Introduction to the HPC Resources at the University of Pittsburgh

Kim F. Wong and Albert DeFusco

Center for Simulation and Modeling

July 18, 2013
This hands-on introductory workshop will cover the following topics:

- Characteristics of the high performance computing cluster at Pitt
- Various methods for accessing the cluster
- Useful common Linux commands
- How to modify your default working environment
- Accessing installed software and compiling code
- How to write a job submission script
- Querying and modifying the status of submitted jobs
- Where to go if you have questions, problems, or complaints
- Main documentation portal --- http://core.sam.pitt.edu/doc
Here are some characteristics of a HPC cluster (or computational grid)

- Comprised of a network (a grid) of computers that can be harnessed simultaneously to tackle a problem requiring large processing power
- Elements of the grid can be connected by a specialized high-speed interconnect or a standard GigE network
- Grid resources are shared among a community of users
- Users request grid resources through some kind of management software

Motivations for grid computing

- Necessary for solving grand challenge problems … problems that require an enormous amount of computation
- Provides more cost-effective usage of computational resources towards common objectives
Frank: Heterogeneous HPC Cluster

A reason for exploiting **parallel computing** is that you want to *divide* up the work load among processes so as to *conquer* the problem faster or to *conquer* a bigger problem.

- Maintained by CSSD & SaM; housed at RIDC Park
- ~7400 CPU cores comprised of dual 8-core Intel Sandy Bridge, quad 16-core AMD Interlagos, dual 6-core Intel Westmere, quad 12-core AMD Magny-Cours, and older Intel Xeon architectures
- Available memory on nodes range from 8-256 GB of RAM
- 4 of the Westmere nodes has 4 NVIDIA Tesla M2070 GPU cards
- Most nodes connected via Infiniband
- ~100s TB of storage; Panasas storage
- Frank is firewalled within Pittnet: Access from outside of Pittnet is enabled via a VPN session

Image courtesy of Luda Doroshkevich (CSSD)
frank.sam.pitt.edu is firewalled, so you will need to establish a Virtual Private Networking (VPN) before connecting to the cluster.

How do you access them remotely?

Point Firefox to [https://sremote.pitt.edu](https://sremote.pitt.edu) and authenticate using your Pitt credentials.

Alternatives to sremote: [http://core.sam.pitt.edu/offcampusaccess](http://core.sam.pitt.edu/offcampusaccess)
Access to Frank is enabled via a secure shell (SSH) connection to the cluster.

A SSH client called PuTTY is available for Windows

Specify these connection properties:

- Hostname: frank.sam.pitt.edu
- Port: 22
- Connection type: SSH

Clicking the Open button will open a SSH terminal

- login as: <Pitt Username>
- password: <my.pitt password>

Linux & Mac Users: type `ssh <username>@frank.sam.pitt.edu` within a terminal
Frank.sam.pitt.edu is firewalled, so you will need to establish a Virtual Private Networking (VPN) before connecting to the cluster.


1. Drag and drop between panels.
For Macs, I have heard that Cyberduck works well: [http://cyberduck.ch/](http://cyberduck.ch/). So I installed the Windows version to give it a try.

After authentication, a new window shows up. Drag and drop between that window and your local desktop/folders.
Transferring using Globus Online?

1. Full instructions for configuring Globus Online to connect to Frank are provided here: [http://core.sam.pitt.edu/globusonline](http://core.sam.pitt.edu/globusonline).

2. Authenticate using your Pitt credentials.
Transferring using Globus Online?

Unfortunately, drag-n-drop does not work. Use the arrow in between the two panels to transfer files from one endpoint to the other.

If you are transferring large genomics data, it is best to use Globus Online.
Your User Environment

For most users, the primary interaction with the cluster is via this \textit{terminal session}.

The line containing the active cursor is call the \textit{Linux commandline}

Through the Linux commandline, you can

- Create, remove, rename files and directories
- Edit files
- Compile and execute code
- Change your environment settings (details provided later)
- Submit jobs to the queue system (details provided later)
- Connect to other remote machines
### Common Linux commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pwd</code></td>
<td>show current location</td>
</tr>
<tr>
<td><code>ls</code></td>
<td>list content of current directory</td>
</tr>
<tr>
<td><code>cd</code></td>
<td>change to home directory</td>
</tr>
<tr>
<td><code>cd &lt;directory&gt;</code></td>
<td>change to <code>&lt;directory&gt;</code></td>
</tr>
<tr>
<td><code>mkdir &lt;directory&gt;</code></td>
<td>create <code>&lt;directory&gt;</code></td>
</tr>
<tr>
<td><code>rmdir &lt;directory&gt;</code></td>
<td>remove empty <code>&lt;directory&gt;</code></td>
</tr>
<tr>
<td><code>cp &lt;file&gt; &lt;new file&gt;</code></td>
<td>copy <code>&lt;file&gt;</code> to <code>&lt;new file&gt;</code></td>
</tr>
<tr>
<td><code>mv &lt;file&gt; &lt;new file&gt;</code></td>
<td>rename <code>&lt;file&gt;</code> to <code>&lt;new file&gt;</code></td>
</tr>
<tr>
<td><code>rm &lt;file&gt;</code></td>
<td>remove <code>&lt;file&gt;</code></td>
</tr>
<tr>
<td><code>man &lt;command&gt;</code></td>
<td>display online manual for <code>&lt;command&gt;</code></td>
</tr>
<tr>
<td><code>more &lt;file&gt;</code></td>
<td>display the content of a file</td>
</tr>
<tr>
<td><code>wc &lt;file&gt;</code></td>
<td>count lines, words, and bytes in a file</td>
</tr>
<tr>
<td><code>grep &lt;pattern&gt; &lt;file&gt;</code></td>
<td>pattern searching</td>
</tr>
</tbody>
</table>

---

Most commands have additional *option flags* that modify the command behavior. Use `man <command>` to learn more about these features.
By now you know of the commands `showq`, `grep`, and `wc`.

And if you have forgotten about the details of a command, you can always `man <command>` to print out the online man pages.

- `man grep`

But perhaps the most important command is the pipe:
The `|` symbol is the `[shift]\key that is typically located above the `[Enter]\key.

It permits you to pipeline several simple commands together into a more complex operation, whereby the output from each command are used as input for each successively command from left to right.

- `showq | grep kimwong | wc`

Effectively, what I have done is to show the number of jobs that are in the queue for user kimwong. `showq` displays all the jobs in the queue and `grep` prunes that list for lines matching kimwong, and finally `wc` counts the number of lines.
Environment Modules

When you log into the cluster, a set of initialization scripts execute and set up some default environment variables.

Your environment contains all information about your programs and preferences.

We use the Environment Modules package to facilitate dynamic modification of a user's environment.

- module avail
- module list
- module load <package>
- module unload <package>
- module purge
- module show

--- displays available packages
--- list loaded packages
--- load a package
--- unload a package
--- unload all packages
--- display content of module
Available Compilers

The Module Environments have default compilers loaded

[kimwong@login0 ~]$ module list
Currently Loaded Modulefiles:

1) modules                        4) openmpi/1.5.1-intel11          7) gold/2.2.0.4(default)
2) sys                            5) mkl/11.1.072/icc-st           8) torque/3.0.5(default)
3) intel/11.1.072(default:11.1)   6) env/intel-openmpi(default)      9) moab/6.1.7(default)

To change to the gcc (GNU Compiler Collection) environment, execute the following commands in your terminal

- module purge
- module load gcc/4.5
- module list

[kimwong@login0 ~]$ module list
Currently Loaded Modulefiles:

1) modules                        2) sys                        3) gcc/4.5.1(4.5)
Compiling Code

We will walk through simple examples of compiling code

First, copy the sample codes

- cp -rp /home/sam/training/cluster/hello_world/ .

Next, go into the hello_world/ directory and look around

- cd hello_world
- ls

[kimwong@login0 hello_world]$ ls
Hello.pbs_script hello_world.c hello_world.f90 hello_world_id.c hello_world_id.f90

Your environment has been configured to use the following commands for compilation

- CC -- C compiler
- CXX -- C++ compiler
- FC -- Fortran compiler
Let us compile `hello_world.c` and `hello_world.f90`:

- `CC hello_world.c -o hi`
- `FC hello_world.f90 -o howdy`

Now, let’s run the programs:

- `./hi`
- `./howdy`

In the C code example above,

- `CC` invokes the gcc compiler
- followed by the name of the source code
- the `-o <name>` option specifies a name for the compiled program
Let us compile with OpenMP support

- `CC -fopenmp hello_world.c -o hi_omp`
- `FC -fopenmp hello_world.f90 -o howdy_omp`

Now, let’s run the OpenMP programs

- `./hi_omp`
- `./howdy_omp`

Now is a good time to tell you about the environment variable `OMP_NUM_THREADS`

`OMP_NUM_THREADS` specifies the number of instances of an execution and you set it using

- `export OMP_NUM_THREADS=<integer>`

Give it try and run your program again. What happens for different `<integer>` values?
The `hello_world/` directory contains another more *complex* OpenMP codes

- CC `-fopenmp hello_world_id.c -o hi_omp_id`
- FC `-fopenmp hello_world_id.f90 -o howdy_omp_id`

What happens when you execute `hi_omp_id`? Try changing the value of `OMP_NUM_THREADS`. Now what happens?

As you might guess by now, the `OMP_NUM_THREADS` variable controls how many parallel execution instances of a particular code region.

If you are curious, view the codes using

- `more hello_world.c`
Because Frank is a shared resource, everyone must get in line to use it.
Backfill (frank-avail)
All interactive and batch access to Frank compute nodes are controlled by the Torque Resource Manager, an open source version of the Portable Batch System (PBS)
- Communicates with nodes and runs jobs
- Accepts resource requests

Moab Workload Manager is a policy-based job scheduler and is utilized on Frank to implement the scheduling
- Determines when jobs are run
- Manages prioritization and resource limits
- Allows small jobs to “backfill” while others wait

Account management is provided by Gold
- Logs and charges service units
Two Classes of Jobs

- **Shared Memory**
  - Applications that can run one or more cores/threads.
  - No communication between nodes

- **Distributed Memory**
  - Applications that efficiently use multiple cores across multiple nodes
  - *Not for single core/thread jobs*
The Queues

### Shared Memory

- **shared**
  - 15 nodes; 48 cores; 2.6 GB/core
  - 1 TB local disk
  - `-q shared -l nodes=1:ppn<=48`

- **shared_large**
  - 8 nodes; 16 cores; 4 GB/core
  - 2 TB local disk
  - `-q shared_large -l nodes=1:ppn<=16`

- **shared_heavy**
  - 2 nodes; 16 cores; 8 GB/core
  - 3 TB local disk
  - `-q shared_heavy -l nodes=1:ppn<=16`
  - *Access restricted by need. Submit a ticket if needed.*

### Distributed Memory

- **distributed**
  - 32 nodes; 12 cores; 4 GB/core
  - 1 TB local disk
  - `-q distributed -l nodes>=2:ppn=12`

- **dist_small**
  - 36 nodes; 16 cores; 2 GB/core
  - 1 TB local disk
  - `-q dist_small -l nodes>=2:ppn=16`

- **dist_big**
  - 36 nodes; 16 cores; 4 GB/core
  - 1 TB local disk
  - `-q dist_big -l nodes>=2:ppn=16`
All CPU-hours are not the same

- Cost is scaled by the relative CPU speeds
- All groups receive a yearly 10,000 SU allocation
  - Several investment options exist
  - Charges are made when a job finishes
- SU = wall-time * cpu-scale * n-cpus

<table>
<thead>
<tr>
<th></th>
<th>Magny Cours</th>
<th>Interlagos</th>
<th>Westmere</th>
<th>Sandy Bridge</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.5</td>
<td>0.75</td>
<td>1.0</td>
<td>1.5</td>
<td>8.0</td>
</tr>
</tbody>
</table>

http://core.sam.pitt.edu/service-unit-policy
Common job submission commands

- **frank-avail <queue>**
  - Determine available resources
  - [http://core.sam.pitt.edu/frank/batch](http://core.sam.pitt.edu/frank/batch)

- **qsub <options> <job-script>**
  - Request resources: queue, memory, walltime
  - Submit your job to be scheduled and run

- **checkjob -v <jobID>**
  -- monitor status of job

- **Showq -w user=<pittID>**
  -- show jobs in queue

- **qstat -n -u <username>**
  -- show jobs for <username>

- **qdel <jobID>**
  -- delete job with <job id>

- **mybalance -h**
  -- query your SU balance
This job submission script is called `hello.pbs_script`. To submit it to the queue, type

- `qsub hello.pbs_script`

For an interactive session, type

- `qsub -I -q shared -l nodes=1:ppn=1,walltime=4:00:00`

For more info on the batch system: [http://core.sam.pitt.edu/node/1891](http://core.sam.pitt.edu/node/1891)
Availability of Nodes

To see current and future availability of machines, run `frank-avail <queue-name>`.

```
defusco@login0:~>frank-avail dist_small

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Nodes</th>
<th>Duration</th>
<th>StartOffset</th>
<th>StartDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>343</td>
<td>24</td>
<td>00:00:30</td>
<td>00:00:00</td>
<td>16:02:28_10/09</td>
</tr>
<tr>
<td>407</td>
<td>28</td>
<td>00:00:04</td>
<td>3:52:31</td>
<td>19:54:59_10/09</td>
</tr>
<tr>
<td>439</td>
<td>30</td>
<td>00:00:04</td>
<td>3:52:31</td>
<td>19:54:59_10/09</td>
</tr>
<tr>
<td>503</td>
<td>34</td>
<td>00:05:31</td>
<td>5:22:56:46</td>
<td>14:59:14_10/15</td>
</tr>
<tr>
<td>535</td>
<td>36</td>
<td>INFINITY</td>
<td>5:23:02:17</td>
<td>15:04:45_10/15</td>
</tr>
</tbody>
</table>
```
Once a job is submitted, `checkjob` will return the state of the job.

`checkjob -v <jobID> or showq -w user=<pittID>`

- **Running**: Job is currently in execution on one or more compute nodes.
- **Deferred**: Job that has been held by MOAB due to an inability to schedule the job under current conditions. Deferred jobs are held for 1 hour before being placed in the idle queue. This process is repeated 24 times before the job is placed in batch hold.
- **Hold**: Job is idle and is not eligible to run due to a user, (system) administrator, or batch system hold
- **Idle**: Job is currently queued and actively waiting for available resources
- **Blocked**: Job is currently queued and ineligible to run due to throttling restricts, such as maximum number of cores already in use, or other policy violations. In some cases, when a policy violation is detected the job will be automatically canceled.
Resource Limitations

- All jobs are limited to 6 day wall times.

- Core limitations exist for every queue
  - Jobs become blocked when you reach your core limit. Blocked jobs will start when your running jobs finish. Run `checkjob -v <jobID>`.
  
  - [http://core.sam.pitt.edu/frank/batch](http://core.sam.pitt.edu/frank/batch)
The complete status of running and queued jobs can only be determined by `checkjob`

- Why is the job queued?
  - When will it start?

- Why is the job blocked?
  - Was there a submission error?

- Why did the job fail?
  - Exhausted allocation
  - System trouble
  - `checkjob` can be run up to 24 hours after completion (cancellation) of the job
defusco@login0:espresso-5.0>checkjob -v 370072
job 370072 (RM job '370072.headnode.localdomain')

AName: TME_50_s_004
State: Idle
Creds: user:pozun group:kjordan account:kjordan class:jordan qos:investor
WallTime: 00:00:00 of 6:00:00:00
BecameEligible: Wed Oct 10 09:40:17
SubmitTime: Wed Oct 10 09:40:13

Total Requested Tasks: 96

Req[0]  TaskCount: 96  Partition: ALL
TasksPerNode: 48
Reserved Nodes: (4:18:55:17 -> 10:18:55:17  Duration: 6:00:00:00)
n[43,45]*48
Checkjob Example: Blocked

defusco@login0:espresso-5.0> checkjob -v 370700
job 370700 (RM job '370700.headnode.localdomain')

AName: 2ion-cl-w33-r6
State: Idle
Creds: user:hac73 group:kjohnson account:kjohnson class:kohn qos:investor
WallTime: 00:00:00 of 2:22:00:00
SubmitTime: Wed Oct 10 13:04:40
(Time Queued Total: 4:02:18 Eligible: 00:00:02)

Total Requested Tasks: 2

Req[0] TaskCount: 2 Partition: ALL

NOTE: job violates constraints for partition Fermi (job 370700 violates active HARD MAXPROC limit of 128 for class kohn user partition ALL (Req: 2 InUse: 128))

BLOCK MSG: job 370700 violates active SOFT MAXPROC limit of 128 for class kohn user partition ALL (Req: 2 InUse: 128) (recorded at last scheduling iteration)
Checkjob Example: Error

defusco@login0:espresso-5.0> checkjob -v 371231
job 371231 (RM job '371231.headnode.localdomain')

AName: casino
State: Idle
Creds: user:defusco group:sam account:sam class:idist qos:investor
WallTime: 00:00:00 of 00:05:00
SubmitTime: Wed Oct 10 17:04:50
(Time Queued Total: 00:00:11 Eligible: 00:00:01)

Total Requested Tasks: 12

Req[0] TaskCount: 12 Partition: ALL
TasksPerNode: 12

NOTE: job cannot run (job has hold in place)

Message[0] job violates class configuration 'procs too low for class 'idist' (12 < 24)'
When a job dies or never runs

- **Always run** `checkjob -v <jobID>`
  - Not enough SU balance
  - Hardware issues
  - Incorrect resource request
    - `ppn` too large or too small
    - `mem` too large or too small
  - Access trouble

- **Submit a ticket**
  - [http://core.sam.pitt.edu/node/add/support-ticket](http://core.sam.pitt.edu/node/add/support-ticket)
Now you know a bit more about the HPC cluster available to the University of Pittsburgh. In particular, you

- Know how to access Frank using a SSH client
- Know how to transfer files to and from Frank
- Know some basic Linux commands for manipulating files and directories and for navigating the file system
- Know the importance of the pipe: |
- Understand how packages and environment variables are set using Environment Modules
- Know how to compile code
- Know how to write a basic PBS submission script
- Know how to submit jobs, query the state of jobs, and remove jobs from the queue

For general documentation about the SaM cluster:

- [http://core.sam.pitt.edu/frank](http://core.sam.pitt.edu/frank)
Problems and Issues Using Frank

1. Login to Core and post a support ticket.

2. Fill in the support form and describe the issue in the **Body**: portion. Clicking on the SAVE button at the bottom will post the ticket to the SaM help desk.