Exacloud Workshop

Ted Laderas (laderast@ohsu.edu) and Ryan Swan

November 3, 2016

What is exacloud?

Exacloud is OHSU's cluster computing environment. It currently has over 6000 CPUs of varying capabilities and memory that can be utilized by users who want to run parallel computing jobs.

However, to run jobs effectively on exacloud, you must understand some basic techniques and how exacloud works.

Architecture of exacloud

Essentially, you can think of exacloud as divided into two different node types: the first type is the *head node*, which is the node that you sign into initially into exacloud. The head node handles all of the scheduling and file transfer to the other nodes in the system. The other nodes are known as subservient *compute nodes*. They don't do anything unless the head node assigns them a task.

How does the head node tell the compute nodes what to do? There is a program called HTCondor (currently version 8.0.6 on exacloud) that schedules computing jobs for all of the nodes on exacloud. How does it decide which jobs to run on which nodes? Part of it has to do with compute requirements - each individual job requires a certain amount of CPU power and Memory. This is balanced with the current load (i.e. other jobs currently running) on the cluster.

You might wonder if you need to load all of your data onto each node that you run your jobs on. The short answer is no, because exacloud has a distributed filesystem called Lustre. Essentially when you copy a file onto Lustre, the file gets copied in such a way across nodes that it is easily accessible by each node. The drawback to Lustre is that it is currently difficult to maintain and can be prone to data loss.

For this reason, do *not* use Lustre for long term storage of your data! It's better to transfer your files off of Lustre when you're done.

The Tragedy of the Commons

Exacloud is a shared resource, which means it is shared across OHSU. We all need to be sensible when we run jobs on it. Try to limit the number of concurrent jobs you are running (I try to keep things under 200), and be aware of how long your job will take. You can adjust the number of jobs running by limiting the number of jobs you queue, or by setting the ConcurrencyLimit variable in your submit file. For more info, see the Exacloud User Guide under 'Scheduling Policy under Exacloud'.

Our Goal for Today

We will be reproducing the following analysis using data pulled from the twitter feed: On Geek Versus Nerd. We want to discover the words that co-occur with "nerd" and "geek" with high frequency in a corpus of tweets.

$$\text{pmi}(w;v) = \log \frac{p(w,v)}{p(w)p(v)} = \log p(w|v) - \log p(w)$$

Figure 1: PMI equation

Essentially, we need to calculate the probability of our two words of interest. Then for every other word, we calculate the probability of co-occurence with one of our words.

Subtasks

We're going to do the counting by splitting our twitter corpus (`fulldata.csv`) into multiple files. Then we'll run the `pmi.py` script to produce counts for each of these files. The `stitchpmi.py` script will take the output from running the scripts, put them together, and calculate the probabilities for each word to produce the final PMI score.

Other fun pairs of words you might want to use: * "british" and "american" * "love" and "hate" * "tweet" and "post"

We will be achieving this by executing 5 tasks, which is the standard workflow for setting up exacloud runs.

1. First we will test our code in an interactive session.
2. Then, we will split our data up into individual files.
3. Next, we will set up the submit script to specify how to run our scripts on these individual files.
4. Then we run our job on exacloud.

5. Then we'll open another interactive session to knit our results back together and produce the pmi scores. In this same session, we'll run a simple R script to plot our interesting results.

Task 0: Getting onto exacloud and understanding the lustre filesystem

Before you can even start with exacloud, you need an exacloud login and password. You will need to talk with ACC for an account and password.

1. To connect with exacloud, use the ssh command and input your password when prompted.

```
ssh USERNAME@exacloud.ohsu.edu
```

2. Your entry point is the ACC (Advanced Computing Center) filesystem, which is shared across all ACC machines (not just exacloud). You can run jobs from here, but you will run into space limitations (10 Gb limit). If you have larger data, it's much easier to use the lustre filesystem. So let's go to the lustre folder for BioDSP:

```
cd /home/exacloud/lustre1/BioDSP/
```

3. Make your own folder in the `BioDSP/users/` folder. Copy the scripts, and example data into your folder. (Obviously, substitute your own username for USERNAME here).

```
cd Users
#make your own directory in the /home/exacloud/lustre1/BioDSP/Users/ folder
mkdir USERNAME
#change to your user folder
cd USERNAME
```

4. Copy the Tutorial tarball to your current directory

```
##Note the Trailing period!
cp /home/exacloud/lustre1/BioDSP/exacloudTutorial.tar.gz ./
```

5. Unzip your tarball and change into the exacloudTutorial directory. This is where you're going to do all of your work (i.e. `/home/exacloud/lustre1/BioDSP/Users/USERNAME/exacloudTutorial/`) for the tutorial.

```
tar -xzvpf exacloudTutorial.tar.gz
cd exacloudTutorial/
```

6. Change permission on all the files in your directory. You will need write access for these files.

```
chmod 777 *
```

Task 1: Testing your code in an interactive session

IMPORTANT: Do not run jobs on the head node! You will be yelled at, and for good reason. The head node is a very busy node, handling job scheduling and file transfer for the entire cluster. If you run jobs on it, you essentially are slowing everyone else down.

Instead, you can test your jobs by opening up an interactive session on exacloud. Essentially, opening up an interactive session guarantees the use of a particular node on exacloud. You can run jobs in an interactive session on the command line, which is what we're are going to do. It's also useful when you just need compute time for short jobs that don't need to be parallelized.

1. Open an interactive session using condor_submit. It may take a few seconds for the interactive session to open up, so be patient. (To exit the interactive shell, you can use the command `exit`.)

   ```
   condor_submit -interactive
   ```

2. Take a look at the n-gram script (*pmi.py*). What are the functions?
3. Modify the two analysis words (the variables are named "woi1" and "woi2").
4. We will be testing out the n-gram counting script on the test data ('test.csv'). Take a look at it. What is the structure of the file. Which column are the tweets in?
5. Let's try running the script:

   ```
   python pmi.py test.csv
   ```

6. What is the output of the python script? How is it named? (list the contents of your folder if you're not sure).
7. Keep your interactive session open as we'll use it in Task 2.

Task 2: Splitting up your problem

We'll be using the unix command split to split our 1 and 2-gram task up into 50 smaller tasks using the -n option. Using the -d option, the output of split will be numeric.

1. Run the split command on the training file in your directory.

```
#numeric-suffixes argument is so split doesn't label the files
#alphabetically (data.aa, data.ab, data.ac, etc)
#after specifying the file fulldata.csv, we can specify a prefix for the
#split files (output will be data00, data01, data02)
split --numeric-suffixes --lines=200000 fulldata.csv data
```

2. List the contents of your directory. How many files did you make using the split command? Remember this number.

```
ls data*
```

3. If you noticed, the output of split gives us a padded numbering (for example, `data.00` instead of `data.0`, `data.01` instead of `data.1`), which makes it difficult to deal with using the built in looping functions in HTCondor. So we're going to rename these files using the `renameFiles.sh` script. (Take a look at it if you like with `nano renameFiles.sh`).

```
#need to provide prefix as argument to renameFiles.sh
./renameFiles.sh data
```

4. Confirm that the files have been renamed properly.

```
ls -l
```

Extension: Using multiple directories to divide your jobs

If you have multiple files to process at a time, another alternative is to set up numbered directories where each file has the identical name. This affects how you set up your submit script.

In this case, your directories might be numbered numerically ("folder0", "folder1", "folder2", etc) so you can programmatically run data in each of them. Your files will probably be named identically in each of these folders so you can run the data.

Task 3: Setting up your submit script to HTCondor

1. While still in your interactive session, look at "pmi.submit" using a text editor such as nano. What do you notice?

   ```
   nano pmi.submit
   ```

2. Set the N variable to the number of files you want to process (you did remember this number, right?).
3. We are going to be using the built in looping mechanism in HTCondor to run the pmi.py script on each file separately. Look at the "arguments" line. We use the built in $(Process) variable to select which file to run.
4. Exit the interactive session using `exit`. We will need to be in the head node to now submit our script.

Task 4: Running your Job on Exacloud

There are two commands that will be necessary to understand running jobs on exacloud: the first is `condor_submit`, which submits the job, and `condor_q`, which shows you current jobs running on exacloud. If you need to run something quick and dirty with minimal setup, you can also use `condor_run`.

1. If you haven't yet left your interactive session, use *exit* to do so:

   ```
   exit
   ```

2. Let's see how busy the cluster is. *condor_q* will show you how many jobs are running and who is running them.

   ```
   condor_q
   ```

3. If things seem kosher, we'll run our job:

   ```
   condor_submit pmi.submit
   ```

4. Run condor_q again to ensure that your job(s) are running.

Task 5: Putting your results back together

Hopefully your jobs executed correctly. If not, ask for help. Now we'll put together all of the .pmioutput files together and create unified CSV that has PMI values for co-occuring words.

1. Enter an interactive session again using *condor_submit -interactive*.
2. Ensure that all files were processed by listing all of the .pmioutput files (if you didn't remove your test.pmioutput file, you may want to remove it before proceeding):

   ```
   ls *.pmioutput
   ```

3. Run the *stitchpmi.py* script to bring it all together. The output of *stitchpmi.py* is a single file, *totalpmioutput.csv*.

   ```
   python stitchpmi.py
   ```

4. Let's plot the non-zero results using R. Run the *plot-pmi.R* script. The output of this will be a single plot, *pmiPlot.pdf*.

   ```
   #replace WORD1 and WORD2 with your words!
   Rscript plotPMIOutput.R totalpmioutput.csv WORD1 WORD2
   ```

5. To look at the plot, you'll need to transfer everything off the file system. Use an FTP program such as WinSCP or Cyberduck to download your plot and totalpmioutput.csv file.
6. Share with the group! Let's see what you came up with.

Task 6: Cleanup

1. Be sure to clean your files out of your folder. At the very least, remove the data files:

```
rm data*
rm fulldata.csv
```

Task 7: What next?

We have showed you the basic way to run jobs on exacloud. Now, you'll need to learn more about writing submit scripts and how to run different languages on exacloud. Note that our version of condor is older (8.0.6) compared to the more sophisticated versions (8.4.0 and beyond), so some of the tricks on the current doc pages won't work.

The one important trick is how to install dependencies that you need for each language in Lustre. You can set important environment variables (such as *R_LIBS_USER*, which is the default location of your personal R library) using an "environment=" line in your submit script. For more info, please see the Exainfo beginner user wiki linked below.

Task 8: Debugging and Further Resources

One extension to the submit script you can add is error logging. This is especially useful in debugging what is going wrong with your jobs.

```
#Note: if you have a lot of processes, this will generate a lot of files!
#standard output gets written to out
output = data.$(Process).out
#error is probably the most useful file for debugging
error = data.$(Process).err
log = data$(Process).log
```

If you want to stop a job, you can use `condor_rm` to remove your job from the queue.

The Exainfo beginner user wiki can be helpful, especially with hints on how to set up your own library and dependencies for particular languages.

Also, the Exacloud User Guide can be helpful as well.

Running Your First Condor Job is a helpful page to get you started.

If your job seems slow, check the exacloud usage display at http://exacloud.ohsu.edu/ganglia/

More information about condor commands here: Useful Condor Commands

More information about writing submit files is here: Writing Condor Submit Files. Note that this is written for Condor 8.4.0, so some of the submit recipes don't work.

Task 3 Optional Extension: Asking for machines with specific requirements

HTCondor has a 'classified' system that allows you to request specific processing and memory requirements for your job. You can specify parameters such as memory:

```
request_memory = 1.5 GB
```

Please note that nodes with your requirements may not be available, and so your job may be waiting in the condor_q until such nodes are available.

If that is too strict, you can also rank machines by preference.

```
Rank  = (Memory >= 4096)
```

This is a less restrictive form, in that the machines will be ranked by available memory and HTCondor will run the jobs on the best nodes available.

Task 3 Optional Extension: Writing a script that generates a submit script

There are other ways to write submit scripts. For example, if your job requires cycling through a list of non-standard files that are not sequentially numbered, this is the best way to process them. (Later versions of HTCondor allow you to use the Queue command to handle different file names.)

For example, we can run the submit script for three different files called "april.csv", "may.csv", and "june.csv" by writing a program to build the following submit script. Note we set some parameters at the beginning and then change the arguments parameter to run the script for each file name. The Queue command starts the job.

```
#Example submit script for running on multiple files
#you can easily write a python script to generate such a file!
universe = vanilla
#set the executable here (could be a shell script, software, anything)
executable = /usr/bin/python
when_to_transfer_output = ON_EXIT

arguments = "pmi.py april.csv"
Queue

arguments = "pmi.py may.csv"
Queue
```

```
arguments = "pmi.py july.csv"
Queue
```

Task 3 Optional Extension: Submit File for Multiple directories

How would you change the above submit file to run files in different directories as set up in the Task 3 extension? *Hint:* adjust your arguments variable!

Optional Extension: Running other languages (R)

How would we run other languages in our submit script? What about a version of a language you installed?

R_LIBS or .libPath() for libraries

The default installation directory for R is in your home directory (`/home/users/LOGIN`). Usually this location is under `R/x86_64-redhat-linux-gnu-library/3.3` (for the current R version).

The compute nodes are dumb, and so they don't know where your packages are, and so you need to pass the location of the library. There are two ways to do this.

1) Set an Environment Variable in your submit script. You do this by adding a line for your environment variable R_LIBS, such as:

```
environment = "R_LIBS = ~/R/x86_64-redhat-linux-gnu-library/3.3/"
```

2) Specify the path in your actual R script using the `.libPaths()` function. Note that you have to do this first, even before loading up packages, or your script will error out.

```
.libPaths("~/R/x86_64-redhat-linux-gnu-library/3.3/")
```

Running multiple files in R

If you are running a script on different files, it is really useful to pass your script an argument about which file to pass. I use the `cmdArgs()` function to pass on different arguments to the script:

```
##let's call this file runRJob.R
library(R.utils)
i <- as.numeric(cmdArg("i"))
#need to add 1 to i value because condor starts with 0 for indexing and
#R starts with 1
i <- i + 1
inputDir <- cmdArg("dir")

#List files in inputDirectory
fileList <- list.files(inputDir, pattern = ".csv")
fileToRun <- fileList[i]

#... do some stuff with the fileToRun
```

So, to run this script on a file in the directory on the command line, you would use the following setup:

```
Rscript runRJob.R i=0 inputDir="~/filesToProcess/"
```

So, now you need to set up your submit script to do something similar. For example, if there were 3 files we wanted to run in our `inputDir` directory, we'd want our submit script to execute the following commands, with each of them run on a separate node.

```
Rscript runRJob.R i=0 inputDir='~/filesToProcess'
Rscript runRJob.R i=1 inputDir='~/filesToProcess'
Rscript runRJob.R i=2 inputDir='~/filesToProcess'
```

In setting up a submit script to do this I'd use these options. Note the `arguments` line uses the `$(Process)` variable to select a number. Also note that we have to know how many files are in the directory to process. This is somewhat kludgey, but I haven't found a more elegant way to do it with our version of HTCondor.

```
N = 3
universe = vanilla
executable = /usr/bin/Rscript
getenv = TRUE
environment = "R_LIBS=~/lustre/rlib"
arguments = "--vanilla runTSNEexacloudCDMarkers.R i=$(Process) inputDir='~/filesToProcess'"
Queue N
```

For a full example, look in the `Rexample/` folder of the exacloudTutorial.