Notes: Parallel Computing

Topic: Running Jobs on a Parallel Computer

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Posted: 01/25/16
Last Update: 01/25/16
Table of Contents

1 Cluster Architecture
   - Hardware
   - Cluster Networks

2 Distributed Resource Management Systems
   - DRMS Overview
   - Portable Batch Scheduler (PBS/TORQUE)
   - MPI Execution Wrapper: mpirun

3 PBS Batch Script: Setting Attributes
   - Oversubscribing Nodes

4 Obtaining Cluster Configuration Information

5 Running Serial Jobs
   - Running Serial Code: From the Command Line
   - Running Serial Jobs Using Batch Queue
   - Printing Out PBS Environment and Job Information
   - Passing Command Line Arguments to a Batch Script
Overview of an HPC Cluster

A Cluster has multiple, separate nodes, each has multiple cores

Figure: Diagram of a cluster

Source: https://www.hpc2n.umu.se/support/beginners_guide
Intel Xeon X5650 system with six cores

- Intel Xeon X5650 system contains six CPUs (Xeon 5650)
- QPI-PCIe bridge.
- PCI-e switch for GPUs. (Peripheral Component Interconnect)
CPU/GPU System:
2 Intel Xeon X5650 and 8 Nvidia GPU Teslas

- Intel Xeon X5650 system
- "Core12": two six-core CPUs (Xeon 5650)
- eight GPUs
- Tylersburg-36D, QPI-PCIe bridge
- PXE8647 PCI-e switch for GPU pairs.

Source: http://hothardware.com/Articles/NVIDIA-GF100-Architecture-and-Feature-Preview
High Performance Network Architecture

http://www.math.umbc.edu/~gobbert/kali/binaries/networkXY.jpg
Myrinet Networks

Myrinet Network Architecture

- Memory
- CPU
- CPU

PCI Bus

Memory Bus

NIC

DMA

Memory

DMA

PCI Bridge

PCI bus to network ratio ≈ 1:1
memcpy() unacceptable

Mem to PCI bus ratio ≈ 10:1
memcpy() OK

Network

July 2, 2001
Sandia Summer Student Seminar Series

Source:
HPC Software Stacks Provide Environment to Run & Operate the Cluster System

Essential Software and Management Tools Needed to Build a Powerful, Flexible, and Highly Available Supercomputer

Cray HPC Software Stack

Source: http://www.hpcwire.com/2014/02/24/comprehensive-flexible-software-stack-hpc-clusters/
Distributed Resource Management Systems (D-RMS)

- **Primary Function:**
  - To control the usage of HPC hard resources: CPU cycles, memory, disk space and network bandwidth.
  - To optimize utilization of resources, maximize system throughput.
  - To orchestrate the process of assigning hard resources to user jobs: Users request resources by submitting jobs (serial or parallel).

- **To gain access to one or more nodes in the cluster**

- **Main components**
  - Job Management subsystem (JMS)
  - Physical Resource Manager
  - Scheduler and Queuing Systems

top - 16:38:31 up 5 days, 8:05, 7 users, load average: 0.28, 0.31, 0.20
Tasks: 176 total, 2 running, 174 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.2%us, 0.8%sy, 0.0%ni, 73.5%id, 1.4%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 12188132k total, 4513528k used, 7674604k free, 29736k buffers
Swap: 33409020k total, 21692k used, 33387328k free, 1665928k cached

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>16744</td>
<td>escalon</td>
<td>20</td>
<td>0</td>
<td>4664m</td>
<td>2.4g</td>
<td>896</td>
<td>R 99.7</td>
<td>20.9</td>
<td>0:11.82</td>
<td>histogram_mod</td>
</tr>
<tr>
<td>15234</td>
<td>hirakaw</td>
<td>20</td>
<td>0</td>
<td>105m</td>
<td>1812</td>
<td>1440</td>
<td>S  0.3</td>
<td>0.0</td>
<td>0:00.15</td>
<td>bash</td>
</tr>
<tr>
<td>16721</td>
<td>mthomas</td>
<td>20</td>
<td>0</td>
<td>15040</td>
<td>1292</td>
<td>940</td>
<td>R  0.3</td>
<td>0.0</td>
<td>0:00.05</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>19364</td>
<td>696</td>
<td>480</td>
<td>S  0.0</td>
<td>0.0</td>
<td>0:02.61</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S  0.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>kthreadd</td>
</tr>
</tbody>
</table>
D-RMS: Some Commonly Used Systems

- **Single parallel systems:**
  - **Portable Batch Systems (PBS)/TORQUE:**
    - [http://www.adaptivecomputing.com](http://www.adaptivecomputing.com)
  - **Simple Linux Utility for Resource Management (SLURM, SL:**
    - [https://computing.llnl.gov/linux/slurm/](https://computing.llnl.gov/linux/slurm/)
  - **IBM Load Lever**
  - **Platform Load Sharing Facility (LSF)**

- **Multiple, distributed, parallel systems:**
  - **Sun Grid Engine (SGE):**
    - [http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html](http://star.mit.edu/cluster/docs/0.93.3/guides/sge.html)
  - **HTCondor: High Throughput computing**
    - [http://research.cs.wisc.edu/htcondor/](http://research.cs.wisc.edu/htcondor/)

D-RMS: Job Management subsystem (JMS)

- Interface between users and RMS
- Different types of jobs
  - simple scripts
  - command line/interactive
  - Apps using MPI, OpenMP, CUDA, parallel libraries
  - job arrays/multi-task
  - workflow/complex dependent jobs
- Manages jobs:
  - Submission: name, type, I/O, parallel environment
  - Resource requirements: #cores, wall-clock time, memory, disk space, network
  - Control: queuing/scheduling, deleting, status/checking, suspension/resume, checkpointing
  - Monitoring, History
  - Accounting
D-RMS: Physical Resource Manager

- **Static Resource Info:**
  - Used to control the use of hardware (CPU cycles, memory, swap, disk, network)
  - Applies resource usage constraints and/or local usage policies.
  - Architecture: #nodes, #cores, OS
  - Memory: amount and architecture (shared, distributed)
  - Network: topology, bandwidth, latency
  - Software: libraries, utilities

- **Dynamic Resource Info:**
  - Resource Load information and thresholding.
  - Memory: percent used.
  - Network: available bandwidth.

- **Accounting/Usage:**
  - Account status and usage of resources.
  - Track job ID, user, history
D-RMS: Scheduler and Queue Systems

- Utilizes information about local resources:
  - high-performance
  - networking, network topology
  - multi-core node configuration
  - cache and memory hierarchy
  - parallel file systems
  - parallel libraries

- Applies scheduling algorithms to organize and optimized jobs

- Enforces policies for usage and charging, etc.

Parallel Matlab Distributed Computing Server.
Note: MOM == machine oriented mini-server.
Portable Batch Scheduler (PBS/TORQUE)

Figure: TORQUE/PBS Pro Installation Architecture (tuckoo.sdsu.edu)
PBS Batch Queuing System.

**Figure:** TORQUE/PBS batch queuing system

Source: https://www.hpc2n.umu.se/support/beginners_guide
Portable Batch Scheduler (PBS) Components

- **PBS Job Server**
  - commands/daemons communicate with Server
  - batch job services: receiving/creating, running, modifying, protecting against system crashes

- **PBS Job Scheduler**
  - control when/where jobs run
  - communicate with machine oriented mini-server (MOM)

- **PBS MOM Processes**
  - Machine Oriented Miniserver
  - Starts job, makes sure it completes in specified time
  - user login session

- **PBS Client Programs**
  - command line or GUI, user, operator, manager
  - submit, monitor, modify, delete
#PBS -N localBlast  
#PBS -S /bin/sh  
#PBS -l nodes=1:ppn=2,walltime=240:00:00  
#PBS -M user@my.organization.com  
#PBS -m ea  
source ~/.bashrc  

cd $HOME/work/dir  

sh myBlast.sh -i -v

This submit script specifies the name of the job (localBlast), what environment to use (/bin/sh), that it needs both processors on a single node (nodes=1:ppn=2), that it will run for at most 10 days, and that TORQUE should email "user@my.organization.com" when the job exits or aborts. Additionally, the user specifies where and what to execute.

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td>Typically, a submit script is written to hold all of the parameters of a job. These parameters could include how long a job should run (walltime), what resources are necessary to run, and what to execute. The following is an example submit file:</td>
</tr>
<tr>
<td>Submission</td>
<td>A job is submitted with the qsub command. Once submitted, the policies set by the administration and technical staff of the site dictate the priority of the job and therefore, when it will start executing.</td>
</tr>
<tr>
<td>Execution</td>
<td>Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with qstat.</td>
</tr>
<tr>
<td>Finalization</td>
<td>When a job completes, by default, the stdout and stderr files are copied to the directory where the job was submitted.</td>
</tr>
</tbody>
</table>
PBS: Common Commands

- **Job control**
  - qsub submit a job
  - qdel delete a batch job
  - qsig send a signal to a batch job
  - qhold hold a batch job
  - qrerun rerun a batch job
  - qmove move a batch job to another queue

- **Job monitoring**
  - qstat show the status of batch jobs
  - qselect select a specific subset of jobs

- **Node status**
  - pbsnodes list the status and attributes of all nodes in the cluster.
  - [http://linuxinfo.physik.hu-berlin.de/pbs.html](http://linuxinfo.physik.hu-berlin.de/pbs.html)

- **Others**
  - qalter alter a batch job
  - qmsg send a message to a batch job

Example of User Guide: [http://rcc.its.psu.edu/user_guides/system_utilities/pbs/](http://rcc.its.psu.edu/user_guides/system_utilities/pbs/)
PBS: Batch Script Example

#!/bin/sh

#####
### request number of cores and the nodes on which to run the job
#PBS -l nodes=2:ppn=8:core4

#####
### Define the job name
#PBS -N myParallelJob

#####
### Define where the output and error messages should go
#PBS -j oe

#####
### Define the name of the queue
#PBS -q batch

#####
### change to the directory from where this script was submitted
cd $PBS_O_WORKDIR

echo ------------------------------------------------------
echo PBS: job identifier is $PBS_JOBID

echo PBS: job name is $PBS_JOBNAME


echo PBS: current home directory is $PBS_O_HOME

echo ------------------------------------------------------

NCORES=`wc -w < $PBS_NODEFILE`

echo "many-test using $NCORES cores..."

mpirun -np 10 -hostfile $PBS_NODEFILE ./myParallelJob
### PBS Batch Script: Environment Variables - Submission Machine

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_O_HOST</td>
<td>The host machine on which the qsub command was run.</td>
</tr>
<tr>
<td>PBS_O_LOGNAME</td>
<td>The login name on the machine on which the qsub was run.</td>
</tr>
<tr>
<td>PBS_O_HOME</td>
<td>The home directory from which the qsub was run.</td>
</tr>
<tr>
<td>PBS_O_WORKDIR</td>
<td>The working directory from which the qsub was run.</td>
</tr>
</tbody>
</table>

(1) PBS User Guide Example:
https://wiki.hpcc.msu.edu/display/hpccdocs/Advanced+Scripting+Using+PBS+Environment+Variables
(2) http://www.ep.ph.bham.ac.uk/general/support/torquepbsdsh.html
### PBS Batch Script: Environment Variables - Execution Machine

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBS_O_QUEUE</td>
<td>Queue where job was submitted to.</td>
</tr>
<tr>
<td>PBS_QUEUE</td>
<td>Queue job is running in (same as PBS_O_QUEUE)</td>
</tr>
<tr>
<td>PBS_JOBID</td>
<td>Job ID – used by qstat, showstart, and dque.</td>
</tr>
<tr>
<td>PBS_JOBNAME</td>
<td>Name of the job. This can be set using -N in script</td>
</tr>
<tr>
<td>PBS_NODEFILE</td>
<td>Name of file that contains list of HOSTS for job</td>
</tr>
<tr>
<td>PBS_VNODENUM</td>
<td>Determine the task number of each processor.</td>
</tr>
</tbody>
</table>

(1) PBS User Guide Example:  
https://wiki.hpcc.msu.edu/display/hpcddocs/Advanced+Scripting+Using+PBS+Environment+Variables  
(2) http://www.ep.ph.bham.ac.uk/general/support/torquepbsdsh.html
Running Parallel Jobs: typically use *mpirun*

- A script/wrapper controls several aspects of program execution in Open MPI.
- Uses the Open Run-Time Environment (ORTE) to launch jobs.
- Used to hide differences in starting jobs on different machines.
- You need to list the hosts on which the jobs will run (created by the PBS resource manager).

- You can run serial or parallel *executables*; language does not matter (C, F90, Python, etc).
- From the command line, our cluster will launch multiple Pthreads (OS/compiler dependent).
- From the batch queue, it will use the PBS resource manager.
- See online manpage:
  
  http://www.linuxcommand.org/man_pages/mpirun1.html
Specifying Attributes:  

Create a batch script and set batch queue attributes. There are a large number of attributes for selecting nodes and cores.

#PBS -l nodes=2:ppn=3

- Requesting 2 nodefiles
- Asking for 3 processors per node (ppn)
- Asking for 6 total processors
- \textit{NPROCS} = \textit{nodes} \times \textit{ppn} = 2 \times 3 = 6

```
[mthomas@tuckoo hello]$ pbsnodes node1
node1
   state = free
   np = 4
   properties = core4,type1
   ntype = cluster
   status = rectime=1423772995,...

[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
#PBS -V
#PBS -l nodes=2:ppn=3
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
echo "PBS: job name is $PBS_JOBNAME"
mpirun -np 6 -hostfile $PBS_NODEFILE ./hello
```

Tutorial:

http://ktchu.serendipityresearch.org/research/CSE/PBS.html
Setting Attributes: Request a specific number of cores

```
#PBS -l nodes=1:ppn=6
```

In this case, the system picks one node, which has 8 cores, because we requested 6 ppn.

```bash
[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
# "man pbs_resources"
#PBS -V
#PBS -l nodes=1:ppn=6
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
mpirun -np 6 -hostfile $PBS_NODEFILE ./hello
```

```bash
[mthomas@tuckoo hello]$ qsub batch.hello
197.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
tuckoo.sdsu.edu:
```

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>Elap</th>
</tr>
</thead>
<tbody>
<tr>
<td>197.tuckoo.sdsu. mthomas batch hello</td>
<td>6487</td>
<td>1</td>
<td>6</td>
<td>--</td>
<td>--</td>
<td>C</td>
<td>00:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```bash
[mthomas@tuckoo hello]$ cat hello.o197
hello-test using 6 cores...
hello, world from node8
hello, world from node8
hello, world from node8
hello, world from node8
hello, world from node8
hello, world from node8
```
Setting Attributes: Request a Specific Node Type

```
#PBS -l nodes=1:ppn=6:core6
```

In this case, the system picks one node, which has 8 cores, because we requested 6 ppn.

```
[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/bash
# "man pbs_resources"
#PBS -V
#PBS -l nodes=1:ppn=6:core6
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
mpirun -np 6 -hostfile $PBS_NODEFILE ./hello

[mthomas@tuckoo hello]$ qsub batch.hello
2019.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK Memory</th>
<th>Time</th>
<th>S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019.tuckoo</td>
<td>h.hello-cpuid mthomas</td>
<td>...</td>
<td>batch</td>
<td>00:00:00</td>
<td>C batch</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
PBS: job name is batch.hello-cpuid
hello, world from node: node5, core: 2
hello, world from node: node5, core: 4
hello, world from node: node5, core: 1
hello, world from node: node5, core: 0
hello, world from node: node5, core: 6
Specifying Attributes: Request Node by Name (v1)

```bash
#PBS -l nodes=1:ppn=8:node8

place node name after 'ppn' attribute

[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
# this example batch script requests hello processors...
# "man pbs_resources"
#PBS -V
#PBS -l nodes=1:ppn=8:node8
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES='wc -w < $PBS_NODEFILE'
echo "hello-test using $NCORES cores..."
mpirun -np 8 -hostfile $PBS_NODEFILE ./hello

[mthomas@tuckoo hello]$ qsub batch.hello
197.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
<th>Elap</th>
</tr>
</thead>
<tbody>
<tr>
<td>197.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello</td>
<td>6487</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
</tr>
</tbody>
</table>

[mthomas@tuckoo hello]$ cat hello.o197
hello-test using 4 cores...
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
```
Specifying Attributes: Request Node by Name (v2)

```bash
#PBS -l nodes=node8:ppn=8

place node name before 'ppn' attribute

[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
# this example batch script requests hello processors...
# "man pbs_resources"
#PBS -V
#PBS -l nodes=node8:ppn=8
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES=`wc -w < $PBS_NODEFILE`
echo "hello-test using $NCORES cores..."
mpirun -np 8 -hostfile $PBS_NODEFILE ./hello

[mthomas@tuckoo hello]$ qsub batch.hello
197.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
<th>Elap</th>
</tr>
</thead>
<tbody>
<tr>
<td>197.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello</td>
<td>6487</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
</tr>
</tbody>
</table>
```

hello-test using 4 cores...
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
Setting Attributes: Using Two Random Nodes

#PBS -l nodes=2:ppn=3

In this case, the system will choose the nodes

[mthomas@tuckoo hello]$ qsub batch.hello-cpuid
221.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a

tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011.tuckoo</td>
<td>mthomas</td>
<td></td>
<td>...h.hello-cpuid</td>
<td>mthomas</td>
<td></td>
<td></td>
<td></td>
<td>00:00:00</td>
<td>C batch</td>
</tr>
</tbody>
</table>

[mthomas@tuckoo hello]$ cat batch.hello-cpuid.o2011
PBS: job name is batch.hello-cpuid
hello, world from node: node2, core: 1
hello, world from node: node2, core: 3
hello, world from node: node2, core: 0
hello, world from node: node1, core: 3
hello, world from node: node1, core: 2
hello, world from node: node1, core: 2
Setting Attributes: Select Two Nodes by Name

```bash
#PBS -l nodes=node1:ppn=3+node4:ppn=3
```

In this case, the system will only use the requested nodes. The plus (+) sign combines the requested resources.

```
[mthomas@tuckoo hello]$ qsub batch.hello-cpuid
221.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a

tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>Req'd</th>
<th>Req'd</th>
<th>Elap</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011.tuckoo</td>
<td></td>
<td></td>
<td>...h.hello-cpuid</td>
<td>mthomas</td>
<td>00:00:00</td>
<td>C</td>
</tr>
</tbody>
</table>

[mthomas@tuckoo hello]$ cat batch.hello-cpuid.o2011
PBS: job name is batch.hello-cpuid
hello, world from node: node1, core: 0
hello, world from node: node1, core: 3
hello, world from node: node1, core: 1
hello, world from node: node4, core: 3
hello, world from node: node4, core: 2
hello, world from node: node4, core: 2
```
Other Ways to Specify Combinations of Nodes

```bash
### one node, 8 processors per node.
#PBS -l nodes=1:ppn=8

#PBS -l nodes="node1:ppn=4+node9:ppn=4+node11:ppn=16"

#PBS -l nodes="1:ppn=4:core4+1:ppn=4:core6+1:ppn=16:core16"

#PBS -l nodes=2:ppn=4:core4+2:ppn=6:core6

#PBS -l nodes=2:ppn=4:core4
```
Oversubscribing Nodes

- Scheduling more processes to run than there are available slots
- Oversubscribing can be useful for applications where multiple threads do not contend for CPU power.
- Oversubscribing can result in performance degradation.
- `mpif90` on tuckoo is `openmpi`.
- Open MPI schedules processes to nodes by asking two questions from each application on the `mpirun` command line:
  - How many processes should be launched? `mpirun -np X`
  - Where should those processes be launched? depends on three factors:
    - The final node list
    - The scheduling policy
    - The default and maximum number of slots on each host/node (slots are #processors on host)
    - "max_slots" value set to be the same as the number of "slots" value for each node
- use `-nooversubscribe` option

**Setting Attributes: asking for the right number of cores**

```bash
#PBS -l nodes=1:ppn=8:node8
```

In this case, the job completes because we requested enough cores, even though we did not use them all.

```
[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
# this example batch script requests hello processors...
# "man pbs_resources"
#PBS -V
#PBS -l nodes=1:ppn=8:node8
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES='wc -w < $PBS_NODEFILE'
echo "hello-test using $NCORES cores..."
mpirun -np 7 -hostfile $PBS_NODEFILE ./hello
```

```
[mthomas@tuckoo hello]$ qsub batch.hello
213.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
tuckoo.sdsu.edu:
```

```
Job ID       Username  Queue  Jobname  SessID  NDS  TSK  Req'd Memory  Time  Elap
----------   -------   ------  --------  ------  ----  ----  -------  ----  ----
213.tuckoo.sdsu. mthomas batch hello 6487  1  8  --  --  C 00:00
```

```
[mthomas@tuckoo hello]$ cat hello.o216
PBS: job name is hello
hello-test using 8 cores.
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
```

```
PBS: job name is hello
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
hello, world from node8 hello, world from node8
```
In this example below, we ask for 8 cores, which is more than were requested and are available on the core4 node. The default setting for the OpenMPI library is to allow cores to be oversubscribed: multiple threads will be launched on the core.

```
#!/bin/sh

#PBS -l nodes=1:ppn=4:core4
#PBS -N hello
#PBS -j oe
#PBS -q batch

cd $PBS_O_WORKDIR

mpirun -np 8 -hostfile $PBS_NODEFILE ./hello
```

```
2586.tuckoo.sdsu.edu
```

```
[mthomas@tuckoo hello]$ qsub batch.hello
```

```
Running: hello, using 4 cores...
hello, world from node1
hello, world from node1
hello, world from node1
hello, world from node1
```
In this case, the job fails because we used the `mpirun` argument `-nooversubscribe`.

```bash
[mthomas@tuckoo hello]$ cat batch.hello
#!/bin/sh
#PBS -V
#PBS -l nodes=1:ppn=4:core4
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES='wc -w < $PBS_NODEFILE'
echo "hello-test using $NCORES cores..."
mpirun -np 7 -hostfile $PBS_NODEFILE -nooversubscribe ./hello
```

```
[mthomas@tuckoo hello]$ qsub batch.hello
qsub batch.hello
2605.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
```
```
tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2605.tuckoo.sdsu mthomas batch hello</td>
<td>6254</td>
<td>1</td>
<td>4</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

There are not enough slots available in the system to satisfy the 8 slots that were requested by the application:

```bash
./hello
```

Either request fewer slots for your application, or make more slots available for use.
## Cluster Configuration Information Sources & Commands

### Information File

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cat /etc/motd</code> Message of the day, typically printed out on login.</td>
</tr>
<tr>
<td><code>cat /proc/cpuinfo</code> Detailed information about the CPU &amp; GPU's.</td>
</tr>
</tbody>
</table>

### Commands

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hostname</code> Prints out local hostname and IP address</td>
</tr>
<tr>
<td><code>uname -a</code> Prints out OS information</td>
</tr>
<tr>
<td><code>ifconfig</code> getting network configuration</td>
</tr>
<tr>
<td><code>pbsnodes</code> PBS/TORQUE Command</td>
</tr>
<tr>
<td><code>ip</code> displays the status of the currently active interfaces (ethernet, infinib...)</td>
</tr>
<tr>
<td><code>ip</code> show / manipulate routing, devices, policy routing and tunnels</td>
</tr>
</tbody>
</table>
## /etc/motd file

Typically printed out when you log in

```bash
[mthomas@tuckoo]$ cat /etc/motd
```

The cluster system has 10 compute nodes with various CPUs:

<table>
<thead>
<tr>
<th>Node name</th>
<th>#Avail Cores</th>
<th>Node Properties**</th>
<th>Got GPUs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1, node2, node3, node4</td>
<td>4 ea.</td>
<td>core4, type1</td>
<td>no</td>
</tr>
<tr>
<td>node8</td>
<td>8</td>
<td>-- type1</td>
<td>yes</td>
</tr>
<tr>
<td>node5</td>
<td>8</td>
<td>core6, type2</td>
<td>no</td>
</tr>
<tr>
<td>node6</td>
<td>6</td>
<td>core6, type2</td>
<td>no</td>
</tr>
<tr>
<td>node9</td>
<td>6</td>
<td>core6, type2</td>
<td>yes</td>
</tr>
<tr>
<td>node10</td>
<td>16</td>
<td>core16, type2</td>
<td>no</td>
</tr>
<tr>
<td>node11</td>
<td>16</td>
<td>core16, type2</td>
<td>yes</td>
</tr>
</tbody>
</table>

**See the output from "pbsnodes -a".

### CPUs & RAM

- **node1 thru node4**, Xeon X3360 @ 2.83GHz, 8GB ea.
- **node5** Xeon E5420 @ 2.50GHz, 20GB
- **node6** Xeon E5-1650 @ 3.20GHz, 64GB
- **node8** Xeon E5620 @ 2.40GHz, 48GB
- **node9** Xeon E5-1660 @ 3.30GHz, 32GB
- **node10** Xeon E5-2650 @ 2.60GHz, 64GB
- **node11** Xeon E5-2650 @ 2.60GHz, 64GB

### GPUs

- **node9** has 2 GTX 480 gpu cards (1.6GB dev ram ea.)
- **node8** has 2 C2075 gpu cards (6GB dev ram ea.)
- **node11** has 1 K40 gpu card ( )
Obtaining Cluster Configuration Information

[/proc/cpuinfo]

[mthomas@tuckoo ~]$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 26
model name : Intel(R) Xeon(R) CPU E5504 @ 2.00GHz
stepping : 5
cpu MHz : 1600.000
cache size : 4096 KB
physical id : 1
siblings : 4
core id : 0
cpu cores : 4
apicid : 16
initial apicid : 16
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe sys
bogomips : 3999.68
clflush size : 64
cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual
power management:

processor : 1
vendor_id : GenuineIntel
cpu family : 6
model : 26
...
**Table:** Table of tuckoo CPU/GPU configurations (Spring, 2014)

<table>
<thead>
<tr>
<th>Property</th>
<th>csrgc-gpu</th>
<th>csrgc-gpu2</th>
<th>csrgc-gpu3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node ID</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>CPU Type</td>
<td>2 Xeon X5650</td>
<td>2 Xeon X5650</td>
<td>2 Xeon E5620</td>
</tr>
<tr>
<td>#CPU cores</td>
<td>6*2=12</td>
<td>4*2=8</td>
<td>3*2=6</td>
</tr>
<tr>
<td>GPU Type</td>
<td>2 Tesla C1060</td>
<td>2 Tesla C2075</td>
<td>2 Tesla C2075</td>
</tr>
<tr>
<td>Multiprocessor (MP)</td>
<td>30</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>#SP/cores</td>
<td>240</td>
<td>448</td>
<td>448</td>
</tr>
<tr>
<td>#GigaFLOPS</td>
<td>512</td>
<td>515</td>
<td>515</td>
</tr>
<tr>
<td>Max Thd/Block</td>
<td>64k x 64k x 1</td>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>Max Grd Dim</td>
<td>64k x 64k x 1</td>
<td>64k x 64k x 64</td>
<td>64k x 64k x 64</td>
</tr>
</tbody>
</table>
Obtaining information on all nodes: `pbsnodes -a`

```
[mthomas@tuckoo hello]$ pbsnodes -a
node1
  state = free
  np = 4
  properties = core4,type1
  ntype = cluster
  status = rectime=1423769834,varattr=,jobs=,state=free,netload=11658237823,gres=,loadave=0.99,ncpus=4,
  physmem=7929488kb,availmem=33728156kb,totmem=34011004kb,idletime=148949,nusers=1,
  nsessions=1,sessions=1811,uname=Linux node1 2.6.32-220.17.1.el6.x86_64 #1
  SMP Wed May 16 00:01:37 BST 2012 x86_64,opsys=linux gpus = 0

node2
  state = free
  np = 4
  properties = core4,type1
  ntype = cluster
  status = rectime=1423769848,varattr=,jobs=,state=free,...
  gpus = 0

node3
  state = free
  np = 4
  properties = core4,type1
  ntype = cluster
  status = rectime=1423769846,varattr=,jobs=,state=free,...
  gpus = 0

node4
  state = free
  np = 4
  properties = core4,type1
  ntype = cluster
  status = rectime=1423769849,varattr=,jobs=,state=free,...
  gpus = 0
...
```
Obtaining information on one node:

```bash
[mthomas@tuckoo hello]$ pbsnodes node11
node11
    state = free
    np = 16
    properties = core16,type2
    ntype = cluster
    status = rectime=1423800490,varattr=,jobs=,state=free,netload=2257827798,gres=,loadave=0.00,ncpus=16,
            physmem=65941452kb,availmem=131961592kb,totmem=133050308kb,idletime=106538,
            nusers=1,nsessions=6,sessions=11134 11138 11139 11147 11149 11171,uname=Linux node11
    2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64,opsys=linux
    gpus = 0

[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$ pbsnodes node8
node8
    state = free
    np = 8
    properties = type1
    ntype = cluster
    status = rectime=1423800494,varattr=,jobs=,state=free,netload=1866286069,gres=,loadave=0.52,ncpus=8,
            physmem=49414804kb,availmem=95577436kb,totmem=96300496kb,idletime=106508,
            nusers=1,nsessions=1,sessions=1884,uname=Linux node8
    2.6.32-220.17.1.el6.x86_64 #1 SMP Wed May 16 00:01:37 BST 2012 x86_64,opsys=linux
    gpus = 0

[mthomas@tuckoo hello]$
```
More condensed summary:

`pbsnodes -a | grep properties`

tuckoo naming convention:
- core4 node has 4 cores
- core6 node has 6 cores
- core16 node has 16 cores

```
[mthomas@tuckoo]$ pbsnodes -a | grep properties
properties = core4,type1
properties = core4,type1
properties = core4,type1
properties = core4,type1
properties = type1
properties = core6,type2
properties = core6,type2
properties = core6,type2
properties = core16,type2
properties = core16,type2
```
## Network information

```bash
[mthomas@tuckoo ~]$ hostname
tuckoo
[mthomas@tuckoo ~]$
[mthomas@tuckoo ~]$ cat /proc/net/dev
```

<table>
<thead>
<tr>
<th>Inter-</th>
<th>Receive</th>
<th>Transmit</th>
</tr>
</thead>
<tbody>
<tr>
<td>face</td>
<td>bytes</td>
<td>packets</td>
</tr>
<tr>
<td>lo:</td>
<td>665015398</td>
<td>704989</td>
</tr>
<tr>
<td>eth0:</td>
<td>22597919775</td>
<td>19820450</td>
</tr>
<tr>
<td>eth1:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eth2:</td>
<td>1995666451</td>
<td>18784458</td>
</tr>
</tbody>
</table>

```bash
[mthomas@tuckoo ~]$
```

```bash
[mthomas@tuckoo ~]$ ssh node1 "cat /proc/net/dev"
mthomas@node1’s password:
```

```bash
```
```bash
Inter- | Receive | Transmit |
face   | bytes   | packets  | errs | drop | fifo | frame | compressed | multicast | bytes   | packets | errs | drop | fifo | carrier | compressed |
lo:    | 3039772 | 10402    | 0    | 0    | 0    | 0     | 0         | 3039772   | 10402   | 0       | 0      | 0     | 0     | 0       | 0          |
eth0:  | 6276061155 | 5974043  | 0    | 0    | 0    | 0     | 0         | 102       | 8119134972 | 7258708 | 0       | 0      | 0     | 0     | 0       | 0          |
ib0:   | 0        | 0        | 0    | 0    | 0    | 0     | 0         | 0          | 0       | 0       | 0      | 0     | 0     | 0       | 0          |

```bash
[mthomas@tuckoo ~]$
```

```bash
[mthomas@tuckoo ~]$ ssh node11 "cat /proc/net/dev"
mthomas@node11’s password:
```

```bash
```
```bash
Inter- | Receive | Transmit |
face   | bytes   | packets  | errs | drop | fifo | frame | compressed | multicast | bytes   | packets | errs | drop | fifo | carrier | compressed |
lo:    | 1838675 | 7213     | 0    | 0    | 0    | 0     | 0         | 1838675   | 7213    | 0       | 0      | 0     | 0     | 0       | 0          |
eth0:  | 2067241887 | 2195727  | 0    | 0    | 0    | 0     | 0         | 37114     | 414102996 | 744255  | 0       | 0      | 0     | 0     | 0       | 0          |
eth1:  | 0        | 0        | 0    | 0    | 0    | 0     | 0         | 0          | 0       | 0       | 0      | 0     | 0     | 0       | 0          |
ib0:   | 0        | 0        | 0    | 0    | 0    | 0     | 0         | 0          | 0       | 0       | 0      | 0     | 0     | 0       | 0          |
```
Network information

[mthomas@tuckoo ~]$ ssh node1 "/sbin/ifconfig eth0"
mthomas@node1’s password:
eth0     Link encap:Ethernet  HWaddr 00:24:81:B2:06:F7
         inet addr:10.0.1.1  Bcast:10.0.1.255  Mask:255.255.255.0
         inet6 addr: fe80::224:81ff:feb2:6f7/64 Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         RX packets:5976844 errors:0 dropped:0 overruns:0 frame:0
         TX packets:7260934 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1000
         RX bytes:6276774881 (5.8 GiB)  TX bytes:8119389701 (7.5 GiB)
         Interrupt:17

[mthomas@tuckoo ~]$ ssh node11 "/sbin/ifconfig ib0"
mthomas@node11’s password:
ib0     Link encap:InfiniBand  HWaddr A0:04:02:20:FE:80:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
         BROADCAST MULTICAST  MTU:4092  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:1024
         RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

Ifconfig uses the ioctl access method to get the full address information, which limits hardware addresses to 8 bytes. Because Infiniband address has 20 bytes, only the first 8 bytes are displayed correctly. Ifconfig is obsolete! For replacement check ip.
Job Example: Serial Hello World

```c
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    char cptr[100];

    gethostname(cptr,100);
    printf("hello, world from \%s\n", cptr);

    return 0;
}
```
Compiling & Running Serial Code – Command Line

[mthomas@tuckoo hello]$ ls hello*
-rw-r--r-- 1 mthomas mthomas 263 Feb 12 10:00 hello.c
-rwx------- 1 mthomas mthomas 791 Feb 12 09:54 hello.f90

===================================
COMPILE USING mpicc
===================================

[mthomas@tuckoo hello]$ mpicc -o hello hello.c
[mthomas@tuckoo hello]$
[mthomas@tuckoo hello]$ ls hello*
-rwxrwxr-x 1 mthomas mthomas 7276 Feb 12 10:03 hello
-rw-r--r-- 1 mthomas mthomas 263 Feb 12 10:00 hello.c
-rwx------- 1 mthomas mthomas 791 Feb 12 09:54 hello.f90
[mthomas@tuckoo hello]$

===================================
RUN SERIAL CODE FROM COMMAND LINE
===================================

[mthomas@tuckoo hello]$ ./hello
hello, world from tuckoo
[mthomas@tuckoo hello]$

===================================
RUN SERIAL CODE FROM COMMAND LINE USING A PARALLEL ENVIRONMENT (mpirun)
===================================

[mthomas@tuckoo hello]$ mpirun -np 4 ./hello
hello, world from tuckoo
hello, world from tuckoo
hello, world from tuckoo
hello, world from tuckoo

How did the serial code run 4 times? mpirun.
Using Batch Queue to run Serial Code

======================================
= You don’t need to recompile code  
======================================
[mthomas@tuckoo hello]$ cat batch.hello.1
#!/bin/sh
# this example batch script requests hello processors...
# "man pbs_resources"
#PBS -V
##### SPECIFY NODE ATTRIBUTES #######
#PBS -l nodes=1:ppn=1:node1 
#####
#PBS -N hello.1
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
NCORES=`wc -w < $PBS_NODEFILE`
echo "Running $PBS_JOBNAME, using $NCORES cores..."
mpirun -np 1 -hostfile $PBS_NODEFILE ./hello
[mthomas@tuckoo hello]$  
[mthomas@tuckoo hello]$ qsub batch.hello.1  
177.tuckoo.sdsu.edu  
[mthomas@tuckoo hello]$ qstat

<table>
<thead>
<tr>
<th>Job id</th>
<th>Name</th>
<th>User</th>
<th>Time Use</th>
<th>S</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>177.tuckoo</td>
<td>hello.1</td>
<td>mthomas</td>
<td>00:00:00</td>
<td>C</td>
<td>batch</td>
</tr>
</tbody>
</table>

[mthomas@tuckoo hello]$  
[mthomas@tuckoo hello]$ cat hello.1.o177
hello.1 using 1 cores...
hello, world from node11
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

/* This code is adapted from an example at: http://brokestream.com/procstat.html*/

int get_cpu_id() {
    /* Get the current process' stat file from the proc filesystem */
    FILE* procfile = fopen("/proc/self/stat", "r");
    long to_read = 8192;
    char buffer[to_read];
    int read = fread(buffer, sizeof(char), to_read, procfile);
    fclose(procfile);

    // Field with index 38 (zero-based counting) is the one we want
    char* line = strtok(buffer, " ");
    int i;
    for (i = 1; i < 38; i++) {
        line = strtok(NULL, " ");
    }
    line = strtok(NULL, " ");
    int cpu_id = atoi(line);
    return cpu_id;
}

int main(void) {
    char cptr[100];
    int cpuid;

    gethostname(cptr,100);
    cpuid = get_cpu_id();
    printf("hello, world from node: %s, core: %d\n", cptr, cpuid);

    return 0;
}
Showing Node & Core ID Using One Node

```
[mthomas@tuckoo hello]$ qsub batch.hello-cpuid
198.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a
```

```
tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>S Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>220.tuckoo.sdsu.mthomas batch hello-cpuid</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>--</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>Q</td>
</tr>
</tbody>
</table>
```

```
[mthomas@tuckoo hello]$ cat hello-cpuid.o197
cat: hello-cpuid.o197: No such file or directory
[mthomas@tuckoo hello]$ cat hello-cpuid.o198
-bash: BASH_FUNC_module(): line 0: syntax error near unexpected token ‘)’
-bash: BASH_FUNC_module(): line 0: ‘BASH_FUNC_module() () { eval `/usr/bin/modulecmd bash $*`’
-bash: error importing function definition for ‘BASH_FUNC_module’
```

```
hello-cpuid-test using 16 cores...
[mthomas@tuckoo hello]$ cat hello-cpuid.o220
PBS: job name is hello-cpuid
hello, world from node: node8, core: 7
hello, world from node: node8, core: 3
hello, world from node: node8, core: 4
hello, world from node: node8, core: 5
hello, world from node: node8, core: 2
hello, world from node: node8, core: 1
hello, world from node: node8, core: 5
hello, world from node: node8, core: 6
```

Set Attributes: #PBS -l nodes=1:ppn=8
Showing Node & Core ID Using Two Nodes

[mthomas@tuckoo hello]$ qsub batch.hello-cpuid
221.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat -a

tuckoo.sdsu.edu:

<table>
<thead>
<tr>
<th>Job ID</th>
<th>Username</th>
<th>Queue</th>
<th>Jobname</th>
<th>SessID</th>
<th>NDS</th>
<th>TSK</th>
<th>Memory</th>
<th>Time</th>
<th>Req'd S</th>
<th>Req'd Time</th>
<th>Elap Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>217.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>25132</td>
<td>2</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>218.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>25164</td>
<td>2</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>219.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>6775</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>220.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>6803</td>
<td>1</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>221.tuckoo.sdsu.</td>
<td>mthomas</td>
<td>batch</td>
<td>hello-cpuid</td>
<td>25198</td>
<td>2</td>
<td>8</td>
<td>--</td>
<td>--</td>
<td>C 00:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[mthomas@tuckoo hello]$ cat hello-cpuid.o221
PBS: job name is hello-cpuid
hello, world from node: node2, core: 1
hello, world from node: node2, core: 3
hello, world from node: node2, core: 2
hello, world from node: node2, core: 0
hello, world from node: node1, core: 2
hello, world from node: node1, core: 3
hello, world from node: node1, core: 3
hello, world from node: node1, core: 2

Set Attributes: #PBS -l nodes=2:ppn=2:core4
Printing Out Detailed PBS Information

[mthomas@tuckoo hello]$ cat batch.details
#!/bin/sh
#PBS -V
#PBS -l nodes=1:ppn=4
#PBS -N batch.details
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
echo ------------------------------------------------------
echo " date: " 'date'
echo "hostname: " 'hostname'
echo "whoami: " 'whoami'
echo " pwd: " 'pwd'
echo ------------------------------------------------------
echo -n 'Job is running on node '; cat $PBS_NODEFILE
echo ------------------------------------------------------
ncores=`wc -w < $PBS_NODEFILE`
ncores=$ncores
ncores=`echo $ncores`
echo "Running: $PBS_JOBNAME, using $ncores cores..."
mpirun -np 4 -hostfile $PBS_NODEFILE ./hello-cpuid
Printing Out PBS Environment and Job Information

[mthomas@tuckoo hello]$ cat batch.details.o2070
------------------------------------------------------
date: Mon Feb 23 13:59:19 PST 2015
hostname: node2
whoami: mthomas
pwd: /home/mthomas/pardev/hello
------------------------------------------------------
Job is running on node	node2
node2
node2
node2
------------------------------------------------------
PBS: qsub is running on tuckoo.sdsu.edu
PBS: originating queue is batch
PBS: executing queue is batch
PBS: working directory is /home/mthomas/pardev/hello
PBS: execution mode is PBS_BATCH
PBS: job identifier is 2070.tuckoo.sdsu.edu
PBS: job name is batch.details
PBS: node file is /var/spool/torque/aux//2070.tuckoo.sdsu.edu
PBS: current home directory is /home/mthomas
PBS: PATH = /usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/sbin:/usr/local/openmpi/bin:/usr/local/torque/bin:/usr/local/torque/sbin:/usr/local/cuda/bin:/usr/local/tau/x86_64/bin:/usr/local/vampirtrace/bin:/opt/pgi/linux86-64/11.0/bin
------------------------------------------------------
Running: batch.details, using 4 cores...
------------------------------------------------------
hello, world from node: node2, core: 3
hello, world from node: node2, core: 1
hello, world from node: node2, core: 0
hello, world from node: node2, core: 2
/*
 * File: hello-arg.c by Mary Thomas, 2/12/15
 */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

/* This code is adapted from an example at: http://brokestream.com/procstat.html */

int get_cpu_id() {
    ..
}

int main (int argc, char* argv[]) 
{
    long cmdarg;
    char cptr[100];
    int cpuid;

    /* read long integer from the command line */
    if (argc != 2) {
        printf("usage: hello-arg <integer number>\n");
        return 0;
    }
    cmdarg = strtol(argv[1], NULL, 10);

    gethostname(cptr,100);
    cpuid = get_cpu_id();
    printf("hello, world from node: %s, core: %d, cmdarg= %ld\n", cptr, cpuid, cmdarg);

    return 0;
}
Example #1: Hard code the value in the script

```
[mthomas@tuckoo hello]$ cat batch.hello-arg
#!/bin/sh
#PBS -V
#PBS -l nodes=2:ppn=4:core4
#PBS -N hello-arg
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
echo "PBS: job name is $PBS_JOBNAME"
NCORES=`wc -w < $PBS_NODEFILE`
echo "$PBS_JOBNAME running using $NCORES cores..."
mpirun -np 8 -hostfile $PBS_NODEFILE --nooversubscribe ./hello-arg 12345
[mthomas@tuckoo hello]$ qsub batch.hello-arg
229.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat
<table>
<thead>
<tr>
<th>Job id</th>
<th>Name</th>
<th>User</th>
<th>Time</th>
<th>Use</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>hello-arg</td>
<td>mthomas</td>
<td>00:00:00</td>
<td>C</td>
<td>batch</td>
</tr>
</tbody>
</table>
[mthomas@tuckoo hello]$ cat hello-arg.o229
PBS: job name is hello-arg
hello-arg running using 8 cores...
hello, world from node: node2, core: 3, cmdarg= 12345
hello, world from node: node2, core: 1, cmdarg= 12345
hello, world from node: node2, core: 2, cmdarg= 12345
hello, world from node: node2, core: 0, cmdarg= 12345
hello, world from node: node1, core: 3, cmdarg= 12345
hello, world from node: node1, core: 2, cmdarg= 12345
hello, world from node: node1, core: 3, cmdarg= 12345
hello, world from node: node1, core: 0, cmdarg= 12345
```
Example #1: Pass the argument to the *qsub* command

```bash
[mthomas@tuckoo hello]$ cat batch.hello-arg-qsubv
#!/bin/sh
# this batch script takes variables passed to the qsub command
# and passes them to the hello-arg executable as command line arguments. Usage:
# %qsub -v CMDARG=7654 batch.hello-arg-qsubv
#
#PBS -V
#PBS -l nodes=1:ppn=4:core4
#PBS -N hello-arg-qsubvars
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
echo "PBS: job name is $PBS_JOBNAME"
NCORES='wc -w < $PBS_NODEFILE'
echo "$PBS_JOBNAME running using $NCORES cores..."
mpirun -np 4 -hostfile $PBS_NODEFILE --nooversubscribe ./hello-arg $CMDARG
```

```
[mthomas@tuckoo hello]$ qsub batch.hello-arg-qsubv
230.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qsub -v CMDARG=9876 batch.hello-arg-qsubv
231.tuckoo.sdsu.edu
[mthomas@tuckoo hello]$ qstat
[mthomas@tuckoo hello]$ cat hello-arg-qsubvars.o231
PBS: job name is hello-arg-qsubvars
hello-arg-qsubvars running using 8 cores...
hello, world from node: node1, core: 0, cmdarg= 9876
hello, world from node: node1, core: 1, cmdarg= 9876
hello, world from node: node1, core: 2, cmdarg= 9876
hello, world from node: node1, core: 3, cmdarg= 9876
```