SI 618 Lab 5: Getting set to run MapReduce jobs with Spark

Objectives:
1. Set up Flux server access, including remote terminal and file client programs.
2. Get familiar with some basic Unix command line utilities
3. Get familiar with Hadoop File System (HDFS) commands
4. Run a sample Map-Reduce job with PySpark

Note: Parts of this lab use, or are designed to follow, excerpts from the Cavium User Guide.

---

1. Getting access to Cavium
To access the Cavium cluster, you need to first obtain a Cavium account.

1a. Obtaining a Cavium account
You should have already received an email recently from [ARC-TS] on your Cavium Account that you'll use for this lab. Please see https://arc-ts.umich.edu/cavium/user-guide/ for a quick guide to accessing this.

1b. Preparing for 2-Factor Authentication using Duo
If you don’t have 2-Factor authentication already set up, please follow these detailed instructions here: http://documentation.its.umich.edu/2fa/enroll-smartphone-or-tablet-duo

Note: If you are using a network other than MWireless, you will need to use a VPN (Virtual Private Network). If needed, follow the instructions here: http://www.itcom.itd.umich.edu/vpn/

2. Working with a remote server
For our Spark assignments, the computing and storage of datasets will no longer be done on your laptop. Instead, you have an account on a server cluster run by UM's Advanced Research Computing (ARC) center. All the datasets we will use will be preloaded for you on the cluster, and all Python scripts you create will run on the cluster. You will use your Cavium Hadoop Account to run jobs and examine the data.

You will use your laptop (or other local client machine) for (a) logging into the cluster and then running commands in a terminal window, and (b) transferring files to/from the server cluster. For example, you might want to grab the output file from a script on the server so that you can upload it to Canvas from your laptop. Or you might edit your Python script in an editor on your laptop and transfer it to the server when you want to run and test it remotely. The next two steps help you get set up to do that.

2a. Logging in

To login, you need to have a terminal window that provides a secure connection to the server.

- If you use a Mac, its Terminal app is all you need.
- If you use Windows, you can install Putty (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html).

Note: Logging into the remote server from off-campus. *For security reasons, if you want to use your Cavium account from off-campus locations, you will need to connect via VPN (Virtual Private Networking). This requires you to install a VPN client if you don't already have one. Please see the University of Michigan instructions here: http://www.itcom.itd.umich.edu/vpn/. Once you follow the instructions to connect using the VPN client, your laptop will be "virtually" on campus, and all other steps to connect, transfer files, etc. are the same.*

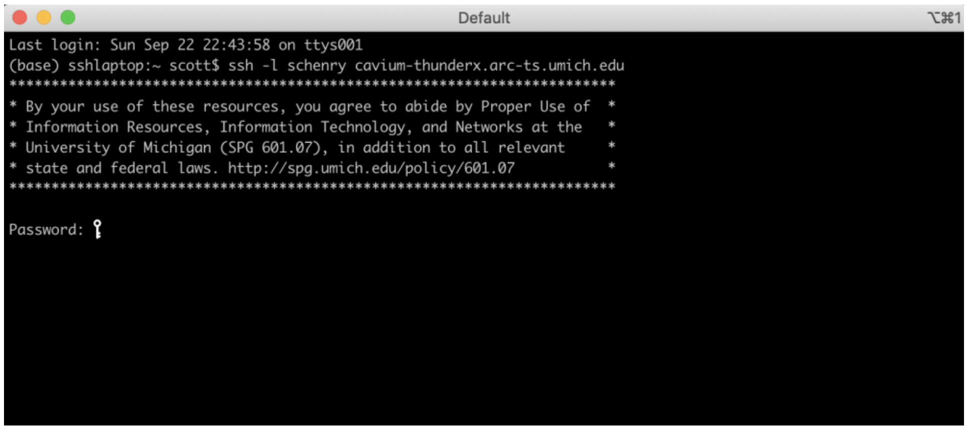
Make sure that you can login by running in your terminal (replace uniqname):

```
ssh uniqname@cavium-thunderx.arc-ts.umich.edu
```

On **Windows**, if you use Putty, launch Putty and enter cavium-thunderx.arc-ts.umich.edu as the host name then click open.

More detailed login instructions can be found here: https://arc-ts.umich.edu/cavium/user-guide/

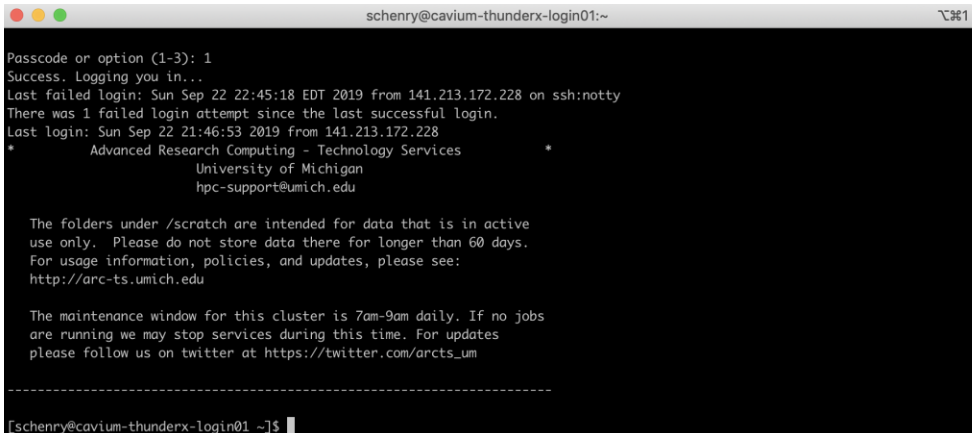
If connected successfully, you should see a login prompt similar to this:



```
Last login: Sun Sep 22 22:43:58 on ttys001
(base) sshlaptop:~ scott$ ssh -l schenry cavium-thunderx.arc-ts.umich.edu
**********************************************************************
* By your use of these resources, you agree to abide by Proper Use of  *
* Information Resources, Information Technology, and Networks at the   *
* University of Michigan (SPG 601.07), in addition to all relevant     *
* state and federal laws. http://spg.umich.edu/policy/601.07           *
**********************************************************************

Password:
```

Once you enter your UM password and pass the Duo two-factor authentication, you should be successfully logged in, and see a message something like this:



schenry@cavium-thunderx-login01:~

```
Passcode or option (1-3): 1
Success. Logging you in...
Last failed login: Sun Sep 22 22:45:18 EDT 2019 from 141.213.172.228 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Sun Sep 22 21:46:53 2019 from 141.213.172.228
*          Advanced Research Computing - Technology Services          *
                        University of Michigan
                        hpc-support@umich.edu

   The folders under /scratch are intended for data that is in active
   use only.  Please do not store data there for longer than 60 days.
   For usage information, policies, and updates, please see:
   http://arc-ts.umich.edu

   The maintenance window for this cluster is 7am-9am daily. If no jobs
   are running we may stop services during this time. For updates
   please follow us on twitter at https://twitter.com/arcts_um

----------------------------------------------------------------------

[schenry@cavium-thunderx-login01 ~]$ 
```

2b. File transfer between your laptop and server:

Download the zipped file si618CaviumSetup.zip from canvas and unzip it. Copy the contents to your home directory on the server. To do this, you can either use:

1) Terminal/Putty
2) Install an SFTP **client** like FileZilla (https://filezilla-project.org/) or CyberDuck.

When Terminal/Putty is running:

- Make sure you're not on the cavium cluster and running on your local machine
- Run command (replace si618CaviumSetup_path with the path in your local computer and uniqname with your uniqname):

Mac:

```
scp -r si618CaviumSetup_path uniqname@cavium-thunderx.arc-ts.umich.edu:/home/uniqname
```

```
Hadoop fs -put si618CaviumSetup
```

Windows:

```
pscp -scp -r si618CaviumSetup_path uniqname@cavium-thunderx.arc-ts.umich.edu:/home/uniqname
```

```
Hadoop fs -put si618FluxSetup
```

- It'll ask you for your password and then to authenticate via duo

When FileZilla/CyberDuck is running:

- File -> Site Manager -> New Site
    - Host: sftp://cavium-thunderx.arc-ts.umich.edu
    - Username: youruniquename
    - Logon Type: Interactive
- Click Connect.

- You should see your Linux home directory on the cluster as a Remote Site panel on the right.

On each side, the top panel helps you navigate the directory structure while the bottom panel shows you the file. The files starting with '.' (dot) are hidden files. Move all files in si618CaviumSetup.zip (unzipped) to your home directory.

Now try this on your login node (on cavium-thunderx.arc-ts.umich.edu):

```
ls -l
```

As you see, the "ls -l" command lists all the files in a directory. If you don't specify the directory, the system assumes you mean the current directory.

Question #1: After running the scp command, what file(s) are now in your home directory? There should be at least two: a ".sh" file, and a ".py" file (disregard the __macosx folder if it is in there).

3. Examine the sample dataset with Hadoop file system commands

In addition to regular files, we'll be working with big data files that live inside the Hadoop File System (HFS) discussed in class. Think of HFS as a special filing cabinet for data.

The big data files we'll be using come from the Google NGrams dataset. These datasets contain counted syntactic n-grams (sequences of one or more words) extracted from the English portion of the Google Books corpus. *(If you're interested, the datasets are described in the following publication. The dataset format and organization are detailed in the README file.)*

The ngrams dataset is tab-delimited and of the form (commas inserted for clarity):

```
ngram,year,total occurrence count,number of volumes ngram occurred in
```

For now, we'll just work with 1-grams, which are single words that occurred in books dating back to 1500, and up to the present day.

The set of "1-gram" files lives in the Hadoop file system (HFS), in an HFS folder that is created under my account that has public access. HFS uses very similar commands to a regular Unix file system. All HFS commands are prefaced with "hadoop fs", followed by the file system command and optional arguments the command needs.

For example, to list all the data files in this HFS directory "/var/umsi618f21/lab5/ngrams/data" you would run the "-ls" command, like so:

```
hadoop fs -ls /var/umsi618f21/lab5/ngrams/data/
```

Question #2: What is the name of the last file in the listing for HFS folder /var/umsi618f21/lab5/ngrams/data/?

You can also dump the contents of an HFS file using the "-cat" command. Here, I've added a pipe "|" symbol followed by the "grep" command. The pipe symbol means the output of the "-cat" command (the file contents) will be piped to the input of the "grep" command, which searches the input for the given regular expression.

```
hadoop fs -cat /var/umsi618f21/lab5/ngrams/data/* | grep "^information_NOUN"
```

Note: The second column is the year, the third column is the total mentions
Question #3: What year was “information” first mentioned (as a noun) in Google Books data?

4. Run a sample PySpark job

In this stage you'll run a sample Python script that launches a Spark job to compute the average word length of words in the Google NGram dataset, per year. This code lives in the ngram-job.py file you copied earlier. To save time we are now going to use only the file corresponding to words starting with “x” of the HDFS NGram dataset.

Type the following command to launch the Spark "average word length" job:

```
./spark-run.sh ngram-job.py /var/umsi618f21/lab5/ngrams/data/googlebooks-eng-all-1gram-20120701-x ./ngrams-out
```

If you get an error about permission denied, run the command line below. chmod +x allows executing file as program. Then, run the code above again.

```
chmod +x spark-run.sh
```

A Unix script called **spark-run.sh** was made to save you from typing most of the following command:

```
PORT="$(shuf -i 10000-60000 -n 1)"
spark-submit \
        --master yarn \ # Run this as a Hadoop job
        --queue umsi618f21 \ # get resources from this allocation
        --num-executors 2 \ # Run with a certain no of executors, e.g.2
        --executor-memory 4g \ # Specify each executor's memory, e.g. 4GB
        --executor-cores 2 \ # Specify each executor's # of CPUs, e.g. 2
        --conf spark.hadoop.validateOutputSpecs=false \
        $1 $2 $3 \
        spark.ui.port="${PORT}"
```

The **spark-submit** command (excluding the comments and backslashes) submits the script, saved in **ngram-job.py**. We've created a Unix script called **spark-run.sh** which includes some parameters that control how the script is to be executed. Normally you shouldn't need to deal with setting a random port but during this lab many of you might try to submit at the exact same time using the same port which might be an issue (hence the bits about the port in the script). Normally *(outside of the lab setting with many of you using the same queue and port at the same exact time)* you could simply run:

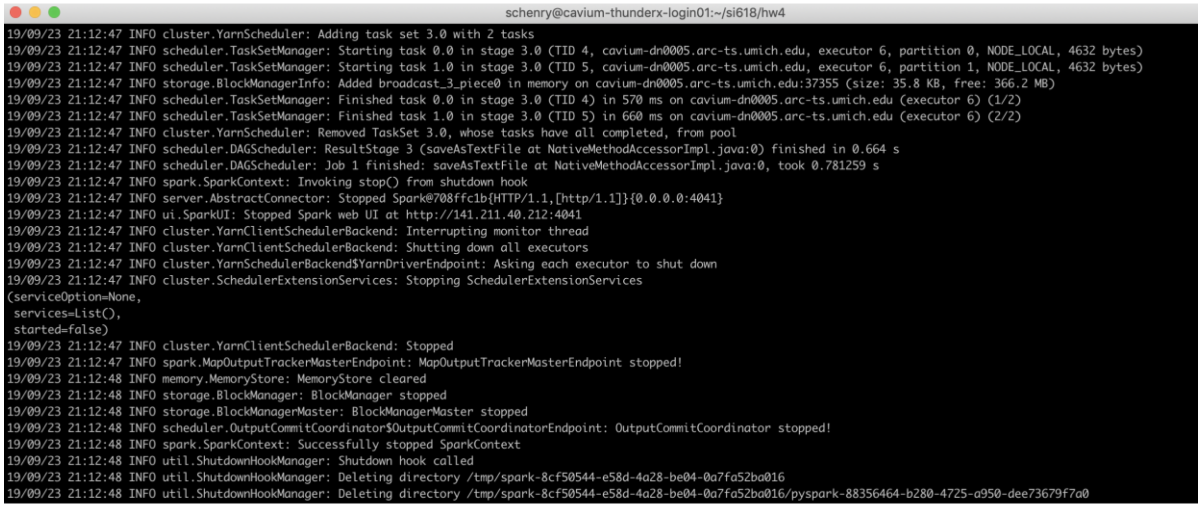
```
spark-submit --master yarn --num-executors 35 --executor-memory 5g --executor-cores 4 ngram-job.py  /var/umsi618f21/lab5/ngrams/data/googlebooks-eng-all-1gram-20120701-x ./ngrams-out
```

And chances are you would have been fine. But it doesn't hurt to be cautious!

The parameters to **spark-run.sh** consist of:

1. The name of the Spark script to run (ngram-job.py).
2. The Hadoop File System directory that contains the input files (/var/umsi618f21/lab5).
3. The Hadoop File System directory to store the output (./ngrams-out). This will create a new directory in <u>your</u> Hadoop File System space.

When launched, you will see a bazillion lines of obscure-looking output that mentions tasks starting and finishing. This means all the cluster machines are being started up and the various mappers and reducers being launched. It will take several minutes for the cluster job to finish, at which point you'll see something like this in your terminal window with words like "shutdown".



schenry@cavium-thunderx-login01:~/si618/hw4

```
19/09/23 21:12:47 INFO cluster.YarnScheduler: Adding task set 3.0 with 2 tasks
19/09/23 21:12:47 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 3.0 (TID 4, cavium-dn0005.arc-ts.umich.edu, executor 6, partition 0, NODE_LOCAL, 4632 bytes)
19/09/23 21:12:47 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 3.0 (TID 5, cavium-dn0005.arc-ts.umich.edu, executor 6, partition 1, NODE_LOCAL, 4632 bytes)
19/09/23 21:12:47 INFO storage.BlockManagerInfo: Added broadcast_3_piece0 in memory on cavium-dn0005.arc-ts.umich.edu:37355 (size: 35.8 KB, free: 366.2 MB)
19/09/23 21:12:47 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 3.0 (TID 4) in 570 ms on cavium-dn0005.arc-ts.umich.edu (executor 6) (1/2)
19/09/23 21:12:47 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 3.0 (TID 5) in 660 ms on cavium-dn0005.arc-ts.umich.edu (executor 6) (2/2)
19/09/23 21:12:47 INFO cluster.YarnScheduler: Removed TaskSet 3.0, whose tasks have all completed, from pool
19/09/23 21:12:47 INFO scheduler.DAGScheduler: ResultStage 3 (saveAsTextFile at NativeMethodAccessorImpl.java:0) finished in 0.664 s
19/09/23 21:12:47 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at NativeMethodAccessorImpl.java:0, took 0.781259 s
19/09/23 21:12:47 INFO spark.SparkContext: Invoking stop() from shutdown hook
19/09/23 21:12:47 INFO server.AbstractConnector: Stopped Spark@708ffc1b{HTTP/1.1,[http/1.1]}{0.0.0.0:4041}
19/09/23 21:12:47 INFO ui.SparkUI: Stopped Spark web UI at http://141.211.40.212:4041
19/09/23 21:12:47 INFO cluster.YarnClientSchedulerBackend: Interrupting monitor thread
19/09/23 21:12:47 INFO cluster.YarnClientSchedulerBackend: Shutting down all executors
19/09/23 21:12:47 INFO cluster.YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
19/09/23 21:12:47 INFO cluster.SchedulerExtensionServices: Stopping SchedulerExtensionServices
(serviceOption=None,
 services=List(),
 started=false)
19/09/23 21:12:47 INFO cluster.YarnClientSchedulerBackend: Stopped
19/09/23 21:12:47 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/09/23 21:12:48 INFO memory.MemoryStore: MemoryStore cleared
19/09/23 21:12:48 INFO storage.BlockManager: BlockManager stopped
19/09/23 21:12:48 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
19/09/23 21:12:48 INFO scheduler.OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/09/23 21:12:48 INFO spark.SparkContext: Successfully stopped SparkContext
19/09/23 21:12:48 INFO util.ShutdownHookManager: Shutdown hook called
19/09/23 21:12:48 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-8cf50544-e58d-4a28-be04-0a7fa52ba016
19/09/23 21:12:48 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-8cf50544-e58d-4a28-be04-0a7fa52ba016/pyspark-88356464-b280-4725-a950-dee73679f7a0
```

We'll be looking at this code in more detail later, but here's a high-level walk through the transformations used in the script:

1. Transform each line of the original dataset into an array of the form [ngram, year, occurrences, volumes]

2. Create a new dataset *length* with each row containing an array of the form [year, number of characters in words]
3. Create a new dataset *word* with each row containing an array of the form [year, number of words]
4. Create a new dataset *average_length*, dividing *length* by *words*, resulting in an array of the form [year, average word length]

Why create a new dataset at each stage? To ensure many processes can operate on the same dataset at once, without worrying that some other process might deliberately or accidentally change it, each object is **immutable** - it cann*ot* be modified once created. This is a common feature of highly **concurrent** programming languages (those designed to make it easy to do large-scale computing in parallel).

5. Looking at Spark output

When the Spark job is finished, the output will go into the Hadoop File System directory you specified as the third parameter on the command line (./ngrams-out). Spark will create multiple output files that correspond to the multiple parallel MapReduce sub-jobs that Spark created (e.g. if the final step was a reduce step, and there are 30 reducers, there will be 30 output files).

Take a look at the output files by typing

```
hadoop fs -ls ./ngrams-out
```

<u>Question #4</u>: After the Spark job completes, what are the top two files listed in your Hadoop File System output directory ./ngrams-out?

To do something useful with the output, you'll need to extract and combine all these output files from the HFS. This is easy to do. Just type the command:

```
hadoop fs -cat ./ngrams-out/part-* > ngrams-output.txt
```

This will add together all the "part-" output files in your HDFS directory and create a single regular file in your current directory called "ngrams-output.txt".

You can then look at the first ten lines of this final output file by typing

```
head -10 ngrams-output.txt
```

The output consists of tuples: (year, average_word_length).

Question #5: **What were the average word lengths of words starting with x observed in books from the years 1810, 1946, and 2002?**

What to submit:

- Lab 5 Quiz (on Canvas)

References:

- Cavium User Guide
- Documentation for Hadoop is provided at: https://hadoop.apache.org/docs/r2.9.2/
- Hadoop User Guide at: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html
- CSAR Workshop on PySpark: https://github.com/caocscar/workshops/blob/master/pyspark/pyspark.md