MRC | Cognition and Brain Sciences Unit

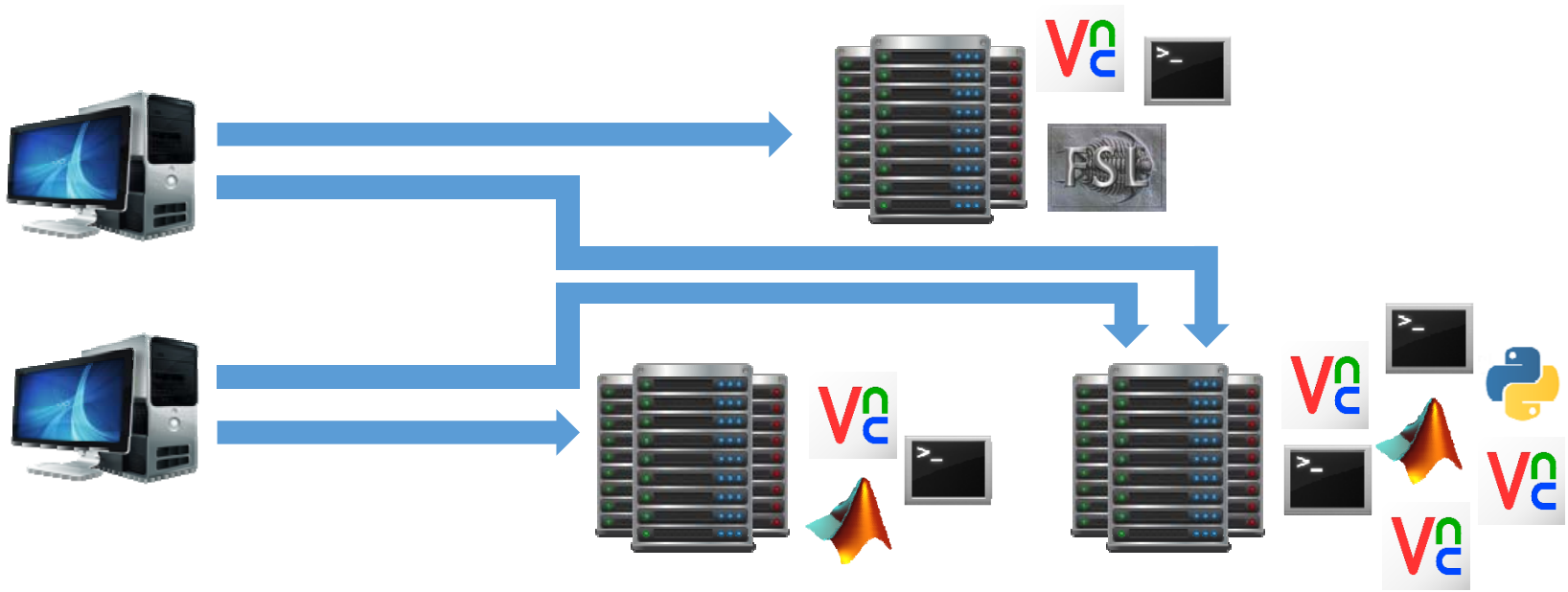
Using the CBU scheduling system

Russell Thompson

Overview

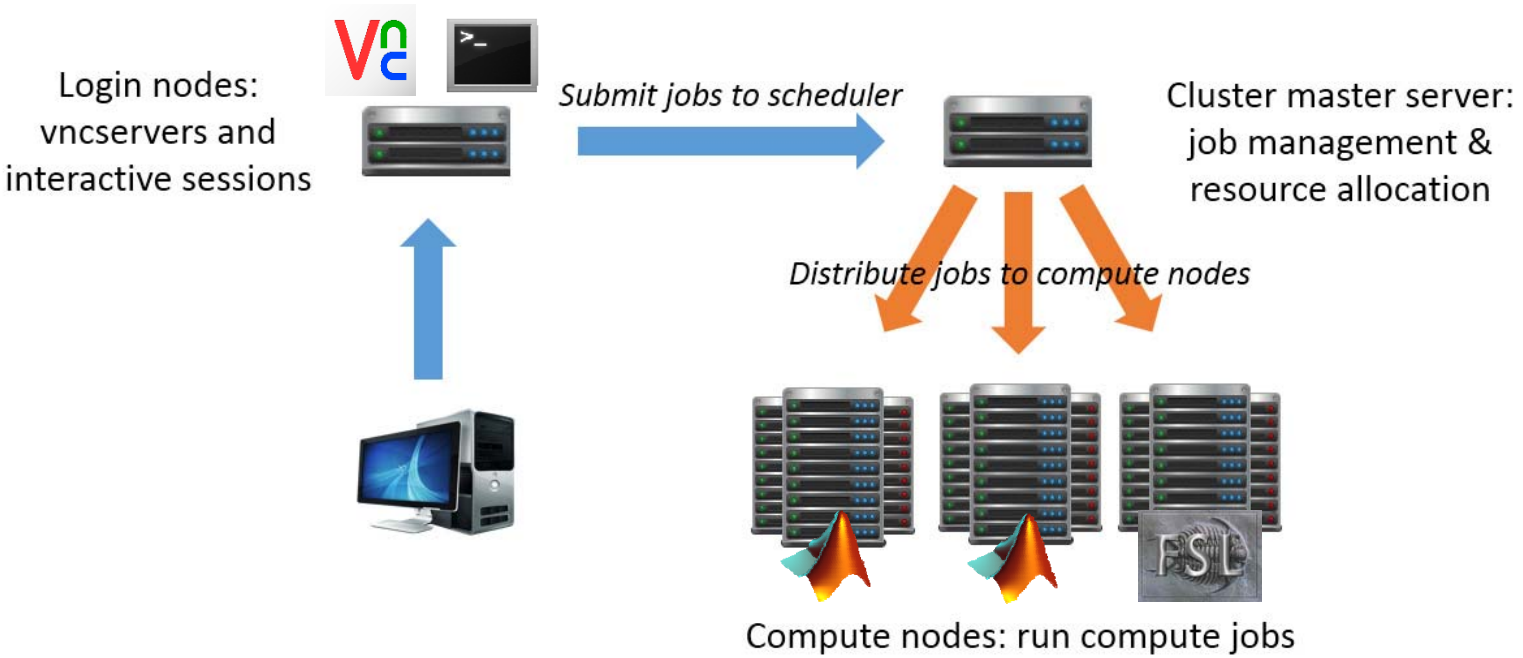
- Architecture of new cluster and scheduling system
- Submitting jobs
- Monitoring and managing jobs
- Matlab / SPM jobs
- Optimal use

Non-scheduler based architecture



- Machines operate independently
- No distinction between login and compute nodes – log in directly and run both interactive sessions and large compute jobs on the same machines
- No management of which jobs run on which machine
- Can cause problems – e.g. machines can run out of memory

Scheduler based architecture



- Distinction between login and compute nodes
- Login and run interactive sessions on a login node
- Run large compute jobs on compute nodes
- Submit compute jobs to a scheduling system (Torque/Maui) that manages allocation of compute resources

Login nodes

Name	CPU (MHz)	N Cores	RAM (GB)	Open GL graphics	CPU Architecture
Login01-6, Login09-10	3	8	16	No	Westmere
Login07-8	3	8	32	No	Harpertown
Login11,12,14	2.67	12	48	No	Westmere
Login13	2.67	16	96	No	Sandy Bridge
Login15-login22	2.67	16	128	No	Ivy Bridge
Login-gpu01	2.67	12	48	Yes	Westmere
Login-gpu02-3	2.67	16	192	Yes	Ivy Bridge

- 304 cores @ ~6.2 GB/core
- All run Scientific Linux 6.4 (64 bit)

Compute nodes

Name	CPU (MHz)	N Cores	RAM (GB)	Open GL graphics	CPU Architecture
Node-c01-06	3	8	16	No	Harpertown
Node-cc01-04	2.67	16	96	No	Sandy bridge
Node-cc05-07	2.67	16	64	No	Sandy bridge
Node-d01	2.67	12	144	No	Westmere
Node-d02-18	2.67	12	48	No	Westmere
Node-e01-20	2.67	16	96	No	Sandy bridge
Node-f01-08	2.67	16	192	No	Ivy Bridge
Node-gpu01 – 02	2.67	16	64	Yes	Sandy bridge
Node-gpu03 – 04	2	12	64	Yes	Sandy bridge

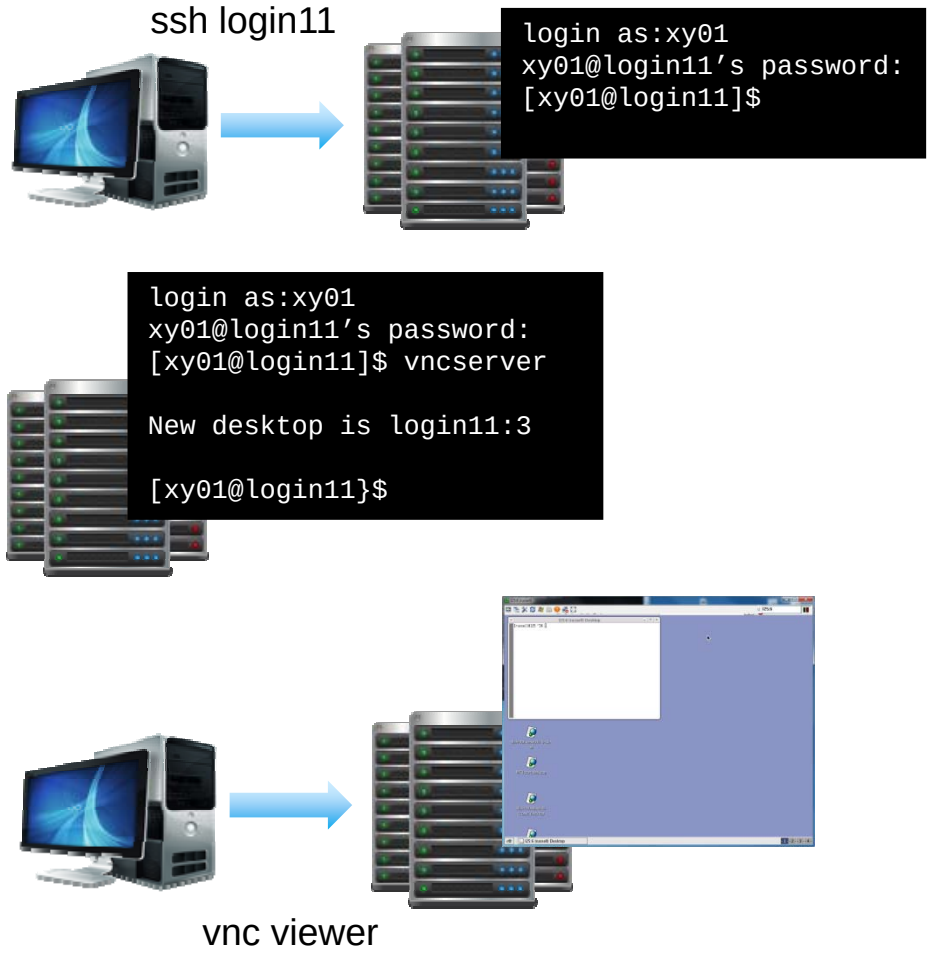
- 880 cores @ ~6 GB/core
- All run Scientific Linux 6.4 (64 bit)

Why use a scheduling system?

- Efficient management of resources
  - Scheduler determines when and where compute jobs will run, rather than allowing jobs from many users to compete for the same resources
  - The scheduler knows what resources are available on each compute machine, what resources are required by each compute job, and will try to make sure that resources are fully utilized, but not overloaded
  - Each node will only run jobs from one user at a time, so any problems will only affect that person.

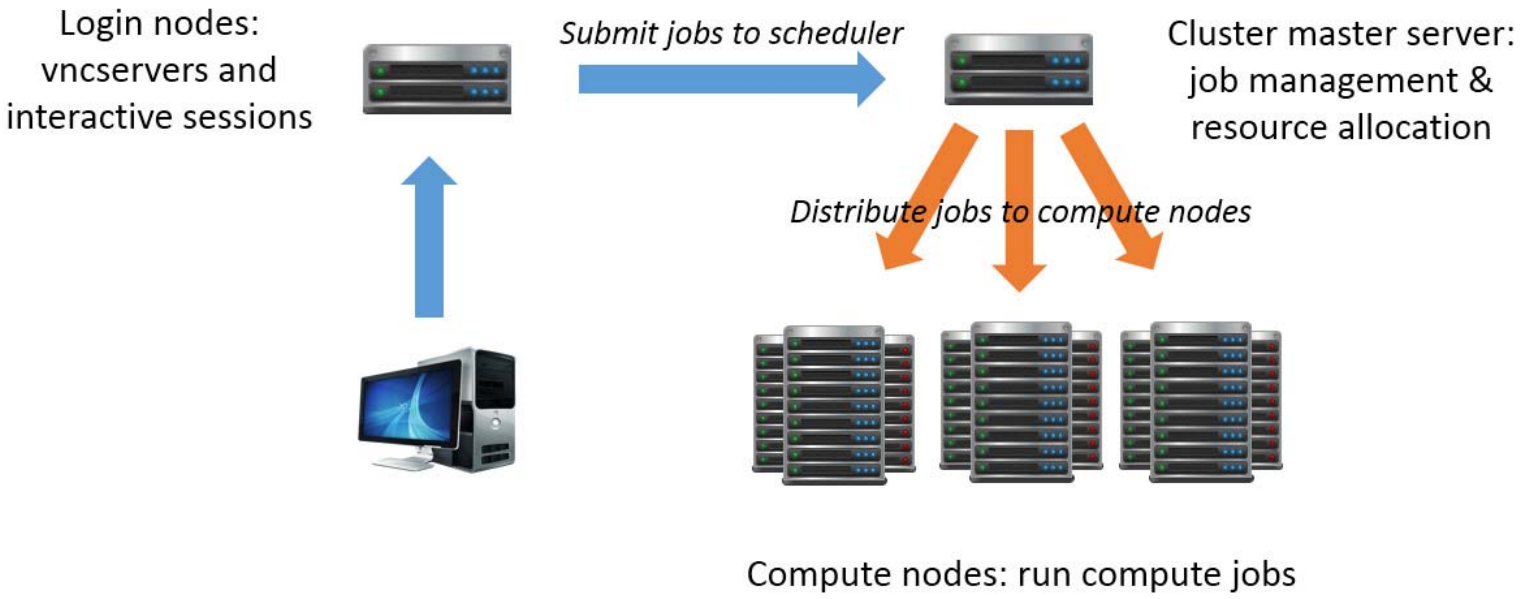
- Management of parallel processing
  - Multiple independent jobs (“embarrassingly” parallel jobs – e.g. pre-processing data from different subjects)
  - Truly parallel jobs, e.g. using protocols such as MPI

Accessing compute machines



- Access to **login** nodes using ssh / PuTTY.
- Can pick a specific machine (login01, login17, login-gpu02, etc), or use the alias "login" in which case a machine will be chosen for you.
- Launch a vncserver to start a graphical session.
- **Compute** nodes are only accessible via ssh when you have a job running on them

General Workflow



- Log in to a login node and start a vnc server
- Create a batch script to run your analyses
- Test the batch script and determine what resources it needs (esp. memory and CPU time)
- Submit the script to the scheduling system
- As part of the submission process, request specific cluster resources

Scheduler resources

- Hardware resources
  - Number of cores, specific processor architecture, amount of RAM, Graphical Processing Units (GPUs)

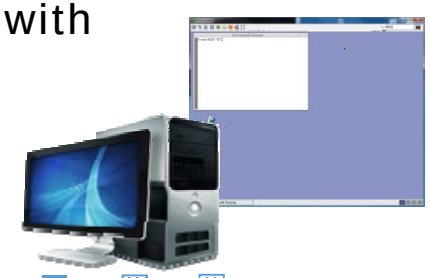
- Software resources
  - Specific packages may only be available on certain machines
  - For example, on our cluster, Maxfilter is only licensed on some machines

- Also specify the length of time for which resources are required. Referred to as the “walltime”. This defines the maximum amount of time for which the resources are required - after the walltime elapses, any remaining processes are killed.
- If a job finishes before the walltime has elapsed, resources will be returned to general availability.

- Requesting the appropriate resources is important – it affects which machines are allocated to particular jobs, how many how many jobs are allocated to a machine, etc.

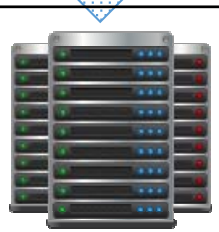
Scheduler job queues

- A job will be executed as soon as the requested resources are available, otherwise the job will be held in a queue.
- Choose a queue for your job based on its resource requirements and submit to that queue.
- Different queues are associated with specific sets of resources – e.g. specific machines, maximum available memory, maximum available CPU time, etc



Example queue 1
20 machines
Max RAM 4GB
Max CPU time 12 hours

Example queue 2
10 machines
Max RAM 96GB
Max CPU time 1 week



Example queue 3
10 machines with GPU's
Max RAM 96GB
Max CPU time 1 week

CBU queues

- Currently a general purpose compute queue ("compute")
- Queue configuration will be reviewed depending on usage patterns – e.g. do we need a separate gpu queue, do we need a short jobs queue, etc

	compute
Machines	All compute nodes
Def walltime	24 hours
Max walltime	1 week
Def memory	0 bytes[1]
Max memory	192GB[2]
N jobs (active/inactive)	128/128[3]

[1] In practice, c.2GB (e.g. if 8 jobs allocated to one of the node-c machines)

[2] Limited by the maximum memory available on a single machine

[3] Can submit more jobs, but additional jobs remain at the bottom of the queue

Submitting jobs to the scheduler

- The qsub command can be used to submit jobs:

```
qsub <arguments> <command to run>
```

- The qsub command takes numerous arguments, including:
  - -n job name
  - -q which queue to use
  - -o path to file where standard output should be redirected
  - -e path to file where standard error output should be redirect
  - -l request specific resources, including memory, CPU time, number of CPU cores
  - -v variable to pass to job batch script
  - -V export environment from interactive session to workers

- E.g.:

```
qsub –q compute –v sub=“CBU130001” my_analysis.pbs
```

submits the script my_analysis.pbs to the compute queue, passing the variable sub with value CBU130001

Submitting jobs to the scheduler

- Resources include:
  - mem=<value kb/mb/gb> — maximum memory required
  - walltime=<value hh:mm:ss> — maximum job execution time
  - walltime=<value s>
  - nodes — number / type of compute nodes
  - nodes=<node name> — use specific node / group of nodes
  - nodes=<numeric value> — reserve <numeric value> nodes
    - :ppn — number of cores per node
    - :gpus — number of gpus per node

- E.g.:

```
qsub -q compute -l mem=16gb,walltime=216000,nodes=2:ppn=14
my_analysis.pbs
```

submits the script my_analysis.pbs to the compute queue, requesting 16GB memory, upto 6 hours of CPU time, and 14 cores on 2 machines

Submitting jobs to the scheduler

- qsub arguments can also be submitted using #PBS directives within the script itself:

```
#!/bin/bash

#PBS –q compute
#PBS –l walltime=6:00:00,mem=16gb,nodes=2:ppn=12

<command 1>
<command 2>
...
```

- This would allow the script to be submitted using simply:

```
qsub my_analysis.pbs
```

Monitoring jobs

- Monitoring jobs:

```
qstat
```

- On it's own, this gives a list of jobs currently held on the queue:

```
Job ID          Name        User    Time Use  S     Queue
-----------     ---------   -----   --------  -     -----
1520.master01   Job1Task1   ab01    00:58:28  C     compute
1521.master01   Job1Task2   ab01    00:45:03  R     compute
```

Monitoring jobs

- For more information, including the amount of memory requested by each job, and the node on which it's running:

```
qstat -n
```

```
Job ID          U'name  Queue    Jobname   Sess ID NDS TSK Req Mem Req Time S Elap Time
-----------     ------  -----    -------   ------- --- --- ------- -------- - ---------
1520.master01   ab01    compute  Job1Task   154451  1   --    47gb   12:00   C   00:58
node-e01/0

1521.master01   ab01    compute  Job1Task2   50846  1   --    47gb   12:00   R   00:45
node-e02/0
```

Monitoring jobs

- Show jobs for a particular user:

```
qstat –nu <username>
```

- Show your jobs:

```
qstat –nu <your username>
qstat –nu `whoami`
```

- Very detailed information about a specific job:

```
qstat –f <job id>
```

Standard output and error messages

- Interactive sessions are attached to a console or terminal emulator that provides a way to send input and receive output from the kernel.
- Output is divided into various standard output streams, e.g. standard out (stdout) and standard error (stderr).

- Jobs sent to the scheduler will run in non-interactive (batch) mode
- Not attached to any terminal, output is directed to files. By default files will be saved in home directory
  - Stdout → ~/<job id>.OU *or* ~/<script name>.o<job id>
  - Stderr → ~/<job id>.ER *or* ~/<script name>.e<job id>
  - e.g. ~/my_script.pbs.o643209
- Can specify output location using the –o and –e arguments to qsub:

```
outdir=/home/russell/analysis1
qsub –o ${outdir}/job1.OU –e  ${outdir}/job1.ER my_analysis.pbs
```

- Useful for monitoring, debugging, and troubleshooting scheduler jobs

Demo 1

- http://intranet.mrc-cbu.cam.ac.uk/computing/cluster-demo/#1

Submitting multiple jobs

- Most applications are single threaded, and each job will be allocated a single CPU core (can request more cores per job for multi-threaded applications though)
- Usually be most efficient to split independent compute tasks (e.g. pre-processing fmri data sets from separate participants) into separate jobs and submit them simultaneously.
- This can be done on the command line, or in an “outer” / multi-subject script, e.g:

```
#!/bin/bash

for s in CBU130001 CBU130002 CBU130003;
do

     qsub -v sub=${s} my_analysis.pbs

done
```

Submitting multiple jobs

- qsub also includes a “job array” feature
- Submit with –t argument followed by a range of numeric values
- qsub launches one worker for each numerical value, passing that value to the worker via the environment variable PBS_ARRAYID

```
qsub –t 0-3,9 my_analysis.pbs
```

- Launch 5 workers, 1 with PBS_ARRAYID=0, 1 with PBS_ARRAYID=1, etc

```
#!/bin/bash
#PBS -q compute
#PBS -l walltime=12:00:00,mem=16gb
all_subs[0]=CBU130001
...
all_subs[9]=CBU130009
current_sub=${all_subs[$PBS_ARRAYID]}
<command 1>
<command 2>
```

Job management

- To delete a job from the queue:

```
qdel <job id>
```

- To remove all your jobs from the queue:

```
qselect -u `whoami` | xargs qdel
```

- To place and release a hold on a job:

```
qhold <job id>
qrls <job id>
```

- To alter the properties of a job (e.g. to change some of the resources requested in a qsub command):

```
qrls <job id>
```

Demo 2

- http://intranet.mrc-cbu.cam.ac.uk/computing/cluster-demo/#2

Submitting Matlab / SPM jobs

- Use qsub directly (not recommended!)

```
qsub matlab-job.pbs

matlab-job.pbs:
#!/bin/bash
#PBS -q compute
#PBS -l walltime=12:00:00,mem=16gb
...
matlab –r <matlab command / script name> <matlab arguments>
```

- Use the matlab parallel functions together with CBU-specific wrapper functions
- Use parfor loops / spmd with CBU cluster profile
- Use aa version 4

Submitting Matlab / SPM jobs

- Matlab Distributed Computing Server (DCS) and Parallel Computing Toolboxes provide functions for running matlab jobs in parallel over multiple compute nodes / CPU cores
- DCS supports 3rd party schedulers such as Torque
- In general, the procedure for submitting to a scheduler using DCS is:
  - Create a script to run your analysis
  - Create a scheduler object using the DCS functions
  - Configure the scheduler object (define properties such as queue name, resources required,etc).
  - Configure the scheduler with a list of jobs to run
  - Call a submit method to submit the jobs
- When using a Torque / PBS scheduler object, Matlab translates the properties of the scheduler into a series of qsub commands.

Submitting Matlab / SPM jobs

- 2 cbu-specific functions have been created to simplify the process submitting to the CBU cluster via DCS:
  - cbu_scheduler – creates and configures a scheduler object
  - cbu_qsub – submit jobs to the queue

```
subjects={'CBU130001','CBU130002};

clear J;
for s=1:size(subjects,2)
   J(s).task=str2func(my_analysis_script);
   J(s).n_return_values=0;
   J(s).input_args=subjects(s);
   J(s).depends_on=0;
end

clear S;
S=cbu_scheduler();
cbu_qsub(J,S,[]);
```

Loop through all subjects. For each subject, add an entry to a structure array containing details of the analysis to run

Create a scheduler object. Without any other arguments, cbu_scheduler will return a default configuration

Submit the jobs

Configuring the scheduler

- Use a pre-defined configuration:
  - basic-compute', 'compute', 'basic‘
    compute queue, 12 workers, 4Gb RAM per worker, 1 hour walltime
  - 'large-compute'
    compute queue, 24 workers, 12Gb RAM per worker, 2 hours walltime
  - 'basic-gpu', 'gpu'
    gpu queue, 12 workers, 4Gb RAM, 1 hour walltime
  - 'custom'
    manually specify scheduler parameters

- e.g.

  S=cbu_scheduler(‘basic’)
  S=cbu_scheduler(‘large-compute’)

Configuring the scheduler

- Use a custom configuration:

  S=cbu_scheduler('custom',{<*parameters*>}),

  where {<*parameters*>} are:

  1. queue name (compute, gpu) [default = 'compute']
  2. n workers [default = 12]
  3. memory (Gb) [Default = 4Gb]
  4. walltime (seconds) [Default = 3600 seconds]
  5. job data directory[1]
  6. Matlab worker path[2]

- e.g.

  S=cbu_scheduler('custom',{'compute',16,64,21600});

  S=cbu_scheduler('custom',{[],[],64,21600});

[1]location matlab uses to store DCS job information, output files, etc. Default = /imaging/<user name>/.cbu-cluster/matlab-jobs

[2]which version of matlab should be used for workers. Default =matlabroot

Configuring the scheduler

- Use a custom configuration with a qsub submission string:

  S=cbu_scheduler(‘custom’,{<*parameters*>}, <*qsub string*>),

- e.g.

  S=cbu_scheduler('custom',{[],24,[],[],[],'/hpc-software/matlab/r2009a'},'-q gpu -l mem=96gb');

- Customise the scheduler object after creation:

  S=cbu_scheduler();

  S.NumWorkers=24;

  S.SubmitArguments=‘-q gpu –l mem=60gb,walltime=21600’;

Parfor and spmd

- Need to open a matlabpool on the cluster, rather than on the local host
- In matlab 2012a onwards:

```
P=parallel.importProfile('/hpc-software/matlab/cbu/CBU_Cluster.settings');
matlabpool(P);
```

- To modify the properties of the CBU_Cluster profile:

```
P=parallel.importProfile('/hpc-software/matlab/cbu/CBU_Cluster.settings');
P=parcluster(P);
P.SubmitArguments='-l walltime=360:00';
P.ResourceTemplate='-l nodes=2:ppn=14';
P.NumWorkers=28;
matlabpool(P);
```

- Once the matlabpool is open, parfor and spmd should work in the same as they would with a local pool

Demo 3

- http://intranet.mrc-cbu.cam.ac.uk/computing/cluster-demo/#3

Best Practice

Login nodes:

- These are a shared resource – think about other users when you're using them
- Close any interactive SPM/Matlab sessions when you have finished using them, especially if your session has been using a lot of memory.
  - Open matlab sessions use 2 limited resources – memory and matlab licenses
  - If you don't want to close your session, run "clear all" to release memory
- Please don't run large compute jobs or matlabpools on the login nodes!

Best Practice

Scheduling system:

- Develop and debug your scripts on the login nodes before submitting to the scheduler
- Make a note of the resources your job requires – especially memory and cpu time
- Requesting the appropriate resources allows the scheduling system to operate most efficiently. The scheduler will try to launch as many jobs on each machine as possible, without overloading that machine
  - Under-requesting (e.g. requesting 4GB RAM when you need 16GB) can cause the machines to run out of memory and become unresponsive
  - Over-requesting (e.g. requesting 64GB RAM when you only need 16GB) means fewer jobs will run simultaneously
  - Over-requesting also means your job could wait for longer (there are more machines available to handle a 4GB job than a 60GB job, there are more machines with 12 cores than with 16 cores, etc)

Best Practice

Storage:

- Imaging space is unquota'd, not infinite…
- Clean up after your analyses – e.g. delete intermediate pre-processing images once you've finished with them
- If you are using AA version 4, make sure garbage collection is turned on
- Don't copy raw data from /mridata or /megata into your /imaging directory
- Don't create multiple copies of the same files
- You can read data from other peoples' imaging space – you don't need to copy data from their space to your own

Further Information

Computing group intranet page:

http://intranet.mrc-cbu.cam.ac.uk/computing/cluster

Imaging wiki:

http://imaging.mrc-cbu.cam.ac.uk/