Panther Cluster QuickStart Guide

Florida International University

December 30, 13

The purpose of this document is to provide members of the FIU community with the basic tools necessary to use the Panther cluster at the Instructional and Research Computing Center (IRCC) for high performance computing. This guide is designed as a starting point for cluster use, and the IRCC expects to host a complete user guide for the cluster at some point in the future.

Obtaining a Cluster Account

To obtain an account on the Panther cluster at FIU, navigate to utshelp.fiu.edu and login using your FIU credentials. Click on the link "Submit or Track Requests for Service" followed by "Submit a New Request". In the Summary field, choose "Account Requests" followed by "HPC Cluster". Fill in answers to the four questions in the Notes field, and click "Save".

Login and File Transfer

First, all HPC servers are behind protection of FIU campus firewall. Therefore, if you are either off campus or use any wireless network including FIU_SECUREWIFI, please authenticate to FIU VPN (http://vpninfo.fiu.edu/anyconnect/) before login. But, desktop/laptop connected to FIU wired network has legitimate access to HPC servers without VPN. Access to the cluster occurs through the login nodes hpclogin01 and hpclogin02. Logging in to the cluster requires either a secure shell client (such as PuTTy in Windows) or the ssh command in a UNIX-like terminal. For example, to log in to hpclogin01 from a UNIX terminal, input the following command:

```
ssh username@hpclogin01.fiu.edu
```

Here, "username" is your FIU student or employee username. After entering this command, you will be prompted to enter your FIU password.

File transfer to the cluster is accomplished via a secure copy client (such as WinSCP in Windows) or via the scp command in a UNIX-like terminal. To copy a file from your local disk to your home directory on the cluster using a UNIX terminal, issue the following command:

```
scp path_to_file username@hpclogin01.fiu.edu:~
```

Note that here, it does not matter whether you copy your file to hpclogin01 or hpclogin02. This is because your home directory on the cluster exists in a logical

volume on a parallel file system. This directory is mounted on both login nodes and every compute node. Hence, any file you copy using the above command will be visible to every computer in the cluster.

The parallel file system also has a 40 TB scratch space consisting of high-speed drives. This is intended for use by jobs that require extensive input and output operations. To copy a file directly to this space from your local computer, issue the following command:

```
scp path_to_file username@hpclogin01:/scratch
```

Note that there are no folders created in scratch for individual users. Therefore, it is recommended that you create one to separate your files from those of other users.

Final Note: Access to the cluster from off-campus requires access to the FIU Virtual Private Network (VPN). For more information, and to obtain the required VPN client software for your compuer, visit vpn.fiu.edu.

Program Loading

After logging in to one of the login nodes, it is necessary to load the program(s) you wish to use through a utility called "module". All software not included in the basic operating system is installed through the module utility. For example, to load the MPI distribution OpenMPI, issue the following command:

```
module load openmpi
```

To view modules that you have loaded in your current session, use

```
module list
```

To view all of the available modules, use

```
module avail
```

Modules may be unloaded in the following way:

```
module unload openmpi
```

This may be necessary for switching from one version of a program to another, or switching the MPI distribution currently in use. Modules are automatically unloaded every time you log out of your session. For this reason, each user that wishes to use a particular piece of software must load it each time they log in.

To avoid the process of loading one or more modules every time you log in to the cluster, it is recommended to add commonly-used module commands to your ~/.bashrc file. If, for example, you wish to use OpenMPI in every session, you would append ~/.bashrc with the above command.

Note that in addition to large-scale programs, some basic utilities must also be loaded with the module command before they can be used. Examples include make and gcc.

Job Submission

Job submission on Panther is handled by the scheduling software IBM Platform-LSF (Load Sharing Facility). LSF provides a suite of utilities for the purpose of submitting jobs, viewing job status, modifying job submission parameters, deleting or suspending jobs, and viewing the status of various cluster resources.

LSF is primarily used for batch job submission. Batch jobs are jobs for which all of the necessary input is contained in one or more input files, and hence no user intervention is required during execution. LSF is also capable of interactive job submission, but that is not discussed in this guide.

Basic Submission

Job submission is accomplished with the bsub command. To run an application called "myprog" in your current working directory on a single compute core, issue the command

```
bsub ./myprog
```

To view the status of the job, use

```
bjobs [-a]
```

Here, the option [-a] must be used if the job has already completed. The output of bjobs consists of one or more jobs arranged in a list. Each entry consists of a unique number identifying a job (JOB_ID), the username of the person who submitted the job, the job status, execution host, job name and submission time. A more detailed discussion of bjobs is presented in a later section of this document.

The name of the job is determined by the name of the application binary. In the above example, the job name would be "myprog". It is possible to give jobs a custom name using the following bsub option:

```
bsub -J "name_of_job" ./myprog
```

Because "myprog" is run on a remote machine (a compute node), there is no output sent to the local terminal window, as there would be when running "myprog" locally. To view this output, use

```
bpeek JOB_ID
```

The integer JOB_ID can be obtained from the output of bjobs. To save this output to a file, submit the job as follows:

```
bsub -oo "output.txt" -eo "error.txt" ./myprog
```

The output is separated into regular output (STDOUT) and error messages (STDERR).

If "myprog" would normally require user-interactive input of quantities (numbers, strings, etc) that are known prior to submitting the job, enter these quantities (in order) into an input file (in this case input.txt) and submit the job using the following option:

```
bsub –i “input.txt” ./myprog
```

If “myprog” requires user-interactive input that is not known prior to submitting the job, then batch submission is inappropriate, and interactive submission is required.

When a job is submitted to LSF, it is placed in a queue prior to running. There are various different queues available to LSF users, and each of these has different properties. A job specification that includes a queue selection looks like this:

```
bsub –q night ./myprog
```

Here, the queue “night” is requested. This queue allows users to submit a job that uses up to 64 cores, however it will only run between the hours of 7pm and 7am. By comparison, the default queue (“normal”) allows jobs to run at any time, but with a maximum size of 32 cores.

For a complete list of queues, issue the command

```
bqueues
```

For a detailed description of the queue “night”, issue the command

```
bqueues –l night
```

Submission Scripts

The bsub command admits many options, and submission lines can become long and difficult to input/remember. A submission script solves this problem. This file contains all of the bsub options that you wish to use, and the name of your application. A submission script combining the examples contains the following:

```
#BSUB –q night
#BSUB –oo “output.txt”
#BSUB –eo “error.txt”
#BSUB –i “input.txt”
./myprog
```

If this file is called “subscript_ex”, then submission is performed through shell input redirection as follows:

```
bsub < subscript_ex
```

Note that this is different from some other schedulers, in which submission scripts are executable shell scripts submitted as a direct argument to the submission command.

Job Management

The commands bjobs is used to display information about jobs.

```
bjobs [-p][-d][-e][-a][-l][-u username][JOB_ID]
```

With no options, bjobs lists only running jobs belonging to the user that submits them. With options, the following behavior occurs: [-p] lists only pending jobs; [-d]

lists only finished jobs; [-e] lists only exited (failed) jobs; [-a] lists all jobs; [-l] provided verbose output; [-u] lists job information for a different user, or with "username" set to "all", for all users; [JOB_ID] lists information for a particular job.

Sometimes it is necessary to suspend or kill pending or running jobs. To suspend a job, use

```
bstop JOB_ID
```

To resume a job, use

```
bresume JOB_ID
```

Finally, to kill a job, use

```
bkill JOB_ID
```

The command bmod can be used to modify options for a previously submitted job that is pending, suspended or running. If the job is pending or suspended before running, almost any option can be changed. The syntax is

```
bmod [-option new_value] JOB_ID
```

For example, suppose a job with JOB_ID = 1234 has been submitted to the normal queue and is currently pending. To change this job to the night queue, submit the following command:

```
bmod –q night 1234
```

Note that this command will not work once the job is running. If the job is running or suspended while running, only resource requests [-R] can be changed. Resource requests are discussed later in this document.

Job Arrays

Sometimes it is necessary to run a program a large number of times, each time with a different set of input files/parameters. The job array mechanism provided by LSF enables the user to accomplish this with a single job submission.

As an example, consider a program "addone" that takes an integer as a command line input argument, adds 1 to it and writes the result to STDOUT, i.e. running "./addone 1" in a local terminal would result in the number 2 being displayed.

A submission script with the following content will submit an array job to the cluster to perform this calculation on input integers 1-500:

```
#BSUB –J "name_of_job[1-500]"
#BSUB –oo "stdout%I.txt"
#BSUB –eo "stderr%I.txt"
./addone $LSB_JOBINDEX
```

Here, an extension of the job name option (-J) results in the creation of 500 jobs indexed from 1 to 500. The %I is a job index variable that works with the file I/O

options of bsub. Specifying the command line argument requires a different job index variable, namely the local environmental variable $LSB_JOBINDEX.

The result of running this submission script will be the creation of 500 output files containing integers 2-501.

Parallel Jobs

Parallel jobs are jobs in which several cores on the cluster are used to accomplish a single task. Shared memory parallelization is accomplished on Panther with OpenMP. OpenMP libraries are part of compiler distributions, and so the instructions for using OpenMP are compiler dependent. To submit an OpenMP job using the GNU Compiler Collection (gcc), the user must first issue the following module command:

```
module load gcc
```

The following submission script submits the application "ompprog" as an OpenMP job using 8 cores:

```
#BSUB -a openmp
#BSUB -n 8
./ompprog
```

Distributed memory parallelization is accomplished on Panther using either MPI or PVM. Only MPI will be discussed here, since it is significantly more popular than PVM. The only versions of MPI currently supported by Panther are OpenMPI and Platform-MPI. Intel MPI and MPICH may be added in the future.

To submit the application "mpiprog" as a 32-core OpenMPI job, first load the module

```
module load openmpi
```

Then, use a submission script with the following features:

```
#BSUB -a openmpi
#BSUB -n 32
mpirun.lsf ./mpiprog
```

To submit the application "mpiprog" as a 32-core Platform-MPI job, first load the module

```
module load platform-mpi
```

Then, use a submission script with the following features:

```
#BSUB -n 32
mpirun -lsf ./mpiprog
```

Note the slight difference between this mpirun command and the previous one. This is not a typographical error.

Only one MPI module should be loaded at any given time. When switching from one version of MPI to another, it may be necessary to recompile your code after loading the new module.

Resource Requests

Sometimes it is necessary to ensure that the compute nodes on which your job runs have the appropriate resources. As an example, a 4-core job that requires 16 GB of system memory per core may be submitted to a 12-core compute node with 64 GB of system memory available. When this job runs however, it will be very important to ensure that no other cluster users can use the 8 unused cores on this node. This is accomplished by using a bsub option to request that all of the system memory is reserved on this node.

In this above example, the relevant resource was system memory. There are many other examples, including hostname, number of cores, swap space available and average core utilization. Parallel licenses for proprietary software can also be administered using the resource request mechanism. For a full list of the names of all resources available on the cluster, run the command

```
lsinfo -r
```

To get an overview of some of the more important resources on a per-node basis, run the commands

```
lshosts
```

or

```
lsload
```

Making a resource request when submitting a job under LSF involves including one or more lines of the following format in your submission script:

```
#BSUB -R "command[string]"
```

There are five different commands available, and many more strings. The number of possible combinations is too large to discuss in full, but the following examples illustrate some common usages.

Choose compute node with >1GB available system memory:

```
#BSUB -R "select[mem>1024]"
```

Choose compute node with >12 cores and reserve 1GB of system memory:

```
#BSUB -R "rusage[mem=1024]"
```

Note here the important difference between requesting a resource and reserving a resource: In the first case, LSF checks whether the resource is available at the time of submission, but doesn't prevent other jobs from using that resource after submission. In the second case, LSF reserves the required amount of memory and prevents other jobs from using it until the reserving job is complete.

Choose a 16-core compute node with the least CPU occupation:

```
#BSUB -R "select[ncpus==16] order[ut]"
```

Choose a 12-core compute node with at least 200MB of swap space available:

```
#BSUB -R "select[ncpus==12 && swp>=200]"
```

Reserve four instances of a software license called "soft_lic_1":

```
#BSUB -R "rusage[soft_lic_1=4]"
```

Force parallel job to use no more than two cores per compute node:

```
#BSUB -R "span[ptile=2]"
```

Choose compute node with hostname n024:

```
#BSUB -R "select[hname==n024]
```

Run parallel job requiring 2GB per core with no more than 2 cores per node:

```
#BSUB -R "rusage[mem=4096] span[ptile=2]"
```

Note that, as illustrated in this last example, resource requests are performed on a per-host basis.