Effective Use of CCV Resources

Mark Howison

User Services & Support
This talk...

- Assumes you have some familiarity with a Unix shell
- Provides examples and best practices for typical usage of CCV systems
- Is a condensed form of the documentation available at:

  http://www.brown.edu/Departments/CCV/doc
Overview

- Connecting to CCV
- Transferring files
- Available software
- Compiling and linking your own code
- Running and monitoring jobs
Logging in

- CCV uses the Secure Shell (SSH) protocol
- You will need an SSH client
  - Linux / OS X comes with a command-line client
  - Windows: try PuTTY or Cygwin
- Connect to the “ssh” server
  - Linux / OS X / Cygwin: `ssh username@ssh.ccv.brown.edu`
  - PuTTY: enter `ssh.ccv.brown.edu` in “Host Name”
Virtual Network Computing

- OR connect through CCV's Virtual Network Computing client, available here:
  - http://www.brown.edu/Departments/CCV/vnc
Transferring files

- Mount your RData directory (~/data on Oscar) with Samba to use Windows Explorer or Mac Finder
- Or from a terminal, use the `scp` command:
  - Copy to Oscar:
    ```
    scp username@ssh.ccv.brown.edu:/remote/path /local/path
    ```
  - Copy from Oscar:
    ```
    scp /local/path username@ssh.ccv.brown.edu:/remote/path
    ```
- Or use a GUI “Secure Copy” program
  - e.g. WinSCP, Fugu (Mac)
Available software

- Software is organized using **Environment Modules**
- Load a software module with `module load <name>`
- Loading a module alters your environment, paths, etc:

```
[mhowison@login001 ~]$ module display intel
-------------------------------------------------------------------
/gpfs/runtime/modulefiles/intel/12.0.4:
setenv MKL -L/gpfs/runtime/opt/intel/12.0.4/mkl/lib/intel64
-1mkl_rt -liomp5 -lpthread
prepend-path PATH /gpfs/runtime/opt/intel/12.0.4/bin
prepend-path MANPATH /gpfs/runtime/opt/intel/12.0.4/man/en_US
prepend-path LD_LIBRARY_PATH /gpfs/runtime/opt/intel/12.0.4/lib/intel64:/gpfs/runtime/opt/intel/12.0.4/mkl/lib/intel64
...

[mhowison@login001 ~]$ module load intel

[mhowison@login001 ~]$ which icc
/gpfs/runtime/opt/intel/12.0.4/bin/icc
```
Available software (Cont'd)

- View available modules with `module avail`
  - Or search for a specific package with `module avail <package>`

- View your loaded modules with `module list`

- Module commands in your `~/.modules` file will automatically execute when you login
  - Add a module to your default list with `echo "module load <name>" >> ~/.modules`

- If you need software that is not installed, submit a request at [http://www.brown.edu/Departments/CCV/protected/software](http://www.brown.edu/Departments/CCV/protected/software)
Compiling

- By default, the GNU compiler suite is loaded
  - gcc (C), g++ (C++), and gfortran (Fortran)
- Portland Group (PGI) compilers are available with
  module load pgi/11.4
  - pgcc (C), pgCC (C++), and pgf90 (Fortran)
- Intel compiler suite (with MKL) is available with
  module load intel
  - icc (C), icpc (C++), and ifort (Fortran)
To use MPI with the PGI or Intel compilers, you must swap out the `openmpi` module!

```bash
module swap openmpi openmpi/1.4.3-pgi
or
module swap openmpi openmpi/1.4.3-intel
```

Otherwise, `mpicc` will point to the wrong compiler.
Linking

> Many modules include environment variable shortcuts that contain linking directions, e.g.

**FFTW**

gcc -o fftw-app fftw-app.c $FFTW

$FFTW = -I/gpfs/runtime/opt/fftw/3.2.2/include
-L/gpfs/runtime/opt/fftw/3.2.2/lib -lfftw3

**GotoBLAS**

gcc -o blas-app blas-app.c $GOTO

$GOTO = -I/gpfs/runtime/opt/gotoblas2/1.13/include
-L/gpfs/runtime/opt/gotoblas2/1.13/lib -lgoto2
-lpthread -lgfortran
Running jobs

- An **interactive** job allows you to:
  - View the output of a program as it runs
  - Interact with a program by typing input, using a GUI, etc.
  - Quickly stop and restart a program, e.g. to test out different input or for debugging
  - Run on a system shared by other users

- A **batch** job allows you to:
  - Submit a script that will run without any intervention
  - Access dedicated resources for your job
  - Run for long periods of time without worrying about other users interfering with your job
Interactive jobs

- When you log into `ssh.ccv.brown.edu`, you will be forwarded to either `login001` or `login002`
  - The login nodes are intended for tasks like:
    - writing, compiling, and debugging code
    - transferring and managing files
    - submitting and managing batch jobs
  - They are a shared resource with many other users concurrently logged in
  - Please don't run large-memory or CPU-intensive programs on these nodes!
Interactive jobs (Cont'd)

- Instead, ssh from the login node to:
  - smp007, smp008, smp009
    - 128GB of memory
    - Use `smpstatus` to check which have a lighter load
  - gpu001, gpu002
    - 24GB of memory, 2x NVIDIA M2050 Fermi cards
    - Identical to the Oscar GPU compute nodes

- Or connect with the VNC client to:
  - vnc001, vnc002
    - 128GB of memory
    - Can run GUI software without X forwarding
Interactive jobs (Cont'd)

- Or you can request nodes interactively from any of the queues!
  - Use the `-I` option of `qsub`

```
[mhowison@login001 ~]$ qsub -I -q debug -l walltime=30:00, nodes=2
qsub: waiting for job 1272761.mgt to start
qsub: job 1272761.mgt ready

Begin PBS Prologue Tue Jul 19 15:42:15 EDT 2011 1311104535
Job ID: 1272761.mgt
Username: mhowison
Group: ccvstaff
Nodes: node181 node184
End PBS Prologue Tue Jul 19 15:42:15 EDT 2011 1311104535
```

```
[mhowison@node184 ~]$
```
Interactive jobs (Cont'd)

- Put long-running jobs in the background using:
  
  ```
  ./myprogram 2>&1 1>mylogfile &
  ```

  - The `2>&1` redirects stderr to stdout
  - The `1>mylogfile` redirects stdout to a log file that you can review later
  - The final `&` backgrounds the job

- Otherwise, your job may stop if your ssh connection is interrupted

- Use `ps x` to see all jobs you have running on the node

- OR: use a batch script! (Unless you need more memory than available on the Oscar compute nodes.)
Batch jobs

- Specify resource requirements and the commands to run in a file called a **batch script**

```bash
#!/bin/bash
#PBS -N jobname
#PBS -r n
#PBS -j oe
#PBS -l nodes=2:ppn8,walltime=00:30:00

# execute an MPI program
NPROCS=`wc -l < $PBS_NODEFILE`
mpirun -np $NPROCS mpiprogram <args>
```

- Submit the script to the **queue** with `qsub <script>`
Some useful Torque options:

#PBS -j oe

Combine your stdout and stderr outputs into a single log file (by default called <jobname>.o<jobid> in the directory where you submitted the script)

#PBS -m abe
#PBS -M oscar_user@brown.edu

Send mail if your job aborts, and when it begins and ends

#PBS -V

Import your current environment when launching the script on the head node
Batch jobs (Cont'd)

- Packing 8 serial jobs onto an 8-core Oscar node:

```bash
#!/bin/bash
#PBS -N myserialjobs
#PBS -r n
#PBS -j oe
#PBS -l nodes=1,walltime=00:30:00

# The "&" causes each program to be run as a background process
# The "wait" causes the batch script interpreter to wait until all
# background processes have completed before continuing.

myprogram args1 &
myprogram args2 &
myprogram args3 &
myprogram args4 &
myprogram args5 &
myprogram args6 &
myprogram args7 &
myprogram args8 &
wait
```
## Available queues on Oscar

- **Choosing a queue:**
  - Default is the `dq` queue
  - At submit time with `qsub -q <name>`
  - In your batch script with `#PBS -q <name>`

<table>
<thead>
<tr>
<th>Queue</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dq</code></td>
<td>200 x (8-core compute nodes)</td>
</tr>
<tr>
<td><code>gpu</code></td>
<td>42 x (8-core + 2 GPU compute nodes)</td>
</tr>
<tr>
<td><code>serial</code></td>
<td>Packs serials jobs onto 3 x 16-core smp nodes with 64GB of memory; must specify a memory limit with <code>#PBS -l mem=?gb</code></td>
</tr>
<tr>
<td><code>debug</code></td>
<td>4 Oscar compute nodes available for fast turn-around; time limit is 40 node-minutes</td>
</tr>
</tbody>
</table>
Job arrays

- Torque has a special feature for parameter sweeps
  - I.e., you want to run (“sweep”) the same program over a set of varying parameters

- Use `#PBS -t <range>`
  - Where range is of the form `0-N` or `0,2,4-8` etc.
  - A sub job with the name `<jobname>[i]` is created for each value in the range
  - In each sub job, the `$PBS_ARRAYID` environment variable is set to a different value of the range
Job arrays

- This job will use 4 nodes (32 cores) to sweep over 32 parameters:

```
#!/bin/bash
#PBS -N MySweep
#PBS -t 0-3
#PBS -l nodes=1,walltime=1:00:00
# 4 IDs (0-3) x 1 node per sub job = 4 nodes in total

HOST=`hostname`
# there are 8 cores per node
PPN=8
# so the ID needs to be multiplied by 8 to yield all 32 parameters
ID=$((PPN*PBS_ARRAYID))
for i in `seq $PPN`; do
    echo "Starting job $ID on $HOST"
    ./myprogram $ID
    ID=$((ID+1))
done
wait
```
Monitoring jobs

- See the status of all queues with `allq`
  - Hint: this list can be long, but you can scroll through it with `allq | more`
- Or, see only a specific queue with `allq <queue>`
- See a summary of your jobs with `myq`
- See another users jobs with `myq <username>`
- Alter a job with `qalter <options> <jobid>`
- Delete a job with `qdel <jobid>`