**Guide for Big Purple**

<http://bigpurple-ws.nyumc.org/wiki/index.php/BigPurple_HPC_Cluster>

**Logon to Big Purple**

When connected to the nyumc network, use putty to login to bigpurple.nyumc.org using your Kerberos ID (e.g. hardsr01)

**Data Transfer**

When you first logon you will be connected to your home directory (e.g /gpfs/home/hardsr01). This directory is only 100 GB, and so should not be used for the storage of data or files.

The lab data directory is located at /gpfs/data/helab/. In this directory there is a folder for each user (e.g. /gpfs/data/helab/hardsr01/) . This is where you should store all of your data and scripts.

**Data Transfer (Linux/Mac)**

The quickest way to transfer data from gago/gogo is using the SCP command

e.g. from gogo you would run

scp -r /data/gogodisk3/Richard/ hardsr01@bigpurple.nyumc.org:/gpfs/data/helab/hardsr01/data/

This will copy all of the files in the directory /data/gogodisk3/Richard/ to the directory /gpfs/data/helab/hardsr01/data/ on bigpurple

To copy data back you would run this command on gogo:

scp -r hardsr01@bigpurple.nyumc.org:/gpfs/data/helab/hardsr01/data/ /data/gogodisk3/Richard/

This will copy all of the files in the directory /gpfs/data/helab/hardsr01/data/ on bigpurple to the directory /data/gogodisk3/Richard/

**Data Transfer (Windows)**

When running from windows you can also use winscp and login to bigpurple.nyumc.org. This will give you a folder view of your local computer, and for the cluster.

**Using cluster**

When you first logon to the cluster you are connected to a login node. This should not be used for anything apart from submitting jobs.

It is possible to get an interactive node. From this you are able to run matlab, and test out your code.

**To get interactive node (Linux/Mac)**

srun -p cpu\_short --mem-per-cpu=4G -t 00-12:00:00 --pty bash

This will get you an interactive node that will last for 12 hours, and 4G memory

From this interactive node, you can run matlab:

module load matlab/R2018a

matlab -nodisplay -nodesktop -singleCompThread

If you want the display version of matlab (i.e. to make figures):

1. Open a gago or gogo vnc session
2. Open a terminal
3. Logon to bigpurple from this terminal using the command (note that you should change hardsr01 to your username)

ssh hardsr01@bigpurple.nyumc.org

1. Open an interactive session with X11 forwarding (this allows display elements)

srun -p cpu\_short --nodes=1 --tasks-per-node=2 --mem-per-cpu=8G -t 00-08:00:00 --x11 --pty bash

1. Run matlab

module load matlab/R2018a

matlab &

**To get interactive node (Linux/Windows)**

If you want to run graphical interfaces on Windows, then you will need to install vcxsrv. This can be downloaded from <https://sourceforge.net/projects/vcxsrv/>. When installing, choose full install. If it asks for administrator rights ask Max or Richard to install it for you.

Once you have installed it, you will need to run vcxsrv -> Xlaunch from the start menu. It will then ask you for some configuration options. I picked the default options.

To connect using putty, use the usual address for bigpurple. You will also need to enable X11 forwarding, which can be found in the menu Connection -> SSH -> X11

You should now be able to start graphical matlab by running steps 4 and 5 from “To get interactive node (Linux/Mac)”

**First Matlab job**

The main advantage of the cluster is the ability to submit jobs. The easiest way to do this is write scripts (which are plain text files). It is easiest to write one script to submit jobs (e.g. Example\_job\_submit.bash), and one script that describes the job (e.g. Example\_job.bash).

**Example job script: (Example\_job.bash)**

#!/bin/bash

#SBATCH -p cpu\_short

#SBATCH --nodes=1

#SBATCH --tasks-per-node=1

#SBATCH --time=12:00:00

#SBATCH --mem-per-cpu=2G

module load matlab/R2018a

matlab -nodisplay -nodesktop -singleCompThread -r "x=$1+1;save('x.mat','x');exit;"

This runs 2 matlab commands.

x = var +1;

save('x.mat','x');

where var is the variable that you submitted when running Example\_job\_submit.bash

**Example job submit script: (Example\_job\_submit.bash)**

#!/bin/bash

sbatch ./Example\_job.bash 2

This will submit a job to be run on the server (Example\_job.bash) with var=2 being submitted

**Running scripts**

pwd (get current directory)

cd Jobs (change directory)

ls (list files in directory)

cat myfirstjob\_submit.bash (display contents of file)

head myfirstjob\_submit.bash (display top lines of file)

rm slurm-96545.out (delete file)

mv myfirstjob\_submit.bash Jobs/ (move file to directory Jobs)

rm slurm\* (delete all files in current directory that start slurm)

To run the script, you first need to make sure that the script can be run. To do this:

chmod +x \*.bash

You can then run the job submit script:

./Example\_job\_submit.bash

**Checking status of jobs**

To check the status of your jobs (i.e. what jobs you have running and queued)

squeue -u hardsr01

This will return a list of your currently running jobs

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)

794090 cpu\_medium Purple\_S hardsr01 R 1-02:24:27 1 cn-0033

808381 cpu\_short Purple\_S hardsr01 R 1:52:17 1 cn-0022

To check the number of jobs running:

squeue -u wmm247 | grep myf | wc –l

where grep myf displays only lines which contain myf

where wc –l counts the number of lines

to check the status of jobs running:

tail -n 2 slurm\*

to check errors/killed processes

grep Error slurm\*

grep Killed slurm\*

to cancel a job:

scancel -i 794090

rm slurm-794090.out (to delete slurm file)

to cancel all of your jobs:

scancel -u hardsr01

rm slurm\*

to selectively cancel jobs:

scancel --name=myfirstjob.bash

to re-submit a failed job:

modify any problem line on myfirstjob.bash on interactive

cat myfirstjob\_submit.bash

sbatch ./myfirstjob.bash /gpfs/data/rudylab/William/180706\_WT\_EM1M3

to check the jobs completed:

grep -l Complete slurm\* | xargs head -n 1

to repeat old specific command:

type beginning of old command

then > ctr+A R Y R (keep pressing R to move through)

to check the memory spent in jobs:

sacct -u dl2820 --units=G --format="JobID,CPUTime,MaxRSS,State" | grep bat

sacct -u dl2820 --units=G --format="JobID,CPUTime,MaxVMSize,State" | grep bat

**Recommendations for altering scripts to run on cluster**

1. Changing paths

When running scripts on the cluster, you will need to have copied all the necessary scripts and data from gago/gogo, as it is not possible to read this data from gago/gogo from the cluster.

Likely the first commands you will need to run will add the path

e.g.

cd('/gpfs/data/helab/hardsr01/project/scripts/')

AddPaths('/gpfs/data/helab/hardsr01/project/')

where AddPaths is a script that you need to write that adds all the necessary directories containing your scripts to the path. I passed a directory name to this function, which the function can then use to add paths relative to the base path.

e.g. AddPaths.m

function AddPaths(baseDirectory)

restoredefaultpath;

addpath(genpath([baseDirectory '/scripts/']));

2. When to use parallel and when to just submit more jobs

My recommendation is that you should only use parallel if you are doing permutations. If instead you are just running 20 subjects, then I would submit a separate job for each subject

**Running a Matlab For loop in Parallel**

If a for loop in your code takes a long time to run, it may be helpful to parallelize it. This makes several iterations of the loop run at the same time, greatly increasing speed. A basic job script, such as the one given above (Example\_job.bash), runs on one core on one computing node. A simple way to parallelize is to still use one node, but utilize multiple cores on that node.

You will need to edit both your job script (such as Example\_job.bash) as well as your matlab code.

First, in your job script:

change the line

#SBATCH --tasks-per-node=1

to

#SBATCH --tasks-per-node=20

(20 is a good bet; some nodes have more available cores)

Add the lines

export SCRATCH=/gpfs/scratch/[YOUR KERBEROS ID]

mkdir -p $SCRATCH/$SLURM\_JOB\_ID

After the command to call matlab, add the following line

rm -rf $SCRATCH/$SLURM\_JOB\_ID

We will call this variable (SCRATCH) in the MATLAB script. This helps eliminate problems if you run multiple parallel jobs at the same time.

Next, in your MATLAB script:

1. Before your for loop, add the following code:

pc = parcluster('local');

% store temporary files in the 'scratch' drive on the cluster, labeled by job ID

pc.JobStorageLocation = strcat(getenv('SCRATCH'), '/', getenv('SLURM\_JOB\_ID'));

% enable MATLAB to utilize the multiple cores allocated in the job script

% SLURM\_NTASKS\_PER\_NODE is a variable set in the job script by the flag --tasks-per-node

% we use SLURM\_NTASKS\_PER\_NODE - 1, because one of these tasks is the original MATLAB script itself

parpool(pc, str2num(getenv('SLURM\_NTASKS\_PER\_NODE'))-1);

2. Change your for statement to parfor

Example:

parfor i = 1:19

disp(i);

end

This loop would now run in parallel, meaning the numbers 1 - 19 would be displayed simultaneously, rather than one after another. Each iteration (i.e., for each value of i between 1 and 19) is run on a separate CPU core. Assuming it takes 1 ms to run each iteration of this loop, it would normally take 19 ms to complete the loop. After parallelizing, the entire loop would complete in just 1 ms.

There are some restrictions on the types of variables you can use with parfor. See matlab documentation for help.

**Potential Errors with parallel**

If a job finishes unexpectedly, then you should check the output file for that job. This will usually be found in the directory you were in when you submitted the job, and be called slurm-xxxxxxx.out where xxxxxxx corresponds to the job ID.

One error I have encountered when running the parpool command is:

Error in test (line 57)

pool = parpool('local',16);

Caused by:

Error using parallel.internal.pool.InteractiveClient>iThrowWithCause (line

676)

Failed to initialize the interactive session.

Error using

parallel.internal.pool.InteractiveClient>iThrowIfBadParallelJobStatus

(line 790)

The interactive communicating job failed with no message.

This error seems to occur if there is not enough memory for the parallel jobs. To fix this you can alter the line in the job script, and increase the amount of memory available per cpu

#SBATCH --mem-per-cpu=2G

Another error:

https://www.mathworks.com/matlabcentral/answers/253765-error-with-parpool-error-using-parpool-line-103-not-enough-input-arguments