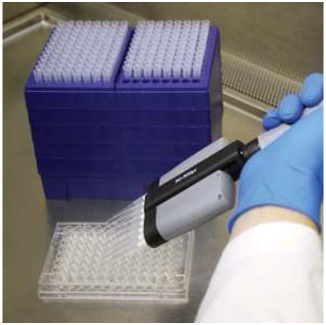
UMass High Performance Computing Center

University of Massachusetts Medical School

November, 2016

Challenges of Genomic Data

It is getting easier and cheaper to produce bigger genomic data every day. Today it is not unusual to have 100 samples getting sequenced for a research project. Say, we have 100 samples sequenced and each sample gave us about 50 million reads. It may easily take half a day to process **just one** library on a desktop computer.



Why Cluster?

Massive data coming from Deep Sequencing needs to be

- stored
- (parallel) processed

It is not feasible to process this kind of data even using a high-end computer.

MGHPCC

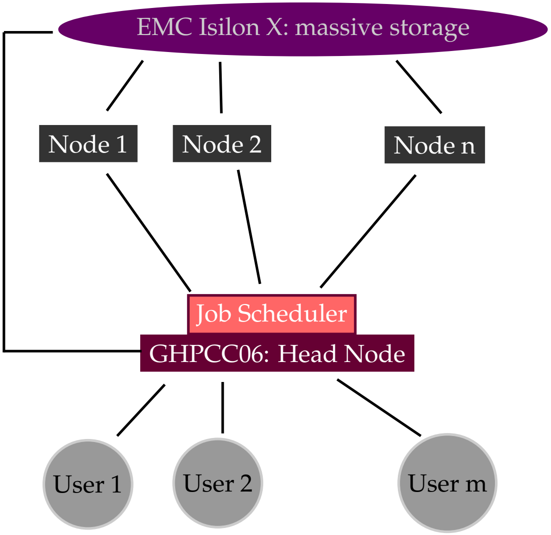
University of Massachusetts Green High Performance Computing Cluster

HPCC $\equiv$ GHPCC $\equiv$ MGHPCC $\equiv$ the Cluster

HPC	:	High performance computing
Cluster	:	a number of similar things that occur together
Computer Cluster	:	A set of computers connected together that work as a single unit

MGHPCC has over 10K+ cores available and 400+ TB of high performance storage. It is located in Holyoke MA and provides computing services to the five campuses of UMass.

Overview


EMC Isilon X: massive storage
Node 1
Node 2
Node n
Job Scheduler
GHPCC06: Head Node
User 1
User 2
User m

Storage Organization

Though there are many file systems mounted on the head node, there are three file systems that are important for us.

Type	**Root Directory**	**Contents**	**Quota**
Home Space	/home/user_name = ~	Small Files, executables, scripts	50 GB
Project Space	/project/umw_PI_name	Big files being actively processed	Varies
Farline Space	/farline/umw_PI_name	Big files for long term storage	Varies
S4S (Storage for Science)	/s4s/s4s_PI_name	Big files for archival storage	Varies

Reaching the Nodes

We do **NOT** use the head node (ghpcc06) to process big data.
We use the cluster nodes to process it.

How do we reach the nodes?

Reaching the Nodes

We do **NOT** use the head node (ghpcc06) to process big data.
We use the cluster nodes to process it.

How do we reach the nodes?

We submit our commands as jobs to a *job scheduler* and the job scheduler finds an available node for us having the sufficient resources (cores & memory.)

Job Scheduler

Job Scheduler is a software that manages the resources of a cluster system. It manages the program execution in the nodes. It puts the *jobs* in a (priority) queue and executes them on a node when the requested resources become available.

Job Scheduler

Job Scheduler is a software that manages the resources of a cluster system. It manages the program execution in the nodes. It puts the *jobs* in a (priority) queue and executes them on a node when the requested resources become available.

There are many Job Schedulers available. In MGHPCC,

IBM LSF (**L**oad **S**haring **F**acility)

is used.

Say we have 20 libraries of RNASeq data. We want to align using tophat.

```
tophat ...  library_1.fastq
```

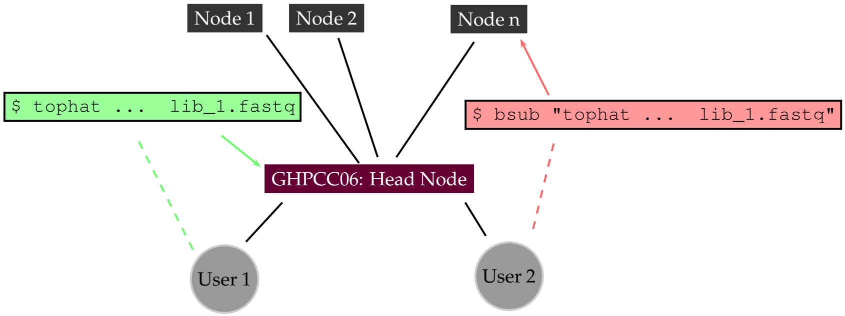
We submit this job to the job scheduler rather than running it on the head node.

Say we have 20 libraries of RNASeq data. We want to align using tophat.

```
tophat ...  library_1.fastq
```

We submit this job to the job scheduler rather than running it on the head node.

Submitting a job vs running on the head node



```
$ tophat ...  lib_1.fastq
```

```
$ bsub "tophat ...  lib_1.fastq"
```

Our First Job Submission

We use the command **`bsub`** to submit jobs to the cluster.
Let's submit a dummy job.

```
$ bsub "echo Hello LSF > ~/firstjob.txt"
```

Specifying Resources

After running

```
$ bsub "echo Hello LSF > ~/firstjob.txt"
```

we got the following warning message

```
Job does not list memory required, please specify memory
...
Job runtime not indicated, please specify job runtime
...
Job <12345> is submitted to default queue <long>
```

Why did the job scheduler warn us?

Specifying Resources

Besides other things, each job requires

1. Core(s) processing units
2. Memory

to execute.

The maximum amount of time needed to complete the job must be provided.

There are different queues for different purposes, so the queue should also be specified as well.

Specifying Resources

Cores	:	Number of processing units to be assigned for the job. Some programs can take advantage of multicores .Default value is 1.
Memory Limit	:	The submitted job is not allowed to use more than the specified memory. Default value is 1 GB
Time Limit	:	The submitted job must finish in the given time limit. Default value is 60 minutes.
Queue	:	There are several queues for different purposes. Default queue is the long queue.

Queues

Let's see the queues available in the cluster.

```
$ bqueues
```

We will be using the queues `interactive` , `short` and `long`.

interactive	:	used for bash access to the nodes
short	:	used for jobs that take less than 4 hours
long	:	(default queue) used for jobs that take more than 4 hours.

Specifying Resources

Hence we must provide

1. The number of cores
2. The amount of memory
3. Time limit
4. Queue

when submitting a job unless we want to use the system default values.

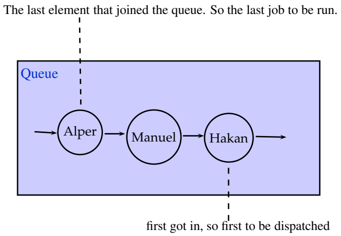
In a system like MGHPCC, where there are over 10K cores and tens of thousands of jobs and hundreds of users, specifying the right parameters can make a big difference!

Job Scheduling

Let’s try to understand how a job scheduler works on a hypothetical example. The IBM LSF system works differently but using similar principles.
Suppose, for a moment, that when we submit a job, the system puts our job in a queue.
A queue is a data type that implements **F**irst **I**n **F**irst **O**ut (FIFO) structure.

Job Scheduling

Say first, Hakan then, Manuel and, lastly, Alper submit a job. Then, the queue will look like


The last element that joined the queue. So the last job to be run.
Queue
Alper
Manuel
Hakan
first got in, so first to be dispatched

What if Hakan's job needs 10 cores and 5 TB of memory in total and 8 hours to run whereas Alper's job only needs one core 1 GB of memory and 20 minutes to run. Also, Alper didn't use the cluster a lot recently but Hakan has been using it very heavily for weeks.

This wouldn't be a nice distribution of resources. A better approach would be prioritizing jobs, and therefore using a priority queue.

Job Scheduling

In a priority queue, each element has a priority score. The first element to be removed from the queue is the one having the highest priority.

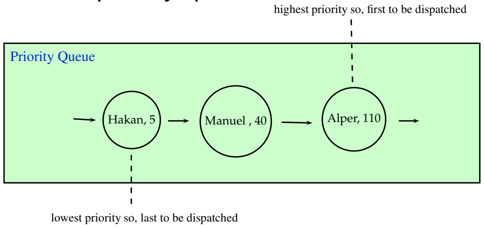
Hakan: I need 10 cores, 5TB of memory, 8 hours of time.
System: A lot of resources requested and heavy previous usage, so the priority score is 5.

Manuel: I need 2 cores, 8 GB of memory and one hour time.
System: A medium amount of resources requested, light previous usage, so the priority is 40.

Alper: I need one core, 1 GB of memory and 20 minutes.
System: Very little amount of resources requested, light previous usage, so the priority is 110.

Job Scheduling

So, in a priority queue, we would have


highest priority so, first to be dispatched
Priority Queue
Hakan, 5
Manuel , 40
Alper, 110
lowest priority so, last to be dispatched

This is a better and fairer sharing of resources. **Therefore** it is important to ask for the right amount of resources in your job submissions.

If you ask more than you need, then it will take longer to start your job.
If you ask less than you need, then your job is going to be killed.
It is a good idea to ask a little more than you actually need.

A more Sophisticated Job Submission

Let’s submit another job and specify the resources this time. To set

1. We explicitly state that we request a single core, `-n 1`
2. The memory limit to 1024 MB, we add `-R rusage[mem=1024]`
3. Time limit to 20 minutes, we add `-W 20`
4. Queue to short, we add `-q short`

```
$    bsub  -n 1 -R rusage[mem=1024] -W 20 -q short "sleep 300"
```

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

We need 4 cores as we'l run our process in 4 threads, so we need `-n 4`.
2 GB = 2048 MB, so we need the parameter `-R rusage[mem=2048]`.
We can **estimate** the running time to be 20 / 4 = 5 hours = 300 mins. So, let's ask for 330 mins to be on the safer side.

Exercise

Say you have a script that runs on on multiple threads. You previously run this script using a single core and it took 20 hours. Assume that you can run your script on the **head-node** by

```
$ ~/bin/myscript.pl -p number_of_threads
```

and assume that the number of threads is linearly proportional to the speed. Write a job submission command to run your script using 4 threads using 2 GB of memory. Use the parameter `-R span[hosts=1]` so that cores are guaranteed to be on the same host. You can specify the number of cores using the parameter `-n number_of_cores`

We need 4 cores as we'l run our process in 4 threads, so we need `-n 4`.
2 GB = 2048 MB, so we need the parameter `-R rusage[mem=2048]`.
We can **estimate** the running time to be 20 / 4 = 5 hours = 300 mins. So, let's ask for 330 mins to be on the safer side.

```
$ bsub -R span[hosts=1] -n 4 -R rusage[mem=2048] -W 330 -q long "~/bin/myscript.pl -p 4"
```

Monitoring Jobs

We will be running jobs that take tens of minutes or even hours.
How do we check the status of our active jobs?

```
$ bjobs
```

Monitoring Jobs

We will be running jobs that take tens of minutes or even hours. How do we check the status of our active jobs?

```
$ bjobs
```

Let’s create some dummy jobs and monitor them.
We run

```
$ bsub "sleep 300"
```

several times. Then

```
$ bjobs
```

```
JOBID     USER    STAT   QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
1499929   ho86w   RUN    long    ghpcc06     c09b01      sleep 300   Oct 6 01:23
1499930   ho86w   RUN    long    ghpcc06     c09b01      sleep 300   Oct 6 01:23
1499931   ho86w   RUN    long    ghpcc06     c09b01      sleep 300   Oct 6 01:23
```

Monitoring Jobs

We can give a name to a job to make job tracking easier.
We specify the name in the `-J` parameter.

```
$ bsub -J lib_1 "sleep 300"
```

```
$ bsub -J lib_2 "sleep 300"
```

```
$ bsub -J lib_3 "sleep 300"
```

Canceling Jobs

```
$ bjobs
```

```
JOBID     USER    STAT   QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME    SUBMIT_TIME
1499929   ho86w   RUN    long    ghpcc06     c09b01      sleep 300   Oct 6 01:23
```

We give the `JOBID` to `bkill` to cancel the job we want.

```
$ bkill 1499929
```

Creating Logs

It can be helpful to have the output and specifications of the jobs in separate files.
Two log files can be created: the standard error output and the standard output of the command run.
The standard output file is created using the `-o` parameter and the standard error output is be created using the `-e` parameter.

```
$ bsub -o output.txt -e error.txt "echo foo 1>&2; echo bar"
```

Using Bash on the Computing Nodes

Can I get a computing node (other than the head node) for myself temporarily?

Using Bash on the Computing Nodes

Can I get a computing node (other than the head node) for myself temporarily?

Yes. The interactive queue can be used for that.

```
$ bsub -q interactive -W 120 -Is bash
```

Determining Resources

How do we determine the queue, time limit, memory and number of cores?

Queue: Use the interactive queue for bash access. The time limit can be 8 hours maximum. If your job requires less than 4 hours, use the **short** queue, if it requires more than 4 hours, you need to submit it to the **long** queue.

Time Limit: This depends on the software and the size of data you are using. If you have a time estimate, request a bit more than that.

Determining Resources

Memory: Depends on the application. Some alignment jobs may require up to 32 GB whereas a simple gzip can be done with 1 GB of memory.

Number of Cores: Depends on the application. Use 1 if you are unsure. Some alignment software can take advantage of multicore systems. Check the documentation of the software you are using.

Advised Practice

- **Do not use the head node for *big jobs!***
  Do not run programs on the head node that will take longer than 5 minutes or that will require gigabytes of memory. Instead submit such commands as jobs. You can also use the interactive queue for command line access to the nodes. **This is mandatory!**
- Remember that MGHPCC is a shared resource among the five campuses of UMass!
- Keep in mind that you are probably sharing the same farline and project space quota with your lab members. Be considerate when using it!
- Keep your password secure.
- Backup your data.

Most Important MGHPCC Policy

Do not use the head node for *big jobs!*
Do not run programs on the head node that will take longer than 5 minutes or that will require gigabytes of memory. Instead submit such commands as jobs. You can also use the interactive queue for command line access to the nodes. **This is mandatory!**

On the head node (ghpcc06), using alignment software, samtools, bedtools and etc, R, Perl , Python Scripts and etc. for deep sequencing data is a **very bad** idea!
You are likely to get a warning and / or termination of your jobs if you do so.

For questions: hpcc-support@umassmed.edu

Advised Practice

- Keep your files organized
- Do not put genomic data in your home folder. Process data in the project space and use farline for long term storage
- Delete unnecessary intermediate files
- **Be considerate when submitting jobs and using disk space. The cluster is a shared resource.**
- Do not process big data in the head node. Always submit jobs instead.

For more detailed information, see
http://wiki.umassrc.org/