*EE3801 2021 L6 Lab (Submission due Oct 20, 11:59 pm)*

In this lab, you will use AWS to set up and execute a pipeline to process the three different kinds of data you worked with in the previous lab.

The lab consists of five parts:

1. Installing and testing software on AWS (Estimated time: 60 mins)
2. Setting up the data pipeline (Estimated time: 10 mins)
3. Executing the data pipeline (Estimated time: 20 mins)
4. Checking the output (Estimated time: 15 mins)
5. Wrapping up (Estimated time: 10 mins)

**Part I) Installing and testing software on AWS**

1. First, we will create a Terminal window, activate your aws conda environment, and edit the config file to modify the instance type for the head and compute nodes (Windows users please be sure to do everything in the WSL environment):

```
(base) $ conda activate aws
(aws) $ nano ~/cluster-config.yaml
```

Modify the following in the "HeadNode" section:

```
HeadNode:
  InstanceType: t3a.nano
```

In order to spread out the load, use the following table to find out the instance type you should use according to the last number before the letter in your student number (e.g. use the number 4 if your student number is A0171234X):

| Last Digit | Type | Last Digit | Type | Last Digit | Type |
|---|---|---|---|---|---|
| 0 | t3a.large | 4 | m5.large | 8 | m5d.large |
| 1 | t3.large | 5 | m6i.large | 9 | m5n.large |
| 2 | m5a.large | 6 | m4.large | | |
| 3 | t2.large | 7 | m5ad.large | | |

and the table below to fill in the "Name" and "InstanceType" in the Scheduling section (replace the "." in InstanceType with "-" when entering the "Name"):

```
Scheduling:
  Scheduler: slurm
  SlurmQueues:
  - Name: queue1
    ComputeResources:
    - Name: t3a-nano
      InstanceType: t3a.nano
```

| Last Digit | Type | Last Digit | Type | Last Digit | Type |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| 0 | m5.4xlarge | 4 | r5n.2xlarge | 8 | r5.2xlarge |
| 1 | z1d.2xlarge | 5 | r5b.2xlarge | 9 | r5a.2xlarge |
| 2 | m5a.4xlarge | 6 | r5d.2xlarge | | |
| 3 | r5dn.2xlarge | 7 | r5ad.2xlarge | | |

You can find out more about different instance types at:

https://aws.amazon.com/ec2/instance-types/
https://ap-southeast-1.console.aws.amazon.com/ec2/v2/home?region=ap-southeast-1#InstanceTypes:

In order to use pcluster, one of the limitations is that the head node and the compute nodes are required to have the same architecture. In order to keep things simple, we will be using "x86_64" architecture for both the head and compute nodes. The head node is just used to submit jobs and handle file system requests, so we will usually find a relatively cheap instance type with the same architecture and 8 GB of memory (in order to handle the file system requests), which includes "t3a.large", "t3.large", "m5a.large", etc. (see the description of the instance types in the first link above, and the prices in the second link, where you can scroll to the right of the table, and click on the "On-Demand Linux pricing" to sort by price). Avoid instance types that have their architecture listed as "i386, x86_64" as they are often incompatible with computer nodes that have architecture listed as "x86_64".

For the compute nodes, we recommend instance types with a minimum of 64 GB of memory. This will allow us to have enough memory to read in and split the .ns5 neural data files, which are around 40 GB in size. Due to the AWS limit of 64 vCPUs per user, if you use an instance with a large number of vCPUs, e.g. m6g.12xlarge, which has 48 vCPUs, you will only be able to create 1 compute node with 48 vCPUs as creating 2 compute nodes will require 96 vCPUs, which will exceed your vCPU limit. On the other hand, if you use instance types with 8 vCPUs, you will be able to create 8 instances for a total of 64 vCPUs. So choosing the right instance types will allow you to have more vCPUs available for your jobs.

2. Next, we will do a one-time set up of your environment on AWS ParallelCluster. We will start by creating a cluster on AWS (enter the command below all on the same line):

```
(aws) $ pcluster create-cluster --cluster-configuration
~/cluster-config.yaml --cluster-name MyCluster01
```

If you get an error like:

```
Beginning cluster creation for cluster: MyCluster01
Creating stack named: parallelcluster-MyCluster01
Status: parallelcluster-MyCluster01 - ROLLBACK_IN_PROGRESS
Cluster creation failed.  Failed events:
  - AWS::CloudFormation::WaitCondition MasterServerWaitCondition Received
FAILURE signal with UniqueId i-0e06294577f6606d0
```

You might need to delete the cluster first by doing:

```
(aws) $ pcluster list-clusters --region ap-southeast-1
MyCluster01  ROLLBACK_COMPLETE   2.9.1
(aws) $ pcluster delete-cluster --region ap-southeast-1 --cluster-name
MyCluster01
```

You can then try re-creating the cluster using a different head node.

If it says that the command plcuster is not found, you might need to redo Step 6 in Lab 4 to make sure your path is set up properly.

3. Once you receive the notification email stating that your head node is running, login to the cluster:

```
(aws) $ pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1
--cluster-name MyCluster01
```

4. Change to the directory where the snapshot you saved in Lab 4 is mounted:

```
[ec2-user@ip-10-0-5-43 ~]$ cd /data
```

If you receive an error like:

```
-bash: cd: /data: No such file or directory
```

You might have to wait a couple of minutes for AWS to mount your /data volume before trying again.

5. Miniconda has already been installed for you, along with an environment called env1. So you can initialize your conda environment by doing:

```
[ec2-user@ip-10-0-5-43 data]$ miniconda3/bin/conda init
```

This will add some commands to your ~/.bashrc, so you can reload it to get the conda commands to work:

```
[ec2-user@ip-10-0-5-43 data]$ source ~/.bashrc
(base) [ec2-user@ip-10-0-5-43 data]$ conda activate env1
(env1) [ec2-user@ip-10-0-5-43 data]$
```

Your prompt should now be prefixed by (env1) instead of (base).

You should also copy the /data/aws directory back to ~/.aws:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ cp -r /data/aws ~/.aws
```

Check that your aws credentials are working properly by sending yourself a notification email (remember to replace the numbers in the topic-arn with your account number):

```
(env1) [ec2-user@ip-10-0-5-43 data]$ aws sns publish --topic-arn
arn:aws:sns:ap-southeast-1:0123456789012:awsnotify --message
"ClusterTest"
```

6. Download software for reading the eye-tracker data files:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ curl -L -o
eyelink-display-software_1.11_x64_debs.tar.gz
https://www.dropbox.com/s/91d1iiffllc4seo/eyelink-display-software_1.11_x
64_debs.tar.gz?dl=1
```

7. Install the eye-tracker software by first uncompressing and extracting the files contained in the archive you downloaded:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ tar -xvzf
eyelink-display-software_1.11_x64_debs.tar.gz
```

Change to the directory containing the extracted files:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ cd eyelink-display-software_1.11_x64
```

Before you install the eye-tracker software, you will need to install a package manager called dpkg:

```
(env1) [ec2-user@ip-10-0-5-43 x64]$ sudo yum install dpkg
```

Enter “y” twice to install the software. Once the installation is complete, you can install the eye-tracker software:

```
(env1) [ec2-user@ip-10-0-5-43 x64]$ sudo dpkg -i ./edfapi_4.0_amd64.deb
```

The last command will result in a series of warnings and errors, but you can just ignore them. Copy the following library to your conda lib directory:

```
(env1) [ec2-user@ip-10-0-5-43 x64]$ cp -p /lib/libedfapi.so
/data/miniconda3/envs/env1/lib/
```

8. In order to check that a spike sorting software package called MountainSort was installed properly, do the following (this might take a couple of minutes):

```
(env1) [ec2-user@ip-10-0-5-43 x64]$ ml-list-processors
banjoview.cross_correlograms
ephys.bandpass_filter
ephys.compare_ground_truth
ephys.compute_cluster_metrics
```

```
ephys.compute_cross_correlograms
```

You should see a list of methods, of which we are only showing the first few lines above.

9. We will now create a directory for you to clone the GitHub repositories:

```
(env1) [ec2-user@ip-10-0-5-43 x64]$ cd /data
(env1) [ec2-user@ip-10-0-5-43 data]$ mkdir src
(env1) [ec2-user@ip-10-0-5-43 data]$ cd src
```

10. We will now clone the GitHub repositories (replace the username for the PyHipp repository with your username):

```
(env1) [ec2-user@ip-10-0-5-43 src]$ git clone
https://github.com/shihchengyen/PyHipp
(env1) [ec2-user@ip-10-0-5-43 src]$ git clone
https://github.com/grero/DataProcessingTools
(env1) [ec2-user@ip-10-0-5-43 src]$ git clone
https://github.com/nwilming/pyedfread
```

11. We will install these modules using pip:

```
(env1) [ec2-user@ip-10-0-5-43 src]$ cd PyHipp
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ pip install -r requirements.txt
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ pip install -e .
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cd ../DataProcessingTools
(env1) [ec2-user@ip-10-0-5-43 DataProcessingTools]$ pip install -e .
(env1) [ec2-user@ip-10-0-5-43 DataProcessingTools]$ cd ../pyedfread
(env1) [ec2-user@ip-10-0-5-43 pyedfread]$ pip install .
(env1) [ec2-user@ip-10-0-5-43 pyedfread]$ cd /data
```

12. In order to try processing the data, we will want to run on one of the compute nodes as it has at least 64 GB of memory. So do the following to login to a compute node:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ srun --pty /bin/bash
```

If you get an error saying that the srun command is not found, you may have to log out and log back in. This is usually due to some network volumes not being mounted yet after the creation of the cluster.

It will take a little time for the compute node to be started up, after which you should receive an email. Your prompt should also change to something like (the highlighted portion should correspond to the instance type you have specified in your cluster config file):

```
(base) [ec2-user@queue1-dy-m54xlarge-1 ~]$
```

Activate the env1 conda environment as usual, and change to the /data directory:

```
(base) [ec2-user@queue1-dy-m54xlarge-1 ~]$ conda activate env1
(env1) [ec2-user@queue1-dy-m54xlarge-1 ~]$ cd /data
```

13. We will now start up ipython (which is a little easier to use than regular python) to test that everything was installed properly:

```
(env1) [ec2-user@queue1-dy-m54xlarge-1 data]$ ipython
In [ ]: import PyHipp as pyh
In [ ]: pyh.pyhcheck('hello')
hello
In [ ]: from pyedfread import edf
In [ ]: cd src/pyedfread
In [ ]: samples, events, messages = edf.pread('SUB001.EDF')
In [ ]: events.shape
Out[ ]: (485, 30)
```

Note that the “In [ ]:” prompt indicates that the command should be entered in iPython.

14. We will now go through and test the software that has been installed. We will start by processing the Ripple .nev files containing the signals sent by Unity via the parallel port. We will use a Python class named RPLParallel defined in the PyHipp repository to create a RPLParallel object:

```
In [ ]: cd /data/picasso/20181105/session01
In [ ]: pyh.RPLParallel(saveLevel=1)
```

which should give you the following:

```
Object created
Opening .nev file, creating new RPLParallel object...
Object saved to file rplparallel_d41d.hkl
Out[ ]: <PyHipp.rplparallel.RPLParallel at 0x7fb0fcaa2160>
```

The “saveLevel=1” argument tells the function to save the RPLParallel object into the current directory after it has been created. It is a feature common to all the classes defined in the PyHipp repository.

We will do the same for the eye fixation session:

```
In [ ]: cd ../sessioneye
In [ ]: pyh.RPLParallel(saveLevel=1)
```

15. The navigation session and the fixation session each contain a .ns5 file that contains the neural data from 110 channels. We can separate out one of the channels (Channel 9) by doing:

```
In [ ]: cd ../session01
In [ ]: pyh.RPLSplit(channel=[9])
```

which will return the following showing that a RPLRaw object for Channel 9 was created (as the .ns5 is pretty large, the entire process might take 10 to 15 minutes):

```
Object created
.ns5 file loaded.
Processing channel 009
Calling RPLRaw for channel 009
Object created
Object saved to file rplraw_d41d.hkl
Channel 009 processed
Out[ ]: <PyHipp.rplsplit.RPLSplit at 0x7f07b25a7828>
```

We do not need to specify the “saveLevel=1” argument as the primary function of the function is to create the appropriate RPLRaw objects, which are then saved by default.

If you would like to be notified when the function is done, you can follow these instructions to use CloudWatch to set up an alarm to notify you when the Head Node’s Network Out Bytes falls below 1,000,000:

https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ConsoleAlarms.html

If you check the current directory by doing:

```
In [ ]: ls
```

you will see a directory named “array01”. If you check that directory and its subdirectory:

```
In [ ]: ls array01
In [ ]: ls array01/channel009
```

you will see a file named rplraw_d41d.hkl that contains the raw data just for Channel 9.

16. In order to generate the low-pass filtered signals, we will call the RPLLFP function from the channel directory:

```
In [ ]: cd array01/channel009
In [ ]: pyh.RPLLFP(saveLevel=1)
```

which will load the RPLRaw object and create a RPLLFP object:

```
Object loaded from file rplraw_d41d.hkl
Object created
Applying low-pass filter with frequencies 1.0 and 150.0 Hz
Object saved to file rpllfp_6eca.hkl
Out[7]: <PyHipp.rpllfp.RPLLFP at 0x7f07b259f940>
```

17. Similarly, to generate the high-pass filtered signals, we will call the RPLHighPass function:

```
In [ ]: pyh.RPLHighPass(saveLevel=1)
```

which will load the RPLRaw object and create a RPLHighPass object:

```
Object loaded from file rplraw_d41d.hkl
Object created
Applying high-pass filter with frequencies 500.0 and 7500.0 Hz
Object saved to file rplhighpass_b59f.hkl
Out[7]: <PyHipp.rplhighpass.RPLHighPass at 0x7f07b23ca400>
```

You can see the two new files created in the current directory by doing:

```
In [ ]: ls
```

which will give you:

```
rplhighpass_b59f.hkl  rpllfp_6eca.hkl  rplraw_d41d.hkl
```

18. We can process the Unity files by doing:

```
In [ ]: cd ../../
In [ ]: pyh.Unity(saveLevel=1)
```

which will return:

```
Object created
Object loaded from file rplparallel_d41d.hkl
Object saved to file unity_71bf.hkl
Out[ ]: <PyHipp.unity.Unity at 0x7facccd89400>
```

You will see that in creating the Unity object, the previously saved RPLParallel object was loaded to extract some information. If the RPLParallel was not saved previously, the information in the object will have to be recomputed from the raw data files. This reduction in unnecessary recomputation was one of the principles on which the objects in the PyHipp repository were designed.

19. This next command will process the eye-tracking files for both the navigation (180702.edf) and fixation (P7_2.edf) sessions, create eyelink objects, and save them to the session01 and sessioneye directories:

```
In [ ]: cd ..
In [ ]: pyh.EDFSplit()
```

which should return the following:

```
Reading calibration edf file.
Object created
Object saved to file eyelink_24d5.hkl
```

```
Reading navigation edf file.
...
Object saved to file eyelink_24d5.hkl
Object created
Out[ ]: <PyHipp.eyelink.EDFSplit at 0x7faccce0f1d0>
```

Similar to the RPLSplit function above, we do not need to specify the “saveLevel=1” argument as the primary function of EDFSplit is to create and save the eyelink objects.

20. In order to align the Ripple, Unity, and Eyelink data, we will call the following function:

```
In [ ]: cd session01
In [ ]: pyh.aligning_objects()
```

which should end with the following:

```
finish aligning objects
```

21. For performing the raycasting, you can do:

```
In [ ]: pyh.raycast(1)
```

You should see some text ending with:

```
Object loaded from file unity_71bf.hkl
Object loaded from file eyelink_24d5.hkl
Found path: /data/RCP/VirtualMaze.x86_64
```

This might take about 20 minutes to complete, so we will instead just check that it started running properly before stopping it. Create a new Terminal window, activate the aws conda environment, and ssh to your cluster:

```
(base) $ conda activate aws
(aws) $ pcluster ssh -i ~/MyKeyPair.pem --region ap-southeast-1
--cluster-name MyCluster01
```

Change to the directory from which you ran the raycast function:

```
(base) [ec2-user@ip-10-0-5-43 ~]$ cd /data/picasso/20181105/session01
```

The raycast function writes to a log file as it is running, so we can check the log file’s contents by doing:

```
(base) [ec2-user@ip-10-0-5-43 session01]$ tail -f VirtualMazeBatchLog.txt
```

You have used the tail function before to look at the last few lines of a file in Lab 4, but by adding the -f flag, it will now continuously show you the last few lines of the log file. This is one of the

ways to monitor a program as it is running. The command above should progressively show you something like:

```
Add "-logfile <log file location>.txt" to see unity logs during the data
generation
There may be a need to copy the libraries found in the directory
'Plugins' to a new folder called 'Mono'

More command line arguments can be found at
https://docs.unity3d.com/Manual/CommandLineArguments.html
Session List detected!
Setting density to : 220
Setting radius to : 1
Queuing /data/picasso/20181105/session01
1 sessions to be processed
Starting(1/1): /data/picasso/20181105/session01
5.037168%: Data Generation is still running. 10/3/2021 2:45:51 AM
```

The last line above indicates that the program has started running, so everything should be installed properly. At this point, you can terminate the tail program by typing “Ctrl-c”. You can go back to your first Terminal window that was running ipython, and type “Ctrl-c” to interrupt the raycast function. You can then exit ipython by typing “Ctrl-d”.

If ipython does not respond, you can do the following from your second Terminal window:

```
(base) [ec2-user@ip-10-0-5-43 session01]$ ps -ef | grep ipython
ec2-user 12958  7414  0 13:37 pts/0    00:00:00
/data/miniconda3/envs/env1/bin/python
/data/miniconda3/envs/env1/bin/ipython
ec2-user 13459 12716  0 13:43 pts/1    00:00:00 grep --color=auto ipython
```

The “ps” command lists running processes, which are then sent using the “pipe” command (“|”) to the “grep” command to search for the string “ipython”. The process ID for ipython is highlighted above, which you can proceed to terminate by doing the following (replace the process ID below with the one you obtained from the command above):

```
(base) [ec2-user@ip-10-0-5-43 session01]$ kill -9 12958
```

If the ipython process was terminated properly, you should see the bash prompt in the first Terminal window again:

```
(env1) [ec2-user@queue1-dy-m54xlarge-1 data]$
```

You can then exit from the compute node by doing:

```
(env1) [ec2-user@queue1-dy-m54xlarge-1 data]$ exit
```

22. From the second Terminal window, you can check that a few files have now been created in the session01 directory:

```
(base) [ec2-user@ip-10-0-5-43 session01]$ ls
 181105_Block1.nev      eyelink_24d5.hkl        slist.txt
 181105_Block1.ns5      logs.txt                unity_71bf.hkl
 array01                missingData.csv         unityfile_eyelink.csv
 binData.hdf            RawData_T1-400          VirtualMazeBatchLog.txt
 data_raw6.hkl          rplparallel_d41d.hkl
```

    Once you have verified the files have been created, you can type Control-d to logout from the head node. **Include a screenshot with the above output in your lab report.**

23. The last thing to do is to copy the following two files for spike sorting from the PyHipp directory to the /data directory:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ cp /data/src/PyHipp/geom.csv
/data/picasso
(env1) [ec2-user@ip-10-0-5-43 data]$ cp /data/src/PyHipp/sort.sh.txt
/data/picasso
```

24. You can now delete the installer for the eye-tracking software (which should be entered in one line with no line breaks):

```
(env1) [ec2-user@ip-10-0-5-43 data]$ rm -r
eyelink-display-software_1.11_x64*
```

    We will also delete the files we created to test the raycasting:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ cd picasso/20181105
(env1) [ec2-user@ip-10-0-5-43 data]$ rm session*/eyelink*hkl
(env1) [ec2-user@ip-10-0-5-43 data]$ cd session01
(env1) [ec2-user@ip-10-0-5-43 session01]$ rm *.hkl *.csv bin*hdf
VirtualMaze* *.txt
```

25. At this point, you should take a snapshot of the /data volume **from your computer**, so you will not have to go through the set up again:

```
(aws) $ update_snapshot.sh data 2
```

    You can move on to the next section while waiting for the snapshot to be completed. Once the snapshot is completed, you should receive an email notification from AWS.

**Part II) Setting up the data pipeline**

26. We are now ready to set up the data pipeline. We will want to create the objects in this order:
    a. RPLParallel (for both session01 and sessioneye)
    b. RPLSplit to create a RPLRaw object for each of the 110 channels (for both session01 and sessioneye)
    c. RPLLFP (which needs the RPLRaw object) for each of the 110 channels (for both session01 and sessioneye)
    d. RPLHighPass (which needs the RPLRaw object) for each of the 110 channels (for both session01 and sessioneye)
    e. Spike sorting (which needs the RPLHighPass objects for both session01 and sessioneye) for each of the 110 channels
    f. Unity (needs RPLParallel object)
    g. EDFSplit to create Eyelink objects (needs RPLParallel, and Unity if available) (for both session01 and sessioneye)
    h. Aligning_objects (needs RPLParallel, Unity, and Eyelink objects)
    i. Raycasting (needs Unity and Eyelink objects)

27. We will first create a script for submitting jobs to a queue by creating a copy of the script you used in Lab 4, and editing it:

```
(env1) [ec2-user@ip-10-0-5-43 data]$ cd /data/src/PyHipp
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cp slurm.sh pipeline-slurm.sh
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ nano pipeline-slurm.sh
```

28. We will now want to enter what we did above in ipython into the script, but we will just process 8 channels instead of the full 110 channels. However, what we did above involved changing directories numerous times, and that involved processing only one out of the 110 neural channels recorded. We will need to change directories quite a few times if we wanted to process 8 or 110 channels. So instead, we will make use of a command called processDirs in the DataProcessingTools that will automatically change directory to the appropriate directory in which to create the specified objects. In addition, we will want to take note of the time taken to process the data. You can copy and paste the following lines to the end of the file (you might want to hit the Esc key followed by $ to wrap long lines so it is easier to see and edit the text):

```
python -u -c "import PyHipp as pyh; \
import DataProcessingTools as DPT; \
import os; \
import time; \
t0 = time.time(); \
print(time.localtime()); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLParallel, saveLevel=1);
\
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, channel=[9, 31,
34, 56, 72, 93, 119, 120]); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLLFP, saveLevel=1); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLHighPass, saveLevel=1);
\
DPT.objects.processDirs(dirs=None, objtype=pyh.Unity, saveLevel=1); \
pyh.EDFSplit(); \
```

```
os.chdir('session01'); \
pyh.aligning_objects(); \
pyh.raycast(1); \
DPT.objects.processDirs(level='channel', cmd='import PyHipp as pyh; from
PyHipp import mountain_batch; mountain_batch.mountain_batch(); from
PyHipp import export_mountain_cells;
export_mountain_cells.export_mountain_cells();'); \
print(time.localtime()); \
print(time.time()-t0);"

aws sns publish --topic-arn
arn:aws:sns:ap-southeast-1:123456789012:awsnotify --message "JobDone"
```

When the processDirs function is called with “level” and “cmd” arguments, it will find all the subdirectories that are at the appropriate level in the data hierarchy (in this case channel directories), and run the specified command in those directories. This will create a job that will perform the spike sorting and save the appropriate spiketrain files into cell directories as discussed in the lecture.

The last Python command will take the difference in time between the start and the end of the job, and print out the difference in the form of number of seconds.

We will also edit the following line in the file to give the job more time (24 hours) to run:

```
#SBATCH --time=24:00:00   # walltime
```

as well as to make the job name and the slurm output files more distinct:

```
#SBATCH -J "pipe"   # job name

#SBATCH -o pipe-slurm.%N.%j.out # STDOUT
#SBATCH -e pipe-slurm.%N.%j.err # STDERR
```

29. Save the file, and change directory to the 20181105 data directory:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cd /data/picasso/20181105
```

**Part III) Executing the data pipeline**

30. You are now ready to submit the script to the slurm queue:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ sbatch
/data/src/PyHipp/pipeline-slurm.sh
```

31. You can use the squeue function to watch the progress of the job:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ squeue
      JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          2   compute     pipe ec2-user PD       0:00      1 (Priority)
```

The last column will say “(Priority)” when the job is first queued, and switch to “(Resources)” when the compute node is starting up. When the job starts running on the compute node, the last column will now state the address of the compute node, and the ST (status) column will change from PD (pending), to CF (configuring), to R (running):

```
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
                 2   compute     pipe ec2-user  R       0:00      1 ip-10-0-1-245
```

You can also use the following function to track the progress of the job by checking the slurm output file (the number in the name of the .out file corresponds to your Job ID above, so you may have to modify the command below to match your Job ID):

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ tail -f pipe-slurm.*.2.out
```

The tail function will show you the last 10 lines of the file or files you specify, and adding the “-f” argument will cause it to continue to monitor the files, and print out new lines as they are added to the files. This allows you to monitor the progress of the jobs by looking at their outputs. You can type Ctrl-C at any point to get out of the tail function.

32. If you make a mistake, you can cancel jobs by specifying the job number:

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ scancel 2
    ```

    The command above will cancel job #2. You can cancel a range of jobs by doing:

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ scancel {2..7}
    ```

    or all jobs by doing:

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ scancel --user=ec2-user
    ```

33. If it is taking a long time (more than 10 mins) for the jobs to start running, it could be because there are no servers available with the compute nodes you specified. In this case, you can switch to one of the other instance types for your compute nodes that has more than 64 GB or memory, but do take note of the price differences, which will consume your AWS credits at a faster rate.

    If you do not have any jobs running, you can use the following command on your computer to change the instance type of your compute nodes after editing the config file without having to delete and re-create the cluster:

    ```
    (aws) $ pcluster update-compute-fleet --status STOP_REQUESTED --region
    ap-southeast-1 --cluster-name MyCluster01
    (aws) $ pcluster update-cluster --cluster-configuration
    ~/cluster-config.yaml --cluster-name MyCluster01
    ```

    You can find more information about pcluster update from:
    https://docs.aws.amazon.com/parallelcluster/latest/ug/using-pcluster-update-cluster-v3.html

https://docs.aws.amazon.com/parallelcluster/latest/ug/pcluster.update-compute-fleet-v3.html
https://docs.aws.amazon.com/parallelcluster/latest/ug/pcluster.update-cluster-v3.html

34. It will take some time for the job to finish, so you can wait till you receive the email notification that your job has been completed to continue.

**Part IV) Checking the output**

35. Once the job has been completed, you should see the following items in the day directory:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ ls
181105.edf              pipe-slurm.ip-xxx.2.err   sessioneye
mountains               pipe-slurm.ip-xxx.2.out
P11_5.edf               session01
```

You should see the following items in the session01 directory:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ ls session01
181105_Block1.nev       binData.hdf             slist.txt
181105_Block1.ns5       eyelink_24d5.hkl        unity_71bf.hkl
array01                 logs.txt                unityfile_eyelink.csv
array02                 missingData.csv         VirtualMazeBatchLog.txt
array03                 RawData_T1-400
array04                 rplparallel_d41d.hkl
```

In total, we expect the following .hkl files to be created:
session01: rplparallel, unity, eyelink
    8 channel directories: rplraw, rpllfp, rplhighpass
sessioneye: rplparallel, eyelink
    8 channel directories: rplraw, rpllfp, rplhighpass
which adds up to 53. There will also be some spiketrain .hkl files, but the number returned for each channel is not predictable.

So we can do the following to find all the .hkl files:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "*.hkl"
```

The find command is a very useful function that can help find files using different parameters. In this case, we are asking it to start from the current directory (the "." in the second argument) and look for files with names ending in ".hkl".

In order to leave out the spiketrain .hkl files, we can send the output of the previous command to the grep function using the pipe feature ("|") in the shell. The grep function looks for lines containing matching strings (specified by the -e argument) inside a file or from the output of another function sent to it via a pipe. However, when we specify the -v argument, it will instead look for lines that do not contain the matching strings. So in the command above, by specifying "-v -e spiketrain -e mountains", the grep function will only return lines that do not contain "spiketrain" nor "mountains", which will allow us to select only the files returned by the find function that do not contain spiketrain or mountains in their names:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "*.hkl" | grep -v
-e spiketrain -e mountains
```

We can count the number of files by again piping the output above to the wc command to make sure they were all created properly:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "*.hkl" | grep -v
-e spiketrain -e mountains | wc -l
53
```

We can also do the following to make sure that the file sizes look correct:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "*.hkl" | grep -v
-e spiketrain -e mountains | xargs ls -hl
-rw-rw-r-- 1 ec2-user ec2-user 129M Aug 10 17:20
./session01/eyelink_24d5.hkl
-rw-rw-r-- 1 ec2-user ec2-user  61K Aug 10 13:55
./session01/rplparallel_d41d.hkl
-rw-rw-r-- 1 ec2-user ec2-user  12M Aug 10 17:20 unity_71bf.hkl
-rw-rw-r-- 1 ec2-user ec2-user 630M Aug 11 02:28
./session01/array01/channel009/rplhighpass_b59f.hkl
-rw-rw-r-- 1 ec2-user ec2-user  22M Aug 11 02:27
./session01/array01/channel009/rpllfp_6eca.hkl
-rw-rw-r-- 1 ec2-user ec2-user 630M Aug 11 02:18
./session01/array01/channel009/rplraw_d41d.hkl
```

**Include a screenshot of your Terminal window with the file sizes above in your lab report. Make sure you increase the size of your Terminal window so that the size of all 53 files can be captured in the screenshot.**

The xargs function used in the command above takes the output of the grep function, and appends them as arguments to the end of the call to the ls function. For instance, if the grep function returns:

```
$ find . -name "*.hkl" | grep -v -e spiketrain -e mountains
./session01/eyelink_24d5.hkl
./session01/rplparallel_d41d.hkl
```

Using xargs will be the equivalent of:

```
$ ls -hl ./session01/eyelink_24d5.hkl ./session01/rplparallel_d41d.hkl
```

For session01, the rplraw and rplhighpass files are typically over 600 MB, while the rpllfp files are around 20 MB. The unity files are typically around 10 MB, and the eyelink files are typically over 100 MB. The files in the sessioneye directory are typically quite a bit smaller.

In order to check the output of the spike sorting, we would expect one of these files to be created for each channel:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find mountains -name
"firings.mda" | wc -l
8
```

**Include a screenshot of your terminal window with the output above in your lab report.**

36. You can check the time taken to complete the job by looking for the printouts of time in the output file for the job:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ tail pipe-slurm*.out
```

**Include the output of the command above in your lab report, and convert the time taken for the job to hours, minutes, and seconds so it is easy to understand. Extrapolate from the time taken for 8 channels to estimate how long it will take to process all 110 channels.**

**Part V) Wrapping up**

37. Once you are done, you can exit your cluster, and then update your snapshot:

```
(aws) $ update_snapshot.sh data 2
```

38. Once you receive the email notification, you can delete the cluster:

```
(aws) $ pcluster delete-cluster --region ap-southeast-1 --cluster-name
MyCluster01
```

39. In the next lab, we will look at ways to parallelize the data processing.

Submission of Lab Report (in PDF format only, and name the file Lab6_YourName.pdf):

1) Screenshot of the output from Step 22
2) Screenshot of the file sizes described in Step 35
3) Screenshot of the number of spike sorting files described in Step 35
4) Screenshot of the output described in Step 36
5) Time taken for the job to complete
6) Time it will take to process 110 channels
7) Screenshot of your EC2 Dashboard showing no instances running (follow instructions from Lab 4)
8) Screenshot of Elastic Compute Cloud charges (follow instructions from Lab 4)
9) Screenshot of AWS credits (follow instructions from Lab 4)
10) Screenshot of Budgets Overview (follow instructions from Lab 5)