A Introduction to The Center for Advanced Computing

Brock Palen
brockp@umich.edu

TBD

Outline

Hardware

Compute Hardware

- 1 Altix node, 32 cores
- 468 Opteron nodes, over 1224 cores
- 400+ nodes on CAEN Grid update this
- Gigabit networking, Myrinet networking, Infiniband networking
- Upto 96GB of memory (64GB public) for SMP work

Hardware: nyx

Nyx

- nyx is the Opteron cluster;
- Login: `nyx-login.engin.umich.edu`
- Currently has 6TB NFS file system
- Running RedHat Enterprise Linux 4

Hardware: bighouse

Bighouse

- bighouse is our Itanium SMP machine;
- Login: `bighouse.engin.umich.edu`
- Shares nyx's 6TB NFS file system
- Running SUsE Linux Enterprise Server 10
- ProPack 5 from SGI
- Only available for benchmarking (Private)

Hardware: Grid

CAEN Grid

- 400+ Nodes, Some Dual Core
- All nodes have 2GB Ram
- FAST Single CPU's
- Some Parallel Ability
- Short Jobs Only

Software

Nyx Defaults

- OpenMPI
- PGI Compilers
- PBS commands

Bighouse Defaults

- Message Passing Toolkit (MPT)
- Intel Compilers
- PBS commands

Manipulating Software

All CAC systems use `modules` to control software. Users *can* and *should* write their own modules if needed.

module commands

- `module list`
  Show loaded modules
- `module load` *`modulename`*
  Load *modulename* for use
- `module avail` *`modulename`*
  Show available versions of module *modulename*
- `module rm` *`modulename`*
  Remove currently loaded module

Module Fun

Module Customization

- `/privatemodules/default`
  Allows users to change their default modules.
- `/privatemodules/`*`module/version`*
  Holds user created module
- man modulefile

Module Example

Module Example

- List Loaded Modules

  `module list`
- Show All Modules

  `module avail`
- Show All Versions of openmpi

  `module avail openmpi`
- Load FFTW

  `module load fftw`
- Show Variables defined by FFTW

  `module show fftw`

  `echo $FFTW_LINK`

Module Example

Module Example

- List Loaded Modules

  `module list`
- Show All Modules

  `module avail`
- Show All Versions of openmpi

  `module avail openmpi`
- Load FFTW

  `module load fftw`
- Show Variables defined by FFTW

  `module show fftw`

  `echo $FFTW_LINK`

Module Example

Module Example

- List Loaded Modules

  ```
  module list
  ```

- Show All Modules

  ```
  module avail
  ```

- Show All Versions of openmpi

  ```
  module avail openmpi
  ```

- Load FFTW

  ```
  module load fftw
  ```

- Show Variables defined by FFTW

  ```
  module show fftw
  echo $FFTW_LINK
  ```

Module Example

Module Example

- List Loaded Modules

  `module list`
- Show All Modules

  `module avail`
- Show All Versions of openmpi

  `module avail openmpi`
- Load FFTW

  `module load fftw`
- Show Variables defined by FFTW

  `module show fftw`

  `echo $FFTW_LINK`

Module Example

Module Example

- List Loaded Modules

  `module list`
- Show All Modules

  `module avail`
- Show All Versions of openmpi

  `module avail openmpi`
- Load FFTW

  `module load fftw`
- Show Variables defined by FFTW

  `module show fftw`

  `echo $FFTW_LINK`

Tools

- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.
- Only use `notepad` on Windows!
- If made on windows fix with `dos2unix filename`

Tools

- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.
- Only use `notepad` on Windows!
- If made on windows fix with `dos2unix filename`

Tools

- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.
- Only use `notepad` on Windows!
- If made on windows fix with `dos2unix filename`

Compile Code

Nyx

- Use: `mpicc`, `mpiCC`, `mpif90` for MPI code
- Use: `pgcc`, `pgCC`, `pgf90` with `-mp` for OpenMP Code

Bighouse

- Use: `icc`, `icpc`, `ifort` with `-lmpi` for MPI code
- Use: `icc`, `icpc`, `ifort` with `-openmp` for OpenMP code

CAEN Grid

TBD

Compile Example

Copy Code: `cp ~brockp/mpicodes.tar ~`

```
tar -xvf mpicodes.tar
cd ~/mpicodes
```

Serial Code

- Fortran 90

  ```
  pgf90 -fastsse -o f90hello helloworld.f90
  ```

- C

  ```
  pgcc -fastsse -o chello helloworld.c
  ```

Compile Example Ctd.

MPI Code

- `make`
- `mpirun -np 2 c_ex01`
- Thats it... Ok not really
- `make clean`
- `mpicc -fastsse -o c_ex01 c_ex01.c`

Compile Example Ctd.

MPI Code

- `make`
- `mpirun -np 2 c_ex01`
- Thats it... Ok not really
- `make clean`
- `mpicc -fastsse -o c_ex01 c_ex01.c`

Compile Example Ctd.

MPI Code

- `make`
- `mpirun -np 2 c_ex01`
- Thats it... Ok not really
- `make clean`
- `mpicc -fastsse -o c_ex01 c_ex01.c`

Introduction to the PBS Batch System

PBS

- All access to the compute nodes (everything other than the login node) is via the batch system
- We use a system called Torque, it is derived from PBS
- The batch system controls access to queues
- The scheduling system decides if and where jobs can run
- There is a single public queue: `cac`
- There are many private queues for people who own or rent nodes
- If you don't know use the `route` queue

Introduction to the PBS Batch System

PBS

- All access to the compute nodes (everything other than the login node) is via the batch system
- We use a system called Torque, it is derived from PBS
- The batch system controls access to queues
- The scheduling system decides if and where jobs can run
- There is a single public queue: `cac`
- There are many private queues for people who own or rent nodes
- If you don't know use the `route` queue

Introduction to the PBS Batch System

PBS Files

The steps to using the batch system are:

1. Create a batch file: this is a short (5-15 lines) text file with some batch commands and the commands to run your program
2. Submit the file to the batch system
3. Check on the status of your job
4. Delete your job if you want to cancel it

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N 1-cpu
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```

Creating a PBS Batch File

A more complicated example:

```
#!/bin/sh
#PBS -N mcnp-8x2
#PBS -l nodes=8:ppn=2,walltime=8:00:00
#PBS -q route
#PBS -M brockp@umich.edu
#PBS -m ae
#PBS -j oe
#PBS -V
cd ${HOME}/input2/
echo "I ran on: "
cat $PBS_NODEFILE
mpirun -np 16 mcnp5.mpi i=input2 o=output2 r=restart2
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub  mcnp.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au brockp
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    brockp   short    mcnp-8x2    18922     8  --     --  08:00 R 00:00

$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub  mcnp.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au brockp
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    brockp   short    mcnp-8x2     18922     8  --     --  08:00 R 00:00

$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

Submitting, Checking, and Deleting Batch Jobs

- After you create your PBS script, you need to submit it:

```
$ qsub  mcnp.q
542.nyx-login.engin.umich.edu
```

- After you submit your script, you can check on the status of your job:

```
$ qstat -au brockp
nyx-login.engin.umich.edu:
Job ID                  Username Queue    Jobname    SessID NDS   TSK Memory Time  S Time
----------------------- -------- -------- ---------- ------ ----- --- ------ ----- - -----
542.nyx-login.engin.    brockp   short    mcnp-8x2     18922     8  --    --  08:00 R 00:00

$ checkjob 542
[... lots of output ...]
```

- If you want to delete your job:

```
$ qdel 542
```

PBS Email

PBS will send an email at the start and end of your job if you use the `-m` and `-M` options in your PBS script. The email after a job completes successfully looks like:

```
Date: Sun, 30 Apr 2006 12:50:17 -0400
From: adm <adm@nyx-login.engin.umich.edu>
To: "Palen, Brock E" <brockp@umich.edu>
Subject: PBS JOB 542.nyx-login.engin.umich.edu
---------------------------------------

PBS Job Id: 542.nyx-login.engin.umich.edu
Job Name:   mcnp-8x2
Execution terminated
Exit_status=0
resources_used.cput=13:17:26
resources_used.mem=1220672kb
resources_used.vmem=11146704kb
resources_used.walltime=00:49:57
```

PBS Example

PBS Example Job

```
cd  /mpicodes
nano run
Edit #PBS -M
Ctl+o
Ctl+x
qsub run
```

Interactive Jobs

The CAC has 8 CPU's for jobs 15 minutes or less

These CPU's can be used for testing PBS scripts and debugging code

Interactive jobs allow users to interact with the shell on a remote node

```
qsub -I -l nodes=2:ppn=2,walltime=15:00 -q cac
```

PBS Example

PBS Example Job

```
cd  /mpicodes
nano run
Edit #PBS -M
Ctl+o
Ctl+x
qsub run
```

Interactive Jobs

The CAC has 8 CPU's for jobs 15 minutes or less

These CPU's can be used for testing PBS scripts and debugging code

Interactive jobs allow users to interact with the shell on a remote node

```
qsub -I -l nodes=2:ppn=2,walltime=15:00 -q cac
```

PBS Example

PBS Example Job

```
cd  /mpicodes
nano run
Edit #PBS -M
Ctl+o
Ctl+x
qsub run
```

Interactive Jobs

The CAC has 8 CPU's for jobs 15 minutes or less

These CPU's can be used for testing PBS scripts and debugging code

Interactive jobs allow users to interact with the shell on a remote node

```
qsub -I -l nodes=2:ppn=2,walltime=15:00 -q cac
```

PBS Example

PBS Example Job

```
cd  /mpicodes
nano run
Edit #PBS -M
Ctl+o
Ctl+x
qsub run
```

Interactive Jobs

The CAC has 8 CPU's for jobs 15 minutes or less

These CPU's can be used for testing PBS scripts and debugging code

Interactive jobs allow users to interact with the shell on a remote node

```
qsub -I -l nodes=2:ppn=2,walltime=15:00 -q cac
```

Understanding the Scheduler

The scheduler determines what jobs can run, when they can run, and where. There are many factors that go into the scheduler's decision.

Limits and Priority

- Maximum number jobs eligible for scheduling: 4
- Maximum number of CPUs in use by one person: depends on queue
- Maximum number of jobs in the queue at one time: no limit

- Who you are: user and group level priorities
- How long you've waited: the longer you wait, the higher your priority
- Your recent usage (fairshare): People with less usage over the past month will have a higher priority than those with a lot of usage

Understanding the Scheduler

The scheduler determines what jobs can run, when they can run, and where. There are many factors that go into the scheduler's decision.

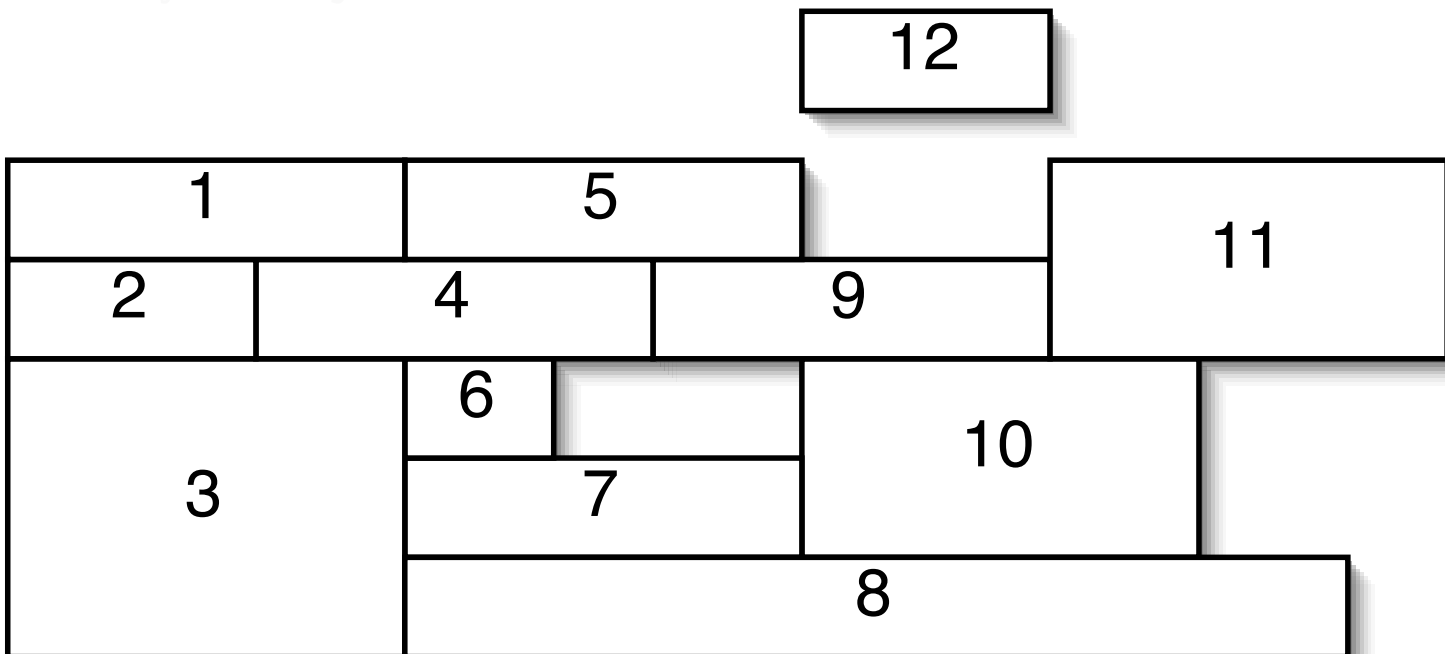
Limits and Priority

- Maximum number jobs eligible for scheduling: 4
- Maximum number of CPUs in use by one person: depends on queue
- Maximum number of jobs in the queue at one time: no limit

- Who you are: user and group level priorities
- How long you've waited: the longer you wait, the higher your priority
- Your recent usage (fairshare): People with less usage over the past month will have a higher priority than those with a lot of usage

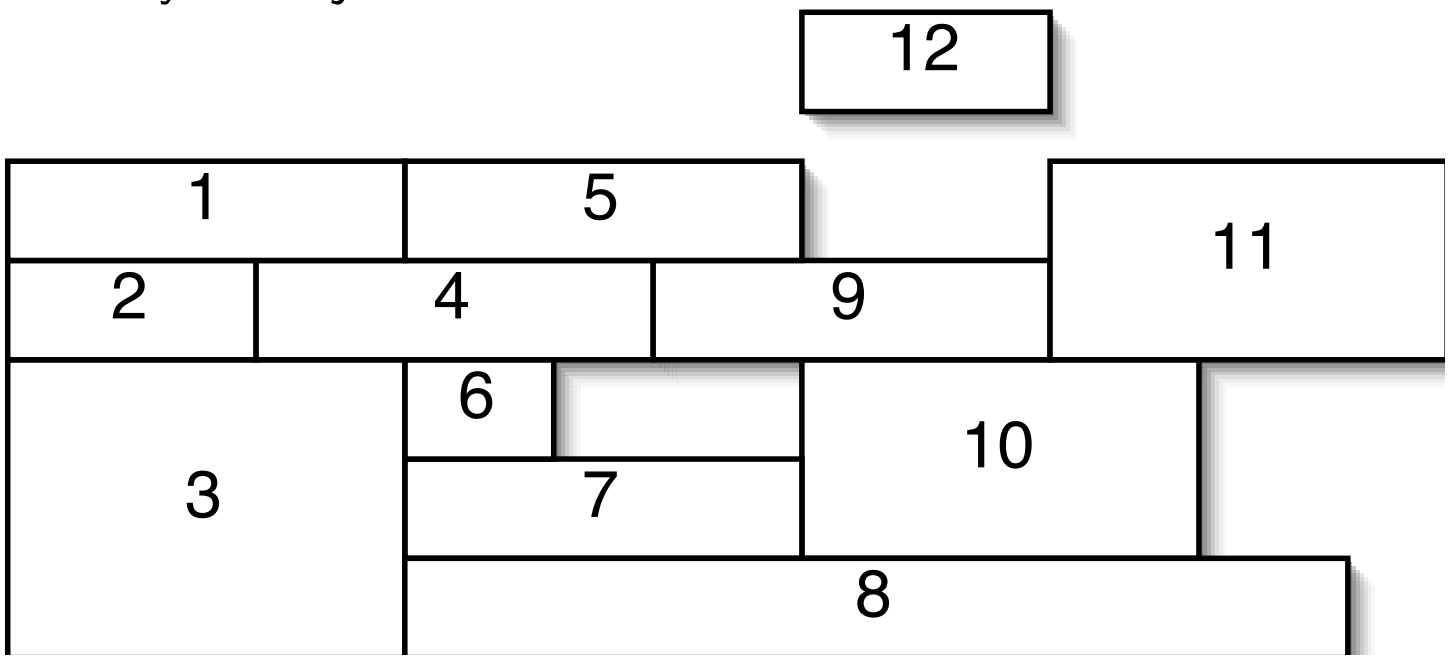
Understanding the Scheduler

- Reservations
  - Advance reservations: holds nodes for users or groups
  - Job reservations: scheduler will reserve nodes for the next several jobs in each queue
- Backfill
  - If the reservations leave holes in the schedule, they may be filled by short jobs that otherwise would have waited.



Understanding the Scheduler

- Reservations
  - Advance reservations: holds nodes for users or groups
  - Job reservations: scheduler will reserve nodes for the next several jobs in each queue
- Backfill
  - If the reservations leave holes in the schedule, they may be filled by short jobs that otherwise would have waited.



Understanding the Scheduler

There are several commands that can give you insight into the scheduler's decisions.

- `showq` — shows the state of the queue at that moment in time, showing the running jobs in order of soonest to finish to longest to finish; the idle jobs in order of priority; and the blocked jobs in the order they were submitted
- `diagnose -p` — shows the factors that go into computing the priority for all of the idle jobs
- `checkjob` *`jobnumber`* — for idle jobs this will show why the job can't start
- `showstart` *`jobnumber`* — this makes a (poor) estimate of when the job will start

Summary

- Resources
  - Lots of CPUs
  - A reasonable amount of software
  - Watch or subscribe to `http://cac.engin.umich.edu` for updates
- Access
  - All access is via the SSH family of commands: `ssh`, `sftp`, `scp`
  - There are lots of clients for these commands for the different platforms
  - There is no graphical access, everything is via the command line

Summary

- Job Submission
  - Every job needs a PBS script file
  - Two most important commands: `qsub` and `qstat -au` *uniqname*
- Job Scheduling
  - Scheduling depends on a lot of factors, it is best to submit jobs and let the scheduler optimize for their start.

Summary

- News: `http://cac.engin.umich.edu`
  - RSS feed
  - New of changes, outages, other pertinent piece of information
- Contact: `cac-support@umich.edu`
  - Questions or concerns should be sent here (not to an individual) since this is read by six people. The odds of a quick reply are best this way.
  - We aren't parallel programmers, but we'll do what we can to help.