EE3801 2021 L8 Lab (Submission due Nov 3, 11:59 pm)

In this lab, we will look at additional ways to speed up the parallel processing pipelines you created in Lab 7.

The lab consists of seven parts:

1. Update files from GitHub (Estimated time: 15 mins)
2. Just-in-time job submission (Estimated time: 20 mins)
3. Managing resource limitations (Estimated time: 70 mins)
4. Maximizing vCPUs (Estimated time: 20 mins)
5. Checking and resubmitting jobs (Estimated time: 20 mins)
6. Processing all remaining data (Estimated time: 5 mins for setup, approximately 6 hours running time)
7. Wrapping up (Estimated time: 5 mins)

Part I) Update files from GitHub

1. For this lab, you can opt to use the scripts you created in Lab 7 if you did not have any problems. However, we have also added a version of the scripts to the PyHipp repository, so if you would like to use those versions instead, you will need to rename your version so it does not create a conflict when you merge the changes in the upstream repository into your local repository (remember to create a cluster from your EC2 instance and not your computer):

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ mv rplparallel-slurm.sh
   myrplparallel-slurm.sh
   ```

   The same goes for these other scripts you created in Lab 7:

   ```
   rplsplit-slurm.sh                    checkfiles.sh
   rs1-slurm.sh                         removefiles.sh
   rs2-slurm.sh                         checkfiles2.sh
   rs3-slurm.sh                         pipe2.sh
   rs4-slurm.sh
   ```

   You can then merge your repository with the changes in the upstream repository:

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git fetch upstream
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git checkout main
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git merge upstream/main
   ```

   You can add your versions of the scripts to your GitHub repository:

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git add my*.sh
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git commit
   ```

   You can push all the changes to your repository by doing:

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git push
   ```

   You will then be prompted to enter your GitHub username and token again.

Part II) Just-in-time job submission

2. The current setup will require waiting for all the rplraw files in an array to be created before the rpllfp and rplhighpass files can be created, and before the spike sorting can start processing the highpass files. This is also inefficient, as processing on each of the rplraw files should be able to start once they are created.

3. Make a copy of the rs1-slurm.sh script, and name it rs1a-slurm.sh. Modify the RPLSplit function in rs1a-slurm.sh to make use of the option to submit a slurm script after each rplraw file has been created:

   ```
   DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit,
   channel=[*range(1,33)], SkipHPC=False, HPCScriptsDir =
   '/data/src/PyHipp/', SkipLFP=False, SkipHighPass=False, SkipSort=False);
   ```

   With SkipHPC=False and SkipLFP=False, it will look in the HPCScriptsDir directory for a script named rpllfp-slurm.sh, and submit it from the channel directory once the rplraw file is created. This script will create the rpllfp files.

   Similarly, with SkipHPC=False, SkipHighPass=False, and SkipSort=False, it will submit a script named rplhighpass-sort-slurm.sh from the channel directory once the rplraw file is created. This script will create the rplhighpass files, and perform the spike sorting. So the Python command to split the files, and to generate the RPLLFP and RPLHighPass objects, can be shortened to:

   ```
   python -u -c "import PyHipp as pyh; \
   import DataProcessingTools as DPT; \
   import time; \
   import os; \
   t0 = time.time(); \
   print(time.localtime()); \
   DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit,
   channel=[*range(1,33)], SkipHPC=False, HPCScriptsDir =
   '/data/src/PyHipp/', SkipLFP=False, SkipHighPass=False, SkipSort=False);
   \
   print(time.localtime()); \
   print(time.time()-t0);"
   ```

   Remember to also change the name of the job and output files.

4. Do the same for the other three rs?a-slurm.sh scripts.

5. However, the spike sorting requires the highpass files from both sessioneye and session01 to be created first, and should be called only from the session01 directory. The scripts above will start the spike sorting once the highpass files in session01 are created, and before the highpass files in sessioneye have been created. It will also start the spike sorting a second time once the highpass files in sessioneye are created. So we will want to generate the highpass files in one of the two session directories first without calling the spike sorting, and then call spike sorting only after the second set of highpass files have been created. We will choose to process the sessioneye files first as they are much smaller, so you should create another slurm script rse-slurm.sh that will call rplsplit for just the data in sessioneye. Since the sessioneye files are fairly small, we can do this with just one job like this (save this into rse-slurm.sh):

   ```
   python -u -c "import PyHipp as pyh; \
   import time; \
   import os; \
   t0 = time.time(); \
   print(time.localtime()); \
   os.chdir('sessioneye'); \
   pyh.RPLSplit(SkipLFP=False, SkipHighPass=False); \
   print(time.localtime()); \
   print(time.time()-t0);"
   ```

   Since the file sizes are small, you can change “cpus-per-task” back to 1 in this slurm script.

6. In addition, create the scripts rpllfp-slurm.sh and rplhighpass-sort-slurm.sh so they can be used by RPLSplit to submit jobs from the channel directory once the rplraw files are created:

   ```
   python -u -c "import PyHipp as pyh; \
   import time; \
   pyh.RPLLFP(saveLevel=1); \
   print(time.localtime());"

   python -u -c "import PyHipp as pyh; \
   import time; \
   pyh.RPLHighPass(saveLevel=1); \
   from PyHipp import mountain_batch; \
   mountain_batch.mountain_batch(); \
   from PyHipp import export_mountain_cells; \
   export_mountain_cells.export_mountain_cells(); \
   print(time.localtime());"
   ```

   Since there will be a lot of these jobs, you will want to omit the SNS message sent at the end of the job.

   In this setup, the rs?a jobs will take a lot less time as they only submit the slurm scripts once the rplraw files are created, and do not actually need to generate the low-pass, high-pass, and spike sorting files. So, in order to compute how much time the entire pipeline took, we will use the start time of the rplparallel job (since it is the first job submitted), and the end time of the last job, which will likely be a rplhighpass-sort job. In order to be safe, we will print out the end time for both the

rpllfp and rplhighpass-sort jobs.

As the rpllfp-slurm.sh and rplhighpass-sort-slurm.sh scripts will be working off the rplraw files, which are much smaller than the .ns5 files, you will not need more memory than is normally available to each vCPU, so you can use “cpus-per-task=1” for these scripts. Remember to also change the name of the job and output files.

7. Modify the pipe2.sh script (and save it as pipe2a.sh) to call rse-slurm.sh before calling the rs?a-slurm.sh scripts, and have the rs?a jobs depend on the rse job:

```
# second job - no dependencies, called from the day directory
jid2=$(sbatch /data/src/PyHipp/rse-slurm.sh)

# third set of jobs - depends on rse-slurm.sh, called from the day
directory
jid3=$(sbatch --dependency=afterok:${jid2##* }
/data/src/PyHipp/rs1a-slurm.sh)
jid4=$(sbatch --dependency=afterok:${jid2##* }
/data/src/PyHipp/rs2a-slurm.sh)
jid5=$(sbatch --dependency=afterok:${jid2##* }
/data/src/PyHipp/rs3a-slurm.sh)
jid6=$(sbatch --dependency=afterok:${jid2##* }
/data/src/PyHipp/rs4a-slurm.sh)
```

8. You can add your new scripts to your GitHub repository:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git add rs?a-slurm.sh rse-slurm.sh
rpllfp-slurm.sh rplhighpass-sort-slurm.sh pipe2a.sh
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git commit
```

You can push all the changes to your repository by doing:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git push
```

You will then be prompted to enter your GitHub username and token again.

Part III) Managing resource limitations

9. Each instance of MountainSort (the spike sorting software we are using) tries to obtain exclusive access numerous times to a processor_specs.json file in /data/miniconda3/envs/cenv3/etc/mountainlab/database by locking the file, presumably so the file cannot be modified while it is being read. However, when there are a lot of jobs trying to lock the file, they end up having to wait for each other to release the file. This means that we will need to resolve this issue in order to be able perform spike sorting simultaneously on multiple channels properly. One way to get around the file locking issues is to create separate conda environments that each contain a copy of the processor_specs.json file so we can run the spike sorting software in parallel. In order to do this, we will need to create multiple clones of the env1 environment that we have been using. We could create a cloned environment for every spike sorting job (i.e. one

for each of the 110 channels), but since the maximum number of vCPUs we can use at one time is 64, it will be wasteful to create more than 64 clones. So in order to reuse the environments, we will need a way to make sure that each job is using a different environment, and jobs that have finished running can return the environment they used so other jobs can use them. This can be done by creating a file containing a list of available environments that jobs can access at the beginning to find an available environment, remove that environment from the list so other jobs cannot use the same environment, and add the environment back to the list when the sorting is done. In order to prevent the file from being changed while it is being read, this method will require creating a lock on the file containing the list of environments temporarily so only one job can modify it at a time.

10. Use the nano editor to modify the envlist.py file in the PyHipp directory to complete the code (look for the lines with #add code here) to allow it to create a list of environment names (cenv0, cenv1, cenv2, … cenv63) by doing (this command is for illustration only, so do not run the command on your cluster yet):

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ /data/src/PyHipp/envlist.py cenv
    64
    ```

    The list will essentially look like:

    ```
    cenv0
    cenv1
    cenv2
    ...
    cenv63
    ```

    This list will be saved into /data/picasso/envlist.hkl using the hickle module.

    When you want to use an environment, you should be able to do (again, these commands are meant to explain what we are trying to do, so do not run them yet):

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ /data/src/PyHipp/envlist.py
    ```

    which should return:

    ```
    cenv0
    ```

    In this example, the first item on the list will be removed from the list and returned. The next time the function is called again in this form, the environment name “cenv1” will be removed from the list and returned. This makes sure that each environment will only be used by one job at a time.

    When the spike sorting is completed, the environment can be added back to the list by doing (as above, do not run this command yet):

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ /data/src/PyHipp/envlist.py
    cenv0
    ```

This will append cenv0 to the end of the list so it can be used by another job.

11. Once you have completed modifying the code, you will need to install the FileLock module on your cluster before you can test the code:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ pip install filelock
```

12. After testing your code by first creating the list of environments, requesting an environment, and then returning that environment (the commands described above in Step 10), you can load the envlist.hkl file in iPython to check the list contained in the file:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ ipython
In[ ]: import hickle
In[ ]: clist = hickle.load('/data/picasso/envlist.hkl')
In[ ]: len(clist)
In[ ]: clist
```

Include your completed envlist.py script, screenshots of the terminal window where you ran the three ways to call your script, and the resulting list in the envlist.hkl file in your lab report.

13. Add envlist.py to your repository and push the change to GitHub again.

14. We will now create multiple clones of the env1 environment using a while-loop in bash that will go from x=0 to x=63 (replace the highlighted digits below with your AWS account ID):

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ x=0; while [ $x -le 63 ]; do
echo $x; conda create --name cenv$x --clone env1 --copy; (( x++ )); done;
aws sns publish --topic-arn
arn:aws:sns:ap-southeast-1:123456789012:awsnotify --message
"CondaCreateComplete"
```

We need to add the --copy argument since by default conda creates clones of environments by creating hard links to the files in the original environment, which will again create conflicts when multiple MountainSort jobs try to obtain exclusive access to the same file.

You might get a warning that the ml_ms4alg package is corrupted, but you can safely ignore the warning.

This will take a while, so we have prepared a snapshot (snap-0e4cf11324a17c5e3) with all the environments created for you. You can type Ctrl-c a couple of times to terminate the command above. You can then delete your current cluster, make a copy of the snapshot we prepared using the instructions in Lab 4 (you can name your snapshot “condaenvs”), copy the new snapshot ID to your clipboard, replace the SnapshotId in the cluster config file on your EC2 instance with the one in the clipboard:

```
SnapshotId: snap-xxxx
```

and create a new cluster with that snapshot. You will have to copy your AWS credentials and your key pair from your EC2 instance to the /data directory again, as well as clone your PyHipp repository to /data/src/PyHipp again.

Once you are done, you can check that the environments were created properly by doing:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ conda env list
```

You should see a list of 66 environments:

```
# conda environments:
#
base                     /data/miniconda3
cenv0                    /data/miniconda3/envs/cenv0
cenv1                    /data/miniconda3/envs/cenv1
cenv10                   /data/miniconda3/envs/cenv10
...
```

Run the envlist.py program to create the files /data/picasso/envlist.hkl and /data/picasso/envlist.hkl.lock:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ /data/src/PyHipp/envlist.py cenv
64
```

15. Now we will need to modify rplhighpass-sort-slurm.sh to make use of the separate conda environments by adding these lines before the python command:

    ```
    /data/miniconda3/bin/conda init
    source ~/.bashrc
    envarg=`/data/src/PyHipp/envlist.py`
    conda activate $envarg
    ```

    As well as the lines below after the python command:

    ```
    conda deactivate
    /data/src/PyHipp/envlist.py $envarg
    ```

16. Push your changes again to GitHub.

Part IV) Maximizing vCPUs

17. In this new pipeline, the number of jobs have greatly increased, so we will want to optimize the pcluster configuration to maximize the number of jobs you can run. Currently, we are using compute nodes that have eight vCPUs. Along with the two vCPUs we have in the master node and the EC2 instance, we can typically only have seven compute nodes before we exceed the 64 vCPUs limit on AWS. That means that we are only using 60 out of the possible 64 vCPUs. In order to make use of the remaining four vCPUs, we can add a compute node with four vCPUs to the same queue. We can do this by modifying the following lines in the cluster config file on your EC2 instance to add a .xlarge instance instead of .2xlarge, e.g. if you are using r5.2xlarge:

```
Scheduling:
  Scheduler: slurm
  SlurmQueues:
  - Name: queue1
    ComputeResources:
    - Name: r5-2xlarge
      InstanceType: r5.2xlarge
      MinCount: 0
      MaxCount: 10
    - Name: r5a-xlarge
      InstanceType: r5a.xlarge
      MinCount: 0
      MaxCount: 1
```

If you are using r5a.2xlarge, you should add r5a.xlarge. Limiting the .xlarge instances to one node makes sure that the bulk of the queue will be made up of .2xlarge compute nodes.

18. You can stop your current compute nodes, and update your cluster by doing:

```
[ec2-user@ip-54.169.221.196 ~]$ pcluster update-compute-fleet --status
STOP_REQUESTED --region ap-southeast-1 --cluster-name MyCluster01
[ec2-user@ip-54.169.221.196 ~]$ pcluster update-cluster
--cluster-configuration ~/cluster-config.yaml --cluster-name MyCluster01
```

If you get something like:

```
{
  "message": "Update failure",
  "updateValidationErrors": [
    {
      "parameter": "Scheduling.SlurmQueues[queue1].ComputeResources",
      "requestedValue": {
        "Name": "r5a-xlarge",
        "InstanceType": "r5a.xlarge",
        "MinCount": 0,
        "MaxCount": 1
      },
      "message": "All compute nodes must be stopped. Stop the compute
fleet with the pcluster update-compute-fleet command"
    }
```

You might have to enter the following command to check on the status of your command to stop the compute nodes:

```
[ec2-user@ip-54.169.221.196 ~]$ pcluster describe-cluster --region
ap-southeast-1 --cluster-name MyCluster01
```

Look for the following message:

```
  "computeFleetStatus": "STOPPED",
```

At that point, you should be able to redo the “update-cluster” command again.

19. You are now ready to submit the jobs. Remember to remove the files created during the last run before running pipe2a.sh from the 20181105 directory.

20. Check that the jobs were submitted successfully, and then submit the consol_jobs.sh script with a dependence on the six jobs submitted by the pipe2a.sh script (replace the job numbers with those in your queue):

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ sbatch
--dependency=afterok:11:12:13:14:15:16 /data/src/PyHipp/consol_jobs.sh
```

Remember to update the Instance ID in your Lambda function with the Instance ID of your head node so the head node can be terminated once the jobs are done.

21. You should receive a notification when the first compute node starts up, which will run the rplparallel and rse jobs. When the rse job finishes, you should get another notification, followed by notifications when four additional compute nodes get started to run the rs?a jobs. Once several rpllfp and rplhighpass-sort jobs get submitted, you will receive notifications that three additional compute nodes have started up for a total of 8 compute nodes totaling 60 vCPUs.

22. You can use the following command to keep track of what is happening with the various jobs:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ tail -f rs?a*slurm*out
```

or (replace rplhps below with the prefix for your slurm output and error files):

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ find session01 -name
"rplhps*slurm*out" | xargs tail -f
```

Once all eight compute notes have started up and running jobs, you can check your queue with the following command to see that you have instances with both 4 vCPUs and 8 vCPUs running that add up to 60 vCPUs:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ squeue | grep R
```

These two commands will find all the lines containing “R” (indicating jobs that are running under the ST column in the output of squeue). You can use the following command to count the number of jobs running, but since the previous command will include two header rows, you will have to subtract two from the output:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ squeue | grep R | wc -l
```

23. If you get “DependencyNeverSatisfied” for some of your jobs, check the error files for your earlier jobs to see if there were any errors (e.g. if you see “You must specify a region…” that probably means that you forgot to copy over your AWS credentials). In order to get proper time measurements for this setup, you will have to delete the remaining queued jobs, remove the files created, and re-submit pipe2a.sh.

24. You can take a break here and wait till you receive the email notification that the snapshot has been completed to continue.

Part V) Checking and resubmitting jobs

25. Once you receive the notification that the snapshot has been completed, remember to run the "pcluster delete-cluster …" command to remove "MyCluster01" from the list of clusters. You can also delete the "condaenvs" snapshot from your AWS Management Console to reduce your storage charges. You can then create a new cluster to compute the time taken to process the files. You might want to use the "pcluster describe-cluster …" command to wait for the delete-cluster to be done before you create another cluster named "MyCluster01" to avoid having to modify the cluster name in the ec2snapshot.sh script.

26. Check the output using your checkfiles2.sh script. In order to compute the time taken properly, you will need to get the start time for the rplparallel job (since it is the first job that is submitted), and the end time of the last rpllfp or rplhighpass-sort job. You can get the former from the output of checkfiles2.sh, but for the latter, you will need to do the following (the "-or" argument for the find function allows you to find files in the session01 directory that match either of the name patterns that you used to name the output files):

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ find session01 -name "hps*out"
    -or -name "lfp*out" | xargs tail -n 1 | sort
    ```

    which should give you something like:

    ```
    time.struct_time(tm_year=2020, tm_mon=10, tm_mday=7, tm_hour=5, tm_min=8,
    tm_sec=6, tm_wday=2, tm_yday=281, tm_isdst=0)
    time.struct_time(tm_year=2020, tm_mon=10, tm_mday=7, tm_hour=6,
    tm_min=23, tm_sec=10, tm_wday=2, tm_yday=281, tm_isdst=0)
    time.struct_time(tm_year=2020, tm_mon=10, tm_mday=7, tm_hour=6,
    tm_min=23, tm_sec=6, tm_wday=2, tm_yday=281, tm_isdst=0)
    ```

    The end times will be sorted by alphabetical order and not numerical order, so you might need to scroll up to make sure you find the latest timestamp. **Include the first five timestamps and the last five timestamps of the output above, and compute the total time taken in your lab report.**

27. If for some reason some of the jobs failed, you can do the following to re-submit those jobs. Since the bulk of the jobs are being called from the channel directory, we will first create a list of channels:

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "channel*" | grep
    -v -e eye -e mountain | sort > chs.txt
    ```

    This command is similar to the one we used in Step 35 in Lab 6 except we are now sorting the output before saving it into a file named "chs.txt' by using the ">" function.

28. Next, we will create a list of channels that already have the appropriate files generated. For instance, if we are looking to generate rplhighpass files for certain channels, we can find the channels that already have rplhighpass files:

    ```
    (env1) [ec2-user@ip-10-0-5-43 20181105]$ find . -name "rplhighpass*hkl" |
    grep -v -e eye | sort | cut -d "/" -f 1-4 > hps.txt
    ```

The cut command allows us to print selected fields that are separated by the delimiter character specified by the -d option, in this case "/". You can check what each of the commands do by removing the other commands. You can use a similar command to find the channels that successfully completed the spike sorting by looking for "firings.mda" files.

29. Now that we have a list of all channels, and a list of channels with rplhighpass files, we can find the channels that do not have rplhighpass files by doing:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ comm -23 chs.txt hps.txt
```

The comm command returns lines that are found only in the first file in the first column, lines that are found in the second file in the second column, and lines that are common to both files in the third column. By specify "-23", you can suppress the second and third columns, which will return just the lines that are found only in the first file, which in this case will correspond to the channel directories that are missing the rplhighpass file.

30. You can now re-run the rplhighpass-sort-slurm.sh script (or whichever script or command you need to run) by doing:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ cwd=`pwd`; for i in `comm -23
chs.txt hps.txt`; do echo $i; cd $i; sbatch
/data/src/PyHipp/rplhighpass-sort-slurm.sh; cd $cwd; done
```

The command above saves the current directory into a shell variable named cwd, and then uses a for-loop to iterate over the directories returned by the "comm" function above. In each directory, it submits a slurm job to process the data in that directory, before changing directories back to the initial directory (saved in the "cwd" variable), and continuing with the next step of the for-loop.

[Optional: You can also create and use a sort-slurm.sh script above (instead of the rplhighpass-sort-slurm.sh script) that just reruns the spike sorting if the highpass files already exist, and only the spike sorting files are missing.]

Include the output of checkfiles.sh showing the correct number of output files in your lab report.

Part VI) Processing all remaining data

31. You can now process the data from the other days with neural data by doing:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ cd ..
(env1) [ec2-user@ip-10-0-5-43 picasso]$ for i in 2018110[12]; do echo $i;
cd $i; bash /data/src/PyHipp/pipe2a.sh; cd ..; done
```

This series of commands will search for directories 20181101 and 20181102, iterate over each directory by first printing the directory name, changing to that directory, calling the pipe2a.sh script, going up one directory, before repeating.

32. For the other days in the picasso directory, we only need to run the rplparallel-slurm.sh script as they do not contain neural data. However, to avoid receiving notification emails for each of the 42 days, add a "# " to the start of the "aws sns …" command to comment the command out. You can then submit the jobs by doing:

```
(env1) [ec2-user@ip-10-0-5-43 picasso]$ for i in 20180??? 201810??; do
echo $i; cd $i; sbatch /data/src/PyHipp/rplparallel-slurm.sh; cd ..; done
```

33. We will also need to submit the script to take a snapshot once all the jobs are done. In order to do that, we will need to find the jobs that will spawn more jobs (i.e. rplsplit jobs), and wait for them to finish before we submit the job that will wait for all the jobs left in the queue to be completed before initiating the snapshot:

```
(env1) [ec2-user@ip-10-0-5-43 picasso]$ squeue | grep -e rs1a -e rs2a -e
rs3a -e rs4a
920   compute      rs1a ec2-user PD       0:00      1 (Resources)
921   compute      rs2a ec2-user PD       0:00      1 (Resources)
922   compute      rs3a ec2-user PD       0:00      1 (Resources)
923   compute      rs4a ec2-user PD       0:00      1 (Resources)
924   compute      rs1a ec2-user PD       0:00      1 (Resources)
925   compute      rs2a ec2-user PD       0:00      1 (Resources)
926   compute      rs3a ec2-user PD       0:00      1 (Resources)
927   compute      rs4a ec2-user PD       0:00      1 (Resources)
(env1) [ec2-user@ip-10-0-5-43 picasso]$ sbatch
--dependency=afterok:4:5:6:7:10:11:12:13 /data/src/PyHipp/consol_jobs.sh
```

We do not need to add the rplparallel-slurm.sh jobs as they will be captured by the consol_jobs.sh script when the consol_jobs.sh script runs after the rplsplit jobs are done.

34. The last thing to do is to replace the instance IDs in your Lambda function with those of your cluster's head node.

Part VII) Wrapping up

35. When all the jobs are completed, delete the cluster from the list, start up another cluster to check that all the files for the other days were generated properly. If not, make sure you submit any additional jobs to complete the processing. **Include screenshots showing the correct number of files for 20181101 and 20181102. In addition, include a screenshot showing the total number of hickle files for the days without neural data (refer to Step 35 in Lab 6 for the expected number per day) in your lab report.**

36. You can now update the snapshot from your EC2 instance, and shut down the cluster.

37. In this lab, you have gone through some of the basics of optimizing a data processing pipeline. In the next lab, you will learn about high-throughput data visualization.

Submission of Lab Report (in PDF format only, and name the file Lab7_YourName.pdf):

1) Screenshot of your envlist.py from Step 12
2) Screenshots of the 3 different types of running envlist.py, and the list created from Step 12

3) Screenshot of the timestamps and total time taken from Step 26
4) Screenshot showing the correct number of files from Step 30
5) Screenshot showing the correct number of files for 20181101 from Step 35
6) Screenshot showing the correct number of files for 20181102 from Step 35
7) Screenshot showing the correct number of files for the remaining days from Step 35
8) Screenshot of your EC2 Dashboard (follow instructions from Lab 4)
9) Screenshot of Elastic Compute Cloud charges (follow instructions from Lab 4)
10) Screenshot of Budgets Overview (follow instructions from Lab 5)