

Lab 1 – MapReduce with Hadoop and AWS ElasticMapReduce

Assigned: 1/28/16

Due: 2/9/16 by 11:59pm

This lab will familiarize you with the MapReduce programming paradigm using the Hadoop framework. You will also learn how to use the ElasticMapReduce service provided by Amazon Web Services.

Make sure your VM is setup before starting this lab. If you haven't setup your VM, follow the guide on Canvas.

Part 1: Word counting on your local machine (20 pts)

For this part we will be using the MapReduce programming paradigm to count word occurrences in a book. We will be using the Hadoop framework for running our MapReduce programs. If you wish to learn more about Hadoop, visit their website <http://hadoop.apache.org/>

First, navigate to your Git directory and update it by issuing a “git pull”. Then, navigate to the lab1 folder inside your Git directory. You will see four folders and a file:

- **data** – this folder contains the data that you will be running your code on. I have given each person a different book in raw text format. The name of the file should be “book.txt”
- **results** – this folder is where you will put your results from running your MapReduce code. This will be explained in detail later on.
- **part1** – this folder contains two files: mapper.py and reducer.py. These are incomplete source files that you will complete for your word count implementation.
- **part2** – this folder contains two files: mapper.py and reducer.py. These are incomplete source files that you will complete in part 2 of this assignment.
- **run_hadoop.sh** – this script will run your MapReduce code on your local machine. This script will take an argument referring to which part of the lab to test. For this part, the argument is 1 (so to test your code you would type in: ./run_hadoop.sh 1)

Your job: fill out the mapper.py and reducer.py files in the part1 directory and implement a word count procedure. That is, your code must count how many occurrences of each word there are in the book given to you. Here are the rules your implementation must follow:

- A “word” is any contiguous sequence of letters/characters/symbols that ends with a space. For example: “hello 245 k&f’fln” is 3 words (the quotes are not part of the example).

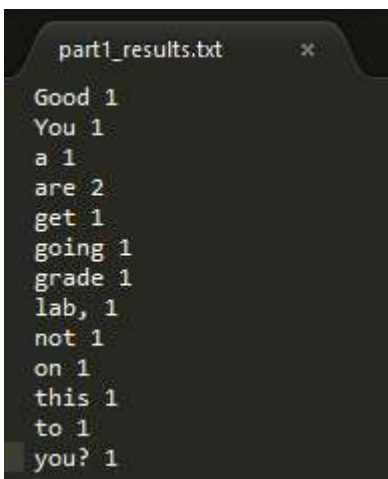
- A “word” cannot begin with a space or a tab. This means that multiple spaces after a word do **NOT** affect the word count. For example: “hi bye” is only 2 words even though there are many spaces between the words.
- Words are **CASE SENSITIVE**. For example, “MapReduce” and “mapreduce” are two different words
- As an added twist, your implementation must resolve some contractions into their constituent parts. For example: “I’m hungry” should be interpreted as the three words “I am hungry”. Here are the contractions you must resolve:
 - I’m = I am
 - I’ve = I have
 - Aren’t = Are not
 - Can’t = Can not
 - Didn’t = Did not
 - Couldn’t = Could not
 - Shouldn’t = Should not
 - Isn’t = Is not
 - You’re = You are

Note that contractions are **ALSO** case sensitive. That is, “You’re” resolves to “You are” but “you’re” resolves to “you are”. Any contractions that are not on this list should **NOT** be broken down and should be counted as whole words.

Your results should be submitted in a file called “part1_results.txt” that contains an **alphabetical** line-wise list of every word followed by a space and then the word-count of that word. For example, consider the following (extremely motivating) sentence:

“You’re going to get a Good grade on this lab, aren’t you?”

This should generate a file which looks like this:



```

part1_results.txt
Good 1
You 1
a 1
are 2
get 1
going 1
grade 1
lab, 1
not 1
on 1
this 1
to 1
you? 1
  
```

Put this file in the “results” directory when you are finished.

Some Notes:

1. Hadoop's core implementation is in Java. As self-respecting humans, we will avoid Java and use Hadoop's **streaming** feature to allow us to write Python for our MapReduce programs. Streaming works by using the standard input and output pipes in Linux. What this means for you is that your mapper program (mapper.py), will see a bunch of lines coming in from the standard input (stdin). You will not have access to the data file directly, but rather on a line by line basis. The shell code given to you currently iterates over all the input lines, and I recommend that you do not modify that behavior. The job of the mapper is to print out lines as its output, which are then read in automatically by the reducer. To learn more about streaming, go to <http://hadoop.apache.org/docs/r2.6.3/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html>

2. You can use the run_hadoop.sh script to test your code. To do so, make sure you are in the lab1 directory and issue the following command:

```
./run_hadoop.sh 1
```

This will dump a lot of text onto your console and (if successful), will create a directory called "output". This directory will have a file called "part-00000", which is the result of your MapReduce job.

3. You first need to allow the run_hadoop.sh script to run. To do this, issue the following command in the terminal (make sure you are in the lab1 directory first):

```
chmod +x run_hadoop.sh
```

4. Be careful when debugging. Since Hadoop relies on the standard input/output, debug "print" statements will screw up your job! If you want to print debug information, using the following print format:

```
print >> sys.stderr, "Hello World"
```

This will print to the standard error pipe (which is visible on the terminal but is not used by Hadoop).

Part 2: Bigram analysis on your local machine (50 pts)

For this part you will be analyzing bigrams in the book given to you. A bigram is simply a sequence of two words. Let's say we have the following input string:

"This lab is fun"

The bigrams for this sentence are:

- "This lab"
- "lab is"
- "is fun"

Your job: For the book given to you, compute the following metrics:

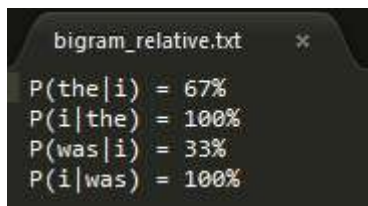
1. Unique bigram count. How many unique bigrams are there? Note that bigrams are **NOT** case sensitive. “Hello world” and “hello World” are the **same** bigram. Put your answer in a file called “bigram_count.txt”
2. Bigram occurrence count. List the top ten bigrams and their counts. Put your answer in a file called “bigram_topten.txt”. For instance: “hello world 10” (without the quotes).
3. Compute bigram relative frequencies. That is, compute how likely you are to see a word given the preceding word. Think of this as “the probability of seeing word X given that I have seen word Y”, or $P(X|Y)$. For example, consider the following (nonsensical) sentence:

“i the I was i the”

For this sentence, here are all of the relative frequencies:

- $P(\text{the}|\text{i}) = 67\%$
- $P(\text{i}|\text{the}) = 100\%$
- $P(\text{was}|\text{i}) = 33\%$
- $P(\text{i}|\text{was}) = 100\%$

Report all the probabilities in the same form shown above (round percentages to nearest integer). Put your results in a file called “bigram_relative.txt”. **For this task, all words should be in lowercase.** Just for reference, your file should look like this:



```
bigram_relative.txt
P(the|i) = 67%
P(i|the) = 100%
P(was|i) = 33%
P(i|was) = 100%
```

Some Notes:

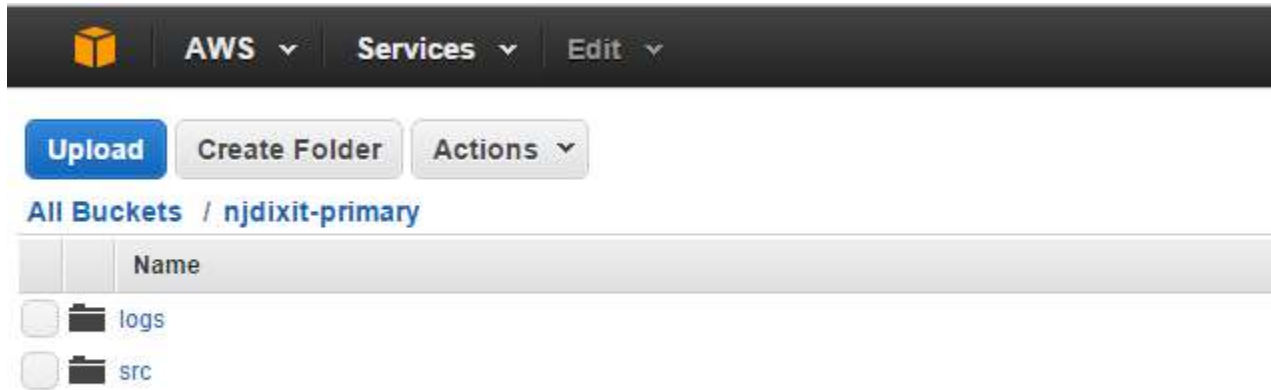
1. For this part write your code in the “part2” subdirectory. You may have to write multiple MapReduce programs. This is fine, but make different files for each MapReduce program. Note that you will have to modify the run_hadoop script to take your filename changes into account.
2. This part will take argument 2 for the run_hadoop script:

```
./run_hadoop.sh 2
```

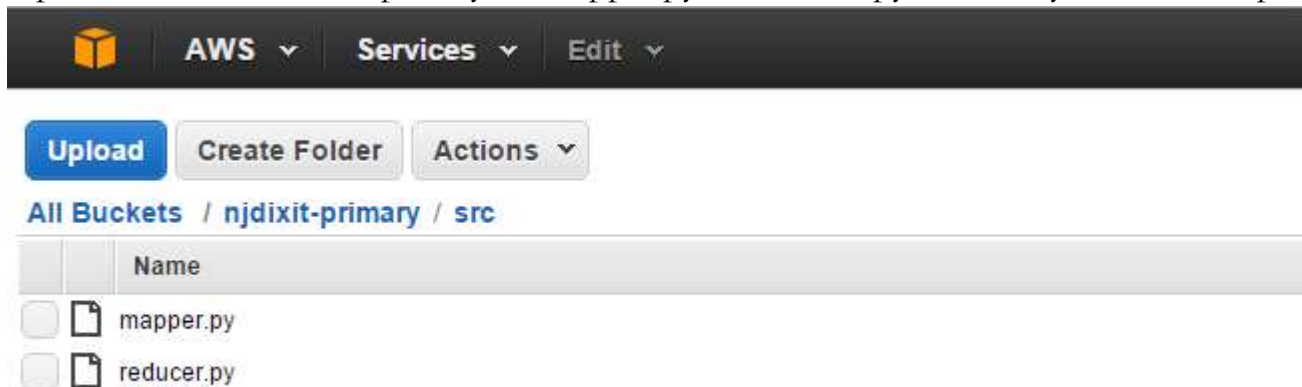
Part 3: Word Counting on AWS ElasticMapReduce (30 pts)

For this part you will be running your word count MapReduce job on an Amazon ElasticMapReduce cluster using a dataset that lives on Amazon S3.

1. Make sure you have sent me your AWS ID. If you have not, you will not be able to access the S3 dataset and will not be able to complete this part.
2. Also make sure that you have applied your credits to your account.
3. Logon to <http://www.aws.amazon.com> with your credentials.
4. The first thing we need to do is create an S3 bucket to store your input data, source code, and logs/results. Go to the dashboard and click on S3.
5. Create a bucket. You can name it whatever you wish. For region, choose “US Standard”
6. Once the bucket is created, open it and create two folders: src and logs



7. Open the src folder and upload your mapper.py and reducer.py files that you wrote for part 1.



8. Now, click on “Services” and choose “EMR”
9. On the EMR page, click on “Create cluster”
10. Fill out the following for general configuration details:
 - Cluster name – whatever you wish.
 - Logging – check
 - S3 folder – the S3 “logs” folder you made in step 6.
 - Launch mode – step execution

General Configuration

Cluster name

☒ Logging ⓘ

S3 folder

Launch mode ☐ Cluster ⓘ ☒ Step execution ⓘ

11. Fill out the following for the “Add Steps” section:

- Choose the “Streaming program” step and click “Configure”
- In the configuration menu that pops up, fill out the following:
- Name – whatever you wish
- Mapper – the mapper code you uploaded in step 7
- Reducer – the reducer code you uploaded in step 7
- Input S3 location - s3://ut-ee379k-labs/lab1/data/book.txt
- Output S3 location – s3://your-bucket-name/results
 - DO NOT create this folder ahead of time! Hadoop creates it for you
- **Action on failure – Terminate cluster**
 - **This is EXTREMELY important. Do NOT leave your instances running.**

Add Step ✕

Step type Streaming program

Name*

Mapper* S3 location of the map function or the name of the Hadoop streaming command to run.

Reducer* S3 location of the reduce function or the name of the Hadoop streaming command to run.

Input S3 location*
s3://<bucket-name>/<folder>/

Output S3 location*
s3://<bucket-name>/<folder>/

Arguments

Action on failure What to do if the step fails.

Cancel Add

- Click on Add

12. Click on “Create cluster”

Your cluster will now run. It usually takes a while (~10 minutes) for the cluster to spool up. The status page will keep you informed on the status of the job. Once the job is finished, make sure that your clusters are terminated and download the result from the “results” folder in your S3 bucket. Note that there may be multiple parts. Please combine all of these into one file (you don’t have to re-alphabetize, just concatenate them) and save the final file into your Git “results” directory as “aws_wordcount.txt”

Ensure once again that your clusters are terminated:

Network and Hardware

Availability zone: us-east-1c

Subnet ID: [subnet-4635f51e](#)

Master: Terminated 1 m3.xlarge

Core: Terminated 2 m3.xlarge

Task: --