

Table of Contents

<u>Effective Use of Resources with PBS</u>	1
<u>Streamlining PBS Job File Transfers from Pleiades to Lou</u>	1
<u>Avoiding Job Failure from Overfilling /PBS/spool</u>	2
<u>Running Multiple Serial Jobs to Reduce Wall-Time</u>	4
<u>Checking the Time Remaining in a PBS Job from a Fortran Code</u>	7

Effective Use of Resources with PBS

Streamlining PBS Job File Transfers from Pleiades to Lou

Some users prefer to streamline the storage of files (created during a job run) to Lou, within a PBS job. Since Pleiades compute nodes do not have network access to the Lou storage nodes, or other nodes outside of Pleiades, all file transfers to Lou within a PBS job must first go through one of the front-ends (pfe[20-27], or bridge[1-4]).

Here is an example of what you can add to your PBS script to accomplish this:

1. **ssh** to a bridge node (for example, bridge2) and create a directory on lou[1,2] where the files are to be copied.

```
ssh -q bridge2 "ssh -q lou mkdir -p $SAVDIR"
```

Here, \$SAVDIR is assumed to have been defined earlier in the PBS script. Note the use of **-q** for quiet-mode, and double quotes so that shell variables are expanded prior to the **ssh** command being issued.

2. Use **scp** via a bridge node to transfer the files.

```
ssh -q bridge2 "scp -q $RUNDIR/* lou:$SAVDIR"
```

Here, \$RUNDIR is assumed to have been defined earlier in the PBS script.

Avoiding Job Failure from Overfilling /PBS/spool

When your PBS job is running, its error and output files are kept in the /PBS/spool directory of the first node of your job. However, the space under /PBS/spool is limited, and when it fills up, any job that tries to write to /PBS/spool may die. This makes the node unusable by jobs until the spool directory is cleaned up manually.

To avoid this situation, PBS may start enforcing a 100-MB limit on the combined sizes of error and output files produced by a job. If this policy goes into effect and a job exceeds that limit, PBS will kill the job.

To prevent this from happening to your job, do *not* write large amounts of content in the PBS output/error files. If your executable normally writes a lot of messages to either standard out or standard error, you should redirect them in your PBS script. Below are a few options to consider:

1. Redirect standard out and standard error to a single file:

```
(for csh)
mpiexec a.out >& output
(for bash)
mpiexec a.out > output 2>&1
```

2. Redirect standard out and standard error to separate files:

```
(for csh)
(mpiexec a.out > output) > error
(for bash)
mpiexec a.out > output 2> error
```

3. Redirect only standard out to a file:

```
(for both csh and bash)
mpiexec a.out > output
```

The files "output" and "error" are created under your own directory and you can view the contents of these files while your job is still running.

If you are concerned that these two files could get clobbered in a second run of the script, you can create unique filenames for each run. For example, you can add the PBS JOBID to "output" using the following:

```
(for csh)
mpiexec a.out >& output.$PBS_JOBID
(for bash)
mpiexec a.out > output.$PBS_JOBID 2>&1
```

where \$PBS_JOBID contains a number (jobid) and the name of the PBS server, such as

12345.pbspl1.nas.nasa.gov.

If you just want to include the numeric part of the PBS JOBID, do the following:

```
(for csh)
set jobid=`echo $PBS_JOBID | awk -F . '{print $1}'`
mpiexec a.out >& output.$jobid
(for bash)
export jobid=`echo $PBS_JOBID | awk -F . '{print $1}'`
mpiexec a.out > output.$jobid 2>&1
```

In the event that you do not redirect your executable's standard out and error to a file, you can see the contents of your PBS output/error files before your job completes by following the two steps below:

1. Find out the first node of your PBS job using "-W o=+rank0" for qstat:

```
%qstat -u your_username -W o=+rank0
JobID          User      Queue   Jobname   TSK Nds    wallt S    wallt  Eff Rank0
-----
868819.pbspl1 zsmith   long    ABC       512  64  5d+00:00 R 3d+08:39 100% r162i0n14
```

This shows that the first node is r162i0n14.

2. Log in to the first node and *cd* to /PBS/spool to find your PBS stderr/out file(s). You can view the contents of these files using *vi* or *view*.

```
%ssh r162i0n14
%cd /PBS/spool
%ls -lrt
-rw----- 1 zsmith a0800 49224236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.OU
-rw----- 1 zsmith a0800 1234236 Aug  2 19:33 868819.pbspl1.nas.nasa.gov.ER
```

Running Multiple Serial Jobs to Reduce Wall-Time

On Pleiades, running multiple serial jobs within a single batch job can be accomplished with following example PBS scripts. The maximum number of processes you can run on a single node will be limited to the core-count-per-node or the maximum number that will fit in a given node's memory, whichever is smaller.

Processor Types	Cores/node	Available Memory/node
Harpertown	8	7.6 GB
Nehalem-EP	8	22.5 GB
Westmere	12	22.5 GB
Sandy Bridge	16	~31.0 GB

The examples below allow you to spawn serial jobs across nodes using the **mpiexec** command. Note that a special version of **mpiexec** from the `mpi-mvapich2/1.4.1/intel` module is needed in order for this to work. This **mpiexec** keeps track of `$PBS_NODEFILE` and places each serial job onto the CPUs listed in `$PBS_NODEFILE` properly. The use of the arguments `-comm none` for this version of **mpiexec** is essential for serial codes or scripts. In addition, to launch multiple copies of the serial job at once, the use of the **mpiexec**-supplied `$MPIEXEC_RANK` environment variable is needed to distinguish different input/output files for each serial job. This is demonstrated with the use of a wrapper script **wrapper.csh** in which the input/output identifier (that is, `{rank}`) is calculated from the sum of `$MPIEXEC_RANK` and an argument provided as input by the user.

Example 1

This first example runs 64 copies of a serial job, assuming that 4 copies will fit in the available memory on one node and 16 nodes are used.

serial1.pbs

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=16:ncpus=4
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

mpiexec -comm none -np 64 wrapper.csh 0
```

wrapper.csh

```
#!/bin/csh -f
@ rank = $1 + $MPIEXEC_RANK
./a.out < input_${rank}.dat > output_${rank}.out
```

This example assumes that input files are named *input_0.dat*, *input_1.dat*, ... and that they are all located in the directory where the PBS script is submitted from (that is, `$PBS_O_WORKDIR`). If the input files are in different directories, then `wrapper.csh` can be modified appropriately to `cd` into different directories as long as the directory names are differentiated by a single number that can be obtained from `$MPIEXEC_RANK` (=0, 1, 2, 3, ...). In addition, be sure that `wrapper.csh` is executable by you, and you have the current directory included in your path.

Example 2

A second example provides the flexibility where the total number of serial jobs may not be the same as the total number of processors requested in a PBS job. Thus, the serial jobs are divided into a few batches and the batches are processed sequentially. Again, the wrapper script is used where multiple versions of the program `a.out` in a batch are run in parallel.

serial2.pbs

```
#PBS -S /bin/csh
#PBS -j oe
#PBS -l select=10:ncpus=3
#PBS -l walltime=4:00:00

module load mpi-mvapich2/1.4.1/intel

cd $PBS_O_WORKDIR

# This will start up 30 serial jobs 3 per node at a time.
# There are 64 jobs to be run total, only 30 at a time.

# The number to run in total defaults here to 64 or the value
# of PROCESS_COUNT that is passed in via the qsub line like:
# qsub -v PROCESS_COUNT=48 serial2.pbs
#

# The total number to run at once is automatically determined
# at runtime by the number of CPUs available.
# qsub -v PROCESS_COUNT=48 -l select=4:ncpus=3 serial2.pbs
# would make this 12 per pass not 30. No changes to script needed.

if ( $?PROCESS_COUNT ) then
  set total_runs=$PROCESS_COUNT
else
  set total_runs=64
endif
```

```
set batch_count=`wc -l < $PBS_NODEFILE`

set count=0

while ($count < $total_runs)
  @ rank_base = $count
  @ count += $batch_count
  @ remain = $total_runs - $count
  if ($remain < 0) then
    @ run_count = $total_runs % $batch_count
  else
    @ run_count = $batch_count
  endif
  mpiexec -comm none -np $run_count wrapper.csh $rank_base
end
```

Checking the Time Remaining in a PBS Job from a Fortran Code

During job execution, sometimes it is useful to find out the amount of time remaining for your PBS job. This allows you to decide if you want to gracefully dump restart files and exit before PBS kills the job.

If you have an MPI code, you can call `MPI_WTIME` and see if the elapsed walltime has exceeded some threshold to decide if the code should go into the shutdown phase.

For example:

```
include "mpif.h"

real (kind=8) :: begin_time, end_time

begin_time=MPI_WTIME()
do work
end_time = MPI_WTIME()

if (end_time - begin_time > XXXXX) then
  go to shutdown
endif
```

In addition, the following library has been made available on Pleiades for the same purpose:

`/u/scicon/tools/lib/pbs_time_left.a`

To use this library in your Fortran code, you need to:

1. Modify your Fortran code to define an external subroutine and an integer*8 variable

```
external pbs_time_left
integer*8 seconds_left
```
2. Call the subroutine in the relevant code segment where you want the check to be performed

```
call pbs_time_left(seconds_left)
print*, "Seconds remaining in PBS job:",seconds_left
```

Note: The return value from **`pbs_time_left`** is only accurate to within a minute or two.

3. Compile your modified code and link with the above library using, for example:

```
LDFLAGS=/u/scicon/tools/lib/pbs_time_left.a
```