An introduction to the HPC at UCI: “The HPC theme park"

Author: Octavi Obiols (oobiols@uci.edu)
March 2020

This document provides an introduction to the campus wide HPC cluster, and it is part of the Lab 1 module of courses EECS120 and EECS224. Reading this document should be the first thing the student should attempt before watching the video. The video is a short summary of the information provided here and its motivation is to have a visualization of the definitions given in this document.

The HPC Cluster at UCI

HPC stands for High-Performance Computing Cluster, and it is a supercomputer part of the UCI community. To understand what HPC exactly is and how to use it, we will go over a dummy analogy throughout all this document. We will be switching back and forth between the nomenclature in the analogy and the actual HPC-related language, so that we can get, in an easy way, a sense of what HPC is/can do.

So let’s start picturing a theme park (like Disneyland) in our minds. We go to a theme park because we want to have fun. When we arrive in a theme park, the first "stand" we usually bump into is the reception desk — where we buy our tickets, talk to a staff member, etc. Once we are given the tickets, we can enter the park where we can decide what attraction to go to according to our preferences, desires, and fun goals. Each of these attractions will provide different types of fun. Once at a specific attraction, we can decide how we are going to make use of it: maybe we only want to explore half of the attraction or maybe we want to explore it entirely. Perhaps, we only want to take some pictures.

A different and possible situation is that we’ve been so many times in Disneyland that we know every inch of it, so we decide to take there our friends and wait in the reception desk while they have fun. Very importantly, we were the ones who recommended our friends what attractions to go to and what type of fun they would find in each one of them.

Let’s go back to HPC now and imagine that HPC is a theme park, with many attractions. Now, we do not go to this theme park to “have fun”, but we go there to **run our software,** or in HPC terms, **run our job** or simply **run a job** (while having fun, obviously :D). When we run our job, we do it on one of the "attractions" existing in HPC. Even though once in the HPC we can generate code and our software idea, the main goal of HPC is to be able to run it “efficiently”— more on this in the class lectures.

We do not go to HPC by driving, obviously, but we go there by virtually (through the internet) connecting to it using the terminal of our laptop. So, let's do this: let's "go" to HPC.

1) Let’s make sure we have a UNIX-like terminal in our laptop (for instance, a Mac, or we have a Linux OS installed). For Windows instructions, please refer to https://

www.howtogeek.com/311287/how-to-connect-to-an-ssh-server-from-windows-macos-or-linux/

2) Open this terminal.
3) Type and enter the following:

```
ssh userid@hpc.oit.uci.edu
```

Side note: a summary of all commands related to the usage of HPC is also provided at the end of this document.

If we have been given access to HPC by the administrators, the *userid* is usually the same as the one in our UCI email. For example: if the UCI email is mhernan23@uci.edu, the *userid* for connecting to HPC is *mhernan23.*

4) The terminal will ask for our password. Type it and press enter. Remember that the password **does not show,** but we promise that it is indeed being typed.

We will see some colors printed in the terminal and an HPC welcome message. Congratulations! now we are connected to HPC. Once we connect to it, we could say that we have arrived in the theme park. Of course, the first place we bump into is the “reception desk”, or in HPC terms, what is called the **login node. Now, we are in the login node.**

Again, let’s place our mind in the actual theme park. Imagine that everybody going to the theme park that day starts to have fun (taking pictures, eating, talking) in the reception and never dare to access the park. The reception would rapidly be full of thousands of people and would easily collapse, while nobody would be using the attractions (they'd be empty). This analogy works for the login node in HPC: it is very important to understand that **no job must be run on the login node**, because if everybody did, it would crash. This is the most important rule when “going to” HPC. However, other actions, like modifying our code or compiling it, are allowed in the login node.

We can do a lot of important things that are not running our job in the login node. The first thing to do is to check if we were taken to our directory (this directory is ours and we can save anything we want, up to 50GB). Now, let’s type and enter:

```
pwd
```

And we should get something like:

```
/data/users/userid
```

Let's continue and let’s start doing what we came to do in the HPC theme park. Still in the login node, we can check what “attractions" are open and available, and how busy they are, the same way you can check the map of Disneyland when you are in the reception desk. We can check what "attractions" we can use to "have fun", that is, what attractions we can use to

run our job efficiently. HPC, like Disneyland, has different attractions — in HPC terms, **machines** or **queues.** In basic terms, **HPC is a group of independent queues.** “I have never heard this word before, what is a queue!?"

A **queue** is a group of computers (**nodes**) connected through a network (yes, an actual physical network of cables). Each of these nodes can have more than one “computing unit"— called **core***. For instance, each core of a node can handle one task. So, if our code has independent tasks in it, our code can run "in parallel”: each task would be handled independently - and *in parallel* (i.e., at the same time) - by its assigned core.

*This definition of core or computing unit is not theoretically precise and should be extended and more taken care of, but we use it to have our first mental approach to HPC. Its correct definition will be discussed throughout the class lectures.

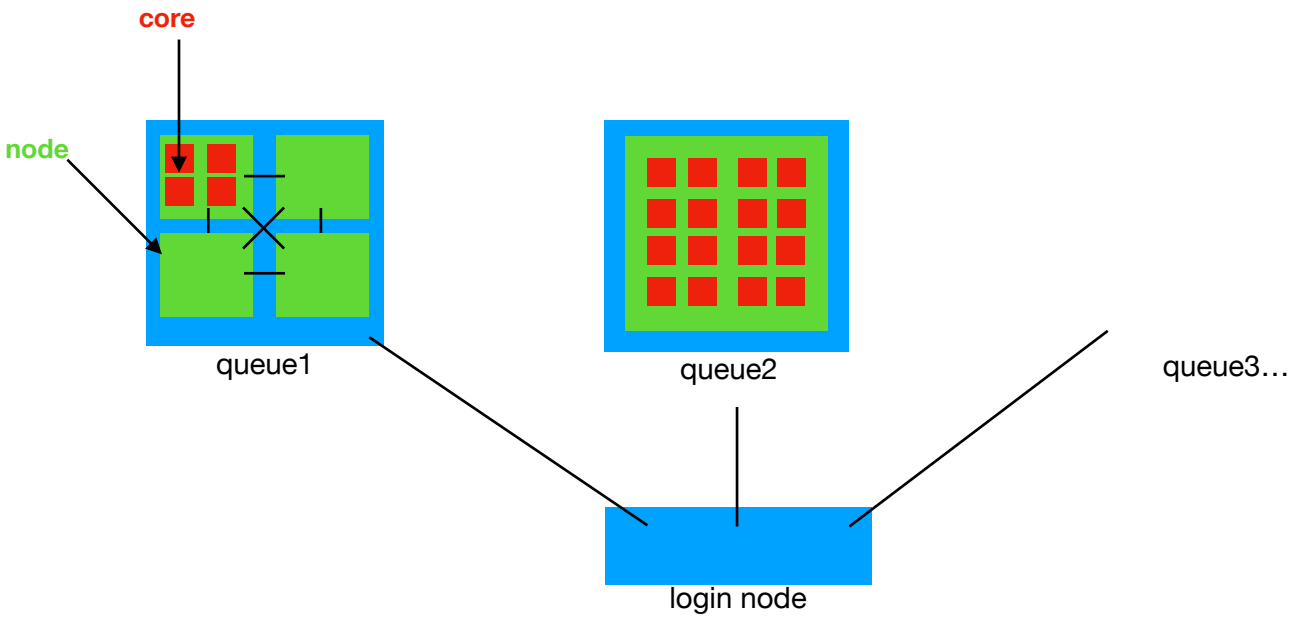

Figure 1. A scheme (idea) of what HPC is. HPC has different independent queues. Each queue consists of connected nodes that have cores. Queue 2, for instance, only has one node with many cores.

Let's take a look at what queues (or “attractions") are existing and available in HPC. In the login node, let's type and enter a single letter:

```
q
```

And we get something like:

```
we have access to the following HPC Queues:

PUBLIC QUEUES:
+------------------+-------+-------+--------+---------+-----------+-----------+
|        Queue     | Wall- | Cores |        |         |           | Available |
|        Name      | Clock | Per   | Nodes  |  Cores  | Available | WHOLE     |
| (S)=Suspend-able | Limit | Node  | Total  |  Total  | Cores     | Nodes     |
+------------------+-------+-------+--------+---------+-----------+-----------+
         pub8i       72:00      8      86       688        128          7
         pub64       72:00     64       7       448          8          0
          epyc       72:00    128       1       128         26          0
       free128 (S)   72:00     64       1        64          0          0     <-- Queue full.
        free64 (S)   72:00     64      94      6016         53          0
        free48 (S)   72:00     48       2        96         16          0
       free88i (S)   72:00     88       6       528          0          0     <-- Queue full.
       free72i (S)   72:00     72       6       432         72          1
       free56i (S)   72:00     56       1        56          0          0     <-- Queue full.
       free48i (S)   72:00     48       1        48          0          0     <-- Queue full.
       free40i (S)   72:00     40       1        40          9          0
       free24i (S)   72:00     24       5       120          0          0     <-- Queue full.
       free32i (S)   72:00     32       4       128         32          0
       free16i (S)   72:00     16       1        16          0          0     <-- Queue full.
       gpu1080 (S)   72:00     88       5       440         22          0
          gpu2       72:00     24       1        24         24          1
           gpu       72:00     24       1        24         20          0
   interactive        none    128       1       128         48          0
        ionode      168:00      1       7         7          7          7
     ionode-lp      168:00      1       1         1          1          1
       jupyter      240:00      8       1         8          8          1
          test       00:05     64       7       448        448          7
   free32i-gpu (S)   72:00     32       1        32          0          0     <-- Queue full.
```

The most-left column shows all the independent existing queues. Let's take a look at the first one: `pub8i`.

`pub8i` is the name of the queue. If we take a look at the "nodes total" column, it indicates how many nodes that queue has (that does not mean all of them are available!). We see that pub8i has 86 nodes (86 "computers").
Each of these nodes, as seen in the "Cores per node" column, consists of 8 cores — or processing units. In a sense, each of these nodes has 8 "CPUs", named *cores*.
If we have 86 total nodes and 8 cores per node, we have a total (86*8) of 688 cores. Because other people are in this theme park and also have access, the last two columns show how "busy" that queue is. We can see that from the 688 total cores, only 128 are available. The rest are being used right now (at the time this document is being made).

The "available cores" and "available WHOLE nodes" column tells us how many cores we have *available to use right now* in each queue, so it is an important column to take a look at.

Now that we have checked and understood what is the current state of all queues, we can use them to run our jobs:

Before, we described that we can personally head to the attractions in the theme park, or we can stay at reception waiting for our friends to enjoy the rides. We also have these two options in HPC: the former is **running an interactive job**, while the latter is **running a batch job.**

<u>Running an interactive job</u>

Here, we decide to personally "go" to one “attraction”: we want to “go” to a queue. In HPC terms: we connect to a node of a queue by requesting it. The command to “go” to a queue is the following. Type and enter:

qrsh -q *name_of_the_queue*

For instance:

```
qrsh -q interactive
```

(We could also have typed, for instance, `qrsh -q pub8i`, to run interactively a job in the pub8i queue). Connecting to it might take some seconds (maybe minutes, but that would be uncommon). This is what we are returned:

```
Warning: Permanently added '[hpc-interactive-1-1.local]:40510,[10.1.255.164]:40510' (RSA) to the list of known hosts.
Last login: Tue May 28 13:18:30 2019 from hpc-login-1-2.local
 ______________________________________________________________
/\                                                             \
\_|                 _   _ ____   ____                          |
  |                | | | |  _ \ / ___|                         |
  |                | |_| | |_) | |                             |
  |                |  _  |  __/| |___                          |
  |                |_| |_|_|    \____|                         |
  |                                                            |
  |  ___       _                      _   _               __   |
  | |_ _|_ __ | |_ ___ _ __ __ _  ___| |_(_)_   _____    / /   |
  |  | || '_ \| __/ _ \ '__/ _` |/ __| __| \ \ / / _ \  / /    |
  |  | || | | | ||  __/ | | (_| | (__| |_| |\ V /  __/ / /     |
  | |___|_| |_|\__\___|_|  \__,_|\___|\__|_| \_/ \___|/_/      |
  |                                                            |
  |  _     _       _     _   ____       _   _                  |
  | | |   (_) __ _| |__ | |_|  _ \ __ _| |_| |__               |
  | | |   | |/ _` | '_ \| __| |_) / _` | __| '_ \              |
  | | |___| | (_| | | | | |_|  __/ (_| | |_| | | |             |
  | |_____|_|\__, |_| |_|\__|_|   \__,_|\__|_| |_|             |
  |          |___/                                             |
  |                _   _           _                           |
  |               | \ | | ___   __| | ___                      |
  |               |  \| |/ _ \ / _` |/ _ \                     |
  |               | |\  | (_) | (_| |  __/                     |
  |               |_| \_|\___/ \__,_|\___|                     |
  |                                                            |
  |   _________________________________________________________|_
   \_/___________________________________________________________/

Note: NFS data traffic going outside HPC WILL NOT work from this
node.  Also software may not be able to get to license servers.
Both will work from all other HPC nodes however.
```

Congratulations! We are no longer in the login node but we "have gone" to **one node of the interactive queue.** Here, we can do anything we want: compile and run our code as the most important actions items.

However, since we did not specify anything else in the command before, we are only given **one core** of **one node** of the **interactive queue** while **still being in our home directory** (/data/users/userid).

In this environment ("going to a queue"), we **can not** request more than one node. However, we can request more than one core! The command would be the following. Type and enter:

```
qrsh -q interactive -pe openmp 2
```

And now we are going to one node of the interactive queue and we have been given 2 cores to use. Remember, this will work as long as there are “available cores", so it is worth to constantly check the state of the queues. If one queue is fully busy at one moment, we will not be able to go there and we will not be given any core.
Also, it is important to note that we can not request more than the # of cores available in one node. For instance, the interactive queue has one node with 128 cores. If all 128 cores were available, we could request up to 128 cores. The pub8i queue has 8 cores per node. So, interactively, we can also request up to 8 cores.

What does `-pe` mean? It stands for “parallel environment” and it is **a mandatory command** to request more than one core. `openmp` is the name of the parallel environment, and it is also necessary to be included to request more than one core — describing and explaining openmp is not the goal of this document and therefore it will not be covered here.

Running a batch job

Now, we are back at the login node. What if instead of personally going to the queue and run our job there, we wanted to just “send" the job to that queue while we stay at the reception? We have the option to remain at the login node and send our job to be run in the queue that we want.

Specifically, here we need to submit a batch script (hence the name of this section) that gives the order of sending our job to the queue that we want with the number of cores and nodes that we want. We need to create a batch script file, that is, something like *filename.sh.*

What does this script file look like? Here it is an example:

filename.sh:

```
#!/bin/bash
#$ -N TEST
#$ -q pub64
#$ -pe one-node-mpi 128
#$ -R y
#$ -m beas

./myprogram
```

`-N TEST`, set the name of the job as TEST.

`-q` , as before, is the name of the queue from which we want to request nodes and cores.

`-pe` is the name of the parallel environment, here “one-node-mpi”. 128 is the number of cores we want to use.

`-R y` is job reservation, and it is necessary for MPI jobs (more on MPI in the class lectures).

`-m beas` sets an order to be notified by email (to our UCI email) when the job starts.

`./myprogram` is the command to run "myprogram" in the queue that we chose.

Now that we have our batch script ready and we understand what is in it, we have to submit it (send it)! The command to do so is the following. Type and enter:

```
qsub filename.sh
```

And this command will take care of submitting the job and start running it. We can check the status of the job (see if it started, it is waiting to start, it is finished, etc.) by the following command. Type and enter:

```
qstat -u userid
```

One of the main differences between an interactive job and a batch job is that here we can request and use more than one node of one queue. To do that, we take a look at the `-q` and `-pe` commands:

```
-q pub64
-pe one-node-mpi 128
```

We are requesting 128 cores of the pub64 queue. Since the list before told us that the queue pub64 has 64 cores per node, we conclude that now we are requesting 2 full nodes (64*2 = 128). This is the way of requesting more than one node: telling the batch script file that we want X number of cores, where X = (cores per node) * (nodes we want). Another example: let’s say we want to use 3 nodes of the pub8i queue. Since each node has 8 cores, we would be requesting (3*8=24):

```
-q pub8i
-pe one-node-mpi 24
```

Modules in HPC

HPC provides some services that the user can found useful. As an analogy: in a theme park, we can ask if there are food chains, cleaning services, gardening, etc.

HPC provides what are called modules, and they are used to load the right libraries, compilers, etc. They are very important (often indispensable, and you will use them quite a lot!) because they allow us to use complimentary software/libraries/compilers that are required for our job to run properly. For instance, let's say that we need to compile our job with a specific compiler type. This compiler is not provided by default by HPC. We need to load it.

To see what modules HPC has, type and enter:

```
module avail
```

From this command, we get a very long list. It is not comfortable to look for a module in this huge list. Therefore, it is good to know the type of module we are after. Let's say that we want to load the intel compiler. We type and enter:

```
module avail intel
```

And this is what we get:

```
------------------------------------------------------- /data/modulefiles/SOFTWARE --------------------------------------------------------
intel-mkl/2019.0.117 intel-tbb/2019_U2

------------------------------------------------------- /data/modulefiles/COMPILERS -------------------------------------------------------
intel/12.0.5                    intel-parallel-studio-xe/15.0.3
intel-mkl/12.0.5                intel-parallel-studio-xe/2018.3.051
```

These are all the intel modules available. Let's say that we want to use the intel 12.0.5 compiler. Since by default it is not loaded in HPC, we need to load it to be able to use it. Type and enter:

```
module load intel/12.0.5
```

To check what modules we have loaded, we can type and enter:

```
module list
```

And this is what we get:

```
Currently Loaded Modulefiles:
  1) gcc/6.4.0          2) Cluster_Defaults   3) intel/12.0.5
```

A more exhaustive and more professional guide for using the HPC in UCI can be found in:

- Introduction: https://hpc.oit.uci.edu/HPC_USER_HOWTO.html
- How to run jobs in HPC: https://hpc.oit.uci.edu/running-jobs

However, the guide in this document serves as a primary introduction to get familiar with the nomenclature and the usage of HPC if it is the very first time we use it — or even hear about it.

Summary of the commands explained in this document:

ACTION:	COMMAND:
Open the terminal and access HPC	ssh userid@hpc.oit.uci.edu
Print current directory	pwd
Check the state of the queues	q
Request one core of one node of the queue named interactive	qrsh -q interactive
Request N cores of one node of the queue named interactive	qrsh -q interactive -pe openmp N
Submit a batch job (through a file named *filename*)	qsub filename.sh
Check status of our submitted job	qstat -u *userid*
print all modules avaialble	module avail
print all modules avaialble of the gcc compiler	module avail gcc
Load the module gcc	module load gcc
See what modules we have loaded	module list