EE3801 2021 L7 Lab (Submission due Oct 27, 11:59 pm)

In the previous lab, we created a serial processing pipeline for processing the data. In this lab, we will create two parallel processing pipelines to speed up processing of the data.

The lab consists of eight parts:

1. Update files from GitHub (Estimated time: 20 mins)
2. Setting up a parallel processing pipeline (Estimated time: 10 mins)
3. Creating bash scripts for checking outputs and deleting files (Estimated time: 5 mins)
4. Executing the parallel processing pipeline (Estimated time: 15 mins)
5. Parallelizing RPLSplit (Estimated time: 30 mins)
6. Automatically deleting the head node (Estimated time: 45 mins)
7. Executing the parallel processing pipeline (Estimated time: 5 mins)
8. Wrapping up (Estimated time: 5 mins)

Part I) Update files from GitHub

1. We will first push to GitHub the script you created in Lab 6. You will need to first create a cluster, ssh to the cluster, and activate the env1 environment. After that you can do:

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

   Create a token to use with GitHub:

   https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

   You can push the file to GitHub by doing:

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git config --global core.editor
   "nano"
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git add pipeline-slurm.sh
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git commit
   ```

   At this point, a nano editor window will open up for you to add a commit message. It is good practice to add a concise description of what changes were made in the nano window so in the future, you can look back and understand what was done. Once you are done, you can exit nano the usual way, making sure to save the file with the suggested filename. After that, you can push the scripts to your GitHub 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.

   If you accidentally cloned the wrong repository in Lab 6 (i.e. shihchengyen/PyHipp instead of your_username/PyHipp), you can reset it by doing:

   ```
   (env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git remote set-url origin
   https://github.com/yourusername/PyHipp.git
   ```

   You can check if it worked by doing:

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

which should return:

```
origin      https://github.com/yourusername/PyHipp.git (fetch)
origin      https://github.com/yourusername/PyHipp.git (push)
upstream    https://github.com/shihchengyen/PyHipp.git (fetch)
upstream    https://github.com/shihchengyen/PyHipp.git (push)
```

If the upstream repository is not set, you can add it by doing:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ git remote add upstream
https://github.com/shihchengyen/PyHipp.git
```

Repeat the command above to check that the upstream repository is now set properly.

2. Now you are ready to merge with the upstream changes:

```
(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
```

This should update a file named “consol_jobs.sh” and add a file named “ec2snapshot.sh”. You can push the merged changes to your forked 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.

3. Update your snapshot using another Terminal window so the changes are saved for use later in this lab.

Part II) Setting up a parallel processing pipeline

4. In Lab 6, we created a serial processing script to process the data. If you have multiple days of data to process, you will be able to use the script to process data from each day in parallel on a cluster. However, if you are processing only one day of data, the time it will take will not be very different from running on your computer. So in this lab, we will look at how we can increase the granularity of the parallelization so we can speed up the processing for even one day of data.

5. Looking at the serial processing pipeline, we notice that there are two groups of objects: 1) RPLParallel object, and all the objects that depend on it: Unity, Eyelink, aligning_objects, and raycasting; and 2) RPLRaw objects, and all the objects that depend on it: RPLLFP, RPLHighPass, and spike sorting. This means that these scripts can be run in parallel. So create a cluster, login to your cluster, copy your AWS credentials to your home directory, initialize conda, activate the env1 conda environment, change directory to /data/src/PyHipp, and edit a copy of the slurm script you created in Lab 6:

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

6. We will now remove the commands to create the second group of objects and change the notification message to distinguish it from the second script we will be creating:

```
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.Unity, saveLevel=1); \
pyh.EDFSplit(); \
os.chdir('session01'); \
pyh.aligning_objects(); \
pyh.raycast(1); \
print(time.localtime()); \
print(time.time()-t0);"

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

Remember to also edit the following lines in the file to make the job name and the slurm output files more distinct:

```
#SBATCH -J "rplpl"   # job name
#SBATCH -o rplpl-slurm.%N.%j.out # STDOUT
#SBATCH -e rplpl-slurm.%N.%j.err # STDERR
```

7. Next, we will create a second copy of pipeline-slurm.sh, name it rplsplit-slurm.sh, and remove the commands to create the first group of objects:

```
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.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);
\
```

```
os.chdir('session01'); \
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
"RPLSplitJobDone"
```

Remember to also edit the following lines in the file to make the job name and the slurm output files more distinct:

```
#SBATCH -J "rplspl"   # job name
#SBATCH -o rplspl-slurm.%N.%j.out # STDOUT
#SBATCH -e rplspl-slurm.%N.%j.err # STDERR
```

Part III) Creating bash scripts for checking outputs and deleting files

8. In order to make it easier to check the outputs of your jobs, we will write a shell script to perform the commands from Step 35 in Lab 6:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ nano checkfiles.sh
```

Copy and paste the following into the script:

```
#!/bin/bash

echo "Number of hkl files"
find . -name "*.hkl" | grep -v -e spiketrain -e mountains | wc -l

echo "Number of mda files"
find mountains -name "firings.mda" | wc -l

echo "Time taken (s)"
tail pipe-slurm*.out
```

9. Run your shell script from the /data/picasso/20181105 directory to make sure you are able to get the same output as in the last lab:

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

10. We will also create a shell script to remove the files created during the last run:

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

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ nano removefiles.sh
```

Copy and paste the following into the script:

```
#!/bin/bash

find . -name "*.hkl" -or -name "*.csv" | xargs rm

find . -name "*slurm*err" -or -name "*slurm*out" | xargs rm

rm -r mountains

cd session01

rm binData.hdf logs.txt slist.txt VirtualMaze*

cd ..
```

11. Use the script to remove the files created during the last run:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cd -
(env1) [ec2-user@ip-10-0-5-43 20181105]$ bash
/data/src/PyHipp/removefiles.sh
```

Run the checkfiles shell script again to check that the files have been removed.

Part IV) Executing the parallel processing pipeline

12. We can now submit both jobs from the day directory:

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

13. Once the jobs start running, you can check what is happening in real time by using the following command:

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

which will display new lines as they appear in the .out files.

14. While waiting for the emails notifying you that the jobs have finished, you can skip to the next step. Once the jobs have finished, make a copy of the checkfiles.sh shell script named checkfiles2.sh, and modify it to produce the following output consisting of the start and end times of each job (highlighted below):

```
(env1) [ec2-user@ip-10-0-14-97 20181105]$ bash
/data/src/PyHipp/checkfiles2.sh
```

```
Number of hkl files
53
Number of mda files
8
Start Times
==> rplpl-slurm.queue1-dy-m54xlarge-1.4.out <==
time.struct_time(tm_year=2021, tm_mon=10, tm_mday=15, tm_hour=14,
tm_min=45, tm_sec=44, tm_wday=4, tm_yday=288, tm_isdst=0)

==> rplspl-slurm.queue1-dy-m54xlarge-1.5.out <==
time.struct_time(tm_year=2021, tm_mon=10, tm_mday=15, tm_hour=14,
tm_min=45, tm_sec=44, tm_wday=4, tm_yday=288, tm_isdst=0)
End Times
==> rplpl-slurm.queue1-dy-m54xlarge-1.4.out <==
time.struct_time(tm_year=2021, tm_mon=10, tm_mday=15, tm_hour=15,
tm_min=16, tm_sec=6, tm_wday=4, tm_yday=288, tm_isdst=0)
1821.3657758235931
{
    "MessageId": "ebc92db5-3ca8-5f1d-9760-7456a1df4587"
}

==> rplspl-slurm.queue1-dy-m54xlarge-1.5.out <==
time.struct_time(tm_year=2021, tm_mon=10, tm_mday=15, tm_hour=15,
tm_min=36, tm_sec=16, tm_wday=4, tm_yday=288, tm_isdst=0)
3031.305139541626
{
    "MessageId": "e02b2fcc-4606-51de-9dd1-762fe4377c35"
}
```

You can now add the time taken for each job to compute the total time it would have taken to run the two jobs serially. In order to compute the actual time taken for both jobs to complete, take the difference between the earlier of the two start times and the later of the two end times. Take the difference between the two durations to compute how much time the parallel processing saved. Compute also how much time it would have taken to process 110 channels of data. When extrapolating the time taken for 110 channels, remember to extrapolate only the time taken for the second job since the first job will only be run once regardless of the number of channels processed.

Include in your lab report a screenshot of your checkfiles2.sh script and its output. Include also a) the sum of the two job durations in hours, minutes, and seconds, b) the actual time taken for both jobs to complete, c) the time saved in parallelizing the jobs, and d) the time it will take to process all 110 channels.

Part V) Parallelizing RPLSplit

15. In the previous setup, we used a single job to split the channels in the .ns5 file serially. One way to speed up the processing will be to use multiple jobs to process different groups of channels so the channels can be split in parallel. One way to do this is to process the channels by arrays of 32 channels.

16. Make a copy of rplsplit-slurm.sh, and edit it using nano:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ cd -
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cp rplsplit-slurm.sh rs1-slurm.sh
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ nano rs1-slurm.sh
```

17. Modify the commands in rs1-slurm.sh to the following:

```
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit,
channel=[*range(1,33)]);
```

    If called from the day directory, this will make RPLSplit split and save any channels in the range of 1 to 32 that it finds in the .ns5 file in both the sessioneye and session01 directories.

18. Once the rplraw files have been created in the channel directories, we would want to convert the files in the first array to rpllfp and rplhighpass files, so we will need to add the following commands:

```
DPT.objects.processDirs(dirs=['sessioneye/array01','session01/array01'],
cmd='import PyHipp as pyh; import DataProcessingTools as DPT;
DPT.objects.processDirs(None, pyh.RPLLFP, saveLevel=1);
DPT.objects.processDirs(None, pyh.RPLHighPass, saveLevel=1);');
```

    Note that specifying “dirs=['sessioneye/array01','session01/array01']” is yet another way to use the processDirs function, which specifies the directories you want to run a particular command, or create an object. Specifying the array01 directories in this way ensures that this slurm job only processes the channels in the first array directory, and not the channels in the other array directories, which we would want other jobs to handle.

19. The spike sorting script will combine the highpass files in both the sessioneye and session01 directories before starting the spike sorting process, but again we need to call it from the session01/array01 directory to process only the channels in the first array:

```
os.chdir('session01/array01'); 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();');
```

20. The python command in the modified file should look like:

```
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.RPLSplit,
channel=[*range(1,33)]); \
DPT.objects.processDirs(dirs=['sessioneye/array01','session01/array01'],
cmd='import PyHipp as pyh; import DataProcessingTools as DPT;
DPT.objects.processDirs(None, pyh.RPLLFP, saveLevel=1);
DPT.objects.processDirs(None, pyh.RPLHighPass, saveLevel=1);'); \
os.chdir('session01/array01'); \
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);"
```

Remember to also edit the following lines in the file to make the job name and the slurm output files more distinct:

```
#SBATCH -J "rs1"   # job name
#SBATCH -o rs1-slurm.%N.%j.out # STDOUT
#SBATCH -e rs1-slurm.%N.%j.err # STDERR
```

Change the AWS SNS notification message so that it is unique to this script.

21. Create scripts to process the other three arrays where the array directory and channel ranges should be:

```
array02 channel=[*range(33,65)]
array03 channel=[*range(65,97)]
array04 channel=[*range(97,125)]
```

22. Instead of having to submit these five scripts manually, we can create a script to submit the jobs:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ nano pipe2.sh
```

Enter the following into the file:

```
#!/bin/bash

# first job called from the day directory
# creates RPLParallel, Unity, and EDFSplit objects, and
# calls aligning_objects and raycast
sbatch /data/src/PyHipp/rplparallel-slurm.sh

# second set of jobs called from the day directory
sbatch /data/src/PyHipp/rs1-slurm.sh
```

```
sbatch /data/src/PyHipp/rs2-slurm.sh
sbatch /data/src/PyHipp/rs3-slurm.sh
sbatch /data/src/PyHipp/rs4-slurm.sh
```

23. Remove the files created during the last run in Lab 6:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp]$ cd -
(env1) [ec2-user@ip-10-0-5-43 20181105]$ bash
/data/src/PyHipp/removefiles.sh
```

24. Run pipe2.sh:

```
(env1) [ec2-user@ip-10-0-5-43 20181105]$ bash /data/src/PyHipp/pipe2.sh
```

25. Check the queue to see how the jobs have been assigned to the compute nodes (it might take some time for the compute node to start up and for the ST (status) column to change from CF (configuring) to R (running)):

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    2   queue1    rplpl ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    3   queue1      rs1 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    4   queue1      rs2 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    5   queue1      rs3 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
    6   queue1      rs4 ec2-user  R       0:02      1 queue1-dy-r52xlarge-1
```

26. As you can see, all five jobs are running on one compute node (queue1-dy-r52xlarge-1 in this example but your instance type will depend on what you specified in your cluster-config.yaml file). This is not ideal as each of the rplsplit jobs will be reading in the .ns5 files, and will require approximately 40 GB of memory. The compute node only has 64 GB of memory, so we can either choose a larger instance type with more memory, or we can modify the slurm script to create more compute nodes so that each of the rplsplit jobs can be run on one compute node. We will now do the latter.

27. You can go ahead and cancel the jobs (replace the job ids below with those from your queue):

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

28. Remove the files created during the last run.

29. Edit each of the rs?-slurm.sh scripts to add the following line (under the line #SBATCH --nodes=1) if you are using a 2xlarge instance type with 8 vCPUs for your compute node (e.g. z1d.2xlarge, r5dn.2xlarge, etc.):

```
#SBATCH --cpus-per-task=5   # number of CPUs for this task
```

and the following line if you are using a 4xlarge instance type with 16 vCPUs (e.g. m5.4xlarge, m5a.4xlarge, etc.):

```
#SBATCH --cpus-per-task=10   # number of CPUs for this task
```

If a compute node instance type we are using has 8 vCPUs (e.g. z1d.2xlarge), once one of the compute nodes is assigned one of the rplsplit jobs that requires 5 vCPUs, there will only be 3 vCPUs left. This will be insufficient to assign another rplsplit job, so another compute node will be started up to run the next job. This will allow all 64 GB of memory in a node to be dedicated to one job instead of being shared between multiple jobs. Using cpus-per-task=5 instead of cpus-per-task=8 allows other jobs that do not require a lot of memory (e.g. rplparallel-slurm.sh) to run concurrently on one of the compute nodes.

30. Re-run pipe2.sh.

31. Check the queue to see how the jobs have been assigned to the compute nodes (it might take some time for the compute nodes to start up):

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
    7    queue1    rplpl ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
    8    queue1      rs1 ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
    9    queue1      rs2 ec2-user  R       1:25      1 queue1-dy-r52xlarge-2
   10    queue1      rs3 ec2-user  R       1:25      1 queue1-dy-r52xlarge-3
   11    queue1      rs4 ec2-user  R       1:25      1 queue1-dy-r52xlarge-4
```

32. As you can see, the rs1, rs2, rs3, and rs4 jobs are now running on different compute nodes (queue1-dy-r52xlarge-1, queue1-dy-r52xlarge-2, queue1-dy-r52xlarge-3, queue1-dy-r52xlarge-4). Increasing the memory requirement by increasing the number of vCPUs required allowed the jobs to run successfully in parallel.

33. You can now cancel the jobs. Update your snapshot using another Terminal window so the changes are saved for use later in this lab.

Part VI) Automatically deleting the head node

34. Before we start to process all the channels, we need to do one more thing. Although the compute nodes will be terminated when all the jobs are completed, the head node will continue to run, which will consume your AWS credits. So we are going to modify your setup that will allow you to cleanly terminate your head node once all the jobs are completed. This will involve using a small EC2 instance that can be supported by your AWS free-tier credits to initiate a snapshot of your cluster, and then use AWS Lambda to shut down your cluster once the snapshot has been completed. You can find more about AWS Lambda, which allows you to write code to interface with your AWS services, at:

    https://aws.amazon.com/lambda/

35. First, open up a new terminal window on your computer, activate your aws environment, and create two new clusters named “testcluster1” and “testcluster2”.

36. Next, **change directories to the directory on your computer that contains the file MyKeyPair.pem**. Then, do the following (which is similar to what you did in Lab 3) to create a small EC2 instance:

```
(aws) $ aws ec2 run-instances --image-id
resolve:ssm:/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2
--instance-type t2.micro --key-name MyKeyPair
```

This will return something that looks like this:

```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0cd31be676780afa7",
            "InstanceId": "i-047ed6d17f231355d",
            "InstanceType": "t2.micro",
            "KeyName": "MyKeyPair",
            "LaunchTime": "2020-08-09T14:17:12+00:00",
            "Monitoring": {
                "State": "disabled"
            },
```

Take note of the InstanceId, as highlighted above. You can hit the space bar to scroll through the rest of the information returned by the function, and then hit “q” when you get to the end, or you can hit “q” at any time to get out of the remainder of the display.

37. While waiting for your clusters and EC2 instance to be set up, you can set up your security settings to enable Lambda. Open the AWS Management Console in your web browser, type “IAM” in the “Services” search bar, click on “Policy” in the sidebar, click the “Create Policy” button, click the “JSON” tab, and copy and paste the code from the “Create an IAM policy and execution role for your Lambda function” section in the following page:

    https://aws.amazon.com/premiumsupport/knowledge-center/start-stop-lambda-cloudwatch/

    Add in “ec2:Terminate*” under “Action”:

    ```
    "ec2:Start*",
    "ec2:Stop*",
    "ec2:Terminate*"
    ```

    Click the “Next: Tags” button in the bottom-right, click the “Next: Review” button on the next page, enter “LambdaPolicy” in the “Name” textfield, and then click the “Create policy” button.

38. Next, click on “Roles” in the sidebar, click on the “Create role” button, select “ Lambda” under “Common use cases”, click the “Next: Permissions” button, type “LambdaPolicy” in the search bar, scroll down to locate “LambdaPolicy”, click the checkbox for “LambdaPolicy”, click the “Next: Tags” button, enter “LambdaRole” in “Role name”, and then click the “Create role” button.

39. We now need to find the InstanceID for the head node of the first test cluster we just created. You can find this by typing “EC2” in the “Services” search bar, and click on “Instances” in the sidebar. Scroll to the right to find the cluster with “testcluster1” in its entry under the “Security group name” column. Scroll back to the left to click on the checkbox beside that instance, and click on the icon beside the Instance ID in the lower half of the window to copy it to your clipboard.

    Your small EC2 instance should be up and running now as well, so note down its Public IPv4 address as you will be using this EC2 instance for the rest of the module. You should also note down the InstanceIDs and the Public IPv4 addresses for the head node of the second test cluster, as well as the head node of your first cluster, i.e. “MyCluster01”.

40. Now we will create an AWS Lambda function that will shut down the head node. You can type “Lambda” in the “Services” search bar, click on “Functions” in the sidebar, click on the “Create function” button, and follow the instructions in the “Create Lambda functions that stop and start your EC2 instances” section in the document above. However, change your region to “ap-southeast-1”, replace the example EC2 instance IDs with the ID you copied above, and replace the following code:

    ```
    ec2.stop_instances(InstanceIds=instances)
    print('stopped your instances: ' + str(instances))
    ```

    with:

    ```
    ec2.terminate_instances(InstanceIds=instances)
    print('terminated your instances: ' + str(instances))
    ```

    You want to terminate instances instead of stopping them as you will still be charged for the volumes attached to instances that are stopped. Click the “Deploy” button to save your function.

    Click the “Test” button to check that you were able to terminate the head node of the first cluster. You should see messages in the “Execution Results” tab that reflect the outcome of the function.

41. Although terminating the head node stops you from being charged, the cluster still exists as you can see with the following command:

    ```
    (aws) $ pcluster list-clusters --region ap-southeast-1
    ```

    This will prevent you from creating another cluster in the future with the same name. So you will still need to delete the cluster by doing:

    ```
    (aws) $ pcluster delete-cluster --region ap-southeast-1 --cluster-name
    testcluster1
    ```

42. Replace the Instance ID in the “instances” variable in the “lambda_function” tab with that of the second test cluster. Click the “Deploy” button to save your changes.

43. Now you can use CloudWatch to set up a new rule that will execute your Lambda function when the snapshot is completed. Follow the instructions in Step 19 in Lab4 to select select “EC2” for “Service Name”, “EBS Snapshot Notification” for “Event Type”, “createSnapshot” for “Specific event(s)”, and “succeeded” for “Specific result(s)” (as shown below). Under the “Target” section, select “Lambda function”, followed by the name of the function you created under “Function*”. Click the “Add target*” button to also send you an email by selecting “SNS topic” from the first drop-down list and “awsnotify” from the Topic drop-down list. Expand “Configure input”, select the “Constant (JSON text)” option, and enter “ClusterTerminated”. Scroll down and click the “Configure details” button. Enter something like “TerminateClusterAfterSnapshot” in the Name field and then click the “Create rule” button.

44. In order to avoid receiving two emails notifying you that the next snapshot has been completed successfully, select the “SnapshotComplete” rule (or whatever you named the rule in Lab 4), select the “Actions” drop-down menu and click on “Disable”.

45. Use the IP address of your small EC2 instance (described in Step 39) to copy your AWS credentials, the pcluster configuration file, your key pair, and the update_snapshot.sh script from your computer to the EC2 instance:

```
(aws) $ scp -rp -i ./MyKeyPair.pem ~/.aws ~/cluster-config.yaml
./MyKeyPair.pem ~/Documents/Python/PyHipp/update_snapshot.sh
ec2-user@54.169.221.196:~/
```

46. Use the address returned above to ssh into your EC2 instance:

```
(aws) $ ssh -i ./MyKeyPair.pem ec2-user@54.169.221.196
```

47. Check that your AWS credentials are set up properly by sending a notification message:

```
[ec2-user@ip-54.169.221.196 ~]$ aws sns publish --topic-arn
arn:aws:sns:ap-southeast-1:123456789012:awsnotify --message "FromEC2"
```

48. Since we will be creating snapshots from this EC2 instance, which will update the cluster config file saved here with the latest snapshot ID, we will need to create clusters from here from now on. Once you are logged in, follow the instructions below to install ParallelCluster:

    https://docs.aws.amazon.com/parallelcluster/latest/ug/install-v3-pip.html

49. Now ssh to “testcluser2” by doing:

```
[ec2-user@ip-54.169.221.196 ~]$ pcluster ssh -i ~/MyKeyPair.pem --region
ap-southeast-1 --cluster-name testcluster2
```

Remember to perform the usual initialization of the cluster.

50. We will now create a simple test to check that the head node for "testcluster2" can be successfully deleted after a snapshot has been completed. First we will create a simple slurm job that will just wait for 30 seconds. We can do this by making a copy of the slurm.sh file in the PyHipp directory, and adding the following lines to the end of the file (similar to what you did in Lab 4):

```
sleep 30

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

51. As we discussed in the lecture, the consol_jobs.sh script is used to wait for all the jobs to be completed before calling the ec2snapshot.sh script that will ssh to the EC2 instance to initiate the snapshot. So for this test, we will make some small changes to the ec2snapshot script to make sure everything works as expected.

```
(env1) [ec2-user@ip-10-0-8-173 PyHipp]$ cat ec2snapshot.sh
#!/bin/bash

ssh -o StrictHostKeyChecking=no -i /data/MyKeyPair.pem
ec2-user@xx.xx.xx.xx "source ~/.bash_profile; pcluster
update-compute-fleet --status STOP_REQUESTED --region ap-southeast-1
--cluster-name MyCluster01; ~/update_snapshot.sh data 2 MyCluster01"
```

Edit the ec2snapshot.sh file found in the /data/src/PyHipp directory to replace the "xx.xx.xx.xx" with the Public IP address for your EC2 instance. Also replace "MyCluster01" with "testcluster2", and "data" (the name for your snapshot) with "testdata".

52. As you can see above, the script sends several commands over ssh to your EC2 instance to be executed. So it will need a copy of your key pair file in order to access your EC2 instance. You can transfer the key pair file from your computer to "testcluster2" by doing (replace the IP address below with the public IP address of "testcluster2":

```
(aws) $ scp -i ~/MyKeyPair.pem ~/MyKeyPair.pem
ec2-user@54.255.166.210:/data
```

Note that we are saving the key pair file in /data so it will be saved in subsequent snapshots.

53. You can now submit the simple slurm job you created above (replace the name below with the name of your script):

```
(env1) [ec2-user@ip-10-0-8-173 PyHipp]$ sbatch sleepslurm.sh
```

Take note of the job number that is returned by slurm.

54. You can now submit the consol_jobs.sh script to depend on the previous job (replace "2" with the job number from the preceding step):

```
(env1) [ec2-user@ip-10-0-8-173 PyHipp]$ sbatch --dependency=afterok:2
/data/src/PyHipp/consol_jobs.sh
```

55. After the first job completes, the second job will call ec2snapshot.sh to stop the compute nodes and initiate a snapshot from your EC2 instance. After the snapshot is complete, the head node will be terminated, at which point, you be logged out of your cluster and returned back to the prompt on your EC2 instance. At this point, you can now call pcluster to delete "testcluster2". At this point, you can also go to your AWS Management Console in your browser, and delete the "testdata" snapshot.

56. You can now copy your key pair from your EC2 instance to /data on your "MyCluster01" cluster, ssh to your "MyCluster01" cluster from your EC2 instance, modify the ec2snapshot.sh script to enter the IP address of your EC2 instance. Check that the name of the cluster is "MyCluster01" and the name of the snapshot is "data". You will also need to change the instance ID in your Lambda function to the one for "MyCluster01".

Part VII) Executing the parallel processing pipeline

57. You are now ready to process all the channels. Return to the 20181105 directory, remove files generated by the previous job, and re-run pipe2.sh.

58. Check that the jobs were submitted successfully, and then submit the consol_jobs.sh script with a dependence on the five jobs submitted by the pipe2.sh script (replace the job numbers with those in your queue):

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

59. Check the queue to make sure everything is set up properly:

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   12    queue1    rplpl ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
   13    queue1      rs1 ec2-user  R       4:31      1 queue1-dy-r52xlarge-1
   14    queue1      rs2 ec2-user  R       1:25      1 queue1-dy-r52xlarge-2
   15    queue1      rs3 ec2-user  R       1:25      1 queue1-dy-r52xlarge-3
   16    queue1      rs4 ec2-user  R       1:25      1 queue1-dy-r52xlarge-4
   17    queue1 consol_j ec2-user PD       0:00      1 (Dependency)
```

60. You can now exit the cluster, and check on the progress either from the EC2 instance (described in Step 55) or from your computer:

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

This command will allow you to run the squeue command on your cluster, and get the output, without actually logging into the cluster.

Part VIII) Wrapping up

61. Once the jobs are completed, remember to first use pcluster to delete the "MyCluster01" cluster. You can then create a new cluster (named "MyCluster02" so you do not have to wait for "MyCluster01" to be deleted) from your EC2 instance to mount the snapshot and check the files.

    Remember to copy your AWS credentials, initialize conda, and activate your env1 environment.

62. Check that all the expected files have been generated (use the instructions in Step 26 to calculate the expected number of files for 110 channels), and compute the time taken to complete the jobs. **Include a screenshot of the output of your checkfiles.sh script showing the number of files created, and the time taken (remember to take the difference between the earliest start time and the latest end time) for the jobs to complete in your lab report.**

63. Once you are done, you can exit your cluster, and then delete the cluster from your EC2 instance.

    You do not need to save another snapshot since presumably you did not modify anything from the last snapshot.

64. You should leave your EC2 instance running since your basic free-tier account allows a small EC2 instance to run 24 hours a day for 31 days. This way, you will not have to go through the setup process again.

65. Whenever you want to create a cluster in the future, you should just do so from your EC2 instance since the snapshot ID in the cluster config file will be the one that is the most updated.

66. In the next lab, we will look at more ways to optimize the parallelization of the data processing.

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

1) Screenshot of your checkfiles2.sh script from Step 14
2) Screenshot of the output from your checkfiles2.sh shell script from Step 14
3) Sum of the two job durations from Step 14
4) Actual time taken for both jobs from Step 14
5) Time saved from parallel processing from Step 14
6) Screenshot of the output from your checkfiles2.sh shell script from Step 62
7) Actual time taken for all five jobs from Step 62
8) Screenshot of your EC2 Dashboard from (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)