An Introduction to
The Center for Advanced Computing

Brock Palen
brockp@umich.edu

TBD

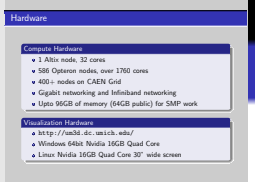
An Introduction to The Center for Advanced Computing

Brock Palen
brockp@umich.edu

TBD

Outline

1. Resources
   - Hardware
   - Default Software
2. Mechanics: Usage
   - Compiling programs
   - The Batch System
3. The Scheduler
   - Understanding the Scheduler
   - Scheduler Commands
4. Summary
   - Resources and Access
   - Job Management
   - Contact



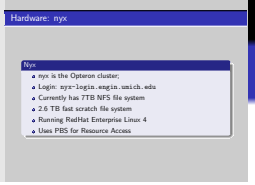
Hardware

Compute Hardware

- 1 Altix node, 32 cores
- 586 Opteron nodes, over 1760 cores
- 400+ nodes on CAEN Grid
- Gigabit networking and Infiniband networking
- Upto 96GB of memory (64GB public) for SMP work

Visualization Hardware

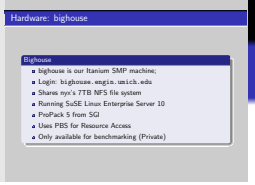
- `http://um3d.dc.umich.edu/`
- Windows 64bit Nvidia 16GB Quad Core
- Linux Nvidia 16GB Quad Core 30" wide screen



Hardware: nyx

Nyx

- nyx is the Opteron cluster;
- Login: `nyx-login.engin.umich.edu`
- Currently has 7TB NFS file system
- 2.6 TB fast scratch file system
- Running RedHat Enterprise Linux 4
- Uses PBS for Resource Access

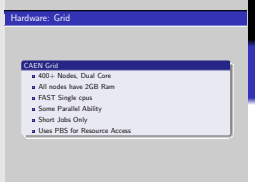


Hardware: bighouse

Bighouse: Available to Aero Space Dept

Bighouse

- bighouse is our Itanium SMP machine;
- Login: `bighouse.engin.umich.edu`
- Shares nyx's 7TB NFS file system
- Running SuSE Linux Enterprise Server 10
- ProPack 5 from SGI
- Uses PBS for Resource Access
- Only available for benchmarking (Private)

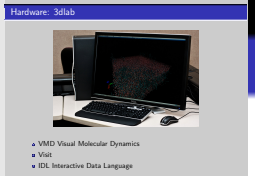


Hardware: Grid

nodes are great for paramater sweaps, hundereds of jobs etconly for engin accounts

CAEN Grid

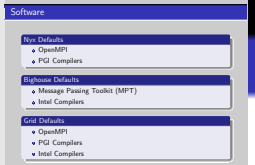
- 400+ Nodes, Dual Core
- All nodes have 2GB Ram
- FAST Single cpus
- Some Parallel Ability
- Short Jobs Only
- Uses PBS for Resource Access



1. VMD `http://www.ks.uiuc.edu/Research/vmd/`
2. Visit `https://wci.llnl.gov/codes/visit/`
3. IDL `http://www.ittvis.com/idl/`

Hardware: 3dlab

- VMD Visual Molecular Dynamics
- Visit
- IDL Interactive Data Language



Software

Nyx Defaults

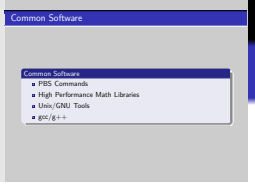
- OpenMPI
- PGI Compilers

Bighouse Defaults

- Message Passing Toolkit (MPT)
- Intel Compilers

Grid Defaults

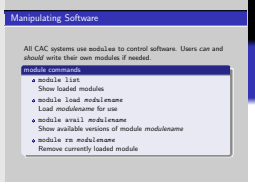
- OpenMPI
- PGI Compilers
- Intel Compilers



Common Software

Common Software

- PBS Commands
- High Performance Math Libraries
- Unix/GNU Tools
- gcc/g++



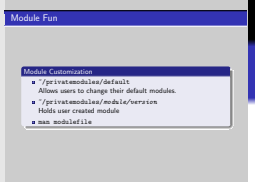
1. Show Example

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

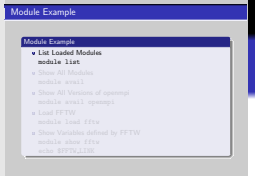


Module Fun

1. example using fftw follows

Module Customization

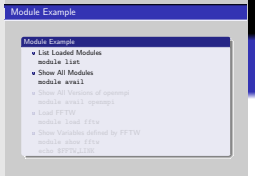
- `~/privatemodules/default`
  Allows users to change their default modules.
- `~/privatemodules/`*`module/version`*
  Holds user created module
- `man modulefile`



Module Example

Module Example

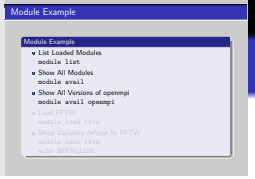
- List Loaded Modules
  `module list`
- Show All Modules
  `module avail`
- Show All Versions of openmpi
  `module avail openmpi`
- Load FFTW
  `module load fftw`
- Show Variables defined by FFTW
  `module show fftw`
  `echo $FFTW_LINK`



Module Example

Module Example

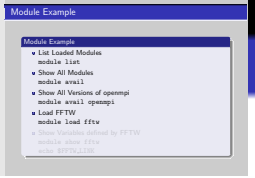
- List Loaded Modules
  `module list`
- Show All Modules
  `module avail`
- Show All Versions of openmpi
  `module avail openmpi`
- Load FFTW
  `module load fftw`
- Show Variables defined by FFTW
  `module show fftw`
  `echo $FFTW_LINK`



Module Example

Module Example

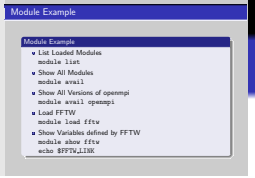
- List Loaded Modules
  `module list`
- Show All Modules
  `module avail`
- Show All Versions of openmpi
  `module avail openmpi`
- Load FFTW
  `module load fftw`
- Show Variables defined by FFTW
  `module show fftw`
  `echo $FFTW_LINK`



Module Example

Module Example

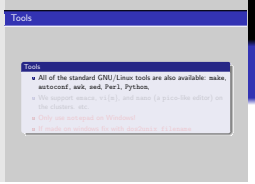
- List Loaded Modules
  `module list`
- Show All Modules
  `module avail`
- Show All Versions of openmpi
  `module avail openmpi`
- Load FFTW
  `module load fftw`
- Show Variables defined by FFTW
  `module show fftw`
  `echo $FFTW_LINK`



Module Example

Module Example

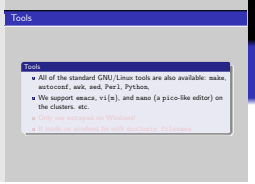
- List Loaded Modules
  `module list`
- Show All Modules
  `module avail`
- Show All Versions of openmpi
  `module avail openmpi`
- Load FFTW
  `module load fftw`
- Show Variables defined by FFTW
  `module show fftw`
  `echo $FFTW_LINK`



Tools

Tools

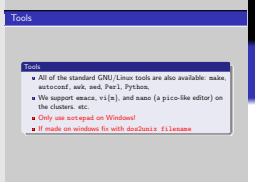
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.
- Only use `notepad` on Windows!
- If made on windows fix with `dos2unix filename`



Tools

Tools

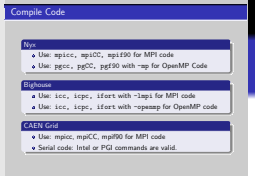
- All of the standard GNU/Linux tools are also available: `make`, `autoconf`, `awk`, `sed`, `Perl`, `Python`,
- We support `emacs`, `vi{m}`, and `nano` (a `pico`-like editor) on the clusters. etc.
- Only use `notepad` on Windows!
- If made on windows fix with `dos2unix filename`



Tools

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`



1. The following applies to the default modules
2. Grid: both compilers support OpenMP

Compile Code

Nyx

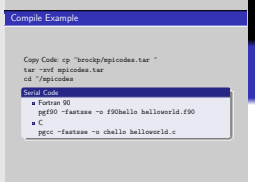
- Use: `mpicc`, `mpiCC`, `mpif90` for MPI code
- Use: `pgcc`, `pgCC`, `pgf90` with `-mp` for OpenMP Code

Bighouse

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

CAEN Grid

- Use: mpicc, mpiCC, mpif90 for MPI code
- Serial code: Intel or PGI commands are valid.



Compile Example

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

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

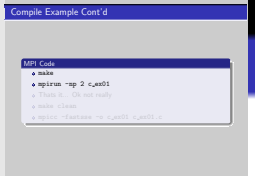
Serial Code

- Fortran 90

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

- C

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

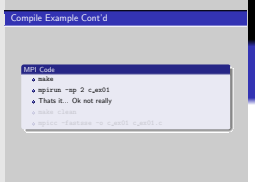


Compile Example Cont'd

1. 'man make' Make lets you manage large bits of code. Works for all source types

MPI Code

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

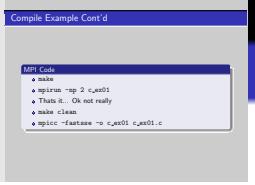


Compile Example Cont'd

1. 'man make' Make lets you manage large bits of code. Works for all source types

MPI Code

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

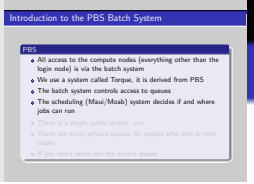


Compile Example Cont'd

1. 'man make' Make lets you manage large bits of code. Works for all source types

MPI Code

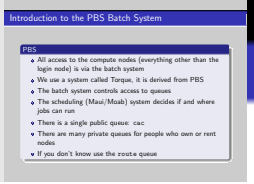
- `make`
- `mpirun -np 2 c_ex01`
- Thats it... Ok not really
- `make clean`
- `mpicc -fastsse -o c_ex01 c_ex01.c`



Introduction to the PBS Batch System

PBS

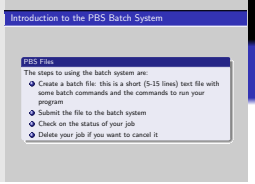
- All access to the compute nodes (everything other than the login node) is via the batch system
- We use a system called Torque, it is derived from PBS
- The batch system controls access to queues
- The scheduling (Maui/Moab) system decides if and where jobs can run
- There is a single public queue: `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 (Maui/Moab) 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

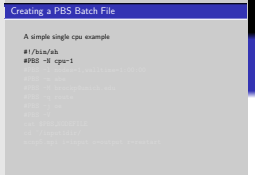


Introduction to the PBS Batch System

PBS Files

The steps to using the batch system are:

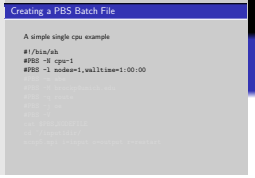
1. Create a batch file: this is a short (5-15 lines) text file with some batch commands and the commands to run your program
2. Submit the file to the batch system
3. Check on the status of your job
4. Delete your job if you want to cancel it



Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N cpu-1
#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

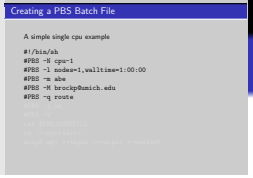
A simple single cpu example

```
#!/bin/sh
#PBS -N cpu-1
#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 cpu-1
#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

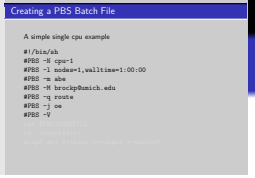
A simple single cpu example

```
#!/bin/sh
#PBS -N cpu-1
#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

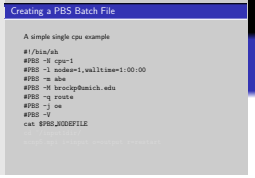
```
#!/bin/sh
#PBS -N cpu-1
#PBS -l nodes=1,walltime=1:00:00
#PBS -m abe
#PBS -M brockp@umich.edu
#PBS -q route
#PBS -j oe
#PBS -V
cat $PBS_NODEFILE
cd ~/input1dir/
mcnp5.mpi i=input o=output r=restart
```



Creating a PBS Batch File

A simple single cpu example

```
#!/bin/sh
#PBS -N cpu-1
#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 cpu-1
#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

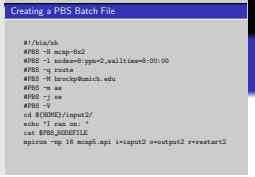
A simple single cpu example

#!/bin/sh
#PBS -N cpu-1
#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 cpu-1
#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
```

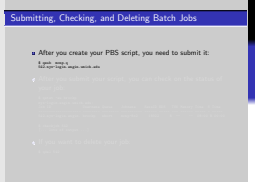


Creating a PBS Batch File

See `man qsub` for all optionsExample pbs file included with example codes `run`

1. -N sets the job name, can not start with a number
2. -l sets the resources, walltime=HH:MM:SS or walltime=SSSSS Total number of cpus is nodes*ppn which must email -np.
3. -q optional: which queue to submit to, use default: `route`
4. -M Who to email, can be more than one address
5. -m when to email a=abort, b=begin, e=end
6. -j optional: join STDOUT and STDERR default is to not
7. -V Copy submit environment to compute environtment, ALWAYS use this

```
#!/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
```

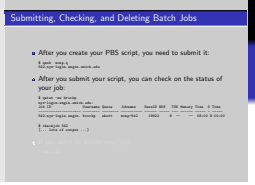
- After you submit your script, you can check on the status of your job:

```
$ 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
```

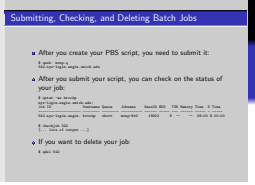
- After you submit your script, you can check on the status of your job:

```
$ 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
```


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
```

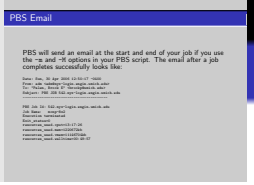
- After you submit your script, you can check on the status of your job:

```
$ 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
```

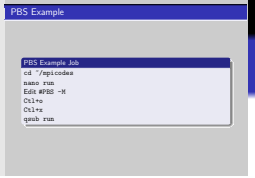


PBS Email

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
```

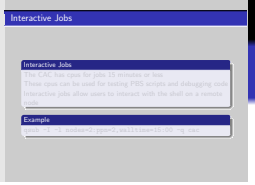


PBS Example

Please be sure to edit the email address I don't want to be getting all your mailNano is a clone of pico

PBS Example Job

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



Interactive Jobs

Interactive jobs can use X11 forwarding also

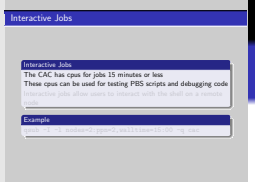
1. `qsub -I -X nodes=2:ppn=2,walltime=15:00 -q route`
2. Used with ddt our parallel debugger

Interactive Jobs

The CAC has cpus for jobs 15 minutes or less
These cpus can be used for testing PBS scripts and debugging code
Interactive jobs allow users to interact with the shell on a remote node

Example

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



Interactive Jobs

Interactive jobs can use X11 forwarding also

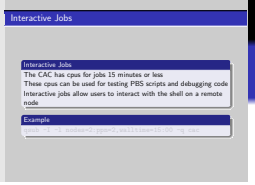
1. `qsub -I -X nodes=2:ppn=2,walltime=15:00 -q route`
2. Used with ddt our parallel debugger

Interactive Jobs

The CAC has cpus for jobs 15 minutes or less
These cpus can be used for testing PBS scripts and debugging code
Interactive jobs allow users to interact with the shell on a remote node

Example

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



Interactive Jobs

Interactive jobs can use X11 forwarding also

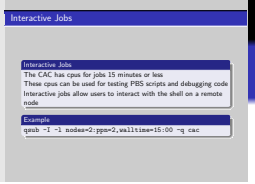
1. `qsub -I -X nodes=2:ppn=2,walltime=15:00 -q route`
2. Used with ddt our parallel debugger

Interactive Jobs

The CAC has cpus for jobs 15 minutes or less
These cpus can be used for testing PBS scripts and debugging code
Interactive jobs allow users to interact with the shell on a remote node

Example

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



Interactive Jobs

Interactive jobs can use X11 forwarding also

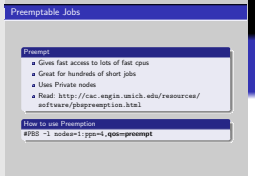
1. `qsub -I -X nodes=2:ppn=2,walltime=15:00 -q route`
2. Used with ddt our parallel debugger

Interactive Jobs

The CAC has cpus for jobs 15 minutes or less
These cpus can be used for testing PBS scripts and debugging code
Interactive jobs allow users to interact with the shell on a remote node

Example

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



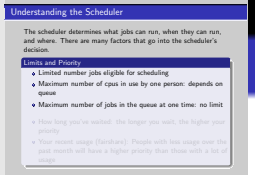
Preemptable Jobs

Preempt

- Gives fast access to lots of fast cpus
- Great for hundreds of short jobs
- Uses Private nodes
- Read: `http://cac.engin.umich.edu/resources/software/pbspreemption.html`

How to use Preemption

`#PBS -l nodes=1:ppn=4,`**`qos=preempt`**



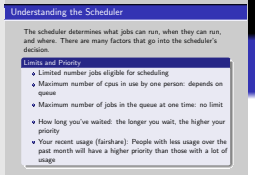
1. We can do priorities and limits in private queues as needed for those queues. Limits on User, group, hardware in use, time of use walltime are all options

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

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



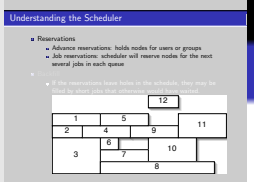
1. We can do priorities and limits in private queues as needed for those queues. Limits on User, group, hardware in use, time of use walltime are all options

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.

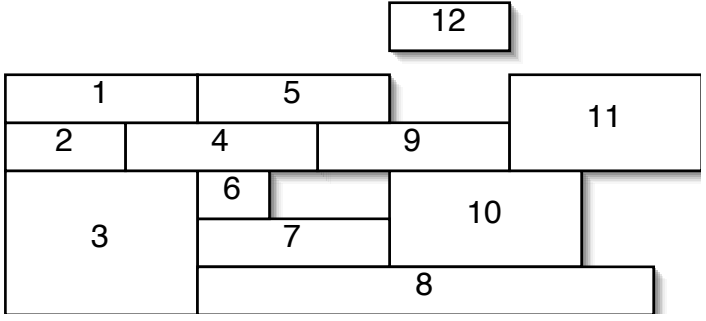
Limits and Priority

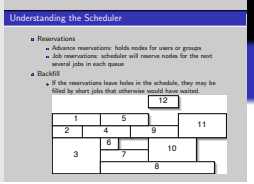
- Limited number jobs eligible for scheduling
- Maximum number of cpus in use by one person: depends on queue
- Maximum number of jobs in the queue at one time: no limit
- How long you've waited: the longer you wait, the higher your priority
- Your recent usage (fairshare): People with less usage over the past month will have a higher priority than those with a lot of usage



Understanding the Scheduler

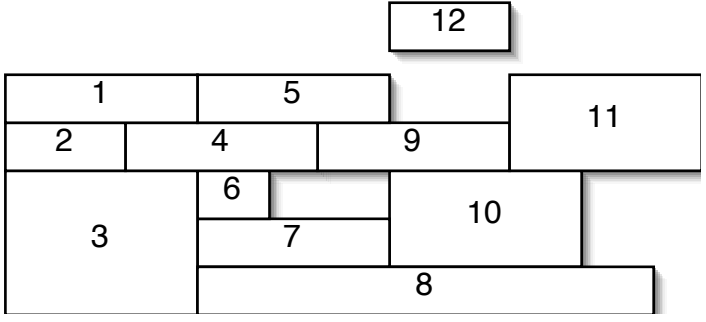
- Reservations
  - Advance reservations: holds nodes for users or groups
  - Job reservations: scheduler will reserve nodes for the next several jobs in each queue
- Backfill
  - If the reservations leave holes in the schedule, they may be filled by short jobs that otherwise would have waited.

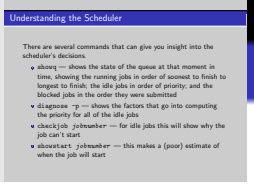




Understanding the Scheduler

- 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.

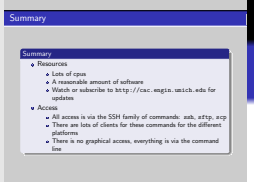




Understanding the Scheduler

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

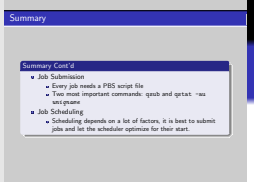
- `showq` — shows the state of the queue at that moment in time, showing the running jobs in order of soonest to finish to longest to finish; the idle jobs in order of priority; and the blocked jobs in the order they were submitted
- `diagnose -p` — shows the factors that go into computing the priority 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

Summary

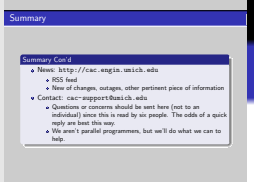
- Resources
  - Lots of cpus
  - A reasonable amount of software
  - Watch or subscribe to `http://cac.engin.umich.edu` for updates
- Access
  - All access is via the SSH family of commands: `ssh`, `sftp`, `scp`
  - There are lots of clients for these commands for the different platforms
  - There is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



Summary

Summary Cont'd

- 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

Summary Con'd

- 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.