Table of Contents

1. Misc Information
2. Resource Management
   - Distributed Resource Management Systems (D-RMS)
   - Portable Batch Scheduler (PBS/TORQUE)
3. Running Jobs on a Cluster
   - Running Parallel Code
   - Requesting cores on nodes
   - Oversubscribing Nodes
   - Passing Command Line Parameters to Batch Job
4. Next Time
Today:
- Resource Management
- Using the PBS queue
- Running jobs on the student cluster

Next HW: tba

Quiz 1: tentative date is 09/25/14 (Thurs)
# CRAY HPC Software Stack

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

<table>
<thead>
<tr>
<th>Performance Monitoring</th>
<th>HPCC</th>
<th>Perfctr</th>
<th>IOR</th>
<th>PAPI/IPM</th>
<th>netperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development Tools</td>
<td>Cray® Compiler Environment (CCE)</td>
<td>Intel® Cluster Studio</td>
<td>PGI (PGI CDK)</td>
<td>GNU</td>
<td></td>
</tr>
<tr>
<td>Application Libraries</td>
<td>Cray® LibSci, LibSci_ACC</td>
<td>MVAPICH2</td>
<td>OpenMPI</td>
<td>Intel® MPI- (Cluster Studio)</td>
<td></td>
</tr>
</tbody>
</table>

### HPC Programming Tools

### Middleware Applications and Management

<table>
<thead>
<tr>
<th>Resource Management / Job Scheduling</th>
<th>SLURM</th>
<th>Grid Engine</th>
<th>MOAB</th>
<th>Altair PBS Pro</th>
<th>IBM Platform LSF</th>
<th>Torque/Maui</th>
</tr>
</thead>
<tbody>
<tr>
<td>File System</td>
<td>NFS</td>
<td>Local FS (ext3, ext4, XFS)</td>
<td>PanFS</td>
<td>Lustre</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provisioning</td>
<td>Cray® Advanced Cluster Engine (ACE) management software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cluster Monitoring</td>
<td>Cray ACE (SCB and OpenIPMI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Power Mgmt</td>
<td>Cray ACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remote Console Mgmt</td>
<td>Cray ACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Operating Systems

| Operating System | Linux (Red Hat, CentOS, SUSE) |

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).

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

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, SLUM, SLAM, SLAMM) or **Simple Linux Utility for Resource Management (SLURM, SLUM, SLAM, SLAMM)**
    - [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
resource management

Distributed Resource Management Systems (D-RMS)

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/TORQUE Job life cycle

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
</table>
| Creation    | 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:  

```bash
#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
csh 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. |
| Submission   | 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. |
| Execution    | Jobs often spend most of their lifecycle executing. While a job is running, its status can be queried with `qstat`. |
| Finalization | When a job completes, by default, the `stdout` and `stderr` files are copied to the directory where the job was submitted. |
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/)
Portable Batch Scheduler (PBS/TORQUE)

PBS: Batch Script Example

#!/bin/sh
# this example batch script requests many processors...
# for more info on requesting specific nodes see "man pbs\_resources"
#PBS -V

# request number of cores and the nodes on which to run them
#PBS -l nodes=2:ppn=2:core4+3:ppn=2:core8

#PBS -N myParallelJob
#PBS -j oe
#PBS -q batch
cd $PBS\_O\_WORKDIR

echo ------------------------------------------------------
echo -n 'Job is running on node '; cat $PBS\_NODEFILE

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

NCORES='wc -w < $PBS\_NODEFILE'

mpirun -np 10 -hostfile $PBS\_NODEFILE --nooversubscribe ./myParallelJob
**Resource Management**

**Portable Batch Scheduler (PBS/TORQUE)**

### PBS: 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
http://www.ep.ph.bham.ac.uk/general/support/torquepbsdsh.html
Resource Management
Portable Batch Scheduler (PBS/TORQUE)

## PBS: 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/hpccdocs/Advanced+Scripting+Using+PBS+Environment+Variables (2)
http://www.ep.ph.bham.ac.uk/general/support/torquepbsdsh.html
Using a Cluster

Cluster: compute nodes

Switch

Login node

Internet

Users, submitting jobs

**Figure:** Diagram of a cluster

Source: [https://www.hpc2n.umu.se/support/beginners_guide](https://www.hpc2n.umu.se/support/beginners_guide)
Welcome to tuckoo (baby) -- a CSRC student cluster (Sept., 2014) 
the cluster system has 9 compute nodes with various CPUs:

<table>
<thead>
<tr>
<th>Node name</th>
<th>#Avail Cores</th>
<th>Node Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>node1,node2,node3,node4</td>
<td>4ea.</td>
<td>core4,mpi**</td>
</tr>
<tr>
<td>node5</td>
<td>6 [8]*</td>
<td>core6,mpi</td>
</tr>
<tr>
<td>node6</td>
<td>6</td>
<td>core6,mpi,bigmem</td>
</tr>
<tr>
<td>node7</td>
<td>6 [12]*</td>
<td>core6,mpi,highmem</td>
</tr>
<tr>
<td>node8</td>
<td>6 [8]*</td>
<td>core6,mpi,highmem</td>
</tr>
<tr>
<td>node9</td>
<td>6</td>
<td>core6,mpi,highmem</td>
</tr>
</tbody>
</table>

------ CPUs & RAM ------
node1 thru node4, Xeon X3360 @ 2.83GHz, 8GB ea.
node5 Xeon E5420 @ 2.50GHz, 20GB
node6 Xeon E5-1650 @ 3.20GHz, 64GB
node7 Xeon X5650 @ 2.67GHz, 48GB
node8 Xeon E5620 @ 2.40GHz, 48GB
node9 Xeon E5-1660 @ 3.30GHz, 32GB

------ GPUs ------
node7 has 2 Tesla C1060 gpu cards (4GB dev ram)
node8 has 2 Tesla C2075 gpu cards (6GB dev ram)
node9 has 2 GTX 480 gpu cards (1.6GB dev ram)
- Intel Xeon X5650 system contains six CPUs (Xeon 5650)
- QPI-PCIe bridge;
- PCI-e switch for GPUs.
System with 2 Intel Xeon X5650 and 8 Nvidia GPU Teslas

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

Source: http://hothardware.com/Articles/NVIDIA-GF100-Architecture-and-Feature-Preview
### Table: Table of tuckoo CPU/GPU configurations (Spring, 2014)

<table>
<thead>
<tr>
<th>Property</th>
<th>csr-c-gpu</th>
<th>csr-c-gpu2</th>
<th>csr-c-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><strong>2 Tesla C1060</strong></td>
<td><strong>2 Tesla C2075</strong></td>
<td><strong>2 Tesla C2075</strong></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>64k x 64k x 64</td>
<td>64k x 64k x 64</td>
</tr>
<tr>
<td>Max Grd Dim</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Myrinet Network Architecture

Memory

CPU

CPU

Memory Bus

PCI Bus

NIC

DMA

DMA

Memory

CPU

DMA

DMA

PCI Bridge

PCI bus to network ratio ≈ 1:1
memcpys() unacceptable
Mem to PCI bus ratio ≈ 10:1
memcpys() OK
Network

Source:
Running Jobs on a Cluster

Getting info about cluster nodes: `cat /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 sysman
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
...```
Running Jobs on a Cluster

Getting info about cluster nodes: `pbsnodes -a`

```
[mthomas@tuckoo ~]$ pbsnodes -a
node1
   state = free
   np = 4
   properties = core4,node1,mpi
   ntype = cluster
   status = rectime=1410904256, varattr=, jobs=, state=free, netload=311077211,
           gres=, loadave=0.78, ncpus=4, physsmem=7929488kb,
           availmem=33747768kb, totmem=34011004kb,
           idletime=2311379, nusers=1, nsessions=1, sessions=1963,
           uname=Linux node1 2.6.32-220.17.1.el6.x86_64 #1 SMP
           Wed May 16 00:01:37 BST 2012 x86_64, opsyst=linux
           gpus = 0

node2
   state = free
   np = 4
   properties = core4,node2,mpi
   ntype = cluster
   status = rectime=1410904256, varattr=, jobs=, state=free, netload=338458065,
           gres=, loadave=0.42, ncpus=4, physsmem=7929488kb,
           availmem=31651424kb, totmem=31914524kb,
           idletime=2311422, nusers=1, nsessions=1, sessions=1985,
           uname=Linux node2 2.6.32-220.17.1.el6.x86_64 #1 SMP
           Wed May 16 00:01:37 BST 2012 x86_64, opsyst=linux
           gpus = 0

node3
   state = free
   np = 4
   properties = core4,node3,mpi
   ntype = cluster
   status = rectime=1410904242, varattr=, jobs=, state=free, netload=316680815,gres=, loadave=0.00, ncpus=4, physsmem=7929488kb, avail
           gpus = 0
```
Getting info about cluster nodes: `pbsnodes -a | grep properties`

```bash
[mthomas@tuckoo ~]$ pbsnodes -a | grep properties
properties = core4,node1,mpi
properties = core4,node2,mpi
properties = core4,node3,mpi
properties = core4,node4,mpi
properties = core4,node5,mpi
properties = core6,node5,mpi
properties = core6,node6,mpi,bigmem
properties = core6,node7,mpi,highmem
properties = core6,node8,mpi,highmem
properties = core6,node9,mpi,highmem
```
int main (int argc, char* argv[]) {
    int rank, num_nodes, ierr, other;
    int tkn;
    char cptr[100];
    MPI_Status status;
    gethostname(cptr, 100);
    ierr = MPI_Init(&argc, &argv);
    if (ierr != MPI_SUCCESS) {
        printf("MPI initialization error\n"); return 0;
    }
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &num_nodes);
    if (num_nodes != 2) {
        if (rank == ROOT) {
            printf("program error: runs on 2 processes!\n");
        }
        MPI_Finalize();
        return 0;
    }
    if (rank == 0) { /* get things started */
        token = 0; other = OTHER;
        MPI_Send(&tkn, 1, MPI_INT, other, 0, MPI_COMM_WORLD);
        MPI_Recv(&tkn, 1, MPI_INT, other, 0, MPI_COMM_WORLD, &status);
        printf("process %d: (%s) received the token...\n", rank, cptr);
        if (token == 1) {
            printf("process %d: token= %d, matches (things look ok).\n", rank, tkn);
        } else {
            printf("process %d: ERR -- val of token (%d) does not match expected (%d)!\n", rank, token, 1);
        }
    } else {
        other = ROOT;
        MPI_Recv(&token, 1, MPI_INT, other, 0, MPI_COMM_WORLD, &status);
        token++;
        MPI_Send(&token, 1, MPI_INT, other, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
Running Jobs on a Cluster

Running Parallel Code

- Copy examples

[mthomas@tuckoo ]$ cp -r /examples/ ex.2014
[mthomas@tuckoo pardev]$ cd ex.2014

- make

[mthomas@tuckoo ex.2014]$ make two
mpicc -o two two.c

- BATCHSCRIPT: batch.twoF

[mthomas@tuckoo hw1]$ cat batch.twoF
#!/bin/sh
# for more info on requesting specific
# nodes see "man pbs_resources"
#PBS -V
#PBS -l nodes=2:ppn=1:core4
#PBS -N twoF
#PBS -j oe
#PBS -q batch
echo NODEFILE: $PBS_NODEFILE
cd $PBS_O_WORKDIR
NCORES=`wc -w < $PBS_NODEFILE`
echo "twoF test using $NCORES cores..."
mpirun -np 2 -host node6,node7 ./two

### use nooversubscribe to run 1 job/pe or core
###mpirun -np 4 -hostfile $PBS_NODEFILE --nooversubscribe ./looptstp

- RUN JOB

[mthomas@tuckoo ex.2014]$ qsub batch.two
16442.tuckoo.sdsu.edu

- OUTPUT

[mthomas@tuckoo ex.2014]$ cat two.o16442
NODEFILE: /var/spool/torque/aux//16442.tuckoo.sdsu.edu
twoF test using 8 cores...
process 0 of 2 (on node6)
process 1 of 2 (on node7)
success (process 0)
Common commands used to run jobs

- **node or PE**: CPU computer, with multiple (typically 4 or 8) processor cores.
- **pbsnodes -a**: nd out which nodes are available, and how the nodes are configured.
- **qstat or qstat -n**: used to see which jobs have been submitted and show up in the queue.
- **qsub batch.name**: used to submit a PBS batch script to the queue.
- **qdel job-id**: used to delete a job from the queue based on the job-id returned by qsub.
MPI Code: Hello.f90 Code

program hello

    implicit none
    include 'mpif.h'

    integer:: rank, npes, i, ierr
    character(200) :: host, date

    call MPI_Init(ierr)
    call MPI_Comm_rank( MPI_COMM_WORLD, rank, ierr )
    call MPI_Comm_size( MPI_COMM_WORLD, npes, ierr )
    call sleep(20)

    call execute_command_line("hostname",cmdhost=host)
    call hostnm(host)
    call fdate(date)
    write(*,fmt='("MyRank",i2,"]: Total #PEs:"i2," running on Node: ",a", Date: ",a)' ) &
            rank,npes, trim(host), trim(date)

    call MPI_Finalize(ierr)

end program hello
batch.hello script:

- Specifies:
  `nodes=2:ppn=2:core4`

- Request for 2 nodes

- Node is type core4:
  each has 4 processors per node (ppn)

- Asking for 2 processors per node

- Specify core type: core4

- \[ NPROCS = nodes \times ppn = 2 \times 4 = 8 \]
Running Jobs on a Cluster

Requesting cores on nodes

[mthomas@tuckoo pardev]$ cat hello/hello.o57464
------------------------------------------------------
Job is running on node node1
node1
node1
node1
------------------------------------------------------
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 57464.tuckoo.sdsu.edu
PBS: job name is hello
PBS: node file is /var/spool/torque/aux//57464.tuckoo.sdsu.edu
PBS: current home directory is /home/mthomas
PBS: PATH = ... l/cuda/bin:/usr/local/tau/x86_64/bin:/usr/local/vampirtrace/bin:/opt/pgi/linux86-64/11.0/bin:/home/mthomas/bin:/usr/lib64/
------------------------------------------------------
hello-test using 4 cores...
Tue Sep 16 15:07:58 PDT 2014
MyRank[ 0]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:08:19 2014
MyRank[ 1]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:08:19 2014
MyRank[ 3]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:08:19 2014
MyRank[ 2]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:08:19 2014
Tue Sep 16 15:08:19 PDT 2014

What happens when we ask for more nodes than we put into request or are available on the nodes? Set np=16
Running Jobs on a Cluster

Oversubscribing Nodes

```
  mpirun -np 16 -hostfile $PBS_NODEFILE ./hello
16 processes ran, but we only asked for 2 core4 nodes (8 max PE's)
```
# Oversubscribing Nodes

- Scheduling more processes to run than there are available slots
- 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

Recall mpi_hello.c: sends local message to master, where it is printed to STDOUT.

```c
if (my_rank != 0) {
    /* Create message */
    printf(greeting, "Greetings from Slave: %d of %d!",
            my_rank, comm_sz);
    /* Send message to process 0 */
    MPI_Send(greeting, strlen(greeting)+1, MPI_CHAR, 0, 0,
             MPI_COMM_WORLD);
} else {
    /* Print my message */
    printf("Greetings from Master: %d of %d!\n", my_rank, comm_sz);
    for (int q = 1; q < comm_sz; q++) {
        /* Receive message from process q */
        MPI_Recv(greeting, MAX_STRING, MPI_CHAR, q,
                  0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        /* Print message from process q */
        printf("%s\n", greeting);
    }
}
```
Simple Hello World Batch script - Fix

#!/bin/sh
# this example batch script requests hello processors...
# for more info on requesting specific nodes see "man pbs_resources"
#PBS -V
#PBS -l nodes=2:ppn=2:core4
#PBS -N hello
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR
echo ------------------------------------------------------------------
echo -n 'Job is running on node '; cat $PBS_NODEFILE
echo ------------------------------------------------------------------
echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: job identifier is $PBS_JOBID
echo PBS: job name is $PBS_JOBNAME
echo PBS: node file is $PBS_NODEFILE
echo PBS: current home directory is $PBS_O_HOME
echo PBS: PATH = $PBS_O_PATH
echo ------------------------------------------------------------------
NCORES='wc -w < $PBS_NODEFILE'
echo "hello-test requesting $NCORES cores..."

mpirun -np 16 -hostfile $PBS_NODEFILE -nooversubscribe ./hello
Running Jobs on a Cluster

Oversubscribing Nodes

Simple Hello World Batch script - now oversubscribing fails

Job is running on node node7
node7
node6
node6

----------------------------------------------------------------------
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 1547.tuckoo.sdsu.edu
PBS: job name is hello
PBS: node file is /var/spool/torque/aux//1547.tuckoo.sdsu.edu
PBS: current home directory is /home/mthomas
PBS: PATH = /usr/lib64/qt-3.3/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/usr/bin:/usr/lib64/openmpi/bin:/usr/local/tau/x86_64/bin:/home/mthomas/bin
----------------------------------------------------------------------

hello-test requesting 4 cores...

----------------------------------------------------------------------

There are not enough slots available in the system to satisfy the 16 slots that were requested by the application:
./hello

Either request fewer slots for your application, or make more slots available for use.

----------------------------------------------------------------------
Running Jobs on a Cluster

Passing Command Line Parameters to Batch Job

Passing command line parameters to job script

```bash
#!/bin/sh
# this example batch script requests hello processors...
# for more info on requesting specific nodes see
# "man pbs_resources"
#PBS -V
#PBS -l nodes=1:ppn=4:core4
#PBS -N $PARAM_NAME
#PBS -j oe
#PBS -q batch
#PBS -V
#PBS -l nodes=1:ppn=4:core4
#PBS -N $PARAM_NAME
#PBS -j oe
#PBS -q batch
cd $PBS_O_WORKDIR

ncores=`wc -w < $PBS_NODEFILE`

printf "hello-test using $ncores cores...
mpirun -np $PARAM_CORES ./hello
```

```bash
------------------------------------------------------
SUBMIT JOB
------------------------------------------------------
[mthomas@tuckoo hello]$ qsub -v PARAM_CORES=4,PARAM_NAME=howdy batch.hello.params
------------------------------------------------------
[mthomas@tuckoo hello]$ cat howdy.o57465
------------------------------------------------------
Job is running on node node1
node1
node1
node1
------------------------------------------------------
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 57465.tuckoo.sdsu.edu
PBS: job name is $PARAM_NAME
PBS: node file is /var/spool/torque/aux/57465.tuckoo.sdsu.edu
PBS: current home directory is /home/mthomas
PBS: PATH = /opt/mpi/linux86-64/2011/mpi/mpich/include/:/usr/lib64/
------------------------------------------------------
hello-test using 4 cores...
MyRnk[ 0]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:33:39 2014
MyRnk[ 1]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:33:39 2014
MyRnk[ 2]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:33:39 2014
MyRnk[ 3]: Total #PEs: 4 running on Node: node1 Date: Tue Sep 16 15:33:39 2014
```
Next class: 09/18/14

Topic: Introduction to MPI