Introduction to HPC and SLURM

MONASH DATA FLUENCY

Instructors :
- Gin Tan
- Simon Michnowicz

Helpers:
- Nick Wong
- Richard Beare
- Trung Nguyen
- Damien Leong

Course Objectives

- Understand basic parallel computing concepts and workflows
- Understand the high-level architecture of a supercomputer
- Introductory Unix
- Use a basic workflow to submit a job and monitor it
- Understand a resource request and know what to expect

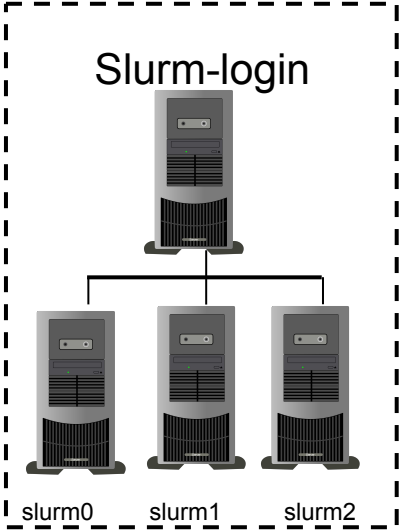
Course Materials

Schedule
https://tinyurl.com/hpc0318
And from there find links to go to..
ETHERPAD
https://biotraining.erc.monash.edu/etherpad/p/introtohpc
COURSE MATERIAL
https://gintan.github.io/intro-to-hpc/

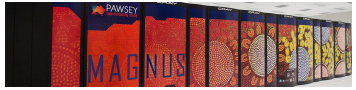
OVERHEAD SLIDES
Released later..

CHAPTER 1: WHY USE A CLUSTER?

HPC CLUSTERS

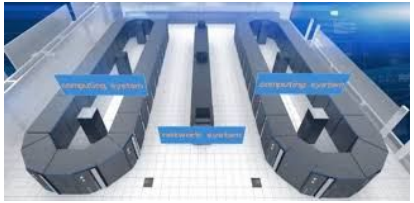


Our cluster for today!



Pawsey
(Perth)

NCI
(Canberra)



Sunway
(China)

And getting bigger all the time

e-Research Workflows

- **Collect**
  - Massive amounts of data are generated by modern instruments
- **Stage**
  - Data has to be stored
- **Process**
  - Single Large Job?
  - Large parallel jobs?
  - Multiple small jobs?
- **Visualise**
  - Examine values, generating images, interact
- **Archive**
  - Long term storage, sharing

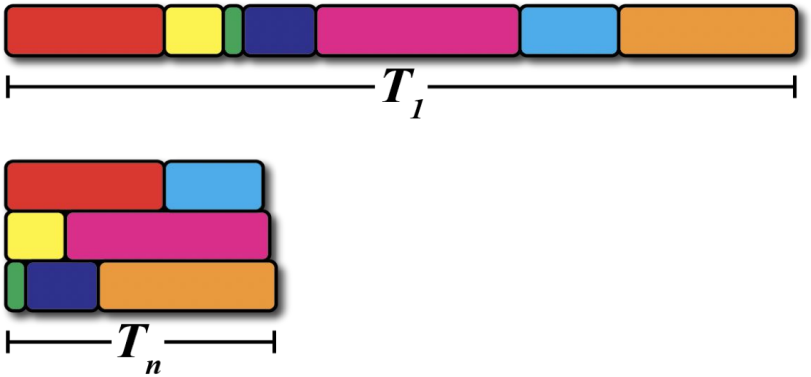
You need a cluster when your research needs..

- **Speed**
  - more CPU cores, often with higher performance specs, i.e. memory, disk speed, network speeds
- **Volume**
  - Terabyte and Petabyte disk storage (Spinning Disk, Solid State Disk, tape)
- **Efficiency**
  - Many HPC systems operate a pool of resources, running most of the time
- **Cost**
  - Often free to researchers due to competitive grants (i.e. NCMAS).
- **Convenience**
  - Professionally managed with advanced user support

Parallelism in Workflows

Exploiting parallelism in a workflow allows us to

- get results faster, and
- break up big problems into manageable sizes.
- A modern supercomputer is not a fast processor. It is many processors working together in parallel



Workflow Example – Cake Baking

It has a sequence of tasks that follow a recipe. Just like a computer program!

- Some tasks are independent
- Can be done in any order.
- Can be done in parallel.
- Some tasks have prerequisites.
    - Must be done sequentially.

Baking a Cake - Staging

- Staging” the ingredients improves access time.
- Ingredients are “prerequisites”.
- Each ingredient does not depend on others, so can be moved in parallel, or at different times.

Supermarket

Pantry

Bench

Baking a Cake - Process

Stage Ingredients ⟹ Measure Ingredients ⟹ Mix Ingredients ⟹ Preheat Oven

Serial

Parallel

Bake and cool cake

Mix icing

Levels of Parallelism

Coarse-grained parallelism (high level)

•Different people baking cakes in their own kitchens.
•Preheating oven while mixing ingredients.
•Greater autonomy, can scale to large problems and many helpers.

Fine-grained parallelism (low level)

- Spooning mixture into cupcake tray.
- Measuring ingredients.
- Higher coordination requirement. Difficult to get many people to help on a single cupcake tray

How many helpers?

What is *your* goal?
– high throughput, to do one job fast, or solve a grand-challenge problem?

High Throughput:

- For many cakes, get many people to bake independently in their own kitchens – minimal coordination.
- Turn it into a production line. Use specialists and teams in some parts.
- Doing one job fast:
- Experience as well as trial and error will find the optimal number of helpers.

When to use supercomputing

Workflows need supercomputing when the resources of a single laptop or workstation are not sufficient:

Workflows need supercomputing when the resources of a single laptop or workstation are not sufficient when:

- The program takes too long to process
- There is not sufficient memory for the program
- The dataset is too large to fit on the computer

If you are unsure whether moving to a supercomputer will help please email:

mcc-help@monash.edu

What to expect in a HPC System

Login Node(s)
Users login into a computer and use it to prepare data and processing jobs
- e.g. **slurm-login** for today's course

Data Transfer Node(s)
Users use this to transfer large data files (so as to not interfere with those on the Login Node)
- We do not have a DTN on our cluster

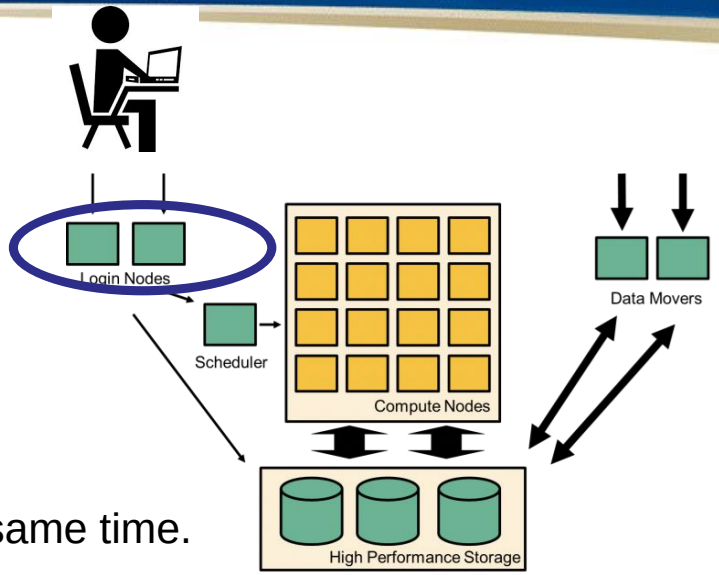
Data Storage
Most have a large parallel file system (e.g.CEPH, Lustre), often with tape archive.
- We have CEPH Storage Volume on **/mnt/nfs**

Batch Scheduler
To ensure a fair usage of compute resources, a scheduler will run your jobs
- SLURM

Software
HPC systems have pre-installed software to use. Don't reinvent the wheel!
- Environment modules

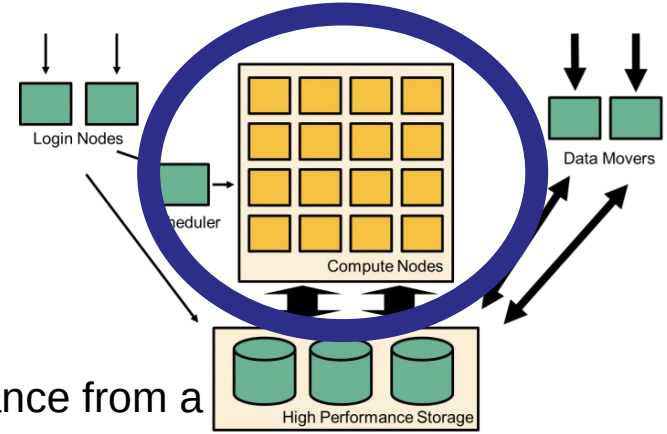
Data
Often large domain-specific data sets are stored on HPC systems
- e.g. bioinformatics databases

Login Nodes



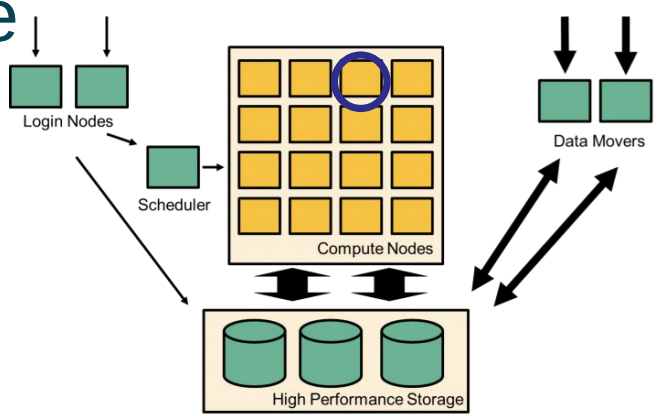
- Remote access to the supercomputer
- Where users should manage workflows
- Many people (~100) can share a login node at the same time.
- **Do not run your programs on the login nodes!**
- Use the login nodes to submit jobs to the queue to be executed on the compute nodes
- Login nodes *can* have different hardware to compute nodes.
  - Some build tests may fail if you try to compile on login nodes.

Compute Nodes



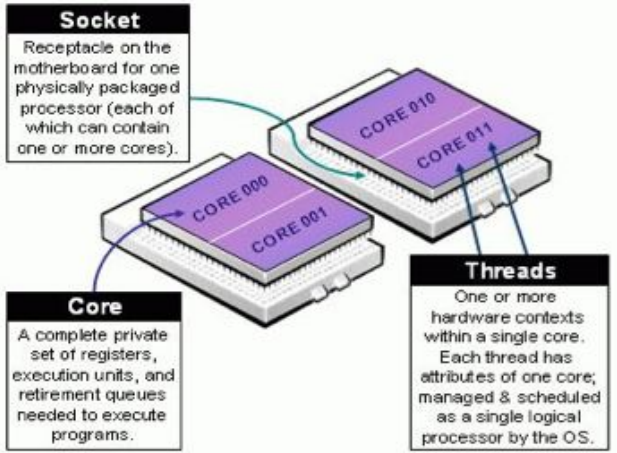
- Programs should run on the compute nodes
- Access is provided via the scheduler
- Compute nodes have a fast interconnect that allows them to communicate with each other
- Jobs can span multiple compute nodes
- Individual nodes are not that different in performance from a workstation
- Parallelism across compute nodes is how significant performance improvements are achieved.
- Nodes typically have access to the shared ‘global’ filesystem

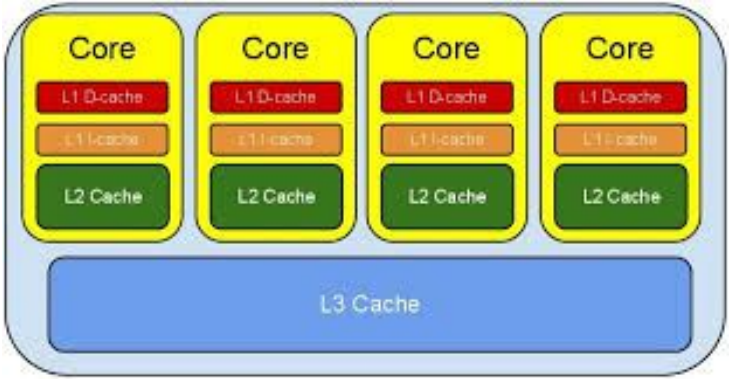
Inside a Compute Node



Each compute node has one or more CPUs:

- Each CPU has multiple cores
- Each CPU has memory attached to it
- Each node has an external network connection
- Some systems have accelerators (e.g. GPUs)





File Systems

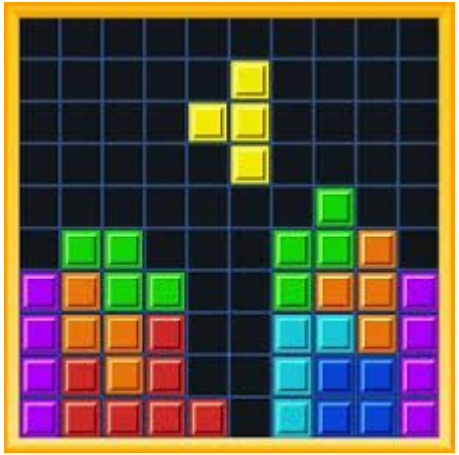
- HPC systems typically have many filesystems
- Each file systems typically has very different properties
  - Users will have quotas on their file systems
    - Amount of data they can have (~GB)
    - The number of files they can have (inodes)
  - Some file systems are:
    - Local only to the compute node (e.g. /tmp)
    - Global across all nodes (e.g. /mnt/home)
- Some file systems are backed up to tape, some not
- Some are high-performance parallel file systems, others not

Please consult the documentation when using a HPC system

Schedulers

All HPC systems have a queuing system that examines your job requirements and finds available hardware, where it is placed in a queue until it is scheduled to run on a compute node.

More will be covered later, but behind the scenes, schedulers are often considered to be like a game of Tetris...



CHAPTER 2: Connecting to the cluster

Secure Shell (ssh)

Logging into servers requires a **ssh** client on your local computer

Common terminal programs:

- **Windows**
  - use putty (download)
  - Or Cygwin if you are an advanced user
- **Linux**, use xterm (preinstalled)
- **OS X**, use Terminal (preinstalled) or xterm (download)

Secure Shell (ssh)

Logging into servers requires a **ssh** client on your local computer

OS	TOOL	Command
Mac	ssh is already in your terminal window	ssh username@43.240.99.84
Unix	ssh is already in your terminal window	ssh username@43.240.99.84
Windows	If you have a Cygwin terminal	ssh username@43.240.99.84

Example. **ssh simon@43.240.99.84**

```
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:CCVXbbNqjWEv9bvtcnzNT3O2n3ii9Y5rhg0GvqOXXiM.
ECDSA key fingerprint is MD5:4b:84:40:45:bd:05:27:cf:c3:33:99:58:96:13:d2:d0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Password:
Last login: Wed Aug  1 05:43:30 2018


--------------------------------------------------------------------------------
Nectar CentOS 7 (Core)
Image details and information is available at
https://support.ehelp.edu.au/support/solutions/articles/6000106269-image-catalog
--------------------------------------------------------------------------------


[simon@slurm-login ~]$
```

The very first time you log in, you will get a ssh-key exchange message, and you will be asked to accept the key. Hit. 'Yes'. This won't happen again.

*You are prompted for password. Your will **not** see the password when you type. This is a security measure*

Using PUTTY (Windows Only)

PUTTY is a Windows GUI often found on computers.

If you are on a windows box, and do not have putty, please download it now

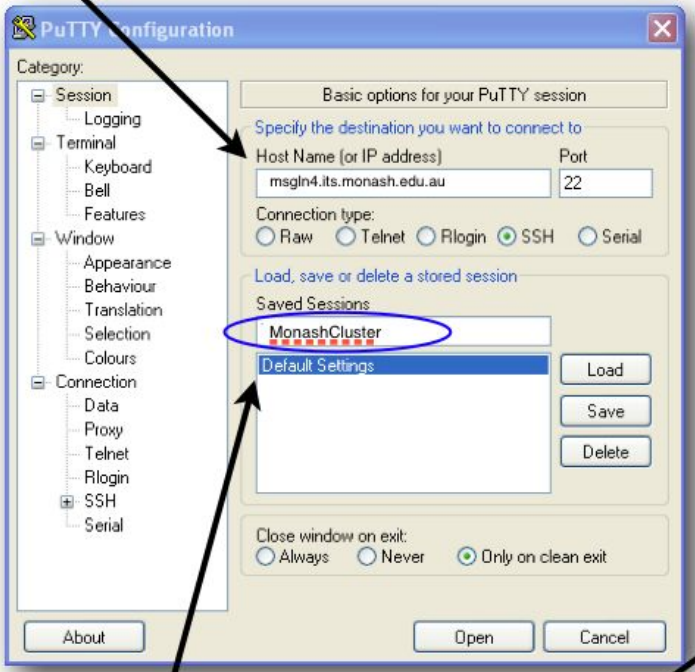
http://www.putty.org/

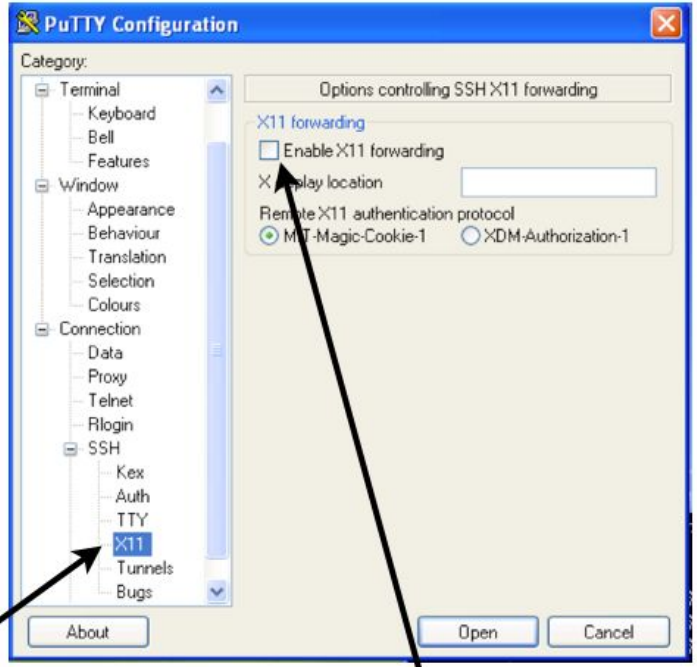
Advanced Users:
If you want a Unix experience on windows, then install Cywgin
https://www.cygwin.com/
(We will use putty today is it is a lot easier to install and configure)

Using PUTTY (Windows Only)

PUTTY is a Windows GUI often found on computers. For the course set 'Host Name' to **43.240.99.84**

step 1: enter hostname





step 2: type descriptive name

step 3: expand the SSH and click X11

step 4: check this box to enable X11 forwarding

Account Security

- `SSH` uses fingerprints to identify computers ... so you don't give your password to someone pretending to be the remote host

- Do not share your account with others, this violates the conditions of use (have the project leader add them to the project)

- Please do **not** provide your password in help desk tickets, nobody ever asks for your password via email as it is not secure

For all exercises use the sticky notes.

Need help

All Good

First Challenge

1. Open the course material in a web page
2. Login to our slurm login node **43.240.99.84**
    1. You will needs the username and password provided to you
3. Then let us know when you are finished…

Need help

All Good

Chapter 3:Scripts, variables

Unix Scripts

Shell commands can be placed in text files and executed. e.g. **file.sh**
-the files should have execute permission on them,
i.e. **chmod gou+x file.sh**
-the files are plain text files (and are not compiled)

The first line of the file is used to specify the shell being used.
#!/path/to/shell
e.g.
#!/usr/bin/bash
Or
#!/bin/bash

This will be machine dependent –where did they put the bash executable?
'#' normally indicates a comment line.
Then put whatever commands you want to execute.
Then you can run the command as if it were an executable.
./file.sh
OR run it with the correct shell command
sh file.sh

Unix Scripts

```
$ nano mycommands.sh
$ chmod gou+x
mycommands.sh
$ ./mycommands.sh
```

mycommands.sh

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

What do you expect to see?

Editors - revision.

UNIX has several in-built editors to create text files.

- **vi** - screen oriented text editor. As old as Unix..
- **emacs** – also created decades ago. Part of GNU project.
- **nano** – simple micro editor
- Plus many more….

Nano is the easiest editor to use in a beginner's course…

```
nano myfile.txt
vi myfile.txt
emacs myfile.txt
```

Either command creates a new file 'newfile.txt' if it does not exist, or edits an existing file.

NANO

```
  GNU nano 2.0.6                    New Buffer

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

Shell Variables

X=1

VARIABLES

Shells are programs like any other, and like all programs, they have variables that control their behaviour. You can view the shell variables you have by typing

env

or

printenv

```
BASH=/bin/bash
BASH_ARGC=()
BASH_ARGV=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gnu")
BASH_VERSION='3.2.25(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=172
CONDOR_CONFIG=/opt/vdt/condor/etc/condor_config
CVS_RSH=ssh
…
…
etc
```

COLUMNS=172

Convention makes variable names upper case

Variable values are always strings, and have to be converted to other types if necessary.

MANPATH=:/usr/share/man:/opt/n1ge62/man

When variables contains lists of values, the *convention* uses the **:** character as the delimiter

The PATH variable controls where the shell looks for executables.

PATH=/usr/lib64/qt-3.3/bin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/opt/n1ge62/bin/lx24-amd64:/opt/dell/srvadmin/bin:/nfs/home/hpcmerc/vlad/bin:/usr/bin:/usr/ucb:/etc:/opt/vdt/condor/bin:/opt/vdt/jdk1.5/bin:.

./myprog.exe ← This runs myprog.exe in the current directory

/bin/myprog.exe ← This runs myprog.exe found in /bin

myprog.exe ← Which program is executed here?

The shell separates the paths found in the PATH variable and searches each one in order for the program to execute.

/usr/lib64/qt-3.3/bin
/usr/kerberos/bin
/usr/local/bin:/bin
/usr/bin
/opt/n1ge62/bin/lx24-amd64
/opt/dell/srvadmin/bin
/nfs/home/hpcmerc/vlad/bin
/usr/bin
/usr/ucb
/etc
/opt/vdt/condor/bin
/opt/vdt/jdk1.5/bin
.

Order of search

Current directory '.' last to be searched, but only because of its position in PATH. Some systems do not put '.' In the PATH at all. Can you think why?

Printing Variables

The ‘echo’ command can be used to print variables. A $ is needed to discriminate between text and variables.

```
echo HOME
HOME
echo $HOME
/nfs/home/hpcmerc/vlad
```

You can create a variable by assigning to it

```
PIN=1234
echo $PIN
1234
PIN=4321
echo $PIN
4321
```

To assign a variable in a shell so that it is copied into child processes that it creates, you use the **export** command.

```
PIN=1234
export PIN
bash
echo $PIN
1234
```

(10 min) Exercise.

- Write a script file that executes the following commands
- **date**
  - Date is a unix shell command that prints the time and date
- **echo 'This script is running on:'**
  - to print the text
- **hostname**
  - Hostname is a unix shell command that prints the name of the computer you are on
- **sleep** 30
  - **sleep N** sleeps for N seconds
- **echo 'Completed by' $USER**
  - to print the text follow by an environment variable for username

Need help

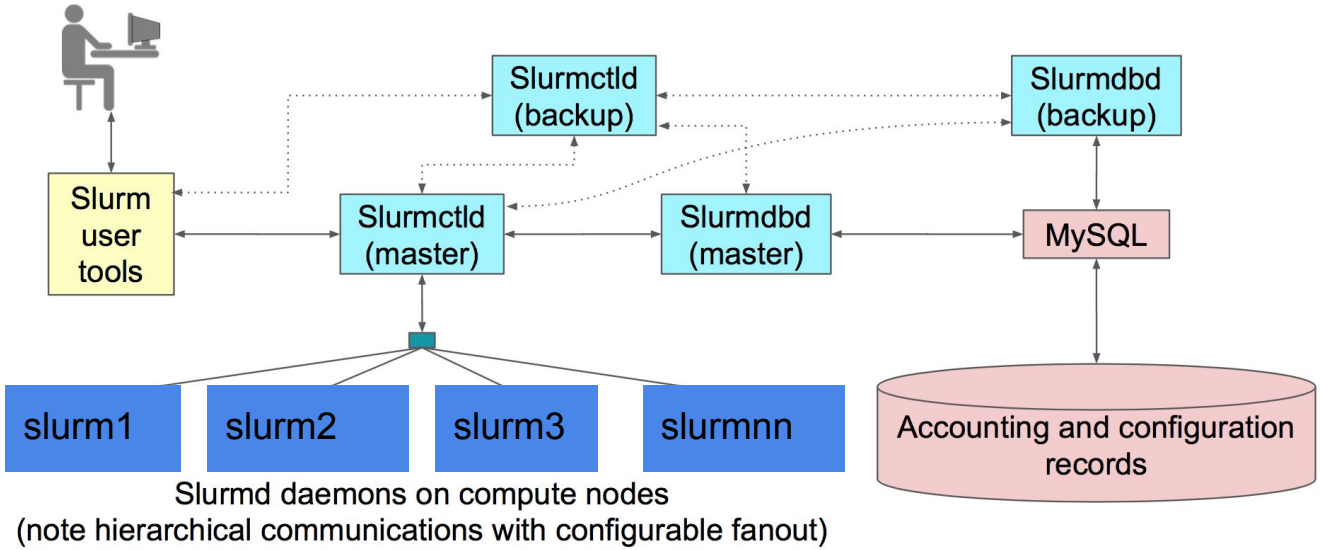
All Good

Execute your script from the shell command line and examine the output

At the end of this exercise, you should be able to put several shell commands into a file and run them

Chapter 5: Working on cluster

SLURM as a job scheduler and resource manager.



SLURM user commands:

sbatch
- Submit a batch script to Slurm

salloc
- Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished

srun
- run parallel jobs

Exercise (5 min):

Now that you've learned how to create a script, we will submit the script to the job scheduler.

Content of firstjob.sh:

```
#!/bin/bash
date
echo 'This script is running on:'
hostname
sleep 120
echo 'Completed by' $USER
```

To submit:

```
sbatch firstjob.sh
```

Do you get a job ID?

The job ID is for record purposes and it's very useful when you are troubleshooting your job.

Useful commands to check your job on the scheduler:

scontrol show job {jobID}
- When you obtain your job ID, you can check the information about your job

squeue
- view information about jobs located in the Slurm scheduling queue

squeue -u {username}
- view information about jobs located in the queue for one or more users

scancel {jobID}
- Used to signal jobs or job steps that are under the control of Slurm

squeue -u {username}

```
[user1@slurm-login ~]$ squeue -u user1
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                47       all firstjob    user1  R       0:03      1 slurm1
```

squeue

```
[user1@slurm-login ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                48       all firstjob    user1  R       0:01      1 slurm1
```

```
[user1@slurm-login ~]$ scontrol show job 47
JobId=47 JobName=firstjob.sh
   UserId=user1(1001) GroupId=user1(1001) MCS_label=N/A
   Priority=20182 Nice=0 Account=slurmclass QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:16 TimeLimit=02:00:00 TimeMin=N/A
   SubmitTime=2018-08-02T22:16:51 EligibleTime=2018-08-02T22:16:51
   StartTime=2018-08-02T22:16:51 EndTime=2018-08-03T00:16:51 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2018-08-02T22:16:51
   Partition=all AllocNode:Sid=slurm-login:7279
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=slurm1
   BatchHost=slurm1
   NumNodes=1 NumCPUs=1 NumTasks=1 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
   TRES=cpu=1,mem=4G,node=1,billing=1
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=1 MinMemoryCPU=4G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   Gres=(null) Reservation=(null)
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=/mnt/nfs/home/user1/firstjob.sh
   WorkDir=/mnt/nfs/home/user1
   StdErr=/mnt/nfs/home/user1/slurm-47.out
   StdIn=/dev/null
   StdOut=/mnt/nfs/home/user1/slurm-47.out
   Power=
```

Exercise (5 min)

OK now resubmit the job, and use the SLURM commands to check the status of your jobs?

Is your job running?

Which node is it running on?

Can you see how many CPU task did you ask for?

How much memory is allocated?
(You need this for the next exercise)

Where is the output going to be located?

Are you able to cancel your job?

Commands:

scontrol show job {job_ID}

squeue

squeue -u {username}

scancel

More useful commands to check the information about a compute node:

sinfo
- To see all the partition in the queue

sinfo -p {partition name}
- if you just want to look at a specific partition, you can parse in -p follow by the name of the partition

sinfo -Nl
- to see all the nodes in the job scheduler

scontrol show node {node name}
- to see a specific node in the queue

```
[user1@slurm-login ~]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*         up 7-00:00:00      3   idle slurm[0-2]
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ sinfo -p all
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
all*         up 7-00:00:00      3   idle slurm[0-2]
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ |
```

```
[user1@slurm-login ~]$ sinfo -Nl
Thu Aug  2 22:28:46 2018
NODELIST   NODES PARTITION       STATE CPUS    S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
slurm0         1      all*        idle    8    8:1:1  31000        0      1   (null) none
slurm1         1      all*        idle    4    4:1:1  11000        0      1   (null) none
slurm2         1      all*        idle    4    4:1:1  11000        0      1   (null) none
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$
[user1@slurm-login ~]$ scontrol show node slurm0
NodeName=slurm0 Arch=x86_64 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=8 CPULoad=0.02
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=slurm0 NodeHostName=slurm0 Version=17.11
   OS=Linux 3.10.0-693.17.1.el7.x86_64 #1 SMP Thu Jan 25 20:13:58 UTC 2018
   RealMemory=31000 AllocMem=0 FreeMem=29215 Sockets=8 Boards=1
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=all
   BootTime=2018-07-26T11:51:18 SlurmdStartTime=2018-08-01T16:30:13
   CfgTRES=cpu=8,mem=31000M,billing=8
   AllocTRES=
   CapWatts=n/a
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Exercise (5 min)

Now use the commands to check the status of a compute node

How many compute node do we have in this cluster?

How many cpu is available for each of the compute node?

How many partition do we have in this cluster?

What is the time limit for each of this partition?

Commands:

sinfo

sinfo -p all

sinfo -Nl

scontrol show node slurm1

What is a partition?

A partition is a pool of resources, it usually consists of one or more nodes that have the same constraints.

For example:

A partition can have a maximum time limit of 7 days and the default memory per cpu is 4096MB

```
PartitionName=all Default=yes Nodes=slurm[0-2] State=Up MaxTime=7-0
DefaultTime=02:00:00 DefMemPerCPU=4096 AllowQOS=ALL
```

Customise your job script?

You can tell the job scheduler how much of the resources do you need to run your job?

- Time limit
- Memory
- Account
- Number of tasks
- Number of nodes

--ntasks or -n	Number of tasks and by default the number of CPUs
--mem	Total memory size
--time or -t	Wall time limit
--input or -i	File used for standard input (default is /dev/null)
--output or -o	File used for standard output
--error or -e	standard error (default is to combine with stdout)

An example of a complex job script:

```
#!/bin/bash
#SBATCH --job-name=desktop
# --exclusive allows the job to consume all resources on the node reguardless of how many cpus/gpus/memory/etc
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:K80:2
#SBATCH --partition=m3c
#SBATCH --nodelist=m3c001
# SBATCH --reservation=Maint-09Jan18

env | grep SLURM
echo " Starting MASSIVE desktop..."
echo " Setting up the system environment..."
# This is required for tcsh desktops to work
# source /etc/profile.d/modules.sh
module purge
```

Exercise: (15 min)

Write a job script to use more than one CPU and assign more memory than what's default?
Also, add in the time limit as well, set it to 5 mins.
Submit the job and examine the job information.

[Hints] Combine the following with your previous script:

```
#!/bin/bash

#SBATCH -n 2
#SBATCH -o mysecondjob
#SBATCH --mem=5120
#SBATCH --time=:0:00
```

Running a parallel job

srun - Run a parallel job on cluster managed by Slurm. If necessary, srun will first create a resource allocation in which to run the parallel job.

```
srun --ntasks=2 --label hostname
```

In this example, we request SLURM to allocate two CPU and precede each standard output or error message with its rank number

The output:

```
0: slurm1
1: slurm1
```

Running a parallel job across two nodes:

```
srun --ntasks=2 -N 2 --label hostname
```

and the output is:

```
1: slurm1
0: slurm0
```

Exercise (15 min)

To demonstrate the performance differences running a job with one core vs two cores.

```
$ mkdir mpi
$ cd mpi
$ cp /usr/local/exercises/mpi/* .
```

- examine run.sh and see what it does
- modify slurm.slm to request one CPU
- submit job with sbatch
- examine the output

[Hints]

```
#!/bin/sh
#SBATCH --job-name=mpitest
#SBATCH --time=5:00:00
#SBATCH --ntasks=2
#SBATCH --cpus-per-task=1

#SBATCH --partition=all

./run.sh
```