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

1 HokieSpeed

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. However, for development and testing, there is an **interact** command which allows a user to request exclusive interactive access to a single compute node.

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

2 Login and File Transfers to HokieSpeed

All students in CMDA3634 have been given accounts on HokieSpeed. From a terminal on your computer, log into *hokiespeed1* or *hokiespeed2* with your PID and password:

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

Your terminal is now "talking" to Hokiespeed, and will continue to do so until you enter the **logout** command. An alternative form of the **ssh** command is

```
ssh -X PID@hokiespeed2.arc.vt.edu
```

which is necessary if you want to do any graphical operations while logged into the remote system. This includes using the graphical interface to the **emacs** editor.

To transfer files between your system and HokieSpeed, a command like

```
sftp PID@hokiespeed2.arc.vt.edu
```

will establish a connection between a directory on your local system and a directory on the remote system, which is initially the login directory. Commands to **sftp** will refer to the remote system, HokieSpeed, unless they are preceded by the letter **l** for "local". Some of these commands include:

ls *pattern*	list files on remote system
lls *pattern*	list files on local system
cd *directory*	change directory on remote system
lcd *directory*	change directory on local system
pwd	report current remote directory
lpwd	report current local directory
mkdir *directory*	make a directory on the remote system
rmdir *directory*	delete a directory on the remote system
rename *old new*	rename remote file from old name to new name
rm *file*	delete a remote file
lrm *file*	delete a local file
get *file*	copy remote file to local directory
put *file*	copy local file to remote directory
quit	terminate **sftp**.

3 Practice: File Transfer with SFTP

1. open a terminal on your laptop;
2. create a directory *alice*;
3. move to that directory;
4. create a file there, with the command **touch bob.txt**;
5. establish an **sftp** connection to HokieSpeed;
6. create a remote directory on HokieSpeed called *cher*;
7. change your remote directory to *cher*;
8. put a copy of *bob.txt* into *cher*;
9. rename the remote file *dirk.txt*;
10. get a remote directory listing to verify;
11. copy the remote file *dirk.txt* to your local directory;
12. get a local directory listing to verify you have *bob.txt* and *dirk.txt*;
13. remove the remote file *dirk.txt*;
14. change remote directory to .. (go up one level);
15. remove the remote directory *cher*;
16. quit **sftp**;
17. remove the files *bob.txt* and *dirk.txt*;
18. change directory to ..
19. remove the directory *alice*.

4 HokieSpeed Environment

HokieSpeed runs a version of Linux, so the operating system is very similar to your Ubuntu machine.

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.

Command	Example	Meaning
`module list`		List the modules currently loaded
`module avail`		List modules that are available to be loaded
`module purge`		Unload all modules
`module show` *name*	`module show cuda`	Print information about the cuda module
`module load` *name*	`module load R`	Load the R module
`module unload` *name*	`module unload ffmpeg`	Unload the ffmpeg module
`module spider` *name*	`module spider namd`	Search for a namd module
`module swap` *names*	`module swap intel gcc`	Replace intel by gcc & reload dependencies

5 Practice: Compile and Run

The integral $I = \int_0^1 \frac{1}{1+x}\,dx = \ln(2) \approx 0.6931471805$.

A left Riemann sum might use n intervals, defining a spacing $h = \frac{1}{n}$, and nodes $x_i = \frac{i}{n}$, resulting in an approximation $Q = h * \sum_{i=0}^{i<n} \frac{1}{1+x_i}$. The approximation error is $E = ||Q - I||$.

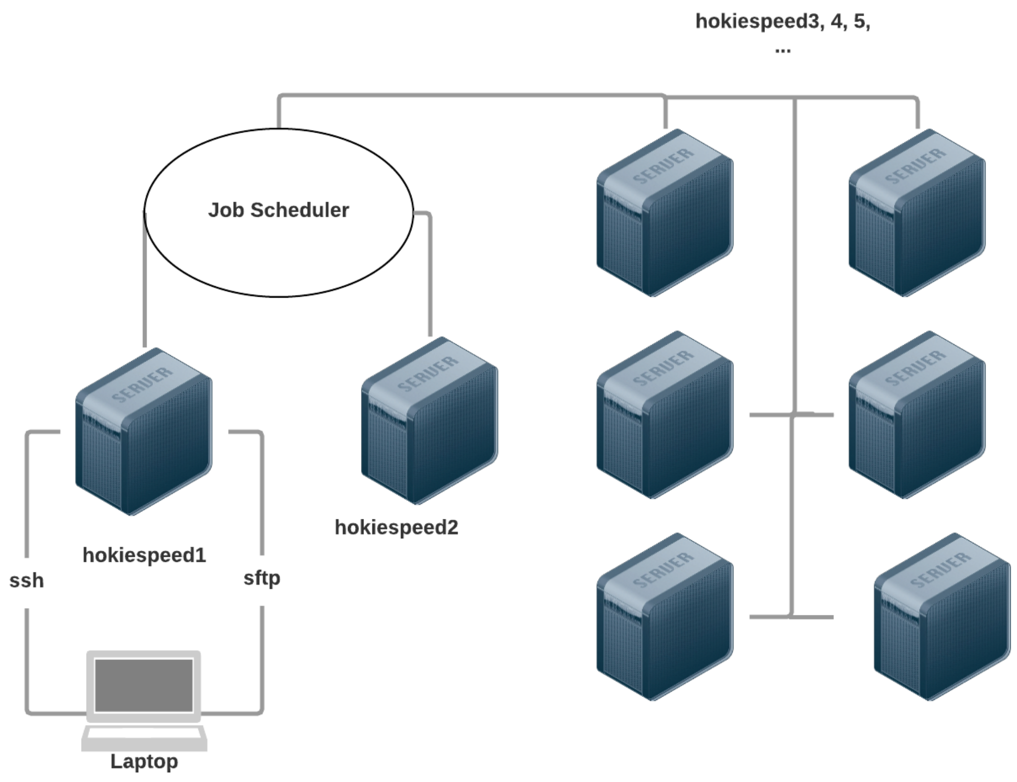

Figure 1: The HokieSpeed cluster has 2 login nodes and 202 compute nodes.

1. on your laptop, write a C program *quad.c* which is given n, and prints n, q, and e.
2. compile and run your program locally, using $n = 1000$.
3. make a note of the value of e, the approximation error.
4. use **sftp** to connect to HokieSpeed.
5. create a remote directory *quad*, move there, and put a copy of *quad.c* there.
6. open a second terminal window, and use **ssh** to connect interactively to HokieSpeed.
7. move to the remote *quad* subdirectory.
8. use a **module** command to determine what modules are loaded.
9. assuming **gcc** is not the current compiler, use **module** commands to make it so.
10. compile and run your program remotely, using $n = 1000$.
11. verify that your remote program got the same value for e, the approximation error.
12. logout of your **ssh** session.
13. quit your **sftp** session.

6 Running Batch Jobs

HokieSpeed has 204 nodes, but you can only log into 2; the rest are *compute nodes.* 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 your commands, you write them down in a file, and notify the scheduler. It adds them 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* might look like:

```
#! /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

module purge
module load gcc

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

gcc -o quad quad.c -lm        <-- need -lm for math library (log, fabs)
./quad 1000                   <-- could read n from commandline
```

7 Practice: Batch Job

1. on your laptop, create the batch script *quad.sh*;
2. use **sftp** to connect to HokieSpeed.
3. move to your remote directory *quad*, and put a copy of *quad.sh* there.
4. open a second terminal window, and use **ssh** to connect interactively to HokieSpeed.
5. move to the *quad* subdirectory.
6. submit your job request using the command **qsub quad.sh**.
7. note the number in the system response **123456.master.cluster**, (I'm pretending it's 123456!).
8. check your directory using the **ls** command.
9. check your job with the command **checkjob -v 123456**.
10. keep issuing **ls** commands until you see a file called *quad.sh.o123456*.
11. use **sftp** to get a copy of *quad.sh.o123456* back to your laptop.
12. verify that your remote batch program got the same value for e, the approximation error.
13. quit your **sftp** session.
14. logout of your **ssh** session.

8 Parallel Programming with MPI

MPI, the Message Passing Interface, is one way to write parallel programs. We can see that our integral estimation could be handled by cooperating programs or processes, each of which works on a separate subinterval. It's MPI's job to show how the problem should be divided up among the processes. The user program is modified so that each process has a copy of the same instructions, but also has an individual ID, which allows it to determine its portion of the problem.

In order to get the total of the individual estimates, MPI allows processes to send messages to each other. Because of the modifications to the user program, a special version of the compiler must be used. Because multiple programs must be initialized, synchronized, and allowed to communicate, a special command is needed to execute the MPI program.

Here is what the MPI version of our program might look like:

```
# include <math.h>
# include <mpi.h>        <-- Need this
# include <stdio.h>
# include <stdlib.h>

int main ( int argc, char *argv[] )
{
  double e;
  double h;
  int i;
  int my_id;              <-- Holds my process number.
  double my_q;            <-- Holds my piece of the answer.
  int n;
  int p_num;              <-- Holds the number of processes.
  double q;
  double x;

  MPI_Init ( &argc, &argv );
  MPI_Comm_rank ( MPI_COMM_WORLD, &my_id );
  MPI_Comm_size ( MPI_COMM_WORLD, &p_num );

  n = atoi ( argv[1] );   <-- read value of n from command line

  h = 1.0 / ( double ) n / ( double ) ( p_num );
  x = ( double ) ( my_id ) / ( double ) ( p_num );

  my_q = 0.0;
  for ( i = 0; i < n; i++ )
  {
    my_q = my_q + h * 1.0 / ( x + 1.0 );
    x = x + h;
  }
  printf ( "  P:%d  MY_Q=%14.6g\n", my_id, my_q );
%
%  MPI_Reduce sends my piece of the answer to process 0,
%  which sums the pieces to get Q.
%
  MPI_Reduce ( &my_q, &q, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD );

  if ( my_id == 0 )
  {
    e = fabs ( q - log ( 2.0 ) );
    printf ( "  N=%6d  Q=%14.6g  E=%10.2e\n", n, q, e );
  }

  MPI_Finalize ( );

  return 0;
}
```

For our job file *quad_mpi.sh* has some significant differences compared to the previous example:

- ask for more than one core; the node has 12, but we'll just ask for 4;
- choose and load an MPI module;
- use an MPI version of the C compiler;
- use the **mpirun** command to create, synchronize and execute the multiple MPI processes.

```
#! /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

module purge
module load gcc
module load openmpi             <-- Load a version of MPI.

cd $PBS_O_WORKDIR

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.
```

The job *quad_mpi.sh* is submitted with the usual **qsub** command.

9 Practice: MPI Batch Job

1. on your laptop, create the MPI program *quad_mpi.c* (note that we will use n=250 here!);
2. on your laptop, create the batch script *quad_mpi.sh*;
3. use **sftp** to connect to HokieSpeed.
4. create a remote directory *quad_mpi*, and put these two files there.
5. open a second terminal window, and use **ssh** to connect interactively to HokieSpeed.
6. move to the *quad_mpi* subdirectory.
7. submit your job request using the command **qsub quad_mpi.sh**.
8. when your output file is returned, make sure the run was successful;
9. use **sftp** to get a copy of your output file back to your laptop;
10. quit your **sftp** session.
11. logout of your **ssh** session;
12. verify that your remote batch MPI program got the same value for e, the approximation error.

10 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.