HokieSpeed, a Cluster for Parallel Computing

CMDA 3634: Computer Science Foundations for Computational Modeling and Data Analytics
Presented by John Burkardt, Advanced Research Computing, 23 February 2017

HokieSpeed is a CPU/GPU cluster of 204 nodes, built with an NSF grant in 2012. Each node is outfitted with 24 GB of memory, two six-core Xeon E5645 CPU's and two NVIDIA M2050 / C2050 GPU's, each with 448 CUDA cores. HokieSpeed is managed and maintained by Advanced Research Computing at Virginia Tech.

There are two login nodes, *hokiespeed1* and *hokiespeed2*, which allow interactive users to login, create directories, transfer and edit files, and to submit jobs to the compute nodes, *hokiespeed3* and up. The compute nodes are normally only accessed indirectly, through a batch queue system.

HokieSpeed retires in June 2017, replaced by ARC's *newriver*, *cascades*, *dragonstooth* and *blueridge* clusters.

1 Exercise: Login to HokieSpeed, Customize Your Prompt

All students in CMDA3634 have been given accounts on the HokieSpeed cluster.

Open a terminal on your Ubuntu laptop; then log into either HokieSpeed login node *hokiespeed1* or *hokiespeed2* with your PID and password:

```
ssh -X PID@hokiespeed1.arc.vt.edu
```

The name of the **ssh** command is short for "Secure Shell". The **-X** switch is needed for any graphics interaction, including the use of **emacs**.

Your Ubuntu terminal window is now talking to HokieSpeed; commands you type are sent to Hokiespeed, executed there, and results printed on your screen. This continues until you break the connection with the **logout** command.

HokieSpeed runs a version of the Linux operating system, so after your experience with your Ubuntu laptop, the basic commands should be quite familiar. However, this means it is easy to look at your terminal and not realize which machine you are talking to, laptop or HokieSpeed. One way to fix this is to reset the prompt on HokieSpeed.

While logged into HokieSpeed, use **emacs** to edit (or create) the file **.bashrc**:

```
emacs .bashrc
```

and type in the following single line

```
PS1="hokiespeed: "
```

Save the file and quit **emacs**.

Your change to the prompt will take effect the very next time you log in. Because we are impatient, let's check right away. Typing the following command should update your prompt:

```
source .bashrc
```

2 Exercise: Copy Example Files, Interactive Compile and Run

A number of example files are available for you in my directory on HokieSpeed.

Copy them with the following command:

```
cp ~/burkardt/demo/* .
```

One file is *hello.c*. Compile and run this program as you would do so on your Ubuntu laptop.

3 Exercise: Transfer Files with SCP

Often, you will have a file on one computer but need it to be copied to another. The **scp** command, whose name is short for "Secure Copy", is one way to do this.

It will be easiest to work with **scp** if you always run it from your laptop. The format of the command is:

```
scp original_login:original_filename copy_login:copy_filename
```

Omit login information for your laptop; abbreviate *copy_filename* to **"."** when it's to have the same name as the original.

Open a second terminal window on your Ubuntu laptop; copy *hello.c* from your HokieSpeed directory:

```
scp PID@hokiespeed1.arc.vt.edu:hello.c .
```

Copy the file *quad.c* from my directory on HokieSpeed to your laptop:

```
scp PID@hokiespeed1.arc.vt.edu:/~burkardt/demo/quad.c .
```

Copying files from the laptop to HokieSpeed is similar. On the laptop, make a copy of *hello.c* called *goodbye.c*. Then transfer this file from your laptop back to HokieSpeed:

```
cp hello.c goodbye.c
scp goodbye.c PID@hokiespeed1.arc.vt.edu:
```

In your **ssh** window that is connected to HokieSpeed, verify that there is now a new file called *goodbye.c*

4 Exercise: Set the Environment with the Module Command

In order to serve a variety of users. HokieSpeed uses the **module** command to customize the software environment. When you log in, the default environment includes a compiler family (probably intel) and a version of MPI (probably openmpi). Using versions of the **module** command, you can list the current software environment, replace items, and load up other software such as the pgi compiler, cuda, python, or OpenFOAM.

When you login, the HokieSpeed environment begins by loading two modules, choosing a compiler and a version of MPI. To see the details of these default choices, type

```
module list
```

A useful Linux command is called **which**. You can put the word **which** in front of any command, and Linux will report the full name of the executable program that that command will invoke...or nothing, if the command doesn't mean anything to Linux.

In our default environment, what does the **mpicc** command mean?

```
which mpicc
```

Now HokieSpeed offers several choices for compilers and MPI libraries. If we want to change the current settings, we can begin by unloading all the current modules with the **module purge** command. Do this, and then ask again about the **mpicc** command:

```
module purge
which mpicc
```

Now we will use the **module load** command to get very specific values for the compiler and MPI, and the **which** command will verify that **mpicc** means something again.

```
module load gcc/4.7.2
module load mvapich2/1.9b
which mpicc
```

At any time, you can type **module avail** to list the modules that can legally be added to the current environment. You can type **module spider** *name* to look up details about a specific program. For example **module spider openmpi** will list all the versions of the **openmpi** library that are available, as well as all the compiler modules that it can work with.

5 Exercise: Interactive MPI Compile and Run

Let's compile and run the file *hello_mpi.c*. Because of the previous exercise, we know exactly which compiler and MPI library we have loaded into our environment. Therefore, all we have to do is issue the same commands we would use on Ubuntu:

```
mpicc -o hello_mpi hello_mpi.c
mpirun -np 4 hello_mpi
```

The file *quad_mpi.c* is another MPI program. It approximates the integral $I = \int_0^1 \frac{1}{1+x}\,dx = \ln(2) \approx 0.6931471805$. If we use m MPI processes, then each process divides a part of the interval [0,1] into n subintervals, and uses a Riemann sum approximation. The individual estimates are combined using a new MPI function: **MPI_Reduce()**. The value of m is determined by how many processes we create. The value of n is a command line parameter.

Compile the program. Compare the results using m=1 process and n=1000 intervals versus m=4 processes and n=250 intervals:

```
mpicc -o quad_mpi quad_mpi.c -lm
mpirun -np 1 quad_mpi 1000
mpirun -np 4 quad_mpi 250
```

6 Exercise: Interactive MPE Run

If we want to use the **jumpshot** program to visualize our MPI execution, there are a number of issues:

- we need to load some special modules;
- we need to use a special version of the MPI compiler;
- we need to specify an extra "logging" option;
- we need to copy the **.clog2* file back to our Ubuntu laptop, because **jumpshot** is not installed on HokieSpeed.

Let's see how a run of **quad_mpi** would work under MPE.

When in doubt, it's best to purge the old modules and start fresh. For MPE, we can type

```
module purge
module load gcc/4.7.2
module load mvapich2/1.9b
module load jdk
module load mpe2
```

or, if you like to save typing:

```
module purge; module load gcc/4.7.2 mvapich2/1.9b jdk mpe2
```

Instead of compiling and loading with **mpicc**, we need to use **mpecc**:

```
mpecc -o quad_mpi quad_mpi.c -lm (MORE STUFF HERE!)
```

We need to add the logging option:

```
mpecc -o quad_mpi quad_mpi.c -lm -mpilog
```

Now we run the program in the usual way:

```
mpirun -np 4 quad_mpi 250
```

After we run the program, we notice a *quad_mpi.clog2* file, which contains the timing informatation that **jumpshot** needs. Since **jumpshot** is on the laptop, we now need to use **scp** to copy the file. Remember that the **scp** command should be issued on your laptop command window, not on the **ssh** window where you see the *hokiespeed:* prompt!

```
scp PID@hokiespeed1.arc.vt.edu:quad_mpi.clog2 .
```

And you can now run **jumpshot** locally on your data from HokieSpeed.

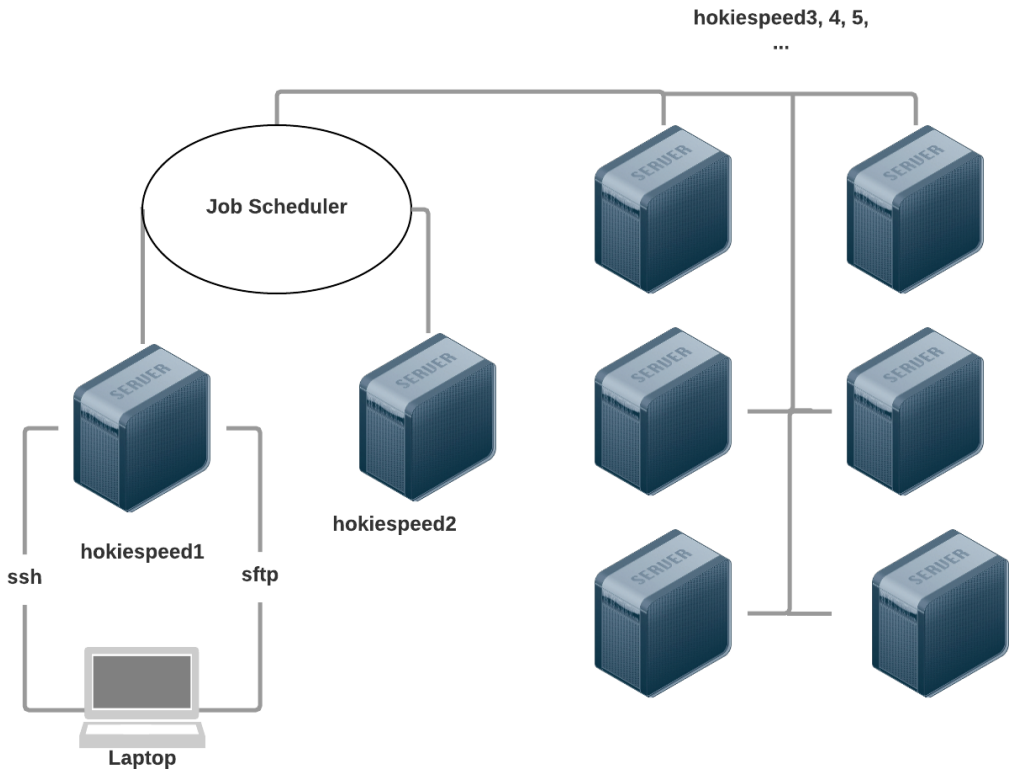

Figure 1: The HokieSpeed cluster has 2 login nodes and 202 compute nodes.

7 Exercise: Compile and Run Through a Batch Job

HokieSpeed has 204 nodes, but you can only log into 2, and we really shouldn't do any big computing on these login nodes, because they are shared by many users. Type **who** right now in your HokieSpeed terminal and you will see many other people logged in. The HokieSpeed *compute nodes* are where real computational work is done. Any subset of these nodes can be assigned to work together on a user task; but in order for this to happen smoothly, a *job scheduler* is used. Instead of you issuing interactive commands, you write them down in a file, and you send this to the scheduler. It adds your request to a list of job requests, and only runs your job when the appropriate number of nodes are available and nobody more important than you wants them!

Your request to run a job is done through a batch file, which we might call *quad.sh*. This file consists of two parts:

1. information to the scheduler, such as job size and time limits;
2. the commands you would have entered interactively.

Here is what the job file *quad.sh* looks like, for compiling and running *quad.c* through the batch system. You should be a little mystified by the first five or six lines, but you should recognize the rest:

```
#! /bin/bash

#PBS -l walltime=00:05:00     <-- At most 5 minutes of time
#PBS -l nodes=1;ppn=1         <-- One node please, and just one of 12 cores
#PBS -W group_list=hokiespeed
#PBS -q normal_q              <-- Specifies the queue to use
#PBS -j oe                    <-- return output and erros in a single file

cd $PBS_O_WORKDIR             <-- Run job where this batch file is

module purge
```

```
module load gcc

gcc -o quad quad.c -lm       <-- need -lm for math library (log, fabs)
./quad 1000                  <-- run the program
```

The one line that needs to be explained is **cd $PBS_O_WORKDIR**. This simply says to the batch system that, before trying to run the job, it should move to the directory that contains the batch file **quad.sh**.

Verify that, in your HokieSpeed directory, you have copies of *quad.c* and *quad.sh*. Now we are ready to run the job in batch. Because it only asks for a small amount of computer resources, we should expect it to run quickly.

1. submit your job request using the command **qsub quad.sh**.
2. note the number in the system response, such as **123456.master.cluster**, (I'm pretending it's 123456!).
3. check your directory using the **ls** command.
4. check your job with the command **checkjob -v 123456**.
5. keep issuing **ls** commands until you see a file called *quad.sh.o123456*.
6. use **emacs** or **more** to display *quad.sh.o123456*. Did an estimate for the integral appear?

8 Exercise: Using MPI in Batch

The whole point of having a cluster is that we get to use lots of computing power.

Our previous job just used 1 core on one node. Each HokieSpeed node has 12 cores, and there are multiple nodes. In the batch file, the statement

```
#PBS -l nodes=?;ppn=?
```

is used to specify how many nodes we want and how many processors per node (cores actually). On HokieSpeed, the **ppn** value can never be more than 12. To get more cores, we leave **ppn** at 12, but start increasing the **nodes** value. Since we're sharing HokieSpeed today, we'll stick to one-node jobs, and we'll just ask for 4 cores.

Here is the modified batch file *quad_mpi.sh*:

```
#! /bin/bash

#PBS -l walltime=00:05:00
#PBS -l nodes=1;ppn=4             <-- One node please, and just 4 of 12 cores
#PBS -W group_list=hokiespeed
#PBS -q normal_q
#PBS -j oe

cd $PBS_O_WORKDIR

module purge
module load gcc
module load mvapich2/1.9b         <-- Load a version of MPI.

mpicc -o quad_mpi quad_mpi.c -lm  <-- MPI-aware compiler
mpirun -np 4 ./quad_mpi 250       <-- run the program using 4 processes
                                  <-- mpiexec -n 4 ... will also work.
```

Repeat the previous exercise, but this time submitting the batch file **quad_mpi.sh**.

9 Exercise: Using MPE in Batch

Suppose that you want to have **jumpstart** analyze your MPI program's execution? Then we pretty much just have to copy all the interactive commands for MPE into the corresponding batch file, submit the job, and when it is finished, transfer the **.clog2* files to our laptop.

The batch file *quad_mpe.sh* has the necessary commands set up. It looks like this:

```
#! /bin/bash

#PBS -l walltime=00:05:00
#PBS -l nodes=1;ppn=4             <-- One node please, and just 4 of 12 cores
#PBS -W group_list=hokiespeed
#PBS -q normal_q
#PBS -j oe

cd $PBS_O_WORKDIR

module purge
module load gcc
module load mvapich2/1.9b
module load jdk
module load mpe2

mpecc -o quad_mpi quad_mpi.c -lm -mpilog
mpirun -np 4 ./quad_mpi 250
```

Submit this job, wait for it to complete, verify that the *quad_mpi.clog2* file was created, and use **scp** to copy it back to your Ubuntu laptop.

10 Summary

- **ssh** logs you into HokieSpeed;
- **scp** commands on Ubuntu transfer files back and forth;
- **module** commands set up your environment;
- **gcc**, **mpicc** or **mpecc** compiles and loads programs;
- **mpecc** needs the -mpilog switch to do jumpstart logging;
- **mpirun -np 4** runs MPI programs on 4 processes;
- you need a batch file *batch.sh* to run programs on the compute nodes;
- request nodes and cores on the **-l nodes=?:ppn=?** line;
- **qsub** *batch.sh* submits the batch file;
- **checkjob -v 123456** checks on job 123456;
- Output comes back in a file *batch.sh.o123456*;
- *.clog2 files must be transferred back to your machine.

11 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.
- http://www.arc.vt.edu/faq provides answers to some frequently asked questions.