**SI 618: 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 example task in the [UM CAEN Spark Guide](http://caen.github.io/hadoop/user-spark.html).

## **1. Getting access to Flux**

To access a Flux Hadoop cluster, you need to first obtain a Flux account and setup MToken

**1a. Obtaining a Flux account**

You should have already received an email recently from [ARC-TS] on your Flux Hadoop Account that you'll use for this lab. To access the Flux Hadoop cluster, you must first have a (free) account on Flux.

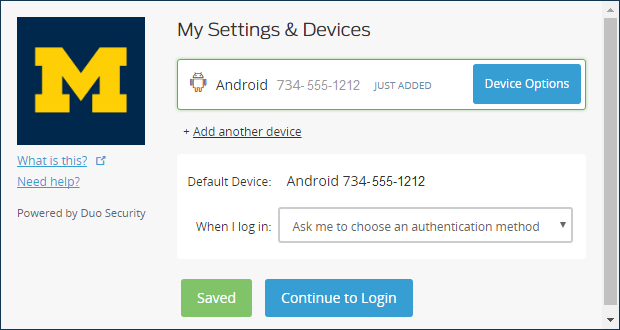
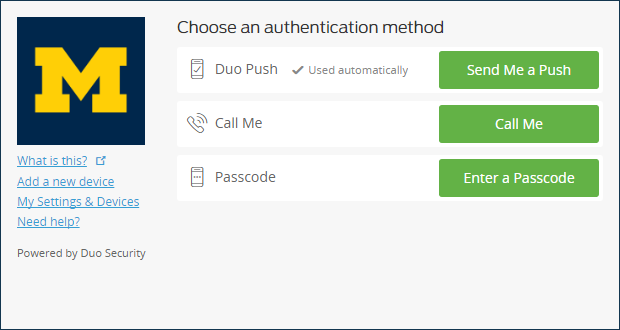
If you don’t already have a Flux account you need to request one at: <http://arc-ts.umich.edu/hpcform/>

In the "Advisor's Name" box, enter **Ceren Budak**  
In the "Advisor's Email Address" box, enter **cbudak@umich.edu**  
In the text box below "Please describe the projects", enter **For use in SI 618**

Once you’ve received confirmation from the Flux administrator about your account creation, you are eligible to login onto the Fladoop cluster.

**1b. Preparing for 2-Factor Authentication using Duo**

These are simplified instructions.   
For detailed ones, follow **Enroll a Smartphone or Tablet in Duo**: <http://documentation.its.umich.edu/2fa/enroll-smartphone-or-tablet-duo>

1. 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. Search for “Duo Mobile” app on your mobile’s app store and install it
3. Register your device
   1. Log in to UMich Account Management via <https://password.it.umich.edu/pwm/public/>
   2. You might have to fill in some security questions, and an external email address for future password resets.
   3. Select Two-Factor (Duo)
   4. Click on “Turn on Two-Factor” > Start Setup
   5. Follow the on-screen instructions
   6. Eventually, you should see that your mobile phone gets added as a device.  
        
      
   7. Click on “Continue to Login” and test the setup by selecting “Send me a Push”  
        
      
   8. You will see a push notification on your phone and you will have to approve the Login Request
   9. **Optionally**, you can enable 2-Factor Authentication for web-login too.

## **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 Flux 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>).

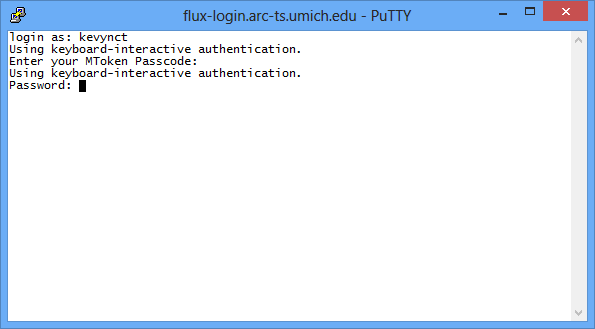
**NOTE: Logging into the remote server from off-campus.** For security reasons, if you want to use your Flux account from off-campus locations, you will need to connect via VPN (Virtual Private Networking). This requires you to install a VPN client if you don’t already have one. Please see the University 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 youruniqname with, well, your UM uniqname):

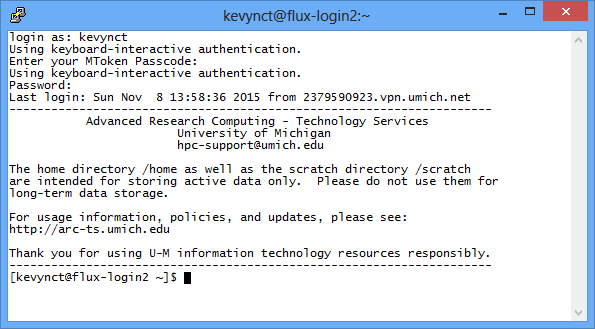
|  |
| --- |
| ssh **youruniqname**@flux-hadoop-login.arc-ts.umich.edu |

On Windows, if you use Putty, you need to put the flux--hadoop-login.arc-ts.umich.edu server name in the Putty dialog box, like so, then click "Open" to start the terminal session.

If connected successfully on Mac or Windows you should see a login prompt similar to this:



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



## **2b. File transfer between your laptop and server:**

Download the zipped file si618FluxSetup.zip and unzip it. Copy the contents to your home directory on the server. To do this, you need to either 1) use unix ”scp” command for the same purpose (for more details, type “man scp” on the command line), or 2) install an SFTP **client** like FileZilla (<https://filezilla-project.org/>) or CyberDuck. If you use FileZilla, be alert about the Optional Offer of installing Yahoo as the search engine and homepage.

When FileZilla is running, enter

sftp://flux-xfer.arc-ts.umich.edu into the Host box,

youruniquename into the Username box,

your password into the Password box,

then click "Quickconnect".

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 si618FluxSetup.zip (unzipped) to your home directory.

Now try this on your login node (on flux-hadoop-login.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.

**Worksheet #1: After running the cp command, what file(s) are now in your home directory? There should be at least two: a ".sh" file, and a ".py" file.**

### **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](http://storage.googleapis.com/books/ngrams/books/datasetsv2.html). 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](http://commondatastorage.googleapis.com/books/syntactic-ngrams/syntngrams.final.pdf). A more popular description is available [here](http://commondatastorage.googleapis.com/books/syntactic-ngrams/TODO). The dataset format and organization are detailed in the [README](https://docs.google.com/document/d/14PWeoTkrnKk9H8_7CfVbdvuoFZ7jYivNTkBX2Hj7qLw/edit?usp=sharing) 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 I created under my own 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 my HFS directory "/var/ngrams" you would run the "-ls" command, like so:

|  |
| --- |
| hadoop fs -ls /var/ngrams |

Worksheet #2: **What is the name of the last file in the listing for HFS folder /var/si618f17?**

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. In this case, we are searching the file(s) that end in -e (there happens to be only one in that folder).

|  |
| --- |
| hadoop fs -cat /var/ngrams/data | grep "^einstein\_NOUN" |

Worksheet #3: **What year was Einstein 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 a small sample of the HDFS NGram dataset stored in /var/si618f17/ngrams\_lab.

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

|  |
| --- |
| ./spark-run.sh ngram-job.py /var/si618f17/ngrams\_lab ./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 |

I've created this Unix script called **spark-run.sh** that saves you from typing most of the following command:

PORT="$(shuf -i 10000-60000 -n 1)"

spark-submit \

--master yarn-client \ # Run this as a Hadoop job

--queue si618f17 \ # Run on your\_queue

--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**. I'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:

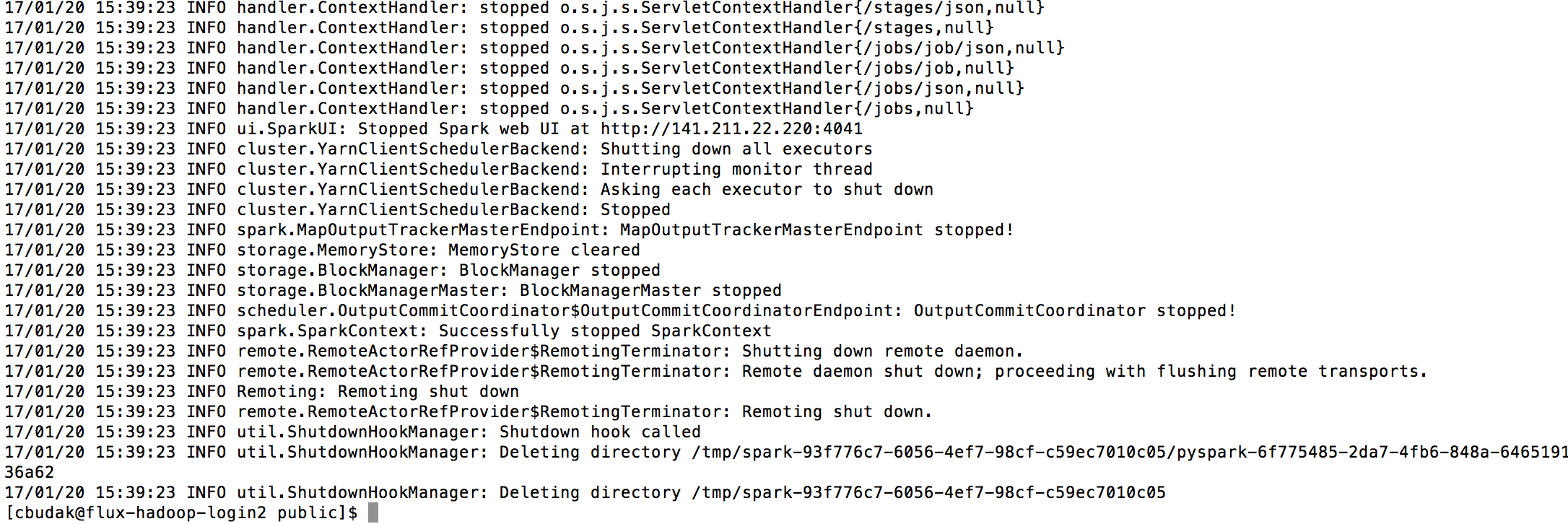
spark-submit --master yarn-client --queue si618f17 --num-executors 35 --executor-memory 5g --executor-cores 4 $1 $2 $3

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/si618f17/ngrams\_lab).
3. The Hadoop File System directory to store the output (./ngrams-out). This will create a new directory in your 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".



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 *words* 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 can *not* 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 |

Worksheet #4: **After the Spark job completes, what are three 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 |

and this will add together all the "part-" output files in your HDFS directory, and create a single regular file in your current directory called "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).

Worksheet #5: **What were the average word lengths observed in books from the years 1520, 1597, and 1598 ?**

### **6. Bonus Challenge**

Using the output file that is placed in your HFS output directory, produce a scatterplot of how average word length has changed over the full time period of the dataset (using your favorite spreadsheet or data viz program). Insert it below.

**What to submit:**

Submit a PDF file of your completed worksheet

si618\_f17\_lab3\_uniquename.pdf

## **References:**

* [UM CAEN Spark Tutorial](http://caen.github.io/hadoop/user-spark.html)
* Documentation for Hadoop is provided at: <http://caen.github.io/hadoop>. Additional documentation is available at<http://umarcts.bitbucket.org/presentations/hadoop/> Watch out for slides which have a blue down-arrow in the lower right corner. When you see such a slide, press the down-arrow key on your keyboard (potentially several times) to see the rest of the content.
* Hadoop User Guide at <http://caen.github.io/hadoop/user-hadoop.html>

## **Google NGram average word length script:**

# Import the necessary Spark library classes, as well as sys  
from pyspark import SparkConf, SparkContext  
import sys  
  
# Ensure that an input and output are specified on the command line  
if len(sys.argv) != 3:  
 print('Usage: ' + sys.argv[0] + ' <in> <out>')  
 sys.exit(1)  
  
# Grab the input and output  
input = sys.argv[1]  
output = sys.argv[2]  
  
# Create a configuration for this Spark job  
conf = SparkConf().setAppName('AnnualWordLength')  
  
# Create a context for the job. The context is used to manage the job at a  
# high level.  
sc = SparkContext(conf=conf)  
  
# Read in the dataset and immediately transform all the lines in arrays  
data = sc.textFile(input).map(lambda line: line.split('\t'))  
  
# Create the 'length' dataset as mentioned above. This is done using the next  
# two variables, and the 'length' dataset ends up in 'yearlyLength'  
yearlyLengthAll = data.map(  
 lambda arr: (int(arr[1]), float(len(arr[0])) \* float(arr[2]))  
)  
yearlyLength = yearlyLengthAll.reduceByKey(lambda a, b: a + b)  
  
# Create the 'words' dataset as mentioned above.  
yearlyCount = data.map(  
 lambda arr: (int(arr[1]), float(arr[2]))  
).reduceByKey(  
 lambda a, b: a + b  
)  
  
# Create the 'average\_length' dataset as mentioned above.  
yearlyAvg = yearlyLength.join(yearlyCount).map(  
 lambda tup: (tup[0], tup[1][0] / tup[1][1])  
)  
  
# Save the results in the specified output directory.  
yearlyAvg.saveAsTextFile(output)  
  
# Finally, let Spark know that the job is done.  
sc.stop()