribs, char *extend)*

23.36.2 Description

Issues a batch request to get the status of PBS vnodes on execution hosts.

Generates a *Status Node* (58) batch request and sends it to the server over the connection specified by connect.

Returns specified attributes or all attributes of specified execution host vnode or all execution host vnodes.

23.36.3 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

target
: Name of execution host vnode whose attributes are to be reported. If this argument is null, returns attributes of all execution host vnodes known to the server.

output_attribs
: Pointer to a list of attributes to return. If this argument is a null pointer or points to a null string, returns all attributes. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend
: Character string for extensions to command. Not currently used.

23.36.3.1 Members of attrl Structure

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

next
: Points to next attribute in list. A null pointer terminates the list.

23.36.4 Return Value

Returns a pointer to a list of batch_status structures describing the vnode(s).

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.36.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char                *name;
    struct attrl        *attribs;
    char                *text;
}
```

23.36.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

23.36.6 See Also

qstat, pbs_connect, pbs_statfree

23.37 pbs_submit

submit a PBS batch job

23.37.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_submit(int connect, struct attropl *attrib_list, char *jobscript, char *destqueue, char *extend)*

23.37.2 Description

Issues a batch request to submit a new batch job.

Generates a *Queue Job* (1) batch request and sends it to the server over the connection specified by connect.

Submits job to specified queue at connected server, or if no queue is specified, to default queue at connected server.

23.37.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

attrib_list

Pointer to a list of attributes explicitly requested for job. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the job; the action depends on the attribute.

jobscript

Pointer to path to job script. Can be absolute or relative. Relative path begins with the directory where the user submits the job.

If null pointer or pointer to null string, no script is passed with job.

destqueue

Pointer to name of destination queue at connected server. If this is a null pointer or points to a null string, the job is submitted to the default queue at the connected server.

extend

Character string for extensions to command. Not currently used.

23.37.3.1 Members of attropl Structure

next
: Points to next attribute in list. A null pointer terminates the list.

name
: Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Points to a string containing the value of the attribute or resource.

op
: Operation to perform on the attribute or resource. In this command, the only allowed operator is SET.

23.37.4 Return Value

Returns a pointer to a character string containing the job ID assigned by the server.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.37.5 Cleanup

The space for the job ID returned by `pbs_submit()` is allocated by `pbs_submit()`. Free it via a call to `free()` when you no longer need it.

23.37.6 See Also

qsub, pbs_connect

23.38 pbs_submit_resv

submit a PBS reservation

23.38.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*char *pbs_submit_resv(int connect, struct attropl *attrib_list, char *extend)*

23.38.2 Description

Issues a batch request to submit a new reservation.

Generates a *Submit Reservation* (70) batch request and sends it to the server over the connection specified by connect.

Returns a pointer to the reservation ID.

23.38.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection over which to send batch request to server.

attrib_list

Pointer to a list of attributes to set, with values. Each attribute is described in an attropl structure, defined in `pbs_ifl.h` as:

```
struct attropl {
        struct attropl *next;
        char *name;
        char *resource;
        char *value;
        enum batch_op op;
};
```

For any attribute that is not specified or that is a null pointer, PBS takes the default action for that attribute. The default action is to assign the default value or to not pass the attribute with the reservation; the action depends on the attribute.

extend

Character string for extensions to command. Not currently used.

23.38.3.1 Members of attropl Structure

next

Points to next attribute in list. A null pointer terminates the list.

name

Points to a string containing the name of the attribute.

resource

Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value

Points to a string containing the value of the attribute or resource.

op

Operator. The only allowed operator for this function is SET.

23.38.4 Return Value

Returns a pointer to a character string containing the reservation ID assigned by the server.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

23.38.5 Cleanup

The space for the reservation ID returned by `pbs_submit_resv()` is allocated by `pbs_submit_resv()`. Free it via a call to `free()` when you no longer need it.

23.38.6 See Also

pbs_rsub, pbs_connect

23.39 pbs_terminate

shut down a PBS batch server

23.39.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*int pbs_terminate(int connect, int manner, char *extend)*

23.39.2 Description

Issues a batch request to shut down a batch server.

Generates a *Server Shutdown* (17) batch request and sends it to the server over the connection specified by connect.

The `pbs_terminate()` command exits after the server has completed its shutdown procedure.

23.39.3 Required Privilege

You must have Operator or Manager privilege to run this command.

23.39.4 Arguments

connect
: Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

manner
: Manner in which to shut down server. The available manners are defined in `pbs_ifl.h`. Valid values: *SHUT_IMMEDIATE, SHUT_DELAY, SHUT_QUICK*. See "qterm" on page 233 of the PBS Professional Reference Guide for information on manner in which to shut down server.

extend
: Character string for extensions to command. Not currently used.

23.39.5 Return Value

The routine returns *0* (*zero*) on success.

If an error occurred, the routine returns a non-zero exit value, and the error number is available in the global integer pbs_errno.

23.39.6 See Also

qterm, pbs_connect

24
TM Library

This chapter describes the PBS Task Management library. The TM library is a set of routines used to manage multi-process, parallel, and distributed applications.

24.1 TM Library Routines

The following manual pages document the application programming interface provided by the TM library.

24.2 tm_init, tm_nodeinfo, tm_poll, tm_notify, tm_spawn, tm_kill, tm_obit, tm_taskinfo, tm_atnode, tm_rescinfo, tm_publish, tm_subscribe, tm_finalize, tm_attach

task management API

24.2.1 Synopsis

```
#include <tm.h>
int tm_init(info, roots)
    void *info;
    struct tm_roots *roots;
    int tm_nodeinfo(list, nnodes)
tm_node_id **list;
    int *nnodes;
int tm_poll(poll_event, result_event, wait, tm_errno)
    tm_event_t poll_event;
    tm_event_t *result_event;
    int wait;
    int *tm_errno;
int tm_notify(tm_signal)
    int tm_signal;
int tm_spawn(argc, argv, envp, where, tid, event)
    int argc;
    char **argv;
    char **envp;
    tm_node_id where;
    tm_task_id *tid;
    tm_event_t *event;
int tm_kill(tid, sig, event)
    tm_task_id tid;
    int sig;
    tm_event_t *event;
int tm_obit(tid, obitval, event)
    tm_task_id tid;
    int *obitval;
    tm_event_t *event;
int tm_taskinfo(node, tid_list, list_size, ntasks, event)
    tm_node_id node;
    tm_task_id *tid_list;
    int list_size;
    int *ntasks;
    tm_event_t *event;
int tm_atnode(tid, node)
    tm_task_id tid;
    tm_node_id *node;
int tm_rescinfo(node, resource, len, event)
    tm_node_id node;
    char *resource;
    int len;
    tm_event_t *event;
int tm_publish(name, info, len, event)
```

```
        char *name;
        void *info;
        int len;
        tm_event_t *event;
    int tm_subscribe(tid, name, info, len, info_len, event)
        tm_task_id tid;
        char *name;
        void *info;
        int len;
        int *info_len;
        tm_event_t *event;
    int tm_attach(jobid, cookie, pid, tid, host, port)
        char *jobid;
        char *cookie;
        pid_t pid;
        tm_task_id *tid;
        char *host;
        int port;
    int tm_finalize()
```

24.2.2 Description

These functions provide a partial implementation of the task management interface part of the PSCHED API. In PBS, MoM provides the task manager functions. This library opens a tcp socket to the MoM running on the local host and sends and receives messages using the DIS protocol (described in the PBS IDS). The tm interface can only be used by a process within a PBS job.

The PSCHED Task Management API description used to create this library was committed to paper on November 15, 1996 and was given the version number 0.1. Changes may have taken place since that time which are not reflected in this library.

The API description uses several data types that it purposefully does not define. This was done so an implementation would not be confined in the way it was written. For this specific work, the definitions follow:

typedef int tm_node_id; /* job-relative node id */

#define TM_ERROR_NODE ((tm_node_id)-1)

typedef int tm_event_t; /* > 0 for real events */

#define TM_NULL_EVENT ((tm_event_t)0)

#define TM_ERROR_EVENT ((tm_event_t)-1)

typedef unsigned long tm_task_id;

#define TM_NULL_TASK (tm_task_id)0

There are a number of error values defined as well: TM_SUCCESS, TM_ESYSTEM, TM_ENOEVENT, TM_ENOTCONNECTED, TM_EUNKNOWNCMD, TM_ENOTIMPLEMENTED, TM_EBADENVIRONMENT, TM_ENOTFOUND.

tm_init() initializes the library by opening a socket to the MoM on the local host and sending a TM_INIT message, then waiting for the reply. The info parameter has no use and is included to conform with the PSCHED document. The roots pointer will contain valid data after the function returns and has the following structure:

struct tm_roots {

tm_task_id tm_me;

tm_task_id tm_parent;

int tm_nnodes;

int tm_ntasks;

int tm_taskpoolid;

tm_task_id *tm_tasklist;

};

tm_me The task id of this calling task.

tm_parent The task id of the task which spawned this task or TM_NULL_TASK if the calling task is the initial

task started by PBS.

tm_nnodes The number of nodes allocated to the job.

tm_ntasks This will always be 0 for PBS.

tm_taskpoolid PBS does not support task pools so this will always be -1.

tm_tasklist This will be NULL for PBS.

The tm_ntasks, tm_taskpoolid and tm_tasklist fields are not filled with data specified by the PSCHED document. PBS does not support task pools and, at this time, does not return information about current running tasks from tm_init. There is a separate call to get information for current running tasks called tm_taskinfo which is described below. The return value from tm_init is TM_SUCCESS if the library initialization was successful, or an error is returned otherwise.

tm_nodeinfo() places a pointer to a malloc'ed array of tm_node_id's in the pointer pointed at by list. The order of the tm_node_id's in list is the same as that specified to MoM in the "exec_host" attribute. The int pointed to by nnodes contains the number of nodes allocated to the job. This is information that is returned during initialization and does not require communication with MoM. If tm_init has not been called, TM_ESYSTEM is returned, otherwise TM_SUCCESS is returned.

tm_poll() is the function which will retrieve information about the task management system to locations specified when other routines request an action take place. The bookkeeping for this is done by generating an event for each action. When the task manager (MoM) sends a message that an action is complete, the event is reported by tm_poll and information is placed where the caller requested it. The argument poll_event is meant to be used to request a specific event. This implementation does not use it and it must be set to TM_NULL_EVENT or an error is returned. Upon return, the argument result_event will contain a valid event number or TM_ERROR_EVENT on error. If wait is zero and there are no events to report, result_event is set to TM_NULL_EVENT. If wait is non-zero an there are no events to report, the function will block waiting for an event. If no local error takes place, TM_SUCCESS is returned. If an error is reported by MoM for an event, then the argument tm_errno will be set to an error code.

tm_notify() is described in the PSCHED documentation, but is not implemented for PBS yet. It will return TM_ENOTIMPLEMENTED.

tm_spawn() sends a message to MoM to start a new task. The node id of the host to run the task is given by where. The parameters argc, argv and envp specify the program to run and its arguments and environment very much like exec(). The full path of the program executable must be given by argv[0] and the number of elements in the argv array is given by argc. The array envp is NULL terminated. The argument event points to a tm_event_t variable which is filled in with an event number. When this event is returned by tm_poll, the tm_task_id pointed to by tid will contain the task id of the newly created task.

tm_kill() sends a signal specified by sig to the task tid and puts an event number in the tm_event_t pointed to by event.

tm_obit() creates an event which will be reported when the task tid exits. The int pointed to by obitval will contain the exit value of the task when the event is reported.

tm_taskinfo() returns the list of tasks running on the node specified by node. The PSCHED documentation mentions a special ability to retrieve all tasks running in the job. This is not supported by PBS. The argument tid_list points to an array of tm_task_id's which contains list_size elements. Upon return, event will contain an event number. When this event is polled, the int pointed to by ntasks will contain the number of tasks running on the node and the array will be filled in with tm_task_id's. If ntasks is greater than list_size, only list_size tasks will be returned.

tm_atnode() will place the node id where the task tid exists in the tm_node_id pointed to by node.

tm_rescinfo() makes a request for a string specifying the resources available on a node given by the argument node. The string is returned in the buffer pointed to by resource and is terminated by a NUL character unless the number of characters of information is greater than specified by len. The resource string PBS returns is formatted as follows:

A space separated set of strings from the uname system call. The order of the strings is sysname, nodename, release, version, machine.

A comma separated set of strings giving the components of the "Resource_List" attribute of the job, preceded by a colon (:). Each component has the resource name, an equal sign, and the limit value.

tm_publish() causes len bytes of information pointed at by info to be sent to the local MoM to be saved under the name given by name.

tm_subscribe() returns a copy of the information named by name for the task given by tid. The argument info points to a buffer of size len where the information will be returned. The argument info_len will be set with the size of the published data. If this is larger than the supplied buffer, the data will have been truncated.

tm_attach() commands MoM to create a new PBS "attached task" out of a session running on MoM's host. The jobid parameter specifies the job which is to have a new task attached. If it is NULL, the system will try to determine the correct jobid. The cookie parameter must be NULL. The pid parameter must be a non-zero process id for the process which is to be added to the job specified by jobid. If tid is non-NULL, it will be used to store the task id of the new task. The host and port parameters specify where to contact MoM. host should be NULL. The return value will be 0 if a new task has been successfully created and non-zero on error. The return value will be one of the TM error numbers defined in tm.h as follows:

TM_ESYSTEM MoM cannot be contacted

TM_ENOTFOUND No matching job was found

TM_ENOTIMPLEMENTED The call is not implemented/supported

TM_ESESSION The session specified is already attached

TM_EUSER The calling user is not permitted to attach

TM_EOWNER The process owner does not match the job

TM_ENOPROC The process does not exist

tm_finalize() may be called to free any memory in use by the library and close the connection to MoM.

24.2.3 See Also

pbs_mom(8B), pbs_sched(8B)

25
RM Library

This chapter describes the PBS Resource Monitor library. The RM library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

25.1 RM Library Routines

The following manual pages document the application programming interface provided by the RM library.

25.2 openrm, closerm, downrm, configrm, addreq, allreq, getreq, flushreq, activereq, fullresp

resource monitor API

25.2.1 Synopsis

```
#include <sys/types.h>
#include <netinet/in.h>
#include <rm.h>
int openrm (host, port)
    char *host;
    unsigned int port;
int closerm (stream)
    int stream;
int downrm (stream)
    int stream;
int configrm (stream, file)
    int stream;
    char *file;
int addreq (stream, line)
    int stream;
    char *line;
int allreq (line)
    char *line;
char *getreq(stream)
    int stream;
int flushreq()
int activereq()
void fullresp(flag)
    int flag;
```

25.2.2 Description

The resource monitor library contains functions to facilitate communication with the PBS Professional resource monitor. It is set up to make it easy to connect to several resource monitors and handle the network communication efficiently.

In all these routines, the variable pbs_errno will be set when an error is indicated. The lower levels of network protocol are handled by the "Data Is Strings" DIS library and the TPP library.

configrm() causes the resource monitor to read the file named. Deprecated.

addreq() begins a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

allreq() begins, for each stream, a new message to the resource monitor if necessary. Then adds a line to the body of an outstanding command to the resource monitor.

getreq() finishes and sends any outstanding message to the resource monitor. If fullresp() has been called to turn off "full response" mode, the routine searches down the line to find the equal sign just before the response value. The returned string (if it is not NULL) has been allocated by malloc and thus free must be called when it is no longer needed to prevent memory leaks.

flushreq() finishes and sends any outstanding messages to all resource monitors. For each active resource monitor structure, it checks if any outstanding data is waiting to be sent. If there is, it is sent and the internal structure is marked to show "waiting for response".

fullresp() turns on, if flag is true, "full response" mode where getreq() returns a pointer to the beginning of a line of response. This is the default. If flag is false, the line returned by getreq() is just the answer following the equal sign.

activereq() Returns the stream number of the next stream with something to read or a negative number (the return from tpp_poll) if there is no stream to read.

In order to use any of the above with Windows, initialize the network library and link with winsock2. To do this, call winsock_init() before calling the function and link against the ws2_32.lib library.

25.2.3 See Also

tcp(4P), udp(4P)

26
TCL/tk Interface

As of version 19.4.1, the PBS TCL API is **deprecated**.

The PBS Professional software includes a TCL/tk interface to PBS. Wrapped versions of many of the API calls are compiled into a special version of the TCL shell, called `pbs_tclsh`. (A special version of the tk window shell is also provided, called `pbs_wish`.). This chapter documents the TCL/tk interface to PBS.

The `pbs_tclapi` is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the `pbs_server` about the state of PBS, jobs, queues, and nodes, and communicate with `pbs_mom` to get information about the status of running jobs, available resources on nodes, etc.

26.1 TCL/tk API Functions

A set of functions to communicate with the PBS server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable pbs_errno to a value to indicate if an error occurred. In all cases, the value "*0*" means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied errno variable. If the function call communicates with the PBS server, any error value will come from the error number returned by the server. This is the same TCL interface used by the `pbs_tclsh` and `pbs_wish` commands.

Note that the `pbs_tclapi pbsrescquery` command, which calls the C API `pbs_rescquery`, is **obsolete**. Any attempt to use it will result in a PBSE_NOSUPPORT error being returned.

26.2 pbs_tclapi

PBS TCL Application Programming Interface

26.2.1 Description

The pbs_tclapi is a subset of the PBS external API wrapped in a TCL library. This functionality allows the creation of scripts that query the PBS system. Specifically, it permits the user to query the pbs_server about the state of PBS, jobs, queues, and nodes, and communicate with pbs_mom to get information about the status of running jobs, available resources on nodes, etc.

26.2.2 Usage

A set of functions to communicate with the PBS server and resource monitor have been added to those normally available with Tcl. All these calls will set the Tcl variable "pbs_errno" to a value to indicate if an error occurred. In all cases, the value "0" means no error. If a call to a Resource Monitor function is made, any error value will come from the system supplied errno variable. If the function call communicates with the PBS Server, any error value will come from the error number returned by the server. This is the same TCL interface used by the pbs_tclsh and pbs_wish commands.

openrm host ?port?

Creates a connection to the PBS Resource Monitor on host using port as the port number or the standard port for the resource monitor if it is not given. A connection handle is returned. If the open is successful, this will be a non-negative integer. If not, an error occurred.

closerm connection

The parameter connection is a handle to a resource monitor which was previously returned from openrm. This connection is closed.

Nothing is returned.

downrm connection

Sends a command to the connected resource monitor to shutdown.

Nothing is returned.

configrm connection filename

Sends a command to the connected resource monitor to read the configuration file given by filename. If this is successful, a "0" is returned, otherwise, "-1" is returned.

addreq connection request

A resource request is sent to the connected resource monitor. If this is successful, a "0" is returned, otherwise, "-1" is returned.

getreq connection

One resource request response from the connected resource monitor is returned. If an error occurred or there are no more responses, an empty string is returned.

allreq request

A resource request is sent to all connected resource monitors. The number of streams acted upon is returned.

flushreq

All resource requests previously sent to all connected resource monitors are flushed out to the network. Nothing is returned.

activereq

The connection number of the next stream with something to read is returned. If there is nothing to read from any of the connections, a negative number is returned.

fullresp flag

Evaluates flag as a boolean value and sets the response mode used by getreq to full if flag evaluates to "true". The full return from a resource monitor includes the original request followed by an equal sign followed by the response. The default situation is only to return the response following the equal sign. If a script needs to "see" the entire line, this function may be used.

pbsstatserv

The server is sent a status request for information about the server itself. If the request succeeds, a list with three elements is returned, otherwise an empty string is returned. The first element is the server's name. The second is a list of attributes. The third is the "text" associated with the server (usually blank).

pbsstatjob

The server is sent a status request for information about the all jobs resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each job. Each element is a list with three elements. The first is the job's jobid. The second is a list of attributes. The attribute names which specify resources will have a name of the form "Resource_List:name" where "name" is the resource name. The third is the "text" associated with the job (usually blank).

pbsstatque

The server is sent a status request for information about all queues resident within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each queue. Each element is a list with three elements. This first is the queue's name. The second is a list of attributes similar to pbsstatjob. The third is the "text" associated with the queue (usually blank).

pbsstatnode

The server is sent a status request for information about all nodes defined within the server. If the request succeeds, a list is returned, otherwise an empty string is returned. The list contains an entry for each node. Each element is a list with three elements. This first is the node's name. The second is a list of attributes similar to pbsstatjob. The third is the "text" associated with the node (usually blank).

pbsselstat

The server is sent a status request for information about the all runnable jobs resident within the server. If the request succeeds, a list similar to pbsstatjob is returned, otherwise an empty string is returned.

pbsrunjob jobid ?location?

Run the job given by jobid at the location given by location. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsasyrunjob jobid ?location?

Run the job given by jobid at the location given by location without waiting for a positive response that the job has actually started. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsrerunjob jobid

Re-runs the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsdeljob jobid

Delete the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsholdjob jobid

Place a hold on the job given by jobid. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsmovejob jobid ?location?

Move the job given by jobid to the location given by location. If location is not given, the default location is used. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsqenable queue

Set the "enabled" attribute for the queue given by queue to true. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsqdisable queue

Set the "enabled" attribute for the queue given by queue to false. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsqstart queue

Set the "started" attribute for the queue given by queue to true. If this is successful, a "0" is returned, otherwise, "-1" is

returned.

pbsqstop queue

Set the "started" attribute for the queue given by queue to false. If this is successful, a "0" is returned, otherwise, "-1" is returned.

pbsalterjob jobid attribute_list

Alter the attributes for a job specified by jobid. The parameter attribute_list is the list of attributes to be altered. There can be more than one. Each attribute consists of a list of three elements. The first is the name, the second the resource and the third is the new value. If the alter is successful, a "0" is returned, otherwise, "-1" is returned.

pbsrescquery resource_list

Deprecated. Obtain information about the resources specified by resource_list. This will be a list of strings. If the request succeeds, a list with the same number of elements as resource_list is returned. Each element in this list will be a list with four numbers. The numbers specify available, allocated, reserved, and down in that order.

pbsconnect ?server?

Make a connection to the named server or the default server if a parameter is not given. Only one connection to a server is allowed at any one time.

pbsdisconnect

Disconnect from the currently connected server.

The above Tcl functions use PBS interface library calls for communication with the server and the PBS resource monitor library to communicate with pbs_mom.

datetime ?day? ?time?

The number of arguments used determine the type of date to be calculated. With no arguments, the current POSIX date is returned. This is an integer in seconds.

With one argument there are two possible formats. The first is a 12 (or more) character string specifying a complete date in the following format:

YYMMDDhhmmss

All characters must be digits. The year (YY) is given by the first two (or more) characters and is the number of years since 1900. The month (MM) is the number of the month [01-12]. The day (DD) is the day of the month [01-32]. The hour (hh) is the hour of the day [00-23]. The minute (mm) is minutes after the hour [00-59]. The second (ss) is seconds after the minute [00-59]. The POSIX date for the given date/time is returned.

The second option with one argument is a relative time. The format for this is

HH:MM:SS

With hours (HH), minutes (MM) and seconds (SS) being separated by colons ":". The number returned in this case will be the number of seconds in the interval specified, not an absolute POSIX date.

With two arguments a relative date is calculated. The first argument specifies a day of the week and must be one of the following strings: "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", or "Sat". The second argument is a relative time as given above. The POSIX date calculated will be the day of the week given which follows the current day, and the time given in the second argument. For example, if the current day was Monday, and the two arguments were "Fri" and "04:30:00", the date calculated would be the POSIX date for the Friday following the current Monday, at four-thirty in the morning. If the day specified and the current day are the same, the current day is used, not the day one week later.

strftime format time

This function calls the POSIX function strftime(). It requires two arguments. The first is a format string. The format conventions are the same as those for the POSIX function strftime(). The second argument is POSIX calendar time in second as returned by datetime. It returns a string based on the format given. This gives the ability to extract information about a time, or format it for printing.

logmsg tag message

This function calls the internal PBS function log_err(). It will cause a log message to be written to the scheduler's log file. The tag specifies a function name or other word used to identify the area where the message is generated. The message is the string to be logged.

26.2.3 See Also

pbs_tclsh(8B), pbs_wish(8B), pbs_mom(8B), pbs_server(8B), pbs_sched(8B)

27 Hooks

This chapter describes the PBS hook APIs. For more information on hooks, see the PBS Professional Administrator's Guide.

27.1 Introduction

A hook is a block of Python code that is triggered in response to queueing a job, modifying a job, moving a job, running a job, submitting a PBS reservation, MoM receiving a job, MoM starting a job, MoM killing a job, a job finishing, and MoM cleaning up a job. Each hook can *accept* (allow) or *reject* (prevent) the action that triggers it. The hook can modify the input parameters given for the action. The hook can also make calls to functions external to PBS. PBS provides an interface for use by hooks. This interface allows hooks to read and/or modify things such as job and server attributes, the server, queues, and the event that triggered the hook.

The Administrator creates any desired hooks.

This chapter contains the following man pages:

- `pbs_module(7B)`
- `pbs_stathook(3B)`

See the following additional man pages:

- `qmgr(1B)`
- `qsub(1B)`
- `qmove(1B)`
- `qalter(1B)`
- `pbs_rsub(1B)`
- `pbs_manager(3B)`

27.2 How Hooks Work

27.2.1 Hook Contents and Permissions

A hook contains a Python script. The script is evaluated by a Python 3 or later interpreter, embedded in PBS.

Hooks have a default Linux umask of 022. File permissions are inherited from the current working directory of the hook script.

27.2.2 Accepting and Rejecting Actions

The hook script always accepts the current event request action unless an unhandled exception occurs in the script, a hook alarm timeout is triggered or there's an explicit call to "pbs.event().reject()".

27.2.3 Exceptions

A hook script can catch an exception and evaluate whether or not to accept or reject the event action. In this example, while referencing the non-existent attribute pbs.event().job.interactive, an exception is triggered, but the event action is still accepted:

```
…
try:
    e = pbs.event()
    if e.job.interactive:
         e.reject("Interactive jobs not allowed")
except SystemExit:
    pass
except:
    e.accept()
```

27.2.4 Unsupported Interfaces and Uses

Site hooks which read, write, close, or alter `stdin`, `stdout`, or `stderr`, are not supported. Hooks which use any interfaces other than those described are unsupported.

27.3 Interface to Hooks

Two PBS APIs are used with hooks. These are `pbs_manager()` and `pbs_stathook()`. The pbs module provides a Python interface to PBS.

27.3.1 The pbs Module

Hooks have access to a special module called "pbs", which contains functions that perform PBS-related actions. This module must be explicitly loaded by the hook writer via the call "import pbs".

The *pbs module* provides an interface to PBS and the hook environment. The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code.

27.3.1.1 Description of pbs Module

27.4 pbs_module

The interface is made up of Python objects, which have attributes and methods. You can operate on these objects using Python code. For a description of each object, see the PBS Professional Administrator's Guide.

27.4.0.1 pbs Module Objects

See "The pbs Module" on page 76 in the PBS Professional Hooks Guide.

27.4.0.2 pbs Module Global Attribute Creation Methods

See "PBS Types and Their Methods" on page 143 in the PBS Professional Hooks Guide.

27.4.0.3 Attributes and Resources

See "Using Attributes and Resources in Hooks" on page 44 in the PBS Professional Hooks Guide.

27.4.0.4 Exceptions

See "Table of Exceptions" on page 43 in the PBS Professional Hooks Guide and "Hook Alarm Calls and Unhandled Exceptions" on page 43 in the PBS Professional Hooks Guide.

27.4.0.5 See Also

The *PBS Professional Administrator's Guide*, `pbs_hook_attributes(7B)`, `pbs_resources(7B)`, `qmgr(1B)`

27.4.1 The pbs_manager() API

The `pbs_manager()` API is described in "pbs_manager" on page 41.

The `pbs_manager()` API contains the following:

- An obj_name called "hook" defined as MGR_OBJ_HOOK, for use with non-built-in hooks
- An obj_name called "pbshook" defined as MGR_OBJ_PBS_HOOK, for use with built-in hooks.
- The following hook commands, which operate only on hook objects:

  MGR_CMD_IMPORT

  This command is used for loading the hook script contents into a hook.

  MGR_CMD_EXPORT

  This command is used for dumping to a file the contents of a hook script.

  The parameters to MGR_CMD_IMPORT and MGR_CMD_EXPORT are specified via the attrib parameter of `pbs_manager()`.

  For MGR_CMD_IMPORT, specify attropl "name" as "content-type", "content-encoding", and "input-file" along with the corresponding "value" and an "op" of SET.

  For MGR_CMD_EXPORT, specify attropl "name" as "content-type", "content-encoding", and "output-file" along with the corresponding "value" and an "op" of SET.

  Functions MGR_CMD_IMPORT, MGR_CMD_EXPORT, and MGR_OBJ_HOOK are used only with hooks, and therefore require root privilege on the server host.

  When obj_name is MGR_OBJ_PBS_HOOK, the only allowed options for command are MGR_CMD_SET, MGR_CMD_UNSET, MGR_CMD_IMPORT, and MGR_CMD_EXPORT.

If MGR_CMD_IMPORT or MGR_CMD_EXPORT is specified when obj_name is MGR_OBJ_PBS_HOOK, the attropl content-type must be "application/x-config".

27.4.1.1 Troubleshooting

You can use `pbs_geterrmsg()` to determine the last error message received from the `pbs_manager()` call. For instance, with a MGR_OBJ_PBS_HOOK where command is either MGR_CMD_IMPORT or MGR_CMD_EXPORT, but attropl 'content-type' is not "application/x-config", `pbs_geterrmsg()` returns:

```
"<content-type> must be application/x-config"
```

If an unrecognized hook configuration file suffix is given, whether for MGR_OBJ_HOOK or MGR_OBJ_PBS_HOOK, `pbs_geterrmsg()` returns:

```
"<input-file> contains an invalid suffix, should be one of: .json .py .txt .xml .ini"
```

If the hook configuration file failed to be precompiled by PBS, `pbs_geterrmsg()` shows:

```
"Failed to validate config file, hook '<hook_name>' config file not overwritten"
```

27.4.1.2 Privilege for Hooks

To run, hooks require root privilege on Linux, and local Administrators privilege on Windows.

27.4.1.3 Examples of Using pbs_manager()

Example 27-1: The following:

```
# qmgr -c 'import hook hook1 application/x-python base64 hello.py.b64'
```

is programmatically equivalent to:

```
static struct attropl imp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-python",
        SET,
        (struct attropl *)&imp_attribs[1]
    },
    {   "content-encoding",
        (char *)0,
        "base64",
        SET,
        (struct attropl *)&imp_attribs[2]},
    {   "input-file",
        (char *)0,
        "hello.py.b64",
        SET,
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_IMPORT, MGR_OBJ_HOOK, "hook1", &imp_attribs[0], NULL);
```

Example 27-2: The following:

```
# qmgr -c 'export hook hook1 application/x-python default hello.py'
```

is programmatically equivalent to:

```
static struct attropl exp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-python",
        SET,
        (struct attropl *)&exp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&exp_attribs[2]},
    {   "output-file",
        (char *)0,
        "hello.py",
        SET,
        (struct attropl *)0
```

```
    }
};

pbs_manager(con, MGR_CMD_EXPORT, MGR_OBJ_HOOK, "hook1", &exp_attribs[0], NULL);
```

Example 27-3: The following:

```
# qmgr -c 'import pbshook hook1 application/x-config default hello.json'
```

is programmatically equivalent to:

```
static struct attropl imp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-config",
        SET,
        (struct attropl *)&imp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&imp_attribs[2]},
    {   "input-file",
        (char *)0,
        "hello.json",
    <   SET,
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_IMPORT, MGR_OBJ_PBS_HOOK, "hook1", &imp_attribs[0], NULL);
```

Example 27-4: The following:

```
# qmgr -c 'export pbshook hook1 application/x-config default hello.json'
```

is programmatically equivalent to:

```
static struct attropl exp_attribs[] = {
    {   "content-type",
        (char *)0,
        "application/x-config",
        SET,
        (struct attropl *)&exp_attribs[1]},
    {   "content-encoding",
        (char *)0,
        "default",
        SET,
        (struct attropl *)&exp_attribs[2]},
    {   "output-file",
        (char *)0,
        "hello.json",
        SET,
        (struct attropl *)0
    }
};

pbs_manager(con, MGR_CMD_EXPORT,
MGR_OBJ_PBS_HOOK, "hook1", &exp_attribs[0], NULL);
```

27.4.2 The pbs_stathook() API

The PBS API called "pbs_stathook()" is used to get attributes and values for site hooks and built-in hooks.

The prototype for pbs_stathook() is as follows:

```
struct batch_status *pbs_stathook(int connect, char *hook_name, struct attrl *attrib, char
    *extend)
```

To query status for site hooks:

The call to pbs_stathook() causes a PBS_BATCH_StatusHook request to be sent to the server. In reply, the PBS server returns a batch reply status of object type MGR_OBJ_HOOK listing the attributes and values that were requested relating to a particular hook or all hooks of type HOOK_SITE. Leave the extend value blank.

To query status for built-in hooks:

Pass PBS_HOOK as the extend value. The server returns a batch reply status of object type MGR_OBJ_PBS_HOOK.

27.4.2.1 Example of Using pbs_stathook()

To list all site hooks using `qmgr`:

```
qmgr -c "list hook"
```

To list all site hooks using the `pbs_stathook()` API:

```
pbs_stathook()
```

The result is the same. For example, if there are two site hooks, c3 and c36:

```
Hook c3
    type = site
    enabled = true
    event = queuejob, modifyjob
    user = pbsadmin
    alarm = 30
    order = 1
Hook c36
    type = site
    enabled = true
    event = resvsub
    user = pbsadmin
    alarm = 30
    order = 1
```

27.5 pbs_stathook(3B)

get status information about PBS site hooks

27.5.1 Synopsis

#include <pbs_error.h>

#include <pbs_ifl.h>

*struct batch_status *pbs_stathook(int connect, char *hook_id, struct attrl *output_attribs, char *extend)*

*void pbs_statfree(struct batch_status *psj)*

27.5.2 Description

Issues a batch request to get the status of a specified site hook or a set of site hooks at the current server.

Generates a *Status Hook* batch request and sends it to the server over the connection specified by connect.

27.5.2.1 Required Privilege

This API can be executed only by root on the local server host.

27.5.3 Arguments

connect

Return value of `pbs_connect()`. Specifies connection handle over which to send batch request to server.

hook_id

Hook name, null string, or null pointer.

If *hook_id* specifies a name, the attribute-value list for that hook is returned.

If *hook_id* is a null string or a null pointer, the status of all hooks at the current server is returned.

output_attribs

Pointer to a list of attributes to return. If this list is null, all attributes are returned. Each attribute is described in an attrl structure, defined in `pbs_ifl.h` as:

```
struct attrl {
        char *name;
        char *resource;
        char *value;
        struct attrl *next;
};
```

extend

Character string where you can specify limits or extensions of your selection.

27.5.3.1 Members of attrl Structure

name

Points to a string containing the name of the attribute.

resource
: Points to a string containing the name of a resource. Used only when the specified attribute contains resource information. Otherwise, resource should be a null pointer.

value
: Should always be a pointer to a null string.

next
: Points to next attribute in list. A null pointer terminates the list.

27.5.4 Return Value

Returns a pointer to a list of batch_status structures for the specified site hook. If no site hook can be queried for status, returns the null pointer.

If an error occurred, the routine returns a null pointer, and the error number is available in the global integer pbs_errno.

27.5.4.1 The batch_status Structure

The batch_status structure is defined in pbs_ifl.h as

```
struct batch_status {
    struct batch_status *next;
    char *name;
    struct attrl *attribs;
    char *text;
}
```

27.5.5 Cleanup

You must free the list of batch_status structures when no longer needed, by calling pbs_statfree().

27.5.6 Error Messages

The following error message is returned by the pbs_geterrmsg() API after calling pbs_manager() operating on a hook object, with the MGR_CMD_IMPORT command, with "content-type" of "application/x-config":

```
"Failed to validate config file, hook 'submit' config file not overwritten"
```

If the input config file given is of unrecognized suffix, then the following message is returned by the pbs_geterrmsg() API after calling pbs_manager() operating on a hook object, MGR_CMD_IMPORT command with "content-type" of "application/x-config":

```
"<input-file> contains an invalid suffix, should be one of: .json .py .txt .xml .ini"
```

If you specify an unknown hook event, pbs_geterrmsg() returns the following after calling pbs_manager():

```
invalid argument (<bad_event>) to event.  Should be one or more of:
    queuejob,modifyjob,resvsub,movejob,runjob,provision,execjob_begin,execjob_prologue,execjob_e
    pilogue,execjob_preterm,execjob_end,exechost_periodic,execjob_launch,exechost_startup or ""
    for no event
```

If you specify an invalid value for a hook's debug attribute, the following error message appears in qmgr's stderr and is returned by pbs_geterrmsg() after calling pbs_manager():

```
"unexpected value '<bad_val>' must be (not case sensitive) true|t|y|1|false|f|n|0"
```

A runjob hook cannot set the value of a Resource_List member other than those listed in "Table: Reading & Setting Built-in Job Resources in Hooks" on page 60 in the PBS Professional Hooks Guide. Setting any of the wrong resources results in the following:

- The hook request is rejected
- The following message is the output from calling `pbs_geterrmsg()` after the failed `pbs_runjob()`:

```
" request rejected by filter hook: '<hook name>' hook failed to set job's
    Resource_List.<resc_name> = <resc_value> (not allowed)"
```

27.5.7 See Also

pbs_connect, pbs_statfree, “Hook Attributes” on page 350 of the PBS Professional Reference Guide

28
Custom Authentication and Encryption Library APIs

This chapter describes LibAuth, which contains APIs you can use to create your own custom authentication or encryption library.

Call the `pbs_auth_set_config` API first before calling any other LibAuth API.

Table of Authentication and Encryption APIs

9.1 pbs_auth_set_config . . . 122
9.2 pbs_auth_create_ctx . . . 123
9.3 pbs_auth_destroy_ctx . . . 125
9.4 pbs_auth_get_userinfo . . . 126
9.5 pbs_auth_process_handshake_data . . . 128
9.6 pbs_auth_encrypt_data . . . 130
9.7 pbs_auth_decrypt_data . . . 131

28.1 pbs_auth_set_config

specify configuration for authentication library

28.1.1 Synopsis

*void pbs_auth_set_config(const pbs_auth_config_t *auth_config)*

28.1.2 Description

Specifies configuration for the authentication library. Use this to specify logging method, where to find required credentials, etc.

Call this API first before calling any other LibAuth API.

28.1.3 Arguments

const pbs_auth_config_t *auth_config

Pointer to a configuration structure

28.1.4 Configuration Structure

typedef struct pbs_auth_config {

char *pbs_home_path;

Path to PBS_HOME directory (aka same value as PBS_HOME in pbs.conf). This must be a null-terminated string.

char *pbs_exec_path;

Path to PBS_EXEC directory (aka same value as PBS_EXEC in pbs.conf). This must be a null-terminated string.

char *auth_method;

Name of authentication method (aka same value as PBS_AUTH_METHOD in pbs.conf). This must be a null-terminated string.

char *encrypt_method;

Name of encryption method (aka same value as PBS_ENCRYPT_METHOD in pbs.conf). This must be a null-terminated string

void (*logfunc)(int type, int objclass, int severity, const char *objname, const char *text);

Function pointer to the logging method with the same signature as log_event from Liblog.

With this, the user of the authentication library can redirect logs from the authentication library into respective log files or stderr in case no log files.

If func is set to NULL then logs will be written to stderr (if available, else no logging at all).

} pbs_auth_config_t;

28.1.5 Return Value

None return value

28.2 pbs_auth_create_ctx

create authentication context

28.2.1 Synopsis

*int pbs_auth_create_ctx(void **ctx, int mode, int conn_type, char *hostname)*

28.2.2 Description

Creates an authentication context for a given mode and connection type. Context is used by other LibAuth APIs for authentication, encryption, and decryption of data.

28.2.3 Arguments

void **ctx

Pointer to auth context to be created

int mode

Specifies type of context to be created. Should be one of *AUTH_CLIENT* or *AUTH_SERVER*.

Use AUTH_CLIENT for client-side (who is initiating authentication) context

Use AUTH_SERVER for server-side (who is authenticating incoming user/connection) context

```
enum AUTH_ROLE {
    AUTH_ROLE_UNKNOWN = 0,
    AUTH_CLIENT,
    AUTH_SERVER,
    AUTH_ROLE_LAST
};
```

int conn_type

Specifies type of connection is for context to be created. Should be one of *AUTH_USER_CONN* or *AUTH_SERVICE_CONN*

Use AUTH_USER_CONN for user-oriented connection (such as when PBS client is connecting to PBS server)

Use AUTH_SERVICE_CONN for service-oriented connection (such as when PBS MoM is connecting to PBS server via PBS comm)

```
enum AUTH_CONN_TYPE {
    AUTH_USER_CONN = 0,
    AUTH_SERVICE_CONN
};
```

char *hostname

The null-terminated hostname of another authenticating party

28.2.4 Return Value

Integer.

0
: On Success

1
: On Failure

28.2.5 Cleanup

When a context created using this API is no longer required, destroy it via `auth_destroy_ctx.`

28.3 pbs_auth_destroy_ctx

destroy an authentication context created using `auth_create_ctx`

28.3.1 Synopsis

*void pbs_auth_destroy_ctx(void *ctx)*

28.3.2 Description

Destroys the authentication context created using `auth_create_ctx`

28.3.3 Arguments

void *ctx
: Pointer to authentication context to be destroyed

28.3.4 Return Value

No return value

28.4 pbs_auth_get_userinfo

extract username with its realm, and hostname of connecting party from authentication context

28.4.1 Synopsis

*int pbs_auth_get_userinfo(void *ctx, char **user, char **host, char **realm)*

28.4.2 Description

Extracts username with its realm, and hostname of the connecting party from the given authentication context.

The extracted user, host, and realm values are null-terminated strings.

This API is mostly useful for authenticating on the server side to get information about an authenticating client.

The authenticating server can use this information from the auth library to match against the actual username/realm/hostname provided by the connecting party.

28.4.3 Arguments

void *ctx
: Pointer to auth context from which information will be extracted

char **user
: Pointer to a buffer in which this API will write the user name

char **host
: Pointer to a buffer in which this API will write hostname

char **realm
: Pointer to a buffer in which this API will write the realm

28.4.4 Return Value

Integer

0
: On success

1
: On failure

28.4.5 Cleanup

When the returned user, host, and realm are no longer required, free them using `free()`, since they use allocated heap memory.

28.4.6 Example

This example shows what the values of user, host, and realm will be. Let's use an example with GSS/Kerberos authentication, where the authentication client hostname is "xyz.abc.com", the username is "test", and in the Kerberos configuration, the domain realm is "PBSPRO". When the client authenticates to the server using the Kerberos authentication method, it is authenticated as "test@PBSPRO", and this API returns user = test, host = xyz.abc.com, and realm = PBSPRO.

28.5 pbs_auth_process_handshake_data

handle and generate handshakes, and generate handshake data

28.5.1 Synopsis

*int pbs_auth_process_handshake_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out, int *is_handshake_done)*

28.5.2 Description

Process incoming handshake data and do the handshake. If required generate handshake data which to be sent to another party. If there is no incoming data then initiate a handshake and generate initial handshake data to be sent to the authentication server.

28.5.3 Arguments

void *ctx
: Pointer to authentication context for which handshake is happening

void *data_in
: Incoming handshake data to process, if any. A NULL value indicates that this API should initiate a handshake and generate initial handshake data to be sent to the authentication server.

size_t len_in
: Length of incoming handshake data, if any, else 0

void **data_out
: Outgoing handshake data to be sent to another authentication party.

A NULL value indicates that the handshake is completed and no further data needs to be sent.

When this API returns 1 (failure), `data_out` contains the error message.

size_t *len_out
: Length of outgoing handshake/auth error data, if any, else 0

int *is_handshake_done
: Indicates whether handshake is completed or not.

0 means that the handshake is not completed.

1 means that the handshake is completed.

28.5.4 Return Value

Integer

0
: On success

1
: On failure

On failure, the value of `data_out` is the error data/message, to be sent to another authentication party as authentication error data.

28.5.5 Cleanup

When the returned `data_out` (if any) is no longer required, free it using `free()`, since it uses allocated heap memory.

28.6 pbs_auth_encrypt_data

encrypt data using specified authentication context

28.6.1 Synopsis

*int pbs_auth_encrypt_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out)*

28.6.2 Description

Encrypt given unencrypted data using the specified authentication context.

28.6.3 Arguments

void *ctx
: Pointer to auth context which will be used while encrypting given unencrypted data

void *data_in
: unencrypted data to encrypt

size_t len_in
: Length of unencrypted data

void **data_out
: Encrypted data

size_t *len_out
: Length of encrypted data

28.6.4 Return Value

Integer

0
: Success

1
: Failure

28.6.5 Cleanup

When the returned `data_out` is no longer required, free it using `free()`, since it uses allocated heap memory.

28.7 pbs_auth_decrypt_data

decrypt data

28.7.1 Synopsis

*int pbs_auth_decrypt_data(void *ctx, void *data_in, size_t len_in, void **data_out, size_t *len_out)*

28.7.2 Description

Decrypt encrypted data using the specified authentication context

28.7.3 Arguments

void *ctx
: Pointer to authentication context which will be used while decrypting given encrypted data

void *data_in
: Encrypted data to decrypt

size_t len_in
: Length of encrypted data

void **data_out
: Unencrypted data

size_t *len_out
: Length of unencrypted data

28.7.4 Return Value

Integer

0
: On success

1
: On failure

28.7.5 Cleanup

When the returned `data_out` is no longer required, free it using `free()`, since it uses allocated heap memory.

Index

A
activereq PG-100
addreq PG-100
allreq PG-100

C
closerm PG-100
commands PG-4
configrm PG-100
credential PG-21

D
downrm PG-100

E
executor PG-4

F
flushreq PG-100
fullresp PG-100

G
getreq PG-100

J
job
    executor (MoM) PG-4
    scheduler PG-4

M
MoM PG-3, PG-4

O
openrm PG-100

P
pbs_alterjob PG-24
pbs_asyrunjob PG-26, PG-57
pbs_auth_create_ctx PG-123
pbs_auth_decrypt_data PG-131
pbs_auth_destroy_ctx PG-125
pbs_auth_encrypt_data PG-130
pbs_auth_get_userinfo PG-126
pbs_auth_process_handshake_data PG-128
pbs_auth_set_config PG-122
pbs_connect PG-21, PG-30
pbs_default PG-32
pbs_deljob PG-33
pbs_delresv PG-35
pbs_disconnect PG-36
pbs_geterrmsg PG-37
pbs_holdjob PG-38
pbs_iff PG-21
pbs_locjob PG-39
pbs_manager PG-41
pbs_module PG-111
pbs_mom PG-3, PG-4
pbs_movejob PG-47
pbs_msgjob PG-49
pbs_orderjob PG-51
pbs_preempt_jobs PG-52
pbs_relnodesjob PG-54
pbs_rerunjob PG-55
pbs_rlsjob PG-56
pbs_runjob PG-26, PG-57
pbs_sched PG-2, PG-3, PG-4
pbs_selectjob PG-59
pbs_selstat PG-62
pbs_server PG-2, PG-3
pbs_sigjob PG-66
pbs_statfree PG-68
pbs_stathook(3B) PG-117
pbs_stathost PG-69
pbs_statjob PG-71
pbs_statnode PG-74
pbs_statque PG-76
pbs_statresv PG-78
pbs_statrsc PG-80
pbs_statsched PG-82
pbs_statserver PG-84
pbs_statvnode PG-86
pbs_submit PG-88
pbs_submit_resv PG-90
pbs_tclapi PG-104
pbs_tclsh PG-103
pbs_terminate PG-92
pbs_wish PG-103

S

scheduler PG-3, PG-4
server PG-3

T

TCL PG-103
tm_atnode PG-94
tm_attach PG-94
tm_finalize PG-94
tm_init PG-94
tm_kill PG-94
tm_nodeinfo PG-94
tm_notify PG-94
tm_obit PG-94
tm_poll PG-94
tm_publish PG-94
tm_rescinfo PG-94
tm_spawn PG-94
tm_subscribe PG-94
tm_taskinfo PG-94



Altair PBS Professional 2021.1

Cloud Guide

You are reading the Altair PBS Professional 2021.1

Cloud Guide (CG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation i

1 Introduction to PBS Cloud 1
- 1.1 Introduction to Cloud Bursting . . . 1
- 1.2 Cloud Bursting Terminology . . . 1
- 1.3 How PBS Cloud Bursting Works . . . 2

2 Installing the PBS Cloud Module 5
- 2.1 Supported Platforms for PBS Cloud . . . 5
- 2.2 Prerequisites . . . 6
- 2.3 Recommended Configurations . . . 6
- 2.4 Installation Steps . . . 7

3 Configuring PBS Cloud 11
- 3.1 Overview of Configuring PBS Cloud . . . 11
- 3.2 Configuring PBS Professional for Cloud Bursting . . . 12
- 3.3 Configuring PBS Cloud . . . 15
- 3.4 Testing Cloud Bursting . . . 31

4 Configuring the Cloud Bursting Hook 33
- 4.1 The Cloud Bursting Hook . . . 33
- 4.2 Configuring the Cloud Bursting Hook . . . 34
- 4.3 Testing Automated Cloud Bursting . . . 40

5 Using Cloud Provider Services 43
- 5.1 Configuring Amazon Web Service Cloud Bursting . . . 43
- 5.2 Configuring Microsoft Azure Cloud Bursting . . . 54
- 5.3 Configure Google Cloud Platform Cloud Bursting . . . 65
- 5.4 Configure Oracle Cloud Platform Cloud Bursting . . . 72
- 5.5 Configure Orange Cloud Flexible Engine for Cloud Bursting . . . 82
- 5.6 Configure HUAWEI Cloud for Cloud Bursting . . . 89
- 5.7 Configure Open Telekom Cloud for Cloud Bursting . . . 95
- 5.8 Configure OpenStack Cloud Bursting . . . 102
- 5.9 Windows Bursting on AWS and Azure . . . 104

6 The Cloud Node Startup Script 107
- 6.1 Introduction . . . 107
- 6.2 Customizing Your Startup Script . . . 108

7 Managing Cloud Bursting 115
- 7.1 Managing Cloud Bursting . . . 115
- 7.2 Starting and Stopping PBS Cloud . . . 117
- 7.3 Logging into PBS Cloud . . . 118
- 7.4 Troubleshooting Cloud Bursting . . . 118

8 Managing Cloud Jobs 121
8.1 Manage Cloud and On-Premise Jobs . . . 121
8.2 Run Cloud Jobs On-Premise Before Bursting . . . 122
9 Running Cloud Jobs 125
9.1 Introduction . . . 125
9.2 Sample Job Script: Cloud Job . . . 125
9.3 Logging into PBS Cloud . . . 126
10 Example Azure Head Node 127
10.1 Example Configuration of Cloud Head Node in Azure . . . 127
11 Command Reference 131
11.1 PBS Cloud CLI pclm Commands . . . 131
11.2 Command Reference and Sample Output for pkr . . . 138
Index 143

1 Introduction to PBS Cloud

1.1 Introduction to Cloud Bursting

PBS Cloud allows PBS Professional to burst nodes in the cloud, so that your site can handle demand peaks. PBS Professional uses a cloud bursting hook called cloud_hook to analyze jobs from the cloud queue(s), estimate the demand, and burst the required cloud nodes having the specified instance type and OS image. The PBS scheduler runs the jobs from the cloud queue in the cloud nodes. The cloud bursting hook dynamically adjusts the number of cloud nodes according to current load and how long you want nodes to wait for jobs to appear.

PBS Cloud provides the framework for the interface to the cloud. PBS Cloud supports multiple cloud vendors as well as private OpenStack clouds. You can use multiple vendors at the same time, and multiple accounts at each vendor. PBS Cloud also supports instance types that are on-demand, preemptable (GCP) and spot (AWS and Azure). PBS Cloud supports Infiniband, MPI, and scaleset jobs.

1.2 Cloud Bursting Terminology

Burst

The action of creating a node in the cloud and adding it to the PBS complex

Instance type

A specification for a machine including characteristics such as CPUs, memory, storage capacity, network technology, etc. The PBS Cloud administrator specifies the instance types that will be available to job submitters. Jobs can request and use only instance types that the administrator has made available.

PBS Cloud supports instance types that are on-demand, preemptable (GCP) and spot (AWS and Azure).

Unburst

The action of removing a node from both the PBS complex and the cloud.

Cloud queue

Each scenario uses its own cloud queue. This is where the jobs for that scenario are enqueued. Cloud jobs must be submitted to the appropriate cloud queue.

Scenario

A bursting scenario encapsulates information needed to burst cloud nodes, such as the default OS image and which cloud-init script should initialize cloud nodes. You create a scenario data structure in PBS Cloud; this is where you specify information about resources provided by the cloud vendor that PBS Cloud uses for bursting. You define other aspects of the same scenario in the cloud bursting hook. A scenario can use one or more instance types to burst cloud nodes, although all the instance types for a scenario must be non-preemptable or be preemptable. You can have as many scenarios as you want.

Cloud Node

A virtual machine that has been created on cloud hardware. Each cloud node is burst using the OS image specified for the job. After the node is burst, it is initialized via cloud-init scripts with everything required to run PBS jobs.

Cloud Bursting Hook

The cloud bursting hook is called cloud_hook, and it is installed when you install PBS Cloud. The PBS cloud bursting hook manages cloud nodes and jobs via PBS Cloud and cloud queues. You specify details for each scenario that you want the hook to handle.

OS Image

A pre-configured OS image in the cloud from which virtual machines can be instantiated. At the vendor, you create an OS image to use as the default for a particular scenario at that vendor. Jobs can request a specific OS image, the cloud bursting hook can specify a default OS image for that scenario, and you can set a default OS image for the cloud queue for that scenario.

1.3 How PBS Cloud Bursting Works

1.3.1 How Node Bursting Works

You create an administrator account with your cloud vendor. PBS Cloud uses this vendor administrator account manage cloud nodes.

You create a cloud queue for each scenario, job submitters request the cloud queue for their jobs, and the cloud bursting hook analyzes the resources needed by those jobs and bursts the required cloud nodes via PBS Cloud.

The PBS scheduler runs the cloud jobs in the cloud nodes.

1.3.1.1 OS Image and Instance Type Assignment to Job

The OS image and instance type used for a particular job depend on whether the job requests it, or whether the job inherits it along the way. Assignment works in this order, with the first one encountered being the one assigned:

- Job request
- Queue default
- Cloud bursting hook default
- PBS Cloud scenario default

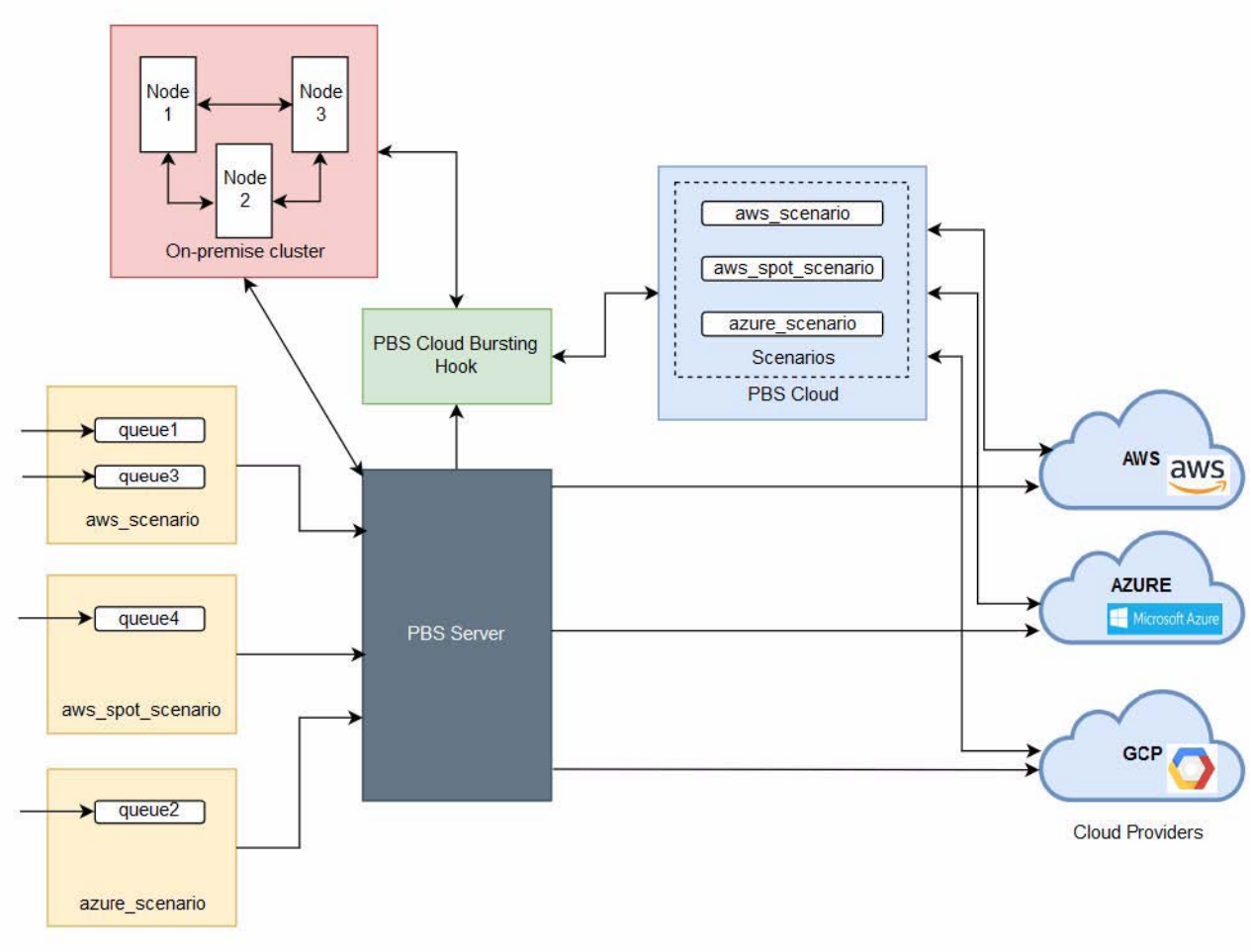


Figure 1-1:Typical Cloud Bursting Setup

2 Installing the PBS Cloud Module

2.1 Supported Platforms for PBS Cloud

2.1.1 Supported Platforms for PBS Cloud Head Node

- CentOS 7, 8
- RHEL 7, 8
- SLES 12, with restrictions:
  - Each SLES host must be registered with the SUSE Customer Center via SUSEConnect, and have a support contract. This happens automatically for cloud nodes.
  - SLES hosts require Docker Enterprise Edition.

2.1.2 Supported Platforms for Nodes Burst in Cloud

- Linux: any Linux platform that supports both PBS MoM and `cloud-init`
- Windows: 10, Server 2012

2.1.3 Supported Cloud Providers

You must already have an account with one of the supported cloud providers. We support the listed variant for each cloud provider:

Table 2-1: Supported Clouds

Cloud Provider	Variant
Microsoft Azure	Azure Compute
Amazon Web Services (AWS)	Elastic Compute Cloud (EC2)
Google Cloud Platform (GCP)	Compute Engine
Oracle Cloud Platform	Oracle Cloud Infrastructure
Orange Technical Cloud (OTC)	Orange Flexible
Deutsche Telekom	Open Telekom Cloud (OTC)
HUAWEI Cloud	Elastic Cloud Server
OpenStack cloud on premise	Stein

2.2 Prerequisites

- A working PBS complex managed by PBS Professional version 2020.1. The PBS complex can be either of the recommended configurations in Recommended Configurations
- docker-ce v19.x or later for most systems
- docker-ee v19.x or later for SLES
- SELinux must be disabled (reboot the system to make this the active config)
- VPN connection to the cloud you will use
  - Except in the case of a cloud hosted head node.
- Cloud provider account with:
  - Correct authorizations
  - Approved method of payment

2.2.1 Licensing

Make sure that the time zone for your on premise hardware and all cloud nodes and your licenses is the same. You will also need the following:

- Altair License Server 14.5.1 or later
- PBSProNodes v20 license

2.2.2 Required Accounts

- To install PBS Cloud, you need to be root.
- To configure PBS Cloud, you must use pbsadmin@altair. This account is created by the PBS Cloud installer software during installation. The default password for pbsadmin@altair is Altair@123. We recommend changing the password to something known only to you.

2.3 Recommended Configurations

Head node and service node can be one of either:

- Both on premises
- Both in cloud

Do not put one on premises and one in the cloud.

See Chapter 10, "Example Azure Head Node", on page 127.

2.3.1 Recommended Configuration for Larger Installations

For larger installations using on premises hosts:

- Hosts on premises for PBS server, scheduler, some MoM daemons, cloud service node
- Head, service, and first N execution nodes are on premises
- Extra execution nodes are burst
- On head node, PBS components running as usual
- On service node
  - PBS Cloud is designed to run in a container
- VPN connection to the cloud you will use

Notes:

- You many not want to run PBS Cloud on the head node, because it runs in a Docker container, which may impose too high a load.
- All components are mix-and-match (with Docker restriction).
- You don't need to configure additional `pbs_comm` daemons for cloud nodes, because cloud can't cause enough throughput to need one.
- For PBS configuration instructions, see the *PBS Professional Administrator's Guide*.

2.3.2 Recommended Configuration for Smaller Installations

For smaller installations cloud-only installations where the workload is low enough:

- All PBS components can be hosted in the cloud
- All components can run on the same node
- You can run Docker on the same node as the PBS components

Notes:

- No VPN is required for this configuration.
- See Chapter 10, "Example Azure Head Node", on page 127.

2.4 Installation Steps

2.4.1 Install Docker

1. Log in as root
- Choose whether your head node will be on premises or in the cloud. If it will be in the cloud, build your cloud head node. For an Azure example, see Chapter 10, "Example Azure Head Node", on page 127.

2. Install, start, and enable `docker-ce`:

   - For CentOS or RedHat:

     Log in to the machine where the PBS Cloud package is to be installed.

     ```
     yum install -y yum-utils
     yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
     yum install docker-ce docker-ce-cli containerd.io
     systemctl enable docker
     systemctl start docker
     ```

   - For SLES12 or 15:

     Log in to the machine where Budget is to be installed.

     For SLES 12:

     ```
     sudo SUSEConnect -p sle-module-containers/12/x86_64 -r ''
     ```

     For SLES 15:

     ```
     sudo SUSEConnect -p sle-module-containers/15.1/x86_64 -r ''
     sudo zypper install docker
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```

     Configure the firewall to allow forwarding of Docker traffic to the external network:

     ```
     Set FW_ROUTE="yes" in /etc/sysconfig/SuSEfirewall2
     ```

   - For Ubuntu:

     Log in to the machine where Budget is to be installed.

     ```
     sudo apt-get update
     sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-proper-
         ties-common
     curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
     sudo apt-key fingerprint 0EBFCD88
     ```

     The key should match the second line in the output; validate the last 8 characters. Example of second line:

     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

     ```
     sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
         $(lsb_release -cs) stable"
     sudo apt-get update
     sudo apt-get install docker-ce docker-ce-cli containerd.io
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```

2.4.2 Install the PBS Cloud Module

1. Log in as root
2. Clean up any previous installations of the PBS Cloud module:
   - If the PBS cloud hook, named "cloud_hook", exists, delete it:
     ```
     qmgr -c 'd h cloud_hook'
     ```
   - If the pclm command exists in $PBS_EXEC/bin or /opt/pbs/bin, delete it:
     ```
     source /etc/pbs.conf
     rm $PBS_EXEC/bin/pclm
     ```
3. Extract the installer:
   ```
   tar xvfz PBSPro-cloud_2021.1-<OS>_x86_64.tar.gz
   ```
4. Install PBS cloud feature, PBS cloud hook, and PBS Cloud command layer. Execute the installation script:
   a. Change directory to pbspro-cloud-installer directory:
      ```
      cd pbspro-cloud-installer
      ```
   b. Run the installer:
      ```
      ./install.sh
      ```

2.4.3 Allow Easy PBS Cloud Status Check

1. Log in as root
2. Create an alias to easily use pkr. Type the following all one line:
   ```
   alias pkr="docker run -ti --network host --rm -e PBSCLOUD_VERSION=pbspro-cloud-2021.1 -e
       PKR_VERSION=pbspro-cloud-2021.1 -v /run/docker.sock:/run/docker.sock -v /root/kard:/pkr/kard
       pbscloudio.azurecr.io/pkr:pbspro-cloud-2021.1 pkr"
   ```

2.4.4 Test the Installation

1. Log in as root
2. Make sure that relevant services are up and running. Each should have an IP address. See section 11.2.3, “Sample pkr Output while Running”, on page 141 for sample healthy output reference data for pkr.
   ```
   pkr ps
   ```
3. If you are running the PBS Cloud module on a head node in the cloud, use the vendor tools to open access through the firewall to port 9980 so that you can use the vendor web interface.
4. Log into PBS Cloud from your web browser:
   http://<PBS Cloud host name or IP address>:<port>/pbspro-cloud/#/login
   - Default port: 9980
   - Username: pbsadmin@altair
   - Password: Altair@123

3 Configuring PBS Cloud

3.1 Overview of Configuring PBS Cloud

Much of the information required to configure PBS Cloud and PBS Professional is generated or chosen while you are logged into your cloud provider, building your cloud components. We provide a list of what to collect while you are doing this so that you will have the information you need later. Alternatively, you may want to have one window open for each purpose simultaneously, rather than performing these steps in sequence. We recommend reading through the instructions once before starting so that you can see where information is transferred from one tool to another.

1. If you have not done so already, install PBS Professional; see the PBS Professional Installation & Upgrade Guide
2. If you have not done so already, install the PBS Cloud module; see Chapter 2, "Installing the PBS Cloud Module", on page 5
3. Configure PBS Professional for cloud bursting. This step is covered in section 3.2, “Configuring PBS Professional for Cloud Bursting”, on page 12, but we outline it here for clarity:
    a. Log into the PBS server host as administrator
    b. Configure PBS Professional for bursting to cloud nodes
        1. Create and configure resources
        1. Create and configure cloud queue(s)
        2. Configure scheduling
4. Log into your cloud provider
5. PBS Cloud will use an account at the cloud provider to manage cloud nodes. Create a cloud provider account for this purpose; see section 3.3.2, “Create Your Cloud Provider Account”, on page 16.
6. Build and configure your cloud components. During this step, capture the information listed; while you are generating or selecting each item, we’ll remind you to collect it. You will use this information in the following steps here. Building and configuring cloud components is different for each cloud provider; use the provider-specific instructions in Chapter 5, "Using Cloud Provider Services", on page 43. We show an outline here so you can see what is covered:
    a. Log into your cloud provider
    b. Create the necessary cloud provider components, such as a virtual network
    c. Create a virtual machine
    d. Install the PBS Professional MoM, and other required software in the VM, including cloud-init
    e. If you will use cloud-init to configure freshly burst nodes, create a startup script for configuring burst nodes; see Chapter 6, "The Cloud Node Startup Script", on page 107
    f. Create the OS image to be used for bursting

7. Configure PBS Cloud. This step is described in section 3.3, "Configuring PBS Cloud", on page 15, but we give you an idea of what's involved here:
   a. Log into PBS Cloud; see section 3.3.1, "Log Into PBS Cloud", on page 15
   b. Add your cloud provider account to PBS Cloud; see section 3.3.3, "Add Your Provider Account to PBS Cloud", on page 16
   c. Create a bursting scenario; see section 3.3.4, "Create a Bursting Scenario", on page 19
   d. Manually test bursting; see section 6.2.5, "Developing the Startup Script", on page 111
   e. For each remaining scenario, create and test the scenario
   f. Disable public IP addresses for the scenario
8. Configure the PBS cloud bursting hook; see section 4.1, "The Cloud Bursting Hook", on page 33
   a. Log into the PBS server host as administrator
   b. Configure and enable the cloud hook
   c. For each scenario, test it with the cloud hook by using it to automatically burst scenario nodes; see section 4.3, "Testing Automated Cloud Bursting", on page 40
9. To prevent running out of PBSProNodes licenses, set a limit on the numbe of cloud nodes that can exist simultaneously. See section 3.3.6, "Manage Licenses", on page 30

3.2 Configuring PBS Professional for Cloud Bursting

Each PBS Cloud scenario requires its own cloud queue, and vice versa. You associate a cloud queue with a scenario by using the same name for both; at the queue you set the cloud_scenario resource to be identical to the name set for the PBS Cloud scenario.

3.2.1 Create and Configure Custom Resources for Cloud Bursting

3.2.1.1 List of PBS Professional Custom Resources for Cloud Bursting

PBS Professional uses the following custom resources to manage cloud jobs:

burst_by_hook
: Used by cloud bursting hook and PCLM to distinguish manually burst cloud nodes from automatically burst cloud nodes. Do not set.

cloud_instance_type
: Queue-level string

  Default cloud provider instance type (machine, shape type or flavor) associated with this queue.

cloud_min_instances
: Server-level integer

  Minimum number of cloud nodes (instances) to be present in the cloud at any time.

cloud_max_instances

Server-level integer

Maximum number of cloud nodes (instances) that can be present in the cloud at any time.

cloud_max_jobs_check_per_queue

Queue-level integer

Maximum number of jobs in the cloud queue to be checked to determine the number of cloud nodes that must be burst based on the requested instance type. The cloud bursting hook computes the number of nodes it must burst in order to run the checked cloud jobs. For example, if the instance type has 2 CPUs, and the first 3 jobs in the queue need a total of 10 CPUs, the hook bursts 5 nodes.

Must be greater than zero. Setting this to zero results in no jobs from this queue being considered for cloud bursting.

Default: 64

cloud_network

Host-level string

For requesting InfiniBand-enabled nodes.

Nodes with the same network name are grouped.

cloud_node_image

Host-level string

Default OS image to use when bursting a cloud node. Overridden if job specifies an image.

cloud_node_instance_type

Host-level string

Set this at the cloud queue to the default instance type for cloud nodes burst for this queue. Make sure you choose an instance type that is enabled in PBS Cloud.

This is set at the cloud node by the cloud bursting hook when that node is burst.

If a job requests an instance type, that overrides the default set at the queue.

cloud_provisioned_time

Host-level integer

Time in seconds from since the cloud bursting hook started bursting cycle; if node doesn't successfully come up in that time, the cloud bursting hook tries again

cloud_queue

Queue-level Boolean

Indicates whether the queue is a cloud queue. When True, the queue is a cloud queue. The cloud bursting hook monitors only cloud queues. Cloud jobs must be submitted to a cloud queue.

cloud_scenario

Node-level string.

Indicates the associated scenario type for the queue and the node. Functions as placement tool.

lic_signature

Host-level string

Contains licensing information.

node_location

Host-level string

Used to differentiate on-premise nodes from cloud nodes. Lets you keep a job either all in the cloud or all on premise. If you have multiple clouds, name this for the cloud.

3.2.1.2 Create Cloud Custom Resources for Cloud Bursting

1. Log in to the PBS server host as the PBS administrator
2. Create the custom resources required for cloud bursting:

```
qmgr -c 'create resource cloud_queue type=boolean'
qmgr -c 'create resource cloud_instance_type type=string'
qmgr -c 'create resource cloud_node_instance_type type=string,flag=h'
qmgr -c 'create resource cloud_min_instances type=long'
qmgr -c 'create resource cloud_max_instances type=long'
qmgr -c 'create resource cloud_provisioned_time type=long,flag=h'
qmgr -c 'create resource lic_signature type=string,flag=h'
qmgr -c 'create resource cloud_node_image type=string,flag=h'
qmgr -c 'create resource cloud_network type=string,flag=h'
qmgr -c 'create resource node_location type=string,flag=h'
qmgr -c 'create resource cloud_max_jobs_check_per_queue type=long'
qmgr -c 'create resource cloud_scenario type=string,flag=h'
qmgr -c 'create resource burst_by_hook type=boolean'
```

3.2.2 Configure PBS Server and Scheduler for Cloud Bursting

1. Log in to the PBS server host as the PBS administrator
2. Change directory to `$PBS_HOME/sched_priv`
3. Edit the `sched_config` file and add cloud_scenario, cloud_node_image, and cloud_node_instance_type to the `resources:` line:

```
resources: "ncpus, mem, arch, host, vnode, netwins, aoe, cloud_scenario, cloud_node_image,
    cloud_node_instance_type"
```

4. Make the scheduler reread its configuration file; HUP the scheduler:

```
kill -HUP <scheduler PID>
```

5. Set server limits:

```
qmgr -c "set server resources_available.cloud_min_instances = 0"
qmgr -c "set server resources_available.cloud_max_instances = <max nodes>"
```

3.2.3 Create and Configure Cloud Queues

Each PBS Cloud scenario requires its own cloud queue, and vice versa. You associate a cloud queue with a scenario by using the same name for both; at the queue you set the cloud_scenario resource to be identical to the name set for the PBS Cloud scenario. You may find it helpful to name each cloud queue with information that makes it easy to match the queue with its purpose.

The instance type that is burst is defined at the queue level, unless overridden at job submission time.

1. Log in to the PBS server host as root.
2. Create a queue:

```
qmgr -c "create queue <queue name> queue_type=execution,enabled=true,started=true"
```

3. Make this queue into a cloud queue:

   ```
   qmgr -c "set queue <queue name> resources_available.cloud_queue = True"
   ```

4. Optionally, set the maximum number of jobs to check in the queue to determine the number of nodes to burst:

   ```
   qmgr -c "set queue <queue name> resources_available.cloud_max_jobs_check_per_queue = <max jobs to check>"
   ```

   Must be greater than 0; setting to zero results in no bursting. Default is 64.

5. Assign a bursting scenario to the queue:

   ```
   qmgr -c "set queue <queue name> resources_available.cloud_scenario = <scenario name>"
   ```

   The queue scenario name must be identical to the scenario name in the cloud bursting hook configuration file. See Chapter 4, "Configuring the Cloud Bursting Hook", on page 33.

   Here is an example scenario named "azure_scenario_1" as it would appear in the cloud bursting hook configuration file:

   ```
   "scenario": {
       "azure_scenario_1": {
           "api_key": "<API key>",
           "cloud_default_image": "azure_bursting_image1",
           "cloud_max_instances": 10,
           "max_vms_for_infiniband_scaleset" : 100,
           "max_nodes_per_burst":50,
           "cloud_node_wait_timeout":180,
           "preemptable": false
       }
   }
   ```

6. Assign a default instance type to the queue:

   ```
   qmgr -c "set queue <queue name> resources_available.cloud_instance_type = <instance type>"
   ```

   This is the default instance type to be burst for this queue. It must match one of the instance types enabled for the bursting scenario that you assigned to the queue. For example:

   ```
   qmgr -c "set queue <queue name> resources_available.cloud_instance_type = Standard_DS2_v2"
   ```

   See section 3.3.4.5.i, “Cloud Provider Instance Types”, on page 24.

3.3 Configuring PBS Cloud

3.3.1 Log Into PBS Cloud

You must be logged into PBS Cloud as pbsadmin@altair account to configure PBS Cloud.

Log into PBS Cloud from your web browser:

```
http://<PBS Cloud host name or IP address>:<port>/pbspro-cloud/#/login
```

- Default port: 9980
- Username: pbsadmin@altair
- Password: Altair@123

We recommend changing the password immediately; use the UX to do this.

3.3.2 Create Your Cloud Provider Account

PBS Cloud uses an administrator account at the cloud provider to manage cloud nodes. Create your cloud provider account; follow the steps for your cloud provider in section , “Using Cloud Provider Services”, on page 43. While you are creating the account, capture the information we list at the start of the provider-specific instructions; you will use that information to add your cloud provider account to PBS Cloud.

3.3.3 Add Your Provider Account to PBS Cloud

PBS Cloud can use multiple cloud providers, and can use multiple accounts at each provider.

Tell PBS Cloud about the cloud account you have created at your cloud provider by filling in the parameters.

The Name parameter is common to all accounts, meaning that all accounts have a name, but the name can be different for each provider. The other parameters for an account are different for each vendor; for example, only Oracle uses a fingerprint ID. Follow the steps for your vendor to capture the information we list so that you can use it to fill in the account parameters. For the vendor-specific steps, see Chapter 5, "Using Cloud Provider Services", on page 43.

The information required varies by cloud provider.

1. Click the Cloud tab.
2. Under Infrastructure, click Cloud.
3. Select your cloud provider
4. Fill in the account name. This is an arbitrary string, and it is not the administrator account name created at the vendor. We recommend making it informative.

5. Add the information for the parameters that are vendor-specific. We list the parameter information by vendor:

   - For an Amazon Web Services (AWS) account:

Table 3-1: Account Parameters for Amazon Web Services (AWS)

Account Parameter	Collected During Configuration at Vendor	Format
Access Key ID	Access Key ID from vendor `.csv` file	String
Secret Access Key	Secret Access Key from vendor `.csv` file	String

   - For an Azure account:

Table 3-2: Account Parameters for Microsoft Azure

Account Parameter	Collected During Configuration at Vendor	Format
Client ID	Application ID generated when registering PBS Cloud with the Azure Active Directory	String
Secret Key	Secret Key generated during account creation at vendor	String
AD Tenant ID	Azure tenant ID generated during account creation at vendor	String
Subscription ID	Subscription ID generated during account creation at vendor	String

   - For a Google Cloud Platform (GCP) account:

Table 3-3: Account Parameters for Google Cloud Platform (GCP)

Account Parameter	Collected During Configuration at Vendor	Format
Project ID	Value of project_id in JSON file created at vendor	String
Client ID	Value of client_id in JSON file created at vendor	String
Client Mail	Value of client_email in JSON file created at vendor	String
Private Key ID	Value of private_key_id in JSON file created at vendor	String
Private Key	Value of private_key in JSON file created at vendor	String

   - For an Oracle account:

Table 3-4: Account Parameters for Oracle

Account Parameter	Collected During Configuration at Vendor	Format
User OCID	User OCID generated when creating Oracle cloud user account at vendor	String
Tenant OCID	Tenancy OCID generated at vendor	String

Table 3-4: Account Parameters for Oracle

Account Parameter	Collected During Configuration at Vendor	Format
Compartment OCID	Root compartment OCID generated at vendor	String
Fingerprint OCID	Fingerprint generated when adding the public SSH key for Oracle user at vendor	String
Private Key	RSA private key generated at vendor	String

- For a Huawei, Deutsche Telekom, Orange, or OpenStack account:

Table 3-5: Account Parameters for Huawei, Deutsche Telekom, Orange, OpenStack

Account Parameter	Collected During Configuration at Vendor	Format
Auth URL	Orange: *https://iam.<orange region>.<console link>* Huawei: *https://iam.ap-southeast-1.myhwclouds.com* Deutsche Telekom: *https://iam.eu-de.otc.t-systems.com/v3* OpenStack: contact Altair support	String
User Domain Name	Orange: Customer ID used to log in to Orange account. Same as domain name Deutsche Telekom: OTC domain name used to log in to OTC console at vendor Huawei: Domain Name provided when your subscription to HUAWEI Cloud was created OpenStack: Domain name for cloud account in private cloud	String
Username	Administrator username created at vendor	String
Password	Huawei & Deutsche Telekom: Administrator password created at vendor Orange: API password generated at vendor OpenStack: Password for administrator account	String

6. Click Create account.

3.3.3.1 Example of Adding Azure Account to PBS Cloud

1. Log in to PBS Cloud.
2. Click the Cloud tab.
3. Under Infrastructure, click Cloud.
4. Click 

5. Enter the following to add a cloud account:
   a. For Account name, enter any name for the cloud account.
      The name can be anything meaningful to your organization, e.g., azure_cloudaccount.
   b. For Client ID, enter the Application ID generated when PBS Cloud was registered with the Azure Active Directory.
   c. For Secret Key, enter the client secret key generated when you register PBS Cloud.
   d. For AD tenant ID, enter your Azure tenant ID.
   e. For Subscription ID, enter your Azure subscription ID.
6. Click Create account.

3.3.4 Create a Bursting Scenario

A bursting scenario encapsulates information needed to burst cloud nodes. Some scenario parameters are common to all scenarios, meaning that all scenarios have an associated hostname prefix, but the hostname prefix can be different for each scenario. Some parameters for a bursting scenario are different for each vendor; for example, only Azure uses a resource group. Follow the steps for your vendor to capture the information we list so that you can use it to fill in the scenario parameters. For the vendor-specific steps, see Chapter 5, "Using Cloud Provider Services", on page 43.

You can create an unlimited number of bursting scenarios. We recommend creating and testing them one at a time.

1. Go to the bursting scenario interface in PBS Cloud:
   a. Open a browser window and log in to PBS Cloud.
   b. Click on Cloud.
   c. Under Infrastructure, click on Bursting.
   d. Click Add Bursting Scenario.
2. Add the information for the following parameters common to all scenarios:

Table 3-6: Common Scenario Parameters

Scenario Parameter	Description	Format
Name	Arbitrary friendly scenario name	String
Description	Arbitrary scenario description	String
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor; see section 3.3.4.1, “Adding Hostname Prefix”, on page 23	String
Add public IP to VMs	Optional public IP address to VMs for initial troubleshooting; see section 3.3.4.4, “Temporarily Adding Public IP for Debugging”, on page 23	Checkbox

Table 3-6: Common Scenario Parameters

Scenario Parameter	Description	Format
cloud-init	Name of startup script launched by `cloud-init` for configuring freshly burst nodes. Browse to a file, and optionally edit the file; see section 3.3.4.9, "Specifying the Cloud Node Startup Script", on page 26	String
SSH keys	Administrator SSH key to give you access for initial debugging if `/home` won't mount in the node; see section 3.3.4.8, "Adding SSH Key for Access to Burst Nodes", on page 26	String
Idle time before unbursting	Time for burst node to sit idle before unbursting. Default: 180 seconds; see section 3.3.4.2, "Setting Idle Time", on page 23	Integer seconds
Tag	Optional labeling system for convenience. You can apply multiple tags. See section 3.3.4.7, "Adding Tags (Labels) to Scenario", on page 25	Key:value pair as *<string>:<string>*
Instances	Instances that will be available for job submitters to select for their jobs. Instances must be all non-preemptable or all preemptable; see section 3.3.4.5, "Managing Instances", on page 24	Checkboxes

3. Add the information for the parameters that are vendor-specific:

   - For an AWS scenario:

Table 3-7: Scenario Parameters for Amazon Web Services (AWS)

Scenario Parameter	Collected During Configuration at Vendor	Format
AMI ID	Name of image to be burst; chosen during configuration at vendor	String
Security Group ID	Name of security group associated with VPC and VM created at vendor	String
Subnet ID	Name of security group subnet for bursting VPC created at vendor. To burst nodes in multiple Availability Zones, use a comma-separated list of subnet IDs	String

   - For a GCP scenario:

Table 3-8: Scenario Parameters for Google Cloud Platform (GCP)

Scenario Parameter	Collected During Configuration at Vendor	Format
Network name	Name of VPC for cloud bursting created at vendor	String
Subnetwork name	Name of VPC network subnet created at vendor	String
OS Image URI	Value for entry called selfLink of REST equivalent of OS image created for cloud bursting at vendor	String

   - For an Azure scenario:

Table 3-9: Scenario Parameters for Microsoft Azure

Scenario Parameter	Collected During Configuration at Vendor	Format
Resource group name	Name of resource group (virtual network, virtual machine, OS image) created at vendor	String
Network name	Name of virtual network created at vendor If the network is in a different resource group from the one specified, enter it as Resource Group Name/Virtual Network Name	String
Subnetwork name	Name of virtual subnet created at vendor If the subnet is in a different resource group from the one specified, enter it as Resource Group Name/Subnet Name	String
Network security group name	Name of network security group for resource group If the network security group is contained in a resource group that is different from the one entered for the bursting scenario, enter it as Resource Group Name/Network Security Group Name.	String

Table 3-9: Scenario Parameters for Microsoft Azure

Scenario Parameter	Collected During Configuration at Vendor	Format
Managed Storage	Managed disk feature selected at vendor	Boolean
OS Image	If using managed disks, name of the image. If not using managed disks, Linux Source BLOB URI. If the OS image is contained in a Resource group that is different from the one entered for the bursting scenario, enter it as Resource Group Name/OS Image Name or Resource Group Name/URI.	String
Maximum number of VMs inside a ScaleSet with Managed Storage and a single Placement Group	Limit selected during configuration at vendor. Default: 100 If you use InfiniBand, Azure limits the number of VMs on a scaleset to 100. Without InfiniBand, for a scaleset with managed disk and custom image, you can specify a higher limit.	Integer

- For an Oracle scenario:

Table 3-10: Scenario Parameters for Oracle

Scenario Parameter	Collected During Configuration at Vendor	Format
Subnet ID	OCID of subnet associated with data center where cloud bursting virtual machine is hosted	String
OS Image URI	Vendor link to bursting image OCID	String

- For a Huawei, Deutsche Telekom, Orange, or OpenStack scenario:

Table 3-11: Scenario Parameters for Huawei, Deutsche Telekom, Orange, OpenStack

Scenario Parameter	Collected During Configuration at Vendor	Format
Subnet ID	ID of subnet for VPC created at cloud vendor	String
Security Group ID	ID of security group created at cloud vendor	String
OS Image URI	ID of OS image created at cloud vendor	String

4. Create the scenario: click on Instantiate scenario.
5. Create an API key and add it to the scenario; see section 3.3.4.3, “Creating API Key for Cloud Hook to Use”, on page 23.
6. **Save the API key** to use when configuring the cloud bursting hook
7. We recommend adding quotas on resource usage, and alerts when those quotas are reached; see section 3.3.4.6, “Adding Quotas and Alerts”, on page 25.

3.3.4.1 Adding Hostname Prefix

The hostname prefix is the base name of each burst node. For example, if you choose "cloudnode", burst nodes are named "cloudnode1", "cloudnode2", "cloudnode3", etc.

Format: string

3.3.4.2 Setting Idle Time

The Idle before unburst parameter specifies the minimum time that a cloud node can be idle before it is unburst.

Default idle time is 180 seconds. We recommend making the idle time more than double the PBS scheduler cycle time.

Format: integer seconds

3.3.4.3 Creating API Key for Cloud Hook to Use

The cloud hook uses an API as a unique identifier for each scenario. You cannot use the same API key for more than one scenario. If you lose an API key, you can generate a new one. Later, you will put this API key in the cloud bursting hook configuration file so that the hook can identify the correct scenario.

The default lifetime of an API key is one year. You can have multiple keys for a scenario; this is to allow overlap near the expiration date. You can only list one API key per scenario in the cloud hook.

You can create an API key only for an existing scenario.

Generate and save the key using the following steps:

1. Log in to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Click *Add token* located at the bottom of the web page.
6. For Name, enter a name for the API key. Format: lowercase alphabetic + numeric
7. For Expiration date, specify the expiration date. Format: MM/DD/YYYY
8. Generate the API key by clicking Add Token.

   PBS Cloud generates the API key, and displays it only once.
9. **Copy and save the API key** so that you can paste it into the cloud hook configuration file later.
10. Click Close.

3.3.4.4 Temporarily Adding Public IP for Debugging

You can add a public IP address to the bursting scenario to make it easier to debug, then remove the IP address when you have your scenario working. To add a public IP address to the scenario:

1. Log in to PBS Cloud.
2. Click on Cloud.
3. Under Infrastructure, click on Bursting.
4. Click the name of the bursting scenario.
5. Click on the Customization tab
6. Click Edit

7. Check the checkbox next to Enable Public IP Address.
8. Click Save.

3.3.4.5 Managing Instances

When you create a scenario, choose the instance types that you want job submitters to be able to use with that scenario. The instance type determines which hardware is used to burst the cloud node. The available instance types depend on which cloud provider you use.

For each scenario, you can use either all non-preemptable instance types, or all preemptable instance types. Do not mix the two. See section 3.3.5, "Using Spot or Preemptable Pricing", on page 28.

For information about spot and preemptable instance types, see section 3.3.5, "Using Spot or Preemptable Pricing", on page 28.

3.3.4.5.i Cloud Provider Instance Types

The instance type (also called shape, machine type or flavor) determines the hardware of the host computer used for your cloud nodes. Each instance type offers different compute, memory, and storage capabilities. The following table lists some instance types by cloud provider:

Table 3-12: Cloud Provider Instance Types

Provider	Classification System	Example Instance Types
Azure	Sizes for virtual machines in Azure and High performance compute VM sizes	Standard_DS1_v2, Standard_D2s_v3, Standard_NC6 Standard_H16r, Standard_H16mr (InfiniBand)
AWS	Amazon EC2 Instance Types	t2.medium, r4.large, p3.2xlarge
GCP	Machine Types	n1-standard-8, n1-highmem-2, n1-highcpu-64
Oracle	VM Shapes and GPU Instances	VM.Standard1.1, VM.DenseIO1.16, VM.GPU3.1
Orange Cloud	Instance Family	s1.medium, s3.large.4, cc3.large.4
HUAWEI Cloud	ECS Types	s2.small.1, s2.medium.4, s2.xlarge.2
OTC	ECS Types	s1.medium, c1.large, m1.xlarge

3.3.4.5.ii Steps to Choose Instance Types

1. Log in to to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Click *Edit Instances*.
6. Choose the instance types you want available for the bursting scenario by clicking the Enabled checkbox located to the far right of the instance type name.
7. Click Save. You will see a list of the enabled instance types.

3.3.4.5.iii Managing Hyperthreading for an Instance Type

You can choose whether or not to enable hyperthreading via a checkbox next to the instance type. By default, PBS cloud enables hyperthreading. The vendor reports hyperthreading to PBS Cloud, and PBS Cloud reports hyperthreading to the cloud bursting hook.

If you turn off hyperthreading for an instance type, PBS Cloud disables hyperthreading in the cloud node, and reports the the number of available cores to the hook, so that job requests are aligned with core availability.

If you leave hyperthreading on, hyperthreading is enabled, and PBS Cloud reports the number of threads to the cloud bursting hook. In this case, job requests are aligned with thread availability.

3.3.4.6 Adding Quotas and Alerts

For each scenario, you can optionally specify quotas on resource usage at any point in time, so that when PBS Cloud hits a quota, it stops bursting nodes until usage drops back down. Use quotas to prevent huge jumps in expenditure.

For each quota, you can set an associated alert so that the web interface displays a notification when the quota is reached.

You can add quotas and alerts only to an existing scenario.

You can set quotas on the following:

- Number of CPUs in use
- Amount of memory in use
- Number of burst nodes

To set each quota and associated alert:

1. Log in to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Click Add Quota.
6. For Resource, choose a resource from the menu.
7. Click Add Quota.
8. For Limit, provide a limit for the resource.
9. To add an alert for this quota, click Add.
10. Provide an alert value.

3.3.4.7 Adding Tags (Labels) to Scenario

You can optionally add tags (labels) to burst nodes in order to categorize them, for example by purpose, owner, or environment. You can add multiple tags.

1. Log in to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Click the Cloud tab, then the global edit pen.
6. The Tags box appears.
1. To add a label to that will be applied to burst nodes, enter a key-value pair, followed by <return>. Format:

   <key>:<value><return>

   The key and the value can contain alphanumeric, dash (-) and an underscore (_). The maximum length for the key is 35 characters, and the maximum length for the value is 42 characters.

The <return> is required.

When you add the tag, it appears within its own bubble:

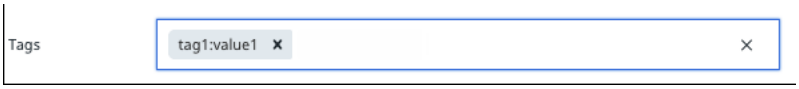


Figure 3-1:Key-Value Tag

Repeat the previous step to add more tags:



Figure 3-2:Additional Tags

2. Click Save.

3.3.4.8 Adding SSH Key for Access to Burst Nodes

You typically add the administrator SSH key so that if `/home` fails to mount during node bursting, you can log in and debug the problem. For example you need an SSH key when, `yum update` turns SELinux back on, and /home won't mount. Public SSH keys in a scenario are copied to each burst node for secure connectivity.

You can add SSH keys for only those users allowed to submit jobs to these burst nodes, although you have to make those users' home directories available on the burst nodes so that PBS Professional can use them for jobs.

To add a public SSH key for access to burst nodes:

1. Log in to the PBS Professional server host.
2. Copy the public SSH key for the user; public key files are usually stored in `/.ssh` in the user's home directory.
3. In the PBS Cloud interface:
   a. Click Add; PBS Cloud displays an editable box
   b. In Public SSH keys, paste the public SSH key.

To remove public SSH keys so that users do not have access to burst nodes, click the "x" located next to the SSH key box.

3.3.4.9 Specifying the Cloud Node Startup Script

You can optionally use the cloud-init tool to run a cloud node startup script when each node is burst, to automate node configuration tasks; see Chapter 6, "The Cloud Node Startup Script", on page 107. The startup script can be located anywhere that your PBS Cloud web interface can browse to. Once a script has been added to a scenario, the script is stored in the PBS Cloud database.

Note that you may not need a startup script. If you configure the VM that you use to create your OS image to have everything you need to run jobs, you do not need a startup script.

3.3.4.9.i Startup Script Prerequisites

- The startup script must run using a shell or language available in the freshly burst node. For example, if you have `bash` and Python available, your script can use `bash`, or it could use a `bash` script to launch a Python script.
- On Windows cloud nodes, use a PowerShell startup script. Enclose the content of the PowerShell script in <powershell> and </powershell>. Refer to Microsoft documentation for more information about PowerShell.
- The startup script can have any name.

3.3.4.9.ii Steps to Add Startup Script to Scenario

You can add and edit the cloud node startup script for this scenario from the web interface:

1. Log in to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Click the edit pen.
6. Click the Browse button, and browse to the file you want
7. Optionally edit the file as needed
8. Click Save.

3.3.4.10 Example of Creating a Scenario

Example 3-1: Create an Azure cloud bursting scenario; see section 5.2.8, “Collect Information for an Azure Cloud Bursting Scenario”, on page 63

We show some recommended options:

- For Name, give the scenario a short name, all in lowercase, e.g. hyperburst
- Select Managed Disks
- For node debugging, select Public IP; you can disable this once your scenario is working properly
- We recommend using the Standard_A4 instance type for minimal cost
- Optionally provide a startup script; for a sample script, see section 6.2.4, “Example cloud-init Startup Script for Linux”, on page 110
- Create an API key and store it safely. It is lost once you close the window, although you can simply create another if you lose it

3.3.4.11 Editing a Bursting Scenario

You can use the web interface to PBS Cloud to edit a scenario, and specify bursting scenario elements, including scenario name, description, domain name, OS image, and VPC details.

1. Before you edit a bursting scenario, stop the bursting process and make sure no jobs are running on the burst nodes; see section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
2. Edit the bursting scenario:
   a. Log in to PBS Cloud.
   b. Click Cloud.
   c. Under Infrastructure, click Bursting.
   d. Select the name of the bursting scenario.
   e. To modify the scenario, click the edit pen.

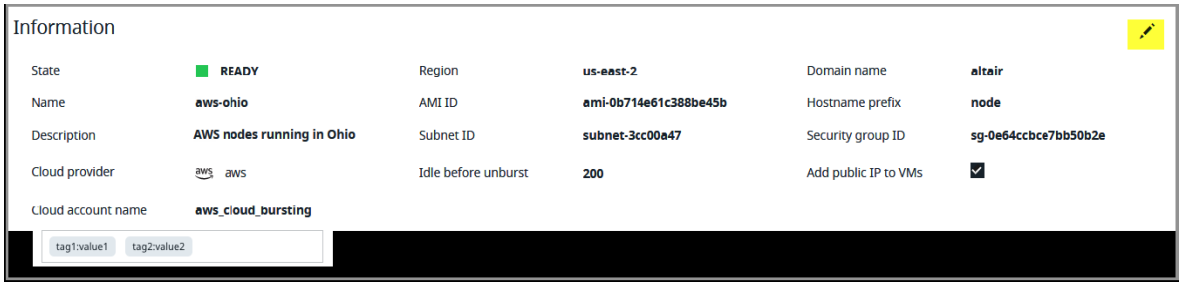


Figure 3-3:Edit Scenario Details

3. After you edit the scenario, enable it; see section 7.1.3, “Enabling or Disabling a Bursting Scenario in PBS Cloud”, on page 116.

3.3.5 Using Spot or Preemptable Pricing

3.3.5.1 When to Use Spot or Preemptable Instances

Preemptable instances offer spare compute capacity available in the cloud at steep discounts compared to on-demand instances. Spot instances are preemptable. Drawbacks:

- Spot and preemptable instances can be interrupted with two minutes' notice when the provider needs the capacity back. AWS, Azure, and GCP can preempt your spot instance when the vendor needs the resource.
- AWS and Azure can interrupt your spot instance when the spot price exceeds your maximum price, when the demand for spot instances rises, or when the supply of spot instances decreases.
- Google Cloud Platform (GCP) will kill your instance if it has been running for 24 hours.

We do not recommend running critical or long-running jobs in spot instances, as jobs may be killed when spot instances are preempted.

You can take advantage of preemptable and spot instances by using them for shorter jobs, and jobs that can be preempted.

3.3.5.2 How Spot and Preemptable Instances Work

- Dedicated on-demand: the instance is guaranteed to be available, but does not need longer-term commitments or up-front payments. You can increase or decrease your compute capacity depending on the demands of your application. You pay only for what you use. Fixed per hour or per second price depending on the instance type.
- Preemptable: vendor can preempt instance with about 2 minutes' notice; You pay a fixed discount. Instance is killed after 24 hours. GCP uses this model.
- Spot: price is set by vendor, based on demand, varies constantly. You bid a max. You pay actual spot price. If current price goes over your bid price, all your instances are preempted. If the vendor needs them, they are preempted. Cost savings of up to 90%, but most volatile. AWS and Azure use this model.

3.3.5.3 Bidding Spot Pricing

For AWS and Azure, you can bid a spot price. Specify the maximum price that you are willing to pay to continue using the spot instance. Your bid should be greater than 0 and less than the current on-demand price.

3.3.5.4 Preemptable Pricing

For GCP, the price for preemptable instances is fixed.

3.3.5.5 Specifying Spot Pricing

The scenario you use for spot pricing must already exist, must contain only preemptable instances, and must have preemptable set to *true* in the cloud hook.

1. Configure the cloud bursting hook with a scenario for this instance type. Make sure that the scenario contains only preemptable instance types. Make sure the instance type is preemptable. In the cloud bursting hook, the scenario must include:

   ```
   "preemptable": true
   ```

   See section 4.2.7, “Creating a Scenario for a Preemptable or Spot Instance”, on page 39.
2. Log in to PBS Cloud.
3. Click the Cloud tab.
4. Under Infrastructure, click Bursting.
5. Select a bursting scenario by clicking on its name.
6. Go to the instance table.

PBS Cloud displays a table of instance types you can enable for spot pricing:

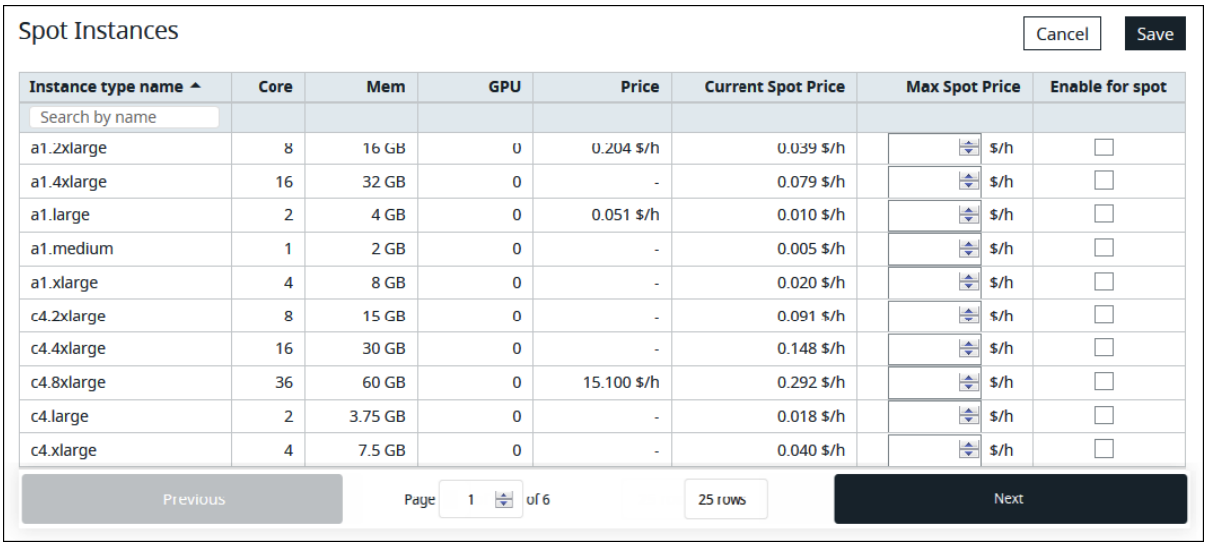


Figure 3-4:Spot Instances

7. Enable spot pricing for an instance type by checking the *Enable for spot* box next to the instance name.
8. For AWS and Azure, bid the spot price your site is willing to pay by entering it in the *Max Spot Price* box.
9. Disable spot pricing for an instance type by un-checking the *Enable for spot* box.
10. For AWS and Azure, enter a maximum spot price that you are willing to pay for the instance type.

3.3.5.6 Example of Choosing Instances for Spot Pricing

Two instance types are selected for spot pricing:

Instance type name	Core	Mem	GPU	Price	Current Spot Price	Max Spot Price	Enable for spot
Search by name							
a1.2xlarge	8	16 GB	0	0.204 $/h	0.039 $/h	0.045 $/h	☑
a1.4xlarge	16	32 GB	0	-	0.079 $/h	$/h	☐
a1.large	2	4 GB	0	0.051 $/h	0.010 $/h	$/h	☐
a1.medium	1	2 GB	0	-	0.005 $/h	0.01 $/h	☑
a1.xlarge	4	8 GB	0	-	0.020 $/h	$/h	☐
c4.2xlarge	8	15 GB	0	-	0.091 $/h	$/h	☐
c4.4xlarge	16	30 GB	0	-	0.148 $/h	$/h	☐
c4.8xlarge	36	60 GB	0	15.100 $/h	0.292 $/h	$/h	☐
c4.large	2	3.75 GB	0	-	0.018 $/h	$/h	☐
c4.xlarge	4	7.5 GB	0	-	0.040 $/h	$/h	☐

Figure 3-5:Spot Instances

3.3.6 Manage Licenses

Make sure that the number of cloud nodes plus the number of on premise nodes does not exceed the number of PBSProNodes licenses for your PBS complex. Set the value of resources_available.cloud_max_instances at the PBS server to be the number of PBSProNodes licenses minus the number of on premise nodes.

3.4 Testing Cloud Bursting

We recommend testing each scenario before using it in the cloud bursting hook. After you can manually burst a working cloud node for that scenario, add it to the cloud bursting hook and test that the hook can run jobs on the cloud nodes.

3.4.1 Test Each Scenario using Manual Bursting

3.4.1.1 Troubleshooting Prerequisites

To be able to troubleshoot cloud nodes, make sure the scenario has the following:

- The Add Public IP to VMs scenario option is enabled; see section 3.3.4.4, "Temporarily Adding Public IP for Debugging", on page 23
- The SSH keys parameter has an administrator SSH key; see section 3.3.4.8, "Adding SSH Key for Access to Burst Nodes", on page 26
- You have the corresponding private key
- Port 22 in the vendor firewall has to be open (already covered when you were configuring vendor components)

3.4.1.2 Testing and Refining a Scenario

Manually burst a single cloud node for that scenario and test its initial configuration, by following the steps in section 6.2.5, "Developing the Startup Script", on page 111.

The typical testing cycle is burst a node, check it, unburst it, modify the cloud-init script, and repeat.

3.4.1.3 Disabling Public IP Address

Once your scenario is working, you can disable its public IP address:

1. Log in to PBS Cloud.
2. Click Cloud.
3. Under Infrastructure, click Bursting.
4. Click the name of the bursting scenario.
5. Disable the public IP address.
6. Click Save.

4 Configuring the Cloud Bursting Hook

4.1 The Cloud Bursting Hook

When you install the PBS Cloud module, the installer creates and imports the cloud bursting hook, which is named cloud_hook. PBS Cloud uses a single cloud bursting hook to handle all cloud bursting. The hook comes with a default configuration file. You configure the cloud bursting hook via its configuration file.

The cloud bursting hook runs at queuejob and periodic events.

4.1.1 Default Cloud Bursting Hook Configuration File

```
{
    "pclm_server": "https://<hostname or IP address of PBS Cloud module>:9980/pbspro-cloud/",
    "_comment_pclm_server_example": "e.g. http://control_server.mydomain:9980/pbspro-cloud/",
    "use_node_hour_license": false,
    "_comment_node_hour": "Node Hour License: True for Control, False for PBS Pro",
    "pclm_no_check_ssl_certificate": false,
    "cloud_min_instances": 1,
    "resources":["ncpus", "mem", "ngpus"],
    "cloud_driver": "PclmDriver",
    "scenario":{
        "<scenario name>":{
            "api_key": "<API key>",
            "cloud_default_image": "<default cloud image for this scenario>",
            "cloud_max_instances": 10,
            "max_vms_for_infiniband_scaleset" : 100,
            "__comment__infiniband": "Use the above option for Azure infiniband scenarios only.",
            "max_nodes_per_burst":50,
            "cloud_node_wait_timeout":180,
            "preemptable": false,
            "__comment__preemptable": "Scenarios are preemptable or on-demand, not mixed.",
            "__comment__preemptable_example": "Only set to true for supported clouds and scenarios
    with spot/preemptable instances selected."
        },
        "<additional scenario name>":{
            "api_key": "<API key>",
            "cloud_default_image": "<default cloud image for this scenario>",
            "cloud_max_instances": 20,
            "max_nodes_per_burst":50,
            "cloud_node_wait_timeout":180,
            "preemptable": false
        }
    }
}
```

4.2 Configuring the Cloud Bursting Hook

4.2.1 Main Configuration Parameters in Cloud Hook

The main section of the cloud bursting hook configuration file contains the following parameters:

pclm_server

Endpoint for accessing PBS Cloud.

Format: either of these:

http://<PBS Cloud hostname>:<port>/pbspro-cloud/

http://<PBS Cloud IP address>:<port>/pbspro-cloud/

Default port: *9980*

cloud_min_instances

Required. Minimum number of instances to be present in the cloud at any time. Does not apply during startup; cloud nodes are not immediately burst on startup. This is the minimum number that are maintained after they are initially burst on demand.

This value is overridden by the value of the cloud_min_instances resource set at the PBS server.

resources

Resources to be considered for calculating the number of nodes to burst. Resource names must be in quotes. Use one of the following strings:

- ["ncpus", "mem", "ngpus"]
- ["ncpus", "mem"]

cloud_driver

Cloud driver used by the cloud bursting hook.

Currently, the only value supported is "PclmDriver". DO NOT change this value.

scenario

Container for bursting scenarios. You can have one scenario for each cloud provider or multiple scenarios for a cloud provider or both. A scenario can contain either all non-preemptable instances or all preemptable instances.

Each bursting scenario must have its own API key.

4.2.2 Scenario Configuration Parameters in Cloud Hook

api_key

API key you generated for a bursting scenario via the PBS Cloud user interface. Each scenario definition in the cloud bursting hook configuration file can have only one API key. You can use the API key for only one scenario; each API key can appear only once in the cloud hook configuration file, and in only one PBS Cloud scenario

cloud_default_image

Default OS image to use when bursting a cloud node. This is the OS image you create via the vendor interface. Overridden when the OS image is provided at job submission via the `qsub` command.

cloud_max_instances

Maximum number of instances that can be made available in the cloud for this scenario. Required.

Must be greater than 0.

Format: Integer

max_vms_for_infiniband_scaleset (Optional)

Limit on number of nodes on a single InfiniBand switch. Set by the vendor. You can negotiate with the vendor to increase this limit; if you do, update this parameter.

Supported on Azure only.

Default: 100

max_nodes_per_burst

Maximum number of nodes allowed to burst in a single hook cycle. Maximum number of cloud node licenses to renew per hook cycle.

cloud_node_wait_timeout

Maximum time to wait for freshly burst node to become usable. Minimum value: 180 seconds. You can set this to a higher value, but not lower.

Default: 180 seconds.

pclm_no_check_ssl_certificate

Specifies whether or not the cloud bursting hook checks that the PCLM platform has an SSL certificate.

If you set this to True, you need to set up an SSL certificate for the PCLM platform. Then you can use https:// for the endpoint for PBS Cloud instead of http://.

Default: False; the hook does not check for an SSL certificate

preemptable

Specifies whether this scenario supports preemptable or spot instances.

Default: False

use_node_hour_license

This parameter must be set to *false* for PBS Cloud.

Default: False

4.2.3 Steps to Configure Cloud Bursting Hook

To summarize the process, you export the cloud bursting hook configuration file, edit it for your site and add scenarios, and import it into the cloud bursting hook. Then you set the hook's frequency and alarm timeout, and enable the hook. You can create as many scenarios as you need.

1. Log in to the PBS server host as root
2. Export the cloud bursting hook configuration to a file:

   ```
   qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
   ```

3. Edit the cloud hook configuration file.
4. Set pclm_server to the endpoint for PBS Cloud:

   Format is either of these:

   http://<PBS Cloud hostname>:<port>/pbspro-cloud/
   http://<PBS Cloud IP address>:<port>/pbspro-cloud/

   Default port: *9980*
5. Set the value of cloud_min_instances to the minimum number of instances to be present in the cloud at any time. Required for bursting.
6. Set resources to a comma-separated list of resources that are to be considered for calculating the number of nodes to burst. Resource names must be in quotes. Use one of the following strings:
   - ["ncpus", "mem", "ngpus"]
   - ["ncpus", "mem"]
7. Create each scenario; see section 4.2.4, "Defining a Scenario in the Cloud Bursting Hook Configuration File", on page 37.
8. Re-import the hook configuration file:

   ```
   qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
   ```

9. The default cloud bursting hook frequency is 2 minutes (120 seconds). Optionally set the frequency; the format is integer seconds:

   ```
   qmgr -c "set hook cloud_hook freq=<number of seconds>"
   ```

10. The default cloud bursting hook alarm timeout is 10 minutes (600 seconds). We recommend setting this to less than 20 minutes (1200 seconds). Consider the following factors:

    - Time required to burst nodes in the cloud
    - Time required to unburst nodes in the cloud
    - Number of cloud queues

    Set the alarm timeout in integer seconds:

    ```
    qmgr -c "set hook cloud_hook alarm=<number of seconds>"
    ```

11. The cloud bursting hook is disabled by default. Enable the cloud bursting hook:

    ```
    qmgr -c "set hook cloud_hook enabled=True"
    ```

4.2.4 Defining a Scenario in the Cloud Bursting Hook Configuration File

When you define a scenario in the cloud bursting hook configuration file, you are telling the cloud bursting hook about a scenario that you have already created using the PBS Cloud interface. The cloud bursting hook uses the API key that you generated using the PBS Cloud interface to identify the correct scenario to burst.

4.2.4.1 Prerequisites for Defining a Scenario in Hook Configuration File

- The cloud queue for this scenario must exist and be configured for this scenario. See section 6.1.3, “Create and Configure Cloud Queues”, on page 141.
- The PBS Cloud scenario must exist and have an API key. See section 3.3.4, “Create a Bursting Scenario”, on page 19.

4.2.4.2 Steps to Define a Scenario in Hook Configuration File

Define each scenario in the "scenario" section, under its own name. You can add as many scenarios as you want.

- Set the value of api_key to the API key you generated for this scenario using the PBS Cloud interface. You can use the API key for only one scenario; each API key can appear only once in the cloud hook configuration file, and in only one PBS Cloud scenario.
- Set the value of cloud_default_image to the OS image identifier (name or ID; see vendor instructions) that should be used for bursting. This is the OS image you created using the vendor interface. If a job submitter specifies an OS image, that overrides the default.
- Set the value of cloud_max_instances to the maximum number of instances that can be made available in the cloud. Required. Must be greater than 0.

  Format: integer seconds
- Optionally, set the value of max_vms_for_infiniband_scaleset to define the maximum number of nodes allowed on a single InfiniBand switch.

This value should match the value of Maximum number of VMs inside a ScaleSet as specified in the Azure bursting scenario. If you are not using InfiniBand, then you can eliminate this parameter. Supported by Azure only.

- Set the value of max_nodes_per_burst to the maximum number of nodes allowed to burst in a single hook cycle.
- Set the value of cloud_node_wait_timeout to the maximum time to wait for freshly burst node to become usable.

  Minimum value: 180 seconds. You can set this to a higher value, but not lower.

  Default: 180 seconds
- Set preemptable to:
  - True: the bursting scenario supports preemptable or spot instances and cloud jobs can be preempted.
  - False: jobs that are run in the cloud should not be preempted.

  Default: *False*

4.2.5 Modifying the Cloud Bursting Hook Configuration File

When you modify the cloud bursting hook configuration file, the hook uses the new configuration information the next time it runs.

1. If you are modifying a scenario, disable bursting for the scenario. See section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
2. Log in to the PBS server host as root
3. Export the cloud bursting hook configuration to a file:

   ```
   qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
   ```
4. Edit the cloud hook configuration file as needed.
5. To add or modify a scenario, follow the steps in section 4.2.4, “Defining a Scenario in the Cloud Bursting Hook Configuration File”, on page 37.
6. Re-import the hook configuration file:

   ```
   qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
   ```
7. If you modified a scenario, re-enable bursting for the scenario; see section 7.1.5, “Re-enabling Bursting for a Scenario and Queue”, on page 117.

4.2.6 Adding or Changing a Scenario in the Cloud Bursting Hook Configuration File

1. Disable bursting for the scenario. See section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
2. Log in to the PBS server host as root
3. Export the cloud bursting hook configuration to a file:

   ```
   qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
   ```
4. To add or modify a scenario, follow the steps in section 4.2.4, “Defining a Scenario in the Cloud Bursting Hook Configuration File”, on page 37.
5. Re-import the hook configuration file:

   ```
   qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
   ```
6. Re-enable bursting for the scenario; see section 7.1.5, “Re-enabling Bursting for a Scenario and Queue”, on page 117.

4.2.7 Creating a Scenario for a Preemptable or Spot Instance

- If you are changing an existing scenario:
  - Disable bursting for the scenario. See section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
  - Make sure the PBS Cloud scenario exists and has only preemptable instance types. If the scenario exists but has non-preemptable instance types:
    1. Create a new PBS Cloud scenario for the instance type
    2. Create and configure a cloud queue for the new scenario
- If the PBS Cloud scenario doesn't exist, create it; see section 3.3.4, “Create a Bursting Scenario”, on page 19

1. Log in to the PBS server host as root
2. Export the cloud bursting hook configuration to a file:

   ```
   qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
   ```

3. To add or modify a scenario, follow the steps in section 4.2.4, “Defining a Scenario in the Cloud Bursting Hook Configuration File”, on page 37.
4. Make sure that the preemptable parameter is set to *true* for the scenario:

   ```
   "preemptable": true
   ```

5. Re-import the hook configuration file:

   ```
   qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
   ```

6. If you disabled bursting for the scenario, re-enable bursting; see section 7.1.5, “Re-enabling Bursting for a Scenario and Queue”, on page 117.

4.2.8 Deleting a Scenario from the Cloud Bursting Hook Configuration File

1. Disable bursting for the scenario. See section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
2. Log in to the PBS server host as root
3. Export the cloud bursting hook configuration to a file:

   ```
   qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
   ```

4. Edit the configuration file: delete the scenario from the "scenario" section.
5. Re-import the hook configuration file:

   ```
   qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
   ```

6. Re-enable bursting for the scenario; see section 7.1.5, “Re-enabling Bursting for a Scenario and Queue”, on page 117.

4.2.9 Changing PBS Cloud Host or Port

If you will move the PBS Cloud module to a different host or port:

1. Disable bursting for each scenario. See section 7.1.4, “Disabling Bursting for a Scenario and Queue”, on page 116.
2. Log in to the PBS server host as root

3. Export the cloud bursting hook configuration to a file:

```
qmgr -c "export hook cloud_hook application/x-config default" > cloud_config.json
```

4. Edit the cloud hook configuration file as needed.
5. Set pclm_server to the new endpoint for PBS Cloud:

```
http://<IP address or hostname of the PBS Cloud host>:<port number>/pbspro-cloud/
```

6. Re-import the hook configuration file:

```
qmgr -c "import hook cloud_hook application/x-config default cloud_config.json"
```

7. Re-enable bursting for each scenario; see section 7.1.5, “Re-enabling Bursting for a Scenario and Queue”, on page 117.

4.3 Testing Automated Cloud Bursting

4.3.1 Prerequisites for Testing Cloud Bursting Hook

- You have installed PBS Professional and PBS Cloud, configured PBS Professional, created a cloud administrator account at your cloud provider and added that account to PBS Cloud, created and configured your cloud provider components including an OS image, and configured PBS Cloud. These steps are outlined in section 3.1, “Overview of Configuring PBS Cloud”, on page 11.
- Make sure the cloud bursting hook is enabled.
- You should have at least one PBS Cloud scenario to test.
- This scenario must be enabled in PBS Cloud; see section 7.1.3, “Enabling or Disabling a Bursting Scenario in PBS Cloud”, on page 116
- You have created a cloud queue for this scenario; see section 6.1.3, “Create and Configure Cloud Queues”, on page 141
- Make sure that the scenarios in the hook configuration file match the scenarios in PBS Cloud, especially whether or not a scenario is preemptable

4.3.2 Steps to Test Automated Cloud Bursting

1. Log into the PBS server host as root
- Add the scenario you are testing to the cloud bursting hook; see section 4.2.6, “Adding or Changing a Scenario in the Cloud Bursting Hook Configuration File”, on page 38

2. Enable the cloud bursting hook:

```
qmgr -c "set hook cloud_hook enabled=True"
```

3. Specify that all log events should be captured in the PBS server logs:

```
qmgr -c "set server log_events=2047"
```

1. Submit jobs to the cloud queue for this scenario:

```
qsub -l select=1:ncpus=4 -q <scenario queue> TestJobScript.sh
qsub -l select=1:ncpus=4 -q <scenario queue> TestJobScript.sh
```

2. Check the status of the jobs:

```
qstat -s
```

3. Tail the PBS Professional server logs:

   tail -f PBS_HOME/server_logs/<current PBS server log file>

4. Check the current log file to verify that the cloud bursting hook is started. Search for the name of the cloud bursting hook:

   <PBS server>@<PBS server host>;Hook;<cloud_hook>;started

5. Log into PBS Cloud and go to your burst scenario. You should see the initiation of the workflow that is triggered by the bursting hook. The workflow should automatically start within a couple of minutes.

6. List the nodes known to PBS Professional in order to verify that the cloud bursting hook has burst cloud nodes:

   pbsnodes -av

7. Once a node is burst, jobs should start running. Check the status of the jobs:

   qstat -s

8. You should see that any nodes that were burst for the test are unburst after the Idle Before Unburst period has elapsed.

9. Go back to your normal server log levels. Reset the server log_events attribute to its previous value:

   qmgr -c "set server log_events=<previous value>"

5 Using Cloud Provider Services

5.1 Configuring Amazon Web Service Cloud Bursting

5.1.1 Types of Amazon Accounts

Amazon has two kinds of accounts: owner (root user), and AWS Identity and Access Management (IAM) users. Here we outline the steps you will follow below:

1. Create and activate an AWS root user account
2. Use the root user account to create an AWS IAM account and give the account administrator permissions.
3. Use the AWS IAM administrator account to do all administrative tasks. This is the account that PBS Cloud will use to manage cloud nodes.
4. Use the AWS IAM administrator account to create the AWS components required for cloud bursting.

5.1.2 Creating and Activating AWS Owner Account

Create and activate your AWS owner account. See How do I create and activate a new Amazon Web Services account?

5.1.3 Creating an AWS IAM User Account

Follow these steps to create an AWS user account and give this account administrative permissions. See Creating an IAM User in Your AWS Account. During this process, make sure you download a CSV file containing the following:

- Access key ID
- Secret access key

We will remind you of this step.

1. Log in to the AWS console.
2. Using the search box located under AWS services, enter IAM.
3. Click the IAM search result to open the Identity and Access Management dashboard.
4. In the navigation pane on the left-hand side of the web page, click Users.
5. Click Add user.
6. Enter the following information for this user:
   a. For User name, enter a name for the user.

      The name can be anything meaningful to your organization, e.g., pc_clouduser.

   b. For Access type, enable Programmatic access.

      The user requires this type of access because PBS Cloud needs to make API calls or use the AWS CLI. The AWS interface generates an access key ID and a secret access key for the user.

7. Click Next:Permissions.
8. Optional: Click Add user to group. This button may already be selected.
9. Click Create group.
10. Enter the following information to create a group, add the user to the group, and choose a permission policy for the group:
    a. For Group name, enter a group name.
       The name can be anything meaningful to your organization, e.g., pc_cloudgroup.
    b. For Policy type, enable Administrator Access.
       This policy provides full access to AWS services and resources.
11. Click Create group.
    This returns you to the Add user page and enables the new group, indicating that the user is added to the new group.
12. Click Next: Tags.
13. Click Next: Review.
14. Click Create user.
15. Click Download.csv.
16. **Download and sav**e this file in a secure location.
    PBS Cloud will use the access key ID and secret access key in this file to manage cloud nodes.
17. Click Close.
    The new user account is displayed.

5.1.4 Multi-Availability Zone Management on AWS

If you are not familiar with AWS regions, Availability Zones, VPCs or subnets, see the following AWS documentation:

- Regions, Availability Zones, and Local Zones
- VPCs and Subnets

Bursting cloud nodes in multiple Availability Zones allows an HPC complex to distribute the load across a region and take advantage of AWS Spot Instances. To use multiple Availability Zones, your virtual private cloud must have a subnet for each Availability Zone; all these subnets must belong to the same VPC.

Once these prerequisites are met, then it is as simple as providing a comma-separated list of subnets when the bursting scenario is created.

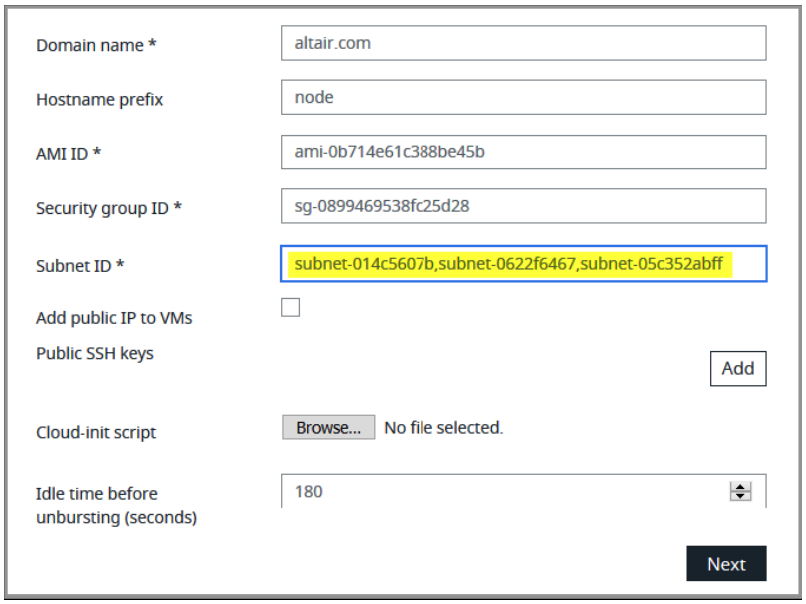


Figure 5-1:List of Subnets

PBS Cloud attempts to burst cloud nodes in the first subnet in the list. If there is no availability in that subnet, then it attempts to burst cloud nodes in the next subnet in the list and will continue until it finds a subnet where it can burst the cloud nodes or until bursting fails because no subnets have availability. The cloud bursting hook attempts to burst all requested cloud nodes in a single subnet. It does not burst cloud nodes across subnets. The cloud bursting hook follows this process each bursting cycle until it finds availability to burst the cloud nodes.

Example 5-1: 10 cloud nodes are requested for bursting.

a. The cloud bursting hook attempts to burst all 10 nodes in subnet-014c5607b.

b. If there is no availability in subnet-014c5607b, the hook attempts to burst all 10 cloud nodes in subnet-0622f6467.

c. If there is no availability in subnet-0622f6467, the hook attempts to burst all 10 cloud nodes in subnet-05c352abff.

d. If there is no availability in subnet-05c352abff, cloud bursting fails for this cycle.

5.1.5 Create a Virtual Private Cloud Network

5.1.5.1 Choose a Region

Log in to your AWS Management Console and choose a region based on the geographical location of your users. All cloud resources that are created are placed in this region.

For more information see Regions and Availability Zones. The menu for selecting a region is located at the upper right-hand corner of the AWS Console menu bar.



Figure 5-2:AWS Region

AWS documentation can be found at Getting Started with IPv4 for Amazon VPC and Working with VPCs and Subnets.

Record the region(s) you selected; you will use this later in the Region parameter in the bursting scenario.

5.1.5.2 Create a VPC

1. Log in to the AWS console.
2. Click aws located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter VPC.
4. Click the VPC search result. The VPC dashboard is opened.
5. In the menu located on the left-hand side of the web page, click Your VPCs.
6. To create a virtual private cloud, click Create VPC.
7. Enter the following to create a VPC:
   a. For Name, enter any name for the VPC.
      The name can be anything meaningful to your organization, e.g., bursting_vpc.
   b. For IPv4 CIDR block, provide an address range in CIDR notation.
   c. For IPv6 CIDR block, enable No IPv6 CIDR Block.
   d. For Tenancy, choose Default.
   e. Click Yes, Create.

5.1.5.3 Create Subnets for the VPC

Create at least one subnet for the VPC. To allow node bursting in several Availability Zones, create a subnet for each Availability Zone that you want to burst in. For more information see Multi-Availability Zone Management on AWS.

1. In the menu located on the left-hand side of the web page, click Subnets.
2. Click Create Subnet.
   a. For Name tag, enter a name for the subnet.
      The name can be anything meaningful to your organization, e.g., bursting_subnet.
   b. For VPC, choose the VPC that was previously created (e.g. bursting_vpc).
   c. For Availability Zone, choose one of the following options:
      - Choose a unique availability zone for each subnet.
      - Choose No Preference to let Amazon choose an Availability Zone for you.
   d. For IPv4 CIDR block, provide an address range in CIDR notation.
   e. Click Create.
3. Click Close.

5.1.6 Create an Internet Gateway

You can SSH into a virtual machine that is used for cloud bursting via an internet gateway. You create an internet gateway and attach it to the bursting VPC. See AWS documentation at Internet Gateways.

1. Log in to the AWS console.
2. Click aws located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter VPC.

4. Click the VPC search result to open the VPC dashboard.
5. In the menu located on the left-hand side of the web page, click Internet Gateways.
6. Click Create internet gateway.
7. Enter a value for Name tag, to be a name for the internet gateway.

   The name can be anything meaningful to your organization, e.g., bursting_gateway
8. Click Create.
9. Click Close.
10. Select the internet gateway that you just created by enabling the check box next to the name of the gateway.
11. You may need to deselect any other internet gateways that are displayed in the list. Amazon creates default resources for your selected region so a default internet gateway may already exist.
12. Click Actions > Attach to VPC.
13. Select the VPC that you created previously (e.g. bursting_vpc).
14. Click Attach.

5.1.7 Update the VPC Route Table

You add a rule to the VPC route table that allows all internet access, and associate the route table with the bursting subnet. You can find AWS documentation at Route Tables.

1. Log in to the AWS console.
2. Click aws located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter VPC.
4. Click the VPC search result to open the VPC dashboard.
5. In the menu located on the left-hand side of the web page, click Route Tables to display a list of route tables.
6. Select the route table attached to your VPC (e.g. bursting_vpc) by enabling the check box next to the name of the route table.

   The VPC column in the route table list specifies the VPC to which the route table is attached.
7. Click the Routes tab at the bottom of the web page.
8. Click Edit routes.

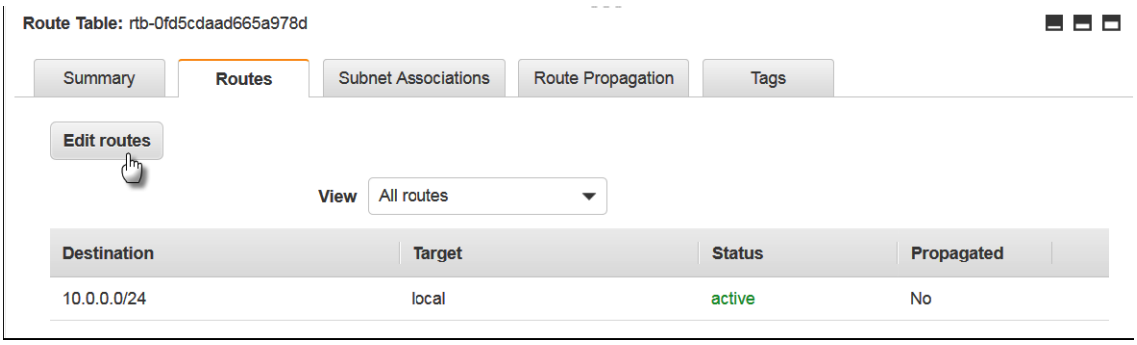


Figure 5-3:Add a Route

9. Click Add route.

10. Enter the following to add a rule that allows all traffic access to the internet gateway:
    - For Destination enter the PBS Cloud firewall IP address.
    - For Target, select Internet Gateway, then the internet gateway that you created previously (e.g. bursting_gateway).
11. Click Save routes.
12. Click Close.
13. Associate the route table to the bursting subnet:
    a. Click the Subnet Associations tab.
    b. Click Edit subnet associations.
    c. Select the subnet you created for cloud bursting from the list.
14. Click Save.

5.1.8 Add Inbound Rules to the VPC Security Group

Add inbound rules to the VPC security group so that a connection can be established with an AWS VM using SSH or RDP.

1. Log in to the AWS console.
2. Click  located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter VPC.
4. Click the VPC search result to open the VPC dashboard.
5. In the menu located on the left-hand side of the web page, under Security, click Security Groups.
6. Select the security group associated with the VPC you created for cloud bursting by enabling the check box next to its name.

   When you created the VPC, the vendor system created a default VPC security group.
7. Click the Inbound Rules tab at the bottom of the web page.
8. Click Edit rules.
9. Click Add Rule.

10. Add security rules based on your site's requirements. If you enable a public IP address for the associated scenario, there is access to these ports, but the rules here filter who is allowed that access.
    - On Linux platforms, add an inbound rule to allow SSH traffic on port 22.
    - On Windows platforms, add an inbound rule to allow RDP traffic on port 3389.
    - Add the IP address of the PBS Cloud firewall (replace what is here):

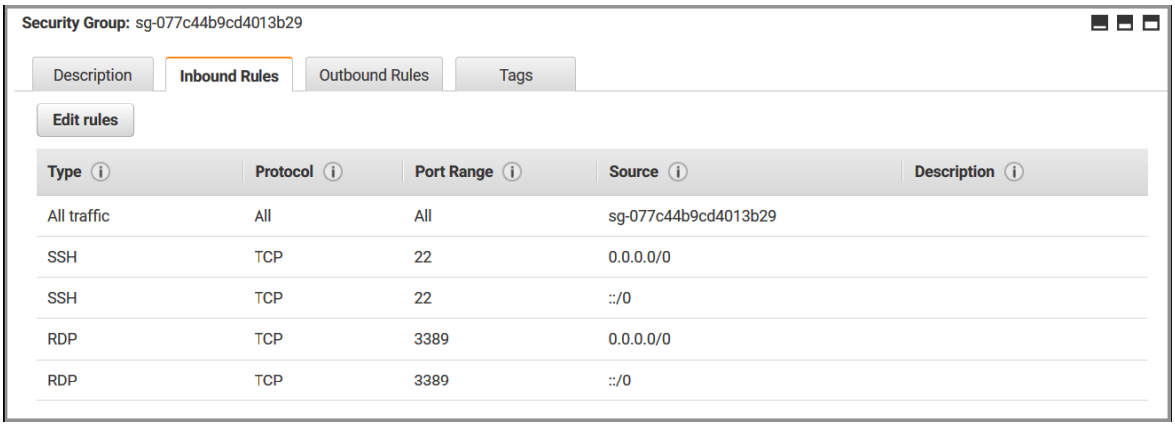


Figure 5-4:Security Rules

Warning: 0.0.0.0/0 enables all IPv4 addresses to access your instance. ::/0 enables all IPv6 address to access your instance. This is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instance.

11. Click Save rules.

5.1.9 Create a Virtual Machine

In this section you create a virtual machine in AWS Elastic Compute Cloud (EC2).

For AWS documentation, see Launch a Linux Virtual Machine and Launching a Virtual Machine with Amazon EC2.

1. Log in to the AWS console.
2. Click aws located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter EC2.
4. Click the EC2 search result to open the EC2 dashboard.
5. In the menu located on the left-hand side of the web page, click Instances.
6. Click Launch Instance.
7. In the menu located on the left-hand side of the web page, click AWS Marketplace.
8. Using the search box:
    - On Linux platforms, choose a Linux platform that is supported for the PBS MoM and press ENTER.
    - On Windows platforms, choose a Windows platform that is supported for the PBS MoM and press ENTER.
9. Locate the appropriate Amazon Machine Image (AMI) and click Select.
10. Click Continue.

11. Select an Instance Type appropriate for your site's workload, based on these criteria:
    - Number of cores
    - Amount of memory
    - Storage
    - Network performance

    Consider the nature of the applications that you plan to deploy on the instance, the number of users that you expect to use the applications, and also how you expect the load to scale in the future. Remember to also factor in the CPU and memory resources that are necessary for the operating system.
12. Click Next: Configure Instance Details.
13. Enter the following to configure instance details:
    a. For Number of instances, specify 1.
    b. For Network, choose the VPC that you previously created (e.g. bursting_vpc). The bursting subnet is populated automatically.
    c. For Auto-assign Public IP, select Enable.
14. Click Next: Add Storage.
15. Specify the storage options your site needs. We recommend enabling Delete on Termination to delete EBS volumes (attached disks) when the virtual machine is terminated.
16. Click Next: Add Tags.
17. Optional: You can add tags for the VM in key-value pairs.
18. Click Next: Configure Security Group.
19. Assign a security group to the VM. Enter the following:
    a. For Assign a security group, enable Select an existing security group.
    b. Select the security group that was automatically created for the cloud bursting VPC by enabling the check box next to its name.
20. Click Review and Launch.
21. Review the information about the VM and click Launch.
22. Create a new public/private key pair for the VM. Enter the following:
    a. Select Create a new key pair.
    b. Provide a name for the key pair.
    c. Click Download Key Pair.
    d. Download and save this file in a secure location.

    PBS Cloud will use the information in this .pem file later to SSH into the cloud node.
23. Click Launch Instances.
24. At the bottom of the web page, click View Instances.

    This displays all virtual machines that have been created.

Your virtual machine is ready when the Instance State is "running" and Status Checks are complete. The virtual machine can be accessed via its IPv4 Public IP.



Figure 5-5:Bursting Virtual Machine

5.1.10 Install a PBS MoM on the VM

5.1.10.1 Installing a PBS MoM on a Linux VM

On Linux platforms, the username for logging into the virtual machine is dependent on the Amazon Machine Image (AMI) that you used to create the virtual machine. For example, the username for a CentOS AMI is "centos". Typically you log in as "centos", using your SSH key, then switch to root:

sudo -

For information about usernames and instructions for connecting and copying files to the Linux virtual machine see Connecting to Your Linux Instance Using SSH.

To establish a connection to the VM, you need the .pem file you downloaded while creating the VM.

1. Copy the PBS Professional installer package to the virtual machine. Use scp to copy the tarball file from the PBS server host to the virtual machine. For more information, see Transferring Files to Linux Instances from Linux Using SCP.
2. Log in to your site's PBS Professional server host.
3. SSH into the virtual machine as the user "centos" using the .pem file and the IPv4 Public IP assigned to the VM:

   `ssh -i /<path to .pem file>/<.pem file> centos@<public IP address of virtual machine>`

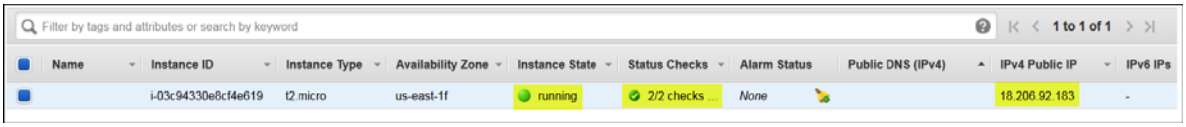


Figure 5-6:Bursting Virtual Machine

4. Switch to root:

   `sudo -i`

5. Copy the PBS Professional installation package to the VM.
6. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS Professional MoM.
7. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount `/home` in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
8. If cloud-init is not installed, install it.

5.1.10.2 Installing the PBS MoM on the Windows VM

Use an RDP client to access the virtual machine. You can establish a connection to the Windows virtual machine through the AWS EC2 console. See Connect to Your Windows Instance for more information.

You will need the .pem file downloaded while creating the VM to establish a connection.

You will copy the PBS Professional installer package to the virtual machine, and use RDP to map a local drive to get access to the installer package. For more information, see Transfer Files to Windows Instances.

1. Log in to the AWS console.
2. Click  located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter EC2.
4. Click the EC2 search result.
5. In the menu located on the left-hand side of the web page, click Instances.
6. Select the Windows virtual machine created for cloud bursting by enabling the check box next to its name.
7. At the top, click Connect.
8. Click Get Password.
9. Browse to the .pem file downloaded while creating the VM.
10. Open the .pem file.
11. Click Decrypt Password.
12. Hover over the decrypted password. You will see a copy to clipboard icon.
13. Click the copy to clipboard icon.
14. Click Download Remote Desktop File.
15. Open the file.
16. Click Connect.
17. For Password, paste the password copied to the clipboard.
18. Click OK.
19. Click Yes to connect, even if there are certificate errors.

    A connection is established with the Windows virtual machine.
20. Copy the PBS Professional installation package to the VM.
21. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS Professional MoM.
22. Configure the VM for your site's environment, for example mounting file systems, connecting to the authentication service, installing any applications, etc.

5.1.11 Create an OS Image

In the following steps, you will create an image of the virtual machine you have configured.

You can find AWS documentation at Create an AMI from an Amazon EC2 Instance.

1. Log in to the AWS console.
2. Click  located in the upper left-hand corner of the web page.

3. Using the search box located under AWS services, enter EC2.
4. Click the EC2 search result to open the EC2 dashboard.
5. In the menu located on the left-hand side of the web page, click Instances.
6. Select the virtual machine created for cloud bursting by enabling the check box next to its name.
7. At the top, click Actions > Instance State > Stop.
8. Click Yes, Stop.

   It may take some time for the virtual machine to be stopped.

   Do not proceed until the Instance State is "Stopped".
9. Click Actions > Image > Create Image.
10. For Image name, enter a name for the image.

    The name can be anything meaningful to your organization, e.g., bursting_image.

    On Windows platforms, the name of the image must contain the string "windows" (case-insensitive). For example, Windows_Server-2012-R2__RTM-English-64Bit-Base-2019.11.13.
11. Click Create Image.
12. Click the View pending image ami-xxxxxxxxxx link. The image is complete when its Status is "available".
13. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.1.12 Collect Information for an AWS Cloud Bursting Scenario

5.1.12.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about them:

Table 5-1: Scenario Parameters for Amazon Web Services (AWS)

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
AMI ID	Name of image to be burst; chosen during configuration at vendor	String
Security Group ID	Name of security group associated with VPC and VM created at vendor	String
Subnet ID	Name of security group subnet for bursting VPC created at vendor. To burst nodes in multiple Availability Zones, save a comma-separated list of subnet IDs	String

5.1.12.2 Steps to Collect Information

1. Open a browser window and log in to your AWS Management Console.
2. Click  located in the upper left-hand corner of the web page.
3. Using the search box located under AWS services, enter EC2.
4. Click the EC2 search result to open the EC2 dashboard.
5. In the menu located on the left-hand side of the web page, click AMIs.
6. Select the Amazon Machine Image (AMI) you created for cloud bursting by enabling the check box next to its name.
7. In the Details tab located at the bottom of the web page, hover over the AMI ID so that the interface displays a copy to clipboard icon.
8. Click the copy to clipboard icon.
9. **Save the image name** to use for the AMI ID scenario parameter.
10. In the menu located on the left-hand side of the web page, under NETWORK & SECURITY, click Security Groups.
11. Select the Security Group associated with the VPC and the VM by enabling the check box next to its Group ID.
12. In the Description tab located at the bottom of the web page, hover over the Group ID so that the interface displays a copy to clipboard icon.
13. Click on the copy to clipboard icon.
14. **Save the security group name** to use for the Security Group ID scenario parameter.
15. Click  located in the upper left-hand corner of the web page.
16. Using the search box located under AWS services, enter VPC.
17. Click the VPC search result to open the VPC dashboard.
18. In the menu located on the left-hand side of the web page, click Subnets.
19. Select subnets for the bursting VPC by checking the box next to their names. Subnets are required in order to make multiple availability zones work.
20. In the Description tab located at the bottom of the web page, hover over the Subnet ID so that the interface displays a copy to clipboard icon.
21. Click the copy to clipboard icon.
22. **Save the security group subnet name** to use for the Subnet ID scenario parameter. To use cloud nodes in several Availability Zones, save a comma-separated list of subnet IDs.

5.2 Configuring Microsoft Azure Cloud Bursting

5.2.1 Prerequisites

Purchase an Azure subscription, get a tenant ID, and get an Azure user account. For more information about subscriptions see What is an Azure subscription. For more information about tenants see How to get an Azure Active Directory tenant.

5.2.2 Register PBS Cloud with Azure

You can find Azure documentation at Quickstart: Register an application with the Microsoft identity platform.

As you work through this section, save the following information in a file. We will remind you about these:

Table 5-2: Account Parameters for Microsoft Azure

Account Parameter	What to Collect During Configuration at Vendor	Format
Client ID	Application ID generated when registering PBS Cloud with the Azure Active Directory	String
Secret Key	Secret Key generated during account creation at vendor	String
AD Tenant ID	Azure tenant ID generated during account creation at vendor	String
Subscription ID	Subscription ID generated during account creation at vendor	String

1. Register PBS Cloud with the Azure Active Directory and create a client secret key:
   a. Log in to to your Microsoft Azure account.
   b. Using the search box, enter app reg. You will see a list of search results.
   c. Under Services, click App registrations.
   d. Click New registration.
   e. Enter the following to register PBS Cloud with the Azure Active Directory:
      1. For Name, enter the name you will use for PBS Cloud at the vendor.

         The name can be anything meaningful to your organization, e.g., pbs_cloud
      2. For Supported account types, choose the option that best suits your organization. Click the Help me choose link for additional information about the available options.
      3. For Redirect URI, select Web and enter the URL https://<PBS Cloud host name or IP address>:<PBS Cloud port>/pc. The default PBS Cloud port is 9980.

where hostname is the hostname of the machine where the PBS Cloud web interface is installed. This is the URL that is used to log in to PBS Cloud.

f. Register the application: click Register.

Once the application registration is complete, its details are displayed, including an Application ID.

g. To get the application ID for PBS Cloud, hover over the Application (client) ID and click the copy-to-clipboard icon when it appears.

h. **Store the application ID to a file**. You will need this later when you add the vendor cloud account to PBS Cloud.

1. Create a client secret key for PBS Cloud:

a. Under, Manage, click Certificates and secrets.

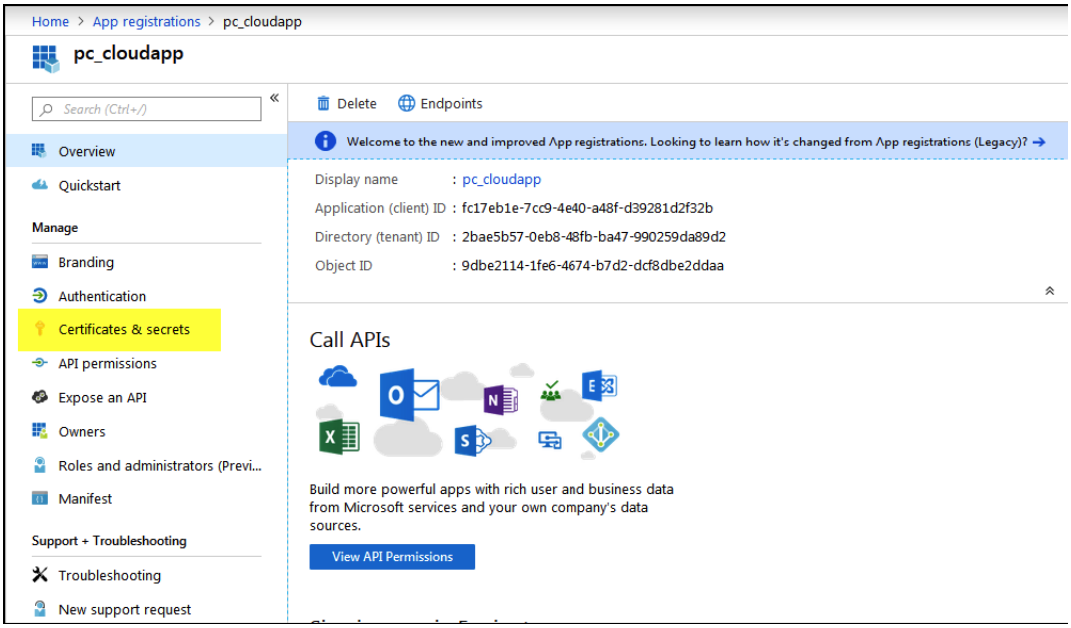


Figure 5-7:Certificates and Secrets

b. Under Client secrets, click New client secret.

c. Enter the following to add a client secret:

1. For DESCRIPTION, enter pc_client_secret.
2. For EXPIRES, select Never.

d. Click Add to generate the client secret key.

You will see the client secret key under the heading VALUE.

e. Click the copy icon next to the client secret key.

f. **Store the client secret key to a file**. The client secret key is used later to create a cloud account in PBS Cloud.

2. Get your Azure subscription ID:
   a. Using the search box, enter subscription. A list of search results is listed.
   b. Under Services, click Subscriptions.
   c. Locate and click your subscription to see details about the subscription, including its Subscription ID.
   d. Hover over the Subscription ID and click on the copy icon when it appears.
   e. **Store the Subscription ID value to a file**. You will use the Subscription ID later when you add the vendor cloud account to PBS Cloud.
3. Assign an access control role to PBS Cloud.
   a. Click Access control (IAM).

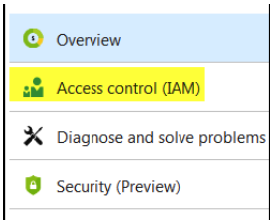


Figure 5-8:Add Access Controls

   b. Click Add.
   c. Click Add role assignment.
   d. In the Add role assignment panel, enter the following to assign a role to PBS Cloud.
      1. For Role, select Contributor.
      2. For Assign access to, select Azure AD user, group, or service principal.
      3. For Select, search for the newly registered application by entering its name, e.g., pbs_cloud.
      4. Select the application by clicking on it.
   e. Click Save.
4. Obtain your Azure tenant ID:
   a. At the top of the web page, click ?.
   b. Click Show diagnostics.

      A dialog box is displayed allowing a file called PortalDiagnostics.json to be saved.
   c. Open the file using any text editor.
   d. Search for tenantId.
   e. **Store the value of tenantId to a file**. You will use the Tenant ID later to add the vendor cloud account to PBS Cloud.

5.2.3 Create a Resource Group

Azure documentation can be found at Manage Azure resources through portal.

A resource group is container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Once the resource group is created, resources that are placed into the resource group are a virtual network, a virtual machine, and an image of the virtual machine.

1. Log in to your Microsoft Azure account.
2. Using the search box, enter resource groups. A list of search results is listed.
3. Under Services, click Resource Groups.
4. Click Add.
5. Enter the following to configure the basic settings for the resource group:
   a. For Project Details enter the following:
      - For Subscription, choose the subscription to be billed for the use of the VM.
      - For Resource group, enter a name for the resource group.

        The name can be anything meaningful to your organization, e.g., bursting_resource_group.
   b. For Resource Details enter the following:
      - For Region, select a location based on the geographical location of users.
6. Click Review + create.
7. Click Create.

It may take a moment to create the resource group. All resources (networks, virtual machines, etc.) that are created are placed within this resource group. The name of the resource group is required for creating a bursting scenario in PBS Cloud.

5.2.4 Create a Virtual Network

Azure documentation can be found at Virtual Network Documentation.

1. Log in to your Microsoft Azure account.
2. Using the search box, enter virtual networks. A list of search results is listed.
3. Under Services, click Virtual networks.
4. Click Add.
5. For Name, enter a name for the virtual network.

   The name can be anything meaningful to your organization, e.g., bursting_virtual_network
6. For Address space, enter an address range for the network using CIDR notation.
7. For Subscription, select the same subscription as was selected for the previously created resource group.
8. For Resource group, select the previously created resource group.
9. For Location, select the same geographical location as was selected for the previously created resource group.
10. For Subnet, enter the following:
    a. For Name, enter a name for the virtual machine's subnet.

       The name can be anything meaningful to your organization, e.g., bursting_subnet
    b. For Address range, enter an address range for the subnet in CIDR notation.
11. Click Create.

It may take a moment to create the virtual network. The name of the virtual network is required for creating a bursting scenario in PBS Cloud.

5.2.5 Create a Virtual Machine

You may want to view the following web page to learn about Azure Managed Disks before creating a VM. Additionally, a video is available from Microsoft that shows how to create a virtual machine: Create a Linux Virtual Machine.

1. Log in to your Microsoft Azure account.
2. Using the search box, enter virtual machines. A list of search results is listed.
3. Under Services, click Virtual machines.
4. Click Add.

Enter the following to configure the basic settings for the virtual machine:

5. For Project Details enter the following:
   a. For Subscription, choose the subscription to be billed for the use of the VM.
   b. For Resource group, choose the previously created resource group.
   c. For Virtual machine name, enter a name for the virtual machine.
      The name can be anything meaningful to your organization, e.g., bursting-vm.
   d. For Region, select the same geographical location as was selected for the previously created resource group.
   e. For Availability options, choose No infrastructure redundancy required.
   f. For Image, click the Browse all public and private images link.
   g. Using the search box:
      - On Linux platforms, enter CentOS 7 or RHEL 7 and press ENTER.
      - On Windows platforms, enter Windows and press ENTER.
   h. Locate the appropriate image and select it.
      - On Linux platforms, cloud bursting has been tested on on CentOS 7.2 - 7.6.
      - On Windows platforms, cloud bursting has been tested on Windows 10 and Windows Server 2012.
   i. For Size, click the Change size link and select a machine size appropriate for your site's workload based on:
      - the number of cores
      - the amount of memory
      - storage
      - network performance

      Consider the nature of the applications that you plan to deploy on the instance, the number of users that you expect to use the applications, and also how you expect the load to scale in the future. Remember to also factor in the CPU and memory resources that are necessary for the operating system.
   j. Click Select.
6. For Administrator Account, enter a user account :

This user will have sudo rights and will be able to connect to the VM to install the PBS MoM.

- On Linux platforms:
  - For Authentication type, enable SSH public key.
  - For Username, enter a username of a user account that exists on your site's PBS Server.
  - For SSH public key, copy the SSH public key (i.e., id_rsa.pub) of the user account and paste it.
- On Windows platforms:
  - For Username, enter a username.
  - For Password, enter a password.

7. For Inbound Port Rules, enter the following:
   a. For Public inbound ports, enable Allow selected ports.
   b. For Select inbound ports:
      - For Linux platforms, enable SSH (22).
      - For Windows platforms, enable RDH (3389).
8. Click Next.

Enter the following to configure the storage settings for the virtual machine:

9. For Disk Options, enter the following:
   a. For OS disk size, choose an appropriate disk size based on your site's needs.
   b. For OS disk type, choose one of the following options:
      - Premium SSD
      - Standard SSD
      - Standard HDD

      Choose SSD for I/O-intensive applications, where low latency and high throughput are critical. For testing, consider HDD to keep costs down, as you scale up and down quickly.
1. For Advanced, enter the following:
   a. Click Advanced.
   b. For Use managed disks, choose one of the following options:
      - Yes to use managed disks.
      - No to not use managed disks.

      Enable this feature to have Azure automatically manage the availability of disks to provide data redundancy and fault tolerance, without creating and managing storage accounts on your own. This option is recommended by Azure as it is a lot more scalable.
1. Click Next.

Enter the following to configure the networking settings for the virtual machine:

2. For Network Interface, enter the following:
   - For Virtual network, choose the virtual network previously created.
3. Click Review + create.
4. Click Create.

   It may take a few minutes for the VM to be deployed. You will use this virtual machine to create an OS image.

Once the virtual machine is deployed a message is displayed indicating success, click on Go to resource.

5.2.6 Install a PBS MoM on the VM

5.2.6.1 Install a PBS MoM on a Linux VM

1. Log in to your site's PBS server host as the user account (username and the public SSH key) provided during the creation of the VM.
2. SSH into the virtual machine using the public IP address of the VM:

   ```
   ssh IPV4PublicIP
   ```

   where IPV4PublicIP is the public IP address of the virtual machine.

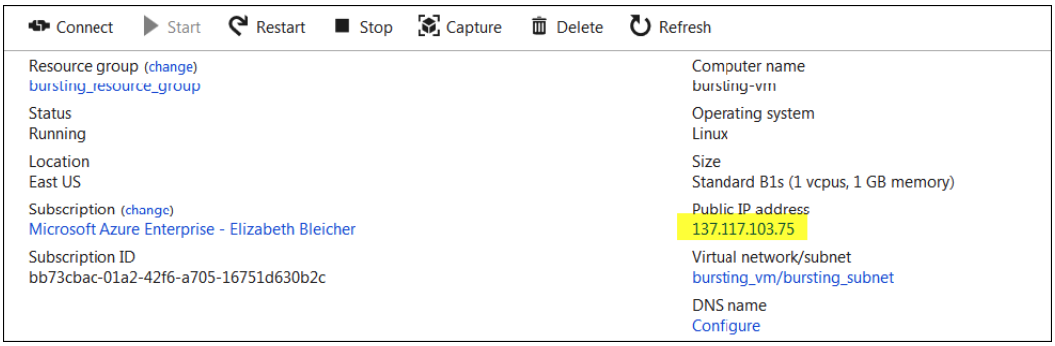


Figure 5-9:Bursting Virtual Machine

3. Enter the command:

   ```
   sudo -i
   ```

4. Copy the PBS Professional installation package to the VM. Use SCP to copy the tarball file from the PBS server host to the virtual machine. For more information, see Move files to and from a Linux VM using SCP.
5. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
6. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount /home in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
7. If cloud-init is not installed, install it.

5.2.6.2 Install a PBS MoM on a Windows VM

You will use an RDP client to access the virtual machine. A connection can be established to the Windows virtual machine through the Azure portal. For more information see How to connect and sign on to an Azure virtual machine running Windows.

You will copy the PBS Professional installer package to the virtual machine. Use RDP to map a local drive to gain access to the installer package.

1. Log in to the Azure portal.
2. Using the search box, enter virtual machines.
3. Under Services, click Virtual machines.
4. Select the Windows virtual machine created for cloud bursting by clicking its name.

5. Click Connect.
6. Click the RDP tab.
7. Click Download RDP File.
8. Open the file.
9. Click Connect.
10. Enter the password that was established for the Administrator Account when creating the virtual machine.
11. Click OK.
12. Click Yes to connect even if there are certificate errors.

    A connection is established with the Windows virtual machine.
13. Copy the PBS Professional installation package to the VM.
14. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MoM.
15. Configure the VM for your site's environment such as mounting file systems, connecting it to the authentication service, installing any applications, etc.

5.2.7 Create an OS Image

5.2.7.1 Create a Linux OS Image

Creating an OS image requires the Azure CLI. Refer to these instructions for installing the CLI: How to install the Azure CLI. We recommend installing the CLI on a Windows or Mac machine and then using the command prompt to execute the CLI commands.

Before you can create an OS image of the previously created VM, you must first SSH into the VM and deprovision it. Next you will use the Azure CLI to deallocate and generalize the VM and then create the image. Generalizing the virtual machine removes any SSH keys and DNS settings from the VM.

Follow Step 1 and Step 2 as documented in "How to create an image of a virtual machine or VHD" to create an image of the VM.

Before you can deallocate the virtual machine you may have to execute the following commands to set your subscription to be the active subscription:

```
az account list
az account set --subscription <your subscription ID>
```

You can now delete the virtual machine so that you are no longer charged for it.

5.2.7.2 Create a Windows OS Image

You will generalize the VM using Sysprep. For more information see Create a managed image of a generalized VM in Azure.

1. Log in to the Azure portal.
2. Using the search box, enter virtual machines.
3. Under Services, click Virtual machines.
4. Select the Windows virtual machine created for cloud bursting by clicking its name.
5. Click Connect.
6. Click the RDP tab.

7. Click Download RDP File.
8. Open the file.
9. Click Connect.
10. Enter the password that was established for the Administrator Account when creating the virtual machine.
11. Click OK.
12. Click Yes to connect even if there are certificate errors.
13. Open a Command Prompt window as an administrator.
14. Using Windows Explorer, navigate to the directory C:\Windows\System32\Sysprep.
15. Right-click sysprep.exe and select Run as Administrator.
16. For System Cleanup Action, choose Enter System Out-of-Box Experience (OOBE).
17. Enable the Generalize check box.
18. For Shutdown Options, choose Shutdown.
19. Click OK.
20. Once the VM is shut down, close the RDP session.
21. Navigate to the browser window where the Azure portal is open and the VM details are displayed.
22. Click Capture.
23. For name, enter a name for the image.

    The name of the image should contain the string "windows" (case insensitive). For example, Windows Server 2012 R2 Datacenter.
24. For Resource group, choose the previously created resource group.
25. For Type the virtual machine name, enter the name of the VM.
26. Click Create.
27. You can now delete the virtual machine so that you are no longer charged for it.

5.2.8 Collect Information for an Azure Cloud Bursting Scenario

5.2.8.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about each one:

Table 5-3: Scenario Parameters for Microsoft Azure

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String

Table 5-3: Scenario Parameters for Microsoft Azure

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Resource group name	Name of resource group (virtual network, virtual machine, OS image) created at vendor	String
Network name	Name of virtual network created at vendor If the network is in a different resource group from the one specified, enter it as Resource Group Name/Virtual Network Name	String
Subnetwork name	Name of virtual subnet created at vendor	String
Network security group name	Name of network security group for resource group	String
Managed Storage	Managed disk feature selected at vendor	Boolean
OS Image	If using managed disks, name of the image. If not using managed disks, Linux Source BLOB URI.	String
Maximum number of VMs inside a ScaleSet with Managed Storage and a single Placement Group	Limit selected during configuration at vendor. Default: 100	Integer

5.2.8.2 Steps to Collect Information

Open a browser window and log in to your Microsoft Azure account.

For information on virtual machine scale sets, see the following Azure article about scale sets: What are virtual machine scale sets.

A bursting scenario requires a resource group, but other scenario resources (network, subnet, network security group and the OS image) can all reside in a different resource group. However, the resource groups must be in the same geographic location for this to work.

28. In the menu located on the left-hand side of the web page, click Resource Groups.
29. Copy the name of the resource group created for cloud bursting.
30. **Save the name of the resource group to a file**. You will use this later when you create the bursting scenario.
31. In the menu located on the left-hand side of the web page, click Virtual Networks.
32. Copy the name of the virtual network created for cloud bursting.
33. **Save the name of the network to a file**. You will use this later when you create the bursting scenario.
34. Select the virtual network created for cloud bursting.

35. Click Subnets.

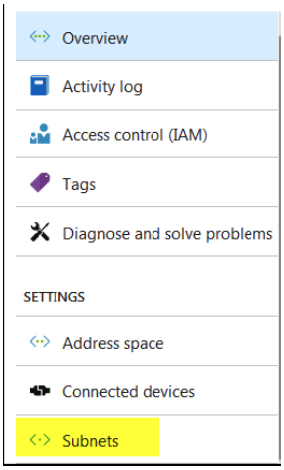


Figure 5-10:Subnet

36. For Subnet name, copy the name of the subnet created for the cloud bursting virtual network.
37. **Save the name of the subnet to a file**. You will use this later when you create the bursting scenario.
38. In the menu located on the left-hand side of the web page, click Resource Groups.
39. Select the Resource Group created for cloud bursting.
40. In the list, locate the Network Security group created for cloud bursting. The Type is Network Security group.
41. Copy the name of the network security group.
42. **Save the name of the network security group to a file**. You will use this later when you create the bursting scenario.
43. In the menu located on the left-hand side of the web page, click Resource Groups.
44. Select the Resource Group created for cloud bursting.
45. In the list, locate the image that was created for cloud bursting. The Type is Image.
46. Select the image.
47. Choose one of the following options:
    - If you chose to use managed disks when you created the VM, copy the name of the image.
    - If you did not choose to use managed disks when you created the VM, copy the Linux Source BLOB URI.
48. **Save the name of the image or the Linux Source BLOB URI to a file**. You will use this later when you create the bursting scenario.

5.3 Configure Google Cloud Platform Cloud Bursting

5.3.1 Sign Up for a GCP Account

Sign up for a GCP user account. This is different from a GCP service account. Go to the Google Account signup page.

5.3.2 Create a Project

Google Cloud Platform projects form the basis for creating, enabling, and using all Cloud Platform services including managing APIs, enabling billing, adding and removing collaborators, and managing permissions for Cloud Platform resources. GCP documentation can be found at Creating and Managing Projects.

1. Log in to the GCP console.
2. Click  located in the upper left-hand corner of the web page.
3. Click Home.
4. Click Create.
5. For Project Name, enter a name for the project.

   The name can be anything meaningful to your organization, e.g., pc_cloudproject.
6. Click Create.

   It may take a few moments to create the project.
7. Using a browser, navigate to the following URL: https://console.developers.google.com/ apis/library/compute.googleapis.com?project=PROJECTNAME

   where PROJECTNAME is the name of the project.
8. Click Enable.

5.3.3 Create a Service Account

GCP documentation can be found at Understanding Service Accounts and Compute Engine IAM Roles .

A service account is a special Google account that is used by applications to use the Google Cloud APIs. PBS Cloud will use a service account to manage cloud nodes.

1. Log in to the GCP console.
2. Click  located in the upper left-hand corner of the web page.
3. Click IAM & admin > Service accounts.
4. Click CREATE SERVICE ACCOUNT.
5. Enter the following to create a service account:
   a. For the Service account name, enter a name for the service account.

      The name can be anything meaningful to your organization, e.g., pc-service-account.
   b. Click CREATE.
   c. For the Project role, click Select a role > Compute Engine > Compute Admin. This role gives full control of all Compute Engine resources.
   d. Click CONTINUE.
   e. Under Create key (optional), click CREATE KEY.
   f. For Key type, enable JSON.
   g. Click CREATE.
   h. **Save the JSON file** in a secure location. Use the dialog box to choose a place to save it. You will need this information later when you add the provider account to PBS Cloud.

6. Make sure you have downloaded a JSON file containing the following:
   - Project ID
   - Client ID
   - Client email
   - Private key ID
   - Private key
7. Click CLOSE.
8. Click DONE.

5.3.4 Create a Virtual Private Cloud Network

GCP documentation can be found at Virtual Private Cloud Documentation and Using VPC Networks .

1. Log in to the GCP console.
2. Click  located in the upper left-hand corner of the web page.
3. Click VPC network > VPC networks.
4. Click CREATE VPC NETWORK.
5. Enter the following to create a VPC:
   a. For the Name, enter a name for the VPC.
      The name can be anything meaningful to your organization, e.g., bursting-vpc.
   b. In the Subnets section, click the Custom tab under Subnet creation mode.
   c. For Name, enter a name for the subnet.
      The name can be anything meaningful to your organization, e.g., bursting-subnet.
   d. For Region, select a Region based on the geographical location of users.
   e. For IP address range, enter an IP address range using CIDR notation
   f. For Private Google access, enable Off.
   g. Click Done.
   h. For Dynamic routing mode, enable Regional.
6. Click Create.
   Creating the VPC network may take some time. Do not proceed until the VPC is created.
7. Select the VPC by clicking on its name.
8. Click the Firewall rules tab.
9. Click CREATE FIREWALL RULE.

10. Enter the following to create a firewall rule:
    a. For Name, enter a name for the firewall rule.
       The name can be anything meaningful to your organization, e.g., ssh-all.
    b. For Direction of Traffic, enable Ingress.
    c. For Action on match, enable Allow.
    d. For Targets, select All instances in the network.
    e. For Source filter, select IP ranges.
    f. For Source IP ranges, enter the IP address of the PBS Cloud firewall
    g. For Protocols and ports, enable Specified protocols and ports.
    h. Enable tcp.
    i. Enter 22.
11. Click Create.

5.3.5 Create a Virtual Machine

GCP documentation can be found at Virtual Machine Instances and Creating and Starting a VM Instance.

1. Log in to the GCP console.
2. Click  located in the upper left-hand corner of the web page.
3. Click Compute Engine > VM instances.
4. Click CREATE INSTANCE.

5. Enter the following to create a virtual machine:

   a. For the Name, enter a name for the virtual machine.

      The name can be anything meaningful to your organization, e.g., bursting-vm.

   b. For Zone, select a zone that is in the same Region as the subnet of the previously created VPC.

   c. In the Machine type, click the Customize link.

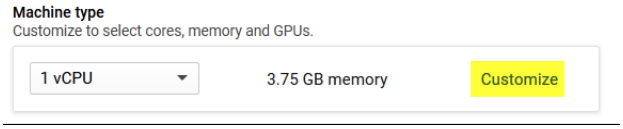


Figure 5-11:Customize the Machine Type

   d. Specify the CPUs, GPUs and RAM.

      Consider the nature of the applications that you plan to deploy on the instance, the number of users that you expect to use the applications, and also how you expect the load to scale in the future. Remember to also factor in the CPU and memory resources that are necessary for the operating system.

   e. For Boot disk, click Change.

   f. Choose CentOS 7.

   g. For Boot disk type, choose one of the following options:

      - Standard persistent disk
      - SSD persistent disk

      Choose SSD for I/O-intensive applications, where low latency and high throughput are critical. For testing, consider Standard persistent disk to keep costs down.

   h. For Size, specify the size of the boot disk.

   i. Click Select.

   j. Under Identity and API access, for Service Account, select No service account.

   k. For Firewall, choose Allow HTTP traffic.

   l. Click Management, disks, networking, SSH Keys.

   m. Click the Networking tab.

   n. Click Add network interface.

   o. For Network, choose the VPC you previously created for bursting.

   p. For Network Service Tier, click Standard.

   q. Click Done.

   r. Delete any default network interfaces that might have been automatically generated.

   s. Click the Security tab.

   t. For SSK Keys, copy the SSH public key (i.e., id_rsa.pub) of the administrator account that exists on your site's PBS server host and paste it.

      This user will have sudo rights and will be able to SSH into the VM to install the PBS MoM.

6. Click Create.

   Creating the virtual machine may take some time.

5.3.6 Install and Configure a PBS MoM on the VM

1. Log in to your site's PBS Server as the user account (public SSH key) provided during the creation of the VM.
2. SSH into the virtual machine using the public IP address of the VM:

   ```
   ssh <public IP address of VM>
   ```

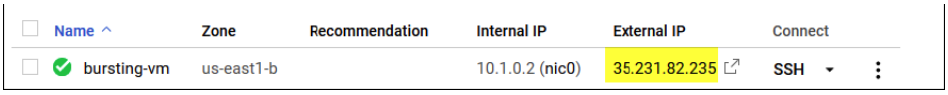


Figure 5-12:Bursting Virtual Machine

3. Switch to root:

   ```
   sudo -i
   ```

4. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
5. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount /home in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
6. If cloud-init is not installed, install it.

5.3.7 Create an OS Image

GCP documentation can be found at Creating, Deleting, and Deprecating Custom Images.

1. Log in to the GCP console.
2. Click located in the upper left-hand corner of the web page.
3. Click Compute Engine > VM instances. A list of virtual machines is displayed.
4. Click the three vertical dots next to the virtual machine that was created for cloud bursting.

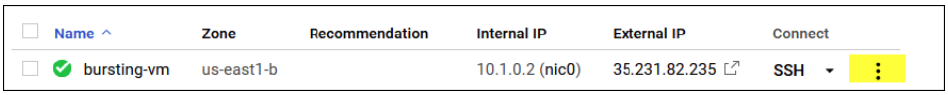


Figure 5-13:Bursting Virtual Machine

5. Click Stop.

   It may take some time for the VM to be stopped. Do not proceed until the VM is stopped.
6. In the menu located on the left-hand side of the web page, click Images .
7. Click CREATE IMAGE.
8. Enter the following to create an image:
   a. For Name, enter a name for the image.

   The name can be anything meaningful to your organization, e.g., bursting-image.

   b. For Source select Disk.
   c. For Source disk, select the previously created virtual machine.
9. Click Create.

It may take some time to create the image. Do not proceed until the image is created.

10. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.3.8 Collect Information for GCP Cloud Bursting Scenario

5.3.8.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about each one:

Table 5-4: Scenario Parameters for Google Cloud Platform (GCP)

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Network name	Name of VPC network for cloud bursting created at vendor	String
Subnetwork name	Name of VPC network subnet created at vendor	String
OS Image URI	Value for entry called selfLink of REST equivalent of OS image created for cloud bursting at vendor	String

5.3.8.2 Steps to Collect Information

1. Open a browser window and log in to your GCP console.
2. Click  located in the upper left-hand corner of the web page.
3. Click VPC network > VPC networks.
4. Click on the name of the VPC that was created for cloud bursting. VPC network details are displayed.
5. Copy the name of the VPC network.
6. **Save the name of the VPC network**. You will use this later when you create a bursting scenario.
7. Copy the name of the VPC network subnet.
8. **Save the name of the VPC subnet**. You will use this later when you create a bursting scenario.
9. Click  located in the upper left-hand corner of the web page.
10. Click Compute Engine > Images.
11. Select the image created for cloud bursting.
12. Click Equivalent REST
13. Copy the value for the entry called selfLink.
14. **Save this value**. You will use this later when you create a bursting scenario.

15. Click Next.

5.4 Configure Oracle Cloud Platform Cloud Bursting

While you are working your way through the following sections, make sure you save the items in the following table for later when you add the vendor account to PBS Cloud. We will remind you about them:

Table 5-5: Account Parameters for Oracle

Account Parameter	What to Collect During Configuration at Vendor	Format
User OCID	User OCID generated when creating Oracle cloud user account at vendor	String
Tenant OCID	Tenancy OCID generated at vendor	String
Compartment OCID	Root compartment OCID generated at vendor	String
Fingerprint OCID	Fingerprint generated when adding the public SSH key for Oracle user at vendor	String
Private Key	RSA private key generated at vendor	String

5.4.1 Sign Up for an Oracle Cloud Account

Sign up for an Oracle Cloud account and get an associated tenancy. Oracle documentation can be found at Adding Users and Resource Identifiers.

5.4.2 Create Oracle Cloud User Account

Click ☰ located in the upper left-hand corner of the web page.

1. Log in to the Oracle Cloud Infrastructure console.
2. Click Identity > Users.
3. Click Create User.
4. Enter the following to create the user:
   a. For NAME, enter a name for the user.
      The name can be anything meaningful to your organization, e.g., pc_clouduser.
   b. For DESCRIPTION, enter a description of the user.
5. Click Create.
   The user account is created and displayed in the users list.
6. Click Show located under the name of the user. The user account's OCID is displayed.
7. Click Copy to copy the OCID.
8. **Store the user OCID to a file**. You will need this later when you add the account to PBS Cloud.
9. Click Group from the menu located on the left-hand side of the web page.
10. Click Create Group.

11. Enter the following:
    a. For Name, enter Administrators.
    b. For Description, enter a description for the group.
    c. Click Submit.

    The group is created and is displayed in the Groups list.
12. Click on the name of the group.
13. Click Add User to Group.
    a. For User, select the user that was previously created (e.g., pc_clouduser).
    b. Click Add.
14. Click ☰ located in the upper left-hand corner of the web page.
15. Click Identity > Policies.
16. Click Create Policy.
    a. For Name, enter a name for the policy.
    b. For Policy Versioning, enable Keep Policy Current.
    c. For Policy Statements, enter: ALLOW GROUP Administrators to manage all-resources IN TENANCY
    d. Click Create.

5.4.3 Generating an SSH Public Key for the Oracle Cloud User

You will use OpenSSL to create a private and public key in a PEM format for the previously created Oracle Cloud user. If you're using Windows, you'll need to install Git Bash for Windows and run the commands with that tool.

1. Generate a private key by executing the following command:
   ```
   openssl genrsa -out oracle_private_key.pem 2048
   ```
   We recommend changing the permissions on this file so that only you have read/write access.
2. **Save the private key file**. You will need this later when you add the vendor to PBS Cloud.
3. Generate the public key by executing the following command:
   ```
   openssl rsa -pubout -in oracle_private_key.pem -out oracle_public_key.pem
   ```
4. Log in to the Oracle Cloud Infrastructure console.
5. Click ☰ located in the upper left-hand corner of the web page.
6. Click Identity > Users.
7. Click the name of the previously created user (e.g., pc_clouduser).
8. Click Add Public Key.
9. Copy and paste the contents of the public RSA key file.

10. Click Add to generate a fingerprint:

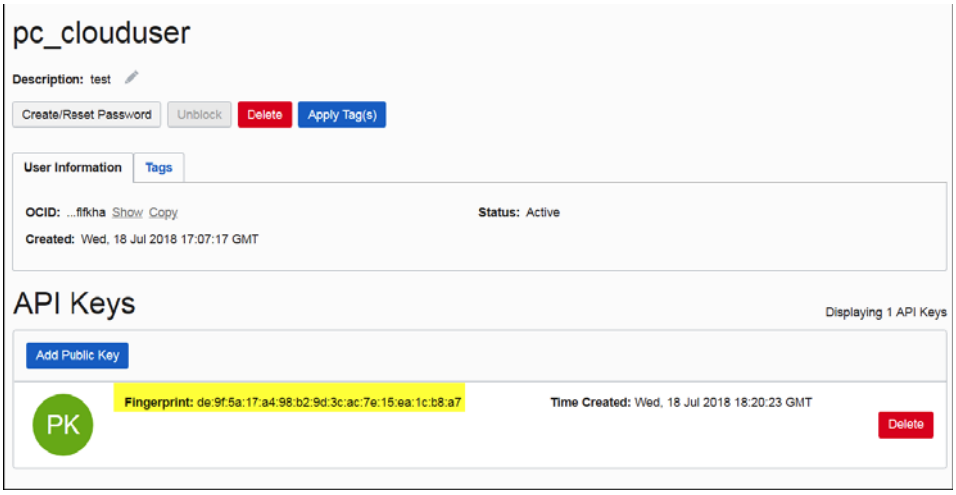


Figure 5-14:Public Key Fingerprint

11. Copy the fingerprint.
12. **Store the fingerprint to a file**. You will need the fingerprint later to add the vendor account to PBS Cloud.

5.4.4 Obtain the Root Compartment Identifier

Oracle documentation can be found at Understand Compartments.

When your tenancy is provisioned, a root compartment is created for you. Compartments can be used to organize and isolate your resources to make it easier to manage and secure access to them. Your root compartment holds all of your cloud resources. You can think of the root compartment like a root folder in a file system. The first time you sign in to the Oracle Cloud Console and select a service, you will see your root compartment. All the resources required for cloud bursting will be contained in this root compartment. You will need the root compartment's resource identifier to add an Oracle cloud account to PBS Cloud.

1. Log in to the Oracle Cloud Infrastructure console.
2. Click ☰ located in the upper left-hand corner of the web page.
3. Click Identity > Compartments.
4. Click Show located under the name of the root compartment. The compartment's OCID is displayed.
5. Click Copy to copy the OCID.
6. **Store the root compartment OCID to a file**. You will need this later when you add the vendor to PBS Cloud.

5.4.5 Obtain the Tenancy Identifier

1. Log in to the Oracle Cloud Infrastructure console.
2. Click ☰ located in the upper left-hand corner of the web page.
3. Click Administration > Tenancy Details.
4. Under Tenancy Information, click Show located to the right of OCID:

The tenancy's OCID is displayed.

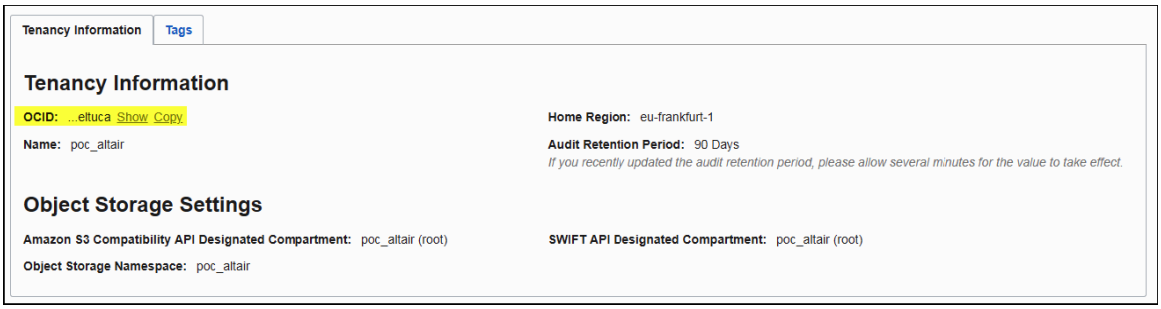


Figure 5-15:Tenancy OCID

5. Click Copy to copy the OCID.
6. **Store the tenancy OCID to a file**. You will need this later when you add the vendor cloud account to PBS Cloud

5.4.6 Create a Virtual Cloud Network

Oracle documentation can be found at Overview of Networking and Creating a Virtual Cloud Network.

Make sure that the VCN has a subnet associated with each of the region's availability domains.

1. Log in to the Oracle Cloud Infrastructure console.
2. Click ☰ located in the upper left-hand corner of the web page.
3. Click Networking > Virtual Cloud Networks.
4. Choose a region based on the geographical location of your users. Use the REGION pull-down menu.
5. Click Create Virtual Cloud Network.
6. Enter the following to create a VCN:
   a. For CREATE IN COMPARTMENT, select the root compartment.
   b. For NAME, enter a name for the VCN.

      The name can be anything meaningful to your organization, e.g., bursting_vcn.
   c. Enable CREATE VIRTUAL CLOUD NETWORK PLUS RELATED RESOURCES.

      Choosing this option automatically creates a VCN with a CIDR block 10.0.0.0/16, an internet gateway, a route rule to enable traffic to and from the internet gateway, the default security list, the default set of DHCP options, and one public subnet per availability domain.
7. Click Create Virtual Cloud Network.

   A summary of the VCN, internet gateway, default route table, and subnets is displayed.
8. Click Close.

A list of VCNs is displayed.

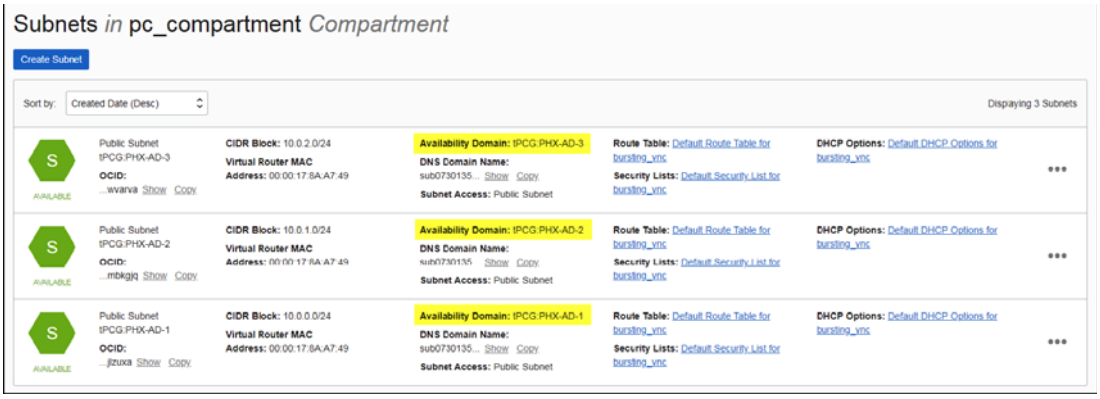


Figure 5-16:Virtual Machine Subnets and Associated Availability Domains

9. Click the name of the VCN.

   The subnets are displayed. A subnet is created for each availability domain (data center) located in the previously selected region.

5.4.7 Check Tenancy Service Limits

When you sign up for Oracle Cloud Infrastructure, a set of service limits are configured for your tenancy. The service limit is the quota or allowance set on a resource. For example, your tenancy is allowed a maximum number of compute instances (virtual machines) per availability domain. These limits are generally established with your Oracle sales representative when you purchase Oracle Cloud Infrastructure. Oracle documentation can be found at Service Limits and Regions and Availability Domains.

When you reach the service limit for a resource, you receive an error when you try to create a new resource of that type. You cannot create a new resource until you are granted an increase to your service limit or you terminate an existing resource.

View your tenancy's limits to ensure that there are sufficient resources available in a region's availability domains.

1. Log in to the Oracle Cloud Infrastructure console.
2. Choose the region where the previously created VCN is hosted. Use the REGION pull-down menu.
3. Click ☰ located in the upper left-hand corner of the web page.
4. Click Governance > Service Limits.
5. Scroll down to the Service Limits section.
6. Click Compute.

Availability domains (data centers) for the region are displayed. For each resource (VM shape) the number of nodes that can be burst in the corresponding availability domains are displayed. In the below example, three nodes can be burst in each data center in the us-phoenix-1 region for the VM Standard1.1 shape.

Resource	Region (us-phoenix-1)		tPCG:PHX-AD-1		tPCG:PHX-AD-2		tPCG:PHX-AD-3	
	Limit	Usage	Limit	Usage	Limit	Usage	Limit	Usage
Custom Images	25	0	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*	*n/a*
VM.DenseIO1.16	*n/a*	*n/a*	0	0	0	0	0	0
VM.DenseIO1.4	*n/a*	*n/a*	1	0	1	0	1	0
VM.DenseIO1.8	*n/a*	*n/a*	0	0	0	0	0	0
VM.DenseIO2.16	*n/a*	*n/a*	0	0	0	0	0	0
VM.DenseIO2.24	*n/a*	*n/a*	0	0	0	0	0	0
VM.DenseIO2.8	*n/a*	*n/a*	0	0	0	0	0	0
VM.Standard1.1	*n/a*	*n/a*	3	0	3	0	3	0
VM.Standard1.16	*n/a*	*n/a*	0	0	0	0	0	0
VM.Standard1.2	*n/a*	*n/a*	3	0	3	0	3	0

Figure 5-17:Virtual Machine Type Limits

7. Verify that the appropriate service limits are set for your tenancy based on the VM shape chosen for the virtual machine and the region's availability domains.

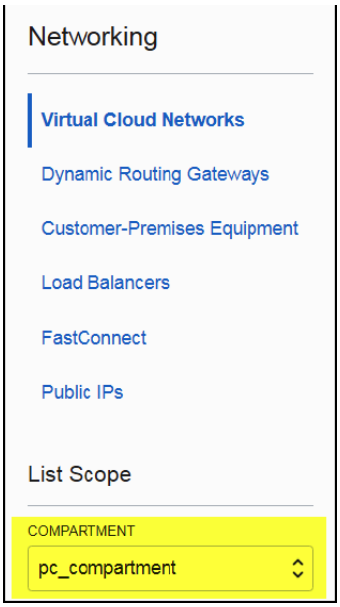
To request an increase a service limits for your tenancy see Requesting a Service Limit Increase.

5.4.8 Creating a Virtual Machine

Virtual machines are hosted in availability domains (data centers) located in a region and are based on predefined VM shapes. Before proceeding, determine the VM shape that your site requires for cloud bursting based on the number of CPUs, memory, disk space, network bandwidth, and virtual network interface cards. While selecting the shape for a VM, consider the nature of the applications that you plan to deploy on the instance, the number of users that you expect to use the applications, and also how you expect the load to scale in the future. Remember to also factor in the CPU and memory resources that are necessary for the operating system.

1. Log in to the Oracle Cloud Infrastructure console.
2. Click ☰ located in the upper left-hand corner of the web page.
3. Click Compute > Instances.
4. Choose the region where the previously created VCN is hosted. Use the REGION pull-down menu.

5. For COMPARTMENT, select the root compartment.



6. Click Create Instance.
7. Enter the following to create a virtual machine:
   a. For NAME, enter a name for the VM.

      The name can be anything meaningful to your organization, e.g., bursting_vm.

   b. For AVAILABILITY DOMAIN, choose one of the region's availability domains.

      The virtual machine is hosted in the chosen availability domain (data center). Choose the availability domain that best suits your site's cloud bursting requirements based on the machine type of the virtual machine (VM shape) and service limits.

   c. For BOOT VOLUME, enable ORACLE-PROVIDED OS IMAGE.
   d. For IMAGE OPERATING SYSTEM, choose CentOS 7.
   e. For SHAPE TYPE, enable VIRTUAL MACHINE.
   f. For SHAPE, select a VM shape.

      Choose the VM shape that best suits your site's cloud bursting requirements based on number of CPUs, memory, disk space, and network bandwidth.

   g. For IMAGE VERSION, select the latest available one.
   h. For BOOT VOLUME CONFIGURATION, enable CUSTOM BOOT VOLUME SIZE and enter a boot volume size in GBs.
   i. Enable PASTE SSH KEYS and copy the SSH public key (i.e., id_rsa.pub) of a user account that exists on your site's PBS Server and paste it.

      This user will have sudo rights and will be able to SSH into the VM to install the PBS MoM.

   j. For VIRTUAL CLOUD NETWORK, choose the VCN that was created for cloud bursting.
   k. For SUBNET, choose the subnet associated with the previously chosen availability domain.
8. Click Create Instance.

Creating the virtual machine may take some time. It is done when the state is "Running".

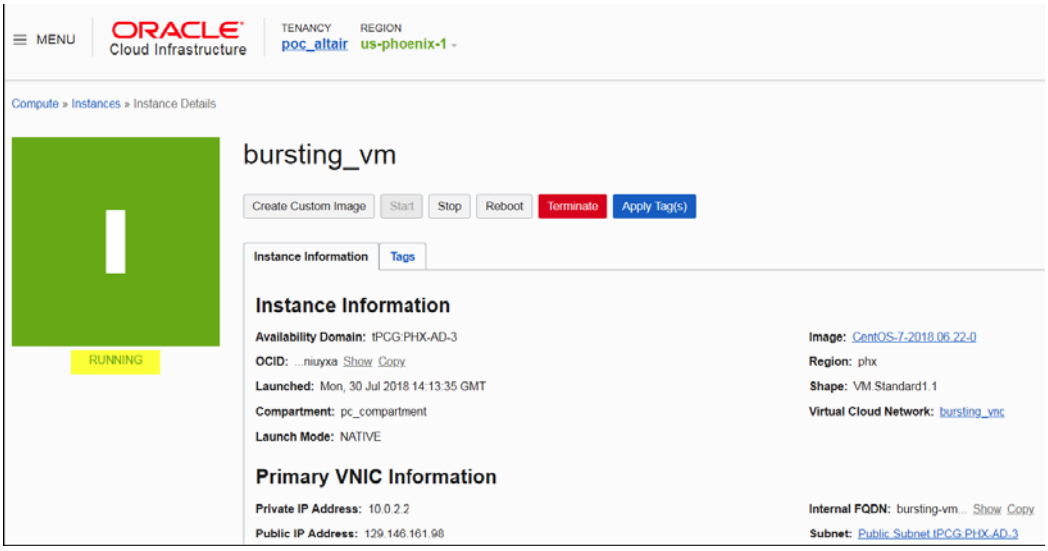


Figure 5-18:Running Virtual Machine

5.4.9 Installing and Configuring a PBS MoM on the VM

1. Log in to your site's PBS Server as the user account provided during the creation of the VM.
2. SSH into the virtual machine using the default user "opc", the private SSH key of the user account provided during the creation of the VM and the External IP assigned to the VM.

   ```
   ssh -i PRIVATE_KEY_PATH opc@PUBLIC_IP_ADDR
   ```

   Where PRIVATE_KEY_PATH is the path to the file that contains the private SSH key of the user account provided during the creation of the VM and PUBLIC_IP_ADDR is the public IP address of the VM.

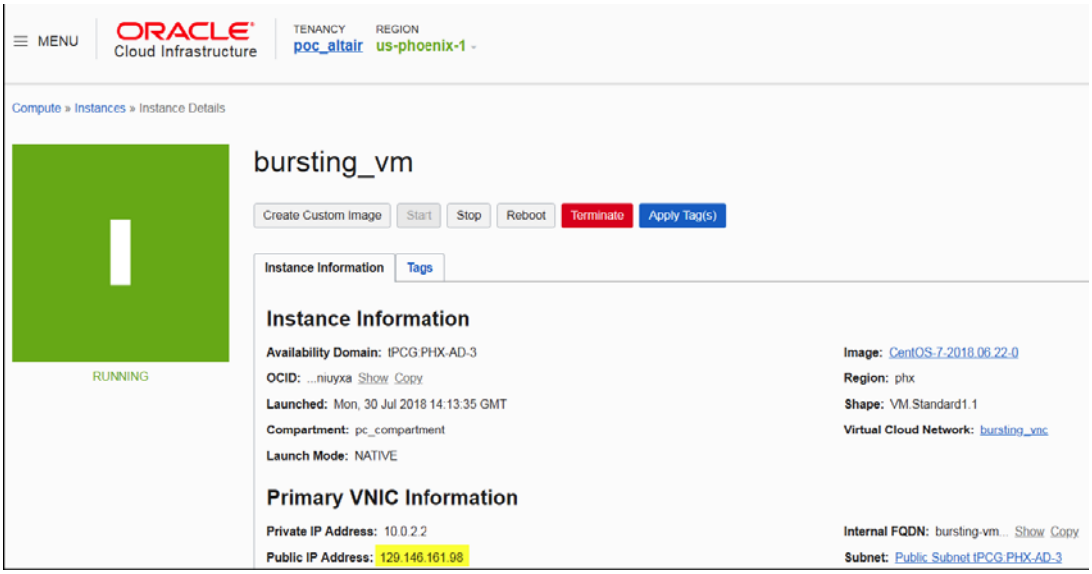


Figure 5-19:Bursting Virtual Machine

3. Switch to root:

   ```
   sudo -i
   ```

4. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
5. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount `/home` in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
6. If cloud-init is not installed, install it.

5.4.10 Create an OS Image

Oracle documentation can be found at Managing Custom Images.

1. Log in to the Oracle Cloud Infrastructure console.
2. Click ☰ located in the upper left-hand corner of the web page.
3. Click Compute > Instances.

   A list of virtual machines is displayed.
4. Click the name of the virtual machine created for cloud bursting.
5. Click Create Custom Image.
6. Enter the following to create a custom image:
   a. For CREATE IN COMPARTMENT, select the root compartment.
   b. For NAME, enter a name for the image.

      The name can be anything meaningful to your organization, e.g., bursting_image.
7. Click Create Custom Image.

   It may take some time to create the image. Do not proceed until the image is created.
8. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.4.11 Collect Information for Oracle Cloud Bursting Scenario

5.4.11.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about them:

Table 5-6: Scenario Parameters for Oracle

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String

Table 5-6: Scenario Parameters for Oracle

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Subnet ID	OCID of subnet associated with data center where cloud bursting virtual machine is hosted,	String
OS Image URI	Vendor link to bursting image OCID	String

5.4.11.2 Steps to Collect Information

Open a browser window and log in to the Oracle Cloud Infrastructure console.

9. Click ☰ located in the upper left-hand corner of the web page.
10. Click Networking > Virtual Cloud Networks.
11. Click the name of the VCN created for cloud bursting.
12. Locate the subnet associated with the availability domain where the cloud bursting virtual machine is hosted.

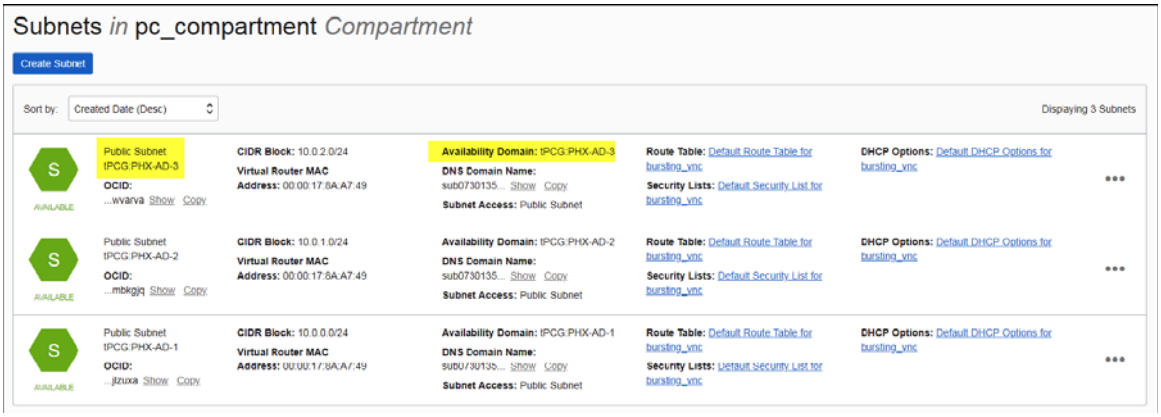


Figure 5-20:Subnet and Associated Availability Domain

13. Click Show located under the name of the subnet.

    The subnet's OCID is displayed.
14. Click Copy to copy the OCID.
15. **Save the subnet OCID to a file**. You will need this later when you create the bursting scenario.
16. Navigate to the Oracle Cloud Infrastructure browser window.
17. Click ☰ located in the upper left-hand corner of the web page.
18. Click Compute > Custom Images. A list of custom images is displayed.
19. Locate the custom image created from the cloud bursting virtual machine.
20. Click the Show link below the name of the image to view the OCID.
21. Click Copy to copy the OCID of the image.
22. **Save the the image OCID to a file**. You will need this later when you create the bursting scenario.

5.5 Configure Orange Cloud Flexible Engine for Cloud Bursting

5.5.1 Purchase an Orange Business Services Account

Purchase an Orange Business Services account. You will use your Orange ID and password to access the Flexible Engine console. For more information visit Orange Cloud.

You should be able to log in to the Orange Cloud Customer space (https://selfcare.cloud.orange-business.com/) with the credentials provided to you with your Orange Cloud account. You will also be provided with a domain name when you sign up for your Orange Cloud account.

5.5.2 Create an Orange Cloud Flexible Engine User Account

PBS Cloud will use an Orange Cloud user account to manage cloud nodes. Make sure that you collect the following information while you are creating the user account (we will also remind you):

Table 5-7: Account Parameters for Orange

Account Parameter	What to Collect During Configuration at Vendor	Format
Auth URL	*https://iam.<orange region>.<console link>*	String
User Domain Name	Orange ID used to log in to Orange account. Same as domain name.	String
Username	Administrator username created at vendor	String
Password	API password generated at vendor	String

1. Go to the Orange Cloud Customer space login page.
2. Enter your Orange Cloud credentials.
3. Click Your services.



Figure 5-21:Orange Cloud Customer Space Services

4. In the navigation bar on the top click Users
5. Click Add user.

6. Enter the following user details:
   a. For Civility, choose the form of address
   b. For Last name, enter the user's last name.
   c. For First name, enter the user's first name.
   d. For Login, enter a login name for the user.
   e. For Email, enter the user's email address.
   f. For Phone number, enter the user's phone number.
   g. For Mobile phone, enter the user's mobile phone number.
   h. For Preferred language, choose the language in which the application should be displayed.
   i. Click next.
7. In the Roles section enter these details:
   a. For Billing, choose Visitor.
   b. For Contracts, choose Account Manager.
   c. For Dashboard, choose Visitor.
   d. For Documents, choose Visitor.
   e. For Orders, choose Visitor.
   f. For Services, choose Visitor.
   g. For Subscriptions, choose Visitor.
   h. For Support, choose Visitor.
   i. For Users, choose Manager.
   j. For Flexible Engine Console, choose admin.
   k. Click next.
8. In the Summary section review your choices. Click previous to edit your choices.
9. Click finish.

   The new user account is created and displayed in the list of users. Emails are sent to the email address you specified. The emails will contain:
   - Orange ID (Domain Name). This is the administrator username PBS Cloud will use to manage cloud nodes.
   - Link to set Orange Password. This password is for the administrator to log into Web interface.
   - Link to access Cloud Customer Space.
   - Link to log in to the Flexible Engine Console.
   - Link to define your API password. This password is used by the administrator account for making API requests; PBS Cloud will use this to make API requests.
10. Click the link in the email to set your Orange Password.
11. Click the link in the email to set your API Password.
12. **Store the API Password to a file**. The API Password is used later to create a cloud account in PBS Cloud.

5.5.3 Select a Region

Define a region in the Orange Cloud Flexible Engine console to set up the infrastructure for cloud bursting.

A region is a geographic area where resources used by your ECSes are located. ECSes in the same region can communicate with each other over an intranet, but ECSes in different regions cannot. Before setting up the infrastructure for cloud bursting, it is important to ensure that all the resources are defined in the same region. An Authorization URL is required for adding the Orange Cloud Flexible Engine cloud account in PBS Cloud. This is based on the region selected.

1. Log in to the Orange Cloud Flexible Engine console.
2. In the navigation bar on the top, pull down the region menu and select the region for setting up your infrastructure.
3. For the Authorization URL, (IAM URL), enter the URL in the following format based on the region you chose in the Orange Cloud Flexible Engine console:

   https://iam.<orange region>.<console link>

   e.g. `https://iam.eu-west-0.prod-cloud-ocb.orange-business.com`
4. **Store the region and Auth (IAM) URL in a file**. You will use these to add the provider cloud account to PBS Cloud.

5.5.4 Check Orange Cloud Flexible Engine Account Service Quota

Quotas are used to limit the number of resources available to users. It is important to ensure you are not exceeding your quota while setting up the resources for cloud bursting. If the existing resource quota cannot meet your service requirements, you can submit a work order to increase your quota. Once your application is approved, Orange Cloud Flexible Engine will update your resource quota accordingly and send you a notification.

1. Log in to the Orange Cloud Flexible Engine console.
2. In the navigation bar on the top right hand side, click 

   Information about resources usage and availability is displayed.

5.5.5 Create a Virtual Private Cloud

Orange Cloud Flexible Engine documentation for VPC can be found at:Virtual Private Cloud Documentation

1. Log in to the Orange Cloud Flexible Engine console.
2. In the top navigation bar select the region where you wish to deploy your cloud infrastructure.
3. From the Network section click Virtual Private Cloud.
4. Click + Create VPC.
5. In the Basic Information section:
   a. For Region, ensure the VPC is the same region as the other resources.
   a. For Name, enter a name for the VPC.
   a. For CIDR Block, enter CIDR values for the VPC.
6. In the Subnet Settings section:
   a. For the Subnet Settings choose the AZ (Availability Zone) as the same as the region.
   b. For Subnet Name, enter a name to match the VPC Name.
   c. Enter CIDR Block for Subnet.
   d. For Advanced Settings, click Default.
7. Review the Configuration information.

8. Click Create Now.
9. Once the VPC is created, click the Back to VPC List.
10. Click Security Group in the left hand side menu.
11. Click + Create Security Group.
12. For Name, enter a name for the Security Group.
13. For Description, enter a suitable Description.
14. Click OK
15. By default, the Outbound and Inbound traffic over IPv4 is open. You can add firewall rules to this security group if required.

5.5.6 Creating a Virtual Machine

1. Log in to the Orange Cloud Flexible Engine console.
2. In the Computing section and click Elastic Cloud Server.
3. From the menu on the left hand side click Key Pair.
4. Click + Create Key Pair.
5. Enter a Name for the Key Pair.
6. Click OK.
7. Save the Key Pair (.pem) file to your local disk in a secure location. The information in this .pem file is used later to SSH into the VM.
8. Click OK to confirm that you have downloaded the Key Pair file.
9. From the menu on the left hand side, click Elastic Cloud Server.
10. Click + Create ECS.
11. For Region, click the region you selected for setting up the infrastructure.
12. For AZ (Availability Zone), select the AZ related to the region.
13. In the Specifications section:
    a. For ECS type, click one of the flavor names. Orange Cloud Flexible Engine provides a set of predefined ECS types for specific requirements. Click a flavor name to get the list of available configurations.
    b. Review the specifications you have selected.
14. In the Image section:
    a. Click Public image.
    b. From the drop down menu select CentOS
    c. From the version drop down menu select Select OBS_U_CentOS_7.2(40GB)
15. In the Disk section select the defaults.

16. In the VPC section:
    a. For VPC, select the VPC you created from the drop down menu.
    a. For NIC, choose the default primary NIC.
    b. For Security Group, select the Security Group you created for the VPC.
    c. For EIP, click Automatically assign
    d. For Bandwidth, specify it as 5 Mbit/s.
17. For Login Mode, select the Key Pair you generated earlier from the drop down menu.
18. For Auto Recovery, click Enable.
19. For Advanced Settings, click Do not configure.
20. For ECS Name, enter a name.
21. For Quantity, specify 1.
22. Review the Current Configuration.
23. Click Create Now.
24. Review the Specifications.
25. Click Submit.

    The ECS (Virtual Machine) is created and displayed in the list of ECS.

5.5.7 Installing and Configuring a PBS MoM on the VM

1. Log in to your site's PBS Server.
2. Log in to the Flexible Engine console.
3. In the Computing section, click Elastic Cloud Server.
4. In the search box above the upper right corner of the ECS list, enter the ECS name, IP address, or ID, and click the search icon.
5. Click the name of the target ECS.
6. The page providing details about the ECS is displayed.
7. Copy the Public IP address (External IP) of the ECS.
8. SSH into the VM using the default user "cloud", the .pem file you generated when creating the VM, and the External IP assigned to the VM. For more information about logging into the Linux ECS, refer to the Elastic Cloud Server User Guide.

   ssh -i /<path to .pem file>/<name of .pem file>.pem cloud@<public IP address of VM>
9. Switch to root:

   sudo -i
10. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
11. Save the file.

12. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
    - Mount /home in the VM
    - Either install applications in the VM or crossmount them from the head node
    - Either add users to the password file or connect the VM to a service such as NIS
13. If cloud-init is not installed, install it.

5.5.8 Create an OS Image

Orange Cloud Flexible Engine documentation can be found at Creating a Linux Private Image Using an ECS.

5.5.8.1 Prerequisites

Before creating the Image from the ECS, you must have.

- A Linux ECS in the Stopped state.
- Configured DHCP for the NICs of the ECS
- Configured Network attributes of the ECS
- Detached Data Disks from the ECS

5.5.8.2 Steps to Create OS Image

1. Log in to the Flexible Engine console.
2. In the Computing section, click Image Management Service.
3. On the Image Management Service page, click + Create Private Image.
4. In the Image Type and Source section, .
    a. For Type, click System disk image
    b. For Source, click ECS
    c. Select the target ECS from the ECS list.
5. Set the required information, such as Name and Description.
6. Click Create Now.
7. Confirm the parameters and click Submit.
8. Switch back to the Image Management Service page to view the image status.

    The time required for creating an image varies depending on the image file size. Generally, it takes about 20 minutes to create an image. The image is successfully created when its image status changes to Normal.

    Do not perform any operation on the selected ECS or its associated resources during image creation.
9. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.5.9 Create Orange Cloud Cloud Bursting Scenario

5.5.9.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about them:

Table 5-8: Scenario Parameters for Orange

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your administrator account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Subnet ID	ID of subnet for VPC created at cloud vendor	String
Security Group ID	ID of security group created at cloud vendor	String
OS Image URI	ID of OS image created at cloud vendor	String

5.5.9.2 Steps to Collect Information

Open a browser window and log in to the Orange Cloud Flexible Engine console.

10. Click Service List in the menu bar.

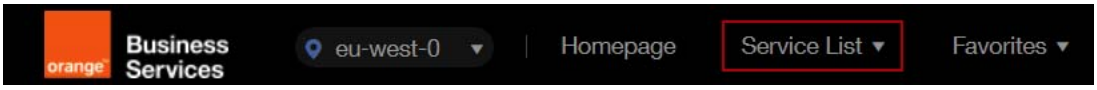


Figure 5-22:Orange Cloud Flexible Engine Console

11. Under Network, click Virtual Private Cloud.
12. Click Virtual Private Cloud from the menu located on the left-hand side of the web page.
13. Click the name of the VPC you created for cloud bursting.
14. Click the name of the Subnet for the VPC.
15. Copy the Subnet ID.
16. **Save the subnet ID to a file**. You will need this later when you create a bursting scenario.
17. Click Security Group from the left hand side menu.
18. Click the name of the Security Group you created for the VPC.
19. Copy the ID of the Security Group.
20. **Save the security group to a file**. You will need this later when you create a bursting scenario.
21. Click Service List in the menu bar.
22. Under Computing, click Image Management Service.
23. Click the Private Images tab.
24. Click the name of the VM image you created for cloud bursting.

25. Copy the ID of the image.
26. **Save the OS Image ID to a file**. You will need this later when you create a bursting scenario.

5.6 Configure HUAWEI Cloud for Cloud Bursting

5.6.1 Create and Activate HUAWEI Account

Create and activate a HUAWEI Cloud account.

5.6.2 Get the HUAWEI Cloud Administrator Credentials

PBS Cloud will use the HUAWEI Cloud administrative user account to manage cloud nodes. While you are getting the credentials for the administrative user account, make sure you capture the following information:

Table 5-9: Account Parameters for Huawei

Account Parameter	What to Collect During Configuration at Vendor	Format
Auth URL	*https://iam.ap-southeast-1.myhwclouds.com*	String
User Domain Name	Domain Name provided when your subscription to HUAWEI Cloud was created If you do not know your Domain name, contact HUAWEI Cloud support.	String
Username	Administrator username created at vendor	String
Password	Administrator password created at vendor	String

5.6.2.1 Choose Administrative User

You can create a new user and give the user administrative privileges, or you can use the administrative user account that is automatically created when you subscribe to HUAWEI Cloud.

The automatically-created user is an administrative user account with permissions for all system operations.

If you create a new user, give the user administrative privileges by setting its User Group to "admin".

5.6.2.2 Get Credentials

1. Log in to the HUAWEI Cloud Console.
2. Click Service List.

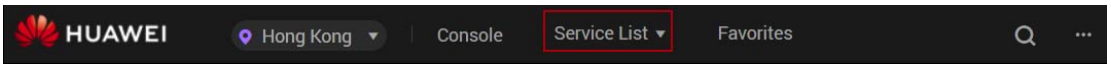


Figure 5-23:HUAWEI Cloud Console

3. Under Management & Deployment, click Identity and Access Management.
4. Click Users from the menu located on the left-hand side of the web page. A list of users is displayed.
5. Click the down-arrow located next to a username to display the user's details.

The user account listed as an "admin" is the account to use to create the cloud account in PBS Cloud.

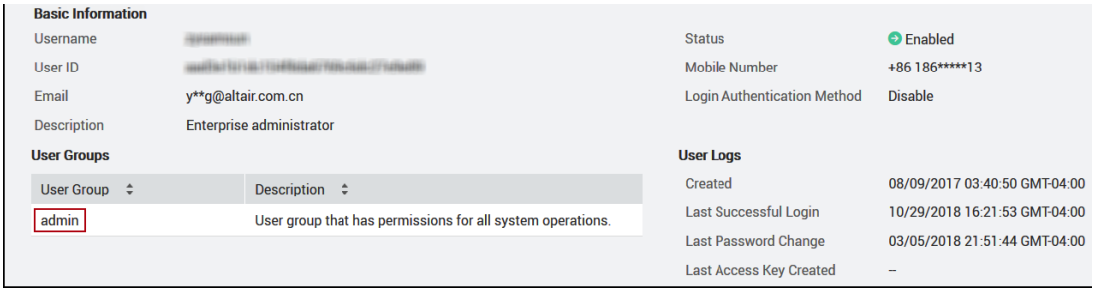


Figure 5-24:User Details

6. If you do not know the password for the administrative user account, click Set Credentials.

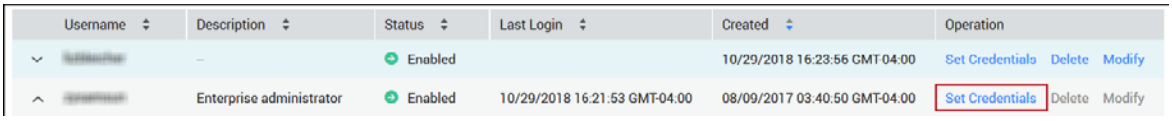


Figure 5-25:Set Password

a. Enable Set manually.

b. For Password, enter a password for the user account.

c. For Confirm Password, enter the password a second time.

d. **Save the password**. You will need this later when you add the administrative user account to PBS Cloud.

e. Click OK.

You may have to confirm the password change either by email or by a SMS text.

5.6.3 Check HUAWEI Cloud Account Service Quotas

View your OTC account resource usage and limits.

Quotas are used to limit the number of resources available to users. It is important to ensure you are not exceeding your quota while setting up the resources for cloud bursting. If the existing resource quota cannot meet your service requirements, you can submit a work order to increase your quota. Once your application is approved, HUAWEI Cloud will update your resource quota accordingly and send you a notification.

1. Log in to the HUAWEI Cloud console.
2. Click Resources > My Quota.

Information about resources usage and availability is displayed.

5.6.4 Create a Virtual Private Cloud

HUAWEI Cloud documentation for creating a VPC can be found at: Creating a VPC and Regions and AZs.

1. Log in to the HUAWEI Cloud Console.
2. Click Service List in the menu bar.
3. Under Network, click Virtual Private Cloud.
4. Click + Create VPC.

5. In the Basic Information section:
   a. For Region, select a region.

      Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

   b. For Name, enter a name for the VPC.
   c. For CIDR Block, enter an address range for the network using CIDR notation.
6. In the Subnet Settings section:
   a. For the Subnet Settings choose the AZ (Availability Zone).

      An Availability Zone is a physical location where resources use independent power supplies and networks. AZs are physically isolated and AZs in the same VPC are interconnected through an internal network.

   b. For Subnet Name, enter a name for the subnet.
   c. For CIDR, enter an address range for the subnet using CIDR notation.
   d. For Advanced Settings, click Default.
7. Click Create Now.
8. Click Back to VPC List.
9. Click Security Group in the left hand side menu.
10. Click + Create Security Group.
11. For Name, enter a name for the security group.
12. For Description, enter a suitable description.
13. Click OK.
14. By default, the Outbound and Inbound traffic over IPv4 is open. You can add firewall rules to this security group if required.

5.6.5 Creating a Virtual Machine

HUAWEI Cloud documentation for creating an ECS (virtual machine) can be found at Purchase an ECS.

1. Log in to the HUAWEI Cloud Console.
2. Click Service List in the menu bar.
3. Under Computing, click Elastic Cloud Server.
4. Click Key Pair from the menu located on the left-hand side of the web page.
5. Click + Create Key Pair.
6. For Name, enter a name for the key pair.
7. Click OK.
8. Save the key pair (.pem) file to your local disk in a secure location. The information in this .pem file is used later to SSH into the VM.
9. Click OK to confirm that you have downloaded the key pair file.
10. Click Elastic Cloud Server from the menu located on the left-hand side of the web page.
11. Click Buy ECS.

12. For Billing Mode, click Pay-per-use.
13. For Region, select the same region that was chosen for the previously created VPC.
14. For AZ (Availability Zone), select the same AZ that was chosen for the previously created VPC.
15. In the Type section:
    a. Choose an ECS type category:
       - General computing
       - General computing-plus
       - Memory-optimized
       - Large-memory
       - High-performance computing
       - Disk-intensive
    b. For ECS type, click one of the flavor based on the needs of your site.
16. In the Image section:
    a. Click Public image.
    b. For Select an OS, select CentOS.
    c. For Select an OS version, select CentOS 7.2 64bit(40GB).
17. In the Disk section, select your system disk requirements.
18. In the VPC section:
    a. For VPC, select the VPC you created for cloud bursting. The NIC information is automatically populated.
    b. For Security Group, select the security group you created for cloud bursting.
    c. For EIP, click Automatically assign
    d. For Bandwidth, specify it as 5 Mbit/s.
19. For Login Mode, select Key Pair.
20. For Key Pair, select the key pair file you generated earlier.
21. For Advanced Settings, click Not required.
22. For ECS Name, enter a name for the virtual machine.
23. For Quantity, specify 1.
24. Click Next.
25. Review the specifications.
26. Enable the I have read and agree to the Huawei Image Disclaimer checkbox.
27. Click Submit Application.
28. Click Back to ECS List.

    It may take some time to create the virtual machine. Once the ECS is created it is displayed in the ECS list.

5.6.6 Installing and Configuring a PBS MoM on the VM

1. Log in to the HUAWEI Cloud console.
2. Click Service List in the menu bar.

3. Under Computing, click Elastic Cloud Server.
4. Copy the Public IP address (External IP) of the ECS.

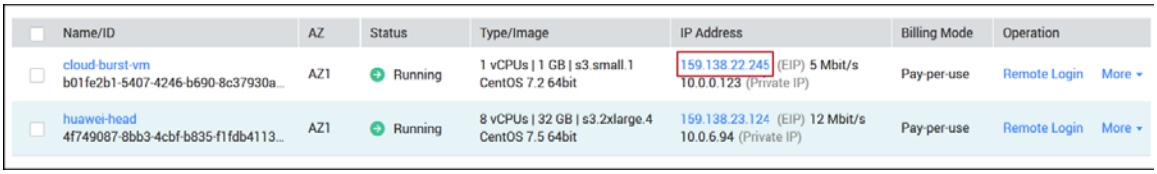


Figure 5-26:VM IP Address

5. Log in to your site's PBS Server.
6. SSH into the VM using the default user "root", the .pem file you generated when creating the VM and the External IP assigned to the VM.

   ```
   sh -i /path/my-key-pair.pem root@IPV4PublicIP
   ```

   where /path/my-key-pair.pem is the path to the .pem file downloaded while creating the virtual machine and IPV4PublicIP is the public IP address of the virtual machine.
7. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
8. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount /home in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
9. If cloud-init is not installed, install it.

5.6.7 Create an OS Image

HUAWEI Cloud documentation can be found at Creating a Linux Private Image.

1. Log in to the HUAWEI Cloud console.
2. Click Service List in the menu bar.
3. Under Computing, click Image Management Service.
4. Click + Create Image.
5. For Region, select the same region that was chosen for the previously created VPC and ECS.
6. In the Image Type and Source section, .
   a. For Type, click System disk image.
   b. For Source, click ECS.
   c. Select the virtual machine created for cloud bursting.
   d. For Name, enter a name for the virtual machine.
7. Click Next.
8. Review the specifications.
9. Enable the I have read and agree to the Statement of Commitment to Image Creation and Huawei Image Disclaimer checkbox.
10. Click Submit.

11. Click Back to Image List.

    The time required for creating an image varies depending on the image file size. Generally, it takes about 20 minutes to create an image. The image is successfully created when its image status changes to Normal.

    Do not perform any operation on the selected ECS or its associated resources during image creation.

12. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.6.8 Collect HUAWEI Cloud Bursting Scenario Information

5.6.8.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about them:

Table 5-10: Scenario Parameters for Huawei

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Subnet ID	ID of subnet for VPC created at cloud vendor	String
Security Group ID	ID of security group created at cloud vendor	String
OS Image URI	ID of OS image created at cloud vendor	String

5.6.8.2 Steps to Collect Information

Open a browser window and log in to the HUAWEI Cloud console.

13. Click Service List in the menu bar.
14. Under Network, click Virtual Private Cloud.
15. Click Virtual Private Cloud from the menu located on the left-hand side of the web page.
16. Click the name of the VPC you created for cloud bursting.
17. Click the name of the VPC's subnet.
18. Copy the Subnet ID.
19. **Save the subnet ID to a file**. You will need this later when you create a bursting scenario.
20. Click Security Group from the menu located on the left-hand side of the web page.
21. Click the name of the security group you created for the VPC.
22. Copy the ID of the security group.
23. **Save the security group to a file**. You will need this later when you create a bursting scenario.

24. Click Service List in the menu bar.
25. Under Computing, click Image Management Service.
26. Click the Private Images tab.
27. Click the name of the VM image you created for cloud bursting.
28. Copy the ID of the image.
29. **Save the OS Image ID to a file**. You will need this later when you create a bursting scenario.

5.7 Configure Open Telekom Cloud for Cloud Bursting

5.7.1 Create and Activate OTC Cloud Account

Create and activate an OTC Cloud account.

5.7.2 Obtain the OTC Administrator Credentials

PBS Cloud will use the OTC administrative user account to manage cloud nodes. While you are getting the credentials for the administrative user account, make sure you capture the following information:

Table 5-11: Account Parameters for Deutsche Telekom OTC

Account Parameter	What to Collect During Configuration at Vendor	Format
Auth URL	*https://iam.eu-de.otc.t-systems.com/v3*	String
User Domain Name	Deutsche Telekom: OTC domain name used to log in to OTC console at vendor	String
Username	Administrator username created at vendor	String
Password	Administrator password created at vendor	String

5.7.2.1 Choose Administrative User

You can create a new user and give the user administrative privileges, or you can use the administrative user account that is automatically created when you subscribe to OTC.

The automatically-created user is an administrative user account with permissions for all system operations.

If you create a new user, give the user administrative privileges by setting its User Group to "admin".

5.7.2.2 Get Credentials

1. Log in to the OTC Console.
2. Click Service List in the menu bar.
3. Under Management & Deployment, click Identity and Access Management.
4. Click Users from the menu located on the left-hand side of the web page. A list of users is displayed.
5. Click the down-arrow located next to a username to display the user's details.

The user account listed as an "admin" is the account to use to create the cloud account in PBS Cloud.

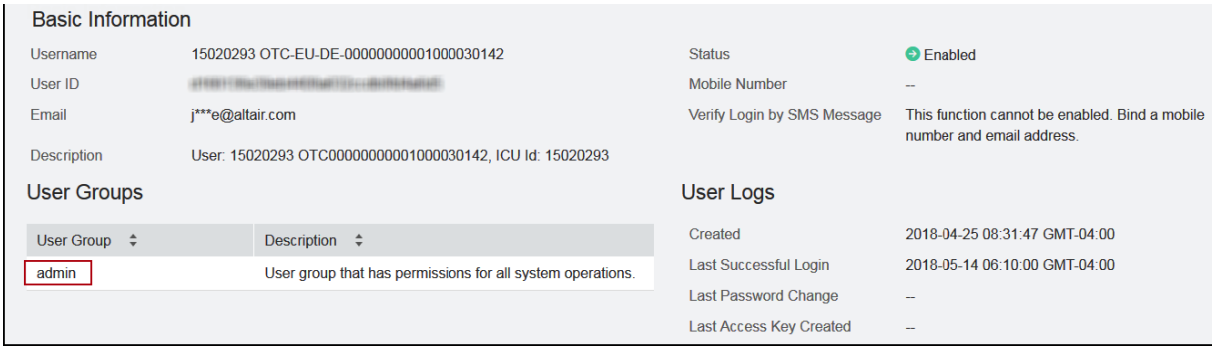


Figure 5-27:User Details

6. If you do not know the password for the admin user account, click Set Credentials

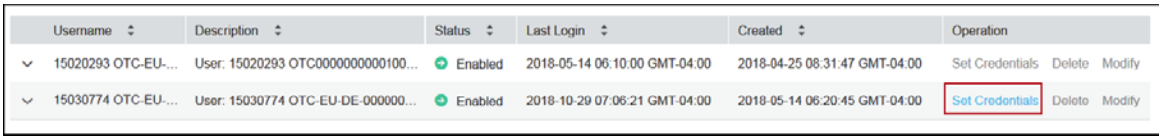


Figure 5-28:Set Password

7. Enable Set manually.
8. For Password, enter a password for the user account.
9. For Confirm Password, enter the password a second time.
10. Click OK.

    You may have to confirm the password change either by email or by a SMS text.

5.7.3 Check OTC Account Service Quotas

Quotas are used to limit the number of resources available to users. It is important to ensure you are not exceeding your quota while setting up the resources for cloud bursting. If the existing resource quota cannot meet your service requirements, you can submit a work order to increase your quota. Once your application is approved, OTC will update your resource quota accordingly and send you a notification.

1. Log in to the OTC Console.
2. Click the three vertical bars in the menu bar:

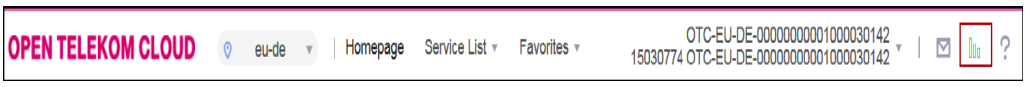
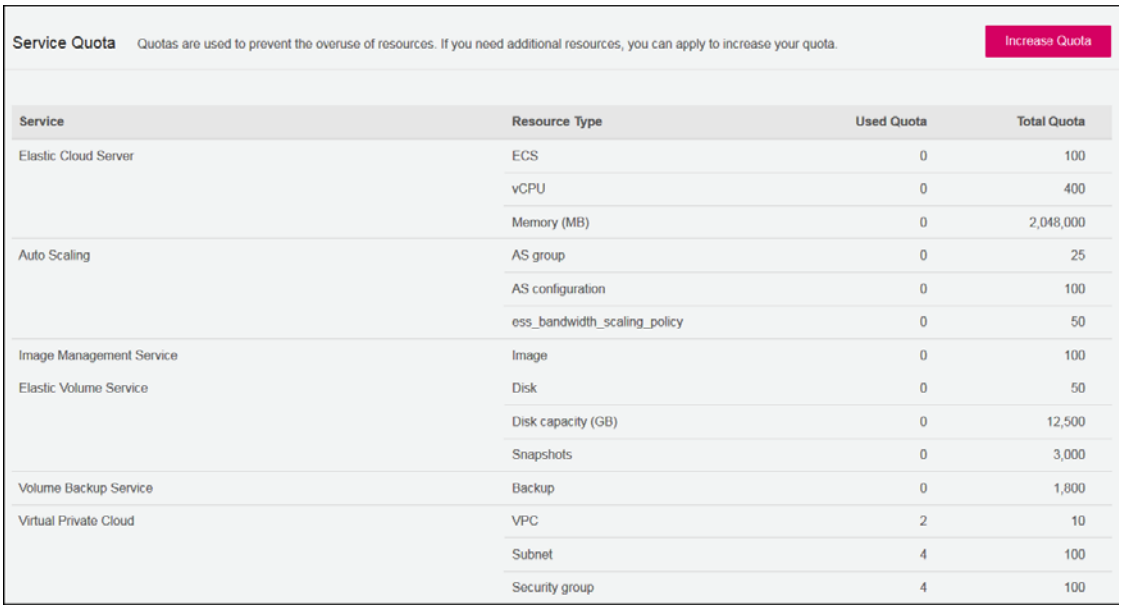


Figure 5-29:Viewing Quotas

You can see resource usage and availability:



Service Quota Quotas are used to prevent the overuse of resources. If you need additional resources, you can apply to increase your quota. Increase Quota

Service	Resource Type	Used Quota	Total Quota
Elastic Cloud Server	ECS	0	100
	vCPU	0	400
	Memory (MB)	0	2,048,000
Auto Scaling	AS group	0	25
	AS configuration	0	100
	ess_bandwidth_scaling_policy	0	50
Image Management Service	Image	0	100
Elastic Volume Service	Disk	0	50
	Disk capacity (GB)	0	12,500
	Snapshots	0	3,000
Volume Backup Service	Backup	0	1,800
Virtual Private Cloud	VPC	2	10
	Subnet	4	100
	Security group	4	100

Figure 5-30:Resource quotas and availability

5.7.4 Create a Virtual Private Cloud

OTC documentation for creating a VPC can be found at: Creating a VPC and Regions.

1. Log in to the OTC Console.
2. Click Service List.
3. Under Network, click Virtual Private Cloud.
4. Click + Create VPC.
5. In the Basic Information section:
   a. For Region, select a region.



OPEN TELEKOM CLOUD eu-de Homepage Service List Favorites

Figure 5-31:Regions

   A region is a geographical areas and can comprise one or more availability zones (AZs). A region is completely isolated from other regions. Only AZs in the same region can communicate with one another through an internal network.

   b. For Name, enter a name for the VPC.
   c. For CIDR Block, enter an address range for the network using CIDR notation.
6. In the Subnet Settings section:
   a. For Subnet Name, enter a name for the subnet.
   b. For CIDR, enter an address range for the subnet using CIDR notation.
   c. For Advanced Settings, click Default.
7. Click Create Now.
8. Click Back to VPC List.

9. Click Security Group in the left hand side menu.
10. Click + Create Security Group.
11. For Name, enter a name for the security group.
12. For Description, enter a suitable description.
13. Click OK.

    The security group rules are displayed.
14. Click the Inbound tab.
15. Click Add Rule.
    a. For Protocol/Application, select TCP.
    b. For Port, enter 22.
    c. For Source, select IP Address. and enter the PBS Cloud firewall IP address.
    d. Click OK.

5.7.5 Creating a Virtual Machine

OTC documentation for creating an ECS (virtual machine) can be found at Create an ECS.

1. Log in to the OTC Console.
1. Click Service List in the menu bar.
2. Under Computing, click Elastic Cloud Server.
3. Click Key Pair from the menu located on the left-hand side of the web page.
4. Click + Create Key Pair.
5. For Name, enter a name for the key pair.
6. Click OK.
7. Save the key pair (.pem) file to your local disk in a secure location. The information in this .pem file is used later to SSH into the VM.
8. Click OK to confirm that you have downloaded the key pair file.
9. Click Elastic Cloud Server from the menu located on the left-hand side of the web page.
10. Click Create ECS.
11. For Region, select the same region that was chosen for the previously created VPC.
12. For AZ (Availability Zone), select an availability zone.

13. In the Specifications section:
    a. Choose an ECS type category:
        - General-purpose
        - Dedicated general-purpose
        - Memory-optimized
        - Large-memory
        - High-performance
        - Disk-intensive
        - GPU-accelerated
    b. For ECS type, choose one of the flavors based on the needs of your site.
14. In the Image section:
    a. Click Public image.
    b. Select a supported OS as the operating system.
    c. Select a supported version of the OS.
15. In the Disk section, select your system disk requirements.
16. In the VPC section:
    a. For VPC, select the VPC you created for cloud bursting. The NIC information is automatically populated.
    b. For Security Group, select the security group you created for cloud bursting.
    c. For EIP, click Automatically assign
    d. For Bandwidth, specify it as 5 Mbit/s.
17. For Log in Mode, select Key Pair.
18. For Key Pair, select the key pair file you generated earlier.
19. For Advanced Settings, click Do not configure.
20. For ECS Name, enter a name for the virtual machine.
21. For Quantity, specify 1.
22. Click Create Now.
23. Review the specifications.
24. Click Submit.
25. Click Back to ECS List.

    It may take some time to create the virtual machine. Once the ECS is created it is displayed in the ECS list.

5.7.6 Installing and Configuring a PBS MoM on the VM

1. Log in to the OTC Console.
2. Click Service List in the menu bar.
3. Under Computing, click Elastic Cloud Server.

4. Copy the Public IP address (External IP) of the ECS.

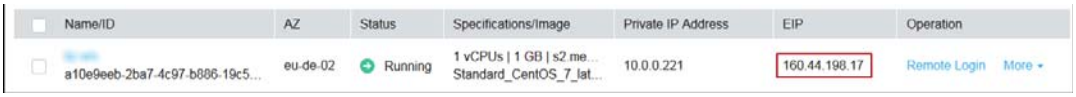


Figure 5-32:VM IP Address

5. Log in to your site's PBS Server.
6. SSH into the VM using the default user "root", the .pem file you generated when creating the VM and the External IP assigned to the VM.

   ```
   ssh -i /path/my-key-pair.pem root@IPV4PublicIP
   ```

   where /path/my-key-pair.pem is the path to the .pem file downloaded while creating the virtual machine and IPV4PublicIP is the public IP address of the virtual machine.
7. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MOM.
8. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount `/home` in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
9. If cloud-init is not installed, install it.

5.7.7 Create an OS Image

OTC documentation can be found at Creating a Linux Private Image.

1. Log in to the OTC Console.
2. Click Service List in the menu bar.
3. Under Computing, click Image Management Service.
4. Click + Create System Disk Image.
   a. For Region, select the same region that was chosen for the previously created VPC and ECS.
   b. For Source, click Server.
   c. For Server Type, click ECS.
   d. For ECS, select the virtual machine created for cloud bursting.
   e. If the virtual machine is not stopped, stop it.
   f. Click OK when prompted to verify that certain operations have been performed on the ECS. You do not need to configure or optimize the ECS.
   g. For Name, enter a name for the virtual machine.
5. Click Create Now.
6. Review the specifications.
7. Click Submit.
8. Click Back to Image List.

   The time required for creating an image varies depending on the image file size. Generally, it takes about 20 minutes to create an image. The image is successfully created when its image status changes to Normal.

Do not perform any operation on the selected ECS or its associated resources during image creation.

9. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.7.8 Create an OTC Cloud Bursting Scenario

5.7.8.1 Scenario Parameters to Collect at Vendor Interface

Make sure that you capture the following scenario parameters. We will remind you about them:

Table 5-12: Scenario Parameters for OTC

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String
Subnet ID	ID of subnet for VPC created at cloud vendor	String
Security Group ID	ID of security group created at cloud vendor	String
OS Image URI	ID of OS image created at cloud vendor	String

5.7.8.2 Steps to Collect Information

1. Open a browser window and log in to the OTC Console.
2. Click Service List in the menu bar.
3. Under Network, click Virtual Private Cloud.
4. Click Virtual Private Cloud from the menu located on the left-hand side of the web page.
5. Click the name of the VPC you created for cloud bursting.
6. Click the name of the VPC's subnet.
7. Copy the Subnet ID.
8. **Save the subnet ID to a file**. You will need this later when you create a bursting scenario.
9. Click Security Group from the menu located on the left- hand side of the web page.
10. Click the name of the security group you created for the VPC.
11. Copy the ID of the security group.
12. **Save the security group to a file**. You will need this later when you create a bursting scenario.
13. Click Service List in the menu bar.
14. Under Computing, click Image Management Service.
15. Click the Private Images tab.

16. Click the name of the VM image you created for cloud bursting.
17. Copy the Image ID of the image.
18. **Save the OS Image ID to a file**. You will need this later when you create a bursting scenario.

5.8 Configure OpenStack Cloud Bursting

You can find OpenStack documentation for the Stein release at https://docs.openstack.org/stein/index.html. For specific details, contact Altair support.

5.8.1 Get OpenStack Administrator Credentials

Get administrator credentials. While you work through the process of getting the administrator credentials, collect the following, and save them to a file:

Table 5-13: Account Parameters for OpenStack

Account Parameter	What to Collect During Configuration in Cloud Interface	Format
Auth URL	OpenStack: contact Altair support	String
User Domain Name	Domain name used to log in to your private cloud; see your OpenStack administrator	String
Username	Administrator username created using cloud interface	String
Password	Administrator password created using cloud interface	String

1. Find the username of the administrative user.
2. If you do not know the password, reset it.
3. **Save the password**. You will need this later when you add the administrative user account to PBS Cloud.

5.8.2 Create Virtual Private Cloud and OS Image

In the following sections, we touch on the steps to create a VPC and an OS image. While you are in the process, collect the following information, and save it to a file:

Table 5-14: Scenario Parameters for OpenStack

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Cloud account	Name of your account at cloud vendor	String
Region	Region selected during configuration at cloud vendor	Drop-down list
Domain name	Domain used by PBS Cloud module head node	String
Hostname prefix	Optional prefix for burst node names; default is "node"; chosen during configuration at vendor	String

Table 5-14: Scenario Parameters for OpenStack

Scenario Parameter	What to Collect During Configuration at Vendor	Format
Subnet ID	ID of subnet for VPC created at cloud vendor	String
Security Group ID	ID of security group created at cloud vendor	String
OS Image URI	ID of OS image created at cloud vendor	String

5.8.2.1 Create a Virtual Private Cloud

Create a virtual private cloud using the Stein release of OpenStack:

1. Choose a region, a name, and an address range for the VPC.
2. Create a subnet and choose a name and an address range for it.
3. Create a security group and choose a name for it.
4. Add an inbound rule:
   - Use TCP
   - Use port 22
   - Choose IP Address
   - Specify your PBS Cloud firewall IP address

     Warning: 0.0.0.0/0 enables all IPv4 addresses to access your instance. ::/0 enables all IPv6 addresses to access your instance. This is acceptable for a short time in a test environment, but it's unsafe for production environments. In production, authorize only a specific IP address or range of addresses to access your instance.

5.8.2.2 Create a Virtual Machine

Create a virtual machine using the instructions for your cloud. We recommend the following:

- Create a key pair, give it a name, and save the .pem file containing the key pair to your local disk in a secure location. PBS Cloud will use this later to manage cloud nodes.
- Choose the same region you chose for the VPC.
- Choose an availability zone, if you have more than one
- Choose a type that fits your needs.
- Use a public image.
- Choose a supported OS
- Choose the disk characteristics you need.
- Assign the VPC you created.
- Assign the security group you created.
- Choose the Key Pair login mode.
- For the key pair, choose the .pem file you generated.
- Give the VM a name.
- Select a quantity of 1 for the number of nodes.

  It may take some time to create the virtual machine.

5. You can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.8.2.3 Install and Configure a PBS MoM on the Virtual Machine

1. Get the public IP address of the VM.
2. Log in to your site's PBS server host.
3. SSH into the VM using the default user "root", the .pem file you generated when creating the VM and the external IP address assigned to the VM.

   ssh -i /<path to .pem>/<.pem filename>.pem root@<public IP of VM>
4. Using the PBS Professional Installation and Upgrade Guide, install and configure the PBS MoM.
5. Configure the VM to work with your site environment, for example mounting file systems, connecting to the authentication service, installing any applications you need, etc. We recommend that you include the following:
   - Mount /home in the VM
   - Either install applications in the VM or crossmount them from the head node
   - Either add users to the password file or connect the VM to a service such as NIS
6. If cloud-init is not installed, install it.

5.8.2.4 Create OS Image from VM

Follow your cloud instructions to create an OS image from the VM you created. Use the same region you chose for the VPC.

On Azure, you can now delete the virtual machine so that you are no longer charged for it.

On AWS, you can delete the virtual machine now to avoid storage costs, or keep the virtual machine and update it over time to create updated OS images for bursting. You will incur storage costs, but this is an effective way to keep your OS images up to date when there are changes in packages, patches, or applications.

5.9 Windows Bursting on AWS and Azure

Bursting of Windows virtual machines is supported on AWS and Azure. Windows cloud bursting is similar to cloud bursting on Linux platforms. Three special requirements are necessary to burst Windows cloud nodes.

5.9.1 OS Image Name

When creating the OS image, the name of the image must contain the term "windows" (case insensitive). For example, on AWS, the AMI Name should look something like this:

```
Windows_Server-2012-R2_RTM-English-64Bit- Base-2019.11.13
```

On Azure, the Image Name should look something like Windows Server 2012 R2 Datacenter.

5.9.2 Inbound Security Rule for RDP

Secondly, an inbound rule to open the port 3389 must be added to the AWS security group or the Azure network security group that is associated with the cloud provider virtual network. This allows a connection to be made to the Windows VM via RDP so that the PBS MoM can be installed.

For more information see AWS: Authorizing Inbound Traffic for Your Windows Instances and Azure: Cannot connect remotely to a VM because RDP port is not enabled in NSG.

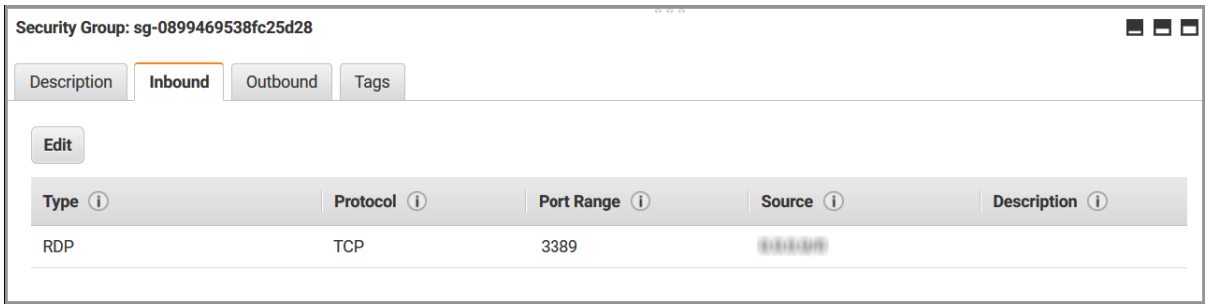


Figure 5-33:AWS Inbound Security Rule for RDP

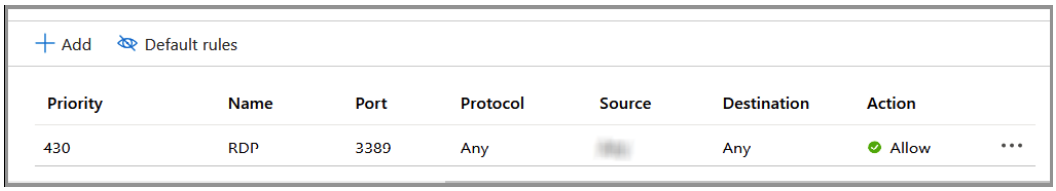


Figure 5-34:Azure Inbound Security Rule for RDP

5.9.3 Startup Script

The cloud node startup script must use a PowerShell script. For more information see PowerShell Scripting. The below PowerShell script example generates a file in C:\Windows\Temp:

```
<powershell>
$file = $env:SystemRoot + "\Temp\" + (Get-Date).ToString("MM-dd-yy-hh-mm")
New-Item $file -ItemType file
</powershell>
```

5.9.4 See Also

- "Configuring Amazon Web Service Cloud Bursting" on page 43
- "Configuring Microsoft Azure Cloud Bursting" on page 54

6 The Cloud Node Startup Script

6.1 Introduction

For each scenario, you will want to do some configuration of each cloud node after it boots. The simplest way to configure an instance at boot time is to use a shell script. You can add a startup script to each bursting scenario that runs when a scenario instance boots, in order to perform automated tasks to customize your cloud nodes. For example, you may want to install packages, perform updates, start services, add users, configure filesystems (`/etc/fstab`), configure NIS (`/etc/yp.conf`), or mount necessary filesystems.

The cloud-init tool is a utility for initializing cloud instances. PBS Cloud uses the cloud-init tool to launch its own startup script first to configure each freshly burst cloud node so that the PBS server can connect to the cloud node. If you use a startup script, it runs after the built-in script.

For more information on cloud-init, see the cloud-init documentation at cloud-init.

Note that you may not need a startup script. If you configure the VM that you use to create your OS image to have everything you need to run jobs, you do not need a startup script.

6.1.1 Making cloud-init Tool Available in OS Image

For each scenario, you create an OS image that includes the cloud-init tool. Make sure that cloud-init is installed in the VM that you use to create the image (we include that step in the instructions). Sometimes it is pre-installed; if it's not there, install it. For example, on CentOS:

```
yum install -y cloud-init
```

6.1.2 Adding a cloud-init Script to a Scenario

You add a cloud-init script to each scenario via the PBS Cloud web interface; see section 3.3.4.9, "Specifying the Cloud Node Startup Script", on page 26. You can add a script while creating a scenario, and you can edit the script or choose a new script later. The startup script to be included in a scenario can be located anywhere that your PBS Cloud web interface can browse to. Once a script has been added to a scenario, the script is stored in the PBS Cloud database.

Make sure you develop the startup script for each scenario. See section 6.2.5, "Developing the Startup Script", on page 111.

6.1.3 Startup Script Prerequisites

- The startup script must run using a shell or language available in the freshly burst node. For example, if you have `bash` and Python available, your script can use `bash`, or it could use a `bash` script to launch a Python script.
- On Windows cloud nodes, use a PowerShell startup script. Enclose the content of the PowerShell script in <powershell> and </powershell>. Refer to Microsoft documentation for more information about PowerShell.
- The startup script can have any name.

6.1.4 Startup Script Recommendations

- We strongly recommend mounting /home to simplify making each cloud node usable for jobs.
- We recommend installing your applications somewhere and then mounting that directory in each cloud node.

6.2 Customizing Your Startup Script

Make sure that your startup script uses the correct locations for PBS_HOME, PBS_EXEC, etc.

6.2.1 Mounting /home Directory

We strongly recommend that you mount /home so that users can run jobs in the cloud node. This makes user SSH keys available, and /home can be used for PBS data transfer. If you do not mount /home, you have to create a /home directory in the image, pre-fill it for each user who will submit jobs, and set up the user space.

```
echo "<PBS server host IP address> <head node> <head node>.<domain name>" >> /etc/hosts
...
...
yum install -y nfs-utils
mount -t nfs <hostname of machine with /home>:/home /home
```

6.2.2 Configuring MoM for Local Copy

We recommend that you set the $usecp MoM configuration parameter to tell the MoM which local directories are mapped to mounted directories, so that MoM can use the local copy mechanism for them:

```
echo "<PBS server host IP address> <head node> <head node>.<domain name>" >> /etc/hosts
...
...
echo "\$usecp <hostname of machine with /home>:/home/ /home/" >> /var/spool/pbs/mom_priv/config
```

See "Configuring MoM for Local Copy" on page 561 in the PBS Professional Administrator's Guide.

6.2.3 Creating Local Scratch Space

You can create local scratch on a fast local disk and use it as the default location to run jobs, if PBS jobs have the sandbox attribute set to *PRIVATE* and $jobdir_root is set to /scratch. See "The sandbox Job Attribute" on page 524 in the PBS Professional Administrator's Guide.

```
mkdir /scratch
chmod 1777 /scratch
echo "\$jobdir_root /scratch" >> /var/spool/pbs/mom_priv/config
```

See "Example of Setting Location for Creation of Staging and Execution Directories" on page 526 in the PBS Professional Administrator's Guide.

To make it so that jobs with sandbox=*PRIVATE* have their staging and execution directories created under /scratch, as /scratch/<job-specific directory>, put the following line in MoM's configuration file:

```
$jobdir_root /scratch
```

When you set $jobdir_root to a shared directory, tell MoM it is shared by setting the *shared* directive after the directory name:

$jobdir_root <directory name> shared

Otherwise sister MoMs can prematurely delete files and directories when nodes are released. This is because when a job's sandbox attribute is set to *PRIVATE* and $jobdir_root is set to a shared directory, PBS can use a shared location for job files. When sister nodes are released, those sister MoMs would normally clean up their own files upon release. So if $jobdir_root is set to a shared directory, you need to tell the sister MoMs not to do the cleanup, and let the primary execution host MoM clean up when the job is finished.

6.2.4 Example cloud-init Startup Script for Linux

Example 6-1: cloud-init startup script for Linux. Do not use this script as is; make sure you adapt it for your site.

```
#!/bin/sh
# Map PBS server host IP address to hostnames via /etc/hosts
echo "/etc/hosts setup"
rm -f /etc/hosts
echo "127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4" > /etc/hosts
echo "<PBS server host IP address> <PBS server host> <PBS server host>.<cloud node domain>" >>
    /etc/hosts

# Disable NetworkManager and use network interface
# so that it does not overwrite the /etc/resolv.conf file
systemctl disable NetworkManager
systemctl stop NetworkManager
systemctl enable network
systemctl start network

# Configure PBS via /etc/pbs.conf
echo "PBS setup"
systemctl stop pbs
rm -f /etc/pbs.conf
echo "PBS_EXEC=/opt/pbs/default" > /etc/pbs.conf
echo "PBS_HOME=/var/spool/PBS" >> /etc/pbs.conf
echo "PBS_START_SERVER=0" >> /etc/pbs.conf
echo "PBS_START_MOM=1" >> /etc/pbs.conf
echo "PBS_START_SCHED=0" >> /etc/pbs.conf
echo "PBS_START_COMM=0" >> /etc/pbs.conf
echo "PBS_SERVER=PBS_SERVER_HOSTNAME" >> /etc/pbs.conf
echo "PBS_CORE_LIMIT=unlimited" >> /etc/pbs.conf
echo "PBS_SCP=/bin/scp" >> /etc/pbs.conf

# Update pbs.conf with the cloud node's IP address
IP=$(ip addr show eth0 | grep "inet\b" | awk '{print $2}' | cut -d/ -f1)
echo "PBS_MOM_NODE_NAME=$IP" >> /etc/pbs.conf

# Configure the MoM by updating PBS_HOME/mom_priv/config
echo "MoM configuration setup"
. /etc/pbs.conf
echo "\$clienthost $PBS_SERVER" >> /var/spool/pbs/mom_priv/config
echo "\$clienthost ${PBS_SERVER//.*}" >> /var/spool/pbs/mom_priv/config
echo "\$restrict_user_maxsysid 999" >> /var/spool/pbs/mom_priv/config

# Restart PBS
systemctl start pbs
```

6.2.5 Developing the Startup Script

Troubleshooting issues with cloud bursting can be difficult and time consuming. Issues that can arise include the following:

- Network issues
- SSH key issues
- Missing users or groups
- Missing packages
- Mounted file system errors

6.2.5.1 Prerequisites for Developing a Startup Script

- You have installed PBS Professional and PBS Cloud, configured PBS Professional, created a cloud administrator account at your cloud provider and added that account to PBS Cloud, created and configured your cloud provider components including an OS image, and configured the PBS Cloud module. These steps are outlined in section 3.1, "Overview of Configuring PBS Cloud", on page 11.
- Make sure the cloud bursting hook is disabled.
- You should have at least one PBS Cloud scenario to test.
- This scenario must be enabled.
- The Add Public IP to VMs scenario option is enabled; see section 3.3.4.4, "Temporarily Adding Public IP for Debugging", on page 23
- The SSH keys parameter has an administrator SSH key; see section 3.3.4.8, "Adding SSH Key for Access to Burst Nodes", on page 26
- You have the corresponding private key
- Port 22 in the vendor firewall has to be open (already covered when you were configuring vendor components)

6.2.5.2 What to Test For

- You can log into the node
- The `/home` directory is mounted
- The directory containing your applications is mounted
- Make sure applications work
- Make sure the node was added to PBS (use `pbsnodes -a`)
- The cloud node can be reached from the PBS server host
- The cloud infrastructure (VPN connectivity, networks, firewalls etc.) is working

6.2.5.3 Steps to Develop the Startup Script

We recommend that you burst a single cloud node, test it, and resolve one issue at a time. Each time you resolve an issue, keep a log of your changes, and incorporate your solution into the cloud-init script. You can log your command-line input as you work to resolve the issue, and convert this record into script inputs.

Once the cloud node is configured correctly, you can enable the cloud bursting hook, use it to burst cloud nodes, and troubleshoot the cloud bursting hook.

6.2.5.4 Example of Developing a cloud-init Script

This example is intended to be used after you have installed PBS Professional and PBS Cloud, configured PBS Professional, created a cloud administrator account at your cloud provider and added that account to PBS Cloud, created and configured your cloud provider components including an OS image, and configured the PBS Cloud module. These steps are outlined in section 3.1, “Overview of Configuring PBS Cloud”, on page 11.

Make sure the cloud bursting hook is disabled.

You should have at least one PBS Cloud scenario to test. This scenario must be enabled.

1. If you have not already prepared the scenario for developing a startup script:
   a. Create a cloud-init script by copying the example cloud-init script
   b. In the script, map the PBS server host to its IP address:

   ```
   echo "<PBS server host IP address> <PBS server hostname> <PBS server hostname>.<cloud node domain name>" >> /etc/hosts
   ```

   c. Make any other necessary changes to the script
   d. Upload the cloud-init script to the scenario. Enable the Add public IP to VMs option for the scenario, and add a suitable public SSH key to the scenario. Make sure you have the corresponding private key.

1. Log in to PBS Cloud.
2. Click the Cloud tab.
3. Under Infrastructure, click Bursting.
4. Select a bursting scenario by clicking on its name.
5. Under Machines (manually burst), click Manual bursting.
6. Choose the instance type and OS. Set the number of cloud nodes to be 1.
7. Once the cloud node is burst, attempt to SSH into the cloud node using the value of the private_ip parameter that is returned from the PBS Cloud CLI bursting command.
8. If you encounter a "Permission denied" error, there is a problem with the cloud node configuration and/or the cloud-init script. Each user's SSH keys are stored in the user's home directory under the .ssh directory and the public key has been added to `.ssh/authorized_keys` in the user's home directory, so SSH should be working. .
9. Using PuTTY or a similar SSH client, SSH into the cloud node using the cloud node's public IP address (the value of the parameter public_ip address returned from the PBS Cloud bursting command) and your previously generated private SSH key (matching the public key used in the bursting scenario).
10. Check the contents of the `/etc/hosts` file. If the PBS server hostname is mapped to its IP address, the cloud-init script is being run.
11. Check which filesystems have been mounted using the `df` command. If `/home` is not mounted, this is the cause of the SSH failure to the private IP address of the cloud node.
12. Use the `ls` command to see whether `/home` exists.
13. To mount the directory, install `nfs-utils` and then mount `/home`:

    ```
    yum install nfs-utils
    mount -t nfs headnode:/home /home
    ```

1. Log in to PBS Cloud.
2. Click the Cloud tab.
3. Under Infrastructure, click Bursting.

4. Select a bursting scenario by clicking on its name.
5. Under Machines (manually burst), click Manual unbursting page.
6. Enabling the check box to the right of the cloud node.
7. Click Unburst.
8. Click Unburst machines to confirm the action.
9. Edit the bursting scenario.
10. Edit the cloud-init script.
11. Copy the two commands to install `nfs-utils` and mount `/home` and paste them into the cloud-init script.
12. Save the cloud-init script.
13. Burst another cloud node and repeat the process.

6.2.5.5 Caveats for Testing Startup Script

Occasionally you will burst a node that appears to be correct in all respects, but it simply won't run a job. Sometimes you just get a bad node. In this case, unburst the node and burst a new one.

7 Managing Cloud Bursting

7.1 Managing Cloud Bursting

7.1.1 Viewing Cloud Account Details

1. Log in to PBS Cloud.
2. Click the Cloud tab.
3. Under Infrastructure, click Cloud.
4. Select a cloud account by clicking on its name.

   The information that was entered to create the cloud account is displayed.
5. Click Close.

7.1.2 Viewing Burst Cloud Nodes

Information that is displayed about cloud nodes:

Machine Name
: Hostname of the node.

IP Address
: IP address assigned to the node.

Instance Type
: Cloud provider instance type (machine type, shape or flavor) of cloud node.

Image
: OS image used to burst the node.

1. Log in to PBS Cloud.
2. Click the Cloud tab.
3. Under Infrastructure, click Bursting.
4. Select a bursting scenario by clicking on its name.

   Any cloud nodes that have been burst are displayed under the Machines category.

Machines

Machine Name	IP Address	Instance type	Image
mcr10-12-3-223	10.12.3.223	t2.micro	ami-b40efcce

Figure 7-1:Burst Cloud Nodes

7.1.3 Enabling or Disabling a Bursting Scenario in PBS Cloud

Each bursting scenario is enabled by default when you create it.

1. Log in to PBS Cloud.
2. Click Cloud.
3. Click the Bursting tab on the left-hand side of the web page.
4. Select the name of the bursting scenario.
   - To enable the bursting scenario, click Enable.
   - To disable the bursting scenario, click Disable.

7.1.4 Disabling Bursting for a Scenario and Queue

How you stop the bursting process depends on whether the remaining jobs in the scenario queue can run using either the existing or edited scenario.

- If the jobs in the queue can run using the changed scenario:
  a. Stop the cloud queue associated with the scenario (prevent jobs in the queue from starting):

     ```
     qmgr -c "set queue <queue name> started=false"
     ```

  b. Allow the burst nodes for that scenario to drain and unburst, or requeue them
- If the jobs in the cloud queue must use the existing scenario:
  a. Disable the cloud queue (prevent jobs being enqueued):

     ```
     qmgr -c "set queue <queue name> enabled=false"
     ```

  b. Log in to PBS Cloud.
  c. Click Cloud.
  d. Click the Bursting tab on the left-hand side of the web page.
  e. Under Infrastructure, click Bursting.
  f. Select the name of the bursting scenario.
  g. Click Disable.
  h. Allow the cloud queue associated with the bursting scenario to drain; allow time for the jobs that are waiting in the queue to run and finish; allow automated unbursting to finish. You can shorten the time to unburst using the PBS Cloud interface or the PBS Cloud CLI.

Verify that all cloud nodes are unburst:

  a. Click on the name of the cloud bursting scenario.
  b. Look under the Machines heading. The following message indicates that all cloud nodes are unburst:

     ```
     No machines are available
     ```

7.1.5 Re-enabling Bursting for a Scenario and Queue

How you start the bursting process depends on how you stopped it.

- If you stopped the queue to prevent jobs from starting, start the cloud queue associated with the scenario (allow jobs to start):

  ```
  qmgr -c "set queue <queue name> started=true"
  ```

- If you disabled the queue to prevent jobs from being enqueued:

  a. Enable the cloud queue (allow jobs to be enqueued):

  ```
  qmgr -c "set queue <queue name> enabled=true"
  ```

  b. Log in to PBS Cloud.

  c. Click Cloud.

  d. Click the Bursting tab on the left-hand side of the web page.

  e. Under Infrastructure, click Bursting.

  f. Select the name of the bursting scenario.

  g. Click Enable.

7.2 Starting and Stopping PBS Cloud

7.2.1 Start PBS Cloud

Starting PBS Cloud must be done as root or as a user with sudo permissions using the sudo command.

When your server hosting the PBS Cloud component reboots, containers are restarted automatically. If you need to manually start PBS Cloud containers, please follow the below instructions.

1. Log in to the machine where PBS Cloud is installed.
2. Enter the following command to start PBS Cloud:

   ```
   pkr start
   ```

7.2.2 Stop PBS Cloud

Stop PBS Cloud after a manual installation.

Stopping PBS Cloud must be done as root or as a user with sudo permissions using the sudo command.

1. Log in to the machine where PBS Cloud is installed.
2. Enter the following command to stop PBS Cloud:

   ```
   pkr stop
   ```

7.2.3 Restart PBS Cloud

Restarting PBS Cloud must be done as root or as a user with sudo permissions using the sudo command.

When your server hosting the PBS Cloud component reboots, containers are restarted automatically. If you need to manually restart PBS Cloud containers, please follow below instructions.

1. Log in to the machine where PBS Cloud is installed.
2. Enter the following command to restart PBS Cloud:

```
pkr restart
```

7.2.4 Determine the Status of PBS Cloud

Determine whether PBS Cloud is up or down.

1. Log in to the machine where PBS Cloud is installed.
2. Enter the following command to display the status of PBS Cloud:

```
pkr status
```

7.2.5 Monitoring Logs and Workflows

PBS Cloud includes a Loki interface for monitoring logs and workflows. To use this, log into PBS Cloud, then choose *Monitoring*, *Logs*, or *Workflows*.

The Monitoring section shows the status of the cloud services, including how many nodes are in the cloud, etc.

The Logs section shows all the PBS Cloud logs in chronological order.

The Workflows section shows you specific information about each workflow stage.

7.3 Logging into PBS Cloud

To log into PBS Cloud, go to the PBS Cloud interface in your web browser:

```
http://<PBS Cloud hostname or IP address>:<port>/pbspro-cloud/#/login
```

The default port is 9980.

7.4 Troubleshooting Cloud Bursting

Log messages can help you troubleshoot cloud bursting:

- Check log messages written to `PBS_HOME/server_logs` on the PBS server host.
- For debugging issues with node creation or issues with starting MoM on the cloud node, SSH to the cloud node and check `PBS_HOME/mom_logs`. This requires:
  - The Add Public IP to VMs scenario option is enabled; see section 3.3.4.4, “Temporarily Adding Public IP for Debugging”, on page 23
  - The SSH keys parameter has an administrator SSH key; see section 3.3.4.8, “Adding SSH Key for Access to Burst Nodes”, on page 26
- You can use PBS Cloud to view the logs through a Loki interface; see section 7.2.5, “Monitoring Logs and Workflows”, on page 118

7.4.1 PBS MoM is Stopped or Down

When you are using cloud bursting and all PBS MoMs are stopped or down, you may find error messages similar to the following in the PBS Server logs:

```
Server@server;Hook;Server@server;CLBR: Error: /opt/pbs/bin/pbsnodes: Server has no node list
Server@server;Hook;Server@server;CLBR: Error: Failed to get nodes info
```

Resolve the issue by starting at least one MoM.

8 Managing Cloud Jobs

8.1 Manage Cloud and On-Premise Jobs

We present solutions to handle various job distribution needs.

Your site may want to run certain types of jobs on-premises or in the cloud. PBS Professional provides various methods to collect and distribute jobs. For more information see Routing Jobs in the PBS Professional Administrator's Guide. Here are some solutions to handle various job distribution needs.

8.1.1 Associate Nodes with Queues

In general, start by associating nodes with queues, so that a job in a specific queue is sent to the appropriate nodes. Associate each on-premise vnode with the appropriate local queue, and associate each bursted vnode with the appropriate cloud queue; see "Associating Vnodes with Queues" on page 103 in the PBS Professional Administrator's Guide.

8.1.2 Use Hooks or Routing Queues to Send Jobs to the Appropriate Queue

To send jobs to the appropriate queue on submission, use hooks or routing queues. For more information see the *PBS Professional Hooks Guide* and "Routing Queues" on page 25 in the PBS Professional Administrator's Guide.

8.1.3 Job Distribution Examples and Solutions

8.1.3.1 Send Small Jobs to the Cloud

You have big machines for on-premise nodes and want to reserve those big machines for big jobs. You want to send smaller jobs to the cloud.

Solution 1:

- Create three queues: a routing queue, a local queue, and a cloud queue.
- Use the routing queue to collect jobs on submission.
- Set resource gating on the local queue to filter out smaller jobs.
- Allow smaller jobs into the cloud queue.

Solution 2:

- Create two queues: a local queue, and a cloud queue.
- Use a queuejob hook to route jobs into the appropriate queue.

8.1.3.2 Send Specific Jobs Only to the Cloud

You want to send specific jobs to the cloud because:

- an application needed by some jobs runs well in the cloud.
- a resource that is available in the cloud is not available locally.
- a department has exhausted its share of local resources, and wants to send its jobs to the cloud.

Solution :

- Create two queues: a local queue, and a cloud queue.
- Use a queuejob hook to route jobs into the appropriate queue.

8.1.3.3 Charge Departments for Resources Used

You have multiple departments and each department should be charged for the resources it uses.

Solution 1:

- Consider using PBS Budget to monitor consumption of on premise and cloud resources

Solution 2:

- Create a cloud queue for each department
- Set separate limits on each cloud queue

8.2 Run Cloud Jobs On-Premise Before Bursting

Use placement sets to run jobs on-premise first and burst cloud nodes when local nodes are not available.

Assign a bursting scenario to an on-premise vnode so that cloud jobs are run on-premise when there is capacity. Cloud nodes are only burst when on-premise nodes are not available.

For more information on placement sets, see "Placement Sets" on page 168 in the PBS Professional Administrator's Guide.

1. Create a custom resource to indicate that a node is on-premise.

   ```
   qmgr -c "create resource node_location type=string,flag=h"
   ```

2. For each on-premise vnode, set the resources_available.node_location resource to "local" to indicate that it is on-premise.

   ```
   qmgr -c "active node <vnode name>"
   qmgr -c "set node <vnode name> resources_available.node_location=local"
   ```

3. For any on-premise vnodes where you want to run cloud jobs, associate a bursting scenario or a list of bursting scenarios to the vnode:

   - To associate a list of bursting scenarios with a vnode:

     ```
     qmgr -c "set node <vnode name> resources_available.cloud_scenario= SCENARIO_1,
         SCENARIO_2,...,SCENARIO_n"
     ```

   - To associate a single bursting scenario with a vnode:

     ```
     qmgr -c "set node <vnode name> resources_available.cloud_scenario= SCENARIO_1"
     ```

   Where SCENARIO_1,..., SCENARIO_n are scenario names that have been added to the cloud bursting hook configuration file.

   ```
   qmgr -c "set node <vnode name> resources_available.cloud_scenario=
       azure_scenario_1,aws_scenario_1,aws_scenario_2"
   ```

4. Create a dummy vnode with a high resources_available.ncpus value and unset resources_available.node_location.
5. Mark the dummy node as offline so that jobs cannot run on it.

   pbsnodes -C "<comment>" -o HOSTNAME

   Where <comment> is a string explaining that this is a dummy node for cloud bursting and HOSTNAME is the hostname of the vnode.
6. Enable placement sets:

   qmgr -c "set server node_group_enable=true"
   qmgr -c "set server node_group_key=node_location"
7. Allow creation of placement sets from unset resources:

   qmgr -c "set sched only_explicit_psets=false"
8. Optionally, prevent any single job from running on both local and cloud vnodes:

   qmgr -c "set sched do_not_span_psets=true"
9. Navigate to PBS_HOME/sched_priv.
10. Edit the sched_config file.
11. Add cloud_scenario and node_location to the resources line:

    "**... ,** cloud_scenario, node_location"
12. For each local (non-cloud) queue set its node_location to "local":

    qmgr -c "set queue <queue name> resources_default.node_location=local"

    Where <queue name> is the name of the local queue.
13. For each cloud queue, verify that the value of resources_default.node_location is unset.
14. Restart or HUP the scheduler:

    kill -HUP <scheduler PID>

8.2.1 Override Instance Type or Image at Job Submission

The OS image that is to be used when a cloud node is burst can be specified at job submission via the `qsub` command. If this information is not provided at job submission, then the OS image defaults to the value of the cloud_default_image parameter in the cloud bursting hook configuration file.

The job submitter can use only the instance types that have been allowed by the administrator. If the job submitter tries to use a non-allowed instance type, the job does not run.

Additionally, the instance type that is burst can be specified at job submission via the `qsub` command (e.g. Standard_DS3_v2). Instance types are defined by the cloud provider. The name of the instance type must exactly match the cloud provider's. If this information is not provided at job submission, then the instance type defaults to the instance type defined for the cloud queue.

1. Override the cloud image at job submission via the `qsub` command:

   qsub -l select=1:ncpus=4 -q <queue name> -v CLOUD_IMAGE=<OS image> <job script>

   Where:

   - IMAGE is the name of the OS image to be used when the node is burst.
2. Override the cloud instance at job submission via the `qsub` command:

   qsub -l select=1:ncpus=4 -q <queue name> -v CLOUD_INSTANCE=INSTANCE_TYPE <job script>

Where:

- INSTANCE_TYPE is the name of cloud provider instance type. This is the type of machine that will be burst.

3. You can also override both the OS image and the instance type as part of the qsub command:

```
qsub -l select=1:ncpus=4 -q <queue name> -v CLOUD_IMAGE=<OS image>,CLOUD_INSTANCE=INSTANCE_TYPE
    <job script>
```

8.2.2 Request InfiniBand Nodes

You will need to create an InfiniBand enabled OS image before you can submit a job to an InfiniBand cloud node.

When the user wants to run HPC workloads like MPI jobs, the user needs InfiniBand supported nodes deployed on the same InfiniBand network. Among the cloud providers, only Azure currently supports InfiniBand nodes. Azure limits the number of nodes on a single Infiniband network (current default limit is 100), however your site may set up multiple Infiniband networks. A restriction for bursting on Infiniband networks is that PBS jobs cannot run across multiple Infiniband networks. To prevent this from happening, placement sets can be used.

At job submission, the user can request InfiniBand nodes by requesting an InfiniBand supported instance type, an InfiniBand enabled OS image, and an InfiniBand network.

Upon bursting an InfiniBand node, the name of the network is obtained from the cloud provider. The custom resource cloud_network on the Infiniband node is updated with this value. All nodes on the same Infiniband network have this custom resource set to the same value. This value is then used to restrict jobs to run within the same network, using placement sets.

1. Enable placement sets:

```
qmgr -c "set server node_group_key=cloud_network"
qmgr -c "set server node_group_enable=true"
```

2. Prevent any single job from running across multiple Infiniband networks:

```
qmgr -c "set sched do_not_span_psets=true"
```

3. Request Infiniband nodes by overriding the cloud image and the cloud instance and requesting an Infiniband network via the qsub command:

```
qsub -l select=1:ncpus=4 -q <queue name> -v CLOUD_IMAGE=<OS
    image>,CLOUD_INSTANCE=INSTANCE_TYPE,CLOUD_NETWORK=IB <job script>
```

Where:

- INSTANCE_TYPE is the name of cloud provider instance type (machine types, shapes or flavors) and should be an instance type that supports Infiniband.
- IMAGE is the name of the Infiniband enabled OS image to be used when the node is burst.
- CLOUD_NETWORK=IB requests an Infiniband network
- JOB_SCRIPT is the name of the script to be executed.

9 Running Cloud Jobs

9.1 Introduction

Jobs are submitted to one or more designated cloud queues, and a server periodic hook monitors the cloud queues, estimates the demand for cloud nodes, and dynamically adjusts the number of nodes by bursting or unbursting as needed.

You can choose any of the instance types allowed by your PBS Cloud administrator. To request an instance type:

```
CLOUD_INSTANCE=<instance type>
```

You can choose an OS image:

```
CLOUD_IMAGE=<name of image file>
```

You can request preemptable instances, including spot instances. When requesting cloud nodes, the request should be for either non-preemptable instances or preemptable instances, but not both. Do not request some cloud nodes that are on-demand and some that are preemptable or spot.

9.2 Sample Job Script: Cloud Job

The following script is simply a job requesting 10 minutes of walltime, it will sleep for 1 minute (or tune $sleeptime as appropriate) and then exit. It requests cloudq; adjust the name depending on your site configuration. You can save the following job script as sleep.sh. Then you can submit it to PBS:

```
qsub sleep.sh
```

9.2.1 Contents of Sample Cloud Job Script

```
#!/bin/bash
#PBS -N testjob
#PBS -j oe
#PBS -m n
#PBS -q cloudq
#PBS -l select=1:ncpus=2:mem=16mb
#PBS -l walltime=0:10:00
sleeptime=60
cmd="sleep $sleeptime"
echo $cmd
$cmd
exit
```

9.2.2 Viewing Job Output

When the job completes you should see the job's output. This will appear where the job was submitted.

9.3 Logging into PBS Cloud

To log into PBS Cloud, go to the PBS Cloud interface in your web browser:

```
http://<PBS Cloud host name or IP address>:<port>/pbspro-cloud/#/login
```

The default port is *9980.*

10 Example Azure Head Node

10.1 Example Configuration of Cloud Head Node in Azure

Here we show an example of how to configure a cloud head node using Azure:

1. Create a new VM based on Centos 7, using Rogue Wave Software's CentOS-based 7.6 image, not an HPC tagged version
   - Use Standard_D14 instance type for head node (Good price/performance option)
   - Add additional storage: 1000GB SSD to allow an ext4 volume to be added; the Centos image uses XFS filesystem which has a bug with docker
2. Provide a public ssh key for the username "centos" via Azure GUI

   This is the only default user in this image
3. Once the node is up, use the provided external IP and your private key to connect to it. PuTTY is more robust for connection to cloud than Moba xTerm
4. Unless noted otherwise all the following commands must be run as root. Use sudo or the following to switch to the root user:

   ```
   sudo su -
   ```
5. Use `cfdisk` to partition `/dev/sdc` using all space for a single volume. Make absolutely sure that `/dev/sdc` is the right target. The target depends on instance type chosen and how many disks are added; if you followed the procedure exactly this should be correct in this case, but formatting is destructive to data on the disk.

   ```
   cfdisk /dev/sdc
   ```
   a. Select New
   b. Select Primary
   c. Accept Default Size (Should be whole disk)
   d. Select Write
   e. Answer yes (You will need to type yes)
   f. Select Quit
6. Create a filesystem on your drive (ext4 preferred):

   ```
   mkfs -t ext4 /dev/sdc1
   ```
7. Make a folder to mount your new volume to:

   ```
   mkdir /data
   ```
8. Set suitable permissions on the folder:

   ```
   chmod 777 /data
   ```

9. Find UUID from /dev/disk/by-uuid, e.g.

```
lrwxrwxrwx 1 root root 10 Apr 1 12:48 640a03fd-aa69-4f8d-98d5-2f0d3d12bb26 -> ../../sdb1
lrwxrwxrwx 1 root root 10 Apr 1 12:48 a505f591-5a7d-424f-a5a1-06dcb72f944c -> ../../sda1
lrwxrwxrwx 1 root root 10 Apr 1 12:48 c6cd262b-3930-48a8-9b21-8981bb479cee -> ../../sdc1
lrwxrwxrwx 1 root root 10 Apr 1 12:48 e0d6ff47-4c69-4a4c-b44a-13ea19d80f96 -> ../../sda2
```

10. Add a line to end of /etc/fstab for this mount (Note: ID of /dev/sdc1 above):

```
UUID=c6cd262b-3930-48a8-9b21-8981bb479cee /data ext4 defaults 0 0
```

11. Mount the new filesystem:

```
mount /data
```

12. Upgrade all system packages to the latest versions:

```
yum upgrade
```

13. Disable SELinux by editing /etc/selinux/config and changing, then save:

```
SELINUX=enforcing -> SELINUX=disabled
```

14. Disable and stop firewalld

```
systemctl disable firewalld
systemctl stop firewalld
```

15. SELinux is still enforcing, so prevent that. Reboot.

16. Add password to user "centos":

```
passwd centos
```

17. Log in as centos

18. Create ssh keys for centos and enable passwordless ssh

    Accept all the defaults for ssh-keygen step:

```
cd $HOME
ssh-keygen
cd .ssh
cat id_rsa.pub >> authorized_keys
```

19. For a cloud head node, make sure you validate that external ssh still works externally, before you disconnect your first session. If there is an issue with ssh keys, which can be sometimes caused by errors in step 18 above, and you disconnect your first session, you could be permanently locked out of your cloud VM. Use ssh from your local machine to your cloud head node to create a second session using your public ssh key. Troubleshoot as required while your first session is still up.

20. Add the key PBS Professional service users:

```
useradd -rm pbsdata; useradd pbsadmin
```

21. Follow steps 17, 18, and 19 for user pbsadmin

22. Install/Start/Enable docker-ce

```
yum install -y yum-utils
yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
yum install docker-ce docker-ce-cli containerd.io
systemctl enable docker
systemctl start docker
```

23. Stop docker:

    systemctl stop docker

24. Move docker data storage to new filesystem mounted on /data

25. Add -g /data/docker \ in file /lib/systemd/system/docker.service. ExecStart should look like this:

    ```
    ExecStart=/usr/bin/dockerd-current ...
    --seccomp-profile=/etc/docker/seccomp.json -g /data/docker $OPTIONS ...
    $REGISTRIES
    ```

26. Reread daemon config files:

    systemctl daemon-reload

27. Look in /data/docker; there should be subdirectories for docker data

28. Start docker:

    systemctl start docker

29. Create a repository for PBS Package

    ```
    mkdir /home/centos/software
    chown -R centos:centos /home/centos/software
    chmod 777 /home/centos/software
    ```

30. Download all the PBS package modules to your software directory (PBS Professional, PBS Cloud)

31. Install and configure NFS Server to share /home from head node, so you can avoid setting up ssh keys for client nodes and have a convenient mount:

    ```
    yum install nfs-utils
    systemctl enable nfs-blkmap
    systemctl enable nfs-rquotad
    systemctl enable nfs-server
    systemctl enable nfs
    systemctl start nfs-blkmap
    systemctl start nfs-rquotad
    systemctl start nfs-server
    systemctl start nfs
    ```

32. Edit /etc/exports to add

    ```
    /home *(rw,sync,no_root_squash,no_all_squash)
    ```

33. Restart NFS Server:

    systemctl restart nfs-server

34. Add local machine name and IP to /etc/hosts, remove IPv6 loopback, e.g.

    ```
    127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
    <PBS server IP address> <PBS server hostname>
    ```

    This would look like:

    ```
    127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
    172.17.0.6 myhost
    ```

35. Follow the *PBS Professional Installation & Upgrade Guide*, the *PBS Professional Administrator's Guide,* and the *PBS Professional Licensing Guide* to install, configure and license PBS Professional

    - Set PBS to have MoM running on the head node before first start

36. Run test job and ensure job output is returned without issues (e.g. scp problems)

    Sample test script using cloudq:

```
#!/bin/bash
#PBS -N testjob
#PBS -j oe
#PBS -m n
#PBS -q cloudq
#PBS -l select=1:ncpus=2:mem=16mb
#PBS -l walltime=0:10:00
sleeptime=60
cmd="sleep $sleeptime"
echo $cmd
$cmd
Exit
```

11 Command Reference

11.1 PBS Cloud CLI pclm Commands

11.1.1 CLI Bursting Scenario Commands

Command line interface commands that can be used to enable and disable a bursting scenario and get bursting scenario information.

A cloud bursting scenario is created using PBS Cloud and provides information needed for cloud bursting including the cloud provider, region where the node is burst, VPC details, cloud nodes booting script, SSH keys, valid instance types, etc.

Use the PBS Cloud CLI to obtain information about previously created bursting scenarios including its status, enable a bursting scenario so that nodes can be burst, or disable a bursting scenario to restrict nodes from being burst.

11.1.1.1 Bursting Scenario States

READY
: When scenario is enabled, this indicates the bursting scenario can be used to burst nodes

: When scenario is disabled, the scenario could be used to burst nodes if you enabled it

PENDING
: The scenario is added but not validated.

BUSY
: The bursting or unbursting workflow is running; we are in the process of bursting or unbursting.

DELETING
: The bursting scenario is being deleted.

ERROR
: The bursting scenario contains an error.

11.1.1.2 PBS Cloud Account States

READY
: The account is ready and usable.

PENDING
: Your login credentials are being validated with the cloud vendor.

BUSY
: The account is busy.

DELETING
: The account is being deleted.

ERROR
: The account contains an error.

11.1.1.3 Some Outputs of PCLM Commands

Scenario ID

Unique label that identifies a bursting scenario. Required to enable, disable or get the status of a bursting scenario. Use the Display a List of Bursting Scenarios command to obtain a list of bursting scenarios including the scenario ID.

Cloud node hostname

Hostname assigned to the cloud node. This information can be obtained by executing the command to display cloud node details.

Machine ID

A unique label that identifies the cloud node. This information can be obtained by executing the command to display cloud node details.

Private IP

Private IP address assigned to the cloud node. This information can be obtained by executing the command to display cloud node details.

11.1.1.4 Display a List of Bursting Scenarios

The below command displays a list of bursting scenarios that have been previously created using PBS Cloud.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario list
```

The output of the command is a list of bursting scenarios including the scenario ID, the state of the scenario, and whether the scenario is enabled or disabled.

11.1.1.5 Enable a Bursting Scenario

The below command enables a bursting scenario so that cloud nodes can be burst using the scenario.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario enable <scenario ID>
```

To verify that the bursting scenario is enabled, use the command Display Bursting Scenario Details.

11.1.1.6 Disable a Bursting Scenario

The below command disables a bursting scenario. Cloud nodes cannot be burst using a scenario that is disabled.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario disable <scenario ID>
```

To verify that the bursting scenario is disabled, use the command Display Bursting Scenario Details.

11.1.1.7 Display Bursting Scenario Details

The below command returns information about the bursting scenario:

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario show --id <scenario ID>
```

The output of the command displays information about the bursting scenario including the scenario ID, the state of the scenario, the associated cloud account. and whether the scenario is enabled or disabled.

If you want to display the amount of time before an idle node is unburst, then use the below command (the --raw option must be provided)

```
pclm --raw --api-endpoint=http://HOST:9980/pbspro-cloud/ --api-key KEY bootstrapper scenario show
```

The output of the command is in a JSON format. The idle time is displayed as a key-value pair in the output: {"idle_before_unburst": 100}.

11.1.1.8 Update the Minimum Time Before an Idle Node is Unburst

We recommend setting the Idle Before Unburst scenario parameter to more than double the PBS scheduler cycle time. The below command updates the bursting scenario setting the minimum time that a cloud node can be idle before it is unburst.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario patch --idle-before-unburst <max idle time before
    unbursting> <scenario ID>
```

To verify that the idle before unburst time is updated, use the command Display Bursting Scenario Details with the --raw option.

11.1.2 CLI Node Bursting Commands

Command line interface commands that can be used to burst and unburst cloud nodes and get status information about a bursting activity.

These commands can be used to test cloud bursting without using the cloud bursting hook. This will ensure that the connectivity from the PBS Server to the cloud infrastructure and the bursting scenarios are working properly. These commands can also be used when a site wants to burst a cloud node so that it remains burst indefinitely.

Cloud nodes burst manually remain up and running until explicitly unburst using the PBS Cloud CLI.

11.1.2.1 Command Options

Options that can be used with the `pclm` command:

--wait

Causes the application to wait for the node to be burst/unburst. This option must follow the keyword "bootstrapper".

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper --wait scenario burst \
```

--raw

Displays the output of the command in JSON format. This option must follow the keyword "pclm".

```
pclm --raw --api-endpoint==http://<PBS Cloud host name or IP address>:<PBS Cloud
    port>/pbspro-cloud/ --api-key <API key>
```

11.1.2.2 Command for Bursting Cloud Nodes

The below command bursts a number of cloud nodes based on <burst count> of type <instance type> for the bursting scenario identified by <API key>.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper --wait scenario burst '{"mom":[{"deployable_id":"<instance
    type>","count":"<burst count>","tags":{"burst-by":"user"}}]}'
```

The PBS cloud bursting hook always adds a tag called "burst-by" and its value is set to "pbs-cloudhook".

By using the same tag with a different value or a different tag altogether, manually burst nodes can be distinguished from those burst via the bursting hook.

The JSON that describes the type of node to burst and how many to burst can be included directly on the command-line, as shown above, or loaded from a separate file. For example, the following JSON can be copied to a file called node_to_burst.json:

```
{"mom":[{"deployable_id": "<instance type>", "count":"10", "tags": {"burst-by": "user"}}]}
```

The file can then be used as part of the bursting command to load the information.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper --wait scenario burst node_to_burst.json
```

11.1.2.3 Command to Display Cloud Node Details

The below command displays details for cloud nodes:

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario status -f tags
```

The status is displayed as a table. Use --raw to display the output of the command in JSON format. The status command displays the machine ID, instance type, hostname, private and public IP addresses, the OS image used to create the node, the node's state, and creation time. When the option -f tags is used, any associated tags are displayed as well.

```
(pclm-cli-env)$ pclm $API bootstrapper scenario status -f tags
+-----------------------------------+--------------+-------------+-----------+---------------+-------------------------------+-------+---------------------+-----------------------+
|             machine_id            | instance_type |   hostname  | private_ip |   public_ip   |           os_image           | state | workflow_created_at |          tags         |
+-----------------------------------+--------------+-------------+-----------+---------------+-------------------------------+-------+---------------------+-----------------------+
| 71b1f132-3a5d-42fc-9f25-16311e4b0ca2 | Standard_B11s | nodea000003 |  10.0.0.7  | 52.168.88.253 | pclm-centos7.4-hpc-cloud-init |   UP  | 2019-09-26 10:50:03 | {u'burst-by': u'user'} |
| f4c6e65b-6631-4571-8bfe-72d43c1c82e5 | Standard_B11s | nodea000002 |  10.0.0.6  | 52.168.88.231 | pclm-centos7.4-hpc-cloud-init |   UP  | 2019-09-26 10:50:03 | {u'burst-by': u'user'} |
+-----------------------------------+--------------+-------------+-----------+---------------+-------------------------------+-------+---------------------+-----------------------+
```

Figure 11-1:Command Output

11.1.2.4 Cloud Node States

The following is a list of possible virtual machine states. Only a virtual machine that is UP is guaranteed to have an IP addresses and a hostname. A virtual machine that has been fully unburst will not be present in the output at all.

DOWN

Indicates the machine is created in the local database, but not yet in the cloud provider. This status is short-lived and rarely visible.

DEPLOYING

Indicates deployment is ongoing.

FAILED_DEPLOYING

Indicates that something has gone wrong and the machine deployment has failed. The virtual machine is automatically unburst in this case.

FAILED_STARTING

Indicates that something has gone wrong and the machine has not started.

FAILED_STOPPING

Indicates that something has gone wrong and the machine has not been stopped.

FAILED_TERMINATING

Indicates that something has gone wrong and the machine has not been terminated.

STARTING

Indicates machine is starting.

STOPPED

Indicates the machine is stopped.

STOPPING
: Indicates machine is stopping.

TERMINATING
: Indicates that machine is being removed.

UP
: Indicates that the machine deployment has completed and the machine is ready.

11.1.2.5 Command for Unbursting Cloud Nodes

The below command unbursts cloud nodes. Use the command to display cloud bursting activity to obtain information about cloud nodes. A cloud node can be unburst using its machine ID, private IP address, or hostname.

pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/ --api-key <API key> bootstrapper --wait scenario unburst <machine ID1>|<private IP address1>|<cloud node hostname1> <machine ID2>|<private IP address2>|<cloud node hostname2> [...]

Multiple nodes can be unburst in a single call and the parameter used to unburst the nodes can be mixed:

unburst <machine ID1> <private IP address2> <cloud node hostname3>

This information can also be supplied in JSON format on the command-line or in a JSON file:

```
unburst '["<machine ID1>", "<private IP address2>", "<cloud node hostname3>"]'
unburst /tmp/machines_to_unburst.json
```

11.1.2.6 Sizing the Network Disk for the Cloud Node Root System

Use the disk_size_gb parameter to define the size of the network disk for the root file system in GBs when bursting a cloud node. By default the minimum size is compatible with the image associated with the bursting scenario defined by <API key>.

```
"mom" : [
    {
        "deployable_id": "c3.xlarge",
        "disk_size_gb": 12
    },
    {
        "deployable_id": "d2.xlarge",
        "count": 2,
        "disk_size_gb": 15
    }
]
```

11.1.2.7 Specifying the Image to Use when Bursting

You can manually burst a virtual machine from a specific image using the image parameter. By default, the virtual machine is burst based on the image specified when creating a bursting scenario (defined by the <API key>). The value of image depends on the cloud provider. For example, for AWS the name of the AMI is specified:

```
"mom":[{"deployable_id": "t2.medium", "count":"10", "image":"ami-123456"}]
```

For Azure, the resource group into which to place the VM and the name of the image must be specified:

```
"mom":[{"deployable_id": "Standard_DS1_v2", "count":"10", "image":"res-group/imagename"}]
```

11.1.2.8 Allowing Users to Request Infiniband Nodes

When the user wants to run HPC workloads like MPI jobs, the user needs InfiniBand supported nodes deployed on the same InfiniBand network. A restriction for bursting on Infiniband networks is that jobs cannot run across multiple Infiniband networks. To prevent this from happening, scale sets can be used. Only virtual machines within the same scale set have Infiniband connectivity. Use the infiniband_network parameter to ensure that all nodes for a bursting request get deployed into the same scaleset.

When bursting Infiniband nodes, the OS image associated with the bursting scenario defined by <API key> must contain everything that is needed to use Infiniband.

To create a new scaleset , set the value of the infiniband_network parameter to "new":

```
"deployable_id":"Standard_H16mr", "infiniband_network": "new"
```

To use an existing scale set, set the value of the infiniband_network parameter to "auto", which will re-use the existing scale sets as long as all requested nodes fit inside the scale set. If they do not, a new scale set is created for the requested nodes.

```
"deployable_id":"Standard_H16mr", "infiniband_network": "auto"
```

Use the command to display the status of the cloud bursting activity to get information about a node's scaleset. For example:

```
{ ...,
    "scaleset": {"nr": "2", "name": "pclmDEVvhoiAACrxTOHioWSkwg"}, ... }
```

Nodes with the same name value are in the same scaleset.

11.1.2.9 Bursting Asynchronously

Sometimes it can take several minutes to burst a cloud node. If you do not want to wait for the bursting command to complete, then eliminate the --wait option and provide a unique request identifier. The bursting call will return instantaneously. For example:

```
REQUEST_ID=$(uuidgen)
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper scenario burst --request-id $REQUEST_ID
    '{"mom":[{"deployable_id": "<instance type>", "count":"<burst count>"}]}'
```

Using the request identifier, poll the list of notifications for the bursting operation to determine its status.

```
pclm --raw --api-endpoint==http://<PBS Cloud host name or IP address>:<PBS Cloud
    port>/pbspro-cloud/ --api-key <API key>
notif thread list --request-id $REQUEST_ID --expand
```

Output from the command will look something like this:

```
[
    {
        "title": "Workflow bootstrapper.deploy_deployables",
        "created_at": "2019-10-01T09:02:57.383000+00:00",
        "open": false,
        "related": [ ... ],
        "notifications": [
            ... ,
            {
                "notification_id": "5d93170000b64a0001c4fe00",
                "sender": "executor",
                "message": "Workflow \"deploy_deployables\" execution succeeded",
                "timestamp": "2019-10-01T09:06:08.752000+00:00",
                "content": {
                    "workflow": "deploy_deployables",
                    "workflow_name": "bootstrapper.deploy_deployables",
                    "machine_ids": [
                        "a8d1681f-173d-4476-9d52-05e7e1b405d5",
                        "535ec0fc-f6f4-4b51-8f9d-abb13984f42d"
                    ],
                    "state": "SUCCEEDED",
                    "machines": [ ... ],
                    ... ,
                },
                ...
            }
        },
    ],
    ...
}
```

When a final message of success or failure is received, the open parameter is set to "false". There can be many notifications so be sure to read the last one (the latest timestamp is usually the last one in the list). The state of the notification content can be SUCCEEDED, RUNNING or FAILED.

11.1.2.10 Bursting Preemptable Instances

You can use the preemptable parameter to indicate that the cloud node is preemptable (currently implemented for AWS. AWS refers to this as spot instances). This parameter takes a Boolean value.

The bursting request must either contain only preemptable instances or only non-preemptable instances.

The bursting scenario associated with the <API key> must have spot instances enabled.

```
pclm --api-endpoint=http://<PBS Cloud host name or IP address>:<PBS Cloud port>/pbspro-cloud/
    --api-key <API key> bootstrapper --wait scenario burst "mom":[{"deployable_id":
    "c3.xlarge","count":"10","preemptable": true}]
```

Possible errors that may occur (returned as JSON, when the --raw option is used):

```
The current spot instance price for Instance Type xx in region yy is higher than the limit
    configured in the Bursting Scenario zz
The Instance Type xx has not been configured in the Bursting Scenario to be used for spot
    instances.
```

11.2 Command Reference and Sample Output for pkr

A kard defines all the container images and container services you need for PBS Cloud. Each kard is specific to its version of PBS Cloud. This version of PBS Cloud comes packaged with the PBS Cloud kard already fully defined. The PBS administrator is not expected to make any changes to the kard.

We use pkr to manage all the containers in the current kard. In this context that means all the services around the Cloud Bursting feature in PBS Professional. Each service runs in its own container.

To start all pkr services:

```
pkr start
```

To stop all pkr services:

```
pkr stop
```

To list all pkr services:

```
pkr ps
```

11.2.1 Sample pkr Output on Startup

```
[root@myhost ~]# pkr start
Starting postgres      ... done
Starting mongodb       ... done
Starting cadvisor      ... done
Starting loki          ... done
Starting fluentd       ... done
Starting prometheus           ... done
Starting fluentd              ... done
Starting bootstrapper-worker  ... done
Starting bootstrapper-worker1 ... done
Starting grafana              ... done
Starting guardian             ... done
Starting notification-center  ... done
Starting websocket-bridge     ... done
Starting pacioli              ... done
Starting keeper               ... done
Starting hype                 ... done
Starting hubble               ... done
Starting mistral-api          ... done
Starting ui                   ... done
Starting executor-api         ... done
Starting mistral-executor     ... done
Starting cloudflow            ... done
Starting bootstrapper-api     ... done
```

11.2.2 Sample pkr Output on Stop

```
[root@myhost ~]# pkr stop
Stopping bootstrapper-api     ... done
Stopping cloudflow            ... done
Stopping ui                   ... done
Stopping mistral-executor     ... done
Stopping executor-api         ... done
Stopping keeper               ... done
Stopping notification-center  ... done
Stopping pacioli              ... done
Stopping websocket-bridge     ... done
Stopping hype                 ... done
Stopping mistral-api          ... done
Stopping hubble               ... done
Stopping guardian             ... done
Stopping grafana              ... done
Stopping bootstrapper-worker1 ... done
Stopping bootstrapper-worker  ... done
Stopping prometheus           ... done
Stopping fluentd              ... done
Stopping node-exporter        ... done
Stopping postgres             ... done
Stopping rabbitmq             ... done
Stopping loki                 ... done
Stopping mongodb              ... done
Stopping cadvisor             ... done
```

11.2.3 Sample pkr Output while Running

A missing IP address indicates that a service is unhealthy, although an IP address does not guarantee health. System is running:

```
[root@myhost ~]# pkr ps
[root@pm-lwood-testbed current]# pkr ps
- cadvisor: 172.18.0.4
- loki: 172.18.0.6
- fluentd: 172.18.0.9
- mongodb: 172.18.0.3
- node-exporter: 172.18.0.5
- postgres: 172.18.0.2
- prometheus: 172.18.0.8
- grafana: 172.18.0.12
- rabbitmq: 172.18.0.7
- guardian: 172.18.0.13
- pacioli: 172.18.0.14
- notification-center: 172.18.0.16
- mistral-api: 172.18.0.20
- mistral-executor: 172.18.0.23
- keeper: 172.18.0.17
- hype: 172.18.0.18
- hubble: 172.18.0.19
- executor-api: 172.18.0.22
- cloudflow: 172.18.0.24
- bootstrapper-worker1: 172.18.0.10
- bootstrapper-worker: 172.18.0.11
- bootstrapper-api: 172.18.0.25
- websocket-bridge: 172.18.0.15
- ui: 172.18.0.21
```

11.2.4 Sample pkr Output while Stopped

System is stopped:

```
[root@myhost ~]# pkr ps
- cadvisor: stopped
- loki: stopped
- fluentd: stopped
- mongodb: stopped
- node-exporter: stopped
- postgres: stopped
- prometheus: stopped
- grafana: stopped
- rabbitmq: stopped
- guardian: stopped
- pacioli: stopped
- notification-center: stopped
- mistral-api: stopped
- mistral-executor: stopped
- keeper: stopped
- hype: stopped
- hubble: stopped
- executor-api: stopped
- cloudflow: stopped
- bootstrapper-worker1: stopped
- bootstrapper-worker: stopped
- bootstrapper-api: stopped
- websocket-bridge: stopped
```

Index

A
Altair License Server CG-6
Amazon Web Services CG-5
AWS CG-5
Azure cloud head node CG-127

B
burst CG-1
burst_by_hook CG-12

C
CentOS CG-5
cloud head node in Azure CG-127
cloud queue CG-1
cloud queues CG-14
cloud_instance_type CG-12
cloud_max_instances CG-13
cloud_max_jobs_check_per_queue CG-13
cloud_min_instances CG-12
cloud_network CG-13
cloud_node_image CG-13
cloud_node_instance_type CG-13
cloud_provisioned_time CG-13
cloud_queue CG-13
cloud_scenario CG-13
configuring PBS for cloud bursting CG-12
custom resources for cloud bursting CG-14

D
Deutsche Telekom CG-5
docker-ce CG-6
docker-ee CG-6

F
firewalld CG-128

G
GCP CG-5
Google Cloud Platform CG-5

H
HUAWEI Cloud CG-5

I
installation script CG-9
instance type CG-1, CG-24

L
lic_signature CG-13

M
Microsoft Azure CG-5

N
node_location CG-13

O
Open Telekom Cloud CG-5
OpenStack CG-5
Oracle Cloud Platform CG-5
Orange Cloud Flexible Engine CG-5
OTC CG-5

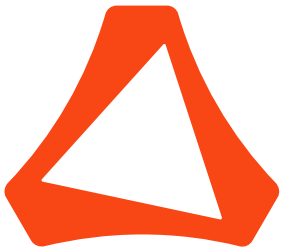
P
PBS
    configuring for cloud bursting CG-12
pkr CG-9, CG-138
    sample output CG-139

R
RHEL CG-5

S
scratch CG-108
SELinux CG-6, CG-128
SLES CG-5
    restrictions CG-5

U
unburst CG-1

V
VPN CG-6



ALTAIR

Altair PBS Professional 2021.1

Budget Guide

You are reading the Altair PBS Professional 2021.1

Budget Guide (BG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation vii

1 Introduction to Budget 1

1.1 Budget Tracks and Manages Credit for PBS Jobs . . . 1
1.2 Budget Terminology . . . 2
1.3 Roles . . . 4
1.4 Investing and Consuming Service Units . . . 5
1.5 Accounts in Budget . . . 7
1.6 Accounting Tools . . . 11
1.7 Caveats and Troubleshooting . . . 18
1.8 Formats in Budget . . . 18

2 Installing and Upgrading Budget 19

2.1 Whether or Not to Start with Failover . . . 19
2.2 Supported Platforms for Budget . . . 19
2.3 Prerequisites . . . 19
2.4 Recommended Configurations . . . 22
2.5 Installation Steps . . . 22
2.6 Validating Budget . . . 29
2.7 Configuring Budget for Failover . . . 29
2.8 Upgrading Budget . . . 32
2.9 Changing Budget Administrator to New Username . . . 33

3 Configuring Budget 35

3.1 Initial Configuration . . . 35
3.2 Defining Billing Periods . . . 39
3.3 Adding a PBS Complex and Setting its Billing Model . . . 40
3.4 Setting Budget Configuration Attributes . . . 45
3.5 Configuring Budget for Peer Scheduling . . . 46

4 Budget Commands 47

4.1 Budget Commands . . . 47
4.2 Commands for Managing Budget Elements . . . 49
4.3 Transaction and Account Checking Commands . . . 83

5 Basic Install and Configure 97

5.1 Basic Install and Configure Instructions . . . 97

6 Using Budget 103

6.1 Managing Credit with Budget . . . 103
6.2 Submitting Jobs with Budget . . . 103
6.3 Tutorial . . . 105

Index 109

Contents

1 Introduction to Budget

1.1 Budget Tracks and Manages Credit for PBS Jobs

Budget allows you do track and manage credit and other resources for jobs at PBS complexes. You can define and track any form of currency you need: dollars, CPU hours, GPU hours, etc. You can define how you want each currency calculated separately for each PBS complex.

You represent each department in your organization as a group in Budget. Each department receives investment which it can then spend on projects and individual users. Funding allocations apply to, are charged against, and are tracked in time periods. Every transaction is available for examination forever.

Before a job runs, Budget checks that the job owner has enough currency to run the job, then transfers that amount from the job owner's account into escrow. After the job runs, Budget reconciles charges for the job: it debits escrow for the amount consumed by the job and returns any excess to the job owner, or if the job consumed more than requested it debits more from the job owner's account. The Budget administrator can provide partial or full refunds if necessary.

Budget is flexible; a department can fund multiple projects and users working in multiple PBS complexes. A project or user can be funded by multiple departments and run jobs in multiple PBS complexes.

Budget uses a Postgres database.

1.1.1 Number of Instances and Workers

A large site will probably find that a single instance of Budget for all PBS complexes works best. With a single instance of Budget, you can use a different charging model for each PBS complex.

Budget is web-based for scalability. By default, the number of workers is two; you can spawn more if you need to.

1.1.2 Hooks and Formulas

Budget is tightly integrated with the job management mechanisms of PBS Professional.

Budget uses hooks to calculate billing at each PBS complex. Budget uses two hooks, am_hook and am_hook_periodic, that are identical except for their triggering events, that are installed in the PBS complexes. These hooks calculate service unit consumption using the billing formula defined in the formula file, which is used as the hook configuration file. You can define the formula to be based on compute resources, time, day of week, time of day, priority of queue, etc. See section 3.3.2, “Define Billing Formula”, on page 40.

1.1.3 Managing Job Submission

When you manage a PBS complex via Budget, each PBS job can run only when the job can be charged to a project or user account that has sufficient credit. The organization’s CFO can fund a group representing a department or other organizational entity; this group can then deposit credit with project or user accounts.

A project spends credit when a user associated with the project runs a job and charges the job to the project account. A job can be charged to a project account only when it is run by an associated user on an associated complex. Any user associated with the project can charge a job to the project account.

An individual user spends credit when they run a job and charge it to their own account. This job must be run on a complex that is associated with the user.

When a group manager deposits group credit to a project account, that allocation is for a specific period and in a specific currency (service unit). Currency is usually defined as resource usage, such as CPU hours or GPU hours, but can be in dollars or other monetary units.

In addition to checking credit balances, Budget checks whether a job owner has hit any quotas for externally-managed resources such as storage, using a mechanism called *dynamic service units*. See section 1.6.2.2, "Dynamic Service Units", on page 13.

1.2 Budget Terminology

Account

Budget has user, project, and group accounts. See User, User Account, Project, Project Account, and Group, Group Account.

Accounting Policy

A policy associated with a project or user. Can be any of *begin_period*, *end_period*, or *proportionate*.

Active or inactive

An active element can perform all its normal functions; an inactive element cannot run jobs or do transactions.

Allocation

A specified amount of one or more service units designated for use by a project or user. An allocation exists for a fixed time period. As the service units are consumed, the allocation is depleted. Allocations can be made for compute resources, budget resources, time resources, or a combination of those.

AMS

Security module used by Budget to retrieve tokens, verify them, and authenticate users.

Billing Formula

An arithmetic formula defining how one or more service units is charged at a PBS complex. This is typically a PBS resource and time, for example, the PBS resource ncpus multiplied by walltime makes the compute resource CPU hours. Other examples are GPU hours, memory hours, etc. The billing formula for each PBS complex is defined in the Budget hook configuration file for that complex. See section 3.3.2, "Define Billing Formula", on page 40.

Cluster

A data structure representing a PBS complex. Named for the PBS server. See section 1.6.5, "Clusters", on page 17.

Element

Blanket term used to refer to a part of Budget, such as an account, a transaction, a role, etc. Can be an entity or something associated with an entity, such as a limit on a service unit.

Entity

Data structure representing a part of Budget: a period, cluster, service unit, user, project, or group.

Group, Group Account

Entity representing an organizational structure such as a department, business unit, etc. A group account has a credit balance, at least one associated investor who invests service units in the group account, and at least one associated manager who uses the group account to fund users and projects.

Instance

An installation of Budget; described by a name and the hostname or IP address of the machine where Budget is installed.

PBS complex

An installation of PBS Professional consisting of daemons including a server daemon and at least one MoM daemon, and the hosts on which those daemons run. The name for a PBS complex is the name of its server daemon, which is found in PBS_SERVER parameter in the `/etc/pbs.conf` file.

Periods, Allocation Periods, Billing Periods

Time period. Budget can use a hierarchical system of time periods.

Project, Project Account

Entity representing a project, for example a workflow, as well as its associated account, including its credit balance. A project has associated users and clusters, and users associated with the project can run jobs on those complexes and charge the project account.

Reconciling Jobs

After a job runs, Budget reconciles charges for the job: it debits escrow for the amount consumed by the job and returns any excess to the job owner, or if the job consumed more than requested it debits more from the job owner's account.

Roles

Roles define available actions and access to features in Budget. Roles can be *admin*, *investor*, *manager*, *teller*, and *user*.

Service Unit

A standard service unit represents a currency, which can be dollars, CPU hours, GPU hours, etc; see section 1.6.2.1, “Standard Service Units”, on page 13.

A dynamic service unit lets you set a quota on an externally-managed resource such as storage; see section 1.6.2.2, “Dynamic Service Units”, on page 13.

Standard Service Units

Currency representing compute resources, budget resources, time resources, or a combination. Defined by the Budget administrator; can be specific to the needs of the client. Examples: CPU hours, GPU hours, dollars.

Dynamic Service Units

Mechanism for placing a quota on an external compute resource such as storage. If a quota is exceeded, jobs that depend on that dynamic service unit do not start.

Transactions

An action that affects a credit balance, such as depositing or refunding service units to an account, or withdrawing service units from an account.

User, User Account

We use these terms to mean two different things:

- Budget entity representing an individual user and their associated account, including its credit balance. This user is typically a job submitter with the *user* role, although an individual user can have any role. Each user account is funded by at least one group manager. See section 1.5.3, “User, User Account”, on page 10.
- Username, password, etc. for administrator, database user, teller, or job submitter. See section 2.3.4, “Required User Accounts”, on page 20.

Worker

Process that handles transactions for Budget. This process is run by the web server.

1.3 Roles

Roles define available actions in and access to the features of Budget. Except for the *teller* role, roles are hierarchical; each role includes the capabilities of the roles below it in the hierarchy. For example, if a user is defined as an *investor*, they can be added later to a group as a manager. A user can have only one role at a time, although that role may have the privileges of lower roles. Roles are case-sensitive and are lowercase. The *teller* role is a special-purpose role for processing transactions, and its capabilities are shared only with the *admin* role.

In hierarchical order, with the most powerful and inclusive role first:

admin

Budget administrator. Configures Budget; adds users, projects, groups, clusters, sets roles, defines periods, gives refunds, etc. Can modify any element in Budget. Can transfer service units from any entity to any entity.

When you install Budget, you specify a username for the Budget administrator. This account automatically gets assigned the *admin* role. This username is typically *pbsadmin*.

Not automatically associated with specific projects or groups.

The *admin* role includes the capabilities of the *investor*, *manager*, *teller*, and *user* roles.

investor

A user who is associated with a group and is responsible for investing service units in the associated group account. Can also withdraw service units from the group account. An investor can be associated with one or more groups.

Created by Budget administrator.

The *investor* role includes the capabilities of the *manager* and *user* roles.

manager

A user who is associated with a group and is responsible for depositing service units from the group account to associated user and project accounts. Can also withdraw service units from these user and project accounts, putting them back into the group account. Can be associated with multiple groups.

Created by Budget administrator.

The *manager* role includes the capabilities of the *user* role.

teller

A user who performs all acquire and reconcile transactions on behalf of projects and users. When the Budget hook performs these actions, it uses the *teller* role and username amteller.

The teller has full permissions for transactions and can read all projects, accounts, groups, etc.

Not associated with groups or projects.

The *teller* role is a special-purpose role for processing transactions, and its capabilities are shared only with the *admin* role.

Created by Budget administrator.

The *teller* role includes the capabilities of the *user* role.

user

PBS job submitters who have an individual account and/or are associated with a project. A job submitter with an individual account can charge jobs to their own account. A job submitter who is associated with a project can charge jobs to the project.

Created by Budget administrator.

Operations:

- When you create any user, you must assign a role to that user; see section 4.2.1.2, “Adding a User”, on page 50.
- To see all roles that have been created, list them; see section 4.2.2.9, “Listing Roles”, on page 59.
- To change a user's role, use `amgr update user`; see section 4.2.3.2, “Updating Users”, on page 60.

1.4 Investing and Consuming Service Units

1.4.1 Investing in Groups (Cost Centers)

Groups in Budget represent organizational entities such as departments or businesses. For example, an organization might have multiple departments including engineering, systems, and software. The CFO, who can be represented in Budget as an investor, deposits service units to a department's group account. An investor can deposit to multiple groups, and a group can have multiple investors.

The amount each investor dispenses to a group is tracked. An investor can withdraw funds from a group, but cannot withdraw more than the amount they deposited to that group.

The group budget pool does not have any defined period. The amount in the budget pool is available for an infinite amount of time.

Operations:

- Investor invests in a group via `amgr deposit group`; see section 4.3.1.3, “Depositing Service Units to Group”, on page 85.
- Investor withdraws from a group via `amgr withdraw group`; see section 4.3.3.3, “Withdrawing Service Units from Group”, on page 88.
- Administrator transfers between groups via `amgr transfer group`; see section 4.3.8.5, “Transferring Service Units for Investors and Group”, on page 95.

1.4.2 Investing in Users and Projects

Once the group has been been allocated service units, group managers can use that pool to fund projects and users. Each allocation to a user or project is for a specific period of time. For example, the engineering group managers can allocate funds to the design project for Quarter1 and to the testing project for Quarter2 and Quarter3.

A manager can use up to the entire pool on one user or project.

Each group manager can be linked to multiple groups, and each group can have multiple managers. Each group can fund multiple users and projects, and each user or project can have multiple groups funding it.

If necessary, a group manager can also withdraw funds from projects and users, but only up to the amount they deposited.

1.4.3 Charging Jobs to User or Project Account

The Budget hook monitors each job submission, and when the job submitter specifies a project via `qsub -P <project name>`, Budget charges the job to that project. If the job submitter does not specify a project, Budget charges the job to the job submitter's own account.

1.4.4 Job and Credit Lifecycle

When a job runs, it can consume multiple kinds of service units, for example both CPU hours and GPU hours.

1. User submits a job, charging it either to the user's account or a project account
2. Optionally, Budget validates credit of the job owner (a user or project)
3. Job is queued at PBS server
4. Before running the job:
   a. Budget checks balance of job owner's account
   b. Budget acquires service units from job owner and puts them in escrow
5. Job runs at execution host(s)
6. When the job finishes, Budget reconciles the job: it debits the amount of credit that was consumed by the job and returns any excess to the user or project, or if the job consumed more than requested it debits more from the job owner's account

1.4.5 Reconciling Jobs

When a job finishes, Budget reconciles the job by debiting the amount of credit used by the job, and returning the excess to the job owner's account, or if the job consumed more than requested it debits more from the job owner's account. However, in rare cases a job cannot be reconciled when the job fails and the execjob_epilogue hook does not run. Budget will try to reconcile a job up to 3 times; if it is not successful, it marks the job as not reconciled.

You can see all unreconciled jobs that have had *count* or more attempts to reconcile:

amgr report transaction -i <job or transaction ID> -N <count>

For example, `amgr report transaction -N 2` displays all non-reconciled jobs with *count* >=2.

See section 4.2.5.5, “Getting Job and Transaction Reports”, on page 79.

You have to manually reconcile jobs with 3 failed attempts:

amgr reconcile job -i <job ID list> --all -u <job owner username>

or

amgr reconcile project -n <project name> -c <cluster> -s <service unit name> <service unit amount> [-D <transaction date>] -i <transaction ID> -d <duration> [-C <comment>]

See section 4.3.6, “Reconciling Service Units”, on page 91.

1.4.6 Refunding Users and Projects

The administrator can refund the user or project that funded a job for some or all of the service units charged for the job. You can choose to provide a refund for situations such as when a job fails or runs multiple times, for reasons which cannot be attributed to the job owner. You can provide refunds to active and inactive projects and users. You can provide multiple refunds for the same job, but the total refund cannot exceed the amount consumed by the job.

Budget knows who paid for the job, so you do not have to specify where the refund goes. The refund amount is calculated by multiplying the specified percentage by the total consumed amount. Total consumed amount is the sum of all transaction amounts of all transactions for a job. Refunds are strictly per job.

You must be *admin* to provide a refund.

When a job is refunded, the comment in the report is always prefixed with "refund:".

See section 4.3.7, “Refunding Service Units”, on page 93.

1.4.7 Retrieving Abandoned Service Units

If service units go unclaimed and become unusable, the *admin* role can transfer them so that they become usable. You can transfer from any period, group, user, or project, to any other period, group, user, or project. You can transfer between investors in the same group or different groups. A transfer gets its own transaction ID.

Service units may become unusable when the following happen:

- An investor is unlinked from a group, leaving funds with the group
- A group is unlinked from a project, leaving funds with the project
- A period expires and leaves service units unused in a user or project account

See section 4.3.8, “Transferring Service Units”, on page 94.

You must be *admin* to run this command.

1.5 Accounts in Budget

1.5.1 Group, Group Account

A group, also called a group account, is a type of account that represents an organizational structure such as a department, a business unit, etc. A group has the following:

- Name
- Credit balance
- One or more investors
- One or more managers
- Is active or inactive

Group managers provide the funding for users and projects to run jobs by depositing service units to those accounts. Group managers can also withdraw service units from associated job and project accounts and return the service units to the group account. All funding for users and projects comes from groups. The group credit balance is in standard service units; see Service Units.

Group managers can also set a quota on an externally-managed resource such as storage by setting a limit for the dynamic service unit representing that resource. A dynamic service unit limit is per user, project, and period, so you can set a different limit for each user and project for each period. Budget checks usage by the job owner against this quota before running any job. See section 1.6.2.2, “Dynamic Service Units”, on page 13.

Groups do not:

- Directly run jobs
- Have associated time periods
- Have associated dynamic service units

Operations:

- Administrator adds group via `amgr add group`; see section 4.2.1.4, “Adding a Group”, on page 52
- Administrator updates group via `amgr update group`; see section 4.2.3.4, “Updating Groups”, on page 63
- User lists group via `amgr ls group`; see section 4.2.2.4, “Listing Groups”, on page 56
- Administrator removes group via `amgr rm group`; see section 4.2.4.4, “Removing a Group”, on page 67. Note that if a group has any associated current or past transactions, removing it results only in making it inactive.
- Group manager deposits service units to users and projects via `amgr deposit`; see section 4.3.1, “Depositing Service Units”, on page 83
- Group manager withdraws service units from users and projects via `amgr withdraw`; see section 4.3.3, “Withdrawing Service Units”, on page 87
- Manager checks credit balance of group via `amgr checkbalance group`; see section 4.3.2.6, “Checking Service Unit Balance for Group”, on page 86
- Group manager transfers service units to and from users and projects via `amgr transfer`; see section 4.3.8, “Transferring Service Units”, on page 94
- Manager gets reports on groups via `amgr report group`; see section 4.2.5.4, “Getting Group Reports”, on page 76
- Investor invests in a group via `amgr deposit group`; see section 4.3.1.3, “Depositing Service Units to Group”, on page 85.
- Investor withdraws from a group via `amgr withdraw group`; see section 4.3.3.3, “Withdrawing Service Units from Group”, on page 88.
- Administrator transfers between groups via `amgr transfer group`; see section 4.3.8.5, “Transferring Service Units for Investors and Group”, on page 95.

The difference between being active and inactive is that an active group can participate in transactions, but an inactive group cannot.

1.5.2 Project, Project Account

A project account is designed to represent a project such as a workflow. It has the following:

- Name
- Credit balance which it can use to run jobs
- One or more associated groups
- One or more associated users
- One or more associated clusters representing PBS complexes
- Accounting policy; see section 1.6.4, “Accounting Policy”, on page 17
- Optional independent start date
- Optional independent end date
- Optional metadata
- State: active or inactive

A project acquires credit when a group manager deposits group credit to the project account. A project spends credit when a user associated with the project runs a job and charges the job to the project account. A job can be charged to a project account only when it is run by an associated user on an associated complex. Any user associated with the project can charge a job to the project account. A project can use multiple clusters.

When a group manager deposits group credit to a project account, that allocation is for a specific period and in a specific currency (service unit). Currency is usually defined as resource usage, such as CPU hours or GPU hours, but can be in dollars or other monetary units.

Operations:

- Administrator adds project via `amgr add project`; see section 4.2.1.3, "Adding a Project", on page 51
- Administrator updates project via `amgr update project`; see section 4.2.3.3, "Updating Projects", on page 61
- Project member lists project via `amgr ls project`; see section 4.2.2.3, "Listing Projects", on page 56
- Administrator removes project via `amgr rm project`; see section 4.2.4.3, "Removing a Project", on page 67. Note that if a project has any associated current or past jobs or transactions, removing it results only in making it inactive.
- Group manager deposits service units to project via `amgr deposit project`; see section 4.3.1.2, "Depositing Service Units to Project", on page 84
- Group manager withdraws service units from project via `amgr withdraw project`; see section 4.3.3.2, "Withdrawing Service Units from Project", on page 87
- Project member, investor, or manager checks credit balance of project via `amgr checkbalance project`; see section 4.3.2.5, "Checking Service Unit Balance for Project", on page 86
- Group manager transfers service units to and from projects via `amgr transfer project`; see section 4.3.8.4, "Transferring Service Units for Project", on page 95
- Administrator or teller reconciles service units for project via `amgr reconcile project`; see section 4.3.6.2, "Reconciling Service Units for Project", on page 91
- Group manager applies limits to dynamic service units for project via `amgr limit project`; see section 4.2.6, "Applying Limits to Dynamic Service Units", on page 80
- Administrator or teller acquires service units for project via `amgr acquire project`; see section 4.3.5.4, "Acquiring Service Units for Project", on page 90
- Project member, investor, or manager prechecks service unit balance for project via `amgr precheck project`; see section 4.3.4.1, "Prechecking a User or Project", on page 89
- Project member, investor, or manager gets reports on projects via `amgr report project`; see section 4.2.5.3, "Getting Project Reports", on page 71

The difference between being active and inactive is that an active project can run jobs and participate in transactions, but an inactive project cannot.

Project start and end times are not periods and are independent of periods.

When you create a project, you must assign an accounting policy and at least one cluster.

1.5.2.1 Project Attributes

Projects have a metadata attribute, consisting of comma-separated key-value pairs, where the keys are undefined. The administrator can set, update, or remove metadata when creating or updating a project.

Example 1-1: Add metadata to project:

```
amgr update project -n project1 -m + Owner:"Owner1"
```

Example 1-2: See metadata in report:

```
amgr report project -n project1
--------------------------------------------------------------------------------
name     | period   | serviceunit | opening_balance | ... | metadata           |
--------------------------------------------------------------------------------
project1 | 2021.feb | cpu_hrs     | 0.0             | ... | {'Owner': 'Owner1'} |
```

Example 1-3: See metadata by listing project:

```
amgr ls project -n project1 -l
project1
    account = project1
    metadata = {'Owner': 'Owner1'}
...
```

Example 1-4: Remove metadata:

```
amgr update project -n project1 -m - Owner
```

Example 1-5: Verify by listing project:

```
amgr ls project -n project1 -l
project1
    account = project1
    metadata = {}
...
```

1.5.3 User, User Account

Entity representing an individual user and their associated account, including its credit balance. This user is typically a job submitter with the *user* role, although an individual user can have any role. A user has the following:

- Name, which is the username
- Credit balance
- One or more associated groups
- One or more associated clusters representing PBS complexes
- Optionally, one or more projects with which the user is associated
- Role; see section 1.3, “Roles”, on page 4
- Accounting policy; see section 1.6.4, “Accounting Policy”, on page 17
- Is active or inactive

A user acquires credit when a group manager deposits group credit to the user account. A user spends credit when that user runs a job and charges the job to their account. A job can be charged to a user account only when it is run by that user on a complex associated with that user. A user can be assigned to zero or more projects. When a user runs a job, they can charge the job to an associated project account, or to their own account.

When a group manager deposits group credit to a user account, that allocation is for a specific period and in a specific currency (service unit). Currency is usually defined as resource usage, such as CPU hours or GPU hours, but can be in dollars or other monetary units.

A user with any role can have an individual user account.

Operations:

- Administrator adds user via `amgr add user`; see section 4.2.1.2, "Adding a User", on page 50
- Administrator updates user via `amgr update user`; see section 4.2.3.2, "Updating Users", on page 60
- Job submitter lists self, or manager lists user user via `amgr ls user`; see section 4.2.2.2, "Listing Users", on page 55
- Administrator removes user via `amgr rm user`; see section 4.2.4.2, "Removing a User", on page 66. Note that if a user has any associated current or past jobs or transactions, removing it results only in making it inactive.
- Group manager deposits service units to user via `amgr deposit user`; see section 4.3.1.1, "Deposit Service Units to User", on page 83
- Group manager withdraws service units from user via `amgr withdraw user`; see section 4.3.3.1, "Withdrawing Service Units from User", on page 87
- Job submitter checks own credit balance, or manager checks credit balance of user via `amgr checkbalance user`; see section 4.3.2.4, "Checking Service Unit Balance for User", on page 86
- Administrator transfers service units to and from users via `amgr transfer user`; see section 4.3.8.3, "Transferring Service Units for User", on page 94
- Group manager applies limits to dynamic service units for user via `amgr limit user`; see section 4.2.6, "Applying Limits to Dynamic Service Units", on page 80
- Administrator or teller acquires service units for user via `amgr acquire user`; see section 4.3.5.3, "Acquiring Service Units for User", on page 90
- Job submitter prechecks own service unit balance, or manager prechecks service unit balance for user via `amgr precheck user`; see section 4.3.4.1, "Prechecking a User or Project", on page 89
- Job submitter gets report on self, or manager gets reports on user via `amgr report user`; see section 4.2.5.2, "Getting User Reports", on page 69

The difference between being active and inactive is that an active user can run jobs and participate in transactions, but an inactive user cannot.

1.5.3.1 Requirements for Adding Job Submitters

When you create a user, you must assign a role, an accounting policy, and at least one cluster.

Each user you add to Budget should already have an entry in the password file, with a password set, and a home directory on the Linux system where Budget is installed.

When you add a user who will run jobs, that user must already be able to run jobs in a PBS complex.

1.6 Accounting Tools

1.6.1 Periods, Allocation Periods, Billing Periods

A period, also called a billing period or an allocation period, is a defined period of time with fixed start and end dates.

When a group manager deposits service units to a project or user account, that allocation is deposited for a specific period. The allocation is available to the project or user for that period only, expires at the end of the period, and is reported against that period.

Group accounts do not have associated periods.

The Budget administrator creates all periods. You can create a hierarchy of billing periods where the parent period encompasses the child periods. For example the parent period can be Year, and the child periods can be Quarter1, Quarter2, etc. There is no limit to the depth of the hierarchy.

Operations:

- Administrator adds period via `amgr add period`; see section 4.2.1.6, “Adding a Period”, on page 53
- Administrator updates period via `amgr update period`; see section 4.2.3.6, “Updating a Period”, on page 64. Note that if a period has any associated jobs or transactions, you cannot update it.
- Any user lists any period via `amgr ls period`; see section 4.2.2.6, “Listing Periods”, on page 57
- Administrator removes period via `amgr rm period`; see section 4.2.4.6, “Removing a Period”, on page 68. Note that if a period has any associated current or past jobs or transactions, you cannot remove it.

1.6.1.1 Caveats for Creating Periods

- Make sure that periods at the same level do not overlap. For example, if Quarter1 ends March 31st, make sure that Quarter2 does not begin sooner than April 1st.
- If you want to create periods with a parent-child relationship, you must create the parent period first. You cannot add a parent to an existing child. For example, if you want Year as the parent and Quarter1, Quarter2, etc., as children, create Year first.
- If you create child periods, make sure that they fit within the parent period.
- Make sure that your period hierarchy is finalized BEFORE doing any transactions or running any jobs; you cannot update or remove periods once transactions have been performed or jobs have started.

1.6.2 Service Units

Budget has two types of service units:

- You use *standard service units* to track and bill for resource usage, such as dollars, CPU hours, or GPU hours; this type is *SU_STANDARD*
- You use *dynamic service units* to set quotas on externally-managed resources such as storage; this type is *SU_DYNAMIC*

You can define as many service units as you need, and you can define each one to be whatever you need. You can define and use different service units for billing at each PBS complex. A job, project, or user can consume multiple service units. The default type is *SU_STANDARD*.

Operations:

- Administrator adds service unit via `amgr add serviceunit`; see section 4.2.1.7, "Adding a Service Unit", on page 54
- Administrator updates service unit via `amgr update serviceunit`; see section 4.2.3.7, "Updating a Service Unit", on page 64. You can update the type of a service unit only when it has no associated transactions or value updates. Use this command to set a service unit active or inactive.
- Any user lists service unit via `amgr ls serviceunit`; see section 4.2.2.7, "Listing Service Units", on page 58
- Administrator removes service unit via `amgr rm serviceunit`; see section 4.2.4.7, "Removing a Service Unit", on page 68. Note that if a service unit has any associated current or past transactions, removing it results only in making it inactive.
- Administrator uses `cron` script to update dynamic service unit usage `via amgr update dynamicvalues`; see section 4.2.3.9, "Updating Dynamic Service Unit Usage", on page 65.
- Administrator sets a limit on the service unit via `amgr limit {user | project}`; see section 4.2.6, "Applying Limits to Dynamic Service Units", on page 80
- Administrator defines the service unit in the billing formula; see section 3.3.2, "Define Billing Formula", on page 40

The difference between being active and inactive is that an active service unit can be used for transactions or for quota checks, but an inactive one cannot; it cannot be invested, transferred, consumed, etc.

1.6.2.1 Standard Service Units

A standard service unit can be a monetary unit such as dollars, or it can represent an internally-managed resource such as CPU hours or GPU hours. Standard service units can be treated like a currency.

The Budget hook runs at the PBS complex where a job runs and tracks standard service unit usage by the job. Budget debits the account of the job owner for the usage by the job. When you manage a PBS complex via Budget, each PBS job can run only when the job can be charged to a project or user account that has sufficient standard service units.

1.6.2.1.i Adding and Removing Standard Service Units

You can create separate standard service units for each cluster.

To create a standard service unit and add it to Budget:

1. Define the service unit in the billing formula; see section 3.3.2, "Define Billing Formula", on page 40
2. Add the service unit to Budget via amgr `add serviceunit -n <name of new service unit>`; see section 4.2.1.7, "Adding a Service Unit", on page 54

To remove a standard service unit:

1. Remove the service unit from all formulas, otherwise jobs will not run
2. Remove the service unit from Budget via `amgr rm serviceunit -n <service unit name>`; see section 4.2.4.7, "Removing a Service Unit", on page 68

1.6.2.2 Dynamic Service Units

A dynamic service unit tracks an external resource such as storage. You use a dynamic service unit by setting a limit on it to establish a quota. You can set separate limits on a dynamic service unit for each user and project, and for each user or project, you can specify a different limit for each period, via `amgr limit {user | project}`; see section 4.2.6, "Applying Limits to Dynamic Service Units", on page 80. To set a limit on a dynamic service unit for a user or project, you must be the group manager for the group that provides standard service units for that user or project.

Budget checks whether a job owner has hit a quota before starting each job. If a job owner hits a quota, they cannot start any more jobs until either the limit is raised or usage goes down.

1.6.2.2.i Caveats for Dynamic Service Units

If a dynamic service unit has no limit set on it, the limit is zero. If there are active dynamic service units, all jobs are checked against those quotas, and a zero quota will stop any job from running. Make sure that you don't unintentionally stop non-target users or projects from running jobs:

- Make sure you set the quota for all users and projects
- When you specify the period, make sure you either:
  - Set the desired quota at the top level of the period hierarchy
  - Set a very high quota at the top level of the period hierarchy, and a more restrictive quota for the period you need to control

1.6.2.2.ii Attributes for Dynamic Service Units

The data_lifetime Budget configuration attribute specifies the maximum time period between updates to dynamic service units. The default value is 3600 seconds. See section 3.4, "Setting Budget Configuration Attributes", on page 45.

1.6.2.2.iii Adding and Removing Dynamic Service Units

To create and use a dynamic service unit, you add it to Budget, set a limit, and keep its value updated:

1. Add the service unit to Budget via amgr `add serviceunit -n <name of new service unit>`; see section 4.2.1.7, "Adding a Service Unit", on page 54
2. Set a limit on the service unit via `amgr limit {user | project}`; see section 4.2.6, "Applying Limits to Dynamic Service Units", on page 80
3. Periodically update the value of the service unit by having a `cron` script call `amgr update dynamicvalues`; see section 4.2.3.9, "Updating Dynamic Service Unit Usage", on page 65. The script should update the value at intervals that are smaller than the limit set in the data_lifetime attribute; the default value is 3600 seconds. See section 3.4, "Setting Budget Configuration Attributes", on page 45

To remove a dynamic service unit from Budget, use `amgr rm serviceunit -n <service unit name>`; see section 4.2.4.7, "Removing a Service Unit", on page 68

1.6.2.3 Examples of Storage Quotas via Dynamic Service Units

Example 1-6: Setting user and project quotas:

Set user quota:

```
amgr limit user -n user1 -s storage 12.0 -p 2021
```

Set project quota:

```
amgr limit project -n project1 -s storage 25.0 -p 2021
```

Example 1-7: Setting consumption by user and project. This call is typically made via a `cron` job, and you would want to make sure you set this more often than the limit in the data_lifetime attribute:

Set consumption for user1:

```
amgr update dynamicvalues -v '{"storage": {"user1": {"total":8}}}'
```

Set consumption for project1:

```
amgr update dynamicvalues -v '{"storage": {"project1": {"total":30}}}'
```

Example 1-8: Reporting storage usage:

Report user storage data:

```
amgr report user -n user1 -t SU_DYNAMIC
--------------------------------------------------------------------------------
name     | serviceunit  | period    | limit   | last_reported_time  | total_consumed
--------------------------------------------------------------------------------
user1    | storage      | 2021.feb  | 12.0    | 2021-02-11 18:58... | 8.0
```

Report project storage data:

```
amgr report project -n project1 -t SU_DYNAMIC
--------------------------------------------------------------------------------
name     | serviceunit  | period    | limit   | last_reported_time  | total_consumed
--------------------------------------------------------------------------------
project1 | storage      | 2021.feb  | 25.0    | 2021-02-11 18:58... | 30.0
```

Example 1-9: Queued job for project1 won't run. Why? Because it is over quota as shown in the report above.

```
qstat 3269
Job id                  Name             User              Time Use S Queue
----------------------  ---------------- ----------------  -------- - -----
3269.testbed            workq_test       user1                    0 Q workq
qstat -f 3269 | grep comment
comment = Not Running: PBS Error: Budget Manager: Consumption has reached the
limit for a dynamic service unit storage
```

1.6.2.4 Rules for Using Service Units

- You can change the type of a service unit, but there are restrictions:
  - You can change standard to dynamic only when no transactions have taken place for that service unit
  - You can change dynamic to standard only when no updates have been made to the usage of that service unit
- In the billing formula file, you can use only standard service units.
- All active dynamic service units must have a limit set in order for jobs to run
- When you create a new child period, it inherits its limits for any dynamic service units from its parent
- If you apply a limit directly to a dynamic service unit for a period, that overrides any inherited limit
- If you apply a limit to a period, all child periods that are not directly limited inherit the limit; similarly, limits on child periods are inherited by children of those child periods, when no direct limits have been set
- Only administrators can update dynamicvalues
- The maximum amount of each service unit an account can hold is 999999999999.99.

1.6.3 Transactions

A transaction is an operation on a service unit, such as a deposit or transfer. We say a transaction is an element in Budget. Budget also uses checks on account balances; these are not transactions, but they are also elements in Budget.

Table 1-1: Transactions and Checks

Name	Operation	Format	When	Purpose
Deposit	Deposit service units to user, project, or group account Depositing Service Units	Unique float	Any time	Investor funds group. Group manager funds user or project from group account.
Withdraw	Withdraw service units from user, project, or group account Withdrawing Service Units	Unique float	Any time	Investor withdraws funds from group. Group manager withdraws service units from user or project and returns them to group account.
Transfer	Move service units from one investor or user, project, or group account to another Transferring Service Units	Unique float	Any time	Administrator transfers service units between investors, groups, periods, projects, and users. Useful when period expires leaving unused funds; these funds can be transferred where needed.
Refund	Refund service units from group account to user or project account Refunding Service Units	Job ID plus date, time stamp, and operation (shown in report)	Any time	Administrator refunds job owner for costs out of job owner's control, for example re-run caused by node failure. Refunds are strictly per-job.
Acquire	Service units from job owner's account are put in escrow before running the job Acquiring Service Units	Job ID plus date, time stamp, and operation (shown in report)	Before job runs	Hook moves credit from job owner's account to escrow. Can be used as a debug tool by administrator.
Reconcile	Service units consumed by job are removed from escrow. Returns unused service units in escrow, or if job consumed more than requested, debits more from job owner account Reconciling Service Units	Job ID plus date, time stamp, and operation (shown in report)	After job runs	Hook reconciles accounts after job ends: debits escrow for amount consumed by job and returns unused credit to job owner, or if job consumed more than requested, debits more from the job owner's account. Can be used as a debug tool by administrator.
Precheck	Check job owner account for sufficient service units; disallow queueing job if insufficient Prechecking Service Unit Balance	Not a transaction; no ID	Before queueing job	Hook can optionally disallow queueing of jobs owned by accounts with insufficient service units. Can be used as a debug tool by administrator.
Checkbalance	Check job owner account for sufficient service units; do not run job if insufficient Checking Service Unit Balance	Not a transaction; no ID	Before job runs	Hook checks job owner credit balance before allowing job to run.

You can get reports on transactions; see section 4.2.5.5, “Getting Job and Transaction Reports”, on page 79.

1.6.3.1 Transaction IDs

Every transaction has a transaction ID; transactions associated with jobs, such as acquiring service units to run the job, consist of the job ID, a date, a timestamp, and an operation, shown in the report. For example:

```
1235.myserver   2020-10-06  17:15:23.760451  ...  acquired  ...
```

Other transactions have a unique floating-point number for their transaction ID. This format has 10 digits before the point and 7 after the point, for example:

```
0123456789.1234567
```

1.6.4 Accounting Policy

You must specify an accounting policy for each user and project when you add them to Budget. The policy determines how the entity is charged for its jobs. You can set the accounting policy when you add or update the entity via the `-A <accounting policy>` option to the `amgr add` or `amgr update` commands. There is no default policy. The accounting policy is case-sensitive, and it is lowercase. You have the following options:

begin_period
: The user or project account is charged when the job begins.

end_period
: The user or project account is charged when the job ends.

proportionate
: The user or project account is charged during all periods when the job runs, and each period is charged in proportion to the usage during that period.

1.6.5 Clusters

A cluster is data structure representing a PBS complex. Named for the PBS server; this is the value of the PBS_SERVER parameter in the `/etc/pbs.conf` file. Cluster names must be unique. The cluster formula must be the same as the formula at the complex, otherwise jobs won't run. If you log into a PBS complex and change the formula there, update the cluster data structure at the Budget host via `amgr update cluster`.

When you run `amgr add cluster` or `amgr update cluster`, you are also updating the database with the formula for the cluster.

A cluster has the following:

- Name; this is the name of the PBS server for the PBS complez, found in the PBS_SERVER parameter in the `/etc/pbs.conf` file
- Billing formula; see section 3.3.2, "Define Billing Formula", on page 40
- Is active or inactive

In order for a user or project to be able to run a job on a cluster, the cluster must be associated with the user or project.

Cluster operations:

- Administrator adds a cluster to Budget via `amgr add cluster`; see section 4.2.1.5, "Adding a Cluster", on page 53
- Administrator updates a cluster via `amgr update cluster`; see section 4.2.3.5, "Updating Clusters", on page 63
- Any user lists a cluster via `amgr ls cluster`; see section 4.2.2.5, "Listing Clusters", on page 57
- Administrator removes a cluster via `amgr rm cluster`; see section 4.2.4.5, "Removing a Cluster", on page 67. If the cluster has any associated jobs or transactions, the remove operation only makes the cluster inactive.

The difference between an active and an inactive cluster is that an active cluster can run jobs and participate in transactions, but an inactive cluster cannot. Otherwise they are the same.

1.7 Caveats and Troubleshooting

- The Budget administrator account must exist and have the *admin* role at all times. Make sure you never remove the *admin* role from your administrator account. If you have no administrator with *admin* role, you cannot use Budget. If you do not have another administrator account with the *admin* role, do not disable your administrator account as shown in section 2.9, “Changing Budget Administrator to New Username”, on page 33, step 3, Permanently disable pbsadmin.
- Soft walltime is not supported.
- Shrink-to-fit jobs are not supported.
- If the license server is not reachable, jobs will continue to run for up to 3 hours. After that, jobs cannot start or be reconciled.
- Losing connection with the AMS module:
  - If users can no longer log into or out of Budget, the connection with the AMS module may have been lost.
  - If Budget stops jobs from running, check the Budget logfile or a job comment. If you see Budget complaining about "connection refused", that means Budget can’t reach the AMS module.
  - If you reinstall the AMS module, that breaks its connection with Budget.

  To reestablish the connection, run this script as root:

  ```
  /opt/am/libexec/am_auth_register
  ```

1.8 Formats in Budget

1.8.1 Name Formats

- Name format for projects, groups, periods:

  Allowed characters: a-z,A-Z,0-9,_,-,.,$

  Max length: 256 characters
- Name format for service units:

  Allowed characters: a-z,A-Z,0-9,_,$

  Max length: 30 characters
- Name format for usernames is OS-dependent

2 Installing and Upgrading Budget

2.1 Whether or Not to Start with Failover

Consider configuring Budget for failover. Without failover, the AM_HOME directory is on a local drive, but with failover, it's on a separate host. We recommend doing a fresh install for failover, which means starting with an empty database. For failover configuration instructions, see section 2.7, "Configuring Budget for Failover", on page 29.

2.2 Supported Platforms for Budget

- CentOS 7, 8
- Red Hat 7, 8
- SLES 12, 15 with restrictions; see section 2.2.1, "SLES Restrictions", on page 19
- Ubuntu 18, 20
- For compute nodes only: Windows 10, 2012R2, 2016, 2019

2.2.1 SLES Restrictions

- Each SLES host must be registered with the SUSE Customer Center via SUSEConnect, and have a support contract. This happens automatically for cloud nodes.
- SLES hosts require Docker Enterprise Edition.

2.3 Prerequisites

2.3.1 Altair Software Components

- Budget module
- Budget client commands
- Budget authentication module (AMS)
- PBS Professional 2020.1 or later, installed and running
  - PBS Professional server/scheduler(s)/comm(s)
  - PBS Professional client commands
- Altair License Manager 14.5.1 or newer, installed and running
- PBSProNodes 20.0 license features

2.3.2 Third-party Software Components

- docker-ce v19.x or later for most systems
- docker-ee v19.x or later for SLES
- python3
- python3-pip
- openssl

2.3.3 Job Requirements

Each job must request the compute resources that are used in the billing formula used at that complex. For the default formula, this means walltime and ncpus. Make sure that every PBS job requests ncpus and walltime when it runs. Each job can have these set at submission by the job submitter or later via `qalter`, can inherit a value from the server or queue, or can be assigned a value by a hook. For ncpus, the server attribute default_chunk.ncpus may take care of the requirement.

Job walltime:

- If a job's walltime is extended, Budget takes that into account.
- Soft walltime is not supported.
- Shrink-to-fit jobs are not supported.

2.3.4 Required User Accounts

2.3.4.1 Budget Administrator

Budget requires an administrator. The administrator configures Budget.

We recommend that the username for the administrator account be *pbsadmin* (same account used for PBS administrator). You can use any username for the Budget administrator. Where you see "pbsadmin" in the instructions, substitute the actual administrator username.

You can install and configure Budget using pbsadmin for the administrator username, then switch to another username later. See section 2.9, "Changing Budget Administrator to New Username", on page 33.

2.3.4.1.i Requirements for Administrator Account

- This account must exist and have the *admin* role at all times. Make sure you never remove the *admin* role from your administrator account. If you have no administrator with *admin* role, you cannot use Budget.
- The administrator account should not be used for the teller.
- The home directory for pbsadmin should exist on the PBS server host, and on the Budget host, if it is a separate machine.
- Administrator account must have an account on the PBS server host, and on the Budget host, if it is a separate machine.
- Administrator should be able to `ssh` to the Budget server host without a password. Administrator must have passwordless `ssh` set up from the PBS server host to the Budget server host. We cover this in section 3.1.6, "Set Up Passwordless SSH from PBS Server to Budget Server", on page 38.

2.3.4.2 Teller

Budget requires a teller user to process its service unit transactions. When the Budget hook performs transactions, it does so as the teller.

2.3.4.2.i Requirements for Teller Account

- We recommend that the username for the teller account be *amteller*
- Teller should be a dedicated account used only for the *teller* role
- Teller account must have an account on the PBS server host, and on the Budget host, if it is a separate machine.
- Teller should be able to `ssh` to the Budget server host without a password. Teller must have passwordless `ssh` set up from the PBS server host to the Budget server host. We cover this in section 3.1.6, "Set Up Passwordless SSH from PBS Server to Budget Server", on page 38.
- The teller account should not be used for pbsadmin
- The teller account does not need to be root or administrator

2.3.4.3 Database User

Budget requires a database user for its database.

2.3.4.3.i Requirements for Database User Account

- We recommend that the username for the database user account be *pbsdata*.
- The pbsdata account should have an ID <1000 so that any processes which run under this user are protected from the Out Of Memory (OOM) killer and run with the correct level of privilege.

2.3.4.4 Job Submitters

Users who submit jobs to PBS Professional have to exist on the system where Budget is installed, but these users can be added to Budget after installation.

2.3.4.4.i Requirements for Job Submitters

- Job submitters must already be able to run jobs in a PBS complex.
- Each user you add to Budget should already have an entry in the password file, with a password set, and a home directory on the Linux system where Budget is installed.

2.3.4.5 Configuring Required Accounts for Budget

- Add pbsadmin as the Budget administrator, and set a password:

```
adduser -u 901 pbsadmin
passwd pbsadmin <password>
```

- Add amteller as the Budget teller, and set a password:

```
adduser amteller
passwd amteller <password>
```

- Add pbsdata as the Budget database user, and set a password:

```
adduser -u 900 pbsdata
passwd pbsdata <password>
```

2.4 Recommended Configurations

Head node and service node can be one of either:

- Both on premises
- Both in cloud

Do not put one on premises and one in the cloud.

2.4.1 Recommended Configuration for Larger Installations

For larger installations using on premises hosts with optional cloud bursting:

- Hosts on premises for PBS server, scheduler, some MoM daemons, cloud service node
- Head, service, and first N execution nodes are on premises
- Extra execution nodes can be burst
- On head node, PBS and Budget
- On service node
  - AMS designed to run in container
- VPN connection to the cloud you will use
- Client commands go on any Linux host

Notes:

- You many not want to run AMS on the head node, because it runs in a Docker container, which may impose too high a load.
- All components are mix-and-match (with Docker restriction).
- For PBS configuration instructions, see the *PBS Professional Administrator's Guide*.

2.4.2 Recommended Configuration for Smaller Installations

For smaller installations and cloud-only installations where the workload is low enough:

- All PBS components can be hosted in the cloud
- Cloud headnode, running PBS Professional server and Budget module
- All components can run on the same node
- You can run Docker on the same node as the PBS components
- Client commands go on any Linux host, but a user must be able to reach the Budget port on the cloud host

Notes:

- No VPN is required for this configuration.

2.5 Installation Steps

You must be root to install Budget. Log in as root.

2.5.1 Install Utilities and Docker

1. Install utilities and `docker`:
   - For CentOS or RedHat:

     Log in to the machine where the Control package or the Cloud package is to be installed.

     ```
     yum install -y yum-utils
     yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
     yum install docker-ce docker-ce-cli containerd.io
     systemctl enable docker
     systemctl start docker
     yum install python3 python3-pip
     yum install openssl
     ```
   - For SLES12 or 15:

     Log in to the machine where Budget is to be installed.

     For SLES 12:

     ```
     sudo SUSEConnect -p sle-module-containers/12/x86_64 -r ''
     ```

     For SLES 15:

     ```
     sudo SUSEConnect -p sle-module-containers/15.1/x86_64 -r ''
     sudo zypper install docker
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```

     Configure the firewall to allow forwarding of Docker traffic to the external network:

     ```
     Set FW_ROUTE="yes" in /etc/sysconfig/SuSEfirewall2
     zypper install python3-pip
     ```
   - For Ubuntu:

     Log in to the machine where Budget is to be installed.

     ```
     sudo apt-get update
     sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-proper-
         ties-common
     curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
     sudo apt-key fingerprint 0EBFCD88
     ```

     The key should match the second line in the output; validate the last 8 characters. Example of second line:

     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

     ```
     sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
         $(lsb_release -cs) stable"
     sudo apt-get update
     sudo apt-get install docker-ce docker-ce-cli containerd.io
     sudo apt-get install python3-pip
     ```

     This can install a number of required dependencies, and may take a few minutes.

     ```
     sudo apt-get install openssl
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```
2. Copy these installer tar files to your host machine:

- The AMS installer, named:

  `ams_installation.tar.gz`
- The Budget installer file, named:

  PBSPro-budget-server_<release number>-<OS>_x86_64.tar.gz

  For example:

  `PBSPro-budget-server_2021.1.0-CentOS7_x86_64.tar.g`

2.5.2 Install AMS Module

The AMS module provides the security framework for Budget. Budget uses AMS to retrieve tokens, verify them, and authenticate users. When you install Budget, the Budget server is registered as a client of AMS. Every Budget request has an authorization token.

Install the AMS module on the Budget server host.

1. Install AMS:

   `tar xvfz ams_installation.tar.gz`

   `cd ams-installation`

   `python3 -m pip install --upgrade --ignore-installed pbsworks-packager/`

   `/usr/local/bin/pkgr` (Please stay in the AMS installer directory for this step)
2. Answer the questions in the dialogue:
   a. Choose option: 0 (choose the AMS package)
   b. Hit *Enter* until you've seen the whole license agreement, then answer *Yes* to accept
   c. Select *Enter* to continue
   d. Choose Option: 1 (yes, add hostname resolution)

      <Budget server hostname>

      <Budget server IP address>
   e. Choose Option: 0 (no, stop adding hostname resolution)
   f. Install Location: <Budget server hostname>
   g. Authentication Server: <authentication daemon hostname>
   h. Authentication Port (this is the port for the authentication daemon, typically `sshd`): <port number>
   i. Provide administrator username: <administrator username>
   j. Install Path: <install path>
3. Once installation completes check the AMS service status:

   `systemctl status altaircontrol`

2.5.3 Enable Passwords in Docker Container Network

AMS creates a network inside the Docker container(s). This network is named "altair_control_main". Enabling passwords for this network allows any Budget user to use the Budget module. This allows them to use their entry in `/etc/passwd`.

Edit `/etc/ssh/sshd_config` and add the following lines:

```
Match Address 10.5.0.0/24
PasswordAuthentication yes
```

Make `sshd` reread its configuration file, and restart it:

```
systemctl daemon-reload
systemctl restart sshd
```

2.5.4 Create Certificates for Budget Daemon Communication Encryption

Create the certificates required to encrypt communication between the Budget and database daemons:

1. Make required directory:

```
cd /home/pbsadmin/
mkdir budget_certificates
export AM_DBUSER=pbsdata
```

2. Create a key pair that will serve as both the root CA and the server key pair. This key pair is for 10 years (3650 days):

```
openssl req -new -x509 -days 3650 -nodes -out budget_certificates/ca.crt -keyout
    budget_certificates/ca.key -subj "/CN=root-ca"
    Generating a 2048 bit RSA private key
    ...
    writing new private key to 'budget_certificates/ca.key'
    ...
```

3. Create the server key and CSR:

```
openssl req -new -nodes -out server.csr -keyout budget_certificates/server.key -subj "/CN=local-
    host"
    Generating a 2048 bit RSA private key
    ...
    writing new private key to 'budget_certificates/server.key'
    ...
```

4. Sign the CSR using the root key:

```
openssl x509 -req -in server.csr -days 3650 -CA budget_certificates/ca.crt -CAkey
    budget_certificates/ca.key -CAcreateserial -out budget_certificates/server.crt
    Signature ok
    subject=/CN=localhost
    Getting CA Private Key
```

5. Create client certificate for database user (typically pbsdata):

```
openssl req -new -nodes -out client.csr -keyout budget_certificates/client.key -subj
    "/CN=${AM_DBUSER}"
    Generating a 2048 bit RSA private key
    ...
    writing new private key to 'budget_certificates/client.key'
    ...
openssl x509 -req -in client.csr -days 3650 -CA budget_certificates/ca.crt -CAkey
    budget_certificates/ca.key -CAcreateserial -out budget_certificates/client.crt
    Signature ok
    subject=/CN=pbsdata
    Getting CA Private Key
```

6. Remove unneeded intermediate files:

```
rm -f server.csr client.csr
```

7. Set suitable permissions to protect certificates:

```
chmod og-rwx budget_certificates/*
```

8. View files:

```
cd budget_certificates/
ls -l
    total 28
    -rw-------. 1 root root 1090 Jan 7 14:45 ca.crt
    -rw-------. 1 root root 1704 Jan 7 14:45 ca.key
    -rw-------. 1 root root 17 Jan  7 14:56 ca.srl
    -rw-------. 1 root root 973 Jan  7 14:56 client.crt
    -rw-------. 1 root root 1704 Jan 7 14:55 client.key
    -rw-------. 1 root root 973 Jan  7 14:54 server.crt
    -rw-------. 1 root root 1704 Jan 7 14:52 server.key
pwd
    /home/pbsadmin/budget_certificates
```

2.5.5 Install Budget Module

You can install and configure Budget using "pbsadmin" as the Budget administrator username, then switch to a different administrator username later; see section 2.9, “Changing Budget Administrator to New Username”, on page 33.

1. Extract software:

   tar xvfz PBSPro-budget-server_<release number>-<OS>_x86_64.tar.gz

   For example:

   tar xvfz PBSPro-budget-server_2021.1.0-CentOS7_x86_64.tar.gz

2. Change to new directory:

   cd /root/software/am/

3. Run the Budget installer, and choose the username you want for the Budget administrator. In our example we use *pbsadmin* for the administrator username:

   ./install -t server -u pbsadmin -c /home/pbsadmin/budget_certificates

```
Installing Budget Manager
You have selected server.
AM_EXEC does not exist. Will make it.
AM_HOME does not exist. Will make it.
Installing Budget Manager server...
**
**
Copying source to /opt/am
**
**
Budget Manager installed successfully.
sed -i 's/AMADMIN/pbsadmin/g' /opt/am/db/am_database.sql
sed -i 's/AMADMIN/pbsadmin/g' /opt/am/libexec/am_postinstall
sed -i 's|@AM_EXEC@|/opt/am|g' /opt/am/libexec/pbs_budget.service
sed -i 's|@AM_HOME@|/var/spool/am|g' /opt/am/libexec/pbs_budget.service
sed -i 's|@AM_SERVER@|testbed|g' /opt/am/libexec/pbs_budget.service
sed -i 's|@AM_PORT@|8000|g' /opt/am/libexec/pbs_budget.service
cp -rp /opt/am/libexec/pbs_budget.service /usr/lib/systemd/system/pbs_budget.service
```

2.5.6 Enable and Start Budget

- Enable and start Budget:

  systemctl enable pbs_budget

  systemctl start pbs_budget

- Check the status of Budget:

  systemctl status pbs_budget

2.6 Validating Budget

1. Log in as pbsadmin
2. Test authentication:

   ```
   amgr login
   ```

3. List users (pbsadmin and amteller):

   ```
   amgr ls user -l
   ```

2.7 Configuring Budget for Failover

Budget uses Pacemaker and Corosync to manage failover for the Budget and data service daemons. You set up a primary Budget server and a secondary Budget server, and a shared filesystem used by either one and available to both, as a *failover cluster*. The primary normally handles all server traffic, but the secondary takes over and becomes active if the primary becomes unavailable. Failover for the Budget daemon and the data service daemon happens together. Budget starts the data service on the same host where the Budget daemon runs. Pacemaker and Corosync manage the flow of traffic so that it goes to the active server and database.

2.7.1 Prerequisites for Configuring Budget Failover

2.7.1.1 Third-party Software Prerequisites

- Pacemaker 1.1.23
- Corosync 2.4.5 or later
- pcs 0.9.169

2.7.1.2 Budget Server Host Prerequisites

- Budget server hosts must be identical
- Both server hosts must have access to shared filesystem
- Each server host should have two network connections:
  - Dedicated private ethernet connection that Pacemaker and Corosync use to manage host state
  - Public network that Budget client commands will use for communication traffic with Budget server
- Each server host should have two hostnames:
  - Private hostname for use with private ethernet connection, for example "PrimaryFailover" and "Secondary-Failover"
  - Public hostname for use with client commands, for example "Primary" and "Secondary"
- Same version of Budget on both servers
- Accounts for administrator, investors, managers, and teller must be identical on both hosts

2.7.1.3 Filesystem Prerequisites

- Shared AM_HOME filesystem on a third host, that is mounted on both Budget server hosts and always available to both hosts
- No root squash on shared filesystem
- Hostname resolution must work both ways between Budget server hosts, and between both servers and any other hosts involved, for example PBS server hosts
- The AM_HOME directory must be readable and writable from both servers by the Budget administrator

2.7.1.4 Optional

- STONITH script, if required. The administrator writes this script.

2.7.2 Installing Corosync, Pacemaker, and pcs

On each Budget server host:

1. Log into each Budget server host as root.
2. Install Budget if it is not already installed. See section 2.5, “Installation Steps”, on page 22.
3. Install Pacemaker, Corosync, and pcs.
4. When you install Pacemaker, the installation creates the hacluster account, which is the proxy user for Pacemaker. You need to give hacluster a password on each server host:

   ```
   passwd hacluster
   <password for hacluster account>
   ```

5. Create a budget directory in the ocf library:

   ```
   mkdir /usr/lib/ocf/resource.d/budgetmanager
   ```

6. Copy ocf files am_ocf and am_data_ocf from the Budget package to the budget directory:

   ```
   cp /root/am/failover/budgetmanager/* /usr/lib/ocf/resource.d/budgetmanager/
   ```

7. Set the permissions for the budget directory:

   ```
   chmod -R +x /usr/lib/ocf/resource.d/budgetmanager
   ```

2.7.3 Configuring Pacemaker

1. If Budget is running on any host, stop Budget:

   ```
   systemctl disable pbs_budget
   systemctl stop pbs_budget
   ```

2. Log into the primary Budget server host as root

3. Authorize Pacemaker to use the server hosts, using their private hostnames:

   ```
   pcs cluster auth <private primary hostname> <private secondary hostname>
   ```

4. Create the failover cluster, name it "am_cluster", and start it, using the private hostnames:

   ```
   pcs cluster setup --start --name am_cluster <private primary hostname> <private secondary host-
       name>
   ```

5. Set the quorum policy to *ignore*:

   ```
   pcs property set no-quorum-policy=ignore
   ```

6. Set whether or not stonith is enabled. If you have a STONITH script and want to use it, you can enable stonith. If not, disable it:

   ```
   pcs property set stonith-enabled=false
   ```

7. Create the virtual_ip resource to represent a virtual IP address. This IP address should be exclusively allocated for this purpose and not used by any physical host on your network. This is the IP address that client commands will use to connect to the active Budget server over the public network. Pacemaker and Corosync direct traffic from this IP address to whichever server is active:

   ```
   pcs resource create virtual_ip ocf:heartbeat:IPaddr2 ip=<virtual IP address> cidr_netmask=32
       nic=<nic-name-of-public-network> op monitor interval=10s
   ```

8. Create the amserver resource to represent the Budget daemon. Note that here we are assuming `am.conf` is in `/etc`. Make sure you use the appropriate location:

   ```
   pcs resource create amserver ocf:budgetmanager:am_ocf conf=/etc/am.conf op monitor interval=15s
   ```

9. Add a constraint to Pacemaker so that it always keeps the virtual_ip and amserver resources on the same host. When one moves, the other goes with it:

   ```
   pcs constraint colocation add amserver virtual_ip INFINITY
   ```

10. Make the primary server host be the preferred server host:

    ```
    pcs constraint location amserver prefers <primary>
    ```

11. Budget is automatically started on the primary server host. You do not have to start Budget.

2.7.4 Caveats and Recommendations for Failover

- Make sure only one Budget server daemon is running at a time. Do not start a second Budget server. If two instances of Budget are active at the same time, the database will become corrupted.
- Make sure that the shared `AM_HOME` directory is always available on both the primary and secondary server hosts. Do not prevent either host from reaching `AM_HOME`.

2.7.5 Starting, Stopping, and Statusing Budget with Failover Configured

- To start Budget when failover is configured:

  On each Budget server host, in any order:

  ```
  pcs cluster start
  ```

- To stop Budget when failover is configured, first stop the secondary, then the primary:

  ```
  pcs cluster stop <secondary server host>
  pcs cluster stop <primary server host>
  ```

- To check the status of Budget when failover is configured:

  ```
  pcs cluster status
  ```

2.8 Upgrading Budget

1. Log in to the Budget server as root
2. Turn off scheduling in all associated PBS Professional complexes
3. Stop Budget (ideal for backup, but not essential)
4. Back up /var/spool/am
5. Make a copy of the formula file used for each cluster. If you don't have these on hand, export them at each PBS server host:

   ```
   qmgr -c 'export hook am_hook application/x-config default' > am_hook.json
   ```

6. Create certificates; see Create Certificates for Budget Daemon Communication Encryption
7. Allow Budget to work with PBS Professional by updating sudoers file; see section 3.1.4, “Allow Interaction with PBS Professional”, on page 36
8. Untar the new install package; this creates the am directory
9. Change to the am directory and install Budget:

   ```
   cd am
   ./install -t upgrade -c /home/pbsadmin/budget_certificates/
   ```

10. Installer asks whether scheduling is disabled; enter "Y"

Installer actions:

- The installer backs up your database. If this fails, it aborts the upgrade. If the upgrade fails, the installer restores the backup you just made, and you can reinstall the older version of Budget to return to operation. You can install it in place over the existing data, and it will pick up where it left off.
- After backing up the data, the installer uninstalls the old Budget, installs the new one, updates the database schema and updates the data. This could take some minutes if there is a lot of data.
- The installer updates `am.conf`.

11. Restart Budget:

```
systemctl restart pbs_budget
```

12. This version of Budget comes with a new hook in a .py file. Configure the hooks; see Create and Configure Budget Hooks.
13. Update the formula for each cluster with the file from its PBS complex:

```
amgr update cluster -n <PBS server> -f <formula filename>
```

14. Enable scheduling at each associated PBS complex.

Budget is ready to track and manage service units.

2.9 Changing Budget Administrator to New Username

You can install and configure Budget using one administrator username, then switch to another later. To switch to a new administrator username:

1. Make sure the new administrator account meets the criteria in section 2.3.4.1, "Budget Administrator", on page 20.

- Create the new administrator username, and set a password:

```
adduser -u 901 <new administrator username>
passwd <new administrator username> <password>
```

2. Add the new account to Budget, and assign it the *admin* role:

```
amgr add user -n <new administrator username> -r admin -A  <accounting policy> -c <PBS server>  -r
    <role> [-h <group list>] [ -a <active>]
```

3. Permanently disable pbsadmin.

   Optional. This is **not recoverable**. You can remove *admin* role from pbsadmin, or deactivate pbsadmin:

```
amgr update user -n pbsadmin -r <new role>
amgr update user -n pbsadmin -a False
```

3 Configuring Budget

3.1 Initial Configuration

3.1.1 Budget Configuration Parameters

The configuration file for Budget is `/etc/am.conf`. Budget uses the following configuration parameters:

Table 3-1: Budget Configuration Parameters

Configuration Parameter	Description	Default Value
AM_AUTH_ENDPOINT	Path to AMS module. Format: *<port>@<hostname>*	*9100@<AMS host>*
AM_AUTH_TIMEOUT	Optional. Number of seconds to wait when authenticating. Integer. Set this to a larger value if you have a problem with authentication timeout.	*10*
AM_BALANCE_PRECHECK	Boolean. Directs hook to precheck account balance when job is queued	*False*
AM_DBPORT	Port number for Postgres database	*9876*
AM_DBUSER	Username of database user	*pbsdata*
AM_EXEC	Path to Budget executables	*/opt/am*
AM_HOME	Home directory for Budget	*/var/spool/am*
AM_LICENSE_ENDPOINT	Path to license daemon. Format: *<port>@<hostname>*	*6200@<ALM host>*
AM_PORT	Port number on which Budget listens	*8000*
AM_SERVER	Hostname of Budget host	Host where Budget is installed
AM_WORKERS	Number of workers to spawn. We recommend not changing this setting. For help, contact Altair support.	*2*

3.1.1.1 Example Configuration File

Example of /etc/am.conf:

```
AM_PORT=8000
AM_EXEC=/opt/am
AM_HOME=/var/spool/am
AM_WORKERS=2
AM_DBUSER=pbsdata
AM_DBPORT=9876
AM_AUTH_ENDPOINT=9100@my_ams_host
AM_LICENSE_ENDPOINT=6200@my_alm_host
AM_SERVER=my_budget_host
```

3.1.2 Set Budget Configuration Parameters

The /etc/am.conf file is set to the defaults for the machine on which it is installed. For a description of the Budget configuration parameters, see section 3.1.1, "Budget Configuration Parameters", on page 35. Edit /etc/am.conf, and set the parameters appropriately.

3.1.3 Set Budget Environment Variables

Prepend executable path to PATH environment variable for all users of Budget (administrator, investors, managers, teller, job submitters):

```
export PATH=$AM_EXEC/python/bin:$PATH
```

For the default path, this is:

```
export PATH=/opt/am/python/bin:$PATH
```

3.1.3.1 Caveats and Advice for Budget Configuration Parameters

Set AM_AUTH_TIMEOUT to a larger value if you have a problem with authentication timeout.

3.1.4 Allow Interaction with PBS Professional

In order to work on hooks, you have to be root, so we add specific actions to the sudoers file on the Budget and PBS server hosts for pbsadmin to work as root when using amgr and qmgr to import and export hook files.

3.1.4.1 Budget Server and PBS Server on Same Host

On the Budget/PBS server host, edit /etc/sudoers, and add the following lines. Replace the $AM_EXEC and $PBS_EXEC variables with the actual paths.

- Allow Budget administrator to act as root for Budget commands:
  Cmnd_Alias AM_SERVER_CMD = $AM _EXEC/python/bin/python3 $AM_EXEC/hooks/pbs/pbs_set_formula.py*
- Allow Budget administrator to set the formula as root:
  Defaults!AM_SERVER_CMD !requiretty
- Allow teller to get a security token:
  Cmnd_Alias AM_CLIENT_CMD = $AM_EXEC/python/bin/amgr sshlogin
- Allow amgr command to update hook formula:
  Cmnd_Alias AM_IMPORTS = $PBS_EXEC/bin/qmgr -c i h am_hook application/x-config default .am/tmp*,
  $PBS_EXEC/bin/qmgr -c i h am_hook_periodic application/x-config default .am/tmp*,
  $PBS_EXEC/bin/qmgr -c export hook am_hook application/x-config default
- Allow the Budget administrator account to act as root for the qmgr commands listed above:
  <budget administrator> ALL=(root) NOPASSWD: BUDGET_MANAGER_IMPORTS
- Allow hooks to work in the background with no tty interface:
  Defaults!AM_CLIENT_CMD !requiretty
- Allow hooks to work in the background with no tty interface:
  Defaults!AM_IMPORTS !requiretty

3.1.4.2 Budget Server and PBS Server on Separate Hosts

3.1.4.2.i Changes to sudoers on Budget Server Host

On the Budget server host, edit /etc/sudoers, and add the following lines. Replace the $AM_EXEC and $PBS_EXEC variables with the actual paths.

- Allow Budget administrator to act as root for the qmgr commands listed above:
  Cmnd_Alias AM_SERVER_CMD = $AM _EXEC/python/bin/python3 $AM_EXEC/hooks/pbs/pbs_set_formula.py*
- Allow the Budget administrator to set the formula as root:
  Defaults!AM_SERVER_CMD !requiretty

3.1.4.2.ii Changes to sudoers on PBS Server Host

On the PBS server host, edit /etc/sudoers, and add the following lines. Replace the $AM_EXEC and $PBS_EXEC variables with the actual paths. We give an example showing default paths below:

- Allow teller to get a security token:
  ```
  Cmnd_Alias AM_CLIENT_CMD = $AM_EXEC/python/bin/amgr sshlogin
  ```
- Allow amgr command to update hook formula:
  ```
  Cmnd_Alias AM_IMPORTS = $PBS_EXEC/bin/qmgr -c i h am_hook application/x-config default .am/tmp*,
      $PBS_EXEC/bin/qmgr -c i h am_hook_periodic application/x-config default .am/tmp*,
      $PBS_EXEC/bin/qmgr -c export hook am_hook application/x-config default
  ```
- Allow the Budget administrator account to act as root for the qmgr commands listed above:
  ```
  <budget administrator> ALL=(root) NOPASSWD: BUDGET_MANAGER_IMPORTS
  ```
- Allow hooks to work in the background with no tty interface:
  ```
  Defaults!AM_CLIENT_CMD !requiretty
  ```
- Allow hooks to work in the background with no tty interface:
  ```
  Defaults!AM_IMPORTS !requiretty
  ```

3.1.5 Create Resources

```
qmgr -c "c r am_job_amount type=string"
qmgr -c "c r am_job_cache type=string,flag=m"
```

3.1.6 Set Up Passwordless SSH from PBS Server to Budget Server

Both the administrator and the teller need to have passwordless ssh available from the PBS server host to the Budget server host. Perform these steps once for the administrator, and once for the teller.

Generate a public authentication key at the PBS server host and append it to the ~/.ssh/authorized_keys file at the Budget server host:

1. Log in to the PBS server host as the username who needs passwordless ssh
2. Get an SSH key pair. If you find existing keys, you can use those, or you can back up the old keys and generate a new pair:
   - To check for an existing SSH key pair:
     ls -al ~/.ssh/id_*.pub
   - To generate a new SSH key pair:
     ssh-keygen
1. Copy the contents of id_rsa.pub
2. Log in to the Budget server as the username who needs passwordless ssh
3. In your home directory, check for the .ssh directory. If it does not exist, create it:

   mkdir -p .ssh
   cd .ssh/
4. Create the authorized_keys file in the .ssh directory:

   vi authorized_keys
5. Paste the contents of id_rsa.pub that you copied from the PBS server host and save the file.
6. Change the permission of authorized_keys to *600*:

   chmod 600 authorized_keys

3.2 Defining Billing Periods

Choose and define your billing periods. We describe billing periods in section 1.6.1, “Periods, Allocation Periods, Billing Periods”, on page 11.

Make sure that periods at the same level do not overlap. For example, if Quarter1 ends March 31st, make sure that Quarter2 does not begin sooner than April 1st.

If you want to create periods with a parent-child relationship, you must create the parent period first. You cannot add a parent to an existing child. For example, if you want Year as the parent and Quarter1, Quarter2, etc., as children, create Year first.

If you create child periods, make sure that they fit within the parent period.

Make sure that your period hierarchy is finalized BEFORE doing any transactions or running any jobs; you cannot update or remove periods once transactions have been performed or jobs have started.

To define each billing period, add it to Budget:

amgr add period -n <period name> -S <start date> -E <end date> [-p <name of parent period>]

See section 4.2.1.6, “Adding a Period”, on page 53.

3.3 Adding a PBS Complex and Setting its Billing Model

Budget interacts with the PBS server via two hooks and hook configuration file containing a billing formula. Budget uses two identical hooks, named am_hook and am_hook_periodic; both hooks use the same configuration file. The formula defines the *billing model* for that PBS complex. You can set a different billing formula for each PBS complex.

To use a PBS complex with Budget:

- At the PBS server host, the administrator creates and configures the hooks, and defines the formula to use when billing
- At the Budget server host, the administrator adds a cluster data structure to Budget that will represent the PBS complex, and sets the formula at the cluster to be the same as the one at the PBS complex

3.3.1 Steps to Add Complex and Set Billing Model

1. Log into the Budget server host
2. Optionally modify the formula file as desired; see section 3.3.2, “Define Billing Formula”, on page 40
3. Log into the PBS server host
4. If the PBS server host is different from the Budget server host, copy the hook and formula files from your Budget installation to the PBS server host
5. At the PBS server host, create and configure the Budget hooks; see section 3.3.3, “Create and Configure Budget Hooks”, on page 44
6. At the Budget server host, add a cluster data structure that will represent the PBS complex and specify the formula you used for the PBS complex:

   amgr add cluster -n <PBS server> -f <formula filename>

   For example, to add a PBS complex whose server is named HPC1, and use the formula defined in `formula_hpc1.json`:

   ```
   amgr add cluster -n HPC1 -f formula_hpc1.json
   ```

   See section 4.2.1.5, “Adding a Cluster”, on page 53.

3.3.2 Define Billing Formula

Budget uses a billing formula, in which you define how service units are calculated. This formula can be different for each cluster.

3.3.2.1 Billing Formula File and Format

We provide a default billing formula file named am_hook.json; you can modify it to fit your needs. Name it <formula file>.json.

The billing model is defined in a billing formula file in JSON format.

You can create formulas using Python.

You must include this line:

```
"auth_user": "amteller",
```

3.3.2.1.i Constants

If you want to use a constant in the formula, you have to define it in the "constants" section.

Define constants by using the prefix "CONST_". Constants should be defined as floating point numbers.

Example 3-1: Formula with variables and a constant:

```
{
    "auth_user": "amteller",
    "constants": {
        "ncpus": "job.ncpus",
        "walltime": "job.walltime",
        "CONST_time": 0.01667
    },
    "formulas": {
        "cpu_hrs": "ncpus*walltime*CONST_time"
    }
}
```

3.3.2.1.ii PBS Resources and Attributes

If you want to use a PBS resource or attribute as a variable in the formula, you have to define it in the "constants" section. You can use the following PBS Professional elements:

- Job, server, queue, and node attributes that are of type float or integer
- Built-in or custom resources that are of type float or integer
- Built-in resources that are of type duration: walltime, cput, and eligible time
- Built-in resources that are of type size: mem and vmem

Note that if the value of an attribute or resource has not been set, you cannot access it.

The syntax for specifying resources in the formula file is different from the syntax you would use in a normal hook:

- Before and while running, job resources are read from the job's Resource_List attribute. After the job runs (meaning after the execjob_epilogue hook runs), job resources are read from the job's resources_used attribute. The syntax for specifying a job resource in the formula file is different from other hook operatons:

  job.<resource name>

  For example, to specify what would be job.Resource_List["ncpus"]:

  `job.ncpus`
- Server, queue, and vnode resources are read from the resources_available attribute. The syntax for specifying resources at the server, queue, or vnode is slightly different from other hook operations (you omit the quotes):

  <object>.resources_available[<resource name>]

  For example, to use the value of what would be server.resources_available["charge_rate"] in the usual hook syntax:

  `server.resources_available[charge_rate]`

For jobs that run on multiple vnodes, the hook uses the sum of the resources used across all sister vnodes.

Example 3-2: Formula using job resources and a queue attribute:

```
{
    "auth_user": "amteller",
    "constants": {
        "ncpus": "job.ncpus",
        "ngpus": "job.ngpus",
        "walltime": "job.walltime",
        "priority": "queue.Priority",
        "CONST_threshold_prio": 1,
        "CONST_low": 0.5,
        "CONST_high": 1.0
    },
    "formulas": {
        "cpu_hrs": "((ncpus+ngpus)*walltime)*(priority < CONST_threshold_prio and CONST_low or
    CONST_high)"
    }
}
```

3.3.2.1.iii Operators

- Logical operators (AND and OR)
- Conditional operators (>,>=,<,<=,==,!=).

3.3.2.1.iv Units

The PBS resources walltime, cput, and eligible_time are in *duration* format (hh:mm:ss). The hook converts the value of these to seconds when used in the formula for the service unit. For example, if a job's walltime is two minutes, the value used in the formula is 120.

The PBS resources mem and vmem are in *size* format (3b, 20kb etc.). The hook converts their values to kb in the service unit formula.

The following table lists the units you can use in a billing formula:

Table 3-2: Units in Billing Formula

Formula Item	Units	Example
Constants	*Floating point*	12.34
Time-based PBS resources, e.g. walltime, cput, eligible_time	*Integer seconds*	120 (2 minutes)
Memory PBS resources, e.g. mem, vmem	*kb*	1048576kb (1gb)
ncpus	*Integer*	4 (ncpus=4)

3.3.2.2 Defining Service Units

In order to use a standard service unit, you must define it in the formula file. You define the formula for each service unit in the "formulas" section. Make sure that the name you use for the service unit in the formula is identical to the name you use in the `amgr add serviceunit` command. The service unit name should consist only of alphanumeric and underscore characters; blank spaces are not allowed.

You do not need to define dynamic service units in the formula file.

The default service unit used for billing is cpu_hrs, which is defined as (ncpus x walltime). We show the default for the "formulas" section here:

```
"formulas": {
    "cpu_hrs": "ncpus*walltime*CONST_time"
}
```

See section 1.6.2, “Service Units”, on page 12.

3.3.2.3 Default Billing Formula File

This is the default am_hook.json configuration file used by the Budget hook. This shows how the default service unit cpu_hrs is computed.

```
{
    "auth_user" : "amteller",
    "constants":{
        "ncpus": "job.ncpus",
        "walltime": "job.walltime"
    },
    "formulas":{
        "cpu_hrs": "ncpus*walltime"
    }
}
```

3.3.2.4 Formula File Examples

Example 3-3: Configuration file using multiple constants and operators. We define the constants CONST_a, CONST_b, and CONST_prio. We use them in the formulas section with logical operators 'and', 'or', the comparison operator '>', and the arithmetic operator '*'.

```
{
    "auth_user" : "amteller",
    "constants":{
        "ncpus": "job.ncpus",
        "walltime": "job.walltime",
        "ngpus": "job.ngpus",
        "CONST_a": 2.0,
        "CONST_b": 1.0,
        "CONST_prio": 150
    },
    "formulas":{
        "cost":"(ngpus and node.resources_available[ncpus] or ncpus)*walltime*(
            queue.Priority > CONST_prio and CONST_a or CONST_b)"
    }
}
```

Example 3-4: Formula with three different service units called cpu_hrs, gpu_hrs, and mics_hrs:

```
{
    "auth_user" : "amteller",
    "constants":{
        "ncpus" : "job.ncpus",
        "walltime" : "job.walltime",
        "ngpus" : "job.ngpus",
        "nmics" : "job.nmics"
    },
    "formulas":{
        "cpu_hrs" : "ncpus*walltime",
        "gpu_hrs" : "ngpus*walltime",
        "mics_hrs" : "nmics*walltime"
    }
}
```

3.3.2.5 Caveats and Advice for Billing Formula

Note that if the value of a PBS attribute or resource has not been set, you cannot access it in the formula.

3.3.3 Create and Configure Budget Hooks

1. Log into the PBS server host.
2. If the PBS server host is different from the Budget server host, copy the hook and configuration files to the PBS server host.
3. Create and configure the am_hook and am_hook_periodic hooks:

```
qmgr -c "c h am_hook"
qmgr -c "s h am_hook event='queuejob,runjob,execjob_epilogue,modifyjob,movejob'"
qmgr -c "s h am_hook order=1000"
qmgr -c "c h am_hook_periodic"
qmgr -c "s h am_hook_periodic event=periodic"
qmgr -c "s h am_hook_periodic freq=120"
qmgr -c "i h am_hook application/x-python default  /opt/am/hooks/pbs/am_hook.py"
qmgr -c "i h am_hook_periodic application/x-python default  /opt/am/hooks/pbs/am_hook.py"
qmgr -c "i h am_hook application/x-config default /opt/am/hooks/pbs/<formula file>.json"
qmgr -c "i h am_hook_periodic application/x-config default /opt/am/hooks/pbs/<formula file>.json"
qmgr -c "s h am_hook enabled=true"
qmgr -c "s h am_hook_periodic enabled=true"
qmgr -c "s h am_hook alarm=90"
```

3.3.4 Changing the Billing Formula

If you change the formula you use at a PBS complex, you can update the cluster with the new formula:

amgr update cluster -n <PBS server> -f <formula filename>

For example, to change the formula for a PBS complex whose server is named HPC1, and use the formula defined in new_formula_hpc1.json:

```
amgr update cluster -n HPC1 -f new_formula_hpc1.json
```

See section 4.2.3.5, "Updating Clusters", on page 63.

3.3.5 Caveats and Advice

- Do not change a formula when jobs are running. To change the formula, drain the system of jobs during a maintenance window and update the formula, then put the system back into production.
- See the *PBS Professional Plugins (Hooks) Guide* for more information.

3.3.5.1 Caveats and Advice on Budget Hook

- Do not try to combine the budget hooks into one hook.
- Do not try to change the formula directly at the PBS complex, either by modifying the configuration file directly or by importing it. If the formula file for the Budget hook is changed directly at the PBS complex, and not through the `amgr update` command, Budget will throw an error.
- The am_hook has to run after all the other hooks for each event. Make sure it has the highest order. For example, you can set it to a high order value such as 1000:

  ```
  qmgr -c "s h am_hook order=1000"
  ```
- For jobs that run on multiple vnodes, the hook uses the sum of the resources used. The hook does not support calculation of variable rates on multiple vnodes.

3.4 Setting Budget Configuration Attributes

Optionally, you can change the setting for the data_lifetime configuration attribute (it's not a parameter in the `am.conf` file; it's an attribute you set via `amgr update config`).

This attribute defines the maximum time allowed between updates to the value of a dynamic service unit. Budget checks the update time for each dynamic service unit, for each project by calculating the value of (now - last update time for this dynamic service unit for this project).

If the value is not updated within the specified amount of time, Budget logs a warning message in `/var/spool/am/<Budget server hostname>.log`, but jobs can continue to run.

Format: integer seconds

Default: 3600

- To set the value of data_lifetime:

  amgr update config -n SU_DYNAMIC -V '{"data_lifetime":<new value>}'

  For example:

  ```
  amgr update config -n SU_DYNAMIC -V '{"data_lifetime":2400}'
  ```

  See section 4.2.3.8, "Updating Configuration Attributes", on page 65.
- To see the value of data_lifetime:

  ```
  amgr ls config   -n data_lifetime
  ```

  See section 4.2.2.8, "Listing Budget Configuration Attributes", on page 58.

3.5 Configuring Budget for Peer Scheduling

In a peer scheduling setup, different PBS complexes are set up to automatically run each others' jobs to dynamically load-balance jobs across the complexes. Budget needs to be aware of all the PBS complexes in a peer scheduling environment.

To use Budget when running jobs in multiple complexes in a peer scheduling environment:

1. Configure the Budget hooks and a formula at all of the PBS complexes involved in peer scheduling, and add one cluster to represent each PBS complex involved in peer scheduling to Budget; see section 3.3, “Adding a PBS Complex and Setting its Billing Model”, on page 40.
2. Add the peer scheduling clusters to the project account or user account that will be running jobs, via `amgr update {user | project} -c <cluster>`; see section 4.2.3.3, “Updating Projects”, on page 61 and section 4.2.3.2, “Updating Users”, on page 60.

4 Budget Commands

4.1 Budget Commands

4.1.1 Command Path

To run Budget commands, export the path of the `am` binaries to the PATH environment variable by using the command:

```
export PATH=$PATH:opt/am/python/bin/
```

4.1.2 Using Budget Commands

All Budget commands are prefixed with "amgr ".

To see a list of Budget subcommands with a single-line description for each command:

```
amgr <enter>
```

To get usage information for a command or subcommand:

<*command*> *--help*

<*command*> <*subcommand*> *--help*

For example:

`amgr add --help` provides information on how to use the main `amgr add` command

`amgr add period --help` provides information on how to use the `period` subcommand.

If you enter a command without the required arguments, Budget will prompt you to enter them.

4.1.3 Tables of Budget Commands

Table 4-1: Budget Commands for Managing Elements

Function	Command	Element Subcommands	Required Privilege	Link
Adding elements	`amgr add`	`user, project, group, cluster, period, service-unit`	*admin*	Adding Elements
Listing elements	`amgr ls`	`user, project, group, cluster, period, service-unit, configuration, role`	*user* can list periods, clusters, service units, own account, and roles, and all groups Member of project can list that project *manager* can list all elements	Listing Elements
Updating elements	`amgr update`	`user, project, group, cluster, period, service-unit, dynamicvalue, configuration`	*admin*	Updating Elements
Removing elements	`amgr rm`	`user, project, group, cluster, period, service-unit`	*admin*	Removing Elements
Reporting elements	`amgr report`	`user, project, group, job, transaction`	*user* can get report on self and own jobs and transactions Project member can get report on that project *manager* or *investor* can get report on all groups and projects	Getting Reports on Elements
Applying limits to dynamic service units	`amgr limit`	`user, project`	Group *manager*	Applying Limits to Dynamic Service Units
Syncing formula to cluster	`amgr sync`	`cluster`	*admin*	Syncing Formula to PBS Complex

Table 4-2: Budget Transaction and Account Checking Commands

Function	Command	Element Subcommands	Required Privilege	Link
Depositing service units	`amgr deposit`	`user, project, group`	*investor* for deposit to group Group *manager* for deposit to user or project account	Depositing Service Units
Checking balance of service units	`amgr check-balance`	`user, project, group`	*user* can check balance for self Project member can check balance for that project *manager* or *investor* can check balance for all groups and projects	Checking Service Unit Balance
Withdrawing service units	`amgr withdraw`	`user, project, group`	*investor* to withdraw from group Group *manager* to withdraw from user or project account	Withdrawing Service Units
Transferring service units	`amgr transfer`	`user, project, group`	*admin*	Transferring Service Units
Prechecking service unit balance	`amgr precheck`	`user, project`	*user* can precheck own balance *admin* or *teller* to check other balances	Prechecking Service Unit Balance
Acquiring service units	`amgr acquire`	`user, project`	*admin* or *teller*	Acquiring Service Units
Reconciling service units	`amgr reconcile`	`job, project`	*admin* or *teller*	Reconciling Service Units
Refunding service units	`amgr refund`	`transaction`	*admin*	Refunding Service Units

4.2 Commands for Managing Budget Elements

You use these commands to add, remove, update, list, get reports on, apply limits to, and synchronize Budget elements.

4.2.1 Adding Elements

amgr add {user | project | group | cluster | period | serviceunit}

4.2.1.1 Required Privilege

You must be *admin* to run this command.

4.2.1.2 Adding a User

4.2.1.2.i Synopsis

amgr add user -n <username> -A <accounting policy> -c <PBS server> -r <role> [-h <group list>] [-a <active>]

4.2.1.2.ii Description

Adds specified user to Budget.

The specified user must already be a PBS complex user. Each user you add to Budget should have an entry in the password file, with a password set, and a home directory on the Linux system where Budget is installed.

When you add a user, you must assign a role, a cluster, and an accounting policy to that user.

4.2.1.2.iii Options

-n, --name <username>

String. Username to add.

The username and project name cannot be the same.

-A, --accounting-policy <accounting policy>

String. Specifies the accounting policy for the user account. Required.

Valid values: *begin_period* | *end_period* | *proportionate*

- A project with the begin_period accounting policy is charged in the period when the job begins.
- A project with the end_period accounting policy is charged in the period when the job ends.
- A project with the proportionate accounting policy is charged during the periods when the jobs were run. Each period is charged in proportion to the use in that period.

See section 1.6.4, “Accounting Policy”, on page 17.

-c, --clusters <PBS server>

String. Associates the user with the specified cluster. Required.

Use the `-c <PBS server>` option once for each cluster.

-r, --role <admin | investor | manager | teller | user>

String. Sets the role of the user. Required.

-h, --groups <group>

String. Associates the user account with specified groups.

Use the `-h <group>` option once for each group.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Activates or deactivates the user account.

Default: active

4.2.1.2.iv Command Examples

Example 4-1: Add a user named "joe", setting the accounting policy to "begin_period", associating joe with cluster1 and group group01, and giving joe the role of *user*:

```
amgr add user -n joe -A begin_period -c cluster1 -r user -h group01
```

Example 4-2: Add a user to Budget, then add them to group01 as a *manager*:

```
amgr add user -n user01 -A begin_period -c user1 -r manager -h group01
amgr update group -n group01 -M + user01
```

Example 4-3: Add user user1 to Budget and add them to groups group01 and engineering as an *admin*, giving them the begin_period accounting policy, and adding them to the testbed1 and testbed2 PBS complexes:

```
amgr add user -n user1 -A begin_period -h group1 -h engineering -c testbed1 -c testbed2 -r admin
```

4.2.1.3 Adding a Project

4.2.1.3.i Synopsis

amgr add project -n <project name> -A <accounting policy> [-S <start date>] [-E <end date>] -c <PBS server> [-u <user>] [-h <group>] [-a <active>] [-m <metadata>]

4.2.1.3.ii Description

Adds specified project to Budget.

4.2.1.3.iii Options

-n, --name <project name>

String. Name of project to add.

The project name cannot be the same as a username.

-A, --accounting-policy <accounting policy>

String. Specifies the accounting policy for the project account. Required.

Valid values: *begin_period* | *end_period* | *proportionate*

- A project with the begin_period accounting policy is charged in the period when the job begins.
- A project with the end_period accounting policy is charged in the period when the job ends.
- A project with the proportionate accounting policy is charged during the periods when the jobs were run. Each period is charged in proportion to the use in that period.

See section 1.6.4, “Accounting Policy”, on page 17.

-S, --start-date <start date>

Date. Start date of the project. Optional.

Format: YYYY-MM-DD

-E, --end-date <end date>

Date. End date of the project. Optional.

Format: YYYY-MM-DD

-c, --clusters <PBS server>

String. Associates the specified cluster with the project account.

Use the `-c <PBS server>` option once for each cluster. To add multiple clusters:

-c <cluster1> -c <cluster2> ... -c <clusterN>

-u, --users <username>

String. Associates the specified user with the project account.

Use the `-u <username>` option once for each user. To add multiple users:

-u <user1> -u <user2> ... -u <userN>

-h, --groups <group name>

String. Associates the specified group with the project.

Use the `-h <group name>` option once for each group. To add multiple groups:

-h <group1> -h <group2> ... -h <groupN>

-a, --active <True|TRUE|true|t|1|False|FALSE|false|f|0>

Boolean. Sets the project to active or inactive.

-m, --metadata <metadata>

String. Modifies metadata attribute for the project.

Format: comma-separated key-value pairs

Syntax:

<key1>:<value1>,<key2>:<value2>...<keyN>:<valueN>

Keys are undefined.

4.2.1.3.iv Example

Example 4-4: Add a project named proj1 and give it metadata consisting of type:weather and region:asia:

```
amgr add project -n proj1 -A begin_period -S 2019-01-02 -E 2019-28-02 -c cluster1 -m
    type:weather,region:asia
```

4.2.1.4 Adding a Group

4.2.1.4.i Synopsis

amgr add group -n <group name> [-I <investor>] [-M <manager>] [-a <active>]

4.2.1.4.ii Description

Define a group and add it to Budget.

4.2.1.4.iii Options

-n, --name <group name>

String. Group to add.

-I, --investors <investor username>

String. Associates this investor with the specified group.

This username must already exist in Budget and have the *investor* or *admin* role.

You can associate one investor per `-I` option, and you can associate multiple investors per command line. To associate multiple investors:

-I <investor1> -I <investor2> ... -I <investorN>

-M, --managers <manager username>

String. Associates this manager with the specified group.

This username must already exist in Budget and have the *manager*, *investor*, or *admin* role.

You can associate one manager per -M option, and you can associate multiple managers per command line. To associate multiple managers:

-M <manager1> -M <manager2> ... -M <managerN>

Group managers can transfer funds from the group to projects and user accounts.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Sets the group active or inactive.

Default: active

4.2.1.5 Adding a Cluster

4.2.1.5.i Synopsis

amgr add cluster -n <PBS server> [-f <billing formula filename>] [-a <active>]

4.2.1.5.ii Description

Adds a cluster data structure to represent the specified PBS complex.

You can specify the billing formula used for each cluster at the time you add the cluster, or later; although if the formula at the cluster does not match the formula at the complex, jobs cannot run at that complex. When you specify the billing formula, this updates the cluster data structure with the formula.

You can set the cluster to active or inactive.

You can add multiple PBS complexes, but only one per command line.

4.2.1.5.iii Options

-n, --name <PBS server>

String. Name of the PBS server for the PBS complex.

-f, --formula <formula filename>

Sets the formula file at the cluster. Make sure this is the same formula as the one at the complex.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Sets cluster active or inactive. When the cluster is active, it can run jobs.

Default: active

4.2.1.6 Adding a Period

4.2.1.6.i Synopsis

amgr add period -n <period name> -S <start date> -E <end date> [-p <name of parent period>]

4.2.1.6.ii Description

Adds the specified period.

Make sure that periods at the same level do not overlap. For example, if Quarter1 ends March 31st, make sure that Quarter2 does not begin sooner than April 1st.

If you want to create periods with a parent-child relationship, you must create the parent period first. You cannot add a parent to an existing child. For example, if you want Year as the parent and Quarter1, Quarter2, etc., as children, create Year first.

If you create child periods, make sure that they fit within the parent period.

Make sure that your period hierarchy is finalized BEFORE doing any transactions or running any jobs; you cannot update or remove periods once transactions have been performed or jobs have started.

See section 1.6.1, "Periods, Allocation Periods, Billing Periods", on page 11.

4.2.1.6.iii Options

-n, --name <period name>

String. Period to add.

-S, --start-date <start date>

Date. Start date of the period.

Format: YYYY-MM-DD

-E, --end-date <end date>

Date. End date of the period.

Format: YYYY-MM-DD

-p, --parent <name of parent period>

String. Specifies the parent period. Optional.

4.2.1.7 Adding a Service Unit

4.2.1.7.i Synopsis

amgr add serviceunit -n <service unit name> [-t <type>] [-a <active>] [-d <description>]

4.2.1.7.ii Description

Adds a service unit to Budget. This service unit is defined in the formula file.

Add a standard service unit to measure consumption of a resource internally tracked by PBS, for example CPU hours.

Add a dynamic service unit to define a quota for an external resource.

Default type is standard service unit (SU_STANDARD).

4.2.1.7.iii Options

-n, --name <service unit name>

String. Name of service unit to add. If this is a standard service unit, name must match name used in formula file.

Blank spaces are not allowed.

-t, --type {SU_STANDARD | SU_DYNAMIC}

String. Specifies type for this service unit.

Default: *SU_STANDARD*

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Sets service unit active or inactive.

-d, --description <service unit description>

String. Describes the service unit.

Can contain alphanumeric and any special characters except double quotes.

If the description contains anything except alphanumeric, enclose it in double quotes.

4.2.1.7.iv Command Examples

Example 4-5: Adding a standard service unit named cpu_hrs to be used for CPU hours:

```
amgr add serviceunit -n cpu_hrs -d "CPU hours"
```

Example 4-6: Adding a dynamic service unit named luster:

```
amgr add serviceunit -n luster -t SU_DYNAMIC
```

4.2.2 Listing Elements

amgr ls {user | period | project | cluster | service unit | role}

Prints out Budget elements. You can list only elements that have already been added to Budget via `amgr add`.

By default, this command lists only active elements. To list active elements:

```
amgr ls
```

To list inactive elements, use the `-a False` option. For example, to list all the clusters which are inactive:

```
amgr ls cluster -a False
```

Use the `-l` switch for `amgr ls` commands for more detailed information.

Use the `-j` switch for `amgr ls` commands to get the detailed information output in JSON format. This output can be used by other programs which can process JSON format.

4.2.2.1 Required Privilege

A *user* can list periods, clusters, service units, own account, own projects, own role, and all groups.

A member of a project can list that project.

A *manager* can list all elements.

4.2.2.2 Listing Users

4.2.2.2.i Synopsis

amgr ls user [-n <username] [-a <active>] [-l | -j]

4.2.2.2.ii Description

When used with no options, lists all the users that exist in Budget. When you specify a username, prints information about that user.

4.2.2.2.iii Options

-n, --name <username>

String. Specifies the name of the user.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Filters users by active status. You can list either active or inactive users, but not both.

Default: active

-l, --list-info

Display information in list format.

-j, --json-info

Display information in JSON format.

4.2.2.3 Listing Projects

4.2.2.3.i Synopsis

amgr ls project [-n <project name] [-a <active>] [-l | -j]

4.2.2.3.ii Description

When used with no options, lists all the projects that exist in Budget. When you specify a project name, prints information about that project.

4.2.2.3.iii Options

-n, --name <project name>

String. Specifies the name of the project.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Filters projects by active status. You can list either active or inactive projects, but not both.

Default: active

-l, --list-info

Display information in list format.

-j, --json-info

Display information in JSON format.

4.2.2.4 Listing Groups

4.2.2.4.i Synopsis

amgr ls group [-n <group name>] [-a <active>] [-l | -j]

4.2.2.4.ii Description

When used with no options, lists all the groups that exist in Budget. When you specify a group name, prints information about that group.

4.2.2.4.iii Options

-n, --name <group name>

String. Specifies the name of the group.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Filters groups by active status. You can list either active or inactive groups, but not both.

Default: active

-l, --list-info

Display information in list format.

-j, --json-info

Display information in JSON format.

4.2.2.5 Listing Clusters

4.2.2.5.i Synopsis

amgr ls cluster [-n <PBS server>] [-a <active>] [-f] [-l | -j]

4.2.2.5.ii Description

By default, lists all active clusters. You can specify whether you want to see active or inactive clusters. To display inactive clusters:

```
amgr ls cluster -a False
```

4.2.2.5.iii Options

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Filters clusters by active status. You can list either active or inactive clusters, but not both.

Default: active

-n, --name <PBS server>

String. Name of the cluster to list.

You can list one cluster per command line.

-f, --formula

Prints formula file to `/tmp`, and prints the path to that file on screen.

-l, --list-info

Display information in list format.

-j, --json-info

Display information in JSON format.

4.2.2.5.iv Examples

Example 4-7: To print the formula file for cluster1:

```
amgr ls cluster -n cluster1 -f
```

Output formula file: `/tmp/cluster1_formula.json`

4.2.2.6 Listing Periods

4.2.2.6.i Synopsis

amgr ls period [-n <period>] [-l | -j]

4.2.2.6.ii Description

Prints out information about periods.

By default, this lists all periods that exist in the hierarchy.

See section 1.6.1, “Periods, Allocation Periods, Billing Periods”, on page 11.

4.2.2.6.iii Options

-n, --name <period name>

String. Name of the period to list.

-l, --list-info
Display information in list format.

-j, --json-info
Display information in JSON format.

4.2.2.7 Listing Service Units

4.2.2.7.i Synopsis

amgr ls serviceunit [-n <service unit name] [-a <active>] [-l | -j]

4.2.2.7.ii Description

When used with no options, lists all the service units that exist in Budget. When you specify a service unit name, prints information about that service unit.

4.2.2.7.iii Options

-n, --name <service unit name>
String. Specifies the name of the service unit.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}
Boolean. Filters service units by active status. You can list either active or inactive service units, but not both.
Default: active

-l, --list-info
Display information in list format.

-j, --json-info
Display information in JSON format.

4.2.2.8 Listing Budget Configuration Attributes

4.2.2.8.i Synopsis

amgr ls config [-n <attribute name>] [-l | -j]

4.2.2.8.ii Description

Prints a list of the Budget configuration attributes. See section 3.4, “Setting Budget Configuration Attributes”, on page 45.

4.2.2.8.iii Options

-n, --name <attribute name>
String. Specifies the name of the configuration attribute.

-l, --list-info
Display information in list format.

-j, --json-info
Display information in JSON format.

4.2.2.8.iv Sample Output

Sample output for `amgr ls config`:

```
SU_DYNAMIC
configuration = {'data_lifetime': 3600}
id = 1
created_user_name = root
created_date = 2020-05-12 09:42:53.312722+05:30
last_updated_user_name = root
last_updated_date = 2020-05-14 15:55:54.339+05:30
```

4.2.2.9 Listing Roles

4.2.2.9.i Synopsis

amgr ls role [-n <role>] [-a <active>] [-l | -j]

4.2.2.9.ii Description

Prints whether the role exists.

Every user in Budget must have an assigned role. See section 1.3, “Roles”, on page 4.

Roles in Budget:

admin
: Can perform all operations.

investor
: Can deposit and withdraw service units to and from groups with which the investor is associated.

manager
: Can deposit and withdraw service units to and from projects and users that are associated with groups with which the manager is associated.

teller
: Special role for performing automated acquire and reconcile transactions on behalf of users.

user
: Assigned to one or more projects; can run jobs using user budget or project budget.

4.2.2.9.iii Options

-n, --name <role>
: String. Name of the role to list.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}
: Boolean. Filters roles by active status. You can list either active or inactive roles, but not both.

  Default: active

-l, --list-info
: Display information in list format. Information includes whether the role is active for each user with the role.

-j, --json-info
: Display information in JSON format.

4.2.2.9.iv Command Example

Example 4-8: To show information for users with *admin* role:

```
amgr ls role -n admin
```

Example 4-9: To show more information about users with *admin* role, we use the -l switch:

```
amgr ls role -n admin -l
```

Example 4-10: To show information about users with *admin* role in JSON format, we use the -j switch:

```
amgr ls role -n admin -j
```

4.2.3 Updating Elements

amgr update {user | period | project | group | cluster | service unit}

Updates the specified element.

You cannot change the name of an element once it is created.

4.2.3.1 Required Privilege

You must be *admin* to run this command.

4.2.3.2 Updating Users

4.2.3.2.i Synopsis

amgr update user -n <username> [-A <accounting policy>] [-s <start date>] [-E <end date>] [-c <cluster list>] [-r <roles>] [-h <group list>] [-a <active>]

4.2.3.2.ii Description

Update information for a user.

4.2.3.2.iii Options

-n, --name <username>

String. Name of user account to update.

-A, --accounting-policy <begin_period | end_period | proportionate>

Specifies the accounting policy for the user account. It can be begin_period, end-period, or proportionate

-S, --start-date <start date>

Start date for the user.

Format: YYYY-MM-DD

-E, --end-date <end date>

End date for the user.

Format: YYYY-MM-DD

-c, --clusters <cluster update>

String. Specifies clusters to link to or unlink from the user.

To link or unlink clusters, use the + or - operator followed by a comma-separated list of clusters. Spaces are not allowed. Use a separate `amgr update user –n <username> –c <cluster update>` command for each + or - operation.

For example, to link clusters:

```
amgr update user –n user1 –c + cluster1,cluster2,cluster3
```

Or to unlink clusters:

```
amgr update user –n user1 –c – cluster4,cluster5,cluster6
```

-r, --roles <role>

Sets the role of the user.

-h, --groups <group update>

String. Specifies groups to link to or unlink from the user.

To link or unlink groups, use the + or - operator followed by a comma-separated list of groups. Spaces are not allowed. Use a separate `amgr update user –n <username> –h <group update>` command for each + or - operation.

For example, to link groups:

```
amgr update user –n user1 –h + group1,group2,group3
```

Or to unlink groups:

```
amgr update user –n user1 –h – group4,group5,group6
```

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Set the project to active or inactive.

4.2.3.3 Updating Projects

4.2.3.3.i Synopsis

amgr update project -n <project name> [-A <accounting policy>] [-S <start date>] [-E <end date>] [-c <cluster update>] [-u <user update>] [-h <group update>] [-a <active>] [-m <metadata update>]

4.2.3.3.ii Description

Updates project information. You can set new values for existing elements, add new elements such as users, groups, or metadata, and you can remove elements. Use the + operator to add or update elements and the - operator to remove elements.

4.2.3.3.iii Options

-n, --name <project name>

String. Name of the project to update.

-A, --accounting-policy <accounting policy>

String. Specifies the accounting policy for the project.

Value is one of *begin_period*, *end_period,* or *proportionate*.

-S, --start-date <start date>

Start date of the project.

Format: YYYY-MM-DD

-E, --end-date <end date>

End date of the project.

Format: YYYY-MM-DD

-c, --clusters <cluster update>

String. Specifies clusters to link to or unlink from the project.

To link or unlink clusters, use the + or - operator followed by a comma-separated list of clusters. Spaces are not allowed. Use a separate `amgr update project -n <project name> -c <cluster update>` command for each + or - operation.

For example, to link clusters:

```
amgr update project -n MyProject -c + cluster1,cluster2,cluster3
```

Or to unlink clusters:

```
amgr update project -n MyProject -c - cluster4,cluster5,cluster6
```

-u, --users <user update>

String. Specifies users to link to or unlink from the project.

To link or unlink users, use the + or - operator followed by a comma-separated list of users. Spaces are not allowed. Use a separate `amgr update project -n <project name> -u <user update>` command for each + or - operation.

For example, to link users:

```
amgr update project -n MyProject -u + user1,user2,user3
```

Or to unlink users:

```
amgr update project -n MyProject -u - user4,user5,user6
```

-h, --groups <group update>

String. Specifies groups to link to or unlink from the project.

To link or unlink groups, use the + or - operator followed by a comma-separated list of groups. Spaces are not allowed. Use a separate `amgr update project -n <project name> -h <group update>` command for each + or - operation.

For example, to link groups:

```
amgr update project -n MyProject -h + group1,group2,group3
```

Or to unlink groups:

```
amgr update project -n MyProject -h - group4,group5,group6
```

-m, --metadata <metadata update>

Adds, updates, or removes metadata from the project.

To add metadata: + operator followed by a comma-separated list of key-value pairs. Spaces are not allowed. For example:

```
amgr update project -n MyProject -m + <key1>:<value>,<key2>:<value2>
```

To remove metadata: - operator followed by a comma-separated list of keys only. Spaces are not allowed. For example:

```
amgr update project -n MyProject -m - <key3>,<key4>
```

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Sets the project to active or inactive.

4.2.3.3.iv Command Example

Add a new metadata element named "currency", set it to "rupee", and update the existing data region to "america":

```
amgr update project -n proj1 -m + currency:rupee,region:america
```

Remove existing metadata elements "type" and "region":

```
amgr update project -n proj1 -m - type,region
```

Multiple operations in single command:

```
amgr update project -n proj1 -m + currency:dollar,"Lead Researcher":"Jane Smith" -m - type,region
```

4.2.3.4 Updating Groups

4.2.3.4.i Synopsis

amgr update group -n <group name> [-i <investors>] [-M <managers>] [-a <active>]

4.2.3.4.ii Description

Update information for a group.

4.2.3.4.iii Options

-n, --name <group name>

String. Name of the group to update.

-i, --investors <investor update>

String. Specifies investors to link to or unlink from the user.

To link or unlink investors, use the + or - operator followed by a comma-separated list of investors. Spaces are not allowed. Use a separate `amgr update group -n <group name> -i <investor update>` command for each + or - operation.

For example, to link investors:

```
amgr update group -n group1 -i + investor1,investor2,investor3
```

Or to unlink investors:

```
amgr update group -n group1 -i - investor4,investor5,investor6
```

-m, --managers <manager update>

String. Specifies managers to link to or unlink from the user.

To link or unlink managers, use the + or - operator followed by a comma-separated list of managers. Spaces are not allowed. Use a separate `amgr update group -n <group name> -m <manager update>` command for each + or - operation.

For example, to link managers:

```
amgr update group -n group1 -m + manager1,manager2,manager3
```

Or to unlink managers:

```
amgr update group -n group1 -m - manager4,manager5,manager6
```

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Sets the group to be active or inactive.

4.2.3.5 Updating Clusters

4.2.3.5.i Synopsis

amgr update cluster -n <PBS server> [-a <active>] [-f <formula filename>]

4.2.3.5.ii Description

Updates the cluster data structure representing a PBS complex with the specified formula file. Pushes the formula file to the PBS complex. Imports the formula file (the Budget hook configuration file) into both Budget hooks at the complex. Updates the database with the new formula for this cluster.

If the formulas are different in the PBS hook (the .CF file) and the cluster data structure, jobs cannot run.

4.2.3.5.iii Options

-n, --name <PBS server>

String. Name of the cluster to update.

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Sets the cluster active or inactive.

Default: active

-f, --formula <formula filename>

String. Updates the specified formula file at the specified cluster data structure.

4.2.3.6 Updating a Period

4.2.3.6.i Synopsis

amgr update period -n <period name> [-S <start date>] [-E <end date>]

4.2.3.6.ii Description

Updates the specified period.

You cannot update a period that has any associated jobs or transactions.

See section 1.6.1, “Periods, Allocation Periods, Billing Periods”, on page 11.

4.2.3.6.iii Options

-n, --name <period name>

String. Period to update.

-S, --start-date <start date>

Date. Start date of the period.

Format: YYYY-MM-DD

-E, --end-date <end date>

Date. End date of the period.

Format: YYYY-MM-DD

4.2.3.7 Updating a Service Unit

4.2.3.7.i Synopsis

amgr update serviceunit -n <service unit name> [-t <type>] [-a <active>] [-d <description>]

4.2.3.7.ii Description

Updates a service unit.

You can change the type of a service unit, but there are restrictions:

- You can change standard to dynamic only when no transactions have taken place
- You can change dynamic to standard only when no updates have been made to the external dynamicvalue

4.2.3.7.iii Options

-n, --name <service unit name>

String. Name of service unit to update.

Blank spaces are not allowed.

-t, --type {SU_STANDARD | SU_DYNAMIC}

String. Specifies type for this service unit.

Default: *SU_STANDARD*

-a, --active {True|TRUE|true|t|1|False|FALSE|false|f|0}

Boolean. Sets service unit active or inactive.

-d, --description <service unit description>

String. Describes the service unit.

Can contain alphanumeric and any special characters except double quotes.

If the description contains anything except alphanumeric, enclose it in double quotes.

4.2.3.8 Updating Configuration Attributes

4.2.3.8.i Synopsis

amgr update config -n <attribute name> -V <update string>

4.2.3.8.ii Description

Update Budget configuration attributes. See section 3.4, “Setting Budget Configuration Attributes”, on page 45.

Updates one attribute at a time.

4.2.3.8.iii Options

-n, --name <attribute name>

String. Name of the Budget configuration attribute to update.

-V, --config-value <update string>

JSON formatted string of configuration key-value pairs.

Format:

amgr update config -n SU_DYNAMIC -V '{"<attribute name>": <value>}'

Example:

```
amgr update config -n SU_DYNAMIC -V '{"data_lifetime": 1000}'
```

4.2.3.8.iv Examples

```
amgr update config -n SU_DYNAMIC -V '{"data_lifetime": 3610}'
```

4.2.3.9 Updating Dynamic Service Unit Usage

4.2.3.9.i Synopsis

amgr update dynamicvalues -v <update specification>

4.2.3.9.ii Description

Reports updated value of the specified dynamic service unit to Budget. Value is for usage by a project or a user.

At a given point in time, a dynamic service unit is a snapshot of the current usage of a particular external resource such as scratch. This usage is reported by an external script that you write. The external script is typically a `cron` job that calls `amgr update dynamicvalues`, which updates the value of the dynamic service unit.

Budget compares the reported value with the specified quotas, and allows jobs to run as long as the usage is below the quota.

4.2.3.9.iii Required Privilege

You must be *admin* to run this command.

4.2.3.9.iv Options

-v, --values <update specification>

String in JSON format.

Format to update usage for a user:

'{"<dynamic service unit>": {"<username>":{"total":<total>}}}'

Format to update total usage for a project:

'{"<dynamic service unit>": {"<project>":{"total":<total>}}}'

You must include the keyword "total".

4.2.3.9.v Examples

Example 4-11: Set user user1's consumption of storage to 8:

```
amgr update dynamicvalues -v '{"storage": {"user1": {"total":8}}}'
```

Example 4-12: Set project project1's consumption of storage to 30:

```
amgr update dynamicvalues -v '{"storage": {"project1": {"total":30}}}'
```

4.2.4 Removing Elements

amgr rm {user | project | group | cluster | period | serviceunit}

4.2.4.1 Required Privilege

You must be *admin* to run this command.

4.2.4.2 Removing a User

4.2.4.2.i Synopsis

amgr rm user -n <username>

4.2.4.2.ii Description

Removes specified user from Budget, unless the user has any past or current associated jobs or transactions, in which case the user is made inactive.

4.2.4.2.iii Options

-n, --name <username>

String. Name of the user to remove.

4.2.4.2.iv Command Examples

Example 4-13: Remove a user named "joe":

```
amgr rm user -n joe
```

4.2.4.3 Removing a Project

4.2.4.3.i Synopsis

amgr rm project -n <project name>

4.2.4.3.ii Description

Removes specified project from Budget, unless the project has any past or current associated jobs or transactions, in which case the project is made inactive.

4.2.4.3.iii Options

-n, --name <project name>

String. Name of the project to remove.

The project name cannot be the same as a username.

4.2.4.3.iv Example

Example 4-14: Remove a project named proj1:

```
amgr rm project -n proj1
```

4.2.4.4 Removing a Group

4.2.4.4.i Synopsis

amgr rm group -n <group name>

4.2.4.4.ii Description

Removes specified group from Budget, unless the group has any past or current associated transactions, in which case the group is made inactive.

4.2.4.4.iii Options

-n, --name <group name>

String. Name of the group to remove.

4.2.4.5 Removing a Cluster

4.2.4.5.i Synopsis

amgr rm cluster -n <PBS server>

4.2.4.5.ii Description

Removes the specified cluster from Budget, unless the cluster has any past or current associated jobs or transactions, in which case the cluster is made inactive.

4.2.4.5.iii Options

-n, --name <PBS server>

String. Name of the cluster to remove.

4.2.4.6 Removing a Period

4.2.4.6.i Synopsis

amgr rm period -n <period name>

4.2.4.6.ii Description

Removes specified period from Budget, if the period and all parent periods have no associated jobs or transactions. If either the period or a parent has any associated jobs or transactions, the period is made inactive.

See section 1.6.1, “Periods, Allocation Periods, Billing Periods”, on page 11.

4.2.4.6.iii Options

-n, --name <period name>

String. Name of the period to remove.

4.2.4.7 Removing a Service Unit

4.2.4.7.i Synopsis

amgr rm serviceunit -n <service unit name>

4.2.4.7.ii Description

Removes specified service unit from Budget, unless the service unit has any past or current associated transactions, in which case the service unit is made inactive.

4.2.4.7.iii Options

-n, --name <service unit name>

String. Name of service unit to remove.

If this is a standard service unit, the name must match name used in formula file.

Blank spaces are not allowed.

If this is a standard service unit, you must also remove this service unit from all formulas, otherwise jobs will not run.

4.2.4.7.iv Command Examples

Example 4-15: Removing a standard service unit named cpu_hrs:

```
amgr rm serviceunit -n cpu_hrs
```

Example 4-16: Removing a dynamic service unit named luster:

```
amgr rm serviceunit -n luster
```

4.2.5 Getting Reports on Elements

The `amgr report` command allows you to get reports for projects, groups, jobs, transactions, and users.

amgr report {user | project | group | job | transaction}

Use the `-l` option to the `amgr report` commands for more detailed information.

4.2.5.1 Required Privilege

A *user* can get a report on their own usage, jobs, and transactions.

A member of a project can get a report on that project.

admin, manager, or *investor* can get a report on any group and project.

4.2.5.2 Getting User Reports

4.2.5.2.i Synopsis

amgr report user -n <username> [-s <service unit name> | -t <service unit type>] [-h] [-p <period>] [-S <start date>] [-E <end date>] [-l] [-o <output file>] [-r] [-A]

4.2.5.2.ii Description

By default, this prints information for all standard service units for the current period that is lowest in the hierarchy (shortest time division). You can refine your output by specifying service unit name and type.

By default, this prints the report to the screen in human-readable tables. You can print to a file, and you can print the report in raw (CSV) format.

4.2.5.2.iii Options

-n, --name <username>

String. Username on which to get report.

-s, --serviceunit <service unit name>

Prints report for specified service unit.

If you do not specify a service unit, the report includes all service units of the specified type. If you do not specify type, it is *SU_STANDARD*.

-t, --sunit-type <service unit type>

Use this option to see dynamic service units. By default, this command prints only standard service units.

Cannot be used with `-l`, `-g`, or `-h` options.

Type can be one of *SU_STANDARD* or *SU_DYNAMIC*.

Default: *SU_STANDARD*

-g, --global-view

Lists project details and a summary of all transactions with a detailed listing of each transaction in JSON format.

Cannot be used with `-t SU_DYNAMIC`.

-h, --stakeholder-info

Lists groups which have invested in the user and their invested amounts.

Cannot be used with `-t SU_DYNAMIC`.

-p, --period <period name>

Specifies report period.

Cannot be used with `-S` and `-E` options.

Default: current period that is lowest in hierarchy (shortest time division)

-S, --start-date <start date>

Reports activity starting at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-E, --end-date <end date>

Reports activity ending at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-l, --list-info

Lists all transactions that have happened for the specified user.

Use the -l switch to get the report in long format, containing the transaction_id, transaction_date, transaction_time, entity, transaction_type, transaction_user, type amount and reconciled status in columns.

Cannot be used with `-t SU_DYNAMIC`.

-o, --out-file <output filename>

Budget prints the report in a human-readable form to the specified output file.

Without this option, Budget prints the report to the screen.

-r, --raw-output

Print report in raw (CSV) form with no padding. This form is easier to parse but harder for a human to read.

-A, --allocated-view

For listing investor information. Lists remaining credit allocated from this investor to each group in which the investor invested.

4.2.5.2.iv Output Format

Output for the report contains the Name, Service Unit, Period, Opening Balance, Total Credits, Total Debits, Total Debits (Reconciled), Total Debits (Authorized), and Net Balance.

Opening Balance

Opening balance of user account on specified start date

Total Credits

Sum of deposits; does not decrease from transfers or withdrawals or usage. Associated with a period.

Total Debits (Reconciled)

Total consumed amount for a period.

Total Debits (Authorized)

Amount held by running jobs plus amount held by unreconciled finished jobs, for a period.

Current balance

Opening Balance + Total Credits - Total Debits - Total Debits (Reconciled) - Total Debits (Authorized). For period.

4.2.5.2.v Command Examples

Example 4-17: User report in long format:

amgr report user -n <username> -s <service unit> -S <start date> -E <end date> -l

Example 4-18: To report user storage usage:

```
amgr report user -n user1 -t SU_DYNAMIC
-------------------------------------------------------------------------------------
name  | serviceunit | period  | limit | last_reported_time             | total_consumed
-------------------------------------------------------------------------------------
user1 | storage     | 2021.feb| 12.0  | 2021-02-11 18:58:09.48498+00:00| 8.0
```

Example 4-19: User report showing dynamic service units:

```
amgr report user -n pbsuser -t SU_DYNAMIC

---------------------------------------
name    | serviceunit | period | limit
---------------------------------------
pbsuser | luster      | 2020   | 600.0
pbsuser | scratch     | 2020   | 800.0

-----------------------------------------------------
| last_reported_time               | total_consumed
-----------------------------------------------------
| 2020-04-07 12:40:22.470078+05:30 | 40
| 2020-04-07 12:40:22.470078+05:30 | 60
```

4.2.5.3 Getting Project Reports

4.2.5.3.i Synopsis

amgr report project -n <project name> [-s <service unit> | -t <service unit type>] [-U] [-h] [-p <period>] [-S <start date>] [-E <end date>] [-l] [-o <output file>] [-r]

4.2.5.3.ii Description

Produces a project report.

By default, this prints information for all standard service units for the current period that is lowest in the hierarchy (shortest time division). You can refine your output by specifying service unit name and type.

By default, this prints the report to the screen in human-readable tables. You can print to a file, and you can print the report in raw (CSV) format.

4.2.5.3.iii Options

-n, --name <project name>

String. Name of project on which to get report.

-s, --serviceunit <service unit name>

Prints report for specified service unit.

If you do not specify a service unit, the report includes all service units of the specified type. If you do not specify type, it is *SU_STANDARD*.

-t, --sunit-type <service unit type>

Use this option to see dynamic service units. By default, this command prints only standard service units.

Cannot be used with `-l`, `-g`, or `-h` options.

Type can be one of *SU_STANDARD* or *SU_DYNAMIC*.

Default: *SU_STANDARD*

-U, --user-wise

Prints consumption, one line per consumer, per service unit. Output includes amount of credit, username, and period.

By default only standard service units are printed; to see dynamic service units, use the -t option.

-g, --global-view

Lists project details and a summary of all transactions with a detailed listing of each transaction in JSON format.

Cannot be used with `-t SU_DYNAMIC`.

-h, --stakeholder-info

Lists groups which have invested in the project and their invested amounts.

Cannot be used with `-t SU_DYNAMIC`.

-p, --period <period name>

Specifies report period.

Cannot be used with `-S` and `-E` options.

Default: current period that is lowest in hierarchy (shortest time division)

-S, --start-date <start date>

Reports activity starting at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-E, --end-date <end date>

Reports activity ending at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-l, --list-info

Lists all transactions that have happened for the specified project.

Use the `-l` option to get the report in long format, containing the transaction_id, transaction_date, transaction_time, entity, transaction_type, transaction_user, type amount and reconciled status in columns.

Cannot be used with `-t SU_DYNAMIC`.

-o, --out-file <output filename>

Budget prints the report in a human-readable form to the specified output file.

Without this option, Budget prints the report to the screen.

-r, --raw-output

Print report in raw (CSV) form with no padding. This form is easier to parse but harder for a human to read.

4.2.5.3.iv Output Format

The report is printed as a columns, for name, start_date, end_date, opening_balance, total_credits, total_debits_reconciled, total_debits_authorized, and net_balance.

4.2.5.3.v Command Example

Example 4-20: Report for all standard service units and current lowest period:

```
amgr report project -n p1
```

Command output:

```
------------------------------------------------------------------------
name | period   | serviceunit | opening_balance | total_credits
------------------------------------------------------------------------
p1   | DEC-2018 | dollar1     | 0.0             | 3000000.0

------------------------------------------------------------------------
| total_debits | total_debits_reconciled | total_debits_authorized
------------------------------------------------------------------------
| 0.0          | 180.0                   | 0.0

----------------
| net_balance
----------------
| 2999820.0
```

Example 4-21: Report for service unit dollar1 for period 2018:

```
amgr report project -n p1 -s dollar1 -p 2018
```

Command output:

```
------------------------------------------------------------------------
name | period | serviceunit | opening_balance | total_credits
------------------------------------------------------------------------
p1   | 2018   | dollar1     | 0.0             | 5000000.0

------------------------------------------------------------------------
| total_debits | total_debits_reconciled | total_debits_authorized
------------------------------------------------------------------------
| 0.0          | 0.0                     | 0.0

-----------------
| net_balance
-----------------
| 5000000.0
```

To get raw output for the project:

```
amgr report project -n p1 -s dollar1 -p 2018 -r
```

Command output:

```
name,period,service-
    unit,opening_balance,total_credits,total_debits,total_debits_reconciled,total_debits_authorized
    ,net_balance
p1,2018,dollar1,0.0,5000000.0,0.0,0.0,0.0,5000000.0
```

Example 4-22: Show individual transactions for the service unit dollar1 for the lowest period:

```
amgr report project -n p1 -s dollar1 -l
```

Command output:

```
-------------------------------------------------------------------------
transaction_id    | transaction_date | transaction_time | entity
-------------------------------------------------------------------------
1544096706.291656 | 2018-12-06       | 17:15:06.290064  | manager
2042.cluster1     | 2018-12-06       | 17:15:23.760451  | job
2043.cluster1     | 2018-12-06       | 17:15:24.695462  | job
2044.cluster1     | 2018-12-06       | 17:16:24.058123  | job

-------------------------------------------------------------------------------
| transaction_type | transaction_user | type   | serviceunit | amount
-------------------------------------------------------------------------------
| grant            | Manager1         | credit | dollar1     | 3000000.0
| acquired         | amteller         | debit  | dollar1     | 60.0
| acquired         | amteller         | debit  | dollar1     | 60.0
| acquired         | amteller         | debit  | dollar1     | 60.0

-------------------------------------------------------------
| reconciled | period   | comment
-------------------------------------------------------------
| yes        | DEC-2018 | Deposit dollar1 to DEC period
| yes        | DEC-2018 |
| yes        | DEC-2018 |
| yes        | DEC-2018 |
```

Example 4-23: Print individual transactions for the service unit dollar1 for a date range:

```
amgr report project -n p1 -s dollar1 -l -S '2018-12-06' -E '2018-12-06'
```

Command output:

```
-------------------------------------------------------------------------
transaction_id    | transaction_date | transaction_time | entity
-------------------------------------------------------------------------
1544096706.100772 | 2018-12-06       | 17:15:06.098180  | manager
1544096706.291656 | 2018-12-06       | 17:15:06.290064  | manager
2042.cluster1     | 2018-12-06       | 17:15:23.760451  | job
2043.cluster1     | 2018-12-06       | 17:15:24.695462  | job
2044.cluster1     | 2018-12-06       | 17:16:24.058123  | job

-------------------------------------------------------------------------------
| transaction_type | transaction_user | type   | serviceunit | amount
-------------------------------------------------------------------------------
| grant            | Manager1         | credit | dollar1     | 5000000.0
| grant            | Manager1         | credit | dollar1     | 3000000.0
| acquired         | amteller         | debit  | dollar1     | 60.0
| acquired         | amteller         | debit  | dollar1     | 60.0
| acquired         | amteller         | debit  | dollar1     | 60.0

-------------------------------------------------------------
| reconciled | period   | comment
-------------------------------------------------------------
| yes        | 2018     | Deposit dollar1 to parent period
| yes        | DEC-2018 | Deposit dollar1 to DEC period
| yes        | DEC-2018 |
| yes        | DEC-2018 |
| yes        | DEC-2018 |
```

4.2.5.3.vi Project Reports Showing Dynamic Service Units

Example 4-24: Project report in short format:

```
amgr report project -n p1 -t SU_DYNAMIC
-------------------------------------------------------------------------------------
name | serviceunit | period | limit |last_reported_time               | total_consumed
-------------------------------------------------------------------------------------
p1   | luster      | 2020   | 500.0 |2020-04-07 12:40:22.470078+05:30 | 80
p1   | scratch     | 2020   | 800.0 |2020-04-07 12:40:22.470078+05:30 | 100
```

Example 4-25: Project report in user-wise format:

```
amgr report project -n p1 -t SU_DYNAMIC -U


-------------------------------------------------------------------------------------
name | serviceunit | period | limit |last_reported_time | total_consumed | user_consumed
-------------------------------------------------------------------------------------
p1   | luster    | 2020   | 500.0 | 2020-04-07 12:40  |  80  | {"u1": 20.0, "u2": 10.0}
p1   | scratch   | 2020   | 800.0 | 2020-04-07 12:40  | 100  | {"u1": 40.0}
```

Example 4-26: Report project storage data:

```
amgr report project -n proj1 -t SU_DYNAMIC
```

Command output:

```
-------------------------------------------------------------------------------------
name  | serviceunit | period  | limit | last_reported_time              | total_consumed
-------------------------------------------------------------------------------------
proj1 | storage     | 2021.feb| 25.0  | 2021-02-11 18:58:28.270385+00:00 | 30.0
```

4.2.5.4 Getting Group Reports

4.2.5.4.i Synopsis

amgr report group -n <group name> [-h] [-A] [-S <start-date>] [-E <end-date>] [-l] [-o <output-file>] [-r] [-p <period>]

4.2.5.4.ii Description

Produces group report showing deposits and withdrawals.

By default, this prints information for all standard service units for the current period that is lowest in the hierarchy (shortest time division). You can refine your output by specifying service unit name.

By default, this prints the report to the screen in human-readable tables. You can print to a file, and you can print the file in raw (CSV) format.

4.2.5.4.iii Options

-n, --name <group name>

String. Name of the group on which to get report.

-h, --stakeholder-info

For each investor in the group, lists remaining balance for that investor, of what they invested.

-A, --allocated-view

Lists remaining credit allocated from this group to users and projects.

Shows for each project, how much this project has remaining of what the group invested.

Shows for each user, how much this user has remaining of what the group invested.

Cannot filter by project or user.

-S, --start-date <start date>

Reports activity starting at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-E, --end-date <end date>

Reports activity ending at this date. Report includes created and updated transactions.

Requires `-l` or `-g` option.

Cannot be used with `-p` option.

Format: YYYY-MM-DD HH:MM:SS

-l, --list-info

Lists all transactions that have happened for the specified group.

Use the `-l` option to get the report in long format, containing the transaction_id, transaction_date, transaction_time, entity, transaction_type, transaction_user, type amount and reconciled status in columns.

-o, --out-file <output filename>

Budget prints the report in a human-readable form to the specified output file.

Without this option, Budget prints the report to the screen.

-r, --raw-output

Print report in raw (CSV) form with no padding. This form is easier to parse but harder for a human to read.

-p, --period-name <period name>

Specifies report period.

Cannot be used with `-s` and `-E` options.

Default: current period that is lowest in hierarchy (shortest time division)

4.2.5.4.iv Group Report Examples

Example 4-27: Default group report:

```
amgr report group -n h1
```

Command output:

```
-------------------------------------------------------------------
name | serviceunit | opening_balance | total_credits | total_debits
-------------------------------------------------------------------
h1   | cpu_hrs     | 0.0             | 100000.0      | 100.0

-------------------------------------------------------------------
| total_allocated | total_accounts_released | net_balance
-------------------------------------------------------------------
| 600.0           | 70.0                    | 99370.0
```

Example 4-28: Long format group report:

```
amgr report group -n h1 -l
```

Command output:

```
-------------------------------------------------------------------------------------
transaction_id     | transaction_date | transaction_time | entity   | transaction_type
-------------------------------------------------------------------------------------
1570782001.2559264 | 2019-10-11       | 13:50:01.253271  | investor | grant
2470123401.5592132 | 2019-10-11       | 13:50:11.967594  | manager  | grant
6470127408.2212197 | 2019-10-11       | 13:50:31.161543  | manager  | grant

-------------------------------------------------------------------------------
transaction_user | type   | serviceunit | amount   | balance  | period | account
-------------------------------------------------------------------------------
root             | credit | cpu_hrs     | 500000.0 | 500000.0 | -      | group1
mgr1             | credit | cpu_hrs     | 1000.0   | 499996.0 | 2019   | Project1
mgr2             | credit | cpu_hrs     | 4.0      | 499992.0 | 2019   | user1
```

Example 4-29: Report in investor format:

```
amgr report group -n h1 -h
```

Command output:

```
-----------------------------------------------------
investor | serviceunit | balance
-----------------------------------------------------
root     | cpu_hrs     | 50000.0
rsv      | cpu_hrs     | 50000.0
```

Example 4-30: Report for primary group accounts:

```
amgr report user -n h1 -h
```

Command output:

```
-------------------------------------------------------------
group | period | serviceunit | balance
-------------------------------------------------------------
h1    | 2019   | cpu_hrs     | 1500.0
h2    | 2019   | cpu_hrs     | 1500.0
```

Example 4-31: Group report in investor format:

This report shows where managers have deposited the group's service units.

```
amgr report group -n h1 -A
```

Command output:

```
------------------------------------------------------------
account | period | serviceunit | balance
------------------------------------------------------------
root    | 2019   | cpu_hrs     | 498976.0
rsv     | 2019   | cpu_hrs     | 498975.0
```

4.2.5.5 Getting Job and Transaction Reports

4.2.5.5.i Synopsis

amgr report transaction [-i <job ID or transaction ID>] [-l] [-N <count>] [-o <output-file>] [-r]

4.2.5.5.ii Description

Produces a report for a job or transaction.

The default report for a job ID is the net cost for the job.

The default report for a transaction ID is a summary of that transaction.

By default, this prints the report to the screen in human-readable tables. You can print to a file, and you can print the report in raw (CSV) format.

4.2.5.5.iii Options

-i, --transaction-id <transaction ID>

String. ID of Job or transaction on which to get report.

-l, --list-info

Lists all transactions that have happened for the specified job or transaction ID.

Use the `-l` option to get the report in long format, containing the transaction_id, transaction_date, transaction_time, entity, transaction_type, transaction_user, type amount and reconciled status in columns.

-N, --non-reconcile <count>

For use without job or transaction ID. Show all non-reconciled jobs that have had *count* or more attempts to reconcile.

For example, `amgr report transaction -N 2` displays all non-reconciled jobs with count >=2.

Budget attempts to reconcile a job three times; if this doesn't work, Budget marks the job as non-reconciled.

-o, --out-file <output filename>

Prints the report in a human-readable form to the specified output file.

Without this option, Budget prints the report to the screen.

-r, --raw-output

Print report in raw (CSV) form with no padding. This form is easier to parse but harder for a human to read.

4.2.5.5.iv Output Format

Output for the report lists columns for transaction_id, project, user_name, and reconciled_service_units.

If a job is not reconciled, it appears only when you use the `-l` option for a long format report.

Use the `-l` switch to get the report for the job or transaction in long format, which lists transaction_id, transaction_date, transaction_time, project, transaction_type, credit or debit, service units, and amounts.

4.2.5.5.v Examples

Example 4-32: Print job report:

```
amgr report transaction -i 2044.cluster1
```

Command output:

```
-------------------------------------------------------------------------------
transaction    | account | user     | reconciled_dollar1 | reconciled_dollar2
-------------------------------------------------------------------------------
2044.cluster1 | p1       | pbsuser | 60.0               | 21120.0
```

Example 4-33: Print long format report for job, showing individual operations:

```
amgr report transaction -i 2044.cluster1 -l
```

Command output:

```
------------------------------------------------------------------------
transaction_id  | transaction_date | transaction_time | account
------------------------------------------------------------------------
2044.cluster1   | 2018-12-06       | 17:17:46.655998  | p1
2044.cluster1   | 2018-12-06       | 17:16:24.058123  | p1
2044.cluster1   | 2018-12-06       | 17:16:24.058123  | p1

-----------------------------------------------
| transaction_type | type  | service_unit
-----------------------------------------------
| acquired         | debit | dollar2
| acquired         | debit | dollar2
| acquired         | debit | dollar1

-----------------------------------
| amount  | period   | comment
-----------------------------------
| 19920.0 | DEC-2018 | overrun:
| 1200.0  | DEC-2018 |
| 60.0    | DEC-2018 |
```

4.2.6 Applying Limits to Dynamic Service Units

amgr limit {user | project}

Apply limits on dynamic service units for projects or users.

4.2.6.1 Synopsis

amgr limit user -n <username> -p <period name> -s <service unit name> <limit value>

amgr limit project -n <project name> -p <period name> -s <service unit name> <limit value>

4.2.6.2 Description

You establish a quota for an external resource by applying a limit to the dynamic service unit representing that resource. Usage of the external resource is reported by an external application. Jobs that use this resource can start only while the job owner has not reached the quota for this period.

Standard service units don't need a limit.

A limit on a dynamic service unit applies to a particular period and for a particular project or user account.

You can apply limits to any period.

All active dynamic service units must have limits set for the current period for jobs to run.

4.2.6.3 Effect of Limits on the Period Hierarchy

- If a parent period has a limit set for a particular dynamic service unit, its children inherit that limit unless the limit is explicitly set by the group manager.
- If you apply a limit to a period, that value overrides any inherited value.
- The default limit is zero unless the period in question inherits a parent's value.

4.2.6.3.i Rules for LImits on Dynamic Service Units

If a dynamic service unit has no limit set on it, the limit is zero. If there are active dynamic service units, all jobs are checked against those quotas, and a zero quota will stop any job from running. Make sure that you don't unintentionally stop non-target users or projects from running jobs:

- Make sure you set the quota for all users and projects
- When you specify the period, make sure you either:
  - Set the desired quota at the top level of the period hierarchy
  - Set a very high quota at the top level of the period hierarchy, and a more restrictive quota for the period you need to control

4.2.6.4 Required Privilege

You must be a manager associated with the group funding the user or project to apply limits to a dynamic service unit for that user or project.

4.2.6.5 Options

-p, --period <period name>

String. Period to which the limit is to be applied.

-s, --serviceunit <service unit name>

String. Service unit to limit.

<limit value>

Float. Budget reads this as being in the same units as the dynamic service unit.

Default: If a limit is unset, it is zero.

4.2.6.6 Examples

Example 4-34: Set a limit for the project MyProject, for the period named MyQuarter, on the service unit luster, to be 100:

```
amgr limit project -n MyProject -p MyQuarter -s luster 100
```

Example 4-35: Set a quota on storage of 12.0 for user user1 for the period of 2021:

```
amgr limit user -n user1 -s storage 12.0 -p 2021
```

Example 4-36: Set a quota of 25 for storage for the project project1 for the period of 2021:

```
amgr limit project -n project1 -s storage 25.0 -p 2021
```

4.2.7 Syncing Formula to PBS Complex

4.2.7.1 Synopsis

amgr sync formula -c <cluster>

4.2.7.2 Description

Pulls formula from the cluster data structure representing the specified PBS complex, pushes it to the specified complex, and imports it into the am_hook and am_hook_periodic hooks at the specified complex.

4.2.7.3 Required Privilege

You must be *admin* to run this command.

4.2.7.4 Options

-c, --cluster <target PBS complex>

String. Name of PBS complex where formula is to be updated.

4.3 Transaction and Account Checking Commands

The Budget service units commands and their respective subcommands are listed here:

Table 4-3: Budget Service Units Commands

Function	Command	Element Subcommands	Required Privilege	Link
Depositing service units	`amgr deposit`	`user, project, group`	*investor* for deposit to group Group *manager* for deposit to user or project account	Depositing Service Units
Checking balance of service units	`amgr check-balance`	`user, project, group`	*user* can check own balance	Checking Service Unit Balance
Withdrawing service units	`amgr withdraw`	`user, project, group`	*investor* to withdraw from group Group *manager* to withdraw from user or project account	Withdrawing Service Units
Transferring service units	`amgr transfer`	`user, project, group`	*admin*	Transferring Service Units
Prechecking service unit balance	`amgr precheck`	`user, project`	*user* can precheck own balance *admin* or *teller* to check other balances	Prechecking Service Unit Balance
Acquiring service units	`amgr acquire`	`user, project`	*admin* or *teller*	Acquiring Service Units
Reconciling service units	`amgr reconcile`	`job, project`	*admin* or *teller*	Reconciling Service Units
Refunding service units	`amgr refund`	`transaction`	*admin*	Refunding Service Units

4.3.1 Depositing Service Units

amgr deposit {user | project | group}

4.3.1.1 Deposit Service Units to User

4.3.1.1.i Synopsis

amgr deposit user -n <username> -s <service unit name> <service unit amount> -p <period> -h <group> [-C <comment>]

4.3.1.1.ii Description

Deposits service units to a user account.

4.3.1.1.iii Required Privilege

You must be *manager* to run this command.

4.3.1.1.iv Options

-n, --name <username>

String. User account to receive service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Specifies the service unit to be deposited and the quantity to deposit.

-p, --period <period>

String. Period during which service units are available.

-h, --group <group name>

String. Name of the group from which service units are to be allocated.

-C, --comment <comment>

String. Comment or reason for the deposit. Optional.

Default: no comment

4.3.1.2 Depositing Service Units to Project

4.3.1.2.i Synopsis

amgr deposit project -n <project name> -s <service unit name> <service unit amount> -p <period> -h <group> [-C <comment>]

4.3.1.2.ii Description

Deposits service units to a project.

4.3.1.2.iii Required Privilege

You must be *manager* to run this command.

4.3.1.2.iv Options

-n, --name <project name>

String. Name of project to receive service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Specifies the service unit to be deposited and the quantity to deposit.

-p, --period <period>

String. Period during which service units are available.

-h, --group <group name>

String. Name of the group from which service units are to be allocated.

-C, --comment <comment>

String. Comment or reason for the deposit. Optional.

Default: no comment

4.3.1.3 Depositing Service Units to Group

4.3.1.3.i Synopsis

amgr deposit group -n <group name> -s <service unit name> <service unit amount> [-C <comment>]

4.3.1.3.ii Description

Deposits investor service units to a group. This command is run by the investor; the funds invested come from the investor running the command.

4.3.1.3.iii Required Privilege

This command must be run by an *investor*.

4.3.1.3.iv Options

-n, --name <group name>

String. Name of group to receive service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Specifies the service unit to be deposited and the quantity to deposit.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-C, --comment <comment>

String. Comment or reason for the deposit. Optional.

Default: no comment

4.3.2 Checking Service Unit Balance

amgr checkbalance {user | project | group}

4.3.2.1 Required Privilege

A *user* can check their own service unit balance.

A member of a project can check the service unit balance for that project.

An *admin, manager,* or *investor* can check the service unit balance for all groups and projects.

4.3.2.2 Output Format

- For `amgr checkbalance {user | project}`, output is a list of standard service units in name-value pairs and dynamic service units in name-limit-value triplets, in the order in which the service units were created:

```
{"<standard unit 1>": <amount available>,
 "<standard unit 2>": <amount available>,
 "<dynamic unit 1>": {"used": <amount used>, "limit": <limit value>},
 "<dynamic unit 2>": {"used": <amount used>, "limit": <limit value>} }
```

- For `amgr checkbalance group`, output is a list of standard service units in name-value pairs, in the order in which the service units were created:

```
{"<standard unit 1>": <amount available>,
 "<standard unit 2>": <amount available>,}
```

4.3.2.3 Command Example

Example 4-37: For output from `amgr checkbalance {user | project}`, dynamic service unit luster, and standard service units cpu_hrs and gpu_hrs:

```
{"luster": {"used": 0.0, "limit": 300}, "cpu_hrs":0.0, "gpu_hrs":0.0}
```

Example 4-38: For output from `amgr checkbalance group`, standard service units cpu_hrs and gpu_hrs:

```
{"cpu_hrs":0.0, "gpu_hrs":0.0}
```

4.3.2.4 Checking Service Unit Balance for User

4.3.2.4.i Synopsis

amgr checkbalance user -n <username> [-p <period>]

4.3.2.4.ii Description

Check available service unit balance for a user account.

4.3.2.4.iii Options

-n, --name <username>

String. User account to check for available service units.

-p, --period <period>

String. Period during which service units are required.

Default: period is the current period that is lowest in the hierarchy, meaning shortest time span.

4.3.2.5 Checking Service Unit Balance for Project

4.3.2.5.i Synopsis

amgr checkbalance project -n <project name> -p <period>

4.3.2.5.ii Description

Checks available service unit balance for a project.

4.3.2.5.iii Options

-n, --name <project name>

String. Name of project to check for available service units.

-p, --period <period>

String. Period during which service units are required.

4.3.2.6 Checking Service Unit Balance for Group

4.3.2.6.i Synopsis

amgr checkbalance group -n <group name>

4.3.2.6.ii Description

Checks available service unit balance for a group.

4.3.2.6.iii Options

-n, --name <group name>

String. Name of group to check for available service units.

4.3.3 Withdrawing Service Units

amgr withdraw {user | project | group}

4.3.3.1 Withdrawing Service Units from User

4.3.3.1.i Synopsis

amgr withdraw user -n <username> -s <service unit type> <service unit amount> <-p <period> -h <group> [-C <comment>]

4.3.3.1.ii Description

Withdraw service units from a user account. You can withdraw only up to the amount you have deposited.

4.3.3.1.iii Required Privilege

You must be *manager* to run this command.

4.3.3.1.iv Options

-n, --name <username>

String. User account from which to withdraw service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Name of service unit to be withdrawn and the quantity to withdraw.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-p, --period <period>

String. Period from which to withdraw service units.

-h, --group <group name>

String. Group to receive withdrawn service units.

-C, --comment <comment>

String. Comment or reason for the withdrawal. Optional.

Default: no comment

4.3.3.2 Withdrawing Service Units from Project

4.3.3.2.i Synopsis

amgr withdraw project -n <project name> -s <service unit type> <service unit amount> <-p <period> -h <group> [-C <comment>]

4.3.3.2.ii Description

Withdraws service units from a project.

A manager can withdraw only up to the amount they deposited.

4.3.3.2.iii Required Privilege

You must be *manager* to run this command.

4.3.3.2.iv Options

-n, --name <project name>

String. Name of project from which to withdraw service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Name of service unit to be withdrawn and the quantity to withdraw.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-p, --period <period>

String. Period from which to withdraw service units.

-h, --group <group name>

String. Group to receive withdrawn service units.

-C, --comment <comment>

String. Comment or reason for the withdrawal. Optional.

Default: no comment

4.3.3.3 Withdrawing Service Units from Group

4.3.3.3.i Synopsis

amgr withdraw group -n <group name> -s <service unit type> <service unit amount> [-C <comment>]

4.3.3.3.ii Description

Withdraw service units from a group. The investor can remove only the amount of service units that the investor deposited.

4.3.3.3.iii Required Privilege

You must be *investor* to run this command.

4.3.3.3.iv Options

-n, --name <group name>

String. Group from which to withdraw service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Name of service unit to be withdrawn and the quantity to withdraw.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-C, --comment <comment>

String. Comment or reason for the withdrawal. Optional.

Default: no comment

4.3.4 Prechecking Service Unit Balance

amgr precheck {user | project}

4.3.4.1 Prechecking a User or Project

4.3.4.1.i Synopsis

amgr precheck user -n <user name> -c <cluster> [-D <transaction date>] -d <duration> -s <service unit name> <service unit amount>

amgr precheck project -n <project name> -c <cluster> [-D <transaction date>] -d <duration> -s <service unit name> <service unit amount>

4.3.4.1.ii Description

This command is intended to be used for debugging. You do not need to run it in the normal course of events.

Checks whether the project or user has sufficient funds and the specified complex is available to run this job at a specified date for a specific amount of service units. If the check fails, the job is not queued.

4.3.4.1.iii Required Privilege

A *user* can precheck their own account.

You must be *admin* or or *teller* to run this command for any other account.

4.3.4.1.iv Options

-n, --name <project name>

String. Name of the project or user to check for available service units.

-c, --cluster <PBS server>

String. PBS complex to check for availability.

-D, --transaction-date <transaction date>

Date. Transaction date and time.

Format: YYYY-MM-DD HH:MM:SS

-d, --duration <duration>

Integer seconds.

Job duration, in seconds.

-s, --serviceunits <service unit name> <service unit value>

String and float. Name of service unit and required amount.

4.3.5 Acquiring Service Units

amgr acquire {user | project}

4.3.5.1 Description

This command is intended to be used for debugging. You do not need to run it in the normal course of events. The Budget hook performs this operation for normal job processing.

Fetches the service units that are required to run a job from the user or project.

The teller uses this command to acquire service units for a user by taking them from a project or user account.

Run the checkbalance command before running a job to make sure there are enough service units available.

4.3.5.2 Required Privilege

You must be an *admin* or *teller* to run this command.

4.3.5.3 Acquiring Service Units for User

4.3.5.3.i Synopsis

amgr acquire user -n <username> -c <cluster> -s <service unit name> <service unit amount> [-D <transaction date>] -i <job ID> -d <duration> -u <username> [-C <comment>] -R <run count> -f <formula file>

4.3.5.3.ii Options

-n, --name <username>

String. User account from which to fetch service units.

-c, --cluster <PBS server>

String. Name of cluster.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Name of service unit and the quantity required.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-D, --transaction-date <date>

Date. Transaction date and time.

Format: YYYY-MM-DD HH:MM:SS

-i, --transaction-id <job ID>

Full PBS job ID.

-d, --duration <duration>

Duration. Job duration, in seconds.

-u, --user <username>

String. User account for whom to acquire service units.

-C, --comment <comment>

String. Comment or reason for the acquisition. Optional.

Default: no comment

-R, --run-count <run count>

Integer. Run count for the job.

-f, --formula <formula filename>

String. Formula file for cluster.

4.3.5.4 Acquiring Service Units for Project

4.3.5.4.i Synopsis

amgr acquire project -n <project name> -c <cluster> -s <service unit name> <service unit amount> -D <transaction date> -i <job ID > -d <duration> -u <username> [-C <comment>] -R <run count> -f <formula file>

4.3.5.4.ii Options

-n, --name <project name>

String. Project from which to fetch service units.

-c, --cluster <PBS server>
String. Name of cluster.

-s, --serviceunits <service unit name> <service unit amount>
String and float. Name of service unit and the quantity required.
Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-D, --transaction-date <date>
Date. Transaction date and time.
Format: YYYY-MM-DD HH:MM:SS

-i, --transaction-id <job ID>
Full PBS job ID.

-d, --duration <duration>
Duration. Job duration, in seconds.

-u, --user <username>
String. User account for whom to acquire service units.

-C, --comment <comment>
String. Comment or reason for the acquisition. Optional.
Default: no comment

-R, --run-count <run count>
Integer. Run count for the job.

-f, --formula <formula filename>
String. Formula file for cluster.

4.3.6 Reconciling Service Units

amgr reconcile {job | project}

Reconciles service units charged and consumed for jobs. Removes consumed service units from escrow, and returns unused service units to the account, or if the job consumed more than requested it debits more from the job owner's account.

To reconcile a batch of jobs after you've cleaned up a problem so that those jobs just need another attempt, use `amgr reconcile job`.

To reconcile a specific job that cannot be cleaned up, use `amgr reconcile project`.

This command is mostly intended to be used for cleanup and debugging. You do not need to run it in the normal course of events. The Budget hook performs all reconcile operations during normal job processing.

4.3.6.1 Required Privilege

You must be root and *admin* or *teller* to run this command. You must run this command at the PBS server host.

4.3.6.2 Reconciling Service Units for Project

4.3.6.2.i Synopsis

amgr reconcile project -n <project name> -c <cluster> -s <service unit name> <service unit amount> [-D <transaction date>] -i <job ID> -d <duration> [-C <comment>] -f <formula file>

4.3.6.2.ii Description

Reconciles service units for job charged to a project account.

4.3.6.2.iii Options

-n, --name <project name>

String. Project for which to reconcile service units.

-c, --cluster <PBS server>

String. Name of cluster.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Name of service unit and the quantity actually consumed.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-D, --transaction-date <date>

Date. Transaction date and time.

Format: YYYY-MM-DD HH:MM:SS

-i, --transaction-id <job ID>

Job ID.

-d, --duration <duration>

Integer. Job duration, in seconds.

-u, --user <username>

String. User account for which service units are to be reconciled.

-C, --comment <comment>

Comment or reason for the reconciliation. Optional.

Default: no comment

-f, --formula <formula filename>

String. Formula file for cluster.

4.3.6.3 Reconciling Service Units for Job

4.3.6.3.i Synopsis

amgr reconcile job -i <transaction ID list> --all -u <teller username> -f <formula filename>

4.3.6.3.ii Description

If you get a message asking you to log in and execute the command, you can do so:

```
sudo -u <teller username> <AM_EXEC>/bin/amgr sshlogin
```

4.3.6.3.iii Options

-i, --transaction-ids <job ID list>

Jobs to reconcile.

Format: String of comma-separated job IDs.

Example:

```
<jobID1>,<jobID2>,<jobID3>
```

Cannot be used with `--all` option.

--all

Reconcile all non-reconciled jobs.

Optional. Cannot be used with `-i` option.

-u, --user <teller username>

Username for teller. Required.

-f, --formula <formula filename>

String. Formula file for cluster.

4.3.7 Refunding Service Units

amgr refund transaction

4.3.7.1 Description

Refunds some or all of the funds charged for a job to the user or project that funded the job.

Budget knows who paid for the job, so you do not have to specify where the refund goes.

You can choose to provide a refund for situations such as when a job fails or runs multiple times, for reasons which cannot be attributed to the job owner. You can provide refunds to active and inactive projects and users. You can provide multiple refunds for the same job, but the total refund cannot exceed the amount consumed by the job.

The refund amount is calculated by multiplying the net job cost by the specified refund percentage by the total consumed amount. Total consumed amount is the sum of all transaction amounts of all transactions for a job.

Each group is refunded according to the percentage investment by that group. You can override this using the `-h` option.

4.3.7.2 Required Privilege

You must be *admin* to provide a refund.

4.3.7.3 Synopsis

amgr refund transaction -i <job ID> -r <refund percentage> -p <period> [-h <group>] [-C <comment>]

4.3.7.4 Options

-i, --transaction-id <job ID>

Job ID. Job for which to give a refund.

-r, --refund-percentage <refund percentage>

Integer between 1 and 100. Percentage of current net charge to refund.

-p, --period <period>

String. Period to refund.

Default: current period

-h, --group <group name>

String. Name of group which receives the refund. Overrides normal policy of refunding according to the percentage investment by each group. For use when you get an error message saying that Budget cannot associate the refund to a group.

-C, --comment <comment>

String. Comment or reason for refund. Optional.

Default: no comment

4.3.8 Transferring Service Units

amgr transfer {user | project | group }

4.3.8.1 Description

Transfers service units from specified project, user, or group, to specified project, user, or group. You can transfer from any of these to any of these.

To transfer between investors, use `amgr transfer group`.

When an investor is unlinked from a group or a group is unlinked from a project, some service units may remain unclaimed and become unusable. The administrator can take ownership of these unclaimed amounts and transfer it back into the budget pool to make it usable again.

A transfer has its own transaction ID.

4.3.8.2 Required Privilege

You must be *admin* to run this command.

4.3.8.3 Transferring Service Units for User

4.3.8.3.i Synopsis

amgr transfer user -n <divesting username> -s <service unit name> <service unit amount> -p <period> -F <divesting group> -T <receiving group> [-U <receiving user>] [-C <comment>]

4.3.8.3.ii Description

Transfer service units from one user to another user, within a group or between groups.

4.3.8.3.iii Options

-n, --name <divesting username>

String. User account from which to take service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Service unit name and amount to transfer.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-p, --period <period>

String. Period from which to take service units.

-F, --from-group <divesting group>

String. Group from which to take service units.

-T, --to-group <receiving group>

String. Group to receive service units.

-U, --dest-user <receiving user>

String. User account to receive service units. Do not use when transferring within same user account.

-C, --comment <comment>

String. Comment or reason for the transfer. Optional.

Default: no comment

4.3.8.4 Transferring Service Units for Project

4.3.8.4.i Synopsis

amgr transfer project -n <divesting project> -s <service unit name> <service unit amount> -p <period> -F <divesting group> -T <receiving group> [-P <receiving project>] [-C <comment>]

4.3.8.4.ii Description

Transfers service units from one project to another, within a group or between groups.

4.3.8.4.iii Options

-n, --name <divesting project>

String. Project from which to take service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Service unit name and amount to transfer.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-p, --period <period>

String. Period from which to take service units.

-F, --from-group <divesting group>

String. Group from which to take service units.

-T, --to-group <receiving group>

String. Group to receive service units.

-P, --dest-project <receiving project>

String. Project to receive service units. Do not use when transferring within the same project.

-C, --comment <comment>

String. Comment or reason for the transfer. Optional.

Default: no comment

4.3.8.5 Transferring Service Units for Investors and Group

4.3.8.5.i Synopsis

amgr transfer group -n <divesting group> -s <service unit name> <service unit amount> -F <divesting investor> -T <receiving investor> -G <receiving group> [-C <comment>]

4.3.8.5.ii Description

Transfers service units from one investor to another investor, within a group or between groups.

4.3.8.5.iii Options

-n, --name <divesting group>

String. Group from which to take service units.

-s, --serviceunits <service unit name> <service unit amount>

String and float. Service unit and amount to transfer.

Examples: `-s cpu_hrs 100` or `-s cpu_hrs 100.0`

-F, --from-investor <divesting investor>

String. Investor from which to take service units.

-T, --to-investor <receiving investor>

String. Investor to receive service units.

-G, --dest-group <receiving group>

String. Group to receive service units. Do not use when transferring within group.

-C, --comment <comment>

String. Comment or reason for the transfer. Optional.

Default: no comment

5 Basic Install and Configure

5.1 Basic Install and Configure Instructions

5.1.1 Assumptions

- You install Budget on the PBS server host
- You install Budget in the default locations
- You use the default billing formula

5.1.2 Installation

1. Log in as root.
2. Install utilities and `docker`:
   - For CentOS or RedHat:

     Log in to the machine where the Control package or the Cloud package is to be installed.

     ```
     yum install -y yum-utils
     yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
     yum install docker-ce docker-ce-cli containerd.io
     systemctl enable docker
     systemctl start docker
     yum install python3 python3-pip
     yum install openssl
     ```
   - For SLES12 or 15:

     Log in to the machine where Budget is to be installed.

     For SLES 12:

     ```
     sudo SUSEConnect -p sle-module-containers/12/x86_64 -r ''
     ```

     For SLES 15:

     ```
     sudo SUSEConnect -p sle-module-containers/15.1/x86_64 -r ''
     sudo zypper install docker
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```

     Configure the firewall to allow forwarding of Docker traffic to the external network:

     ```
     Set FW_ROUTE="yes" in /etc/sysconfig/SuSEfirewall2
     zypper install python3-pip
     ```
   - For Ubuntu:

     Log in to the machine where Budget is to be installed.

     ```
     sudo apt-get update
     sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-proper-
         ties-common
     curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
     sudo apt-key fingerprint 0EBFCD88
     ```

     The key should match the second line in the output; validate the last 8 characters. Example of second line:

     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88

     ```
     sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
         $(lsb_release -cs) stable"
     sudo apt-get update
     sudo apt-get install docker-ce docker-ce-cli containerd.io
     sudo apt-get install python3-pip
     ```

     This can install a number of required dependencies, and may take a few minutes.

     ```
     sudo apt-get install openssl
     sudo systemctl enable docker.service
     sudo systemctl start docker.service
     ```

3. Copy these installer tar files to your host machine:
   - The AMS installer, named:
     `ams_installation.tar.gz`
   - The Budget installer file, named:
     PBSPro-budget-server_<release number>-<OS>_x86_64.tar.gz
     For example:
     `PBSPro-budget-server_2021.1.0-CentOS7_x86_64.tar.g`
4. Install AMS:

```
tar xvfz ams_installation.tar.gz
cd ams-installation
python3 -m pip install --upgrade --ignore-installed pbsworks-packager/
/usr/local/bin/pkgr
```
(Please stay in the AMS installer directory for this step)

5. Answer the questions in the dialogue:
   - Choose option: 0
   - Hit Enter until license agreement complete and then answer Yes to accept.
   - Select Enter to continue
   - Choose Option 1 (Provide server hostname/IP address)
     <hostname>
     <IP address>
   - Choose Option: 0 (Skips providing any more machine details)
   - Install Location: <hostname>
   - Authentication Server: <hostname>
   - Authentication Port: Accept Default (If alternative to default 22 for sshd is used then provide alternative port ID)
   - Provide *admin* username: pbsadmin
   - Install Path: Accept default
6. Edit `/etc/ssh/sshd_config` and add the following lines:

```
Match Address 10.5.0.0/24
PasswordAuthentication yes
```

7. Make `sshd` reread its configuration file, and restart it:

```
systemctl daemon-reload
systemctl restart sshd
```

8. Create Certificates

```
cd /home/pbsadmin/
mkdir budget_certificates
export AM_DBUSER=pbsdata
openssl req -new -x509 -days 3650 -nodes -out budget_certificates/ca.crt -keyout
   budget_certificates/ca.key -subj "/CN=root-ca"
openssl req -new -nodes -out server.csr -keyout budget_certificates/server.key -subj "/CN=local-
   host"
openssl x509 -req -in server.csr -days 3650 -CA budget_certificates/ca.crt -CAkey
   budget_certificates/ca.key -CAcreateserial -out budget_certificates/server.crt
```

```
openssl req -new -nodes -out client.csr -keyout budget_certificates/client.key -subj
    "/CN=${AM_DBUSER}"
openssl x509 -req -in client.csr -days 3650 -CA budget_certificates/ca.crt -CAkey
    budget_certificates/ca.key -CAcreateserial -out budget_certificates/client.crt
rm -f server.csr client.csr
chmod og-rwx budget_certificates/*
```

9. Install Budget Module

   Extract software:

   tar xvfz PBSPro-budget-server_2021.1.0-<OS>_x86_64.tar.gz

   For example:

   ```
   tar xvfz PBSPro-budget-server_2021.1.0-CentOS7_x86_64.tar.gz
   ```

   ```
   cd /home/centos/software/am/
   ./install -t server -u pbsadmin -c /home/pbsadmin/budget_certificates
   ```

5.1.3 Configuration

1. Edit /etc/am.conf, and set the parameters appropriately.
2. Set PATH for all users:

   ```
   export PATH=/opt/am/python/bin:$PATH
   ```

3. On the Budget/PBS server host, edit /etc/sudoers, and make it look like this:

   ```
   Cmnd_Alias AM_SERVER_CMD = $AM _EXEC/python/bin/python3 $AM_EXEC/hooks/pbs/pbs_set_formula.py*
   Defaults!AM_SERVER_CMD !requiretty
   Cmnd_Alias AM_CLIENT_CMD = $AM_EXEC/python/bin/amgr sshlogin
   Cmnd_Alias AM_IMPORTS = $PBS_EXEC/bin/qmgr -c i h am_hook application/x-config default .am/tmp*,
       $PBS_EXEC/bin/qmgr -c i h am_hook_periodic application/x-config default .am/tmp*,
       $PBS_EXEC/bin/qmgr -c export hook am_hook application/x-config default
   pbsadmin ALL=(root) NOPASSWD: BUDGET_MANAGER_IMPORTS
   Defaults!AM_CLIENT_CMD !requiretty
   Defaults!AM_IMPORTS !requiretty
   ```

4. Enable and start Budget:

   ```
   systemctl enable pbs_budget
   systemctl start pbs_budget
   ```

5. Create resources:

   ```
   qmgr -c "c r am_job_amount type=string"
   qmgr -c "c r am_job_cache type=string,flag=m"
   ```

6. Define your billing periods.
7. Set up passwordless ssh for both pbsadmin and amteller. Do these steps once for each one:
   a. Log in to the PBS server as the username who needs passwordless ssh
   b. Check for an existing SSH key pair:

      ```
      ls -al ~/.ssh/id_*.pub
      ```

      If you find existing keys, you can use those or back up the old keys and generate a new one.

      To generate a new SSH key pair:

      ```
      ssh-keygen
      ```

   c. Copy the contents of id_rsa.pub
   d. Log in to the Budget server as the username who needs passwordless ssh
   e. Check for the .ssh directory. If it does not exist, create it:

      ```
      mkdir -p .ssh
      cd .ssh/
      ```

   f. Create the authorized_keys file in the .ssh directory:

      ```
      vi authorized_keys
      ```

   g. Paste the contents of id_rsa.pub that you copied from the PBS server and save the file.
   h. Change the permission of authorized_keys to *600*:

      ```
      chmod 600 authorized_keys
      ```

8. Log into the PBS server host

9. Create and configure the am_hook and am_hook_periodic hooks:

```
qmgr -c "c h am_hook"
qmgr -c "s h am_hook event='queuejob,runjob,execjob_epilogue,modifyjob,movejob'"
qmgr -c "s h am_hook order=1000"
qmgr -c "c h am_hook_periodic"
qmgr -c "s h am_hook_periodic event=periodic"
qmgr -c "s h am_hook_periodic freq=120"
qmgr -c "i h am_hook application/x-python default  /opt/am/hooks/pbs/am_hook.py"
qmgr -c "i h am_hook_periodic application/x-python default  /opt/am/hooks/pbs/am_hook.py"
qmgr -c "i h am_hook application/x-config default /opt/am/hooks/pbs/am_hook.json"
qmgr -c "i h am_hook_periodic application/x-config default /opt/am/hooks/pbs/am_hook.json"
qmgr -c "s h am_hook enabled=true"
qmgr -c "s h am_hook_periodic enabled=true"
qmgr -c "s h am_hook alarm=90"
```

10. At the Budget server host, add a cluster to represent the PBS complex with its billing model:

```
amgr add cluster <PBS server> -f am_hook.json
```

6
Using Budget

6.1 Managing Credit with Budget

You must log into Budget to perform any administrative tasks, or to manage credit as an investor or a manager, or to reconcile transactions as the teller.

To log in to Budget:

```
amgr login
<local host password>
```

To log out of Budget:

```
amgr logout
```

6.2 Submitting Jobs with Budget

Job submitters do not need to log into Budget.

6.2.1 Credit Balance

Each project has its own credit balance, and each job submitter has their own credit balance. Credit is measured in standard service units. A service unit represents usage of a PBS-tracked resource, such as CPU hours or GPU hours. A service unit can be treated like a currency. Group managers are responsible for depositing service units into user and project accounts.

You can charge jobs to your own account as long as you have sufficient credit. If you are part of a project, you can charge project jobs to that project's account, as long as the project has sufficient credit.

You can start a job only if the credit account it will use has sufficient credit. When the job starts, the requested credit is put into escrow. When the job finishes, any unused credit is returned to the account. Budget does not allow your credit balance to become negative.

You can run project jobs on the PBS complexes associated with the project. You can run your own jobs on the complexes associated with your account. You can run jobs only when your account is active.

6.2.2 Accounting Policy

Jobs are charged to the periods in which they run. A project job is charged according to the project's accounting policy. Your jobs are charged according to your accounting policy. An accounting policy is one of the following:

begin_period
: The user or project account is charged when the job begins.

end_period
: The user or project account is charged when the job ends.

proportionate

The user or project account is charged during all periods when the job runs, and each period is charged in proportion to the usage during that period.

6.2.3 Allocation Periods

Your credit balance is allocated in specific time periods. Each period has a defined start and end date. Credit in a period cannot be used when the period is over, although a group manager or administrator can transfer credit from one period to another. To see what periods have been defined:

```
amgr ls period
```

See section 4.2.2.6, "Listing Periods", on page 57.

6.2.4 Quotas on External Resources

There may be quotas set on externally-managed resources such as storage. A quota is a limit on a dynamic service unit. To see quotas, list all service units:

```
amgr ls serviceunit
```

See section 4.2.2.7, "Listing Service Units", on page 58.

6.2.5 Checking Your Credit Balance

When you check your credit balance, specify the period:

amgr checkbalance -n <your username> -p <period>

For example, if your username is MyUsername and the period in question is Q4:

```
amgr checkbalance -n MyUsername -p Q4
```

See section 4.3.2.4, "Checking Service Unit Balance for User", on page 86.

6.2.6 Listing Clusters

You can list the clusters that represent PBS complexes associated with Budget:

amgr ls cluster [-n <PBS server>] [-a <active>] [-f] [-l | -j]

For example:

```
amgr ls cluster -n Cluster1 -f
```

See section 4.2.2.5, "Listing Clusters", on page 57.

6.2.7 Resource Requirements for Jobs

Each job must request the compute resources that are used in the billing formula used at that complex. For the default formula, this means walltime and ncpus. Make sure that every PBS job requests ncpus and walltime when it runs. Each job can have these set at submission by the job submitter or later via `qalter`, can inherit a value from the server or queue, or can be assigned a value by a hook. For ncpus, the server attribute default_chunk.ncpus may take care of the requirement.

6.2.8 Charging Jobs to User or Project Account

To charge a job to a project account, use `qsub -P <project name>` to specify the project. For example, to run a job using ProjectScript for one hour and charge it to the project named "Project1":

```
qsub -P Project1 -l select=1:ncpus=1:mem=1gb -l walltime=1:00:00 ProjectScript
```

To charge a job to your own account, do not specify a project. For example, to run a job using MyScript for one hour and charge it to your own account:

```
qsub  -l select=1:ncpus=1:mem=1gb -l walltime=1:00:00 MyScript
```

6.2.9 Getting Reports on Usage and Transactions

You can get reports on your own usage:

amgr report user -n <username> [-s <service unit name> | -t <service unit type>] -h <group names> -p <period> -S <start date> -E <end date> [-l] [-o <output file>] [-r]

For example, to get a report on MyUsername for the period Q4:

```
amgr report user -n MyUsername -p Q4
```

See section 4.2.5.2, “Getting User Reports”, on page 69.

You can get reports on your transactions

amgr report transaction -i <transaction ID> [-l] -N <count> [-o <output-file>] [-r]

6.3 Tutorial

6.3.1 Tutorial on Configuring Budget

6.3.1.1 Prerequisites

- A working installation of PBS Professional, with at least two accounts that can submit and run jobs at the complex. In our example, the cluster is named Cluster1, and the users are User1 and User2. Substitute in your own names for the cluster and the users when going through the tutorial.
- Install Budget: follow the instructions in section 2.3, “Prerequisites”, on page 19, choose a configuration from section 2.4, “Recommended Configurations”, on page 22, and install Budget according to section 2.5, “Installation Steps”, on page 22.
- Create test accounts: In addition to an administrator account, you will need investor, manager, and job submitter accounts. Log in as root:

```
adduser Investor1
passwd Investor1 <password>
adduser Manager1
passwd Manager1 <password>
adduser User1
passwd User1 <password>
adduser User2
passwd User2 <password>
```

6.3.1.2 Tutorial Steps to Configure Budget

6.3.1.2.i Create Periods

1. Log in as, or switch user to pbsadmin:

   ```
   su pbsadmin
   ```

2. Log into amgr:

   ```
   amgr login
   ```

3. Create a parent period named "2021" representing the year 2021. See section 1.6.1, “Periods, Allocation Periods, Billing Periods”, on page 11.

   ```
   amgr add period -n 2021 -S 2021-01-01 -E 2021-12-31
   ```

4. Create a child period named "2021.Q2" that represents the second quarter of the year 2021, and add it to the parent period:

   ```
   amgr add period -n 2021.Q2 -S 2021-10-01 -E 2021-12-31 -p 2021
   ```

   For the purposes of this tutorial, we are somewhere in the second quarter of 2021, so that this is the default period.

6.3.1.2.ii Add PBS Complex to Budget

5. Add a cluster named Cluster1 to represent your PBS complex:

   ```
   amgr add cluster -n Cluster1
   ```

6. Deactivate Cluster1:

   ```
   amgr update cluster -n Cluster1 -a False
   ```

7. List Cluster1:

   ```
   amgr ls cluster -a False
   ```

8. Activate Cluster1:

   ```
   amgr update cluster -n Cluster1 -a True
   ```

9. List Cluster1 again:

   ```
   amgr ls cluster
   ```

6.3.1.2.iii Create Standard Service Unit

10. Create a standard service unit named "cpu_hrs" to represent CPU hours. See section 1.6.2, “Service Units”, on page 12. The service unit name must be identical to the one that is configured in the formulas section of the am_hook.json configuration file:

    ```
    amgr add serviceunit -n cpu_hrs -d "CPU Hours"
    ```

6.3.1.2.iv Add Users to Budget

11. Add a user who will be a group manager, a user who will be an investor, and users User1 and User2 who will submit jobs. When you add a user, you must assign a role, an accounting policy, and at least one cluster:

    ```
    amgr add user -n Manager1 -r manager -A begin_period -c Cluster1
    amgr add user -n Investor1 -r investor -A begin_period -c Cluster1
    amgr add user -n User1 -r user -A begin_period -c Cluster1
    amgr add user -n User2 -r user -A begin_period -c Cluster1
    ```

6.3.1.2.v Create Group

12. Create a group named "test_group" with *manager* Manager1 and *investor* Investor1:

```
amgr add group -n test_group -M Manager1 -I Investor1
```

6.3.1.2.vi Associate Job Submitters with Group

13. Associate User1 and User2 with the group test_group:

```
amgr update user -n User1 -h + test_group
amgr update user -n User2 -h + test_group
```

6.3.1.2.vii Create Project and Give It Cluster and User

14. Create a project named "P1" with accounting policy *begin_period*, cluster Cluster1, user User1 and group test_group:

```
amgr add project -n P1 -A begin_period -c Cluster1 -u User1 -h test_group
```

6.3.1.2.viii Invest in Group

15. Log in as Investor1:

```
amgr login
```

16. Invest 10000 CPU hours in test_group:

```
amgr deposit group -n test_group -s cpu_hrs 10000
```

6.3.1.2.ix Deposit Service Units to Project

17. Log in as Manager1:

```
amgr login
```

18. Allocate 1200 CPU hours from test_group to project P1 for the period 2021.Q2:

```
amgr deposit project -n P1 -s cpu_hrs 1200.00 -p 2021.Q2 -h test_group
```

19. Check the balance of project P1 for the 2021.Q2 period:

```
amgr checkbalance project -n P1 -p 2021.Q2
```

20. Withdraw 200 CPU hours from project P1 for the period 2021.Q2:

```
amgr withdraw project -n P1 -s cpu_hrs 200.00 -p 2021.Q2 -h test_group
```

6.3.1.2.x Deposit Service Units to Users

21. Allocate 1300 CPU hours from test_group to user User1 for the period 2021.Q2:

```
amgr deposit user -n User1 -s cpu_hrs 1300.00 -p 2021.Q2 -h test_group
```

22. Allocate 1400 CPU hours from test_group to user User2 for the period 2021.Q2:

```
amgr deposit user -n User2 -s cpu_hrs 1400.00 -p 2021.Q2 -h test_group
```

6.3.1.2.xi Run User Job

23. Log in as User1.
24. Run a sleep job for 10 seconds with a walltime of 2 minutes, and charge it to user User1:

```
qsub -lwalltime=00:02:00 -- /bin/sleep 10
```

25. Check the credit balance for user User1. It will have decreased:

```
amgr checkbalance user -n User1 -p 2021.Q2
```

26. After the job is finished, check the balance again. You should see that the unused amount has been returned:

```
amgr checkbalance user -n User1 -p 2021.Q2
```

6.3.1.2.xii Run Project Job

27. Run a sleep job for 10 seconds with a walltime of 2 minutes, and charge it to project P1:

```
qsub -P P1 -lwalltime=00:02:00 -- /bin/sleep 10
```

28. To see the job running:

```
qstat -sw
```

29. Log in as Manager1.
30. Check the credit balance for project P1. It will have decreased:

```
amgr checkbalance project -n P1 -p 2021.Q2
```

31. After the job is finished, check the balance again. You should see that the unused amount has been returned:

```
amgr checkbalance project -n P1 -p 2021.Q2
```

6.3.1.2.xiii Non-project User Tries to Run Project Job

32. Log in as User2.
33. Run a sleep job for 10 seconds with a walltime of 2 minutes, and charge it to user User2:

```
qsub -lwalltime=00:02:00 --/bin/sleep 10
```

34. Check the credit balance for user User2. It will have decreased:

```
amgr checkbalance user -n User2 -p 2021.Q2
```

35. After the job is finished, check the balance again. You should see that the unused amount has been returned:

```
amgr checkbalance user -n User2 -p 2021.Q2
```

36. Run a sleep job for 10 seconds with a walltime of 2 minutes, and charge it to project P1:

```
qsub -P P1 -lwalltime=00:02:00 -- /bin/sleep 10
```

This job cannot run, because User2 is not part of project P1.

Index

A

account
  group BG-7
  project BG-8
  user BG-10
accounting policy
  definition BG-17
accounts
  required user accounts BG-20
acquire BG-16
active BG-2
adding cluster BG-40
adding job submitters BG-11
admin
  role BG-4
administrator BG-4, BG-20
  user account BG-20
allocation BG-2
allocation period BG-11
  defining BG-39
Altair License Manager BG-19
am.conf BG-35, BG-36
AM_AUTH_ENDPOINT BG-35
AM_AUTH_TIMEOUT BG-35
AM_BALANCE_PRECHECK BG-35
AM_DBPORT BG-35
AM_DBUSER BG-35
AM_EXEC BG-35
AM_HOME BG-35
am_hook BG-40
  creating and configuring BG-44
am_hook.json BG-40
am_hook_periodic BG-40
AM_LICENSE_ENDPOINT BG-35
AM_PORT BG-35
AM_SERVER BG-35
AM_WORKERS BG-35
amgr
  help BG-47
amgr acquire BG-89
amgr add BG-49
amgr checkbalance BG-85
amgr deposit BG-83
amgr limit BG-80
amgr ls BG-55
amgr precheck BG-88
amgr reconcile BG-91
amgr refund BG-93
amgr rm BG-66
amgr sync formula BG-82
amgr transfer BG-94
amgr update BG-60
amgr withdraw BG-87
AMS BG-2, BG-24
  basic installation BG-99
  installing BG-24
attribute
  PBS, in billing formula BG-41
  project
    metadata BG-10

B

basic installation
  AMS BG-99
  Budget BG-100
billing formula BG-2
  constants BG-41
  defining BG-40
  defining service units BG-43
  file BG-40
  operators BG-42
  PBS attributes BG-41
  PBS resources BG-41
billing period BG-11
  creating BG-39
Budget
  administrator BG-4, BG-20
  basic configuration BG-101
  basic installation BG-100
  configuration tutorial BG-105
  enabling BG-28
  entity BG-2
  hooks BG-1
    configuration file BG-43
    creating and configuring BG-44
  installing BG-28
  instance BG-1
  logging in and out BG-103
  requirements
    user accounts BG-20
  starting BG-28
  teller BG-21

upgrading BG-32

C

certificates
creating BG-26
charging for jobs BG-5
checkbalance BG-16
cluster
adding BG-40
definition BG-17
commands
list BG-48
PATH BG-47
complex BG-3
adding BG-40
configuration
Budget
file BG-35
failover BG-29
parameters
basic configuration BG-101
tutorial BG-105
constants
in billing formula BG-41
consuming credit BG-6
Corosync BG-29
creating
certificates for encryption BG-26
service units
dynamic BG-14
standard BG-13
credit
consuming BG-6
investing in groups BG-5
investing in users and projects BG-5
reconciling BG-6
cron BG-14
currency BG-1, BG-13

D

data_lifetime BG-14
setting value BG-45
database
user account BG-21
deposit BG-16
Docker
basic installation BG-98
installing BG-23
passwords BG-24
docker-ce BG-20
docker-ee BG-20
dynamic service units
definition BG-13
updating values BG-14

E

element BG-2
enabling Budget BG-28
entity BG-2
environment variables
PATH BG-36
escrow BG-1, BG-6, BG-16, BG-91, BG-103

F

failover BG-29
configuring BG-29
file
billing formula BG-40
Budget configuration BG-35
formula BG-40
hook configuration BG-40
sudoers
modifications for Budget BG-36
formats
name BG-18
formula BG-2
formula file BG-40

G

group BG-7
account BG-7
definition BG-7

H

hooks BG-1
configuration file BG-40
creating and configuring BG-44

I

inactive BG-2
installation
basic BG-97
Docker BG-98
utilities BG-98
installing
AMS BG-24
Budget BG-28
Docker BG-23
utilities BG-23
instance
definition BG-3
instance of Budget BG-1
instructions
for job submitters BG-103
investing BG-5

in groups BG-5
in projects BG-5
in users BG-5
investor
actions BG-5
role BG-4

J

job submitters
adding BG-11
instructions BG-103
user account BG-21
jobs
charging BG-5
reconciling BG-6
requirements BG-20
submitting BG-103

L

list of commands BG-48
logging into Budget BG-103
logging out of Budget BG-103

M

manager
actions BG-5
role BG-4
metadata BG-10

O

operators
in billing formula BG-42

P

Pacemaker BG-29
passwordless ssh BG-38
PATH
for commands BG-47
setting BG-36
PBS attribute
in billing formula BG-41
PBS complex BG-3
adding BG-40
PBS Professional BG-19
PBS resource
in billing formula BG-41
pcs BG-29
peer scheduling BG-46
period BG-11
defining BG-39
definition BG-11
hierarchy BG-11
Postgres BG-1
precheck BG-16
project BG-8
account
definition BG-8
attributes
metadata BG-10
definition BG-8
python3 BG-20
python3-pip BG-20

Q

quotas
example BG-14
setting BG-13

R

reconcile BG-16
reconciling
credit BG-6
jobs BG-6
refund BG-16
requirements
for job submitters BG-21
for jobs BG-20
resource
in billing formula BG-41
resources for Budget BG-38
role BG-4
admin BG-4
investor BG-4
actions BG-5
manager BG-4
actions BG-5
teller BG-4
user BG-4

S

service units
abandoned BG-7
definition BG-12
dynamic
creating BG-14
updating values BG-14
usage BG-13
in billing formula BG-43
standard
creating BG-13
usage BG-13
SLES
restrictions BG-19
software
third-party BG-20

ssh
    passwordless BG-38
standard service units
    definition BG-13
starting Budget BG-28
submitting jobs BG-103
sudoers file
    modifications for Budget BG-36

T
teller BG-21
    role BG-4
third-party software BG-20
transaction
    definition BG-16
transaction ID BG-17
transfer BG-16
transferring
    abandoned service units BG-7
tutorial
    configuring Budget BG-105

U
units
    in billing formula BG-42
upgrading Budget BG-32
user
    account BG-10
    accounts
        required BG-20
    role BG-4
user account
    administrator BG-20
    database user BG-21
    job submitter BG-21
    teller BG-21
utilities
    basic installation BG-98
    installing BG-23

V
VPN BG-22

W
withdraw BG-16
worker
    definition BG-3
workers BG-1



Altair PBS Professional 2021.1

Simulate Guide

You are reading the Altair PBS Professional 2021.1

Simulate Guide (SG)

Updated 3/16/21

Copyright © 2003-2021 Altair Engineering, Inc. All rights reserved.

ALTAIR ENGINEERING INC. Proprietary and Confidential. Contains Trade Secret Information. Not for use or disclosure outside of Licensee's organization. The software and information contained herein may only be used internally and are provided on a non-exclusive, non-transferable basis. Licensee may not sublicense, sell, lend, assign, rent, distribute, publicly display or publicly perform the software or other information provided herein, nor is Licensee permitted to decompile, reverse engineer, or disassemble the software. Usage of the software and other information provided by Altair (or its resellers) is only as explicitly stated in the applicable end user license agreement between Altair and Licensee. In the absence of such agreement, the Altair standard end user license agreement terms shall govern.

Use of Altair's trademarks, including but not limited to "PBS™", "PBS Professional®", and "PBS Pro™", "PBS Works™", "PBS Control™", "PBS Access™", "PBS Analytics™", "PBScloud.io™", and Altair's logos is subject to Altair's trademark licensing policies. For additional information, please contact Legal@altair.com and use the wording "PBS Trademarks" in the subject line.

For a copy of the end user license agreement(s), log in to https://secure.altair.com/UserArea/agreement.html or contact the Altair Legal Department. For information on the terms and conditions governing third party codes included in the Altair Software, please see the Release Notes.

This document is proprietary information of Altair Engineering, Inc.

Contact Us

For the most recent information, go to the PBS Works website, www.pbsworks.com, select "My PBS", and log in with your site ID and password.

Altair

Altair Engineering, Inc., 1820 E. Big Beaver Road, Troy, MI 48083-2031 USA www.pbsworks.com

Sales

pbssales@altair.com 248.614.2400

Please send any questions or suggestions for improvements to agu@altair.com.

Technical Support

Need technical support? We are available from 8am to 5pm local times:

Location	Telephone	e-mail
Australia	+1 800 174 396	anz-pbssupport@india.altair.com
China	+86 (0)21 6117 1666	pbs@altair.com.cn
France	+33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany	+49 (0)7031 6208 22	pbssupport@europe.altair.com
India	+91 80 66 29 4500 +1 800 208 9234 (Toll Free)	pbs-support@india.altair.com
Italy	+39 800 905595	pbssupport@europe.altair.com
Japan	+81 3 6225 5821	pbs@altairjp.co.jp
Korea	+82 70 4050 9200	support@altair.co.kr
Malaysia	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
North America	+1 248 614 2425	pbssupport@altair.com
Russia	+49 7031 6208 22	pbssupport@europe.altair.com
Scandinavia	+46 (0)46 460 2828	pbssupport@europe.altair.com
Singapore	+91 80 66 29 4500 +1 800 425 0234 (Toll Free)	pbs-support@india.altair.com
South Africa	+27 21 831 1500	pbssupport@europe.altair.com
South America	+55 11 3884 0414	br_support@altair.com
UK	+44 (0)1926 468 600	pbssupport@europe.altair.com

Contents

About PBS Documentation vii

1 Installing Simulate 1

1.1 Supported Platforms for Simulate . . . 1
1.2 Prerequisites . . . 1
1.3 Where to Install Simulate . . . 1
1.4 Installation . . . 1
1.5 Configuration . . . 2
1.6 Setting Up User Environment . . . 2

2 Using Simulate 3

2.1 Introduction . . . 3
2.2 Simulate Commands . . . 3
2.3 Evaluating and Adjusting Job Order . . . 4
2.4 Using Simulate on a Snapshot . . . 5

Index 11

Contents

1 Installing Simulate

1.1 Supported Platforms for Simulate

Simulate supports the following platforms:

Table 1-1: Supported Platforms

Maker	Version	Chipset	Notes
Red Hat Enterprise Linux RHEL	7, 8	x86_64	
	7, 8	ARM64	
SUSE SLES	12	x86_64	
	15	x86_64	
	15	ARM64	
Ubuntu	18.04	x86_64	
	18.04	ARM64	
	20.04	x86_64	
	20.04	ARM64	

1.2 Prerequisites

- Altair License Manager 14.5+
- PBSProNodes or PBSProSockets 20.0 License Feature

1.3 Where to Install Simulate

You can install Simulate on any host that can reach the Altair License Manager. We recommend not installing it on the PBS server host, to avoid slowing that host.

For information about PBS Professional, see the *PBS Professional Administrator's Guide*.

1.4 Installation

1. Untar the Simulate package:
   tar xvfz PBSPro-sim_2021.1.0-<OS><OS version>_x86_64.tar.gz
   For example:
   `tar xvfz PBSPro-sim_2021.1.0-SLES15_x86_64.tar.gz`

2. Change directory:

```
cd PBSPro-sim_2020.1.0
```

1.5 Configuration

1.5.1 Configure Licensing for Simulate

The Simulate configuration file is named sim.conf and is in the installation directory.

In this file, edit the licensing parameter, and set it to <port>@<license server host>:

```
SIM_LICENSE_LOCATION=6200@<license server>
```

For example:

```
SIM_LICENSE_LOCATION=6200@mylicensehost
```

1.5.2 Set Path to Snapshot Directory

Optionally, if you want to, you can set the PBS_SNAPSHOT_PATH parameter in `sim.conf` to be the path to the snapshot directory. If you are working repeatedly on the same snapshot, this can make your command execution easier.

1.6 Setting Up User Environment

Set up your environment to point to the `simsh` wrapper command for simulator:

```
export PATH=/<your_install_path>/PBSPro-sim_2020.1.0:$PATH
```

2 Using Simulate

2.1 Introduction

Simulate is useful for seeing the order in which jobs will run, and whether jobs will run, based on a snapshot taken from a working PBS complex. You can use this information to tune your PBS Professional configuration and change the workload behavior in order to meet your site's objectives.

2.2 Simulate Commands

2.2.1 The simsh Wrapper Script

simsh <path to snapshot> <command to execute> [command options]

simsh <path to snapshot> sim [sim options]

Command to execute for manipulating snapshots: `qmgr`, `qstat`, `qsub`, `pbsnodes`, `qdel`, `qselect`, `pbsfs`, `pbs_rstat`, `pbs_rsub`, `pbs_stat`, `tracejob`

Option for running simulations: `sim`

Other options: `--help`, `--version`

2.2.2 The sim command

This command performs simulation. You call `sim` as a subcommand to the `simsh` script.

2.2.2.1 Options to the sim Command

-o <output path>
: Path to output snapshot (default: <path to input snapshot>_out)

-t <duration>
: Time in seconds to to run simulated universe. Actual time required is much shorter. When used with the `-n <cycles>` option, simulation runs for the shorter time. For example, if you specify "-t 600 -n 2" and it only takes 30 seconds to run two scheduling cycles, the simulation ends after 30 seconds.

-h, --help
: Display this help message

-l
: Generate simulation log

-L
: Generate scheduling logs

-n <cycles>

Number of scheduling cycles to run. When used with the `-t <duration>` option, simulation runs for the shorter time. For example, if you specify "-t 600 -n 2" and it only takes 30 seconds to run two scheduling cycles, the simulation ends after 30 seconds.

--version

Print the version of this command

--monitor=<duration>

Update the snapshot after every period specified by *duration*

--O0

Turn optimizations off

2.2.3 Command Examples

To get server information:

```
simsh <path to snapshot> qstat -Bf
```

To list all nodes:

```
simsh <path to snapshot> qmgr -c "list node @default"
```

To run a simulation for 10 cycles and generate scheduling logs:

```
simsh <path to snapshot> sim -L -n 10
```

To list all vnodes:

```
simsh <path to snapshot> pbsnodes -av
```

2.2.4 Command Paths

Many Simulate commands have the same names as PBS commands. The wrapper script `simsh` prevents naming clashes.

2.3 Evaluating and Adjusting Job Order

The `--eval-formula` option to the `pbs_stat` command computes the execution priority for each job, and displays the jobs in priority order along with their priorities. You can see the order in which jobs would run using this:

```
simsh <snapshot> pbs_stat --eval-formula
```

Once you see the job ordering and priorities, you can tweak your formula or formula inputs, repeating until you have the desired order and priorities; see section 2.4.7, “Modifying Your Snapshot”, on page 7. After the jobs have the desired ordering, you can run a simulation to see the exact ordering and timing of job execution.

See section 4.9.21, “Using a Formula for Computing Job Execution Priority”, on page 149.

2.3.1 Prerequisites

Your snapshot must contain a valid scheduler formula and the list of queued jobs. You can capture this using pbs_snapshot --basic -o <output path>.

2.4 Using Simulate on a Snapshot

Overview:

- Get a snapshot from a running PBS Professional complex
- Check snapshot contents
- Run Simulate on the snapshot
- Review your results
- Modify your snapshot
- Run a simulation on your modified snapshot

2.4.1 Creating a Snapshot

This should be done on a system with queued workload. Simulate is useful for seeing what order jobs will run in and if they will run based on a "now" status taken from a working PBS complex:

```
pbs_snapshot --basic -o <output_path>
```

Note: A tar archive will be created in this folder.

Unpack the snapshot:

```
tar xvfz <snapshot_name>.tgz (e.g. tar xvfz snapshot_20200708_15_54_57.tgz)
```

2.4.2 Checking Snapshot Contents

To use `qstat` to inspect the contents of a snapshot:

simsh <snapshot_directory_name> qstat -as1

For example:

```
simsh snapshot_20200708_15_54_57 qstat -as1
```

Try the different `qstat` options; many are available. (-f is not available on a snapshot.)

To use `qmgr`:

simsh <snapshot_directory_name> qmgr

For example:

```
simsh snapshot_20200708_15_54_57 qmgr
```

You can use `qmgr` commands by prepending them with `simsh <snapshot>`. If you want to target all queues or nodes at a server use @default as the server name.

You can also use the commands to alter the configuration, though we would recommend running a simulation first to see how it runs through with the system configuration.

To use `pbsnodes`:

simsh <snapshot_directory_name> pbsnodes -av

For example:

```
simsh snapshot_20200708_15_54_57 pbsnodes -av
```

2.4.3 Running a Simulation

simsh <snapshot_directory_name> sim

For example:

```
simsh snapshot_20200708_15_54_57 sim
```

This should give some output showing how the workload ran through, e.g.:

```
### Snapshot: snapshot_20200708_15_54_57 ###

Creating usage database for fairshare.
Number of scheduling cycles run: 32
Number of jobs submitted: 0
Number of jobs run: 22
Number of jobs left over: 0
Time taken to simulate: 0
```

2.4.3.1 Simulation Errors

If you get a licensing error: "Error: No valid licenses found, quitting", make sure the licensing prerequisites are met.

2.4.3.2 Simulation Caveats

The simulator ignores job environment variables (it does not use the value of the job's Variable_List attribute).

2.4.4 Simulation Output

2.4.4.1 Output Name

By default, running a simulation on snapshot_20200708_15_54_57 produces the result snapshot named "snapshot_20200708_15_54_57_out". However, you can specify the name of the output using sim -o <snapshot name>.

If you don't specify a name, when you run a simulation on a snapshot, the output is a snapshot with "_out" appended to the name of the original. So <snapshot_directory_name> results in <snapshot_directory_name_out>.

For example, running a simulation on "snapshot_20200708_15_54_57" creates a snapshot named "snapshot_20200708_15_54_57_out".

2.4.4.2 Output Contents

When you run the following:

```
simsh <snapshot> sim
```

the simulator produces these outputs:

- Number of scheduling cycles run: <number>
- Number of jobs submitted <number>
- Number of jobs run: <number>
- Number of jobs left over: <number>

  These are the jobs which couldn't be run by the simulator for reasons such as the job requests an unavailable queue or resource, or the job exceeded a limit (which might have been placed on the snapshot).
- Time taken to simulate: <seconds>

2.4.5 Reviewing Simulation Results

Review the snapshot produced by the simulation. You can use the commands listed in "Checking Snapshot Contents" on page 5.

To use `qstat` to review the results created when you ran a simulation and produced <snapshot_directory_name_out>:

simsh <snapshot_directory_name_out> qstat -as1

For example, if the result of the simulation is "snapshot_20200708_15_54_57_out":

```
simsh snapshot_20200708_15_54_57_out qstat -as1
```

2.4.6 Using Scheduler Logs

When you run a simulation, you can use the `sim -L` command to produce scheduler logs, which you can use to see how jobs were scheduled:

simsh <snapshot> sim -L

2.4.7 Modifying Your Snapshot

You can modify your simulated workload environment, to see how jobs would run in those circumstances. You modify the parent snapshot, not the output snapshot, then re-run the simulation using the modified parent snapshot. You can modify many of the characteristics of a PBS complex; we mention a few ways here.

2.4.7.1 Modifying Available Resources

You can change the available resources in your shapshot, such as memory, CPUs, number of execution hosts, etc. To add more nodes you edit the node configuration in the parent snapshot. To add CPUs, change the ncpus resource on the parent snapshot. For example, here we use `qmgr` on the parent snapshot to modify the node to have 32 cores:

```
simsh <snapshot_directory_name> qmgr -c "s n testing resources_available.ncpus=32"
```

2.4.7.2 Creating Nodes

You can create new nodes:

```
simsh <snapshot_directory_name> qmgr -c "c n newnode resources_available.ncpus=8"
```

2.4.7.3 Submitting Jobs

You can use `qsub` to submit more jobs:

```
simsh <snapshot_directory_name> qsub ...
```

2.4.7.4 Changing Limits

You can change limits, for example:

```
simsh <snapshot_directory_name> qmgr -c "set server max_run += [u:PBS_GENERIC=10]"
```

2.4.7.5 Changing Scheduler Configuration File

You can change the scheduler configuration file.

Change directory into the snapshot directory:

```
cd <path to snapshot>/sched_priv
```

Edit the configuration file named sched_config.

2.4.8 Rerunning a Simulation

To rerun a simulation:

simsh <snapshot_directory_name> sim

For example:

```
simsh snapshot_20200708_15_54_57 sim
```

2.4.9 Output of Rerunning a Simulation

Rerunning a simulation produces an output snapshot with the name of the parent snapshot with "_out" appended, in addition, the previous output snapshot has been renamed with an additional "_out" appended to it: the previous output is now at <snapshot_directory_name_out_out> in case you need to reference or review the results.

So if you start with a snapshot named "A", and run a simulation on it, you get "A_out". But if you run another simulation on A, you get a new A_out, and the previous output snapshot is now named A_out_out.

2.4.10 Reviewing Second Results

Review the new output snapshot, using the methods in "Checking Snapshot Contents" on page 5. You can use any of the commands in "Simulate Commands" on page 3.

In our example, the new output snapshot is snapshot_20200708_15_54_57_out, and to examine it using `qstat`:

```
simsh snapshot_20200708_15_54_57_out qstat -as1
```

You can look for changes in job run order between the accounting logs from the original and modifed snapshots.

Note that `qstat -s` gives empty results because all the jobs have run through the system.

2.4.11 Command Options for Post-processing Simulation Output Snapshot

2.4.11.1 Find Jobs that Did Not Run

You can find jobs that have not yet run in an input snapshot, and jobs that never did run in an output snapshot.

simsh <snapshot> qstat <options>

2.4.11.2 Figure Out Why Job Did Not Run

For a specific job, you can see the output of every scheduler cycle for the duration you specify, by examing a simulation output snapshot that was created using the `-L` option to the `sim` command.

simsh <snapshot> tracejob -n <days> <job ID>

The tracejob command finds the data for the specified job, and presents it in chronological order. It examines all of the days from today back to the limit in number of days back you specify in <days>. If the snapshot was produced on a previous date, take the difference between that date and today into account when you specify <days>. For example, if the snapshot was taken 200 days ago, and you want 5 days of log information, specify 205 days. Usually you want to cover the entire snapshot log, so you can just make this very large, for example 1000.

2.4.11.3 Examine Job Execution Timing

After a simulation, you can see the order in which jobs ran, along with their start and end times:

simsh <snapshot> pbs_stat -S

Index

A
Altair License Manager SG-1

C
checking snapshot contents SG-5
configuration file SG-2
creating a snapshot SG-5

M
modifying snapshot SG-7

O
output of rerunning a simulation SG-8
output of simulation SG-6

P
pbs_rstat SG-3
pbs_rsub SG-3
pbs_sim SG-3
PBS_SNAPSHOT_PATH SG-2
pbs_stat SG-3
pbsfs SG-3
pbsnodes SG-3

Q
qdel SG-3
qmgr SG-3
qselect SG-3
qstat SG-3
qsub SG-3

R
rerunning a simulation SG-8
results of simulation SG-7
running a simulation SG-5

S
SIM_LICENSE_LOCATION SG-2
simsh SG-3
Simulate configuration file SG-2
simulation
    output SG-6
    output of rerun SG-8
    rerunning SG-8
    results SG-7
    running SG-5
snapshot
    checking contents SG-5
    creating SG-5
SuSE SG-1

T
tracejob SG-3

U
user environment SG-2
using Simulate on a snapshot SG-5

W
wrapper script SG-3

Main Index

$action RG-240
$alps_client RG-240
$alps_release_jitter AG-476, RG-240
$alps_release_timeout RG-240
$alps_release_wait_time AG-477, RG-241
$checkpoint_path RG-241
$clienthost RG-241
$cputmult RG-241
$dce_refresh_delta RG-241
$enforce RG-241
$job_launch_delay RG-243
$jobdir_root RG-242
$logevent RG-244
$logevent MoM parameter AG-548
$max_check_poll RG-244
$max_load RG-244
$max_poll_downtime RG-244
$min_check_poll RG-244
$pbs_accounting_workload_mgmt RG-245
$prologalarm RG-245
$reject_root_scripts RG-245
$restart_background RG-245
$restart_transmogrify RG-245
$restrict_user AG-391, RG-245
$restrict_user_exceptions AG-391, RG-245
$restrict_user_maxsysid AG-391, RG-246
$restricted RG-246
$sister_join_job_alarm RG-246
$suspendsig RG-246
$tmpdir RG-246
$usecp RG-246
$vnodedef_additive RG-247
$wallmult RG-247
.rhosts AG-377
_NEC_HCA_LIST_IO UG-209
_NEC_HCA_LIST_MPI UG-210
_NECMPI_VE_NODELIST UG-209
_NECMPI_VE_NUM_NODES UG-209
_VENODELIST UG-209
job.resources_used HG-126

A

accelerator AG-474, RG-265, UG-191
accelerator_memory AG-474, RG-266, UG-191
accelerator_model AG-474, RG-266, UG-191
accept an action HG-5, RG-1
access
    by group AG-362, RG-8
    by user AG-362, RG-20
        effect of flatuid AG-376
    control lists AG-362
    from host AG-362, RG-8
    to a queue AG-362, RG-1
    to a reservation AG-362, RG-1
    to server AG-362
    to the server RG-1
access control list RG-1
account
    group BG-7
    installation IG-13
    PBS service IG-14
    project BG-8
    user BG-10
account string RG-1
Account_Name
    job attribute RG-328
accounting UG-221
    account AG-632, AG-637
    account string RG-1
    alt_id AG-632, AG-637
    authorized_hosts AG-630
    authorized_users AG-630
    ctime AG-630, AG-631, AG-632, AG-634, AG-637, AG-638, AG-639, AG-640
    duration AG-630
    end AG-630, AG-632, AG-638
    etime AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-640
    Exit_status AG-633, AG-638
    group AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
    jobname AG-631, AG-633, AG-637, AG-638, AG-639, AG-641
    name AG-630
    owner AG-630
    qtime AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
    queue AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
    Resource_List AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641
    session AG-631, AG-634, AG-635, AG-638, AG-

640, AG-641
start AG-630, AG-631, AG-634, AG-635, AG-638, AG-640, AG-641
user AG-631, AG-634, AG-635, AG-637, AG-638, AG-640, AG-641
accounting log entry
format RG-353
accounting policy
definition BG-17
accounting_id AG-632, AG-637
job attribute RG-328
accounts
required user accounts BG-20
accrue_type
job attribute RG-328
ACCT_TMPDIR UG-221
ACL RG-1, RG-379, RG-382, RG-383, RG-385
acl_group_enable
queue attribute AG-370, RG-311
acl_groups
queue attribute AG-370, RG-311
acl_host_enable RG-281
queue attribute AG-370, RG-311
server attribute AG-370
acl_host_moms_enable RG-281
acl_hosts
queue attribute AG-370, RG-311
server attribute AG-370, RG-281
acl_resv_group_enable
server attribute RG-281
acl_resv_groups
server attribute RG-281
acl_resv_host_enable
server attribute RG-281
acl_resv_hosts
server attribute RG-282
acl_resv_user_enable
server attribute RG-282
acl_resv_users
server attribute RG-282
acl_roots AG-394
server attribute RG-282
acl_user_enable
queue attribute AG-370, RG-311
server attribute AG-370, RG-282
acl_users
queue attribute AG-370, RG-311
server attribute AG-370, RG-282
ACLs AG-362
default behavior AG-363
format AG-363
group AG-364
host AG-364
matching entry AG-365
modifying behavior AG-363
overrides AG-376
removing entity AG-367
rules for creating AG-367
user AG-364
who can create AG-368
acquire BG-16
action HG-5, RG-1
accept RG-1
reject RG-16
activate a power profile AG-319
active BG-2
active (failover) RG-1
Active Directory IG-13, RG-1
activereq PG-100
adding cluster BG-40
adding job submitters BG-11
addreq PG-100
Admin IG-13, RG-1
admin
role BG-4
administrator RG-2, BG-4, BG-20
user account BG-20
Administrators RG-2
administrators IG-13
advance reservation AG-198, AG-630, RG-2, RG-392, UG-135
creation UG-137
aggressive_provision AG-329, RG-255
alarm
hook attribute RG-350
allocation BG-2
allocation period BG-11
defining BG-39
allreq PG-100
ALM license server RG-2
alt_id
job attribute RG-328
Altair License Manager BG-19, SG-1
Altair License Server CG-6
am.conf BG-35, BG-36
AM_AUTH_ENDPOINT BG-35
AM_AUTH_TIMEOUT BG-35
AM_BALANCE_PRECHECK BG-35
AM_DBPORT BG-35
AM_DBUSER BG-35
AM_EXEC BG-35
AM_HOME BG-35

am_hook BG-40
creating and configuring BG-44
am_hook.json BG-40
am_hook_periodic BG-40
AM_LICENSE_ENDPOINT BG-35
AM_PORT BG-35
AM_SERVER BG-35
AM_WORKERS BG-35
Amazon Web Services CG-5
Ames Research Center RG-411
amgr
help BG-47
amgr acquire BG-89
amgr add BG-49
amgr checkbalance BG-85
amgr deposit BG-83
amgr limit BG-80
amgr ls BG-55
amgr precheck BG-88
amgr reconcile BG-91
amgr refund BG-93
amgr rm BG-66
amgr sync formula BG-82
amgr transfer BG-94
amgr update BG-60
amgr withdraw BG-87
AMS BG-2, BG-24
basic installation BG-99
installing BG-24
AOE AG-327, RG-2, UG-215
using UG-216
aoe RG-266
aoe resource
defining AG-336
API RG-2
application checkpoint RG-2
application license
floating AG-276
definition AG-231
floating externally-managed AG-276
application licenses AG-274
floating UG-54
floating license PBS-managed AG-277
license units and features AG-275
node-locked
per-CPU UG-55
overview AG-257
per-host node-locked example AG-279
types AG-274
application operating environment RG-2
arch RG-267
argument_list
job attribute RG-329
array
job attribute RG-329
array job RG-2, RG-9
array_id
job attribute RG-329
array_index
job attribute RG-329
array_indices_remaining
job attribute RG-329
array_indices_submitted
job attribute RG-329
array_state_count
job attribute RG-330
ASAP reservation AG-198, RG-2, RG-10
attribute
definition RG-2
log_events RG-298
PBS, in billing formula BG-41
project
metadata BG-10
rerunnable RG-16
attribute name
format RG-353
authentication AG-609
authorization IG-12, AG-609
Authorized_Groups
reservation attribute RG-303
Authorized_Groups reservation attribute AG-371
Authorized_Hosts
reservation attribute RG-303
Authorized_Hosts reservation attribute AG-371
Authorized_Users
reservation attribute RG-304
Authorized_Users reservation attribute AG-371
average CPU usage enforcement AG-307
average_cpufactor AG-307
average_percent_over AG-307
average_trialperiod AG-307
avoid_provision AG-329, RG-255
AWS CG-5
Azure cloud head node CG-127

B

backfill RG-250
backfill_depth
queue attribute RG-311
server attribute RG-282
backfill_prime AG-195, RG-250
backfilling RG-2
backup directory
overlay upgrade IG-72, IG-73, IG-83, IG-85, IG-96
Windows upgrade IG-111, IG-126, IG-127
basic fairshare AG-139

basic installation
AMS BG-99
Budget BG-100
batch job RG-9
batch processing RG-3
batch requests AG-548
billing formula BG-2
constants BG-41
defining BG-40
defining service units BG-43
file BG-40
operators BG-42
PBS attributes BG-41
PBS resources BG-41
billing period BG-11
creating BG-39
block
job attribute RG-330
blocking jobs UG-120
Boolean
format AG-237, RG-259, RG-359, UG-49
borrowing vnode AG-230, AG-271, RG-3
Budget
administrator BG-4, BG-20
basic configuration BG-101
basic installation BG-100
configuration tutorial BG-105
enabling BG-28
entity BG-2
hooks BG-1
configuration file BG-43
creating and configuring BG-44
installing BG-28
instance BG-1
logging in and out BG-103
requirements
user accounts BG-20
starting BG-28
teller BG-21
upgrading BG-32
built-in hook HG-5, RG-3
built-in resource AG-231, RG-3
burst CG-1
burst_by_hook CG-12
busy RG-365
by_queue RG-250

C

capmc IG-139
CentOS IG-23, CG-5
certificates
creating BG-26
changing order of jobs UG-168
charging for jobs BG-5
checkbalance BG-16
checking snapshot contents SG-5
Checkpoint
job attribute RG-330
checkpoint AG-630, AG-665, RG-240, RG-390, RG-409
preemption via AG-188
restart RG-17
restart file RG-17
restart script RG-17
checkpoint and abort RG-3
checkpoint and restart RG-3
checkpoint/restart RG-3
checkpoint_abort RG-3, RG-240
checkpoint_min
queue attribute RG-312
child vnode RG-3
chunk AG-231, RG-3, UG-51, UG-53
chunk set RG-3
chunk-level resource RG-3, UG-51
CLE 6 and 7 IG-139
client commands IG-4
clienthost AG-391
closerm PG-100
cloud head node in Azure CG-127
cloud queue CG-1
cloud queues CG-14
cloud_instance_type CG-12
cloud_max_instances CG-13
cloud_max_jobs_check_per_queue CG-13
cloud_min_instances CG-12
cloud_network CG-13
cloud_node_image CG-13
cloud_node_instance_type CG-13
cloud_provisioned_time CG-13
cloud_queue CG-13
cloud_scenario CG-13
cluster RG-3
adding BG-40
definition BG-17
comm RG-4
commands IG-4, RG-4, UG-2, PG-4
and provisioning UG-218
list BG-48
PATH BG-47
comment UG-178
job attribute RG-331
scheduler attribute RG-298
server attribute RG-283
vnode attribute RG-320
communication daemon RG-4

complex RG-4, BG-3
adding BG-40
Linux-Windows RG-11
mixed-mode RG-12
Windows-Linux RG-20
configrm PG-100
configuration
Budget
file BG-35
failover BG-29
file staging AG-613
parameters
basic configuration BG-101
rsync AG-613
server AG-19
tutorial BG-105
configuration file SG-2
hook HG-6
version 1 RG-20
version 2 RG-20
configuring PBS for cloud bursting CG-12
constants
in billing formula BG-41
consumable resource AG-231, RG-4
consuming credit BG-6
Corosync BG-29
count_spec UG-138
CPU AG-231, RG-4
cpuaverage AG-307
cpus_per_ssinode RG-250
cput AG-141, RG-267
cray_compute AG-483
creating
certificates for encryption BG-26
service units
dynamic BG-14
standard BG-13
creating a hook HG-5, RG-4
creating a snapshot SG-5
creating queues AG-22
creation of provisioning hooks AG-338
credential PG-21
credit
consuming BG-6
investing in groups BG-5
investing in users and projects BG-5
reconciling BG-6
cron BG-14
CSA UG-221
csaswitch AG-492, AG-654
ctime
job attribute RG-331
reservation attribute RG-304
currency BG-1, BG-13
current_aoe AG-336
vnode attribute RG-320
current_eoe AG-322, RG-320
custom resource AG-231, RG-4
custom resources
application licenses AG-274
floating managed by PBS AG-277
overview AG-257
per-host node-locked AG-279
types AG-274
how to use AG-255
scratch space
overview AG-257
static host-level AG-270
static server-level AG-268
custom resources for cloud bursting CG-14
cycle harvesting
ideal_load AG-123
max_load AG-123
cygwin UG-16

D

data service account RG-4
data service management account RG-4
data_lifetime BG-14
setting value BG-45
database
user account BG-21
date
format RG-353
datetime
format RG-354
deactivate a power profile AG-319
debug
hook attribute RG-350
debuginfo AG-663
decay AG-142
dedicated time AG-125
dedicated_prefix RG-250
default server RG-5
default_chunk
queue attribute RG-312
server attribute RG-283
default_qdel_arguments
server attribute RG-283
default_qsub_arguments
server attribute RG-283
default_queue
server attribute RG-283
defining aoe resource AG-336
defining provisioning policy AG-339
defining resources
multi-vnode machines AG-272

degraded reservation AG-198, RG-16
delegation IG-13, RG-5
delete_idle_time RG-304
deleting jobs UG-166
department AG-139
depend
  job attribute RG-332
deposit BG-16
destination
  definition RG-5
destination identifier RG-5
  format RG-354
destination queue RG-5
destination server RG-5
Deutsche Telekom CG-5
directive RG-6
directory
  staging and execution RG-19
DIS IG-59, AG-540, HG-136, RG-369
display
  non-running jobs UG-177
DNS IG-38, AG-672
do_not_span_psets AG-475
  scheduler attribute RG-298
Docker
  basic installation BG-98
  installing BG-23
  passwords BG-24
docker-ce CG-6, BG-20
docker-ee CG-6, BG-20
documentation
  PBS Professional UG-213
  SELinux UG-213
Domain Admin Account IG-13, RG-6
Domain Admins IG-13, RG-6
Domain User Account IG-13, RG-6
Domain Users IG-13, RG-6
domains
  mixed IG-17
down RG-365
downrm PG-100
dynamic fit AG-169
dynamic resource scripts/programs AG-265
dynamic service units
  definition BG-13
  updating values BG-14

E

egroup AG-139
  euser AG-139
  job attribute RG-332
element BG-2
eligible wait time AG-126
eligible_time AG-127, AG-129, AG-632, AG-638
  job attribute RG-333
eligible_time_enable
  server attribute RG-284
empty queue, node configurations
  migration under Linux IG-100, IG-115, IG-116, IG-130
enabled
  hook attribute RG-350
  queue attribute RG-312
enabling Budget BG-28
endpoint RG-6
energy AG-322, RG-267
enforcement AG-610
Enterprise Admins IG-13, RG-6
entity RG-6, BG-2
entity share RG-6
environment variables RG-399
  PATH BG-36
eoe AG-315, AG-322, RG-267
error codes RG-389
Error_Path
  job attribute RG-333
errors
  fgetfilecon UG-213
  filecon UG-213
  malloc UG-213
escrow BG-1, BG-6, BG-16, BG-91, BG-103
est_start_time_freq
  server attribute RG-284
estimated
  job attribute RG-334
etime
  job attribute RG-334
euser AG-139
  job attribute RG-334
event HG-5, RG-6
  hook attribute RG-351
events
  exechost_periodic HG-95, HG-106, HG-107
  execjob_begin HG-96, HG-98
  execjob_end HG-105
  execjob_epilogue HG-104
  execjob_preterm HG-103
  execjob_prologue HG-97
  modifyjob HG-92
  movejob HG-93
  queuejob HG-91
  resvsub HG-90
  runjob HG-94
exclhost UG-65
exclusive UG-65
exec_host AG-630
  job attribute RG-335

exec_vnode RG-267
job attribute RG-335
exechost_periodic HG-89
exechost_periodic events HG-95, HG-106, HG-107
execjob_attach HG-89
execjob_begin HG-88
execjob_begin events HG-96, HG-98
execjob_end HG-89
execjob_end events HG-104, HG-105
execjob_epilogue HG-89
execjob_launch HG-88
execjob_postsuspend HG-89
execjob_preresume HG-89
execjob_preterm HG-89
execjob_preterm events HG-103
execjob_prologue HG-88
execjob_prologue events HG-97
executable
job attribute RG-334
execution event hooks HG-6, RG-6
execution host RG-6
execution queue RG-6
Execution_Time
job attribute RG-335
executor PG-4
exit status
job arrays UG-156
Exit_status
job attribute RG-336
exiting AG-127
express_queue AG-184, RG-300
externally-provided resources RG-239

F

fail_action
hook attribute RG-352
failover RG-6, BG-29
configuring BG-29
idle RG-8
migration IG-73, IG-85, IG-97, IG-112, IG-128
primary scheduler RG-15
primary server RG-15
secondary scheduler RG-17
secondary server RG-17
failover and hooks HG-21
failure action HG-6, RG-7
fair_share RG-250
scheduler parameter AG-138
fairshare AG-138, AG-184, RG-7, RG-300
fairshare entities AG-139
fairshare ID AG-140
fairshare_decay_factor RG-251
fairshare_decay_time RG-251
fairshare_enforce_no_shares RG-251
fairshare_entity RG-251
fairshare_perc AG-151, RG-252
fairshare_usage_res RG-251
fgetfilecon error UG-213
file RG-268
.rhosts IG-12
.shosts IG-12
billing formula BG-40
Budget configuration BG-35
formula BG-40
hook configuration BG-40
hosts.equiv IG-15, IG-39
pbs.conf IG-43
services IG-59
stage in RG-18
stage out RG-18
staging UG-31
sudoers
modifications for Budget BG-36
vnodedefs RG-20
file staging RG-7
configuration AG-613
filecon error UG-213
files
MoM
config RG-381
nodes RG-380
pbs.conf AG-672
policy AG-610
location AG-610
finished jobs AG-531, RG-7
firewalld CG-128
flatuid
server attribute RG-284
flatuid server attribute AG-376
FLicenses
server attribute RG-284
float
format AG-237, RG-259, RG-359, UG-50
floating license RG-7
definition AG-231
example AG-276
example of externally-managed AG-276
floating licenses UG-54
flushreq PG-100

format
accounting log entry RG-353
attribute name RG-353
Boolean AG-237, RG-259, RG-359, UG-49
date RG-353
datetime RG-354
destination identifier RG-354
float AG-237, RG-259, RG-359, UG-50
host name RG-354
job array identifier RG-354
job array name RG-355
job array range RG-355
job identifier RG-355, RG-357
job name RG-355
limit specification RG-356
logfile-date-time RG-356
pathname RG-357
PBS NAME RG-357
PBS password RG-357
project name RG-357
queue identifier RG-357
queue name RG-357
reservation name RG-358
size AG-238, RG-260, RG-360, UG-50
string resource value AG-238, AG-243, RG-260, RG-360, UG-50
string_array AG-238, AG-244, RG-260, RG-360, UG-51
subjob identifier RG-358
username RG-358
Windows RG-358
vnode name RG-358
formats
name BG-18
formula BG-2
formula file BG-40
forward_x11_cookie
job attribute RG-336
forward_x11_port
job attribute RG-336
free RG-365, UG-65
freq
hook attribute RG-352
freq_spec UG-138
from_route_only
queue attribute RG-312
fullresp PG-100
furnishing queue RG-7

G

GCP CG-5
gethostbyaddr IG-58
gethostname AG-391
getreq PG-100
global resource AG-231, RG-7
Globus AG-19
Google Cloud Platform CG-5
group RG-7, BG-7
access AG-362, RG-8
account BG-7
ACLs AG-364
definition BG-7
ID (GID) RG-8
limit AG-231, AG-289
generic AG-289
individual AG-289
group limit RG-8
group=resource UG-65
group_list
job attribute RG-336

H

half_life RG-251
hasnodes
queue attribute RG-312
hbm_cache_pct AG-484
hbmem AG-483, RG-268
HCA AG-501, UG-201
headnode IG-21
help, getting AG-675
help_starving_jobs RG-251
here document UG-22
high-bandwidth memory AG-484
history jobs AG-531, RG-8
hold RG-8
Hold_Types
job attribute RG-336
hook RG-8
configuration file HG-6
creating HG-5, RG-4
importing HG-6, RG-8
provisioning RG-16
hook configuration file HG-6

hooks BG-1
accept RG-1
action RG-1
and failover HG-21
configuration file BG-40
creating and configuring BG-44
creating empty hooks HG-30
creation of provisioning AG-338
deleting HG-31
enabling and disabling HG-38
event types HG-15
events
exechost_periodic HG-95, HG-106, HG-107
execjob_begin HG-96, HG-98
execjob_end HG-104, HG-105
execjob_preterm HG-103
execjob_prologue HG-97
modifyjob HG-92
movejob HG-93
queuejob HG-91
resvsub HG-90
runjob HG-94
exechost_periodic events HG-95, HG-106, HG-107
execjob_begin events HG-96, HG-98
execjob_end events HG-104, HG-105
execjob_preterm events HG-103
execjob_prologue events HG-97
execution event HG-6, RG-6
exporting HG-35
importing HG-34
interface objects HG-76
job attributes HG-55
log level objects HG-152
modifyjob events HG-92
MoM HG-6
movejob events HG-93
non-job event HG-6, RG-12
overview of creating HG-30
pbs.exec_vnode HG-129
pbs.job HG-122
pbs.queue() HG-121
pbs.resv HG-131
pbs.server() HG-118
pbs.vchunk HG-131
pbs.vnode HG-133
pre-execution event HG-6, RG-15
provisioning AG-327
queuejob events HG-91
reject action RG-16
reservation attributes HG-59
resources HG-47
resvsub events HG-90
runjob events HG-94
setting order of execution HG-38
setting timeout HG-39
setting trigger events HG-31
simple how-to HG-11
vnode attributes HG-57
host RG-8, RG-268
access AG-362, RG-8
ACLs AG-364
host channel adapter AG-501, UG-201
host name
format RG-354
hostname RG-8
hosts.equiv AG-377
Hot_Start
server state RG-364
HTT RG-8
HUAWEI Cloud CG-5
HUP MoMs AG-483
hyperthreading AG-490

I

ideal_load
cycle harvesting AG-123
identifier UG-12
Idle
server state RG-364
idle (failover) RG-8
IETF IG-9, IG-58
importing a hook HG-6, RG-8
IMPS IG-139
in_multivnode_host
vnode attribute RG-320
inactive BG-2
index
subjob RG-19
indirect resource AG-231, AG-271, RG-9
ineligible_time AG-127
InfiniBand AG-466, RG-48, RG-49, UG-98, UG-99
initial_time AG-127
installation
basic BG-97
Docker BG-98
utilities BG-98
Windows MoMs IG-37
installation account IG-13, RG-9
installation script CG-9
installing
AMS BG-24
Budget BG-28
Docker BG-23
utilities BG-23
instance AG-198, RG-13, UG-135
definition BG-3
instance of a standing reservation AG-198, UG-135

instance of Budget BG-1
instance type CG-1, CG-24
instantiation AG-610
instructions
for job submitters BG-103
Intel MPI
examples UG-86
interactive
job attribute RG-337
reservation attribute RG-305
interactive job RG-9
interface objects for hooks HG-76
interval_spec UG-138
investing BG-5
in groups BG-5
in projects BG-5
in users BG-5
investor
actions BG-5
role BG-4

J

ja
CSA command UG-221
job
attribute RG-16
attributes in hooks HG-55
batch RG-9
comment UG-178
definition UG-2
dependencies UG-107
executor (MoM) PG-4
identifier RG-10, UG-12
identifier syntax UG-150
interactive RG-9
kill RG-11
owner RG-13
rerunnable RG-16
route RG-17
scheduler PG-4
shrink-to-fit RG-18
state RG-10
states RG-361
submission options UG-24
substates RG-362
job array RG-9
identifier RG-9, UG-149
range RG-9, UG-149
states UG-151
subjob RG-19
subjob index RG-19
job array identifier
format RG-354
job array name RG-10
format RG-355
job array range
format RG-355
job arrays UG-149
exit status UG-156
prologues and epilogues UG-152
job attributes
setting UG-16
job history AG-530
changing settings AG-532
configuring AG-531
enabling AG-532
setting duration AG-532
job ID RG-10
job identifier
format RG-355, RG-357
job name RG-10
format RG-355
Job Submission Description Language RG-10
job submitters
adding BG-11
instructions BG-103
user account BG-21
job that can never run AG-666
job.array_indices_submitted HG-123
job.Checkpoint HG-124
job.delete() HG-128
job.depend HG-124
job.exec_host HG-124
job.exec_vnode HG-124
job.Execution_Time HG-124
job.group_list HG-124
job.Hold_Types HG-124
job.id HG-123
job.in_ms_mom() HG-128
job.is_checkpointed() HG-127
job.job_state HG-125
job.Mail_Points HG-126
job.Mail_Users HG-126
job.rerun() HG-129
job.stagein HG-126
job.stageout HG-126
job.User_List HG-127
job_history_duration
server attribute RG-284
job_history_enable
server attribute RG-285
Job_Name
job attribute RG-337
Job_Owner
job attribute RG-337
job_priority RG-252

job_requeue_timeout
server attribute RG-285
job_sort_formula
server attribute RG-285
job_sort_formula_threshold
scheduler attribute RG-298
job_sort_key RG-252
job_state
job attribute RG-338
job-busy RG-365
jobdir
job attribute RG-337
job-exclusive RG-365
jobs
changing order UG-168
charging BG-5
deleting UG-166
moved RG-12
moving between queues UG-169
reconciling BG-6
requirements BG-20
sending messages to UG-167
sending signals to UG-168
submitting BG-103
vnode attribute RG-320
jobscript_max_size
server attribute RG-285
job-specific ASAP reservation AG-198, RG-2, RG-10, UG-135
job-specific now reservation AG-198, RG-10, RG-13, UG-135
job-specific reservation AG-198, RG-10, UG-135
Job-specific start reservation RG-10
job-specific start reservation AG-198, RG-19, UG-135
job-wide resource RG-11, UG-51, UG-52
Join_Path
job attribute RG-339
JSDL RG-10

K

Keep_Files
job attribute RG-339
kill job RG-11
kill_delay
queue attribute RG-313

L

last_state_change_time RG-320
vnode attribute AG-323
last_used_time RG-321
vnode attribute AG-323
leaf RG-11
lic_signature CG-13
license
application
floating AG-276
external RG-384
floating
definition AG-231
vnode attribute RG-321
license server RG-11
ALM RG-2
license server configuration
redundant RG-16
License Server List Configuration RG-11
license_info
vnode attribute RG-321
limit AG-232, AG-288, RG-11
attributes AG-294
cput AG-305
file size AG-305
generic group limit AG-231, AG-289, RG-7
generic project limit AG-289, RG-7
generic user limit AG-231, AG-289, RG-7
group limit AG-231, AG-289, RG-8
individual group limit AG-232, AG-289, RG-9
individual project limit AG-289, RG-9
individual user limit AG-232, AG-289, RG-9
overall AG-232, AG-289, RG-13
pcput AG-305
pmem AG-305
project RG-15
project limit AG-289
pvmem AG-305
user limit AG-232, AG-289, RG-20
walltime AG-305
limit specification
format RG-356
limits
generic and individual AG-292
group AG-287
overall limits AG-292
project AG-287
resource usage AG-287, UG-61
scope AG-290
setting limits AG-296
user AG-287
Linux-Windows complex RG-11
list of commands BG-48
load balance RG-11
load_balancing AG-121, RG-253
load_balancing_rr RG-253
local resource AG-232, RG-12
location
policy files AG-610

log events
MoM AG-548
scheduler AG-548
server AG-548
log levels AG-547
log_events
scheduler attribute RG-298
server attribute AG-548, RG-285
log_filter RG-253
logfile-date-time
format RG-356
logging
hooks log level objects HG-152
logging into Budget BG-103
logging out of Budget BG-103
logs
permissions AG-612

M

mail_from
server attribute RG-286
Mail_Points
job attribute RG-339
reservation attribute RG-305
Mail_Users
job attribute RG-339
reservation attribute RG-305
maintenance RG-365
maintenance reservation AG-198
maintenance_jobs RG-321
malloc error UG-213
Manager RG-12
privilege AG-361
manager
actions BG-5
role BG-4
managers
server attribute RG-286
managers server attribute AG-361
managing vnode AG-232, AG-271, RG-12
master provisioning script AG-327, AG-337, RG-12
master script AG-327, AG-337, RG-12
matching ACL entry AG-365
max_array_size
queue attribute RG-313
server attribute RG-286
max_concurrent_provision AG-339
server attribute RG-286
max_group_res AG-303
queue attribute RG-313
max_group_res_soft
queue attribute RG-313
max_group_run AG-303
queue attribute RG-313
max_group_run_soft AG-303
queue attribute RG-313
max_job_sequence_id RG-287
max_load
cycle harvesting AG-123
max_queuable AG-303
queue attribute RG-314
max_queued AG-295
queue attribute RG-314
max_queued_res AG-295
queue attribute RG-314
max_run AG-294
queue attribute RG-314
max_run_res AG-295
queue attribute RG-314
max_run_res_soft AG-295
queue attribute RG-315
max_run_soft AG-294
queue attribute RG-315
max_run_subjobs RG-340
max_running AG-303
queue attribute RG-315
max_starve RG-253
max_user_res AG-303
queue attribute RG-315
max_user_res_soft AG-303
queue attribute RG-315
max_user_run AG-303
queue attribute RG-316
max_user_run_soft AG-303
queue attribute RG-316
max_walltime AG-216, RG-268, UG-113
mem RG-268
mem_per_ssinode RG-253
memory
high-bandwidth AG-484
memory-only vnode AG-232, RG-12
memreserved RG-244
metadata BG-10
Microsoft Azure CG-5
migration upgrade IG-65
Linux IG-93
Windows IG-109, IG-125
min_walltime AG-216, RG-269, UG-113
minimizing power cycles AG-320
mixed domains IG-17
mixed-mode complex RG-12
modifying snapshot SG-7
modifyjob HG-88
modifyjob events HG-92

MoM IG-4, RG-12, UG-2, PG-3, PG-4
    log events AG-548
    subordinate RG-19
Mom
    vnode attribute RG-321
MoM hooks HG-6
mom_resources RG-253
monitoring RG-12, UG-1
Mother Superior RG-12
moved jobs RG-12
movejob HG-88
movejob events HG-93
moving jobs
    migration upgrade under Linux IG-107, IG-123
moving jobs between queues UG-169
MPI
    Intel MPI
        examples UG-86
    MPICH2
        examples UG-100
    MPICH-MX
        MPD
            examples UG-93
        rsh/ssh
            examples UG-94
    MVAPICH1 UG-98
        examples UG-98
MPI_USE_IB AG-466
MPICH UG-89
MPICH2
    examples UG-100
MPICH-MX
    MPD
        examples UG-93
    rsh/ssh
        examples UG-94
mpiexec AG-465
MPI-OpenMP UG-105
mpiprocs RG-269
MRJ Technology Solutions RG-411
mtime
    job attribute RG-340
    reservation attribute RG-306
multihost placement sets AG-170
multinodebusy RG-240
multi-vnode complex RG-380
MUNGE AG-379
MVAPICH1 UG-98
    examples UG-98

N

naccelerators AG-474, RG-269, UG-191
name
    vnode attribute RG-321
NASA
    and PBS RG-411
natural vnode AG-38
nchunk RG-270
NCPUS RG-399
ncpus RG-270
NEC SX-Aurora process swapping AG-504
NEC SX-Aurora TSUBASA AG-501, UG-201
NEC_PROCESS_DIST UG-204
network
    ports IG-58
    services IG-58
nhcas AG-502, UG-202
nice RG-270
no_multinode_jobs
    vnode attribute RG-322
no_stdio_sockets
    job attribute RG-340
node
    definition RG-14
node_group_key
    queue attribute RG-316
    server attribute RG-290
node_idle_limit
    server attribute AG-323
node_location CG-13
node_sort_key RG-253
nodect RG-270
nodes RG-270
    NUMA AG-484
    Xeon Phi AG-483
non-consumable resource AG-232, RG-12
non-job event hooks HG-6, RG-12
non-primetime RG-15
nonprimetime_prefix AG-195, RG-254
normal_jobs AG-184, RG-300
now reservation RG-10, RG-13
nppcu RG-13
NTFS IG-41
ntype
    vnode attribute RG-322
NUMA nodes AG-484
numa_cfg AG-484
nves AG-502, UG-202

O

object RG-13
occurrence of a standing reservation RG-13
offline RG-365
OMP_NUM_THREADS RG-399
ompthreads RG-271

only_explicit_psets
scheduler attribute RG-298
Open Telekom Cloud CG-5
OpenMP UG-103
openrm PG-100
OpenStack CG-5
Operator RG-13
privilege AG-360
operators
in billing formula BG-42
server attribute RG-291
operators server attribute AG-361
opt_backfill_fuzzy AG-109
scheduler attribute RG-299
Oracle Cloud Platform CG-5
Orange Cloud Flexible Engine CG-5
order
hook attribute RG-352
OTC CG-5
output files IG-12
output of rerunning a simulation SG-8
output of simulation SG-6
output plugin AG-318
Output_Path
job attribute RG-341
overall limit AG-232, AG-289, RG-13
overlay upgrade IG-65
backup directory IG-72, IG-73, IG-83, IG-85, IG-96
Linux IG-70
owner RG-13

P

Pacemaker BG-29
pack UG-65
Parallel Virtual Machine (PVM) UG-102
parameter RG-13
parent vnode RG-13
partial process swapping AG-504
partition RG-316, RG-322
scheduler attribute RG-299
password
invalid AG-665
passwordless ssh BG-38
PATH
for commands BG-47
setting BG-36
pathname
format RG-357
PBS RG-400
configuring for cloud bursting CG-12
pbs RG-28
PBS Administrator RG-14
PBS attribute
in billing formula BG-41
PBS complex BG-3
adding BG-40
PBS entity RG-6, RG-14
PBS environmental variables UG-151
pbs module HG-6, RG-14
PBS NAME
format RG-357
PBS object RG-13, RG-14
PBS password
format RG-357
PBS Professional RG-14, BG-19
PBS resource
in billing formula BG-41
PBS service account IG-14
pbs.acl() HG-143
pbs.args() HG-143
pbs.checkpoint( ) HG-144
pbs.conf AG-319, AG-613, AG-672
pbs.depend() HG-144
pbs.duration() HG-144
pbs.email_list() HG-144
pbs.event().accept() HG-116
pbs.event().alarm HG-109
pbs.event().hook_name HG-111, HG-112
pbs.event().hook_type HG-111
pbs.event().pid HG-112
pbs.event().reject() HG-116
pbs.event().requestor HG-113
pbs.event().requestor_host HG-113
pbs.event().type HG-113
pbs.exec_host() HG-145
pbs.exec_vnode HG-129
pbs.exec_vnode() HG-145
pbs.get_local_nodename() HG-151
pbs.group_list() HG-145
pbs.hold_types() HG-145
pbs.job HG-122
pbs.job_sort_formula() HG-145
pbs.join_path() HG-146
pbs.keep_files() HG-146
pbs.license_count() HG-146
pbs.logmsg() HG-152
pbs.mail_points() HG-146
pbs.node_group_key() HG-146
pbs.path_list() HG-146
pbs.pbs_env() HG-146
pbs.pid HG-81
pbs.place() HG-147
pbs.queue HG-121
pbs.queue() HG-121
pbs.queue.job() HG-122
pbs.range() HG-148

pbs.reboot() HG-153
pbs.resv HG-131
pbs.route_destinations() HG-148
pbs.select() HG-148
pbs.server() HG-118
pbs.server(). HG-118
pbs.server().job() HG-119
pbs.server().jobs() HG-119
pbs.server().name HG-118
pbs.server().queue() HG-120
pbs.server().queues() HG-120
pbs.server().resv() HG-120
pbs.server().resvs() HG-120
pbs.server().scheduler_restart_cycle() HG-120
pbs.server().vnode() HG-120
pbs.server().vnodes() HG-120
pbs.size() HG-150
pbs.software() HG-150
pbs.staging_list() HG-150
pbs.state_count() HG-151
pbs.user_list() HG-151
pbs.vchunk HG-131
pbs.version() HG-151
pbs.vnode HG-133
pbs_accounting_workload_mgmt AG-491, AG-653
pbs_alterjob PG-24
PBS_ARRAY_ID RG-399, UG-151
PBS_ARRAY_INDEX RG-399, UG-151
pbs_asyrunjob PG-26, PG-57
pbs_attach RG-55
pbs_auth_create_ctx PG-123
pbs_auth_decrypt_data PG-131
pbs_auth_destroy_ctx PG-125
pbs_auth_encrypt_data PG-130
pbs_auth_get_userinfo PG-126
PBS_AUTH_METHOD AG-540, HG-136, RG-369
pbs_auth_process_handshake_data PG-128
pbs_auth_set_config PG-122
PBS_BATCH_SERVICE_PORT IG-59, AG-540, HG-136, RG-369
PBS_BATCH_SERVICE_PORT_DIS IG-59, AG-540, HG-136, RG-369
pbs_comm RG-4, RG-57
PBS_COMM_LOG_EVENTS AG-540, HG-136, RG-369
PBS_COMM_ROUTERS AG-540, HG-136, RG-369
PBS_COMM_THREADS AG-540, HG-136, RG-369
PBS_CONF_FILE RG-399
PBS_CONF_SYSLOG AG-544, AG-553, HG-140, RG-373
PBS_CONF_SYSLOGSEVR AG-544, AG-553, HG-140, RG-373
pbs_connect PG-21, PG-30
PBS_CORE_LIMIT AG-540, HG-136, RG-370
PBS_DATA_SERVICE_PORT IG-59, AG-540, HG-136, RG-370
pbs_dataservice RG-60
pbs_default PG-32
pbs_deljob PG-33
pbs_delresv PG-35
pbs_disconnect PG-36
pbs_ds_password RG-61
PBS_ENCRYPT_METHOD AG-541, HG-136, RG-370
PBS_ENVIRONMENT AG-541, HG-136, RG-370, RG-399
PBS_EXEC IG-21, IG-43, AG-410, AG-541, HG-137, RG-14, RG-370
PBS_EXEC/pbs_sched_config
    overlay upgrade IG-76, IG-88, IG-101, IG-117, IG-131
PBS_EXEC/share AG-610
pbs_geterrmsg PG-37
pbs_holdjob PG-38
PBS_HOME IG-21, IG-43, AG-410, AG-541, HG-137, RG-14, RG-370
pbs_hostn RG-63, UG-4
pbs_idled RG-64
pbs_iff AG-672, UG-213, PG-21
pbs_interactive RG-67
PBS_JOBCOOKIE RG-399
PBS_JOBID RG-399, UG-151
PBS_JOBNAME RG-399
pbs_lamboot RG-68
PBS_LEAF_NAME IG-62, AG-541, HG-137, RG-370
PBS_LEAF_ROUTERS AG-541, HG-137, RG-370
pbs_license_info
    server attribute RG-291
pbs_license_linger_time
    server attribute RG-291
pbs_license_max
    server attribute RG-291
pbs_license_min
    server attribute RG-292
PBS_LOCALLOG AG-541, AG-553, HG-137, RG-370
pbs_locjob PG-39
PBS_LOG_HIGHRES_TIMESTAMP AG-541, HG-137, RG-371, RG-400
pbs_login UG-3
PBS_LR_SAVE_PATH AG-541, HG-137, RG-371
PBS_MAIL_HOST_NAME IG-62, AG-21, AG-542, HG-137, RG-371
pbs_manager PG-41
PBS_MANAGER_SERVICE_PORT IG-59, AG-542, HG-137, RG-371
pbs_mkdirs AG-665, RG-70
pbs_module PG-111
pbs_mom IG-4, RG-71, PG-3, PG-4
    starting during overlay IG-78

PBS_MOM_HOME AG-410, AG-542, HG-137, RG-371
PBS_MOM_HOST_NAME IG-62
PBS_MOM_NODE_NAME AG-477, AG-542, HG-138, RG-371
PBS_MOM_SERVICE_PORT IG-59, AG-542, HG-138, RG-371
PBS_MOMPORT RG-400
pbs_movejob PG-47
PBS_MPI_DEBUG AG-466
pbs_mpihp RG-76
pbs_mpilam RG-78
pbs_mpirun RG-79
pbs_msgjob PG-49
PBS_NODENUM RG-400
PBS_O_HOME RG-400
PBS_O_HOST RG-400
PBS_O_LANG RG-400
PBS_O_LOGNAME RG-400
PBS_O_MAIL RG-400
PBS_O_PATH RG-400
PBS_O_QUEUE RG-400
PBS_O_SHELL RG-400
PBS_O_SYSTEM RG-400
PBS_O_TZ RG-400
PBS_O_WORKDIR RG-400
pbs_orderjob PG-51
PBS_OUTPUT_HOST_NAME IG-62, AG-542, HG-138, RG-371
pbs_preempt_jobs PG-52
PBS_PRIMARY IG-62, AG-410, AG-542, HG-138, RG-372
pbs_probe IG-63, AG-664, RG-81, UG-4
pbs_python RG-83, UG-3
PBS_QUEUE RG-400
PBS_RCP AG-542, AG-613, HG-138, RG-372
pbs_rdel RG-90, UG-3
pbs_relnodesjob PG-54
PBS_REMOTE_VIEWER AG-540, HG-136, RG-370
pbs_rerunjob PG-55
pbs_rlsjob PG-56
pbs_rstat RG-94, UG-3, SG-3
pbs_rsub AG-371, RG-96, UG-3, SG-3
pbs_runjob PG-26, PG-57
pbs_sched IG-3, IG-4, RG-105, PG-2, PG-3, PG-4
PBS_SCHED_THREADS AG-543, HG-138, RG-372
PBS_SCHEDULER_SERVICE_PORT IG-59, AG-542, HG-138, RG-372
PBS_SCP AG-543, AG-613, HG-138, RG-372
PBS_SECONDARY IG-62, AG-410, AG-543, HG-139, RG-372
pbs_selectjob PG-59
pbs_selstat PG-62
PBS_SERVER IG-62, AG-410, AG-543, HG-139, RG-372, RG-400
pbs_server IG-3, IG-4, RG-108, PG-2, PG-3
PBS_SERVER_HOST_NAME IG-62, AG-543, HG-139, RG-372
PBS_SID RG-400
pbs_sigjob PG-66
pbs_sim SG-3
PBS_SNAPSHOT_PATH SG-2
PBS_START_COMM IG-159, AG-543, HG-139, RG-372
PBS_START_MOM IG-159, AG-410, AG-543, HG-139, RG-372
PBS_START_SCHED IG-159, AG-410, AG-543, HG-139, RG-372
PBS_START_SERVER IG-159, AG-410, AG-543, HG-139, RG-372
pbs_stat SG-3
pbs_statfree PG-68
pbs_stathook(3B) PG-117
pbs_stathost PG-69
pbs_statjob PG-71
pbs_statnode PG-74
pbs_statque PG-76
pbs_statresv PG-78
pbs_statrsc PG-80
pbs_statsched PG-82
pbs_statserver PG-84
pbs_statvnode PG-86
pbs_submit PG-88
pbs_submit_resv PG-90
PBS_SUPPORTED_AUTH_METHODS AG-543, HG-139, RG-373
PBS_TASKNUM RG-401
pbs_tclapi PG-104
pbs_tclsh RG-119, UG-4, PG-103
pbs_terminate PG-92
PBS_TMPDIR AG-544, HG-140, RG-373, RG-401
pbs_tmrsh RG-120
pbs_version
  scheduler attribute RG-299
  server attribute RG-292
  vnode attribute RG-322
pbs_wish RG-122, RG-124, PG-103
pbsadmin RG-14
pbs-alps-inventory-check hook AG-483
PBScrayhost AG-240, AG-474, RG-271, UG-191
PBScraylabel AG-474, UG-191
PBScraylabel_ AG-240, RG-271
PBScraynid AG-240, AG-474, RG-271, UG-191
PBScrayorder AG-240, AG-474, RG-272, UG-191
PBScrayseg RG-272
pbsdsh RG-29, UG-3
pbsfs AG-142, RG-31, UG-4, SG-3
pbshook HG-6, RG-14
pbsnodes RG-35, UG-4, SG-3

pbs-report UG-4
pbsrun RG-40
pbsrun_unwrap RG-50
pbsrun_wrap RG-51
pcap_accelerator AG-324, AG-633, AG-637, AG-639, RG-341
pcap_node AG-324, AG-633, AG-637, AG-639, RG-341
PCIe AG-501, UG-201
pcpus
  vnode attribute RG-322
pcput RG-272
pcs BG-29
peer scheduling RG-14, BG-46
per-CPU node-locked licenses UG-55
period BG-11
  defining BG-39
  definition BG-11
  hierarchy BG-11
permissions
  logs AG-612
pgov AG-324, AG-633, AG-637, AG-639, RG-341
p-governor AG-317, AG-324, RG-341
pkr CG-9, CG-138
  sample output CG-139
placement
  task AG-169
placement pool AG-169, RG-15
placement set AG-169, RG-14
placement set series RG-14
placement sets
  multihost AG-170
pmem RG-272
pnames
  vnode attribute RG-322
policy RG-15
  defining provisioning AG-339
  files AG-610
    location AG-610
  scheduling RG-17
Port
  vnode attribute RG-322
POSIX RG-15
Postgres BG-1
power cycles
  minimizing AG-320
power profile
  activate AG-319
  deactivate AG-319
power profiles AG-315
power_off_iteration
  server attribute AG-324
power_provisioning AG-322
  server attribute AG-322, AG-324, RG-292
  vnode attribute AG-324, RG-322
poweroff_eligible AG-324
  vnode attribute RG-322
PPS AG-504
precheck BG-16
preempt RG-15
preempt_order AG-180, RG-254
preempt_prio AG-181, RG-254
preempt_queue_prio AG-181, RG-254
preempt_sort AG-181, RG-254
preempt_targets RG-273
preemption AG-180
  level RG-15
  method RG-15
  target RG-15
preemption via checkpoint AG-188
preemptive scheduling AG-180
preemptive_sched AG-180, RG-254
pre-execution event hooks HG-6, RG-15
primary execution host RG-15
primary scheduler RG-15
primary server IG-62, AG-542, HG-138, RG-15, RG-372
prime_spill AG-196, RG-255
primetime RG-15
primetime_prefix AG-195, RG-254
printjob RG-125, UG-4
Priority
  job attribute RG-342
  queue attribute RG-316
  vnode attribute RG-323
privilege
  Manager AG-361
  Operator AG-360
  user AG-360
project AG-289, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641, RG-15, BG-8
  account
    definition BG-8
  attributes
    metadata BG-10
  definition BG-8
  job attribute RG-342
project limit AG-289, RG-15
  generic AG-289
  individual AG-289
project limits AG-287
project name
  format RG-357
prologue AG-319
prologues and epilogues
  job arrays UG-152
provision RG-16, UG-215
provision_enable
  vnode attribute RG-323
provision_policy AG-329, RG-255

provisioned vnode RG-16, UG-215
provisioning RG-366, UG-216
  allowing time UG-219
  and commands UG-218
  AOE restrictions UG-217
  creation of hooks AG-338
  defining policy AG-339
  hook RG-16
  hooks AG-327
  host restrictions UG-216
  master script AG-337
    writing AG-337
  overview AG-328
  policy AG-329
  rebooting AG-328
  requesting UG-218
  reservations AG-331
  using AOE UG-216
  vnode selection AG-329
  vnode states AG-332
  vnodes UG-215
provisioning tool RG-16
pset
  job attribute RG-342
pstate AG-323, RG-342
pulling queue RG-16
PVM (Parallel Virtual Machine) UG-102
pvmem RG-273
python_restart_max_hooks
  server attribute RG-292
python_restart_max_objects
  server attribute RG-292
python_restart_min_interval
  server attribute RG-292
python3 BG-20
python3-pip BG-20

Q

qalter IG-16, RG-127, UG-3
qdel AG-665, RG-140, UG-3, SG-3
qdisable RG-143, UG-4
qenable RG-145, UG-4
qhold RG-147, UG-3, UG-118
qmgr AG-19, AG-672, RG-149, RG-380, UG-4, SG-3
qmove RG-172, UG-3, UG-169
qmsg RG-174, UG-3, UG-167
qorder RG-176, UG-3, UG-168, UG-169
qrerun AG-665, RG-178, UG-4
qrls RG-180, UG-3, UG-118
qrun RG-182, UG-4
qselect RG-186, UG-3, UG-187, SG-3
qsig RG-192, UG-3
qstart RG-195, UG-4
qstat AG-672, RG-197, UG-3, UG-118, UG-166, UG-169, UG-172, UG-174, UG-176, SG-3
qstop RG-211, UG-4
qsub IG-16, AG-665, RG-213, UG-3, SG-3
qterm RG-233, UG-4
qtime
  job attribute RG-342
query_other_jobs AG-612
  server attribute RG-292
queue AG-30
  access to a AG-362, RG-1
  ACL AG-363
  attribute
    acl_group_enable AG-370
    acl_groups AG-370
    acl_host_enable AG-370
    acl_hosts AG-370
    acl_user_enable AG-370
    acl_users AG-370
  definition RG-16
  execution RG-6
  furnishing RG-7
  job attribute RG-342
  pulling RG-16
  reservation attribute RG-306
  routing RG-17
  vnode attribute RG-323
queue identifier
  format RG-357
queue name
  format RG-357
queue. HG-121
queue.job() HG-122
queue.jobs() HG-122
queue.name HG-121
queue_rank
  job attribute RG-342
queue_softlimits AG-184, RG-300
queue_type
  job attribute RG-343
  queue attribute RG-317
queued jobs AG-289
queued_jobs_threshold AG-295
  queue attribute RG-316
queued_jobs_threshold_res AG-295
  queue attribute RG-317
  server attribute RG-293
queuejob HG-87
queuejob events HG-91
queues
  creating AG-22
queuing RG-16, UG-1

quotas
example BG-14
setting BG-13

R

rcp AG-542, HG-138, RG-372
rebooting
provisioning AG-328
reconcile BG-16
reconciling
credit BG-6
jobs BG-6
recurrence rule UG-138
Red Hat Enterprise Linux IG-23
redundant license server configuration RG-16
refund BG-16
reject an action HG-6, RG-16
Release Notes
upgrade recommendations IG-65, IG-93
release_nodes_on_stageout RG-343
report UG-221
requesting provisioning UG-218
requeue RG-16
require_cred
queue attribute RG-317
require_cred_enable
queue attribute RG-317
requirements
for job submitters BG-21
for jobs BG-20
Rerunable
job attribute RG-343
rerunning a simulation SG-8
reservation AG-630
access to a AG-362, RG-1
ACL AG-363
advance AG-198, RG-2, UG-135, UG-137
ASAP AG-198, RG-2, RG-10
attribute
Authorized_Groups AG-371
Authorized_Hosts AG-371
Authorized_Users AG-371
attributes in hooks HG-59
control of creation AG-363
degradation RG-16
degraded AG-198, RG-5, UG-135
deleting UG-143
instance AG-198, RG-13, UG-135
job-specific AG-198, RG-10, UG-135
ASAP AG-198, RG-2, RG-10, UG-135
now AG-198, RG-10, RG-13, UG-135
start AG-198, RG-10, RG-19, UG-135
maintenance AG-198
now AG-198, RG-10, RG-13
occurrence RG-13
reservation ID AG-199
setting start time & duration UG-138
soonest occurrence AG-198, RG-18, UG-136
standing AG-198, RG-19, UG-136
instance AG-198, RG-13, UG-135
soonest occurrence AG-198, RG-18, UG-136
standing reservation UG-138
start RG-10
submitting jobs UG-146
reservation degradation RG-16
reservation ID RG-16
reservation identifier RG-16
reservation name
format RG-358
reservations AG-197
provisioning AG-331
time for provisioning UG-219
reserve_count
reservation attribute RG-306
reserve_duration
reservation attribute RG-306
reserve_end
reservation attribute RG-306
reserve_ID
reservation attribute RG-306
reserve_index
reservation attribute RG-307
reserve_job RG-307
Reserve_Name
reservation attribute RG-307
Reserve_Owner
reservation attribute RG-307

reserve_retry
reservation attribute RG-307
reserve_retry_cutoff
server attribute RG-293
reserve_retry_init
server attribute RG-293
reserve_retry_time
server attribute RG-293
reserve_rrule
reservation attribute RG-308
reserve_start
reservation attribute RG-308
reserve_state
reservation attribute RG-309
reserve_substate
reservation attribute RG-309
resource AG-232, RG-17
built-in AG-231, RG-3
consumable AG-231, RG-4
custom AG-231, RG-4
in billing formula BG-41
indirect AG-231, AG-271, RG-9
job-wide RG-11, UG-51, UG-52
non-consumable AG-232, RG-12
shared AG-232, AG-271, RG-18
resource limits AG-287
resource usage limits AG-287
Resource_List AG-630, AG-631, AG-633, AG-634, AG-637, AG-638, AG-639, AG-641, UG-24
job attribute RG-344
reservation attribute RG-310
Resource_List.eoe AG-322, RG-267
resource_unset_infinite RG-256
resources RG-256
in hooks HG-47
unset AG-161
resources for Budget BG-38
resources_assigned AG-639
queue attribute RG-317
server attribute RG-294
vnode attribute RG-323
resources_available
queue attribute RG-318
server attribute RG-294
vnode attribute RG-323
resources_available.eoe AG-322, RG-267
resources_default
queue attribute RG-318
server attribute RG-294
resources_max
queue attribute RG-318
server attribute RG-295
resources_min
queue attribute RG-318
resources_released RG-344
resources_released_list RG-344
resources_used
job attribute RG-344
resources_used.energy AG-319, AG-322
restart RG-17, RG-240
restart file RG-17
restart script RG-17
restrict_res_to_release_on_suspend RG-295
restrict_user AG-391
restrict_user_exceptions AG-391
restrict_user_maxsysid AG-392
restrictions
AOE UG-217
provisioning hosts UG-216
results of simulation SG-7
resv
vnode attribute RG-324
resv. HG-132
resv.resvid HG-132
RESV_BEING_DELETED RG-367
RESV_CONFIRMED RG-367
RESV_DEGRADED RG-367
RESV_DELETED RG-367
RESV_DELETING_JOBS RG-367
resv_enable AG-371
vnode attribute RG-324
resv_enable server attribute AG-363
RESV_FINISHED RG-367
RESV_IN_CONFLICT RG-367
resv_nodes UG-135
reservation attribute RG-310
RESV_NONE RG-367
resv_post_processing_time
server attribute RG-295
RESV_RUNNING RG-367
RESV_TIME_TO_RUN RG-367
RESV_UNCONFIRMED RG-367
RESV_WAIT RG-367
resv-exclusive RG-366
resvsub HG-87
resvsub events HG-90
RHEL CG-5
role BG-4
admin BG-4
investor BG-4
actions BG-5
manager BG-4
actions BG-5
teller BG-4
user BG-4
roles AG-359
round_robin RG-256
route RG-17

route queue RG-379, RG-382
route_destinations
queue attribute RG-319
route_held_jobs
queue attribute RG-319
route_lifetime
queue attribute RG-319
route_retry_time
queue attribute RG-319
route_waiting_jobs
queue attribute RG-319
routing queue RG-17
RPM
debuginfo AG-663
rpp_highwater
server attribute RG-295
rpp_max_pkt_check RG-295
rpp_retry
server attribute RG-295
rsync
configuration AG-613
run_count AG-631, AG-634, AG-641, RG-137, RG-228, UG-25, UG-119
job attribute RG-345
run_time AG-127
run_version
job attribute RG-345
runjob HG-88
runjob events HG-94
running a simulation SG-5
RUR AG-318

S

sandbox RG-228
job attribute RG-345
scatter UG-65
sched_cycle_length
scheduler attribute RG-301
sched_host
scheduler attribute RG-301
sched_log
scheduler attribute RG-301
sched_port
scheduler attribute RG-301
sched_preempt_enforce_resumption AG-182
scheduler attribute RG-301
sched_priv
scheduler attribute RG-301
schedselect
job attribute RG-345
scheduler IG-4, RG-17, UG-2, PG-3, PG-4
log events AG-548
scheduler_iteration
scheduler attribute RG-300
server attribute RG-296
Scheduling
server state RG-364
scheduling UG-1
policy RG-15, RG-17
scheduler attribute RG-300
server attribute RG-296
scheduling jobs RG-17
Schema Admins IG-14, RG-17
scp IG-12, AG-543, HG-138, RG-372
scratch CG-108
scratch space AG-257
dynamic
host-level AG-274
server-level AG-273
static
host-level AG-274
server-level AG-274
script
master provisioning AG-337
writing provisioning AG-337
secondary scheduler RG-17
secondary server IG-62, AG-543, HG-139, RG-17, RG-372
secure copy IG-12
security_context job attribute AG-610
SELinux CG-6, CG-128
sequence number RG-17, UG-149
server IG-4, RG-18, UG-2, PG-3
access to AG-362
access to the RG-1
ACL AG-363
attribute
acl_host_enable AG-370
acl_hosts AG-370
acl_user_enable AG-370
acl_users AG-370
flatuid AG-376
log_events AG-548
managers AG-361
operators AG-361
resv_enable AG-363
default RG-5
job attribute RG-346
log events AG-548
name RG-18
parameters AG-18
primary IG-62, AG-542, HG-138, RG-372
recording configuration AG-19
reservation attribute RG-310
secondary IG-62, AG-543, HG-139, RG-372

server attributes
node_idle_limit AG-323
power_off_iteration AG-324
power_provisioning AG-322
server hook HG-6
server_dyn_res RG-256
server_dyn_res_alarm RG-301
server_softlimits AG-184, RG-300
server_state
server attribute RG-297
service account
PBS IG-14
service units
abandoned BG-7
definition BG-12
dynamic
creating BG-14
updating values BG-14
usage BG-13
in billing formula BG-43
standard
creating BG-13
usage BG-13
session_id
job attribute RG-346
set_power_cap AG-324, RG-341
setting hook trigger events HG-31
setting job attributes UG-16
setting limits AG-296
share UG-65
shared resource AG-232, AG-271, RG-18
shares AG-139
sharing
vnode attribute RG-324
Shell_Path_List
job attribute RG-346
shrink-to-fit job RG-18
SIGKILL UG-168
SIGNULL UG-168
SIGTERM UG-168
SIM_LICENSE_LOCATION SG-2
simsh SG-3
Simulate configuration file SG-2
simulation
output SG-6
output of rerun SG-8
rerunning SG-8
results SG-7
running SG-5
single_signon_password_enable
server attribute RG-297
sister RG-18
sisterhood RG-18
site RG-273
definition RG-18
size
format AG-238, RG-260, RG-360, UG-50
sleep
vnode state AG-324
SLES CG-5
restrictions CG-5, BG-19
smp_cluster_dist RG-256
snapshot
checking contents SG-5
creating SG-5
snapshot checkpoint RG-18
soft_walltime RG-274
software RG-273
third-party BG-20
soonest occurrence AG-198, RG-18, UG-136
sort key AG-145
sort_priority RG-252
sort_queues RG-257
ssh IG-12
passwordless BG-38
sshd AG-605
stage
in RG-18
out RG-18
stagein UG-25
job attribute RG-346
stageout UG-25
job attribute RG-346
Stageout_status
job attribute RG-347
staging and execution directory RG-19
stale RG-366
standard service units
definition BG-13
standing reservation AG-198, RG-19, UG-136, UG-138
start reservation AG-198, RG-10, RG-19, UG-135
start_time RG-274
started
queue attribute RG-319
starting
MoM IG-166
starting Budget BG-28
starving_jobs AG-184, RG-251, RG-300
state RG-19
scheduler attribute RG-301
server
Hot_Start RG-364
Idle RG-364
Scheduling RG-364
Terminating RG-364
Terminating_Delayed RG-364
vnode attribute RG-326

state_count
queue attribute RG-319
server attribute RG-297
states
job array UG-151
vnodes and provisioning AG-332
state-unknown, down RG-366
static fit AG-169
stime
job attribute RG-347
strict ordering RG-19
strict_fifo RG-257
strict_ordering RG-257
strict_ordering and backfilling AG-225
string AG-243
string resource value
format AG-238, AG-243, RG-260, RG-360, UG-50
string_array AG-244
format AG-238, AG-244, RG-260, RG-360, UG-51
subject RG-19
subjob RG-19, UG-149
subjob identifier
format RG-358
subjob index RG-19, UG-149
Submit_arguments
job attribute RG-347
submitting a PBS job UG-11
submitting jobs BG-103
subordinate MoM RG-19
substate
job attribute RG-347
sudoers file
modifications for Budget BG-36
support team AG-675
SuSE IG-23, SG-1
sw_index
job attribute RG-347
SX-Aurora AG-501, UG-201
sync_time RG-257
syntax
identifier UG-150
syslog AG-552

T

tar file
overlay upgrade IG-73, IG-85
task RG-19
task placement AG-169, RG-19
TCL PG-103
teller BG-21
role BG-4
terminate RG-240

Terminating
server state RG-364
Terminating_Delayed
server state RG-364
third-party software BG-20
three-server configuration RG-19
throughput_mode
scheduler attribute RG-302
time between reservations UG-147
time-sharing RG-379, RG-380
tm_atnode PG-94
tm_attach PG-94
tm_finalize PG-94
tm_init PG-94
tm_kill PG-94
tm_nodeinfo PG-94
tm_notify PG-94
tm_obit PG-94
tm_poll PG-94
tm_publish PG-94
tm_rescinfo PG-94
tm_spawn PG-94
tm_subscribe PG-94
tm_taskinfo PG-94
TMPDIR RG-401
tolerate_node_failures RG-348
topjob_ineligible
job attribute RG-348
topology_info
vnode attribute RG-326
total_jobs
queue attribute RG-319
server attribute RG-297
TPP RG-20
tracejob RG-235, UG-3, SG-3
transaction
definition BG-16
transaction ID BG-17
transfer BG-16
transferring
abandoned service units BG-7
TSUBASA AG-501, UG-201
tutorial
configuring Budget BG-105
type
hook attribute RG-352
type codes AG-548

U

UID RG-20
umask
job attribute RG-348
unburst CG-1

units
in billing formula BG-42
unknown node AG-139
unknown_shares AG-138, RG-257
unset resources AG-161
until_spec UG-138
upgrade
migration IG-65
migration under Linux IG-93
migration under Windows IG-109, IG-125
overlay IG-65
upgrading
Linux IG-70
Windows IG-109, IG-125
upgrading Budget BG-32
usage limits AG-287
user
access AG-362, RG-20
account BG-10
accounts
required BG-20
ACLs AG-364
definition RG-20
hook attribute RG-352
ID RG-20
privilege AG-360
role BG-4
roles AG-359
user account
administrator BG-20
database user BG-21
job submitter BG-21
teller BG-21
user environment SG-2
user job accounting UG-221
user limit AG-232, AG-289, RG-20
generic AG-289
individual AG-289
user limits AG-287
User_List
job attribute RG-349
username
format RG-358
Windows
format RG-358
using Simulate on a snapshot SG-5
utilities
basic installation BG-98
installing BG-23

V

Variable_List
job attribute RG-349

vchunk RG-20
vchunk.chunk_resources.keys() HG-131
vchunk.vnode_name HG-131
VE AG-501, UG-201
VE offloading AG-501, UG-201
ve_cput AG-502, AG-504, UG-202, UG-209
ve_mem AG-502, AG-504, UG-202, UG-209
VE_NODE_NUMBER UG-209
vector engine AG-501, UG-201
vector host AG-501, UG-201
version 1 configuration file RG-20
version 2 configuration file RG-20
version information AG-664
VH AG-501, UG-201
virtual nodes AG-37
vmem RG-274
vnode AG-37, RG-20, RG-274
  attributes in hooks HG-57
  borrowing AG-230, AG-271, RG-3
  managing AG-232, AG-271, RG-12
  memory-only AG-232, RG-12
  natural AG-38
  selection for provisioning AG-329
  states and provisioning AG-332
vnode attributes
  last_state_change_time AG-323
  last_used_time AG-323
vnode name
  format RG-358
vnode types UG-49
vnode.topology_info HG-134
vnode_pool AG-481, RG-327
vnodedef_additive AG-475
vnodedefs file RG-20
vnodes
  provisioning UG-215
vntype RG-275
vp RG-20
VPN CG-6, BG-22
vscatter UG-65

W

waiting for job completion UG-120
wait-provisioning RG-366
walltime RG-275
Windows IG-15, IG-17, IG-23
  mixed-mode complex RG-12
  password AG-665
Windows-Linux complex RG-20
withdraw BG-16
worker
  definition BG-3
workers BG-1

wrapper script SG-3
writing provisioning script AG-337

X

X forwarding IG-63, AG-545
xauth IG-63, AG-545
Xeon Phi nodes AG-483