ECE 575
BIG DATA ANALYSIS
Fall 2016

ASSIGNMENT 1

Due Date: *Sunday, November 13th, 2016, 23:59*

Assignment Submission: *Turn in your assignment by the due date through LMS. No late submissions will be accepted. Prepare and upload* ***one zip file*** *that should have* ***one subdirectory for each question****. Name the file as <your first name>_<your last name>_ assignment1. Name the subdirectories as Q<question number>. See individual questions for the list of what you should return.*

You can and encouraged to discuss Hadoop environment, how to run programs and MapReduce JAVA API with each other. ***However, all work in questions (algorithm and implementation) must be your own; you must neither copy from nor provide assistance to anybody else. If you need guidance for any question, talk to the instructor.***

The objective of this assignment is to get familiar with Hadoop environment and MapReduce JAVA API and subsequently write and execute MapReduce programs.

An easy way to start running Hadoop jobs is to use a virtual machine (VM). If you prefer, you can also install Hadoop yourself into your computer (see http://hadoop.apache.org/) or use a Hadoop cluster that you may have access to.

VM will be running a single node Hadoop cluster in your computer. However, as you will see later, you can still have multiple map and reduce tasks.

Cloudera provides VMs (Cloudera Quickstart VM) that come pre-installed with their Hadoop releases. In order to use Cloudera Quickstart VM, download and uncompress the virtual machine image from the following link.

http://www.cloudera.com/content/cloudera/en/documentation/DemoVMs/Cloudera-QuickStart-VM/cloudera_quickstart_vm.html

You will need a virtualization software (i.e., VirtualBox, VMware) to run VM in your computer. The following steps are for VirtualBox. You can use VMware as well. Make sure to get the correct version of Cloudera Quickstart VM.

Download and install VirtualBox from the following link for your operating system.

https://www.virtualbox.org/

In VirtualBox menu, follow File→Import Appliance to load Cloudera Quickstart VM. VM Memory is configured to 4GB. If your computer has only 4GB, lower it before your start (Settings→ System, 2GB seems to work fine in my computer). If your computer have a larger memory, consider increasing it.

Select Cloudera Quickstart VM and click 'Start' to start the VM.

Once VM started, if your keyboard layout is different than default setting, you can change it at:

System→ Preferences→Keyboard→ Layouts→ select the proper keyboard layout

Start a terminal and you are ready to go. User name is 'cloudera' and home directory is '/home/cloudera'. Try 'hadoop version' as your first command. Now, let's go through the steps to run a MapReduce program.

Open an internet browser (i.e. firefox) and go to ECE 575 LMS folder. Under MyBox→Assignments→ Assignment 1, you fill find various files for this assignment.

Download WordCount.java from LMS as an example MapReduce program (I suggest you create a directory under the home directory and save it there). It implements the algorithm to find the occurance frequency of each word in a dataset as we discussed in the class.

In order to execute a MapReduce program, first compile it:

```
mkdir wordcount
javac –cp /usr/lib/hadoop/*:/usr/lib/hadoop/client-0.20/* -d wordcount WordCount.java
```

And create a JAR file:

```
jar –cvf wordcount.jar –C wordcount/ .
```

You need data to run the program on. Create a few small test files:

```
echo 'aaa bbb ccc aaa' > file_test_1
echo 'bbb ddd aaa' > file_test_2
echo 'bbb ddd bbb e b' > file_test_3
```

These files are at your local file system. You need to put these files to HDFS (Hadoop Distributed File System). HDFS shell is invoked by 'hadoop fs' command. Run it to see the command options.

hadoop fs : will show the command options

As you will notice, many options are similar to Unix commands. A few examples you will use frequently:
hadoop fs –ls : list files. If you don't specify any path, it will list HDFS home (/user/cloudera). Note that HDFS directory structure is different than your local file system. Check whether /user/cloudera exists in your local file system.
hadoop fs –mkdir : create directory
hadoop fs –rmdir : remove directory

hadoop fs –rm : remove file. –r option exists same as Unix.
hadoop fs –cp : copy files within HDFS
hadoop fs –cat : prints the content of input files to screen (stdout)
To copy from the local file system to HDFS:
hadoop fs -put
hadoop fs -copyFromLocal

To copy to the local file system from HDFS:
hadoop fs -get
hadoop fs -copyToLocal

Let's put the sample files to HDFS:

hadoop fs –mkdir wordcount
hadoop fs –mkdir wordcount/input
hadoop fs –mkdir wordcount/input/test
hadoop fs -put file_test_* /user/cloudera/wordcount/input/test
hadoop fs -ls /user/cloudera/wordcount/input/test
hadoop fs -cat /user/cloudera/wordcount/input/test/*

Once the files are in HDFS and JAR file is ready, use the following command to execute your first MapReduce program on Hadoop:

hadoop jar wordcount.jar org.mapreduce.wordcount.WordCount
/user/cloudera/wordcount/input/test /user/cloudera/wordcount/output

MapReduce program runs on the files located in /user/cloudera/wordcount/input/test directory and the reduce outputs are put into /user/cloudera/wordcount/output directory. One output file is created for each reduce task. Output file names start with "part-" and the numbers in file names are partition numbers (Remember partitioning step we discussed in class). List and view the content of output files:

hadoop fs -ls /user/cloudera/wordcount/output
hadoop fs -cat /user/cloudera/wordcount/output/part*

If you run the command again, Hadoop will not override the existing directory and error out. Delete it first if you would like to output to the same place.
hadoop fs -rm -r /user/cloudera/wordcount/output

Check the messages printed on screen when running WordCount program. What was the number of map and reduce tasks? What was the size of map/reduce/combine input/output records?

You will notice that combiner was not used. You can enable it by uncommenting job.setCombinerClass(Reduce.class) in JAVA code. Reduce function (implemented in Reduce class) is re-used as Combine function in this particular case (Combine function can be different than Reduce function). Re-compile & re-create JAR file & run and check the run log again.

You can change the number of reduce tasks by job.setNumReduceTasks(N) statement in JAVA code (Default is one reduce task). Try different numbers and check log.

Now, try this program with large files. Download three britannica* file from LMS (under Question 1 directory) and load them to HDFS and run WordCount on them. Try with/without combiner and different reducer count and analyze the printed messages.
hadoop fs -mkdir /user/cloudera/wordcount/input/britannica
hadoop fs -copyFromLocal britannica* /user/cloudera/wordcount/input/britannica
hadoop jar wordcount.jar org.mapreduce.wordcount.WordCount
/user/cloudera/wordcount/input/britannica /user/cloudera/wordcount/output

Once you are comfortable with environment and running program, analyze JAVA file. Understanding classes in MapReduce program, map/reduce input/output types, input/output file types, … should be your main objective in this assignment. There are numerous sources on the internet. A couple of them can be found at:

http://www.drdobbs.com/database/hadoop-writing-and-running-your-first-pr/240153197?pgno=1 (quick 2 page summary)

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

As you see in the code, map and reduce functions extend from Mapper & Reducer classes and the data types for input/output (key, value) pairs are defined. Hadoop provides various data types: Text (string), LongWritable (long integer), IntWritable (integer), BooleanWritable (boolean), etc.. that implement Writable interface.

However, as we have seen in numerous examples in class (natural join, matrix multiplication, average,…), we frequently need to output complex data types as key or value (i.e., more than one primitive value). *You are expected to write your own custom data types in* ***Question 2 and 3*** *in this assignment to hold the complex data*. ***Please investigate and learn how to do write custom data types.***

Hint: You need to write a class that implements Writable interface. You need to override *write()* and *readFields()* functions. If you are going to write this custom data type to output file (in text output format), you need *toString()* function as well. Following is a good source to start with:

https://developer.yahoo.com/hadoop/tutorial/module5.html

If you are interested, you can try Eclipse IDE to write and debug your MapReduce codes. Eclipse has been already installed in VM. See following link to get started:

http://blog.cloudera.com/blog/2013/08/how-to-use-eclipse-with-mapreduce-in-clouderas-quickstart-vm/

QUESTION 1 ***(15 points)***

Character Count. We discussed in class a MapReduce algorithm to find out how many times each unique *word* occurs in a dataset. In this question, similarly, develop an algorithm to find out how many times each unique ***character*** occurs in a dataset (ignore the spaces in the dataset). Please write a JAVA code to implement this algorithm. *Note that this program will need just minor changes to WordCount program.* Enable the combiner (you should be able use the reduce function you wrote as combiner), set reduce task count to 4, use three britannica* files that you downloaded as dataset and run your program on this dataset.

Please return (in Q1 directory):

- *Java file (name it CharCount.java)*
- *JAR file*
- *Output files from your run on the given dataset consisting of britannica* files*
- *Copy of the messages printed on screen when running your code (name it Q1.log)*

QUESTION 2 ***(35 points)***

City Average Incomes. You are given a dataset where each line has a city name and the income of a person in this city (in thousands, rounded to the closest integer) such as 'city10 41'. The names have been removed for privacy reasons and all incomes are a range from 0 to 100 (integer only). You would like to calculate and report ***average*** income for each city.

Note that this is a grouping and aggregate operation. Grouping is done based on first column (city name) and aggregate function is average. Please implement a one-step MapReduce algorithm to find the average income per city. **Your implementation should have a combiner**. **Do not** transmit all individual income numbers per city to the reducers in order to calculate the average value.

Please note that we already discussed how we implemented group & average operation in class. Since there is only a small set of possible income numbers (0…100), a combiner can substantially reduce the size of data sent to the reducers.

You will need to output more than one value in the value field of map/reduce functions. ***You are required to use a custom data type that encapsulates them in this question.*** Download four income* files from LMS (in Question 2 directory) and use them as your dataset. Enable the combiner, set reduce task count to 4 and run your program on this dataset.

Please return (in Q2 directory):

- *Pseudo code of your algorithm*
- *Java file (name it GroupAvg.java)*
- *JAR file*
- *Output files from your run on the given dataset (income* files)*
- *Copy of the messages printed on screen when running your code (name it Q2.log)*

QUESTION 3 ***(50 points)***

Twitter Influence Score. You are given a dataset where each line has a pair of Twitter user names such as 'user1 user2' that indicates that ***user1 is following user2***. You are asked to design a method to score each user based on their influence. You decided to simply calculate each user's influence score based on the total number of followers of the user (the number of people that are following the user). However, in order to assess the influence of a user outside of his/her friend/family circle, you decided to count a follower only if the user is **not** following this follower. In other words, if two users are following each other, they don't have any effect on the influence scores of each other. Please design and implement a one-step MapReduce algorithm to find the user influence scores.

Your implementation **should** have a combiner. At minimum, if an individual map task finds out that two users are following each other, the corresponding data should not be sent to the reducers in order to eliminate unnecessary data transfer.

If you need to output more than one value (such as user name, count, score, etc.) in key or value field of map/reduce function, use a custom data type that encapsulates them. Download three usergraph* files from LMS (in Question 3 directory) and use them as your dataset. Enable the combiner, set reduce task count to 3 and run your program on this dataset.

Please return (in Q3 directory):

- *Pseudo code of your algorithm*
- *Java file (name it InfluenceScore.java)*
- *JAR file*
- *Output files from your run on the given dataset (usergraph* files)*
- *Copy of the messages printed on screen when running your code (name it Q3.log)*