**Lab Practice2: Map/Reduce**

**2-1 Running Wordcount with Hadoop streaming, using Python code**

**Week5: 11-17 Esfand**

This assignment will build upon the previous assignment. You will examine the options for streaming that control the number of reducers. In the Cloudera Quickstart VM terminal, follow the instructions below to execute a simple word count example in Python.

1. Open a Terminal (Right-click on Desktop or click Terminal icon in the top toolbar)

2. Review the following to create the python code

**NOTE: If you have not programmed with Python please read the following:**

Python notes:

# 1 indentations are required to indicate blocks of code,

# 2 all code to be executed as part of some flow control

# (e.g. if or for statements) must have the same indentation

# (to be safe use 4 space per indentation level, and don't

# mix with tabs)

# 3 flow control conditions have a ':' before

# the corresponding block of code

#

**wordcount\_mapper.py**

#This mapper code will input a line of text and output <word, 1>

The reducer code has some basic parts

**wordcount\_reducer.py**

# ---------------------------------------------------------------

#This reducer code will input a line of text (which is the result of mapper.py) and

# output <word, total-count>

# if this current key is same as the last one Consolidate otherwise Emit

# ---------------------------------

You can cut and paste your code into a text file as follows from the terminal prompt in Cloudera VM. Type in the following to open a text editor (using gedit), and then cut and paste your code for wordcount\_mapper.py into the text editor, save, and exit. Repeat for wordcount\_reducer.py

> gedit wordcount\_mapper.py

> gedit wordcount\_reducer.py

Enter the following to see that the indentations line up as above

> more wordcount\_mapper.py

> more wordcount\_reducer.py

Enter the following to make it executable

> chmod +x wordcount\_mapper.py

> chmod +x wordcount\_reducer.py

Enter the following to see what directory you are in

> pwd

It should be /user/cloudera , or something like that.

3. Create some data:

> echo "A long time ago in a galaxy far far away" > /home/cloudera/testfile1

> echo "Another episode of Star Wars" > /home/cloudera/testfile2

4. Create a directory on the HDFS file system (if already exists that’s OK):

hdfs dfs -mkdir /user/cloudera/input

5. Copy the files from local filesystem to the HDFS filesystem:

hdfs dfs -put /home/cloudera/testfile1 /user/cloudera/input

hdfs dfs -put /home/cloudera/testfile2 /user/cloudera/input

6. You can see your files on HDFS

hdfs dfs -ls /user/cloudera/input

7. Run the Hadoop WordCount example with the input and output specified.

Note that your file paths may differ. The ‘\’ just means the command continues on next line.

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \

-input /user/cloudera/input \

-output /user/cloudera/output \

-mapper /home/cloudera/wordcount\_mapper.py \

-reducer /home/cloudera/wordcount\_reducer.py

Hadoop prints out a whole lot of logging or error information.

8. Check the output file (**part-00000**) to see the results:

9. View the output directory:

hdfs dfs -ls /user/cloudera/output

Look at the files there and check out the contents, e.g.:

hdfs dfs -cat /user/cloudera/output/part-00000

10. Streaming options:

Try: hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar –help or see [hadoop.apache.org/docs/r1.2.1/](http://hadoop.apache.org/docs/r1.2.1/)

Let’s change the number of reduce tasks to see its effects. Setting it to 0 will execute no reducer and only produce the map output. (Note the output directory is changed in the snippet below because Hadoop doesn’t like to overwrite output)

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \

-input /user/cloudera/input \

-output /user/cloudera/output\_new\_0 \

-mapper /home/cloudera/wordcount\_mapper.py \

-reducer /home/cloudera/wordcount\_reducer.py \

-numReduceTasks 0

11. Change the number of reducers to 2 and check the results.

Try to notice the differences between the output when the reducers are run in Step 9, versus the output when there are no reducers and only the mapper is run in step 10. Also compare results with the results of step 11 with 2 reducers.

The point of the task is to be aware of what the intermediate results look like. A successful submission will have words and counts that are not accumulated (which the reducer performs). Hopefully, this will help you get a sense of how data and tasks are split up in the map/reduce framework.

**What to provide as the result of your practice ():**

1. Intermediate and final results of the 3 task (with 1,0 and 2 reducers)
2. Answers to the following questions:

1- Map/Reduce performs a ‘shuffle’ and grouping. That means it...

1. Shuffles <key,value> pairs into random bins and then within a bin it groups keys.
2. Shuffles <key,value> pairs into different partitions according to the key value, and then aggregates all pairs in 1 partition into 1 group.
3. Shuffles <key,value> pairs into different partitions according to the key value, and sorts within the partitions by key.

2- In the word count example, what is the key?

1. The line number that contains the word.
2. The document id that contains the word.
3. The word itself.

3- Streaming map/reduce allows mappers and reducers to be written in what languages?

4- when you use 2 reducers instead of 1 reducer, what is the difference in global sort order?

1. With 1 reducer, but not 2 reducers, the word counts are in global sort order by word.
2. With 2 reducers, but not 1 reducer, the word counts are in global sort order by word.
3. With 1 reducer or 2 reducers, the word counts are in global sort order by word.
4. With 1 reducer or 2 reducers, the word counts are NOT in global sort order by word.

**Please check your final results with your TA**

**2-2 Joining data with streaming using Python code**

**Week6: 18-24 Esfand**

In this assignment, you are given a Python mapper and reducer to perform the course. The purpose of Part 1 of this assignment is to provide an example for Part 2. In Part 2 you will need to write Python code to perform a different Join and upload the output file(s).

Please read through all the instruction and the programming notes, it is not a hard programming assignment, but is worth the effort to understand the nature of map/reduce framework.

**PART 1**

1. Follow the steps from the Wordcount assignment to set up the following files on Cloudera: join1\_mapper.py, join1\_reducer.py, join1\_FileA.txt, and join1\_FileB.txt (see the*Code and Text Files* section at the bottom)

*Don’t forget to enter the following at the unix prompt to make it executable*

> chmod +x join1\_mapper.py

> chmod +x join1\_reducer.py

2. Follow the steps from the Wordcount assignment to set up the data in HDFS

3. Test the program in serial execution using the following Unix utilities and piping commands:

*(‘cat’ prints out the text files standard output; ‘|’ pipes the standard output to the standard input of the join\_mapper program, etc.. )*

> cat join1\_File\*.txt | ./join1\_mapper.py | sort | ./join1\_reducer.py

To debug programs in serial execution one should use small datasets and possibly extra print statements in the program. Debugging with map/reduce jobs is harder but hopefully not necessary for this assignment.

4. Run the Hadoop streaming command:

*(Note that your file paths may differ. Note the ‘\’ just means the command continues on next line. )*

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \

-input /user/cloudera/input \

-output /user/cloudera/output\_join \

-mapper /home/cloudera/join1\_mapper.py \

-reducer /home/cloudera/join1\_reducer.py

**PART 2: A new join problem**

1. First generate some datasets using the scripts (see the *Code and Text Files* section at the bottom) as follows:

> sh make\_data\_join2.txt

(this is a script that produces 6 files:

python make\_join2data.py y 1000 13 > join2\_gennumA.txt

python make\_join2data.py y 2000 17 > join2\_gennumB.txt

…)

2. Use HDFS commands to copy the 6 files created in step 1 into one HDFS directory, just like step 2 in Part 1 and in the wordcount assignment.

*Note: These datasets are pseudo-randomly generated so the output is the same for any environment. The files are not large, however they are big enough that it would be time consuming to solving the assignment by hand. One could put the data in a database but that would defeat the purpose of the assignment!*

3. The datasets generated in step 1 contain the following information:

join2\_gennum\*.txt consist of <TV show, count> (A TV show title and the number of viewers)

Example join2\_gennum\*.txt:

Almost\_News, 25

Hourly\_Show,30

Hot\_Cooking,7

Almost\_News, 35

Postmodern\_Family,8

Baked\_News,15

Dumb\_Games,60

…

join2\_genchan\*.txt consists of <TV show title, channel> (A TV show title and the channel it was on)

Example join2\_genchan\*.txt:

Almost\_News, ABC

Hourly\_Show, COM

Hot\_Cooking, FNT

Postmodern\_Family, NBC

Baked\_News, FNT

Dumb\_Games, ABC

…

**4. Your Task: Implement the following join request in Map/Reduce:**

***What is the total number of viewers for shows on ABC?***

The show-to-channel relationship is Many-to-Many. In other words, each show might appear on many channels, and each channel might broadcast many shows. In pseudo-SQL it might be something like:

*select sum( viewer count) from File A, File B where FileA.TV show = FileB.TV show and FileB.Channel='ABC' grouped by TV show*

5. Upload the resulting output from the reducers, use numReduceTasks=1

The output will look like: <TVshow\_title total\_viewers>

Example Output:

**Almost\_News 60**

**Dumb\_Games 60**

**…**

Data Notes:

* TV show titles do not have spaces
* Channels have 3 letters
* TV show titles can appear multiple times, with different counts
* A TV show and channel combination might appear multiple times
* TV shows could appear on multiple channels
* The output should have no commas or punctuation, only 1 space between the TV show title and number

**Programming Notes and Detailed Suggestions:**

You should be able to use 1 map/reduce job. You should consider starting with the join1\_mapper and the join1\_reducer code. All the logic and functional examples you will need are in the wordcount and the join1 code.

**Hint 1**: The new join mapper is like the join1 mapper but instead of stripping dates from the key field it should be selecting **rows related to 'ABC'.**

In Python strings are character arrays. Strings can be declared and accessed as follows:

my\_string = 'LMNOP'

my\_string[0:3]='LMN'

In Python the string function to check if a string is just digits is as follows:

my\_string.isdigit() # will return True or False.

**Hint 2**: The new join reducer is like the join1\_reducer but instead of building up a list of dates & counts, it should be summing viewer counts to keep a running total. Make sure you understand what will be in the intermediate output files that become reducer input (after the mapper and after Hadoop shuffle & group). Look carefully at the groups that are present in the reducer input to see what conditions your reducer will have to check for. You can test the mapper output using the Unix piping mentioned in Part 1.

**Hint 3**: This new join task has some overlap with wordcounting task. Use the wordcount code to review the counting and updating logic, as well as functions that handle strings and integers.

**Hint 4**: Here is a possible pseudo code example of how to implement this join using tv-show as the key:

**join2\_mapper**:

* read lines, and split lines into key & value
* if value is ABC or if value is a digit print it out

Note: you can test just the mapper by running something like:

> cat join2\_gen\*.txt | ./join2\_mapper.py | sort

Also, if we did have huge files partitioned across a cluster you might have information about viewer counts in one partition and information about which show is on ABC in another partition. The mapper and the Hadoop shuffle will bring those bits of information together to the same reducer if a key is a show.

**join2\_reducer**:

* read lines and split lines into key & value
* if a key has changed (and it's not the first input)
* then check if ABC had been found and print out key and running total,
* if value is ABC then set some variable to mark that ABC was found (like abc\_found = True)
* otherwise keep a running total of viewer counts

**Hint 5**:The first two lines of your output should be:

Almost\_Games 49237

Almost\_News 46592

**Code and Text Files**

join1\_mapper.py

#!/usr/bin/env python

import sys

# --------------------------------------------------------------------------

#This mapper code will input a <date word, value> input file, and move date

    into the value field for output

join1\_reducer.py

#!/usr/bin/env python

import sys

# --------------------------------------------------------------------------

#This reducer code will input a <word, value> input file, and join words

    together

# Note the input will come as a group of lines with same word (ie the key)

# As it reads words it will hold on to the value field

#

# It will keep track of current word and previous word, if word changes

# then it will perform the 'join' on the set of held values by merely

    printing out the word and values. In other words, there is no need

# to explicitly match keys b/c

# Hadoop has already put them sequentially in the input

# At the end it will perform the last join

#

# Note, there is NO error checking of the input, it is assumed to be correct,

    Meaning it has word with correct and matching entries, no extra spaces, etc.

# see https://docs.python.org/2/tutorial/index.html for python tutorials

#

# see https://docs.python.org/2/tutorial/datastructures.html for list details

join1\_FileA.txt

able,991

about,11

burger,15

actor,22

join1\_FileB.txt

Jan-01 able,5

Feb-02 about,3

Mar-03 about,8

Apr-04 able,13

Feb-22 actor,3

Feb-23 burger,5

Mar-08 burger,2

Dec-15 able,100

make\_join2data.py

#!/usr/bin/env python

import sys

# --------------------------------------------------------------------------

# (make\_join2data.py) Generate a random combination of titles and viewer

    counts, or channels

# this is a simple version of a congruential generator,

# not a great random generator but enough

# --------------------------------------------------------------------------

chans = ['ABC','DEF','CNO','NOX','YES','CAB','BAT','MAN','ZOO','XYZ','BOB']

sh1 =['Hot','Almost','Hourly','PostModern','Baked','Dumb','Cold','Surreal'

    ,'Loud']

sh2 =['News','Show','Cooking','Sports','Games','Talking','Talking']

vwr =range(17,1053)

chvnm=sys.argv[1] #get number argument, if its n, do numbers not channels,

lch=len(chans)

lsh1=len(sh1)

lsh2=len(sh2)

lvwr=len(vwr)

ci=1

s1=2

s2=3

vwi=4

ri=int(sys.argv[3])

for i in range(0,int(sys.argv[2])): #arg 2 is the number of lines to output

if chvnm=='n': #no numuber

print('{0}\_{1},{2}'.format(sh1[s1],sh2[s2],chans[ci]))

else:

print('{0}\_{1},{2}'.format(sh1[s1],sh2[s2],vwr[vwi]))

ci=(5\*ci+ri) % lch

s1=(4\*s1+ri) % lsh1

s2=(3\*s1+ri+i) % lsh2

vwi=(2\*vwi+ri+i) % lvwr

if (vwi==4): vwi=5

make\_data\_join2.txt (a short command line script)

python make\_join2data.py y 1000 13 > join2\_gennumA.txt

python make\_join2data.py y 2000 17 > join2\_gennumB.txt

python make\_join2data.py y 3000 19 > join2\_gennumC.txt

python make\_join2data.py n 100 23 > join2\_genchanA.txt

python make\_join2data.py n 200 19 > join2\_genchanB.txt

python make\_join2data.py n 300 37 > join2\_genchanC.txt

**What to Check with your TA**

1- Your python code for mappers and reducers

2-The join results

3- Answer to the question : “***What is the total number of viewers for shows on ABC?”***