Research Computing Club Presents

Hyak Training Session

October 27, 2020
Jesse Prelesnik (Training Coordinator)
Outline

Where I assume you’re at right now:

“I finished the steps to getting access* to Hyak, but don’t really know what to do now. How do I get up and running with the computing I want to do?”

I. Hyak overview
   A. Hyak architecture
   B. Logging on to Hyak

II. Navigating Hyak
   A. Basic commands
   B. Important locations
   C. Transferring files

III. Slurm
   A. Running a job
   B. Commands
   C. Modules
   D. Interactivity

IV. Other resources
   A. Topics outside of this tutorial
   B. Places to get help

* https://depts.washington.edu/uwrcc/getting-started-2/getting-started/
Hyak overview

Hyak is a “condominium” supercomputing cluster:

- Research groups own nodes in partitions.
  - There’s one partition for all tuition-paying students:
    - Student Technology Fee – “stf”
  - In addition to our partitions, groups have access to the build and ckpt partitions

- ~10,000 cores in total
  - A typical laptop might have 4-8 cores (CPUs)

One cluster for right now: “mox” (current), soon “clone” (future)
An old cluster, “ikt”, was just retired and is no longer usable
Accessing Nodes - Hyak “Architecture”

- All nodes share the same filesystem (except /tmp)

**Hyak / Mox**

- on your screen, but a separate computer

**Login Nodes**

- (Shared space)
- for navigation, not calculation
- Transfer files to/from Mox
- Submit work to other nodes

**Build Nodes**

- For compiling new software
- Can access the internet (!)
- Not allowed to use whole node

**Computing Nodes**

- High performance, lots of nodes
- Takes batch submission
- Interactive optional
- Actual work is done here

Your Computer

![Diagram showing network connections and node types]

- login via ssh command (next slide)
Logging on to Hyak

Terminal – “command line”
Easily accessible on Mac/Linux

“Secure Shell” ssh command
Enables connection to remote machine

```
(base) Jesses-MacBook-Pro:~ home$ ssh -X prelj@mox.hyak.uw.edu
Password: [redacted]
```

ssh <uwnetid>@mox.hyak.uw.edu

Two-factor authentication required
(Through UW-IT)

You’re in!

This login node is meant for interacting with the job scheduler and transferring data to and from Hyak. Please work by requesting an interactive session on (or submitting batch jobs to) compute nodes.

Visit the Hyak user wiki for more details:
http://wiki.hyak.uw.edu/Hyak-mox+Overview

Questions? E-mail help@uw.edu with "hyak" in the subject.

Run "scontrol show res" to see any reservations in place that will prevent your jobs from running with a "(ReqNodeNotAvail,*" error.

```
(base) [prelj@mox1 ~]$ [redacted]
```

- Notice you are defaulted to the login node
- Jobs are not run here

“Graphical user interface” (GUI)

No GUI
Navigate via command line
What if I’m on Windows?

You’ll need a terminal somehow, and Windows does not provide a good one by default.

My preferred route: “Virtual Machine” (VM)
Sets up a mini environment that emulates a different operating system
Lets your Windows machine have some non-Windows capabilities (like terminal)

“Oracle” is one example of a VM

Other options for Windows:

- PuTTY (Terminal emulator)
- CygWin (Runtime environment)
- GitBash
- Windows Subsytem for Linux
- WinSCP (just for transferring files)
- cmder
- xshell
- … and probably more!

How you get a terminal does not matter, they are all effectively the same.

More help:
http://wiki.cac.washington.edu/display/hyakusers/Logging+In
I’m in

... Now what?
Exploring Hyak

Hyak file systems through command line

Where in the computer am I?
pwd - “Print working (current) directory”

What folders and files are here?
ls - “List” files in current directory (folder)

Let’s go to where everybody stores files

cd - “Change directory”

Essential First Actions:

- mkdir
  - “Make directory”
  - Create a place to put your files

- vim, emacs, nano (etc.)
  - Command line text editors

Each research group has a folder in gscratch

Note: On a computer, a “folder” and a “directory” are the same thing
Important locations on Hyak (get there with cd)

🌟 /gscratch/stf
  ○ Main work location for stf users
  ○ Any files untouched for >30 days will be auto-deleted!!

🌟 /usr/lusers/<username>
  ○ Home directory “~”, you are put here by default
  ○ Only 10 GB of storage per user

• /tmp
  ○ Node local storage (separate from shared filesystem)

• /sw
  ○ All software installs

• /sw/contrib
  ○ User installed software

• /sw/modules-1.775/modulefiles/contrib
  ○ User added modulefiles

Where pre-installed programs (and their supporting files) are housed
You will seldom need to go here
Basic shell (bash) commands: Try them out!

File system manipulation:

- **ls**
  - “List” files in current directory (folder)
- **cd**
  - “Change directory”
- **pwd**
  - “Print working (current) directory”
- **mkdir**
  - “Make directory”
- **mv**
  - Move (rename) file or directory
- **cp**
  - “Copy” files and/or directories (-r)
- **rm**
  - “Remove” files and/or directories (-r)

File editing and compression:

- **nano**
  - Edit files
  - Other editors: vim, emacs, etc.
- **tar**
  - Compress for a “tape archive”
- **zip** (and **unzip**)
  - Compress via zip algorithm
  - Windows friendly
- **man**
- **find**
- **top**
- **kill**
- **chmod**
- **curl**
- **grep**
- **sed**

Many, many more
Transferring files

- **Send file To Mox:**
  
  ```
  scp <path/to/file> <username>@mox.hyak.uw.edu:<path/to/dest>
  ```

  On your computer  
  Your login from ssh  
  Place on Mox you want to put the file

- **Get file From Mox:**
  
  ```
  scp <username>@mox.hyak.uw.edu:<path/to/file> <path/to/dest>
  ```

  Flag `–r` needed to copy whole folder at once

  An alternative copier for synchronizing directories is `scopy`

---

**Lolo:**

- Magnetic tape archive (lolo archive)
- For long term storage - **only store compressed large files**!
- STF location: `/archive/hyak/stf`
- Transfer files the same as between local and Hyak
Running Your First Job

Real computing jobs are passed to Slurm

- Ensures fair share between users
- Run interactive or batch jobs
- Allows for running on the ckpt queue
  - (more on this later)

Some information is needed during job setup:

- **Partition**: whose CPUs/GPUs are we using? (we’ll use stf)
- **Allocation**: which bank account pays for these? (also stf)
- **Locations**: where to write/read files to/from
- **Resources**: how many CPUs/GPUs are needed, and for how long?
- **Modules**: which programs should be loaded to use?
Anatomy of a slurm script (written in “bash”)

In a text file called “submit.slurm”

```bash
#!/bin/bash
## The first line has to say this, as a bash script

## Job Name
#SBATCH --job-name=test_python

## Partition and Allocation
#SBATCH -p stf
#SBATCH -A stf

## Resources
#SBATCH --nodes=1
#SBATCH --time=0:01:00
#SBATCH --ntasks=1
#SBATCH --mem=100G

## Specify the working directory for this job
#SBATCH --chdir=

## Import any modules here
module load contrib/anaconda/anaconda4.4.0

## Scripts to be executed here
python test_run.py

## Clean up
exit 0
```

Any line that begins with `##` is a comment and is not read
Each line that begins with `#SBATCH` specifies info for `slurm`

The `#SBATCH` options can be provided in any order

- Name is optional
- Use stf nodes from the stf budget
- Request (1) node
- Time – Hours : Minutes : Seconds
- A single node has 28 cores and 128 GB memory
  - This dummy job is not parallel, so only (1) core
- Directory “.” is current location to read/write files
  (the same place `submit.slurm` is saved)
- Load in a pre-installed python package
- The real run command

To execute this job,

```
sbatch submit.slurm
```
Then We Wait

- sbatch <script>
  - Submits a script for non-interactive use
  - Used this to submit a job on previous slide

- squeue
  - Flag –u <uwnetid> for only your jobs
  - Specify –p or –A for whole queue

Want to check the status?

What it looks like to check queue:

```
(base) [prelj@mox1 move]$ s queue -p stf

  JOBID PARTITION  NAME  USER   ST      TIME NODES NODELIST(REASON)
  417044  stf       test_pyt  prelj  CG 0:02   1 n2282
  416180  stf       prelj     prelj  PD 0:00   1 (Dependency)
  416186  stf       prelj     prelj  PD 0:00   1 (Dependency)
  413137  stf       Other peoples job names and IDs
  415317  stf       (censored)  R 1-04:27:53 10 n[2143,2151,2166,2168,2171-2172,
  414068  stf       Other peoples job names and IDs
  413181  stf       (censored)  R 1-04:04:16 6 n[2139,2144,2146,2179,2284,2312]
```

Something wrong? / Taking too long?

- scancel <jobid>
  - Cancels an unfinished job
  - Can only cancel your own

Always check your output!

JOBID is used to refer to specific jobs
NODELIST has ID #s for individual nodes being used
ST column is status
- R – Running (currently active)
- PD – Pending (in queue)
- CG – Completing (in process of termination)
Slurm: Commands

- `sbatch <script>`
  - Submits a script for non-interactive use

- `squeue`
  - Get status of jobs in batch queue

- `scancel <jobid>`
  - Cancels an unfinished job

- `srun`
  - Submits job for interactive use or initiate job steps inside batch script
    - (More on interactive jobs in a minute)

- `sinfo`
  - Get state of partitions and nodes (is the system operational)

- `sacct`
  - Gets accounting information about active and completed jobs

---

I use these most often

`Slurm docs and man <slurm-command>` are very useful!
- (“man” for manual)
Running Jobs Interactively (command line, not script)

```
srun -p stf -A stf --ntasks=8 --mem=10G --time=0:10:00 --pty bash -l
```

- **Partition**
- **Account**
- **Number of processes (*)**
- **Amount of RAM**
- **Time**
- **Command**

Before running interactive job command
```
(base) [prelj@mox1 jesse]$
```

After running interactive job command
```
(base) [prelj@n2232 jesse]$
```

Be aware of difference between:
- Number of tasks (processes, MPI)
  - `--ntasks (-n)`
  - `--ntasks-per-node`
- Number of cpus per task (threads, OpenMP)
  - `--cpus-per-task (-c)`

Interactive nodes are computing nodes. You **can** submit jobs to other nodes from interactive nodes.

https://slurm.schedmd.com/srun.html

man srun
Loading software: the modules system

- module avail
  - Show all available modules
- module load <module>
- module unload <module>
  - (Un)load a given module
  - Provide full module name
- module list
  - List loaded modules
- module purge
  - Unload all modules
- module help
  - Print help with commands

Advanced user’s note:
The modules system works by keeping track of and modifying environment variables (e.g. PATH, LD_LIBRARY_PATH, CPATH, etc.)
https://modules.readthedocs.io/en/latest/

Running module avail

```
$ module avail
```

```
anaconda3_5.3.1 contrib/multi/20090121
anaconda3_5.3 contrib/multi/2009-027
anaconda3_4.6.1 contrib/multi/2018-06-27
anaconda3_3.6.5 contrib/multi/2018-01-08
anaconda2_5.1 contrib/multi/2018-01-08
anaconda2_5.0 contrib/multi/2018-01-08
anaconda2_4.7.1 contrib/multi/2018-01-08
anaconda2_4.6.5 contrib/multi/2018-01-08
anaconda2_4.6.3 contrib/multi/2018-01-08
anaconda2_4.6.1 contrib/multi/2018-01-08
anaconda2_4.5.1 contrib/multi/2018-01-08
anaconda2_4.4.1 contrib/multi/2018-01-08
anaconda2_4.3.1 contrib/multi/2018-01-08
anaconda2_4.2.1 contrib/multi/2018-01-08
anaconda2_4.1.1 contrib/multi/2018-01-08
anaconda2_4.0.1 contrib/multi/2018-01-08
anaconda2_3.9.1 contrib/multi/2018-01-08
anaconda2_3.8.1 contrib/multi/2018-01-08
anaconda2_3.7.1 contrib/multi/2018-01-08
anaconda2_3.6.1 contrib/multi/2018-01-08
anaconda2_3.5.1 contrib/multi/2018-01-08
anaconda2_3.4.1 contrib/multi/2018-01-08
anaconda2_3.3.1 contrib/multi/2018-01-08
anaconda2_3.2.1 contrib/multi/2018-01-08
anaconda2_3.1.1 contrib/multi/2018-01-08
anaconda2_3.0.1 contrib/multi/2018-01-08
anaconda2_2.9.1 contrib/multi/2018-01-08
anaconda2_2.8.1 contrib/multi/2018-01-08
anaconda2_2.7.1 contrib/multi/2018-01-08
anaconda2_2.6.1 contrib/multi/2018-01-08
anaconda2_2.5.1 contrib/multi/2018-01-08
anaconda2_2.4.1 contrib/multi/2018-01-08
anaconda2_2.3.1 contrib/multi/2018-01-08
anaconda2_2.2.1 contrib/multi/2018-01-08
anaconda2_2.1.1 contrib/multi/2018-01-08
anaconda2_2.0.1 contrib/multi/2018-01-08
anaconda2_1.9.1 contrib/multi/2018-01-08
anaconda2_1.8.1 contrib/multi/2018-01-08
anaconda2_1.7.1 contrib/multi/2018-01-08
anaconda2_1.6.1 contrib/multi/2018-01-08
anaconda2_1.5.1 contrib/multi/2018-01-08
anaconda2_1.4.1 contrib/multi/2018-01-08
anaconda2_1.3.1 contrib/multi/2018-01-08
anaconda2_1.2.1 contrib/multi/2018-01-08
anaconda2_1.1.1 contrib/multi/2018-01-08
anaconda2_0.9.1 contrib/multi/2018-01-08
anaconda2_0.8.1 contrib/multi/2018-01-08
anaconda2_0.7.1 contrib/multi/2018-01-08
anaconda2_0.6.1 contrib/multi/2018-01-08
anaconda2_0.5.1 contrib/multi/2018-01-08
anaconda2_0.4.1 contrib/multi/2018-01-08
anaconda2_0.3.1 contrib/multi/2018-01-08
anaconda2_0.2.1 contrib/multi/2018-01-08
anaconda2_0.1.1 contrib/multi/2018-01-08
anaconda2_0.0.1 contrib/multi/2018-01-08
anaconda1_4.7.1 contrib/multi/2018-01-08
anaconda1_4.6.1 contrib/multi/2018-01-08
anaconda1_4.5.1 contrib/multi/2018-01-08
anaconda1_4.4.1 contrib/multi/2018-01-08
anaconda1_4.3.1 contrib/multi/2018-01-08
anaconda1_4.2.1 contrib/multi/2018-01-08
anaconda1_4.1.1 contrib/multi/2018-01-08
anaconda1_4.0.1 contrib/multi/2018-01-08
anaconda1_3.9.1 contrib/multi/2018-01-08
anaconda1_3.8.1 contrib/multi/2018-01-08
anaconda1_3.7.1 contrib/multi/2018-01-08
anaconda1_3.6.1 contrib/multi/2018-01-08
anaconda1_3.5.1 contrib/multi/2018-01-08
anaconda1_3.4.1 contrib/multi/2018-01-08
anaconda1_3.3.1 contrib/multi/2018-01-08
anaconda1_3.2.1 contrib/multi/2018-01-08
anaconda1_3.1.1 contrib/multi/2018-01-08
anaconda1_3.0.1 contrib/multi/2018-01-08
anaconda1_2.9.1 contrib/multi/2018-01-08
anaconda1_2.8.1 contrib/multi/2018-01-08
anaconda1_2.7.1 contrib/multi/2018-01-08
anaconda1_2.6.1 contrib/multi/2018-01-08
anaconda1_2.5.1 contrib/multi/2018-01-08
anaconda1_2.4.1 contrib/multi/2018-01-08
anaconda1_2.3.1 contrib/multi/2018-01-08
anaconda1_2.2.1 contrib/multi/2018-01-08
anaconda1_2.1.1 contrib/multi/2018-01-08
anaconda1_2.0.1 contrib/multi/2018-01-08
anaconda1_1.9.1 contrib/multi/2018-01-08
anaconda1_1.8.1 contrib/multi/2018-01-08
anaconda1_1.7.1 contrib/multi/2018-01-08
anaconda1_1.6.1 contrib/multi/2018-01-08
anaconda1_1.5.1 contrib/multi/2018-01-08
anaconda1_1.4.1 contrib/multi/2018-01-08
anaconda1_1.3.1 contrib/multi/2018-01-08
anaconda1_1.2.1 contrib/multi/2018-01-08
anaconda1_1.1.1 contrib/multi/2018-01-08
anaconda1_1.0.1 contrib/multi/2018-01-08
anaconda1_0.9.1 contrib/multi/2018-01-08
anaconda1_0.8.1 contrib/multi/2018-01-08
anaconda1_0.7.1 contrib/multi/2018-01-08
anaconda1_0.6.1 contrib/multi/2018-01-08
anaconda1_0.5.1 contrib/multi/2018-01-08
anaconda1_0.4.1 contrib/multi/2018-01-08
anaconda1_0.3.1 contrib/multi/2018-01-08
anaconda1_0.2.1 contrib/multi/2018-01-08
anaconda1_0.1.1 contrib/multi/2018-01-08
anaconda1_0.0.1 contrib/multi/2018-01-08
```

> 400 modules available on Hyak right now!

Useful programs you’d use on your own computer, i.e.
- Python/Anaconda, Gromacs, NAMD, R, Mathematica, Matlab, Gaussian, compilers and more
- Parallel computing tools
- CUDA, IMPI, etc.
Checkpoint queue – special case

The checkpoint queue allows any user to run on other groups’ unused nodes!

- Partition: ckpt
- Account: <group>-ckpt (stf-ckpt)
- Jobs can be interrupted at any time
- Jobs will be interrupted after 4 contiguous hours
- Jobs will be resubmitted after interruption (if under total requested time limit)

**Your code must be checkpointed to take advantage of this**
- Save a binary file containing state of some objects, restart from logs, etc.
- Your script should also account for any checkpointing that is done
Topics not covered

- How to install new software
  - Installation and build systems
  - Writing your own modulefiles
- How to parallelize your code
  - Different paradigms for parallelism
  - Data locality and contiguity
  - GPU parallelization
  - Parallel patterns
  - Some programs have parallelization options built-in
- Anything with the cloud (This is the Hyak training session!)
- General architecture of HPC systems
  - Interconnectivity and node locality
  - Physical infrastructure
Where to get help

- Documentation (man or webpages)
- Hyak wiki: https://wiki.cac.washington.edu/display/hyakusers/WIKI+for+Hyak+users
- Slack channel: https://uw-rcc.slack.com/
- Website: https://depts.washington.edu/uwrcc/
- Emails: hpcc@uw.edu or uwrcc@uw.edu
- Office hours: Fridays from 1-3 pm
Happy computing!