Cal Poly Pomona HPC Handbook For Students

Introduction

As part of its commitment to providing a state-of-the-art research and teaching environment to its faculty and students, Cal Poly Pomona operates a High Performance Computing (HPC) Cluster, managed by the campus Division of Information Technology & Institutional Planning. The new system entered production as a campus-wide resource in the Fall 2017 term and has been used to teach multiple courses in multiple disciplines. It is also currently being used as a resource for NSF-funded research and as a teaching resource in Cyber Security projects for Cal Poly Pomona undergraduates.

The new CPP HPC system is designed to serve researchers across campus. Initial research opportunities have already been identified within the Colleges of Science, Engineering, Business and CLASS and work is currently underway to integrate the HPC Cluster service into the campus Cyber Security Instructional Research Project (CSIRP), which serves as a resource hub for cross-disciplinary research in this growing field.

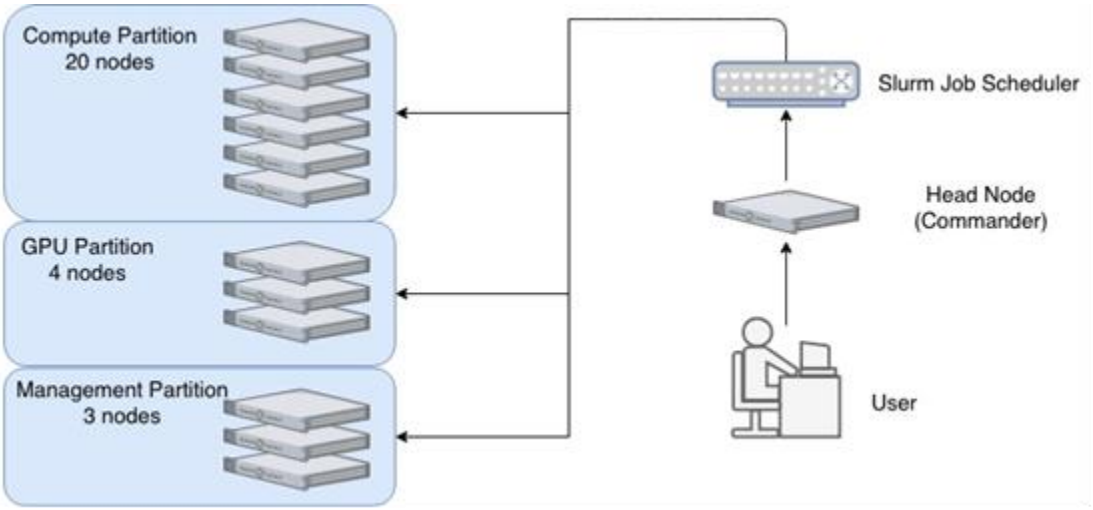
Hardware and Specifics

The High Performance Computing Cluster consists of multiple dedicated processor nodes, connected together via a specialized high-speed network and managed by specialized job scheduling software. The CPP HPC software management suite utilizes the HP Enterprises HPC Software Stack, which includes the open source S*lurm* job scheduler, *Insight CMU* for cluster Management, and other HPE software for node deployment and configuration.

The Slurm scheduler manages the allocation, dispatching and execution of jobs. Slurm is a popular and well documented package currently used by a large number of campus HPC systems and one that allows a task to be dispatched in a variety of ways, including allowing jobs to be run in real time, or in batch mode for longer running tasks.

The CPP HPC Cluster nodes are configured as a set of “partitions” to allow dispatching jobs to appropriate nodes for a variety of computational tasks. The “General Compute Partition” is used for general purpose jobs that benefit from running multiple compute tasks in parallel, while the “GPU Partition” allows a task to access dedicated GPU processors where the task would benefit from additional numerical processing capability.

The new CPP HPC cluster is based upon the HP Proliant server platform and currently includes a total of two DL360 management nodes, 20 DL160 compute nodes, and four GPU nodes with a total of 8 Tesla P100 GPUs The cluster contains 3.3TB of RAM and is connected through a dedicated internal 40GBit Infiniband switching fabric and 10 GBit external ethernet connections. The overall system throughput is currently approximately 36.6 Tflp in double precision mode or 149.6 Tflp in half precision mode. This current configuration is expected to grow over time, as researchers identify collaborative research initiatives and develop future funding for expansion of the system through external grants and donations.



Getting Started on the HPC

The CPP HPC system is available to all faculty for either individual research or instructional use and is made available to students either because they are taking a specific course which requires access, or as a resource needed as part of a supervised project.

Gaining Access

If access to the HPC is required for a course, your professor will arrange for account activation at the start of the term and your access will remain active through to the end of the semester. If you require access for a specific project you wish to develop, your supervising professor should contact the IT support team and your account will be created. Details on how to do this are available in a companion *HPC Guide for Faculty*.

Student quotas

Each student account is also currently being given 25GB of file storage space by default.

Note: Access to the GPU partition must be explicitly requested when a job is run with the slurm scheduler and these also have their own restrictions. By default, student jobs are restricted to 2 GPUs, access to more may be granted, if needed.

Familiarizing yourself with the Technology

Logging On

To access and run jobs on the CPP HPC system, you must first log onto the system Command Partition using your CPP login credentials (BroncoName and associated password). To do this, you will need access to an ssh-enabled terminal emulation program, such as PuTTY on Windows, iterm2 for Mac or the Linux ssh command https://www.ssh.com/ssh/command.

When using the GUI-based tools, you will need to specify the hostname (“hpc.cpp.edu”), as well as the connection type (“SSH”) and port number (22). You should be prompted for your login name and password and once supplied, be placed into the Command partition shell.

To do this on a Linux machine, type the following command:

```
ssh -l bronconame hpc.cpp.edu
```

NOTE: You must be connected to the campus network, or be connected to the campus network via VPN in order to access the CP HPC cluster. To connect the CPP VPN:

refer to this eHelp article.

Familiarizing Yourself With the Command Line

Once logged on, you will find yourself in a traditional Linux command shell with the usual associated system commands, as well as access to the Anaconda package management system, the slurm job scheduler and other related tools. In this section, we provide a brief overview of the most common Linux commands, a brief summary of the Ananconda package management system and basic slurm command line options.

Common Linux Commands

The following Table summarizes some of the most common Linux commands you will need:

Useful Linux Commands		
Command	**Summary**	**Example**
`cd path`	Changes directory	`cd project`
`ls`	Lists all items in specific directory	`ls`
`cat filename`	Displays contents of a file	`cat example.sh`
`touch newfilename`	Creates a new file by that name	`touch example`
`chmod filename`	Changes the permissions of the file Options: (r, w, x) read write execute	`chmod +x example.sh`
`time command`	Written before any command, it outputs the time the command took. In the output, it will show three different times: – Real- wall clock time from start to finish – User- amount of CPU time spent executing the program – Sys - the amount of CPU time spent in the kernel	`time hostname`
`man command`	Shows the manual page for the command	`man time`

View additional Linux commands here

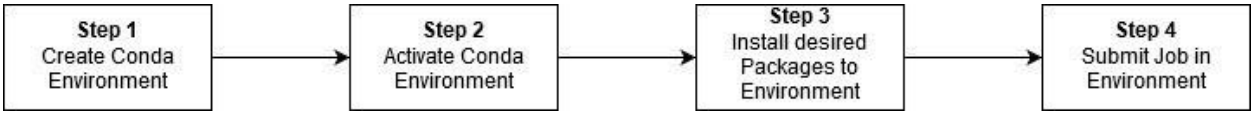
Refer here for an introductory Linux tutorial

Introduction to Anaconda

If you are developing a basic HPC program from scratch, you will likely need only a text editor and the common Linux command line options to create and manipulate your files, but if your work involves importing and configuring a system or package that has been developed elsewhere you will need access to a package management tool. Some programming languages allow you to import modules directly into your projects. In addition, HPC users have access to the Anaconda package management system, which is pre-installed on the Command Partition head node.

Many projects require providing access to specific versions of supporting packages and dependencies which could, if installed globally could potentially cause problems for other users due to mismatches between specific package versions. To avoid this problem, Anaconda allows you to import needed packages, as well as any needed dependencies, into your local file system, allowing you to run your project independently, when needed.

You will need to carry out the following steps to install a set of packages to your home directory:



The following Table summarizes the Commands needed to carry out these steps:

Anaconda Cheat Sheet	
Function	**Command**

Create Conda Environment	`conda create -n environmentname`
	
	Or, if you already know what package is needed to be installed, try:
	
	`conda create -n environmentname packagename`

List Conda Environments	`conda env list`
Activate Conda Environment	`conda activate environmentname`
Search for available packages to install	`conda search packagename`
Install package to environment	`conda install packagename`
Submit Job in Environment	Refer to “Running a Program” below
Deactivate Environment	`conda deactivate`

Running a Job with Slurm

Know common SLURM commands as these will be used to submit and manage jobs on the HPC

Useful SLURM Commands		
Command	**Summary**	**Example**
`srun -parameter job`	Obtain a job allocation and execute an application (see Running a Program - SRUN)	`srun -N1 -n1 example.py`
`sbatch -parameter batchscript`	Submit a batch script for later execution (see Running a Program - SBATCH)	`sbatch -o example.sh`
`sacct`	Display accounting data Use -j jobid to see status of specific job	`sacct -j 1576`
`sinfo`	View the status of the cluster's nodes and partitions	`sinfo`
`squeue`	Displays information of jobs in queue	`squeue`

View additional Slurm commands here

Running a Program

There are two ways to run a program in the cluster: the command `srun` and the command `sbatch`.

SRUN

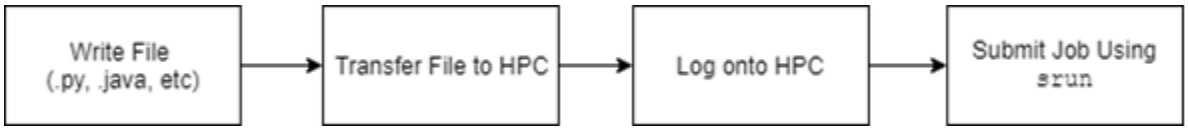


Diagram showing process of submitting a job through srun

Jobs submitted through the command srun are interactive, meaning they run in the foreground rather than the background. Using the command `srun` runs a parallel job on the cluster managed by Slurm. It interrupts the command line to run the job, and the output goes directly to the command line unless an output file is specified. As an example, type into your command line:

```
srun hostname
```

The node you are currently on is printed in the terminal. The head node, the one that you will begin on, is called commander.

To run a program you have written, the commands vary for different interpreters. To run a java program, first compile the .java file with:

```
srun javac filename.java.
```

Then to run the file:

```
srun java filename
```

Srun has many parameters that will be helpful and necessary when running your program. Below is a chart of different parameters that are often useful when using srun. For a more comprehensive manual on the different parameters and functions of srun, refer here.

Srun Parameters/Options

Command	Summary	Example
`-N#`	Amount of nodes to be used for this job	`-N1` Informs the head node to allocate 1 node for this job
`-n#`	Number of tasks in this job	`-n1` Establishes that there is 1 task in this job
`-p` or `--partition=partition name`	Specific partition to be used for this job	`-p compute` Tells the head node to use the compute partition
`--mem=<MB>`	Memory to be allocated for this job	`--mem=2GB`
`-o filename`	Establishes a file to show the output	`-o program.out`
`-w hostname`	Tells Slurm to allocate a specified node	`-w cn04`
`--test-only`	Slurm will merely test your job and find out when your job is estimated to run, but does not actually submit job for execution	`--test-only`

With srun you can also set up an interactive session. Type

```
srun -N1 -n1 -p gpu --pty bash
```

This will begin an interactive session with one node in the gpu partition. You can run any number of commands within this shell. Once in the interactive session, you have direct access to the node. You can see the modules that the node has with the command

```
module avail
```

This will show the resources that this particular node has. Here is a list of commands that are useful when navigating through the modules of a node:

`module load` - command adds application to your PATH variable pulling dependencies`module purge` - removes all currently loaded modules `module unload` - removes a specific module`module avail` - browses entire list of modules

SBATCH

Diagram showing process of submitting a job through sbatch

The command `sbatch` submits a batch script (it must be a batch script, not any other type) to Slurm that allocates the job a Job ID and submits it for later execution. Once the job is submitted, you will receive this message:

```
Submitted batch job jobid
```

You can see the queue at any time using the command `squeue`. To see the list of jobs you have submitted, use the command:

```
squeue --user bronconame
```

By default, the output is sent to a new file named `slurm-jobid.ou`. However, you can change this default by using the parameter `-o filename`. This will print the output onto the file specified instead of creating the default.

Since both srun and sbatch submit a job to SLURM for the schedule manager to run, the parameters for both commands are identical.

For more information about sbatch, the manual for the command can be found here.

Batch Scripts

A batch file is a text script that contains certain commands executed in sequence. Batch scripts enable the user to run multiple jobs in one command line. There are three parts to a batch script.

The first part is the program that will run the script. For a batch script, this is always:

```
#!/bin/bash
```

The second part of the batch script consists of directives for any jobs that are going to be called in the batch script. These directives tell Slurm the computational resources that the script requires. The directives use

```
#SBATCH -parameter
```

The most commonly used directives for batch scripts are shown below:

Common Batch Script Directives

Directive	Description	Example
`--job-name=name` or `-J name`	Custom job name	`--job-name=example`
`--partition=name` or `-p partition`	Partition to run on	`--partition=compute`
`--nodes=#` or `-N#`	Total number of nodes	`--nodes=1`
`--ntasks=#` or `-n#`	Number of "tasks". For use with distributed parallelism. See below.	`--ntasks=1`
`--cpus-per-task=#` or `-c#`	# of CPUs allocated to each task. For use with shared memory parallelism.	`--cpus-per-task=1`
`--ntasks-per-node=#`	Number of "tasks" per node. For use with distributed parallelism. See below.	`--ntasks-per-node=2`
`--time=[[DD-]HH:]MM:SS` or `-t [[DD-]HH:]MM:SS`	Maximum walltime of the job in Days-Hours:Minutes:Sec	`--time=10:00` 10 minutes
`--mem=#`	Memory requested per node in MB	`--mem=1G`

The final part is the actual script, consisting of commands to be run. Run jobs using `srun` as normal. You can reference other files and programs within the batch script to run.
Once the batch file is created, you must ensure that the user has permission to execute the script. To check the permissions of the file, use the command

```
chmod  filename
```

To give the user permission to execute the file, use the command

```
chmod +x filename
```

Other Useful Tools

Copying Files From Your Local Machine to the HPC

Using WinSCP or FileZilla
Log on with your bronconame and the cluster hostname: `hpc.cpp.edu` . Use the graphical interface to drag and drop files from local machine to the HPC. Alternatively, you may use your preferred FTP method.

NOTE: After copying a file, use command `chmod +x filename` to establish appropriate executable permissions

Available Languages and Resources

A variety of software is installed on the cluster for student and faculty use, including but not limited to:

Available Languages and Resources				
Python	Java	R	Keras	Pandas
Bioperl	Conda	Pycuda	Singularity	Tensorflow

In order to see which libraries are available on specific nodes, start an interactive session (see **Running a Program** for more information) and type:

```
module avail
```

Next Steps

You are now ready to get started with the Cal Poly Pomona HPC system. If you have any additional questions additional information may also be found online at:

https://www.cpp.edu/lrt/hpc/hpc-support.shtml