Pydpiper on Graham

Compute Canada's Graham cluster is an HPC system at the University of Waterloo, replacing our old use of the SciNet system. In general, the HPF grid is slightly more convenient for typical uses, but Graham has some advantages (in particular, it's much easier to get Oxford folks or external collaborators a Graham account).

For general references, see the Compute Canada wiki here and here.

Compute Canada account creation

If you don't have one already, apply for a Compute Canada account here. Most people will need an identifier corresponding to their PI; contact Ben if you're associated with Jason.

Login

Once your CC account is approved and created, you can log in:

```
$ ssh user@graham.computecanada.ca -o ServerAliveInterval=30
```

where *user* is your Compute Canada username and your password is your Compute Canada web password.

Graham and Niagara seem to drop ssh connections frequently, which is prevented by the above ServerAliveInterval option. (You can also specify this in your /home/yourname/.ssh/config file on your local machine, e.g.

~/.ssh/config

```
Host graham
    HostName graham.computecanada.ca
    User bcdarwin
    ServerAliveInterval 30
Host hpf
    HostName hpf.ccm.sickkids.ca
    User bdarwin
```

solves this problem and additionally lets you log in simply via 'ssh graham'.

There's also no separation of login vs. dev/submit nodes, so you're ready to use the

Scheduler

Graham runs the Slurm scheduler (and `squeue`, `sbatch`, ..., suite) just as at MICe; see, e.g., here. We have `qbatch` installed as a convenient interface.

Its admins request you do **not** submit large numbers of separate jobs in a short time but either use an array job or wait 1s between submissions. Similarly, they request you don't spam 'squeue' via a script.

Initial setup

Add the following to your Graham `~/.bashrc`:

```
module use /project/def-jlerch/tools/modulefiles
```

and then source this file or log in again.

Special allocations

If your PI has a Compute Canada RAC allocation above the default (currently no MICe PIs do, so you can skip this section), you'll need to tell the scheduler which allocation to use to run your jobs or you'll get an error from sbatch ("You are associated with multiple _cpu allocations..."). One way to do so is via environment variables:

```
export SLURM_ACCOUNT=rrg-jlerch-ac    # or 'def-jerch' for the default allocation
export SBATCH_ACCOUNT=$SLURM_ACCOUNT
export SALLOC_ACCOUNT=$SLURM_ACCOUNT
```

You can also pass `--account=...` to `squeue` - although the above way is more generally useful since, e.g., Pydpiper is unaware of any Slurm-related details - and there might be a way to set a per-user default through the CCDB web portal.

Looking up allocation information

From https://ccdb.computecanada.ca/ (My Account -> Account Details) one sees:

RAPI	Group Name	Title	Allocations
iww-954-aa	def-jlerch	Default Resource Allocation Project	2 allocations
...	...	...	...
iww-954-ad	rrg-jlerch-ac	Using Medical Imaging to Understand the Relationship Between Genetics, Development and Disease	5 allocations, RAC 2015

The idea here is that jobs are by default submitted under the allocation of group def-jlerch.

Running a Tmux(/Screen) session

Idle SSH sessions from MICe to Graham seem to be dropped (but see the above section on `~/.ssh` configuration, which resolves this). For this and other reliability reasons, you may with to run a more permanent interactive process - i.e., a terminal multiplexer, likely either screen or tmux - on a Graham login node. (See the "Pydpiper on HPF" page for more details.) If a subsequent SSH connection to Graham is routed to a different node than the one running your multiplexer, simply connect from the former to the latter (e.g., '[gra-login-1 ~]$ ssh gra-login-3').

(Note that due to the non-interactive mode of running Pydpiper on Graham, this won't cause your pipeline to stop.)

Interactive use

Unlike HPF, the login nodes are not intended for interactive work such as running CPU-intensive statistical models; instead, use `salloc` to get an interactive session (for up to 24:00:00) as per the Graham wiki. I recommend doing this from *inside* a screen/tmux session (see above) as at present there seems to be an issue with ssh connections being dropped.

Disk space

Your main disk spaces are $HOME, $SCRATCH, and $PROJECT. $HOME is small and should be used mostly for text files, etc. $SCRATCH should be used for running pipelines. It is not backed up and inactive files are deleted regularly (you will receive emails prior to any deletion). $PROJECT can be used for storing completed pipelines and analyses.

Writing to $PROJECT

Note that while $SCRATCH has a large per-user quota, $PROJECT allocation is per-group. As such, you should always write files with group given by your allocation (e.g., `def-jlerch`) or you'll fill up your tiny (2MB!) personal $PROJECT quota. Thus, either `chgrp` files before moving them to $PROJECT or, if creating files, run

```
newgrp def-jlerch
```

beforehand automatically create them with appropriate group ownership.

You only need to run this command if you're creating files in $PROJECT, while pipelines and so on should be run in $SCRATCH.

A couple caveats:

- don't add this to your ~/.bashrc - it breaks it somehow.
- As of March 2020, running this command seems to unset $LD_LIBRARY_PATH, so you need to run it *before* loading any environment modules.

Data transfer

Broadly similar to SciNet.

For transferring data such as your input `.mnc` files, you can use `scp` or `rsync` as usual:

```
scp -r /hpf/largeprojects/MICe/yourname/inputfiles/ yourname@graham.computecanada.ca:/scratch/yourname
```

This code must be run from a terminal running in hpf or our local system, not Graham.

Software

```
module load mice-env
```

(This module is missing our visual tools such as Display and register; see the Visualization on Graham page for those.)

Of course, you may as usual load a specific version of this module, e.g., to fix software versions for a specific analysis.

Running Pydpiper

Since long-running jobs such as Pydpiper servers are discouraged on the login nodes (although the Pydpiper server uses relatively little CPU, so for smaller pipelines it might be acceptable), we'll submit the Pydpiper server itself to a compute node. The salloc command seems to have a 24hr time limit, so we'll submit it as a non-interactive job, e.g., using qbatch (the maximum walltime allowed is 672hr, which should be more than enough for most pipelines; try to choose a more reasonable limit):

```
$ cd $SCRATCH/pipeline-dir
$ qbatch --mem=4G --walltime 96:00:00 -- MBM.py ...
# note these resources are for the server, not executors
```

The server is submitted as a job to the queue; once launched, it will submit additional executors itself as needed.

Monitoring your pipeline

As usual information is written to the pipeline.log file. The server's standard output which is normally visible in your terminal is redirected to logs/slurm-STDOUT-<jobid>.out. If you like, you can redirect it to a different location via submitting with echo 'MBM.py ... > some-file.txt' | qbatch ... ; note that the output will be buffered by the OS so you won't always see things immediately as they are written out.

TODO are the default stdout logs visible in real time; can you make Pyro (TCP) connections to the compute nodes via check_pipeline_status.py?

Acknowledging Graham/Compute Canada

See here; please edit if you find a paper for Graham (as there was for SciNet).