CSE4334/5334 Data Mining
Spring 2015, Prof. Chengkai Li
Department of CSE, University of Texas at Arlington
Programming Assignment 3 (P3)
Due: 11:59pm April 16th (Thursday), 2015

In this assignment, you will write MapReduce programs in Python. You will use Apache Hadoop, an open-source implementation of Google's proprietary MapReduce system. Since we don't have a cluster to use for this course, you will set up a single-node Hadoop environment on your own personal computer. The programs you write will work in a real cluster. It is just that in the single-node setup you won't be able to observe performance advantage against a centralized system. In fact, you will observe worse execution efficiency, since the overhead of Hadoop environment cannot pay off in a single-node setup.

Setting up your own Hadoop environment can be non-trivial, even if it has only one node. To avoid the hassle, we will use a virtual machine. The virtualization software we will use is VMware Player. The Hadoop-ready virtual machine we will use is Hortonworks Sandbox. There are other virtualization software and Hadoop virtual machines. Our following discussion is based on the setup of VMware Player + Hortonworks Sandbox.

You need quite large free space on your hard drive. 5GB for a virtual machine file, 11GB for the virtual machine loaded from the file, and at least 400MB for the files produced during running your programs.

In following the instructions in this document, make sure you type in exactly what the instructions say. If something doesn't work as expected, look for typos first. If you also copy & paste from this document, it will reduce typos.

1 What to Submit

Submit the following files through Programming Assignment 3 (P3)'s entry in Blackboard.

Task 1 (Section 6): submit `speakerwcMapper.py` and `speakerwcReducer.py`

Task 2 (Section 7): submit the mappers and the reducers, in 6 files, named `step1_mapper.py`, `step1_reducer.py`, `step2_mapper.py`, `step2_reducer.py`, `step3_mapper.py`, and `step3_reducer.py`, respectively. That will be the outcome of following the "Hints" in Section 7 and assuming the same commands of executing the programs as listed in Section 7. It is okay if you choose a different method. In that case, submit all python files and a "`README.txt`" that lists the commands to run your program.

2 Program and Data Files

In this assignment, we will use the following program and data files. They are provided in a ZIP file that can be downloaded from the Programming Assignment 3 (P3) entry in Blackboard.

* *debates*: This folder has 30 files that contain the transcripts of all presidential debates in history. Each file is for one debate. Each line is in the form of *speaker : sentence*.
* *wordcount*: This folder contains `mapper.py` and `reducer.py`--- Python files for the word count example.
* *graphs*: This folder contains 3 files, `toy-graph.txt`, `small-graph.txt` and `web-Google.txt`. They are three graphs with different sizes.

3 Setting up the Environment

3.1 Enable BIOS Support for Virtualization

Here is an example of how to do it: http://bit.ly/1ncbAqk

This probably is only necessary if you have a PC. It appears to be irrelevant to Mac, but I couldn't verify. If you encounter troubles using an Mac, this page might have some useful information for you: http://bit.ly/1BZoaKe.

3.2 Download and Install VMware Player 7 from http://vmw.re/1J8x5At

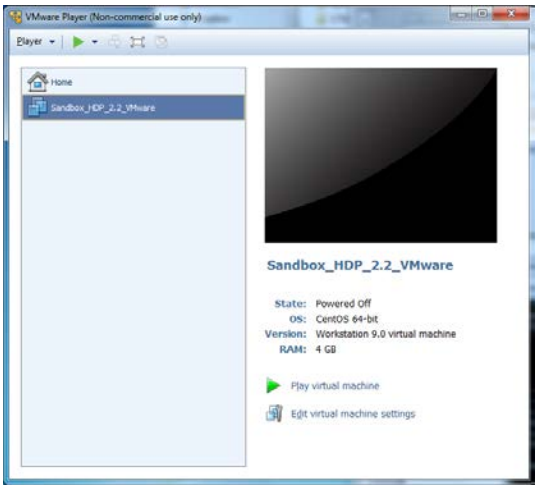
VMware Player 7 is what I used and tested for this assignment. Other versions may work as well. For instance, if you have an Mac, it seems that you need to use VMware Fusion.

3.3 Download Hortonworks Sandbox HDP 2.2 from http://bit.ly/19xTc4B

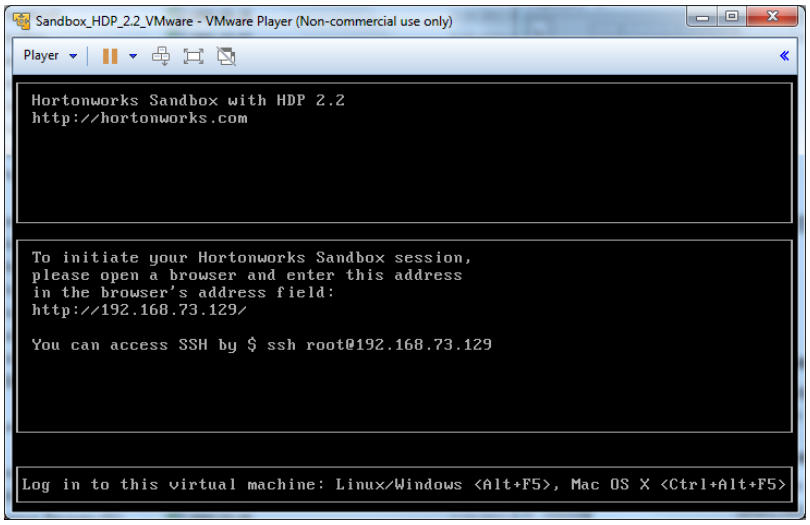
According to this page, this virtual machine should work on 32-bit and 64-bit OS (Windows XP, Windows 7, Windows 8 and Mac OSX).

3.4 Install Hortonworks Sandbox HDP 2.2 in VMware

From the above page, you can see that, for VMware, they only have "Install Guide" for Mac, at http://bit.ly/19tJfpK. But the steps for Windows are pretty similar. You don't really need a guide. Just "Open" the file Sandbox_HDP_2.2_VMware.ova in VMware. After a while, you should be able to see the following screenshot:



The virtual machine with Hadoop environment is ready. Just click the Play (Power on) button or click “Play virtual machine” to turn on the guest OS. After a while, you should see the following screenshot:



As said in the above screen, you can log into the virtual machine by pressing Alt+F5 or Ctrl+Alt+F5 in the virtual machine. But we will remotely log in using SSH, so that you can log in through multiple consoles, which will make programming and debugging easier.

You can turn on and off this virtual machine, as if it is a real computer. And the files will not be lost. If it becomes necessary, you can delete the whole virtual machine, and load the Hortonworks Sandbox again to get a fresh virtual machine and start from scratch.

4 Test Example

To verify the environment is working, we will run the classic word count example.

4.1 Transfer data files into the local file system of the guest OS

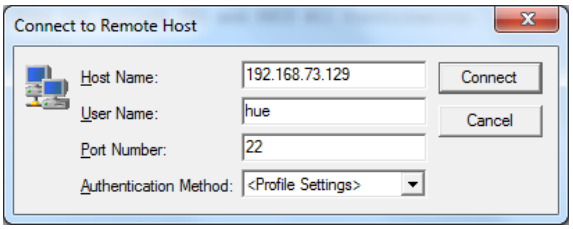
Use SSH and SFTP tools to upload all files mentioned in Section 2 into the guest OS with the Hadoop environment. Login information as follows:

IP address: 192.168.73.129 (you may need a different IP. Refer to the screenshot you get, similar to the above one.)
Username: hue (you can also login as “root”. But I assume “hue” from now on”.)
Password: hadoop

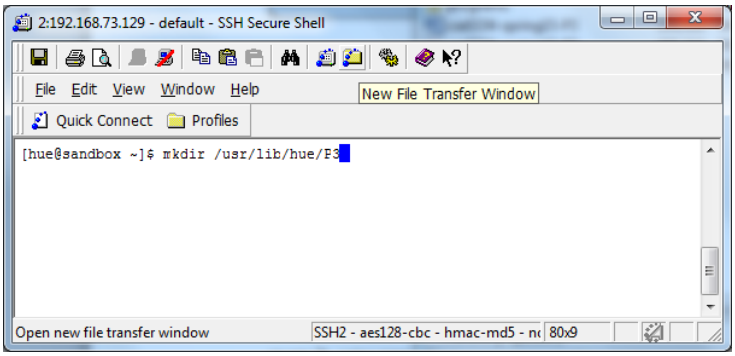
We now explain the detailed steps in Windows, using SSH Secure Shell Client and SSH Secure File Transfer which can be downloaded from http://www.uta.edu/oit/cs/unix/ssh/Secure-Shell-Client.php. There are similar tools on Mac and Unix/Linux. And you can also use command line.

(a) Login to the guest OS using SSH Secure Shell Client. Below is the screenshot of my SSH Secure Shell Client that is about to the make the connection.

Below is the screenshot of my SSH Secure File Transfer that is about to connect to the guest OS.



After I press "connect" and enter "hadoop" as the password, I get the following screenshot:



(b) Create a folder `/usr/lib/hue/P3` by typing the following command:

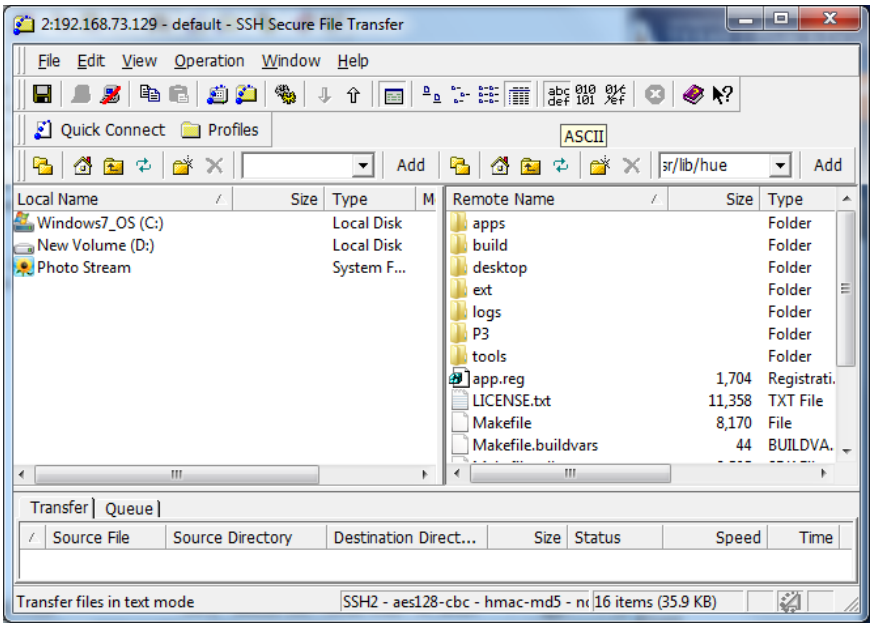
```
mkdir /usr/lib/hue/P3
```

From now on, I assume you are always in folder `/usr/lib/hue/P3`. If the commands in ensuing discussion do not work, make sure to go to the right folder by command:

```
cd /usr/lib/hue/P3
```

(c) Open SSH Secure File Transfer.

Notice the "SSH Secure File Transfer" icon in the above screenshot? When you move your mouse over it, you see "New File Transfer Window". Click on it, you will get the following window:



Now you can figure out how to transfer files into the guest OS. From now on, I assume all the folders and files mentioned in Section 2 are uploaded to folder `/usr/lib/hue/P3`. Make sure to use ASCII mode to transfer all data files and source codes in this assignment. (Click the icon labelled as "ASCII" in the above screenshot, before you transfer the files.)

4.2 Create HDFS data folder and prepare data files

Go back to the SSH Secure Shell Client. Type the following commands:

`hdfs dfs -mkdir debates` (Create a folder named "`debates`" in Hadoop Distributed File System (HDFS) for input data files.)

`hdfs dfs -put /usr/lib/hue/P3/debates/*.txt debates` (Copy 30 debate files from the local file system to the created HDFS folder)

4.3 Run an Hadoop program

The guest OS already has a word count example implemented, in a JAR file `hadoop-mapreduce-examples.jar`.

Type the following command:

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar wordcount debates wordcount_output
```

The command will run word count on the 30 presidential debate files in HDFS folder "`debates`". The output will be stored in HDFS folder "`wordcount_output`". During its execution, you will see various messages. When it is done, you will see "map 100% reduce 100%" and some statistics about its execution.

Now, let's see what output files were produced in the output folder:

```
hdfs dfs –ls wordcount_output
```

You should see a file named `part-r-00000` in the folder. Let's see its content:

```
hdfs dfs –cat wordcount_output/part-r-00000
```

It is a long file. So you might want to see it page by page. Type the following command instead.

```
hdfs dfs –cat wordcount_output/part-r-00000 | more
```

You will see some special characters. That's due to noises in the original files we downloaded. You don't need to worry about it.

Note: If you want to run the above command again, you need to first delete `wordcount_output` from HDFS, by the following command. The same applies to our programs you are about to write.

```
hdfs dfs –rm -r -f wordcount_output
```

Even though the Hadoop environment allows you to run multiple Hadoop jobs at the same time, it is unnecessary in this assignment and may freeze your guest and host OS, due to limited resources.

In case the virtual machine appears to be very slow, you can use the following commands to list current Hadoop jobs and kill some.

```
hadoop job -list
```

```
hadoop job –kill <jobID>
```

5 Hadoop Programs in Python

Hadoop was implemented for Java programs. However, it also has a way to support programs in Python, through Hadoop streaming. In this step, we will do Word Count, using Python.

(The following page explains Hadoop streaming. It can be handy.

http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html)

(4.1) By following the instructions in Section 2, you should already have `mapper.py` and `reducer.py` in folder `/usr/lib/hue/P3/wordcount`.

(4.2) Execute the following commands:

```
chmod +x /usr/lib/hue/P3/wordcount/*.py
```

This makes sure you can run `mapper.py` by just command `mapper.py` (without the longer form `python mapper.py`). This is enabled by the shebang in the first line of `mapper.py`---`#!/usr/bin/env python`. The same for `reducer.py`.

Transferring the files in ASCII format from your host OS' file system into the guest OS' file system was critical. If you did't use ASCII, Unix line ending of the files might be destroyed, which will make the shebang not working properly.

(4.3) One nice thing about using Python for Hadoop is that we can even test it without Hadoop environment. Run the following command. It uses Unix pipes to connect `mapper.py` and `reducer.py` with standard Linux commands `cat` and `sort`.

```
cat debates/*.txt | wordcount/mapper.py | sort -k1,1 | wordcount/reducer.py
```

What do you see in the output? Word count results!

If this doesn't work, it is likely due to shebang not working properly. You can use the following commands instead:

```
cat debates/*.txt | python wordcount/mapper.py | sort -k1,1 | python wordcount/reducer.py
```

(4.4) Now run the Python programs in Hadoop.

Execute the following command to remove the current HDFS folder wordcount_output:

```
hdfs dfs -rm -r -f wordcount_output
```

Then run the Python Hadoop program:

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -D stream.map.output.field.separator="\t" -files wordcount/mapper.py,wordcount/reducer.py -input debates -output wordcount_output -mapper mapper.py -reducer reducer.py
```

(As the parameters in the command indicate, the input files are in HDFS folder `debates` and the output files will be placed in HDFS folder `wordcount_output`).

Remember to use the following commands to find out the output file name and check its content:

```
hdfs dfs –ls wordcount_output
```

```
hdfs dfs –cat wordcount_output/part-00000
```
(Note that the file name is different from the one produced by the Java wordcount program in Section 3.)

6 Task 1: Counting Keywords Mentioned by Speakers in Presidential Debates

This task will require some simple adaptation of the word count example in Section 4. It also prepares you for the more challenging Task 2 in Section 6.

What to submit: submit the mapper and the reducer, in 2 files, named speakerwc`Mapper.py` and `speakerwcReducer.py`, respectively.

Problem to solve: In this question, you will use Python to write a mapper and a reducer that will execute in Hadoop streaming. The program counts the frequencies of keywords mentioned by individual speakers in presidential debates, based on the 30 data files described in Section 2. Each result is a tuple <`speaker, word, count`>, where `count` is the number of times the `speaker` has mentioned the `word` in all debates which they have participated in.

-- Only return those tuples where count > 200.

-- No stemming is required.

-- The whole program should be case-insensitive. I.e., "Peace" and "peace" should be considered the same term.

-- Remove all stopwords. We may be able to install NLTK in the virtual machine. But let's simply use the following list of stopwords.

```
stopwords = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other",
```

```
"some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too",
"very", "s", "t", "can", "will", "just", "don", "should", "now"]
```

Expected Results: The correct output is:

```
 Barack Obama    going   321
Barack Obama    make    272
Barack Obama    that's  253
Barack Obama    we've   218
Bill Clinton    people  244
George Bush     think   223
George W. Bush  it's    249
George W. Bush  people  246
George W. Bush  think   234
Gerald R. Ford  uh      359
Jim Lehrer      president       296
Jimmy Carter    uh      509
John Kerry      president       243
```

Hints:

(1) Composite keys

The correct way of solving this problem is to make the mapper producing key-value pairs where the keys are composite and have two fields <speaker, word>. Suppose Obama used "health" in one sentence. Then the corresponding key-value pair produced the mapper would be (<"Barack Obama", "health">, 1). The reducer then also uses <speaker, word> as the key.

More specifically, the mapper can produce a key-value pair by `print('%s\t%s\t%s' % (speaker, word, 1))`. To make sure `<speaker, keyword>` together (instead of `speaker` only) is treated as the key, you need to tell Hadoop, by running the program using the following command.

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-
streaming.jar -D stream.map.output.field.separator="\t" -D
stream.num.map.output.key.fields=2 -D
mapreduce.partition.keypartitioner.options=-k1,2 -files
speakerwc/speakerwcMapper.py,speakerwc/speakerwcReducer.py -input
debates -output speakerwc_output -mapper speakerwcMapper.py -
reducer speakerwcReducer.py
```

Compared with the command in Section 4 from running the word count example, there are 2 new options. `-D stream.num.map.output.key.fields=2 -D mapreduce.partition.keypartitioner.options=-k1,2` They tell Hadoop to use

the first 2 fields of each line from the output of the mapper as the key, in preparing the input to the reducer.

(2) Other options

Also pay attention to the option `-D stream.map.output.field.separator=\t` and make sure to use tab to separate the fields, i.e., `print('%s\t%s\t%s' % (speaker, word, 1))`.

Since we use the option `-output speakerwc_output`, you can find the output in HDFS folder `speakerwc_output`.

(3) Shell command

Before you run the Hadoop program, you can also use Linux shell commands to test your program:

```
cat debates/*.txt | speakerwc/ speakerwcMapper.py | sort -k1,2 | speakerwc/speakerwcReducer.py
```

Note that you need to use `sort -k1,2` instead of `sort -k1,1`.

7 Task 2: Counting Triangles in a Web Graph

What to submit: submit the mappers and the reducers, in 6 files, named `step1_mapper.py`, `step1_reducer.py`, `step2_mapper.py`, `step2_reducer.py`, `step3_mapper.py`, and `step3_reducer.py`, respectively. That will be the outcome of following the "Hints" below and assuming the same commands of executing the programs as listed later in this section. It is okay if you choose a different method. In that case, submit all python files and a "`README.txt`" that lists the commands to run your program.

Problem to solve: You are given a web graph stored in a tab-separated 2-column file. Each line represents a hyperlink, e.g., "101 12" represents a hyperlink from page 101 to page 12. Your job is to count the number of triangles in this directed graph. A triangle is formed by 3 hyperlinks: A->B, B->C, C->A.

It is guaranteed that none of the graphs given to you is a multi-graph, i.e., there doesn't exist more than 1 edge from any node A to any node B.

Expected Results: To help you verify your programs, we give you two graphs: small-web.txt and web-Google.txt. (This is a file used by Google programming contest. Provide link.) Here are some statistics on these two graphs, including the execution time of my programs on these 2 graphs.

Graph	# edges	# triangles
toy-graph.txt	9	2
small-graph.txt	1000	768
web-Google.txt	5105039	3889771

Hints:

Here is how I solved the problem: Use 3 MapReduce jobs in order to fully accomplish this task.

MapReduce Job 1: find all 2-hop paths A -> B -> C. The output should list all such paths, as well the original edges, but in reverse order. For example, if there is an edge C -> A, the output should list A -> C.

MapReduce Job 2: take the output of Job 1 as input, use the first 2 fields as key, so that "A C B " (representing A->B->C; yes, no typo here) and "A C" (representing C -> A; again, there is no typo) appear together. Such two edges will give us a triangle. The output should be in the form of "1 A C 2" (or simply "1 2"). The first field is always the same. The last field is a count. For example, if the output of job 1 contains "A C", "A C B", and "A C D". It means there are 2 triangles going from A through another node to C which goes back to A.

MapReduce Job 3: take the output of Job 2 as input. It simply sums up all the counts, since all lines in the output of Job 2 use "1" as the key.

To make it more efficient, you don't want to count the same triangle 3 times. Think about how to ensure that. If you don't take care of that, the number you will get will be 3 times the real count of triangles. That itself is not a problem (as long as you know that you should divide it by 3). But your program will be 3 times less efficient, at least.

Here are the commands I used when I execute my program, with some explanation.

```
hdfs dfs -rm -r -f triangle_output
hdfs dfs -mkdir triangle_output
```

(This is necessary, because the output directories of the 3 steps are subdirectory of "triangle_output".)

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar
-D stream.map.output.field.separator="\t" -D
stream.num.map.output.key.fields=1 -D
mapreduce.partition.keypartitioner.options=-k1,1 -files
triangle/step1_mapper.py,triangle/step1_reducer.py -input graphs/web-
Google.txt -output triangle_output/step1 -mapper step1_mapper.py -reducer
step1_reducer.py
```

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar
-D stream.map.output.field.separator="\t" -D
stream.num.map.output.key.fields=2 -D
mapreduce.partition.keypartitioner.options=-k1,2 -files
triangle/step2_mapper.py,triangle/step2_reducer.py -input
triangle_output/step1/part-00000 -output triangle_output/step2 -mapper
step2_mapper.py -reducer step2_reducer.py
```

(Note that we need to use the first 2 fields as the composite key in step 2.)

```
hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar
-D stream.map.output.field.separator="\t" -D
```

```
stream.num.map.output.key.fields=1 -D
mapreduce.partition.keypartitioner.options=-k1,1 -files
triangle/step3_mapper.py,triangle/step3_reducer.py -input
triangle_output/step2/part-00000 -output triangle_output/step3 -mapper
step3_mapper.py -reducer step3_reducer.py

hdfs dfs -cat triangle_output/step3/part-00000
```

(This displays the output of Job3, which is a single number.)

8 Beyond

Triangle counting has a lot of important applications. While counting triangles, people are often interested in viewing the graph as an undirected one (i.e., ignoring edge directions if they were directed). Counting undirected triangles is just a little more challenging and more time-consuming than our Task 2. If you are fascinated by it, you may want to do it on your own. Our web-Google.txt has 13391903 undirected triangles, without considering mutli-graph.