Introduction to HokieSpeed

CMDA 3634: Comp Sci Foundations for CMDA

Justin Krometis, Advanced Research Computing, 3 March 2016

1 Hardware

HokieSpeed is a large-scale CPU/GPU cluster built with an NSF grant in 2012. Each HokieSpeed node (computer) is outfitted with 24 GB of memory, two six-core Xeon E5645 CPUs and two NVIDIA M2050 / C2050 GPU. HokieSpeed is managed and maintained by Advanced Research Computing at Virginia Tech. Instructions and examples for using HokieSpeed are available on ARC's website: http://www.arc.vt.edu/hokiespeed

2 Login

1. Log into the HokieSpeed login node with your PID and password:
   (a) Mac/Linux: Open a terminal and type
   ```
   ssh YOURPID@hokiespeed2.arc.vt.edu
   ```
   (b) Windows: Download PuTTY and in the Host Name field type: `YOURPID@hokiespeed2.arc.vt.edu`
2. Copy over the files for this session to your directory:
   ```
   cp /home/jkrometi/cmda3634/* .
   ```

3 Software Stack

Software on the HokieSpeed cluster is managed with a module environment. There are three layers of modules:

1. Base: Require neither a compiler or MPI stack.
2. Compiler: Require that a compiler (e.g. `gcc` or `intel`) be loaded.
3. MPI: Require that an MPI stack (e.g. `mvapich2`, `openmpi`, or `impi`) be loaded.

Some key commands include:

Command	Example	Meaning
`module list`	`module list`	List the modules currently loaded
`module avail`	`module avail`	List modules that are available to be loaded
`module show`	`module show cuda`	Print information about a module
`module load`	`module load cuda`	Load a module
`module unload`	`module unload cuda`	Unload a module
`module spider`	`module spider cuda`	Search for a module
`module swap`	`module swap intel gcc`	Replace one module with another & reload dependencies
`module purge`	`module purge`	Unload all modules

4 Submitting a Job

Before you can do computationally-intensive work on HokieSpeed, you need to move from the login node (a computer where lots of people may be doing basic tasks) to a compute node (a computer dedicated to computing). HokieSpeed is a shared resource, so this is done by submitting a job to a schedule, a piece of software that tries to balance many users' needs. Note: This sometimes means that you will have to wait until the resources that you need are available (i.e. until other users are done)!

There are two main types of jobs: interactive and batch.

4.1 Interactive Job

You can use an interactive job to get a short(-ish) interactive session on a compute node. To do this, use the development queue (`dev_q`) as follows (here we request one node for one hour):

```
qsub -I -W group_list=hokiespeed -q dev_q -l nodes=1:ppn=12 -l walltime=1:00:00
```

4.1.1 Hands on

In this example, we will compile and run an MPI "Hello World" program. For today's class (March 3, 2016), a script to start an interactive job is included in `interactiveMar03.sh`.

1. Request an interactive job:

   ```
   ./interactiveMar03.sh
   ```

2. Load a compiler and MPI stack (feel free to substitute `intel` for `gcc` or `mvapich2` for `openmpi`):

   ```
   module purge; module load gcc openmpi
   ```

3. The MPI "Hello World" C code is in `mpi_hello_world.c`. Compile it:

   ```
   mpicc mpi_hello_world.c -o mpihw
   ```

   Here the resulting executable is called `mpihw`.

4. Run it:

   ```
   mpiexec -np $PBS_NP ./mpihw
   ```

   The variable `$PBS_NP` just refers to the number of cores available to the job. If your interactive job requested one node, `$PBS_NP` is 12.

4.2 Batch Job

Batch jobs are used to launch non-interactive sessions to run larger and longer computationally-intensive programs. These jobs are typically submitted via a *submission script*. The submission script has two main parts:

1. The header, which describes the job, e.g. how many resources it needs and for how long. These lines begin with `#PBS`.
2. The body, which describes what to do once those resources have been obtained.

Here is an example submission script for submitting a two-node job for three hours to the production queue (`normal_q`), an to run the MPI program `mpiProg` compiled against `gcc` and `openmpi` on those resources:

```
#!/bin/bash
#PBS -l walltime=03:00:00
#PBS -l nodes=2:ppn=12
#PBS -W group_list=hokiespeed
#PBS -q normal_q

#Load modules
module purge; module load gcc openmpi

#Change to the directory from which the job was submitted
cd $PBS_O_WORKDIR

echo "Run␣mpiProg␣with␣$PBS_NP␣cores!"
mpiexec -np $PBS_NP ./mpiProg

exit;
```

4.2.1 Submitting and Checking a Batch Job

This script can be submitted to the scheduler as follows (where `JobScript.sh` is the name of the script):

```
qsub JobScript.sh
```

This will return your job name of the form

```
176618.master.cluster
```

Here `176618` is the job number. Once a job is submitted to a queue, it will wait until requested resources are available within that queue, and will then run if eligible. Eligibility to run is influenced by the resource policies in effect for the queue.

To check a job's status, use the `checkjob` command:

```
checkjob -v 176618
```

You can also view all of your jobs using the `showq` or `qstat` commands:

```
showq -u $USER
qstat -u $USER
```

To check resource usage on the nodes available to a running job, use:

```
jobload 176618
```

4.2.2 Results

When your job has finished running, any outputs to stdout or stderr will be placed in the files .o and .e. These two files will be in the directory that you submitted the job from. For example, for a job submitted from `JobScript.sh` and with job ID `176618`, the output would be in:

```
JobScript.sh.o176618 #Output will be here
JobScript.sh.e176618 #Any errors will be here
```

4.2.3 Hands on

1. Edit `hs_sample.qsub` to run your MPI "Hello World" executable (using, e.g. `emacs` or `vim`)
2. Submit the submission script to the scheduler:

   ```
   qsub hs_sample.qsub
   ```

3. Check the job's status as mentioned above
4. Once the job is complete, view its results as described above
5. Try compiling and running your own MPI code!

5 References

- http://www.arc.vt.edu/hokiespeed describes HokieSpeed's hardware, usage policies, and software, and provides some step-by-step examples
- http://www.arc.vt.edu/modules describes how to use ARC's software modules
- http://www.arc.vt.edu/scheduler describes how to interact with the scheduler - submit and check jobs, view output, etc.
- http://www.arc.vt.edu/faq provides answers to some frequently asked questions