Altair PBS Professional 2024.1

User's Guide

You are reading the Altair PBS Professional 2024.1

User's Guide (UG)

Updated 2/27/24

Contact Us

Altair

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

Sales

hpcsales@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	Language	Telephone	e-mail
Australia		61 37 068 9972	anz-pbssupport@altair.com
Brazil	Portuguese	55 113 884 0414	br_support@altair.com
Canada	English	1 416 447 6463	
China		86 216 146 9080	pbs@altair.com.cn
France		33 (0)1 4133 0992	pbssupport@europe.altair.com
Germany		49 7031 309 9519	pbssupport@europe.altair.com
India		91 80 66 29 4500	pbs-support@india.altair.com
Japan		81 34 571 1454	pbs@altairjp.co.jp
Malaysia		60 39 212 1216	pbs-support@india.altair.com
Mexico	English	52 557 005 7890	mx-support@altair.com
Russia ###		+49 7031 6208 22	pbssupport@europe.altair.com
Singapore ###		+91 80 66 29 4500	pbs-support@india.altair.com
South Africa		27 21 140 4668	pbssupport@europe.altair.com
South America		+55 11 3884 0414	mx_support@altair.com
South Korea		82 26 105 2473	support@altair.co.kr
Sweden	English	4 646 590 2142	
United Kingdom	English	44 204 519 7852 44 204 519 4195	pbssupport@europe.altair.com
United States	English	1 248 614 2425 1 978 275 8350	
Americas	English		pbssupport@altair.com
Asia-Pacific			
Europe, Middle East, & Africa	English		pbssupport@europe.altair.com

Contents

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

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
```

Multiple Alternative Choices

When there are multiple options and you should choose one, the options 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 in order to burst jobs to the cloud.

PBS Professional Budgets Guide

How to configure Budgets 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

If you're not using the *Big Book*, make cross-references work by putting 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.

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.

Returning Output

PBS returns job output to the location you specify. See Chapter 3, "Job Input & Output Files", on page 33.

1.2.2 PBS Components and Process

PBS consists of a set of commands and system daemons/services for running jobs:

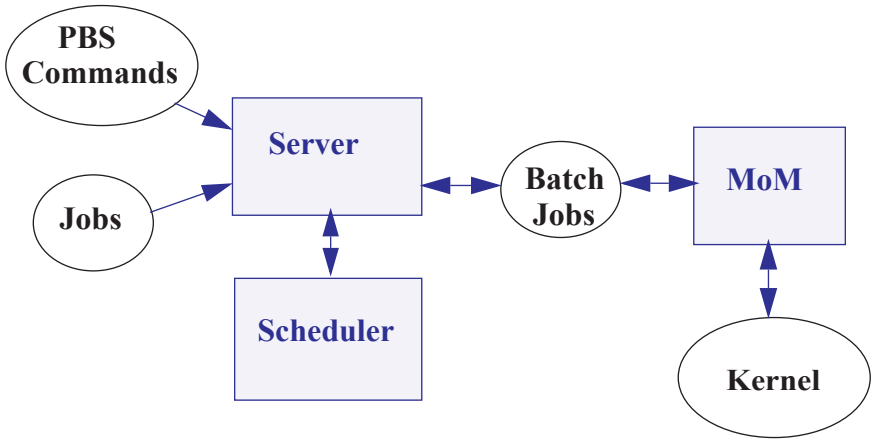

Figure 1-1: Jobs are submitted to the PBS server. The scheduler chooses where and when to run the jobs, and the server sends the jobs to MoM. PBS commands communicate with the server.

The server and scheduler 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. Communication between daemons is handled by communication 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 or command options can be run by any PBS user, while some require elevated privilege.

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

A MoM manages jobs once they are sent to its execution host. One MoM manages the jobs on each execution host. The 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. The 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.

Communication daemon

The *communication daemon*, pbs_comm, handles communication between the other PBS daemons.

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.altair.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 "List of Commands" on page 22 of the PBS Professional Reference Guide.

Table 1-1: PBS Commands

Command	Action
mpiexec	Runs MPI programs under PBS on Linux
pbsdsh	Distributes tasks to vnodes under PBS
pbsnodes	Queries PBS host or vnode status, marks hosts free or offline, changes the comment for a host, or outputs vnode information
pbs_attach	Attaches a session ID to a PBS job
pbs_hostn	Reports hostname and network address(es)
pbs_login	Caches encrypted user password for authentication
pbs_mpihp	Runs an MPI application in a PBS job with HP MPI
pbs_mpirun	Runs MPI programs under PBS with MPICH
pbs_python	Python interpreter for debugging a hook script from the command line
pbs_ralter	Modifies an existing advance, standing, or job-specific reservation
pbs_rdel	Deletes a PBS advance, standing, or job-specific reservation
pbs_release_nodes	Releases sister hosts or vnodes assigned to a PBS job
pbs_rstat	Shows status of PBS advance, standing, or job-specific reservations
pbs_rsub	Creates a PBS advance, standing, or job-specific reservation
pbs_tclsh	**Deprecated**. TCL shell with TCL-wrapped PBS API
pbs_tmrsh	TM-enabled replacement for rsh/ssh for use by MPI implementations
pbs_wish	**Deprecated**. TK window shell with TCL-wrapped PBS API
qalter	Alters a PBS job
qdel	Deletes PBS jobs

Table 1-1: PBS Commands

Command	Action
qhold	Holds PBS batch jobs
qmgr	Administrator's command interface for managing PBS
qmove	Moves a PBS job from one queue to another
qmsg	Writes message string into one or more job output files
qorder	Swaps queue positions of two PBS jobs
qrls	Releases holds on PBS jobs
qsig	Selects specified PBS jobs
qsig	Sends signal to PBS job
qstat	Displays status of PBS jobs, queues, or servers
qsub	Submits a job to 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 9.7, "Setting File Transfer Mechanism", on page 441 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 6.
- Account must have correct user authorization to run jobs.
  - For Linux, see section 1.4.2.7, "User Authorization Under Linux", on page 6.
  - For Windows, see section 1.4.3.4, "User Authorization under Windows", on page 7

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 129.

1.4.2.7 User Authorization Under Linux

The server's flatuid attribute determines whether it assumes that identical usernames mean identical users. If *True*, it assumes that if User1 exists on both the submission host and the server host, then User1 can run jobs on that server. If not *True*, the server calls `ruserok()` which uses `/etc/hosts.equiv` or `.rhosts` to authorize User1 to run as User1. 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 User1 at the server must have a `.rhosts` file listing User1.

Table 1-2: Linux User ID and flatuid

Value of flatuid	Submission Host Username vs. Server Host Username	
	User1 Same as User1	User1 Different from UserA
True	Server assumes user has permission to run job	Server checks whether User1 can run job as UserA
False/unset	Server checks whether User1 can run job as User1	Server checks whether User1 can run job as UserA

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

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 19.

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, if specified, 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 temporary scratch 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

- 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.

12. The job is run as you on the primary execution host.
13. The job's associated tasks are run as you on the execution host(s).
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

For more detail about the lifecycle of a job, see section 3.2.8, “Detailed Description of Job Lifecycle”, on page 39.

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 limit for the largest possible job ID defaults to the 7-digit number 9,999,999, but your administrator may have set it to a larger value. After the largest job ID has been assigned, PBS starts assigning job IDs 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. Your administrator can specify a root for the temporary directory on each execution host using the $tmpdir MoM parameter.

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 86.

2.1.5.1 Temporary Scratch Space Location Under Linux

If your administrator has not specified a temporary directory, the root of the temporary directory is `/var/tmp`. PBS sets the TMPDIR environment variable to the full path to the temporary scratch directory.

2.1.5.2 Temporary Scratch Space Location Under Windows

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`.

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 123.

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 33.

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 Caveats

- 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
  ```

- Make sure you specify Windows as the architecture when you submit a Windows job. When you create a selection statement that describes the resources your job needs, include the architecture for each chunk. We describe selection statements in Chapter 4, "Allocating Resources & Placing Jobs", on page 51. For example:

  ```
  #PBS -l select=ncpus=2:mem=1gb:arch=windows
  ```

  Note that "windows" is case-sensitive here.

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

- Make sure you specify Windows as the architecture when you submit a Windows job. When you create a selection statement that describes the resources your job needs, include the architecture for each chunk. We describe selection statements in Chapter 4, "Allocating Resources & Placing Jobs", on page 51. For example:

  ```
  #PBS -l select=ncpus=2:mem=1gb:arch=windows
  ```

  Note that "windows" is case-sensitive here.

- 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 -WError_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:arch=windows
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 17.

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:arch=windows
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. If this is a Windows job, we add ":arch=windows" to the chunk description. We will cover requesting resources in Chapter 4, "Allocating Resources & Placing Jobs", on page 51.

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 45.

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 4.

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 216 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 -WShell_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
  ```

  Note that "windows" is case-sensitive here.

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. The "arch=windows" is case-sensitive. For example:

  ```
  export PBS_AUTH_METHOD=pwd; qsub -lselect=1:arch=windows -- pbs-sleep 100
  ```

2.3.6.3 Submitting Windows and Linux Jobs from Linux Clients

You can submit both Windows and Linux jobs from a Linux client, but you do need to set your authentication method correctly for each kind of job. For example, you can submit a Linux job using MUNGE authentication, then set your authentication method to *pwd* and submit a Windows job:

```
export PBS_AUTH_METHOD=munge; qsub -lselect=1:arch=linux -- pbs-sleep 100
export PBS_AUTH_METHOD=pwd; pbs_login
qsub -lselect=1:arch=windows -- pbs-sleep 100
```

To override the value of the PBS_AUTH_METHOD configuration parameter, set the authentication method in the PBS_AUTH_METHOD environment variable. You can set this in your profile.

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 121
-C "<directive prefix>"	"Changing the Directive Prefix" on page 17
-c <interval>	"Using Checkpointing" on page 115
-e <path>	"Paths for Output and Error Files" on page 44
-f	"Running qsub in the Foreground" on page 31
-G	"Submitting Interactive GUI Jobs on Windows" on page 127
-h	"Holding and Releasing Jobs" on page 117
-I	"Running Your Job Interactively" on page 123
-J X-Y[:Z]	"Submitting a Job Array" on page 156
-j <join>	"Merging Output and Error Files" on page 45
-k <keep>	"Keeping Output and Error Files on Execution Host" on page 46
-l <resource list>	"Requesting Resources" on page 53
-M <user list>	"Setting Email Recipient List" on page 27
-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 44
-p <priority>	"Setting Priority for Your Job" on page 122
-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 120
-R <remove options>	"Avoiding Creation of stdout and/or stderr" on page 45
-S <path list>	"Specifying the Top Shell for Your Job" on page 19
-u <user list>	"Specifying Job Username" on page 28

Table 2-1: Options to the `qsub` Command

Option	Function and Page Reference
`-V`	"Exporting All Environment Variables" on page 128
`-v <variable list>`	"Exporting Specific Environment Variables" on page 128
`-W <attribute>=<value>`	"Setting Job Attributes" on page 16
`-W block=true`	"Making qsub Wait Until Job Ends" on page 122
`-W create_resv_from_job=<value>`	"Job-specific Start Reservations" on page 142
`-W depend=<list>`	"Using Job Dependencies" on page 109
`-W group_list=<list>`	"Specifying Job Group ID" on page 28
`-W release_nodes_on_stageout=<value>`	"Releasing Unneeded Vnodes from Your Job" on page 129
`-W run_count=<value>`	"Controlling Number of Times Job is Re-run" on page 121
`-W sandbox=<value>`	"Staging and Execution Directory: User Home vs. Job-specific" on page 33
`-W stagein=<list>`	"Input/Output File Staging" on page 33
`-W stageout=<list>`	"Input/Output File Staging" on page 33
`-W umask=<value>`	"Changing Linux Job umask" on page 47
`-X`	"Receiving X Output from Interactive Linux Jobs" on page 126
`-z`	"Suppressing Printing Job Identifier to stdout" on page 31
`--version`	Displays PBS version information.

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

Table 2-2: Points in Job/Reservation Lifecycle when PBS Sends Mail

Point in Lifecycle	Always Sent or Optional?
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 usernames 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>`
- After submission, via `qalter -P <project name>`; see "qalter" on page 130 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 usernames 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 usernames is stored in the User_List job attribute. The value of this attribute defaults to the username 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 at submission time by using `qsub -u <username>` or later via `qalter -u <username>` .

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 6.
  - For Windows, see section 1.4.3.4, "User Authorization under Windows", on page 7.
- 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 PBS Jobs on Cray HPE Cray System Management

Submitting a PBS job on an HPE Cray System Management system is exactly like submitting a job on a standard Linux machine.

2.7 Job Submission Caveats

2.7.1 Caveats for Mixed Linux-Windows Operation

- You cannot submit a Linux job from a Windows client
- In order to submit a Windows job, specify that the architecture is Windows. For example:

```
export PBS_AUTH_METHOD=pwd; qsub -lselect=1:arch=windows -- pbs-sleep 100
```

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

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. This directory is either your home directory or a job-specific directory created by PBS just for this job.

PBS can create temporary directories specific to each job to be used as job staging and execution directories. If each job has its own directories, you avoid filename collisions. PBS creates these either under your home directory or under some other location depending on how the execution host is configured.

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.

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 have to create a job-specific staging and execution directory?	No	Yes if not in home directory
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

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
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

Whether or not PBS creates job-specific staging and execution directories for a job is controlled by the job's sandbox attribute:

- If the job's sandbox attribute is set to *PRIVATE*, PBS creates a staging and execution directory for each job.
- If the job's sandbox attribute is set to *HOME* or is unset, PBS does not create job-specific staging and execution directories. Instead PBS uses your home directory.

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.

3.2.2.2 Where to Find the Staging and Execution Directory

PBS sets the job's jobdir attribute 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. This is a read-only attribute.

PBS sets the environment variable PBS_JOBDIR 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 3-2: 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. When set to *PRIVATE*, PBS creates job-specific directories. 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.
stagein attribute	Sets list of files or directories to be staged in. User-settable per job via `qsub -W`. Format: *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.
stageout attribute	Sets list of files or directories to be staged out. User-settable per job via `qsub -W`. Format: *execution_path@storage_host:storage_path* The *execution_path* is the path to the staging and execution directory. On stageout, *storage_path* is the path where output files will end up.
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 and execution directory is removed at job end along with its contents. If direct write for files is specified via the `-d` suboption to the `-k` argument, files are not removed. See section 3.3.5, "Keeping Output and Error Files on Execution Host", on page 46.
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, `pbs_tm` 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 path 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 `data1`. 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 need to 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, but you can avoid this by having PBS create job-specific staging and execution directories; to do so, set sandbox=*PRIVATE* for your jobs.

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 if PBS did not create a job-specific staging and execution directory. 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 temporary scratch directory.
- PBS sets the TMPDIR environment variable to the pathname of the temporary scratch 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.
- If PBS created them, 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 on the primary execution host. 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 stages files in to the primary execution host. 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, unless you specify that files should be written directly to their final destination via the `-d` sub-option to the `-k` option.

3.2.8.7.i Job-specific Staging and Execution Directories

If you set sandbox to *PRIVATE*, and you specified the `qsub -k` option, the `stdout` and `stderr` files are **not** automatically copied out of the staging and execution directory at job end; they will be deleted when the directory is automatically removed. Note that if you specified that files should be written directly to their final destination via the `-d` sub-option to the `-k` option, they are not created in the staging and execution directory in the first place.

3.2.8.7.ii User Home Directory as Staging and Execution Directory

If you set sandbox to *HOME* or left it unset, and you specified the -k option to qsub, 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 them up at the end of the job; it removes the staging and execution directory and all of its contents, 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 157.

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 48.

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 staging and execution directory, and runs the job script there. See section 3.2.2.1, “Setting the Job Staging and Execution Directory”, on page 34.

k

k {e | o | eo | oe | n}

When used with the `-e`, `-o`, `-eo`, `-oe`, and `-n` suboptions, specifies whether and which of `stdout` and `stderr` is retained in the job's execution directory. When set, this option overrides `-o <output path>` and `-e <error path>`. See section 3.3.5, “Keeping Output and Error Files on Execution Host”, on page 46.

kd {e | o | eo | oe}

When used with the `-d` suboption, specifies that output and/or error files are written directly to the final destination. Requires `e` and/or `o` suboptions. See section 3.3.6, “Writing Files Directly to Final Destination”, on page 47.

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 44.

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 44.

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 45.

The following table shows how these options control creation and copying of `stdout` and `stderr`:

Table 3-3: 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 <o and/or e>*	Final destination specified in `-o <output path>` and/or `-e <error path>`, 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 44.
- You can merge `stdout` and `stderr`: see section 3.3.4, “Merging Output and Error Files”, on page 45.
- You can prevent creation of `stdout` and/or `stderr`: see section 3.3.3, “Avoiding Creation of stdout and/or stderr”, on page 45.
- 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 46.
- 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 47.
- 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 45.

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 3-3, “How k, sandbox, o, and e Options to qsub Affect stdout and stderr,” on page 43.

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. Requires `-o <output path>` and/or `-e <error path>` options. See section 3.3.6, “Writing Files Directly to Final Destination”, on page 47.

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 <discard option>`
- Use `#PBS Keep_Files=<discard 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, if PBS created a job-specific staging and execution directory, PBS deletes the job-specific staging and execution directory, and all files in that directory. 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`. Indicate which files to write via the `e` and/or `o` suboptions.

For example, to directly write both output and error to their final destinations:

```
qsub -koed -o <output path> -e <error path> job.sh
```

To directly write output to its final destination, and let error go through normal spooling and staging:

```
qsub -kod -o <output path> job.sh
```

3.3.7 Changing Linux Job `umask`

On Linux, whenever your job stages or copies files or directories to the execution host, or writes `stdout` or `stderr` 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 value *077*. You can specify a value using the following methods:

- Use `qsub -W umask=<value>`
- Use `#PBS umask=<value>`

This does not apply when your job script creates files or directories.

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

- In Linux, because the main job shell is spawned without passing through PAM modules, any settings in `/etc/login.defs` that change login shell umasks for `login`, `ssh`, `rsh` etc. through PAM do not apply to any PBS job main task shell.
- 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 441 in the PBS Professional Administrator's Guide.

For troubleshooting file delivery, see "Troubleshooting File Transfer" on page 446 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 44.

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 321 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 227 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]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 when sorting vnodes, 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 63.

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 54 and section 4.3.3, "Requesting Resources in Chunks", on page 55.

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 65.

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 72 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 55 and section 4.3.2, “Requesting Job-wide Resources”, on page 54.

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 61.

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 216 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 61.

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 241 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 325 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 72 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

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.

4.8.4.7 Old -l nodes Syntax Incompatible with Cgroups

The cgroups hook does not transform old "-lnodes" syntax into the new select and place directives. If you need to use the old syntax on hosts managed by the cgroups hook, your site can use a queuejob hook to do that for you, or you can explicitly specify mem, vmem, and cgswap for jobs.

5 Multiprocessor Jobs

5.1 Submitting Multiprocessor Jobs

Before you read this chapter, please read Chapter 4, "Allocating Resources & Placing Jobs", on page 51.

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 69.

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 86
Intel MPI	2.0.022 3 4	See section 5.2.7, “Intel MPI 2.0.022, 3, and 4 with PBS”, on page 87
Intel MPI	4.0.3 on Linux	See section 5.2.5, “Intel MPI 4.0.3 On Linux with PBS”, on page 86
Intel MPI	4.0.3 on Windows	See section 5.2.6, “Intel MPI 4.0.3 On Windows with PBS”, on page 87
MPICH-P4 **Deprecated**.	1.2.5 1.2.6 1.2.7	See section 5.2.8, “MPICH-P4 with PBS”, on page 90
MPICH-GM **Deprecated**.		See section 5.2.9, “MPICH-GM with PBS”, on page 91
MPICH-MX **Deprecated**.		See section 5.2.10, “MPICH-MX with PBS”, on page 94
MPICH2 **Deprecated**.	1.0.3 1.0.5 1.0.7 On Linux	See section 5.2.11, “MPICH2 with PBS on Linux”, on page 96
MPICH2	1.4.1p1 on Windows	See section 5.2.12, “MPICH2 1.4.1p1 On Windows with PBS”, on page 99
MVAPICH **Deprecated**.	1.2	See section 5.2.13, “MVAPICH with PBS”, on page 99
MVAPICH2	1.8	See section 5.2.14, “MVAPICH2 with PBS”, on page 100
Open MPI	1.4.x	See section 5.2.15, “Open MPI with PBS”, on page 102
Platform MPI	8.0	See section 5.2.16, “Platform MPI with PBS”, on page 102
HPE MPI	Any	See section 5.2.17, “HPE MPI with PBS”, on page 103

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 86. 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:

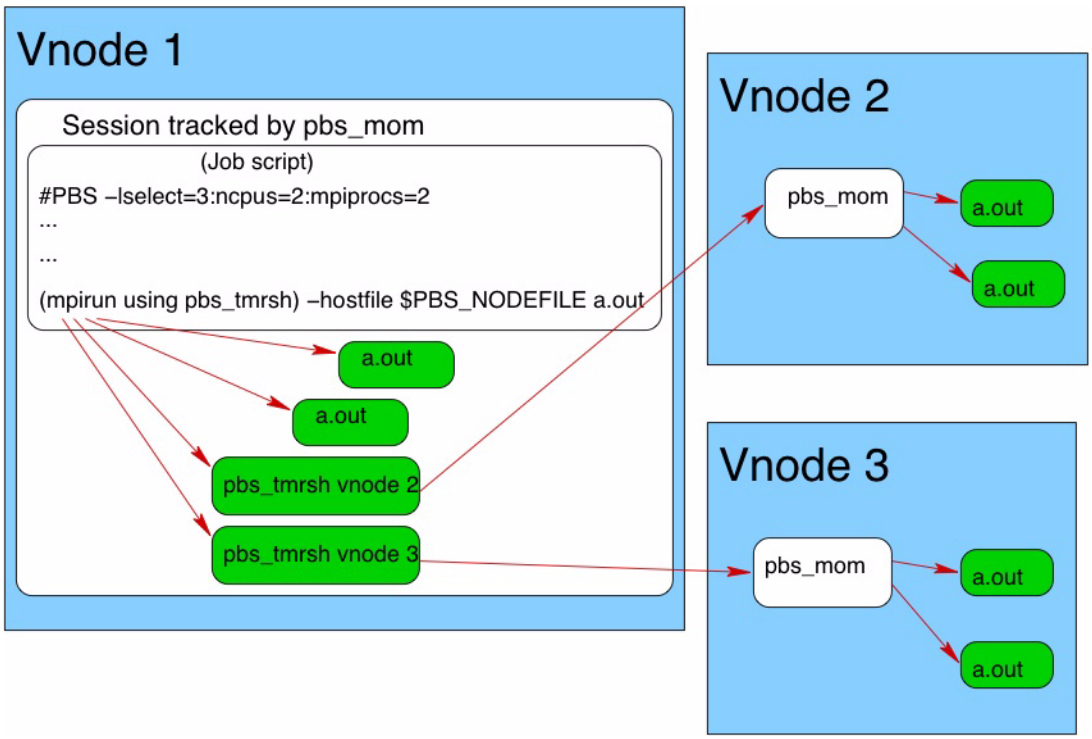

Figure 5-1:PBS knows about processes on vnodes 2 and 3, because `pbs_tmrsh` talks directly to `pbs_mom`, and `pbs_mom` starts the processes on vnodes 2 and 3

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 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 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.

5.2.8.1 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.

5.2.8.2 Example of Using MPICH-P4 with PBS

Example of using `mpirun`:

```
#PBS -l select=arch=linux
#
mpirun a.out
```

5.2.8.3 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
```

You also need to specify "arch=windows" in each chunk specification.

5.2.8.3.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.

5.2.9 MPICH-GM with PBS

5.2.9.1 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.

5.2.9.1.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.

5.2.9.1.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.

5.2.9.1.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.

5.2.9.2 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.

5.2.9.2.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.

5.2.9.2.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.

5.2.9.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-GM is the number of entries in $PBS_NODEFILE.

5.2.10 MPICH-MX with PBS

5.2.10.1 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.

5.2.10.1.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 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.

5.2.10.1.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.

5.2.10.2 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.

5.2.10.2.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.

5.2.10.2.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.

5.2.10.3 Restrictions

The maximum number of ranks that can be launched under integrated MPICH-MX is the number of entries in $PBS_NODEFILE.

5.2.11 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`".

5.2.11.1 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.

5.2.11.2 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.

5.2.11.3 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
```

5.2.11.4 Restrictions

The maximum number of ranks that can be launched under integrated MPICH2 is the number of entries in $PBS_NODEFILE.

5.2.12 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`.

5.2.13 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.

5.2.13.1 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.

5.2.13.2 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>
```

5.2.13.3 Restrictions

The maximum number of ranks that can be launched under integrated MVAPICH is the number of entries in $PBS_NODEFILE.

5.2.14 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.

5.2.14.1 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.

5.2.14.2 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.

5.2.14.3 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>
```

5.2.14.4 Restrictions

The maximum number of ranks that can be launched under MVAPICH2 is the number of entries in $PBS_NODEFILE.

5.2.15 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.

5.2.15.1 Using Open MPI with PBS

You can run jobs under PBS using Open MPI without making any changes to your MPI command line.

5.2.16 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.

5.2.16.1 Using Platform MPI with PBS

You can run jobs under PBS using Platform MPI without making any changes to your MPI command line.

5.2.16.2 Setting up Your Environment

In order to override the default `rsh`, set PBS_RSHCOMMAND in your job script:

```
export PBS_RSHCOMMAND=<rsh command>
```

5.2.17 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`.

5.2.17.1 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.

5.2.17.2 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
```

5.2.17.3 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.

5.2.17.4 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.

5.2.17.5 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 80.

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, (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 80.

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 5.

Job exit codes are listed in section 10.9, "Job Exit Status Codes", on page 470 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 160.

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 122.
- For a blocking job, the exit status is returned before staging finishes. See section 6.10.2, “Caveats for Blocking Jobs”, on page 123.
- 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>
: 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

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 109.

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 10.15, “Managing Job History”, on page 480.

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 117.

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 118
- 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 118
- 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 118

You hold a job using the qhold command; see "qhold" on page 150 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 183 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 115.

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 115.
- 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 109.

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 126. For Windows, see section 6.11.10, “Submitting Interactive GUI Jobs on Windows”, on page 127.

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.
- 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 223 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 397 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.

6.15.10 Restrictions and Caveats for Cloud Bursting with PBS

- Cloud bursting is supported only on Linux.
- Reservations are not supported on cloud nodes.

6.16 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 196 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 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` to submit a job or when you `qalter` a job to set the job's create_resv_from_job attribute to *True*.

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 216 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 -Wcreate_resv_from_job=true myscript.sh
```

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 143.

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 304 of the PBS Professional Reference Guide.

7.7 Modifying Reservations

You can use the pbs_ralter command to alter an existing reservation, whether it is an individual job-specific or 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>

You can modify 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 a reservation be deleted automatically, use pbs_ralter -Wdelete_idle_time=<allowed idle time> and specify the number of seconds as an integer, or the duration as *HH:MM:SS*. Note that you cannot change any other reservation attributes when you change this one.

You cannot change the start time of a reservation in which jobs are running.

When changing the select specification, the behavior depends on whether there are jobs running.

- If jobs are running in the reservation:
  - You cannot release chunks where reservation jobs are running
  - Vnodes where jobs are running cannot change, but everything else can
- If no jobs are running, the select specification can be changed completely

When requesting chunks, make sure each chunk request specifies chunks of a single type.

To find unused chunks in a running reservation, you can compare the reservation's resv_nodes attribute to the exec_vnode attribute of the jobs running in the reservation.

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 8.4, "Reservation Fault Tolerance", on page 401 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 85 of the PBS Professional Reference Guide.

7.7.0.0.i Examples of Modifying Reservations

Example 7-1: Grow a reservation:

Existing:

```
select=100:ncpus=20:mem=512gb
```

```
pbs_ralter -l select=150:ncpus=20:mem=512gb
```

Example 7-2: Grow and shrink a reservation:

Existing:

```
select=100:ncpus=20+10:ncpus=10:mem=512gb
```

```
pbs_ralter -l select=150:ncpus=20+5:ncpus=10:mem=512gb
```

Example 7-3: Grow a reservation, and get rid of a type of chunk:

Existing:

```
select=100:ncpus=20+10:ncpus=10:mem=512MB+15:ncpus=40
```

```
pbs_ralter -l select=150:ncpus=20+30:ncpus=40
```

Example 7-4: No running jobs; change select completely:

Existing:

```
select=100:ncpus=20+10:ncpus=10:mem=512GB
```

```
pbs_ralter -l select=150:ncpus=20:mem=1024GB+5:ncpus=15:mem=512GB
```

Example 7-5: Job is running on 50 vnodes of the first type of chunk; grow and shrink reservation:

Existing:

```
select=100:ncpus=20+50:ncpus=40
```

```
pbs_ralter -l select=50:ncpus=20+100:ncpus=40
```

Example 7-6: Negative example. With job running on 50 vnodes on the first type of chunk, we try to do an invalid alteration by trying to remove chunks from running jobs:

Existing:

```
select =100:ncpus=20+50:ncpus=40
```

```
pbs_ralter -l select=25:ncpus=20+100:ncpus=40
ALTER DENIED
```

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 59. 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 200 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 216 of the PBS Professional Reference Guide and "qmove" on page 175 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 216 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 304 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 200 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 128.

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 529 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.

8.3.4 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 363 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 need to 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 33.

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 144 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 59.

For more information, see “qalter” on page 130 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 Selecting Jobs to Examine

When you want to examine jobs, you can see them all at once, or you can select a subset. You can perform this selection via the following:

- Use the qsig command to select jobs according to your criteria and return a list of job IDs, which becomes the input to the `qstat` command; see section 10.1.1, “Selecting Jobs via qselect”, on page 175
- Use options to the qstat command to filter the jobs it will display; see section 10.1.2, “Filtering Jobs via qstat”, on page 177

10.1.1 Selecting Jobs via qselect

Use the qsig command to list the job identifiers of the jobs, job arrays or subjobs that meet your selection criteria. The command prints a list of selected jobs to standard output. You can select jobs according to name, priority, project, state, etc. In this section, we describe a few ways to select jobs.

10.1.1.1 Selecting Jobs by Resource and Attribute Value

You can select jobs where attribute and/or resource values are equal to, not equal to, greater than, greater than or equal to, less than, or less than or equal to a particular value. The default relation is "equal to", specified by ".eq.".

For example, you can list the jobs owned by barry that requested more than 16 CPUs, and discover that there are three at the default server (named "south"):

```
qselect -u barry -l ncpus.gt.16
121.south
133.south
154.south
```

10.1.1.2 Selecting Jobs by Time Criteria

You can use the `qselect -t` option to list queued, running, finished and moved jobs, job arrays, and subjobs, according to values of their time attributes. You can use the `-t` option twice to bracket a time period.

Example 10-1: Select jobs with end times between noon and 3PM:

```
qselect -te.gt.09251200 -te.lt.09251500
```

Example 10-2: Select finished and moved jobs with start times between noon and 3PM:

```
qselect -x -s "MF" -ts.gt.09251200 -ts.lt.09251500
```

Example 10-3: Select all jobs with creation times between noon and 3PM:

```
qselect -x -tc.gt.09251200 -tc.lt.09251500
```

Example 10-4: Select all jobs including finished and moved jobs with qtime of 2.30PM. Here we use the default relation of ".eq". by omitting any specification for a relation:

```
qselect -x -tq09251430
```

10.1.1.3 Selecting 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 PBS administrator sets job history preservation duration.

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.

To see a list of job IDs for finished and moved jobs:

```
qselect -H
```

10.1.1.4 Passing List of Selected Jobs to qstat

To see information about a selected list of job IDs, use the output of the qselect command as input to the qstat command. Syntax:

qstat [qstat options] `qselect [qselect options]`

For example, to see all queued and running jobs belonging to barry that requested more than 16 CPUs (in alternate format; see section 10.2.1.2, “Extended Job List: Job Status in Alternate Format”, on page 182):

- 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
--------  ----- ----- ------- ---- --- --- ---   ---- - ----
54.south  barry workq airfoil  --   --   1  --   0:13 Q   --
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 )
54.south
121.south
133.south
154.south
```

10.1.1.5 Passing List of Finished and Moved jobs to qstat

To use qstat to examine a list of finished or moved (history) jobs, make sure you tell both qstat via its -x option, and qselect via its -H option:

```
qstat -x  `qselect -H [qselect options] `
```

10.1.1.6 Restrictions and Caveats for Selecting Jobs via qselect

- Each time you call the qselect command, you can select jobs from just one server.
- You must use the backtick syntax to use the output of qselect as the input for qstat. You cannot use the pipe symbol to pipe output from qselect to qstat.

10.1.2 Filtering Jobs via qstat

By default, qstat displays information for queued or running jobs, not finished or moved jobs, and not job arrays or subjobs. However, you can tell `qstat` to display information for all jobs, whether they are running, queued, finished, or moved. Job history for finished and moved jobs is kept for a period defined by your administrator.

You can specify to `qstat` that you want information for a job identifier, a list of job identifiers, or all of the jobs at a destination, for example all jobs at a specified queue or server. You can get job information in three main formats:

- *Default format*: a basic table that lists each job ID on one line along with the username of the job owner, the state of the job, its queue, etc. See section 10.2.1.1, "Basic Job List: Job Status in Default Format", on page 181
- *Alternate format*: a more detailed table that lists each job ID on one line which also includes session ID, requested time, elapsed time, etc. See section 10.2.1.2, "Extended Job List: Job Status in Alternate Format", on page 182
- *Long format*: jobs are listed one at a time, and each job attribute and resource is listed on its own line. See section 10.2.1.3, "Complete Job Information: Job Status in Long Format", on page 183

10.1.2.1 Expanding and Filtering Job ID List

In addition to using `qselect` to select job IDs, you can use the following criteria:

- To see job arrays (not subjobs): `-J`
- To see job arrays and subjobs: `-t`
- To see only subjobs: `-Jt`
- To see finished and moved jobs in alternate format: `-H` ; see section 10.1.2.6.iii, "Restricting to Finished and Moved Jobs", on page 180
- To see finished and moved jobs in addition to running and queued jobs: `-x`; see section 10.1.2.6.ii, "Including Finished and Moved Jobs", on page 179

Formats for job IDs:

- 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.

10.1.2.2 Specifying Destination

If you don't specify a destination, you get jobs at all queues at the default server. You can specify queue and/or server. Formats for destinations:

- To display status for all jobs in the specified queue at the default server:

  <queue name>
- To display status for all jobs in the specified queue at the specified server:

  <queue name>@<server name>
- To display status for all jobs at all queues at the specified server:

  @<server name>

10.1.2.3 Filtering Jobs by User

Use the "-u" option to qstat to display jobs owned by any of a list of usernames you specify. Syntax:

qstat -u <username>[@<host>][,<username>[@<host>],...]

Host names are not required, and may be wildcarded on the left end, e.g. "**.mydomain.com*". Entering "*<username>*" without a "@*<host>*" is equivalent to "*<username>*@*".

```
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
-------- ------ ----- ------- ---- --- --- --- ---- - ----
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 my_job   --   --   1  -- 0:10 Q  --
54.south barry  workq airfoil  --   --   1  -- 0:13 Q  --
```

10.1.2.4 Looking for Running and Suspended Jobs

Use the "-r" option to qstat to display the status of all running and suspended jobs in alternate format. 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
```

10.1.2.5 Looking for Non-Running Jobs

Use the "-i" option to qstat to display the status of all non-running jobs (queued, held, and waiting) in alternate format. 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     --
```

10.1.2.6 Looking for Finished and Moved Jobs (History 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.

10.1.2.6.i Looking for Jobs Moved to Another Server

If your job is running at another server, you can examine it. If your site is using peer scheduling, your job may be moved to a server that is not your default server. For example, you submit a job to ServerA, and it returns the job ID as "123.ServerA". Then 123.ServerA is moved to ServerB.

- To see information about all jobs, whether running, queued, finished, or moved:

  ```
  qstat -x
  ```

- To see specific jobs, give the job ID as an argument to `qstat`:

  ```
  qstat 123
  ```

  or

  ```
  qstat 123.ServerA
  ```

- To list all jobs at ServerB:

  ```
  qstat @ServerB
  ```

Example 10-5: Viewing moved job:

- There are three servers with hostnames ServerA, ServerB, and ServerC
- barry submits job 123 to ServerA
- After some time, barry moves the job to ServerB
- After more time, the administrator moves the job to QueueC at ServerC
- barry runs "qstat 123"

```
Job id            Name         User    Time Use  S   Queue
----------------  -----------  -----   -------  ---  ------
123.ServerA       STDIN        barry   00:00:00  M   QueueC@ServerC
```

10.1.2.6.ii Including Finished and Moved Jobs

You can use the `-x` option to the `qstat` command to examine finished, moved, queued, and running jobs, in default format.

- To display information for queued, running, finished, and moved jobs, in default format:

  qstat -x

- To display information for a job, regardless of its state, in default format:

  qstat -x <job ID>

- To see status for jobs, job arrays and subjobs that are queued, running, finished, and moved:

  qstat -xt

- To see status for job arrays that are queued, running, finished, or moved

  qstat -xJ

When information for a moved job is displayed, the destination queue and server are shown as <queue>@<server>.

Example 10-6: Showing finished and moved jobs with queued and running jobs, and showing that job 102 was moved to destq at server2:

```
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
```

10.1.2.6.iii Restricting to Finished and Moved Jobs

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.

- To display information for finished or moved jobs, in alternate format:

  qstat -H
- To display information for a specific job in alternate format, whether or not it is finished or moved:

  qstat -H <job ID>
- To display information for finished or moved jobs at a specific destination:

  qstat -H <destination>
- To see alternate-format status for jobs, job arrays and subjobs that are finished and moved:

  qstat -Ht
- To see alternate-format status for job arrays that are finished or moved:

  qstat -HJ

Example 10-7: 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   --
```

The -H option is incompatible with the `-a`, `-i`, `-f`, and `-r` options.

10.1.2.7 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. This option is useful when you have an unordered list of job IDs and you want to see ordered results grouped by server. 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 they are 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

10.2 Examining Jobs

10.2.1 How to See Job Information (Output Formats)

You can get job information in three main formats:

- *Default format*: a basic table that lists each job ID on one line along with the username of the job owner, the state of the job, its queue, etc. See section 10.2.1.1, "Basic Job List: Job Status in Default Format", on page 181
- *Alternate format*: a more detailed table that lists each job ID on one line which also includes session ID, requested time, elapsed time, etc. See section 10.2.1.2, "Extended Job List: Job Status in Alternate Format", on page 182
- *Long format*: jobs are listed one at a time, and each job attribute and resource is listed on its own line. See section 10.2.1.3, "Complete Job Information: Job Status in Long Format", on page 183

10.2.1.1 Basic Job List: Job Status in Default Format

The default qstat output format shows you a list of jobs, one to a line. Syntax:

qstat
qstat [-E] [-J] [-p] [-t] [-w] [-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 output format 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
```

10.2.1.2 Extended Job List: Job Status in Alternate Format

The alternate `qstat` output format shows you a list of jobs, one to a line, with more detail than the basic job information. Syntax:

qstat -a

qstat [-a | -H | -i | -r] [-E] [-G | -M] [-J] [-n [-1]] [-s [-1]] [-t] [-T] [-u <user list>] [-w] [[<job ID> | <destination>] ...]

The alternate format 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.

10.2.1.3 Complete Job Information: Job Status in Long Format

The long format output of `qstat` shows you complete information about a job, including values for its attributes and resources. Syntax and example:

qstat -f
qstat -f [-F json|dsv [-D <delimiter>]] [-E] [-J] [-p] [-t] [-w] [-x] [[<job ID> | <destination>] ...]

```
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>
```

See "Job Attributes" on page 328 of the PBS Professional Reference Guide for a description of each job attribute.

10.2.1.4 Showing Additional Job Information for Default and Alternate Formats

The long format shows everything about a job, but if you want to see your jobs in a more compact format (default or alternate), you can use the following options.

10.2.1.4.i Listing Hosts Assigned to Jobs

Use the "-n" option to qstat to display the hosts allocated to any running job, in alternate format. This shows the exec_host information immediately below the job. 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  --
 --
```

10.2.1.4.ii Displaying Job Comments

The "-s" option to qstat displays the job comments, in addition to the other information presented in the alternate 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  --
    --
```

10.2.1.4.iii 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
```

10.2.1.4.iv 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.

10.2.1.4.v 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 ("?").

10.2.1.4.vi 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

10.2.1.5 Changing Output Format Characteristics

10.2.1.5.i Displaying Size in Gigabytes or Megawords

By default qstat displays size in the smallest displayable units. You can use the `-G` or `-M` options to `qstat` to display sizes in gigabytes or megawords, respectively. Both of these options trigger display in alternate format. If you specify `-G` and the actual size is less than 1GB, the output is rounded up to 1GB. A word is considered to be 8 bytes.

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    --
```

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    --
```

10.2.1.5.ii Viewing Job Status in Wider Columns

You can use the `-w qstat` option to display job status in wider columns with the default and alternate formats. 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`.

10.2.1.5.iii 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.

10.2.2 Examining Job Resource Usage

10.2.2.1 Examining Resource Usage by Running and Queued Jobs

You can see resource usage by running jobs, job arrays, and subjobs, by displaying the job in long format:

```
qstat -f <job ID>
```

10.2.2.2 Examining Resources Used by Finished and Moved Jobs

You can see the resources that finished and moved jobs and job arrays have used, but not finished or moved subjobs.

10.2.2.2.i Examining Resource Usage by Finished and Moved Jobs and Job Arrays

You can see resource usage via the long format `-f` output option to `qstat`. To see the resources used by finished and moved jobs and job arrays, use the output of the `qselect` command to filter the jobs that you list via `qstat`. Tell `qstat` to look at all jobs and give you the full output showing resources in long format:

Linux:

```
qstat -fx `qselect -H`
```

Windows:

```
for /F "usebackq" %%j in (`"\Program Files\ PBSPro\ exec\ bin\qselect" -H`)
    do ("\Program Files\PBS\exec\bin\qstat" -fx %%j)
```

10.2.2.2.ii Examining Resource Usage by Finished and Moved Subjobs

Resource usage by finished and moved subjobs is available only via the accounting logs, which are available only to root and the PBS administrator.

10.2.3 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.

10.3 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> ...]

10.3.0.1 Specifying Destination

If you don't specify a destination, you get the default server. You can specify a server. Format:

<server name>

Example 10-8: Getting status of non-default server S1:

```
qstat -B S1
Server      Max  Tot  Que  Run  Hld  Wat  Trn  Ext Status
----------- --- ---- ---- ---- ---- ---- ---- ---- ------
S1.example   0   14   13    1    0    0    0    0 Active
```

10.3.1 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 the following: 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
```

10.3.2 Viewing Server Information in Long Format

You can see server status in JSON or delimiter-separated value formats; see "Job, Queue, and Server Status Options" on page 210 of the PBS Professional Reference Guide.

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_2024.1.41640
```

10.4 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> ...]

10.4.1 Specifying Destination

If you don't specify a destination, you get jobs at all queues at the default server. You can specify queue and/or server.

- To display status for the specified queue at the default server:
  <queue name>
- To display status for the specified queue at the specified server:
  <queue name>@<server name>
- To display status for all queues at the specified server:
  @<server name>

10.4.2 Viewing Queue Information in Default Format

The "-Q" option to qstat displays the status of specified queues. 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

10.4.3 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
```

10.4.4 Viewing Queue Information in Long Format

You can see queue information in JSON or delimiter-separated value formats; see "Job, Queue, and Server Status Options" on page 210 of the PBS Professional Reference Guide.

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
```

10.4.5 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 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 Running Jobs in the Cloud

11.1 Introduction

You run a job in the cloud by putting that job in a cloud queue. You can submit the job to the queue, or move it there from another queue. Each cloud queue gives its jobs access to a specific scenario; that scenario offers specific instance types, OS images, and application licenses. Each scenario has a default instance type and OS image. Each job can request any instance type, OS image, or application license offered by the scenario. A job cannot request an instance type, OS image, or application license that is not offered by the scenario.

11.2 Running Your Job in the Cloud

Each cloud scenario is associated with a specific cloud queue and vice versa. Each scenario offers specific instance types, OS images, and application licenses. Submit your job to the cloud queue that offers the right combination. You specify the queue via the **-q <queue name>** option to **qsub**. You can override the default instance type and OS image by requesting them at job submission.

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
```

11.2.1 Requesting Instance Type

Each scenario has a default instance type, specified in the cloud_instance_type queue resource. You can choose any of the instance types offered by the queue scenario. To request an instance type, specify the instance via the cloud_node_instance_type chunk resource:

qsub -lselect=...:cloud_node_instance_type=<instance type>:... -q <queue name> <job script>

For example:

```
qsub -lselect=1:ncpus=2:mem=1gb:cloud_node_instance_type=e2-highmem-8
```

Make sure that the instance type you request matches the offered instance type exactly. Each chunk can request or default to a different instance type.

11.2.1.1 Requesting Preemptable and Spot Instances

If the scenario includes them, 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.

11.2.2 Requesting OS Image

Each scenario has a default OS image, specified in the cloud_default_image scenario parameter. You can use the default, or choose any of the OS images offered by the queue scenario. To request an OS image, specify the image via the cloud_node_image chunk resource in the select statement.

qsub -lselect=...:cloud_node_image=<OS image>:... -q <queue name> <job script>

For example:

```
qsub -lselect=1:ncpus=2:mem=1gb:cloud_node_image="myimages/image-1" myscript
```

Each chunk can use a different OS image.

11.2.3 Running Your Job on Cloud Nodes Connected by a High Speed Network

You may want to run jobs on cloud nodes where all job nodes are on the same high speed network. Cloud providers can allow PBS Cloud to burst groups of nodes where each group is connected by a high speed switch. For example, Azure provides InfiniBand *scale sets*, and Oracle provides InfiniBand *instance pools*. To simplify the discussion, we call a group of nodes on a high speed network a *proximate node group*.

When PBS Cloud bursts a group of nodes on a high speed network, it labels all of the nodes in that proximate node group with the same network name. You do not need to request the actual network name; you only need to request that your job is on such a node group via the `cloud_network=ib` chunk request. See our example below.

11.2.3.1 Running Your Job on Cloud Instances Connected by a High Speed Network and Burst on Bare Metal

Additionally, PBS lets you run jobs on cloud nodes connected by a high speed network where the instances are burst on bare metal. There is no difference between running a job on a high speed network with or without using bare metal. To burst instances on bare metal, make sure that you choose the correct instance type and matching OS (but this is true for any job using a high speed network).

In this version, Oracle is the only provider that allows you to burst instances on bare metal.

11.2.3.2 Caveats and Restrictions for Jobs on High Speed Networks

You can run your job using a high speed network only where the network is offered by the cloud provider and supported by PBS Cloud. Currently PBS Cloud supports high speed networks on Oracle and Azure.

The current default limit for the number of Azure nodes with InfiniBand is *100*.

11.2.3.3 How to Run a Job on Cloud Nodes on a High Speed Network

To run on a group of cloud nodes connected by a high speed network (a proximate node group), request `cloud_network=ib` for each job chunk. O

Make sure that each chunk in your job gets the following, and make sure they are the same across all chunks of the job:

- Instance type with high speed network enabled

  via the cloud_node_instance_type resource
- OS image enabled with a high speed network

  via the cloud_node_image resource

You can submit your job to the queue that is associated with the bursting scenario you want that has the instance type and OS image you want (for example, an Azure scenario with InfiniBand enabled). You can also request the instance type and OS image.

Here is the syntax for running your job in a proximate node group and requesting instance type and OS image:

qsub -q <cloud queue> -lselect=...:cloud_network=ib:cloud_node_instance=<instance type w/high speed network>:cloud_node_image=<OS image w/high speed network> <job script>

For example:

```
qsub -q cloudq -lselect=1:ncpus=2:mem=1gb:cloud_network=ib:cloud_node_instance_type= e2-high-mem-8:cloud_node_image="projects/images/myimage" -- /bin/sleep 60
```

Do not request the cloud_scenario resource.

11.2.4 Running Jobs Requiring Application Licenses

To run a job that needs an application license:

- Choose a scenario that offers that application license
- Request the application license

  Each application license is represented by two PBS resources; one is static, and one is dynamic. If your job requires an application license, your job script must include requests for both resources. For example, if your job requires an App1 license, represented by the resources app1_static and app1_dynamic, your job script should contain the following:

```
#PBS -l app1_static=1
#PBS -l app1_dynamic=1
```

11.3 Sample Job Scripts for Cloud Jobs

11.3.1 Example of Simple Sleep Job Script

Example 11-1: Simple job requesting 10 minutes of walltime that 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
```

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
```

11.3.2 Example of Radioss Cloud Job Script

Example 11-2: Job script for cloud job that uses 25 Radioss licenses. This script uses Intel MPI. The static resource is named "Rad_stat" and the dynamic resource is named "Rad_dyn":

```
#!/bin/bash
#PBS -N RunRad
#PBS -j oe
#PBS -m n
#PBS -q CloudRadq
#PBS -P project1
#PBS -l select=1:ncpus=16:mem=16gb
#PBS -l walltime=2:00:00
#PBS -l Rad_stat=25
#PBS -l Rad_dyn=25
/usr/local/altair/scripts/radioss -mpi i -nt $NCPUS -np 1 -hostfile $PBS_NODEFILE -both
   SEAT_DYREL_0000.rad
```

11.3.3 Viewing Job Output

When the job completes you should see the job's output. This will appear where the job was submitted.

12 Using Budgets

12.1 Budgets Commands

12.1.1 Command Path

To run Budgets commands, export the path of the `am` binaries to the PATH environment variable by using the command:

```
export PATH=$PATH:/opt/am/python/bin/
```

12.1.2 Using Budgets Commands

All Budgets commands are prefixed with "amgr ".

To see a list of Budgets 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, Budgets will prompt you to enter them.

See "Budgets Commands" on page 87 in the PBS Professional Budgets Guide.

12.2 Submitting Jobs with Budgets

Before you submit a job, you can ask Budgets to give you an estimate for the cost of the job.

Any job that is submitted must be validated by Budgets in order to be queued. Make sure that you are charging the job to a valid user or project account.

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.

Job submitters do not need to log into Budgets.

12.2.1 Getting Job Cost Estimate from Budgets

You can get Budgets to give you a quote for the cost of your job before you submit that job. It can tell you how much of each currency the job would require if you were to submit the job. You request an estimate, then PBS prints the estimated costs associated with the job. You can get cost estimates for cloud or on premise jobs.

Estimated cost may be different from actual cost because Budgets estimates the cost of each job based on the job's requested resources, but computes the final cost for a job that has run based on resources actually used.

If your administrator has configured Budgets with the `quote` command, you can use that to get a job quote. Otherwise you use the `qsub` command to get a job quote. When you use the `qsub` command to get a quote, you use normal job submission language, but the job does not actually run.

12.2.1.1 Requesting Cost Estimate via quote Command

To use the `quote` command to get an estimate for the cost of a job, you use the same job script or resource request as you would to submit the job, but you pass it to the `quote` command:

quote <job script>

quote -l <resource>=<value> -lselect=...

For example:

```
quote my_job_script.sh
quote -l walltime=1:00 -lselect=2:ncpus=4:mem=8GB --/bin/sleep 30
```

12.2.1.1.i Examples of Requesting Estimate of Costs via quote Command

Example 12-1: You submit a sleep job requesting 30 seconds of walltime and your sleep command calls for 30 seconds, and the charge rate is a penny for every CPU-second:

```
quote -l walltime=30 --/bin/sleep 30
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

If you run the job, you are charged 30 cents (one penny for each CPU-second).

Example 12-2: You submit a sleep job requesting 30 seconds of walltime but your sleep command calls for 20 seconds, and the charge rate is a penny for every CPU-second:

```
quote -l walltime=30 --/bin/sleep 20
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

If you run the job, you are charged 20 cents (one penny for each CPU-second).

12.2.1.2 Requesting Cost Estimate via qsub Command

To use the `qsub` command to get an estimate for the cost of a job, you use the same `qsub` command to submit the job, but you include "`-l am_job_quote=true`":

```
qsub ... -l am_job_quote=true
```

When you include this option, the job is not actually submitted, only evaluated.

12.2.1.3 Estimate Format

Budgets prints an estimate for the amount of each currency your job would require. Format:

qsub: Budgets estimate for job cost: {"<currency name>": <value>, "<currency name>": <value>, ...}

12.2.1.3.i Examples of Requesting Estimate of Costs via qsub Command

Example 12-3: You submit a sleep job requesting 30 seconds of walltime and your sleep command calls for 30 seconds, and the charge rate is a penny for every CPU-second:

```
qsub -l walltime=30 -l am_job_quote=t --/bin/sleep 30
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

If you run the job, you are charged 30 cents (one penny for each CPU-second).

Example 12-4: You submit a sleep job requesting 30 seconds of walltime but your sleep command calls for 20 seconds, and the charge rate is a penny for every CPU-second:

```
qsub -l walltime=30 -l am_job_quote=t --/bin/sleep 20
qsub: Budgets estimate for job cost: {"cpu_sec": 30.0, "dollar": 0.30}
```

If you run the job, you are charged 20 cents (one penny for each CPU-second).

12.2.1.4 Caveats and Restrictions for Getting Job Cost Estimate

Some nodes may have special costs associated with those nodes. Budgets does not know where the job will run, so it does not know about costs associated with specific nodes. The estimate you get from Budgets will not include those costs, if any.

12.2.2 Checking Whether You Have Enough Credit to Run Job

You can query Budgets to find out whether you have sufficient credit to run a specific job or jobs. Note that you can give the command a list of jobs to check; the command examines each job individually, without considering the other jobs in the list. So for example if you have 10 credits, and you check two jobs each requiring 10 credits, the command will tell you that you can run both. To query Budgets about jobs, use `amgr precheck jobs`. See "Prechecking Jobs" on page 135 in the PBS Professional Budgets Guide.

12.2.3 Charging Jobs to User or Project Account

When you submit a job, the job is charged to an 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
  ```

12.2.4 Credit

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.

12.2.5 Submitting Jobs in Postpaid Mode

In postpaid mode, you do not need credit to run a job. Running jobs in postpaid mode is like using a credit card, except that the amount you can charge is not limited by Budgets and you don't have to pay off the bill.

If you are in postpaid mode, you can charge jobs to your own account or any project accounts to which you belong.

When a job finishes, Budgets debits the amount of credit that was consumed by the job.

You can optionally pay down the bill, and the administrator can optionally refund you for a job.

12.2.6 Submitting Jobs in Prepaid Mode

In prepaid mode, you need credit in order to run a job.

If you are in prepaid mode, 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. Budgets does not allow your credit balance to become negative.

If you do not have sufficient credit to run a job, the job is held with a *user* hold. To allow the job to run, first acquire enough credit to run the job, then use qrls to release the hold on the job.

12.2.7 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.

12.2.8 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.

12.2.9 Allocation Periods

Your credit balance is allocated in specific time periods. Each period has a defined start and end date.

To see what periods have been defined:

```
amgr ls period
```

See "Listing Periods" on page 100 in the PBS Professional Budgets Guide.

In prepaid mode, 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.

12.2.10 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 "Checking Service Unit Balance for User" on page 131 in the PBS Professional Budgets Guide.

12.2.11 Listing Clusters

You can list the clusters that represent PBS complexes associated with Budgets:

amgr ls cluster [-n <PBS server>] [-a <active>] [-f] [-l | -j]

For example:

```
amgr ls cluster -n Cluster1 -f
```

See "Listing Clusters" on page 99 in the PBS Professional Budgets Guide.

12.2.12 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 "Listing Service Units" on page 100 in the PBS Professional Budgets Guide.

12.2.13 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 "Getting User Reports" on page 118 in the PBS Professional Budgets Guide.

You can get reports on your transactions

amgr report transaction -i <transaction ID> [-l] -N <count> [-o <output-file>] [-r]

12.3 Tutorials

12.3.1 Tutorial on Using Budgets in Prepaid Mode

12.3.1.1 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. User1 is associated with a cluster named Cluster1 and a project named P1. User1 has credit of 1200 cpu_hrs, and P1 has credit of 1000 cpu_hrs.

Substitute in your own names for the cluster, project, and users when going through the tutorial.

12.3.1.2 Tutorial Steps to Use Budgets

12.3.1.2.i Run User Job

1. Log in as User1.
2. 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
   ```

3. Check the credit balance for user User1. It will have decreased:

   ```
   amgr checkbalance user -n User1 -p 2022.Q2
   ```

4. 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 2022.Q2
   ```

12.3.1.2.ii Run Project Job

5. Run a sleep job for 36 seconds with a walltime of 1 hour, and charge it to project P1:

   ```
   qsub -P P1 -lwalltime=01:00:00 -- /bin/sleep 36
   ```

6. To see the job running:

   ```
   qstat -sw
   ```

7. Log in as Manager1.
8. Check the credit balance for project P1. It will have decreased:

   ```
   amgr checkbalance project -n P1 -p 2022.Q2
   ```

9. 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 2022.Q2
   ```

12.3.1.2.iii Non-project User Tries to Run Project Job

10. Log in as User2.
11. 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
    ```

12. Check the credit balance for user User2. It will have decreased:

    ```
    amgr checkbalance user -n User2 -p 2022.Q2
    ```

13. 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 2022.Q2
    ```

14. 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.

Example 12-5: Report for all standard service units and current lowest period:

```
amgr report project -n p1
```

12.3.1.2.iv Manager Runs Report on Project

15. Log in as Manager1:

    amgr login

16. Get report on project P1:

    amgr report project -n P1

    Command output:

```
--------------------------------------------------------------------------
name | period   | serviceunit | opening_balance | total_credits
--------------------------------------------------------------------------
P1   | 2022     | cpu_hrs     | 0.0             | 1000.0

--------------------------------------------------------------------------
| total_debits | total_debits_reconciled | total_debits_authorized
--------------------------------------------------------------------------
| 0.01         | 1.99                    | 0.0

-------------------------
| net_balance | metadata
-------------------------
| 999.99      | {}
```

12.3.2 Tutorial on Using Budgets in Postpaid Mode

12.3.2.1 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. User1 is associated with a cluster named Cluster1 and a project named P1.

Substitute in your own names for the cluster, project, and users when going through the tutorial.

12.3.2.2 Tutorial Steps to Use Budgets

12.3.2.2.i Run User Job

17. Log in as User1.

18. 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

19. After the job is finished, check the credit used by user User1. It will have increased:

    amgr checkbalance user -n User1 -p 2022

12.3.2.2.ii Run Project Job

20. Run a sleep job for 36 seconds with a walltime of 1 hour, and charge it to project P1:

```
qsub -P P1 -lwalltime=01:00:00 -- /bin/sleep 36
```

21. To see the job running:

```
qstat -sw
```

22. Log in as Manager1.
23. After the job is finished, check the credit used by project P1. It will have increased:

```
amgr checkbalance project -n P1 -p 2022
```

12.3.2.2.iii Non-project User Tries to Run Project Job

24. Log in as User2.
25. 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
```

26. After the job is finished, check the credit used by user User2. It will have increased:

```
amgr checkbalance user -n User2 -p 2022
```

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
```

This job cannot run, because User2 is not part of project P1.

12.3.2.2.iv Manager Runs Report on Project

28. Log in as Manager1:

```
amgr login
```

29. Get report on project P1:

```
amgr report project -n P1
```

Command output:

```
-------------------------------------------------------------------------
name    | period   | serviceunit   | total_outstanding   | metadata
-------------------------------------------------------------------------
P1      | 2022     | cpu_hrs       | 000.01              | {}
```

13 Submitting Jobs to 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. 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.

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 section 13.4.3.4, “Using VE Offloading”, on page 210.

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 Running Your Job on NEC SX-Aurora TSUBASA

13.4.1 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.

13.4.1.1 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.

13.4.2 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 13-1: To run 32 ve.out processes on VEs:

```
mpirun -np 32 ve.out
```

Example 13-2: 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 13-3: 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
```

13.4.2.1 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".

13.4.2.1.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 13-4: 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 ...
```

13.4.2.1.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 13-5: 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
```

13.4.3 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 13.4.3.1, "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 13.4.3.2, "Replicating Process Distribution Across VEs in a Chunk"
  - Let PBS distribute the processes for direct launch as evenly as possible, as in Section 13.4.2.1, "Letting PBS Distribute VE Processes in a Chunk"
  - Use VE offloading to indirectly launch processes on VEs, as in Section 13.4.3.4, "Using VE Offloading"
- Specify which processes should run on the vector host, as in Section 13.4.3.3, "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 13.4.3.5, "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 ("+").

13.4.3.1 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 13-6: 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
```

13.4.3.1.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.

13.4.3.2 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>

13.4.3.2.i Implying Perfect Distribution

You can use implied specification for perfect distribution via direct launch by NEC MPI over VEs.

Example 13-7: 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
```

13.4.3.2.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 13-8: 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
```

13.4.3.3 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 13-9: 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
```

13.4.3.3.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.

13.4.3.4 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 13-10: 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
```

13.4.3.4.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.

13.4.3.5 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 13-11: 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 13-12: 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
```

13.4.3.6 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 13-13: 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 13-14: 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 13-15: 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 13-13 and 13-14):

```
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 13-16: 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 13-17: 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 13-18: 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 13-19: 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
```

13.4.3.7 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 13.4.3.2.i, “Implying Perfect Distribution”, on page 209.
- 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 13.4.3.2.i, “Implying Perfect Distribution”, on page 209.
- When you are not using VE offloading, you cannot specify a count of zero processes on any VEs.

13.5 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.

13.6 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

14 Using MLS with PBS Professional

14.1 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.

14.2 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.

14.3 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.

14.3.1 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
```

14.4 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.

14.5 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.

14.6 Errors and Logging

14.6.1 Logging

PBS may log the following messages in the server and/or MoM logs:

Table 14-1: 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)

14.6.2 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"
```

14.7 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

15 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.

15.1 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

15.2 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.

15.2.1 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.

15.2.2 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.

15.2.3 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 361 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 15-1: 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*

15.3 Requirements and Restrictions

15.3.1 Host Restrictions

15.3.1.1 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.

15.3.1.2 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.

15.3.2 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
```

15.3.2.1 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

15.3.2.2 Provisioning Job Restrictions

A job that requests an AOE will not be backfilled around.

15.3.2.3 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.

15.3.3 Requirements for Jobs

15.3.3.1 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.

15.4 Using Provisioning

15.4.1 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
```

15.4.2 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 15-2: 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 15-2: 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

15.4.3 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.

15.5 Caveats and Errors

15.5.1 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.

15.5.2 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.

15.5.3 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.

15.5.4 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

15.5.5 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.

15.5.6 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.

16 Using Accounting

16.1 Using Accounting

16.1.1 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.

16.1.2 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
```

16.1.3 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 109

16.1.4 Advice and Caveats for Using Accounting

16.1.4.1 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 83.

Index