

## 7. Advice for Users

The following sections provide information necessary to the general user community concerning use of PBS. Please make this information available.

### 7.1. Modification of User shell initialization files

A user's job may not run if the user's start-up files (.cshrc, .login, or .profile) contain commands which attempt to set terminal characteristics. Any such activity should be skipped by placing a test of the environment variable **PBS\_ENVIRONMENT** (or for NQS compatibility, **ENVIRONMENT**). This can be done as shown in the following sample .login:

```
setenv PRINTER printer_1
setenv MANPATH /usr/man:/usr/local/man:/usr/new/man
if ( ! $?PBS_ENVIRONMENT ) then
    do terminal stuff here
endif
```

If the user's 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.

### 7.2. Parallel Jobs

If you have set up PBS to manage a cluster of systems or on a parallel system, it is likely with the intent to manage parallel jobs. As discussed in section **2.1 Planning** and **3.2 Multiple Execution Systems**, PBS allocated nodes to one job at a time, called space-sharing. It is important to remember that the entire node is allocated to the job regardless of the number of processors or the amount of memory in the node.

To have PBS allocate nodes to a user's job, the user must specify how many of what type of nodes are required for the job. Then the user's parallel job must execute tasks on the allocated nodes.

#### 7.2.1. How User's Request Nodes

The *nodes* resources\_list item is set by the user to declare the node requirements for the job. It is a string of the form

```
-l nodes=node_spec[+node_spec...]
```

where *node\_spec* is

```
number | property[:property...] | number:property[:property...]
```

The *node\_spec* may have an optional global modifier appended. This is of the form #property. For example:

```
6+3:fat+2:fat:hippi+disk
```

or

```
6+3:fat+2:fat:hippi+disk#prime.
```

Where *fat*, *hippi*, and *disk* are examples of property names assigned by the administrator in the {PBS\_HOME}/server\_priv/nodes file. The above example translates as the user requesting 6 plain nodes plus 3 "fat" nodes plus 2 nodes that are both "fat" and "hippi" plus one "disk" node, a total of 12 nodes. Where #prime is appended as a global modifier, the global property, "prime" is appended by the Server to each element of the spec. It would be equivalent to

```
6:prime+3:fat:prime+2:fat:hippi:prime+disk:prime .
```

A major use of the global modifier is to provide the *shared* keyword. This specifies that all the nodes are to be temporarily-shared nodes. The keyword *shared* is only recognized as such when used as a global modifier.

### 7.2.2. Parallel Jobs and Nodes

PBS provides a means by which a parallel job can spawn, monitor and control tasks on remote nodes. See the man page for `tm(3)`. *Unfortunately*, no vendor has made use of this capability though several contributed to its design. Therefore, spawning the tasks of a parallel job fall to the parallel environment itself. PVM provides one means by which a parallel job spawns processes via the `pvmd` daemon. MPI typically has a vendor dependent method, often using `rsh` or `rexec`.

All of these means are outside of PBS's control. PBS cannot control or monitor resource usage of the remote tasks, only the ones started by the job on Mother Superior. PBS can only make the list of allocated nodes available to the parallel job and hope that the vendor and the user make use of the list and stay within the allocated nodes.

The names of the allocated nodes are placed in a file in `{PBS_HOME}/aux`. The file is owned by root but world readable. The name of the file is passed to the job in the environment variable **PBS\_NODEFILE**. For IBM SP systems, it is also in the variable `MP_HOSTFILE`.

If you are running an open source version of MPI, such as MPICH, then the `mpirun` command can be modified to check for the PBS environment and use the PBS supplied host file.

### 7.3. Shell Invocation

When PBS starts a job, it invokes the user's login shell (unless the user submitted the job with the `-S` option). PBS passes the job script which is a shell script to the login in one of two ways depending on how PBS was installed.

#### Name of Script on Standard Input

The default method (PBS built with `--enable-shell-pipe`) is to pass the name of the job script to the shell program. This is equivalent to typing the script name as a command to an interactive shell. Since this is the only line passed to the script, standard input will be empty to any commands. This approach offers both advantages and disadvantages:

- + Any command which reads from standard input without redirection will get an EOF.
- + The shell syntax can vary from script to script, it does not have to match the syntax for the user's login shell. The first line of the script, even before any `#PBS` directives, should be `#!/shell` where `shell` is the full path to the shell of choice, `/bin/sh`, `/bin/csh`, ... The login shell will interpret the `#!` line and invoke that shell to process the script.
- An extra shell process is run to process the job script.
- If the script does not include a `#!` line as the first line, the wrong shell may attempt to interpret the script producing syntax errors.
- If a non-standard shell is used via the `-S` option, it will not receive the script, but its name, on its standard input.

#### Script as Standard Input

The alternative method for PBS (built with `--disable-shell-invoke`), is to open the script file as standard input for the shell. This is equivalent to typing `shell < script`. This also offers advantages and disadvantages:

- + The user's script will always be directly processed by the user's login shell.
- + If the user specifies a non-standard shell (any old program) with the `-S` option, the script can be read by that program as its input.
- If a command within the job script reads from standard input, it may read lines from the script depending on how far ahead the shell has buffered its input. Any command line so read will not be executed by the shell. A command that reads from standard input with out explicit redirection is generally

unwise in a batch job.

The choice of shell invocation methods is left to the site. It is recommended that all PBS execution servers (pbs\_mom) within that site be built to use the same shell invocation method.

#### 7.4. Job Exit Status

The exit status of a job is normally the exit status of the shell executing the job script. If a user is using *cs*h and has a *.login* file in the home directory, the exit status of *cs*h becomes the exit status of the last command in *.logout*. This may impact the use of job dependencies which depend on the job's exit status. To preserve the job's status, the user may either remove *.logout* or add the following two lines to it. Add as the first line:

```
set EXITVAL = $status
```

and as the last executable line:

```
exit $EXITVAL
```

#### 7.5. Delivery of Output Files

To transfer output files or to transfer staged-in or staged-out files to/from a remote destination, PBS uses either *rcp* or *scp* depending on the configuration options. PBS includes the source of a version of the **rcp**(1) command, from the *bsd 4.4 lite* distribution. The resulting object program, **pbs\_rcp**(1B), is used. This version of *rcp* is provided because it, unlike some *rcp* implementation, always exits with a non-zero exit status for any error. Thus Mom knows if the file was delivered or not. Fortunately, the secure copy program, *scp*, is also based on this version of *rcp* and exits with the proper status code.

Using *rcp*, the copy of output or staged files can fail for (at least) two reasons.

1. If the user's *.cshrc* script outputs any characters to standard output, e.g. contains an *echo* command, **pbs\_rcp** will fail. See the section in this document entitled **Modification of User shell initialization files**.
2. The user must have permission to *rsh* to the remote host. Output is delivered to the remote destination host with the remote file owner's name being the job owner's name (job submitter). On the execution host, the file is owned by the user's execution name which may be different. For information, see the *-u user\_list* option on the **qsub**(1) command.

If the two names are identical, permission to *rcp* may be granted at the system level by an entry in the destination host's */etc/host.equiv* file calling out the execution host.

If the owner name and the execution name are different or if the destination host's */etc/hosts.equiv* file does not contain an entry for the execution host, the user must have an ".rhosts" file in her home directory of the system to which the output files are being returned. The *.rhosts* must contain an entry for the system on which the job executed with the user name under which the job was executed. It is wise to have two lines, one with just the "base" host name and one with the full *host.domain\_name*.

If PBS is built to use the *Secure Copy Program*, *scp*, then PBS will first try to deliver output or stage-in/out files using *scp*. If *scp* fails, PBS will try again using *rcp* [assuming that *scp* might not exist on the remote host]. If *rcp* also fails, the above cycle will be repeated after a delay in case the problem is caused by a temporary network problem. All failures are logged in Mom's log.

For delivery of output files on the local host, PBS uses the **/bin/cp**(1) command. Local and remote Delivery of output may fail for the following additional reasons:

1. A directory in the specified destination path does not exist.
2. A directory in the specified destination path is not searchable by the user.

3. The target directory is not writable by the user.

Additional information as to the cause of the delivery problem might be determined from Mom's log file. Each failure is logged.

### 7.6. Stage in and Stage out problems

The same requirements and hints discussed above in regard to delivery of output apply to staging files in and out. It may also be useful to note that the stage-in and stage-out option on qsub both take the form

```
local_file@remote_host:remote_file
```

regardless of the direction of transfer. Thus for stage-in, the direction of travel is

```
local_file <-- remote_host:remote_file
```

and for stage out, the direction of travel is

```
local_file --> remote_host:remote_file
```

Also note that all relative paths are relative to the user's home directory on the respective hosts. PBS uses rcp or scp (or cp if the remote host is the local host) to perform the transfer. Hence, a stage-in is just a

```
rcp -r remote_host:remote_file local_file
```

and a stage out is just

```
rcp -r local_file remote_host:remote_file
```

As with rcp, the remote\_file may be a directory name. Also as with rcp, the local\_file specified in the stage in/out directive may name a directory. For stage-in, if remote\_file is a directory, then local file must also be a directory. For stage out, if local\_file is a directory, then remote\_file must also be a directory.

If *local\_file* on a stage out directive is a directory, that directory on the execution host, including all files and subdirectories, will be copied. At the end of the job, the directory, including all files and subdirectories, will be deleted. Users should be aware that this may create a problem if multiple jobs are using the same directory.

Stage in presents another problem. Assume the user wishes to stage-in the contents of a single file named *poo* and gives the following stage-in directive:

```
-W stagein=/tmp/bear@somehost:poo
```

If /tmp/bear is an existing directory, the local file becomes /tmp/bear/poo. When the job exits, PBS will determine that /tmp/bear is a directory and append /poo to it. Thus /tmp/bear/poo will be deleted. If however, the user wishes to stage-in the contents of a directory named *cat* and gives the following stage-in directive:

```
-W stagein=/tmp/dog/newcat@somehost:cat
```

where /tmp/dog is an existing directory, then at job end, PBS will determine that /tmp/dog/newcat is a directory and append /cat and then fail on the attempt to delete /tmp/dog/newcat/cat.

On stage-in when remote\_file is a directory, the user should not specify a new directory as local\_name. In the above case, the user should go with

```
-W stagein=/tmp/dog@somehost:cat
```

which will produce /tmp/dog/cat which will match what PBS will try to delete at job's end.

Wildcards should not be used in either the local\_file or the remote\_file name. PBS does not expand the wildcard character on the local system. If wildcards are used in the remote\_file name, since rcp is launched by rsh to the remote system, the expansion will occur. However, at job end, PBS will attempt to delete the file whose name actually contains the wildcard character and will fail to find it. This will leave all the staged in files in place (undeleted).

### 7.7. Checkpointing MPI Jobs on SGI Systems

Under Irix 6.5 and later, MPI parallel jobs as well as serial jobs can be checkpointed and restarted on SGI systems provided certain criteria are met. SGI's checkpoint system call cannot checkpoint processes that have open sockets. Therefore it is necessary to tell mpirun

to not create or to close an open socket to the array services daemon used to start the parallel processes. One of two options to mpirun must be used:

- cpr      This option directs mpirun to close its connection to the array services daemon when a checkpoint is to occur.
- miser    This option directs mpirun to directly create the parallel process rather than use the array services. This avoids opening the socket connection at all.

The -miser option appears the better choice as it avoids the socket in the first place. If the -cpr option is used, the checkpoint will work, but will be slower because the socket connection must be closed first.

Note, interactive jobs or MPMD jobs (more than one executable program) can not be checkpointed in any case. Both use sockets (and TCP/IP) to communicate, outside of the job for interactive jobs and between programs in the MPMD case.