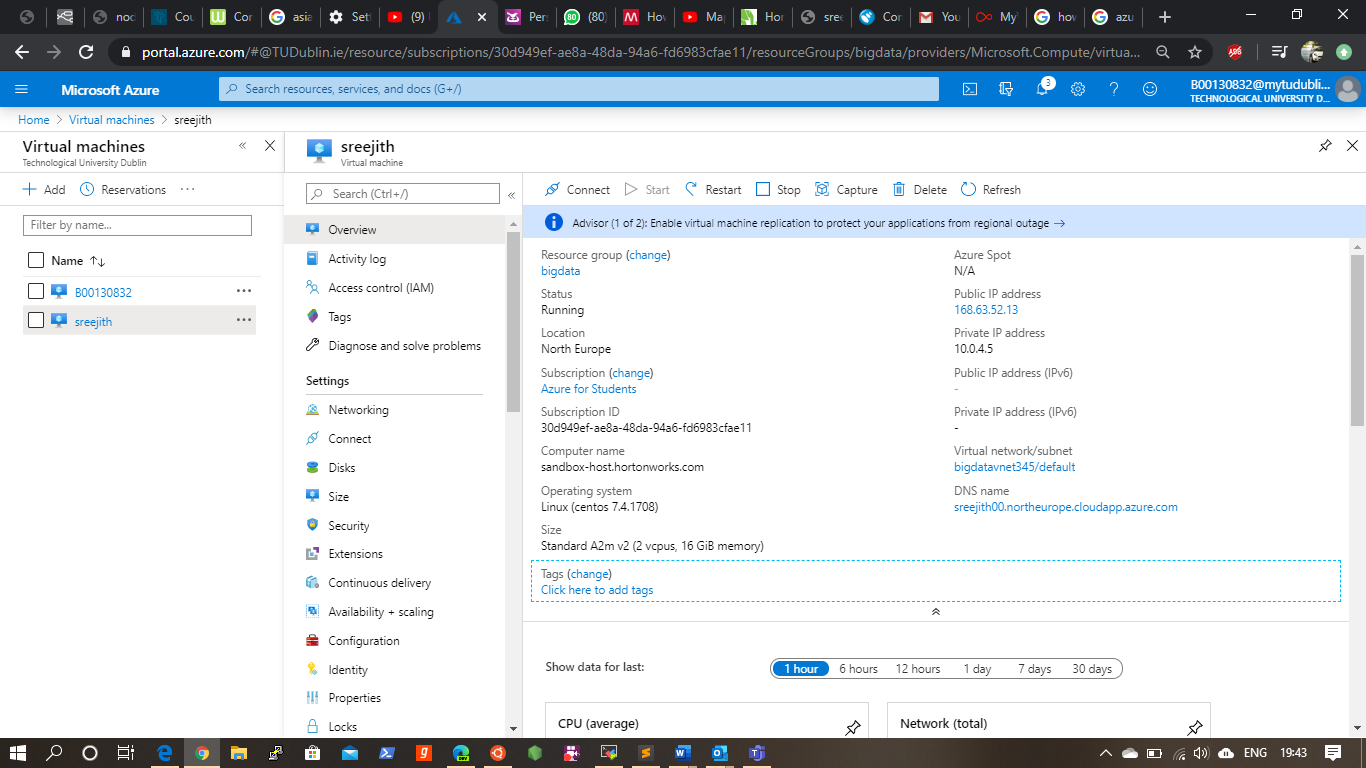
***Bigdata CA3***

*A part WordCount MapReduce job*

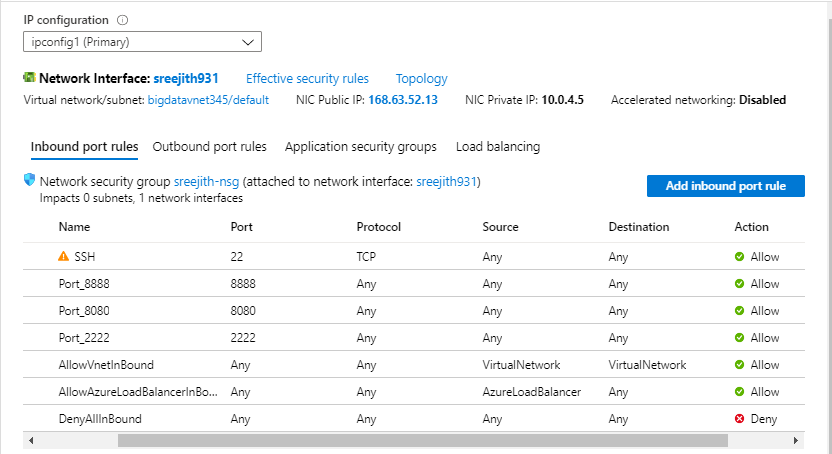
**Part(a) – 10%:** Set up a single node Hadoop cluster and run the WordCount MapReduce job in standard mode (use the example jar file that comes with Hadoop) and streaming mode through Python.

*Setup Azure Hortonworks Data Platform (HDP) Sandbox*

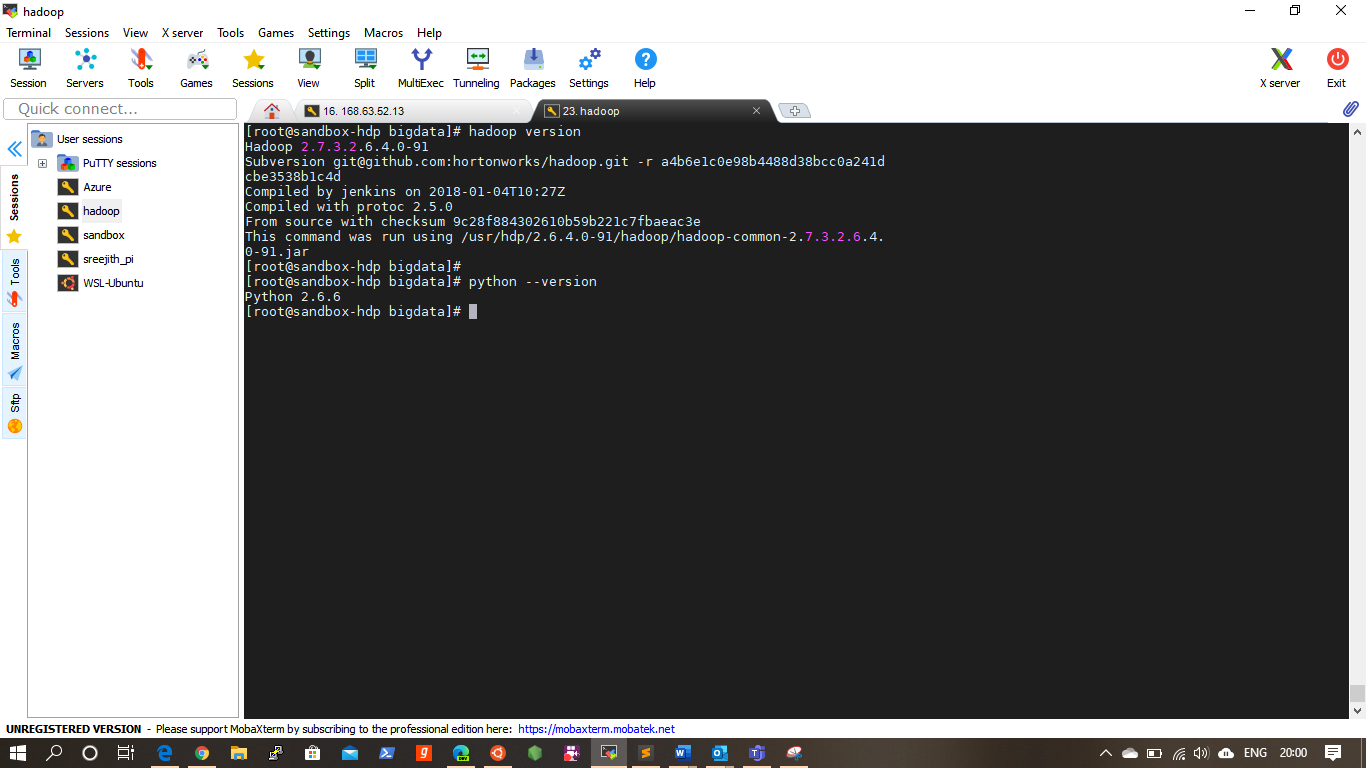


Deploy Azure Hadoop platform successfully which consist of 14GB memory, most of laptops struggle to run Hortonworks which require more memory for running, so I opted azure Hortonworks. Which includes many supporting packages with the platform.

*Enable necessary ports for Azure Hortonworks*

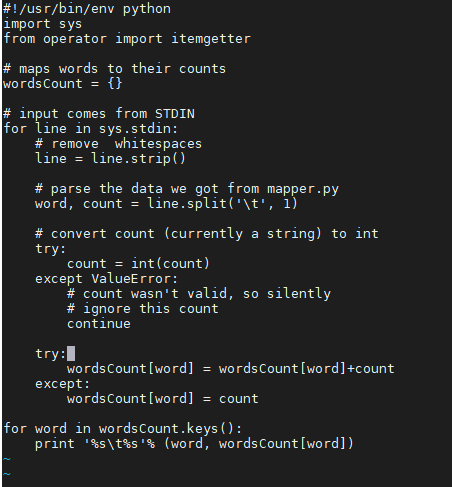


*Accessing Virtual machine via MobaXterm*

Checking versions: Hadoop 2.7.7.3.2 , Python 2.6.6 in Hadoop Terminal, Linux CentOS release 7.5.1804 (Core) is running 

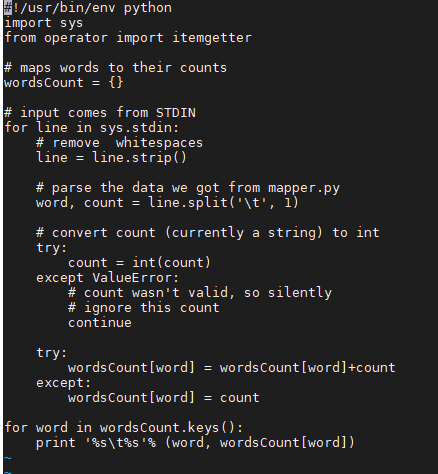
Now I am going to import sample text file for word counting, I am using text file which is previously used on previous assignment CA1 Mobi.txt via mobaXterm.

*Word Count Mapper python code*



Program fetch lies from stdin and remove unwanted whitespace using strip function split function convert sentences into list of words generate Output with delimited format(word,1) printing final output

*Word Count Reducer python code*



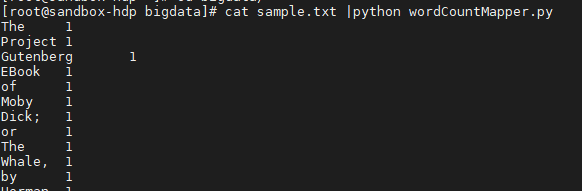
Script remove leading and trailing whitespace from the input we got from wordcountMapper.py and convert count (currently a string) to int type, then check the count of each word marked in try except

*Execution of scripts*

Scripts can be executed in a single line by separating by pipe symbol. In the result we got all words and its counts.

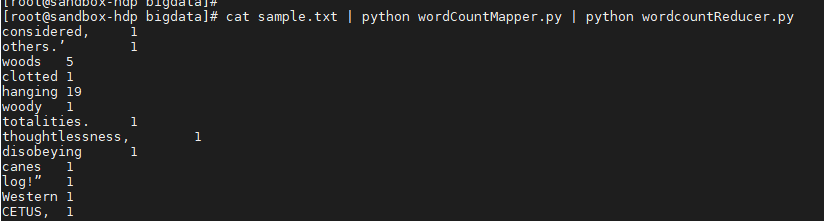
Run mapper command

*“cat sample.txt | python wordCountMapper.py”*



Run mapper and reducer combined

*“cat sample.txt | python wordCountMapper.py | python wordcountReducer.py”*



Result shows each single word and the count of the word in the file. Rename python files for easy readability as mappy.py and reducer.py

Now we I am running same program which feed data from Hadoop file system HDFS

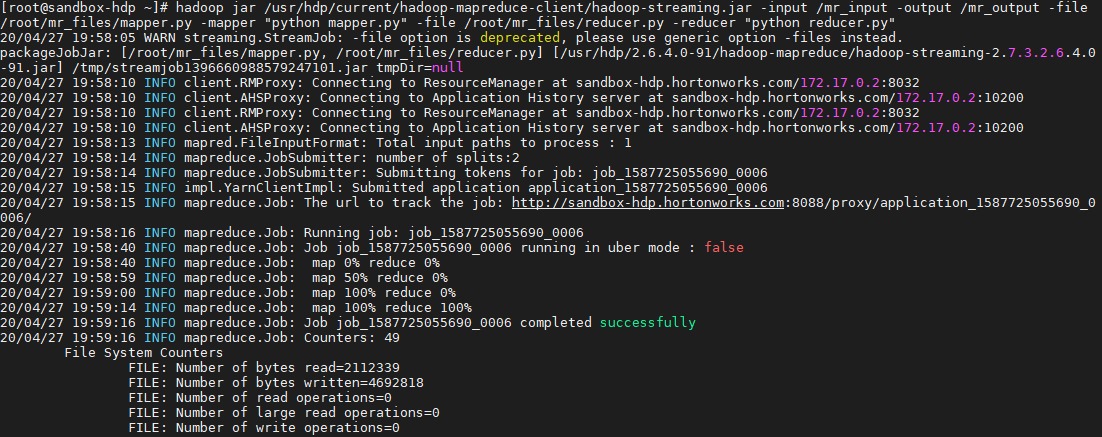
For that I create a directory on Hadoop using following command and putt my input text file using ssh ,

“hdfs dfs -mkdir mr\_input /”

“hdfs dfs -put /root/sample.txt /"mr\_input"

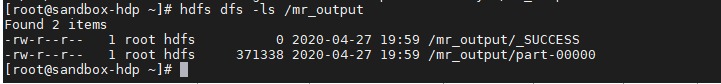
Command for run mapper and reducer in Hadoop streaming in HDFS

“hadoop jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-streaming.jar -input /mr\_input -output /mr\_output1 -file /root/hello/mapper.py -mapper "python mapper.py" -file /root/hello/reducer.py -reducer "python reducer.py””



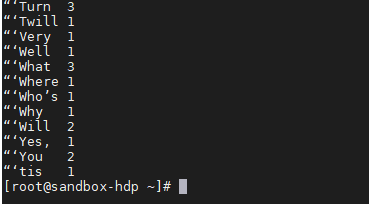
Map reduce problem is running as per the following command and successfully completed.

now result is written into mr\_output.



Now we can check result inside /mr\_output/part-00000 using command

“hdfs dfs -cat /mr\_output/part-00000”



Now we can see the words and count of words. This is simple demonstration of MapReduce job in HDFS on azure hornworks sandbox.

*B part*

**Part (b) – 10%:** Consider a very large file of integers and design MapReduce algorithms to find the following:

1. The largest integer.

**Mapper Algorithm**

Step 1: Fetch data from HDFS via sys.

Step 2: Loop through integers and split.

Step 3: Generate a key value pair integer and count.

Step 4: Return key and value pair

**Reducer Algorithm**

Step 1: Intilize max\_vale as 0.

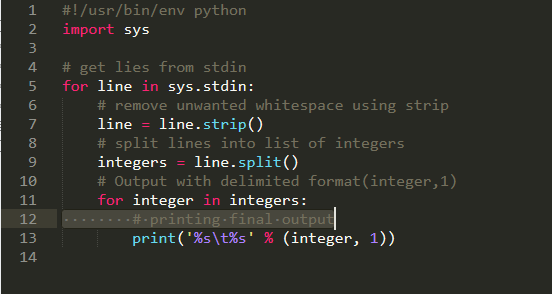
Step 2: Loop through key value pair and set max\_value

if max\_value < integer

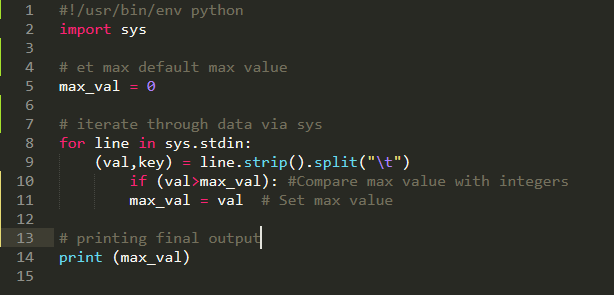
Step 3: Return max\_val

Test the algorithm in Hadoop system

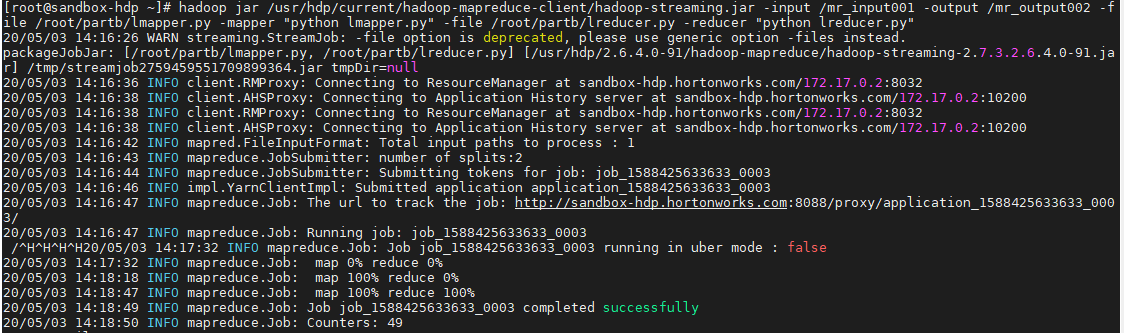
Mapper code



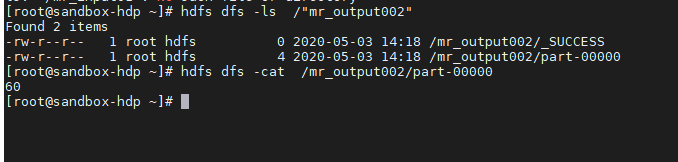
Reducer Code



Running code on HDFS streamer.



Following commands for check output of data



1. The average of all the integers.

**Mapper Algorithm**

Step 1: Fetch integers via sys

Step 2: Get the count and the average of integers

Step 3: Generate a constant key 1

Step 4: Value as average and count (average, count)

Step 5: Return the key value pair (1, (average, count))

**Reduce Algorithm**

Step 1: Loop through key value pairs (1, (average, count))

Step 2: Get sum of (average \* count) and sum of count to total number of integers

Step 3: Return average as sum of (average \* count) / sum of count

1. The same set of integers, but with each integer appearing only once.

**Mapper Alorithm**

Step 1: Loop through each integer in list.

Step 2: Generate a key value pair key is integer and value is count.

Step 3: Return keys (integers) are grouped and values

**Reduce Alorithm**

Step 1: fetch key values from mapper.

Step 2: Reduced key value pairs is obtained.

Step 3: Return list of keys are the set integers.

1. The count of the number of distinct integers in the input.

**First map reduce job**

Mapper Algorithm:

Step 1: Generate a key value pair, (integer, count)

Step 2: Return list of keys.

**Reducer Algorithm**

Step 1: list of keys obtained as input and generate key value pair with integers as values (1, integers).

Step 2: Return the count of distinct integer as values of the constant key.

--------------------------------------------------------------------------------------------------------

**Part C Friends You may know**

Friends you may know Algorithm is used for suggestion of friend’s user may know with regarding to common friends of users. Most of the social media platform use friend suggestion algorithms.

We have a data set of adjacency list of graphs consist of users’ ID and their friends list 50k user information in the format.

<USER><TAB><Friends>

All data are represented as unique integer values. Now our task is to find friend recommendations for the following users’ IDs: 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993. In such a way, 0 is friend with 1, 1 is friend with 0. Program need to generate a set of Above User ID and their recommendation based on common friends. It will list of other friends who have more common friends, so number of suggestions is limited by N = 10. Output will be like this.

<USER><TAB><Recommendations>

Considering an example, 0 [1,2,3,]

User ID = 0

Friends IDs = 1,2,3

So, the possible combinations of common friends

*Key Value (Common friends)*

0U,1U 0

0U,2U 0

0U,3U 0

1U,2U 1

1U,3U 1

2U,1U 1

2U,3U 1

3U,1U 1

3U,2U 1

U1, U2, U3 is friend with U0, but U1, U2, U3 have no friendship, but they have a common friend U0, so there is chance to know U1 with U2, U1 with U3, this is how algorithm works.

U1 may know U2 and U3

U2 may know U1 and U3

U3 may know U1 and U2

***Mapper Algorithm and code***

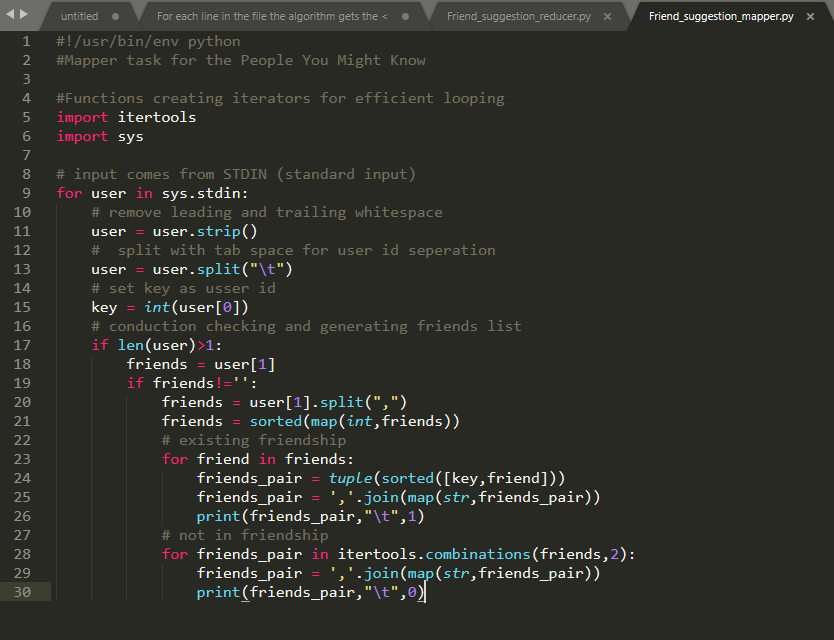
*Step 1: Fetch data from input file*

*Step 2: loop through lines and fetch <USER\_ID> and list of friends*

*Step 3: Generate all possible pair of friends.*

*Step 4: Additional value as 1 for already in friendship and 0 for no friendship*

*Step 5: return friends and common friends*



***Reducer Algorithm and code***

*Step 1: Group all the repeated keys and indicates the number of common friends.*

*Step 2: Loop through lines and recover the key (formed by a pair of individuals) and Flag.*

*Step 3: Append to usser\_id dict*

*Step 4: Geneate friend suggestion based on number of common friends*

*Step 5: Return required usser suggestion*

**Code for Reducer**

*#!/usr/bin/env python*

*#Reduce program for People You Might Know*

*#Functions creating iterators for efficient looping*

*import itertools*

*import sys*

*# count of reccomantation*

*top\_reccomentation = 10*

*# user\_id of required ussers*

*required\_user = [924,8941,8942,9019,9020,9021,9022,9990,9992,9993]*

*# check friendship status of friend cobination*

*def addtoCombinbation(usser\_id,i,j,flag):*

*if flag==1:*

*flag = True*

*else:*

*flag = False*

*if i not in usser\_id:*

*usser\_id[i] = {}*

*usser\_id[i][j] = [1,False]*

*else:*

*if j in usser\_id[i]:*

*usser\_id[i][j][0] += 1*

*else:*

*usser\_id[i][j] = [1,False]*

*if flag==True:*

*usser\_id[i][j][0] -= 1*

*usser\_id[i][j][1] = True*

*# input comes from STDIN is append to dict usser*

*usser\_id = {}*

*for line in sys.stdin:*

*# remove leading and trailing whitespace*

*line = line.strip()*

*line = line.split("\t")*

*key = tuple(map(int,line[0].strip().split(",")))*

*flag = int(line[1])*

*i,j = key*

*addtoCombinbation(usser\_id,i,j,flag)*

*addtoCombinbation(usser\_id,j,i,flag)*

*for i in usser\_id.keys():*

*friend\_recomentation = []*

*for j in usser\_id[i].keys():*

*n,flag = usser\_id[i][j]*

*if flag==False:*

*friend\_recomentation.append((j,n))*

*friend\_recomentation = sorted(friend\_recomentation,key=lambda x: x[0])*

*friend\_recomentation = sorted(friend\_recomentation,key=lambda x: x[1],reverse=True)*

*if len(friend\_recomentation)>0:*

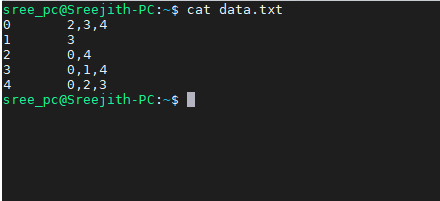
*friend\_recomentation = list(map(str,zip(\*friend\_recomentation)[0]))*

*# filtering required user id*

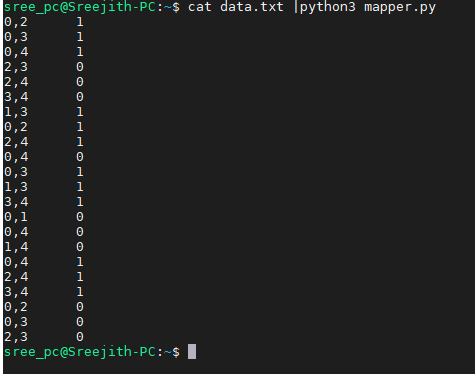
*if i in required\_user:*

*print "USER\_ID : "i,"\t",','.join(friend\_recomentation[:top\_reccomentation])*

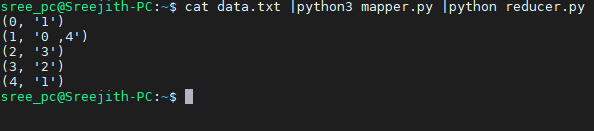
Testing code on local machine on sample data



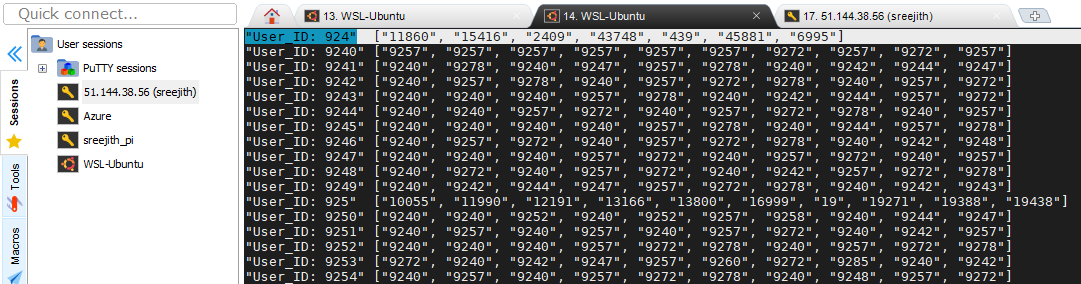
Testing Mapper code



Testing mapper and Reducer code



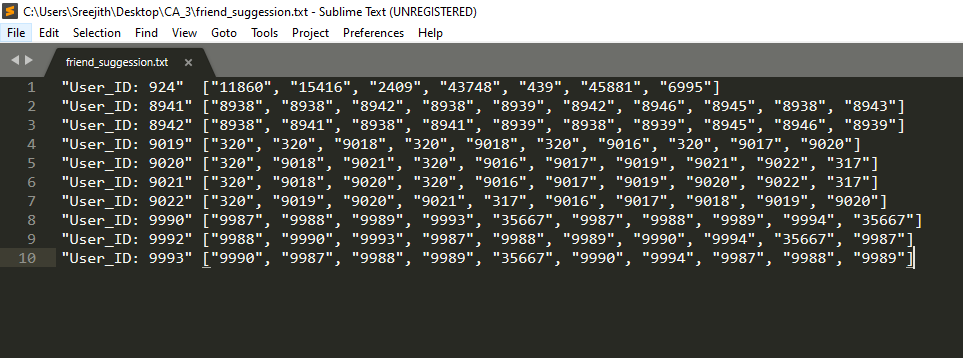
Test with full data and Final suggestion for the user ID 924, 8941, 8942, 9019, 9020, 9021, 9022, 9990, 9992, 9993. On this stage I didn’t apply filtering so full friend suggestion



For better understandability I write data into txt file

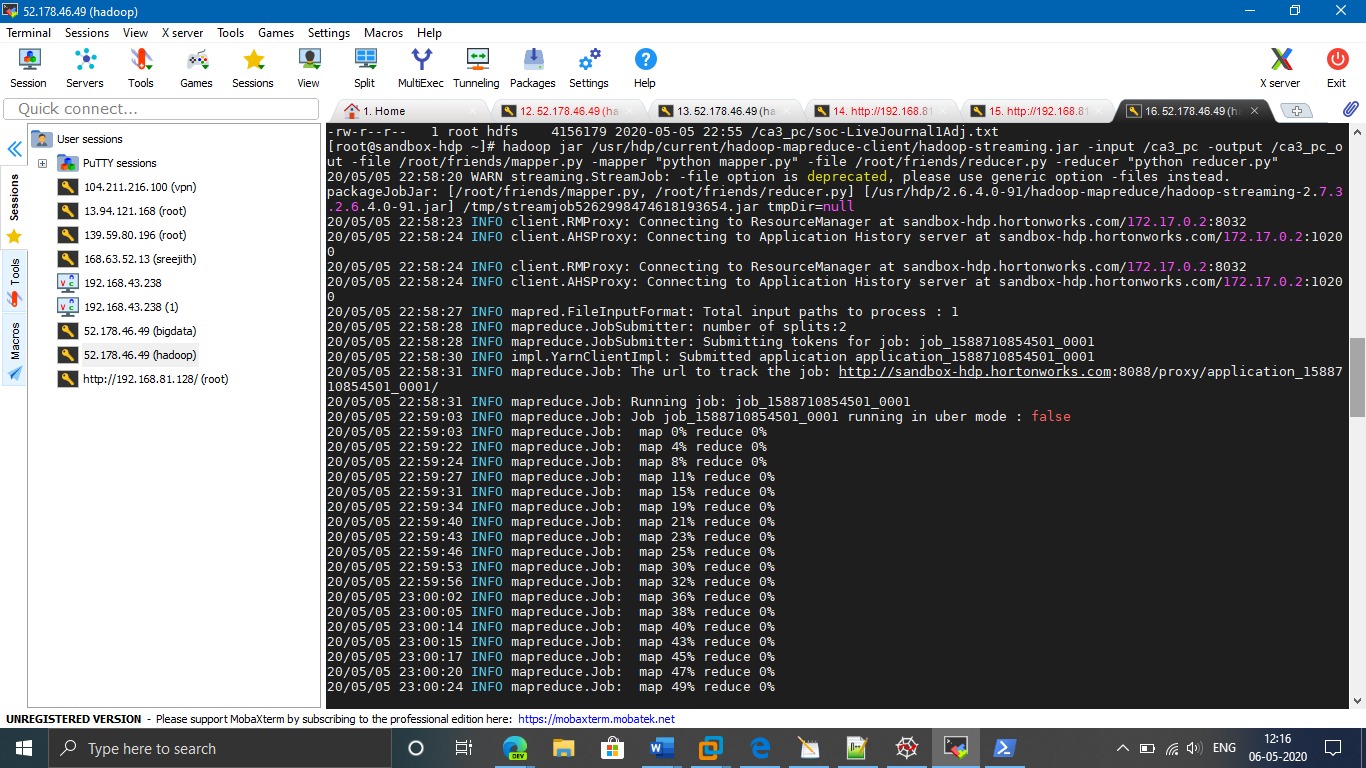


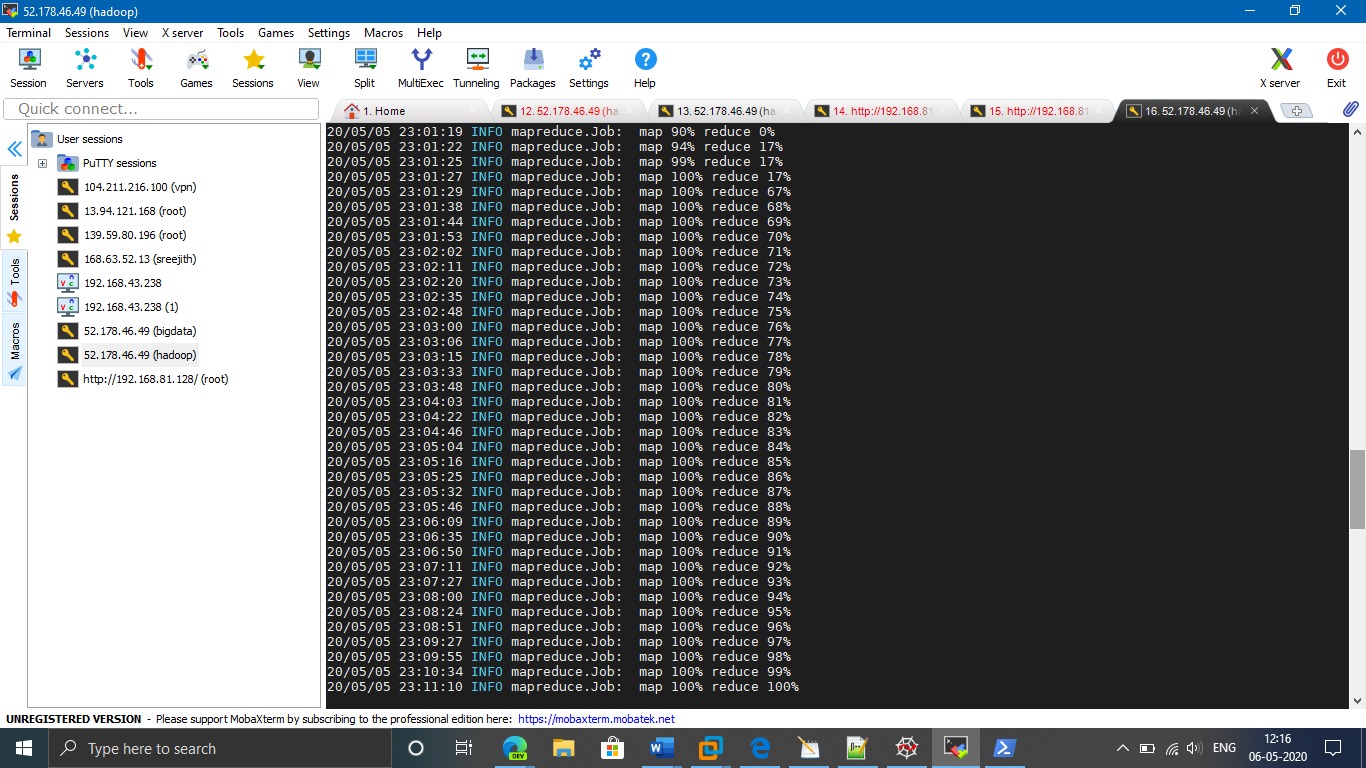
Final filtered friend suggestion obtained from MapReduce Algorithm code.



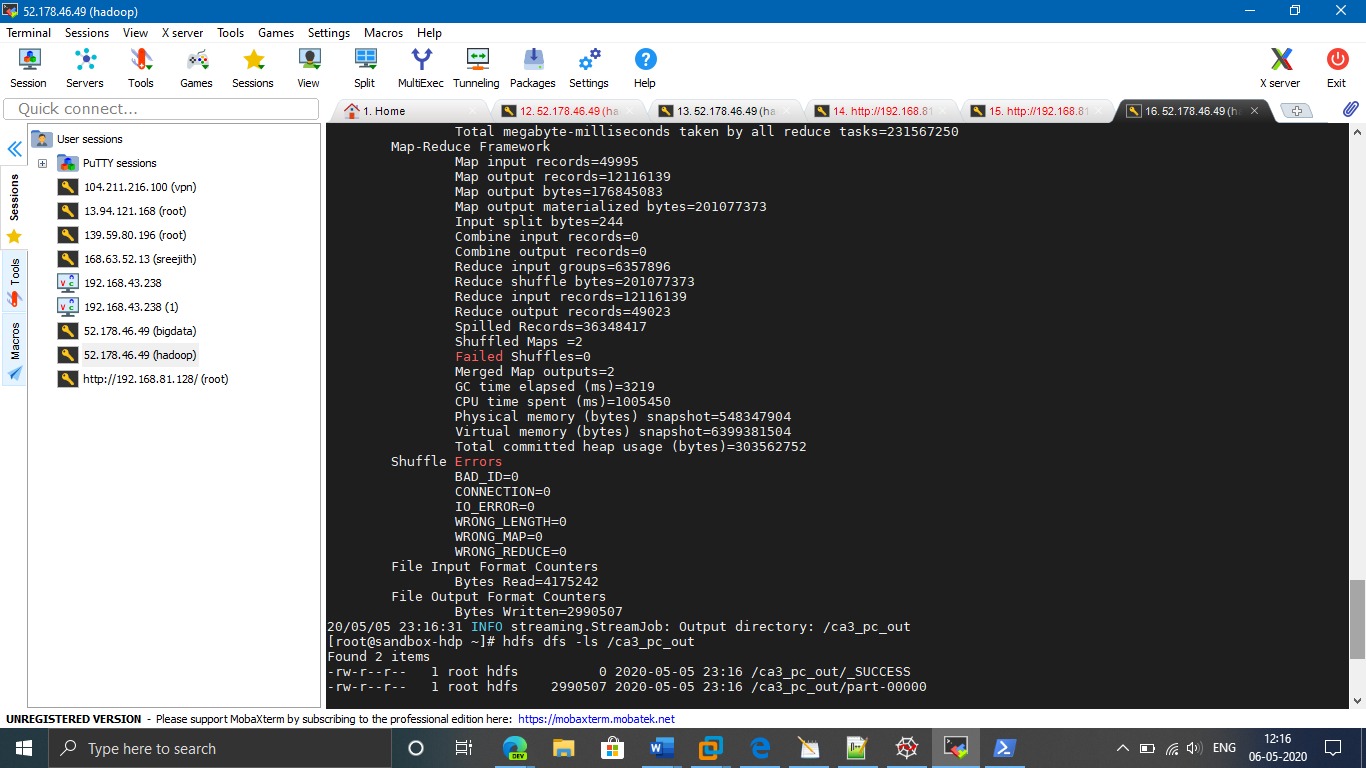
For me, its nearly 30minits to generate this result on normal system lets check the Speed and efficiency of HDFS.

***HDFS file streaming***





To complete the execution on HDFS it took nearly 10 minutes. Now checking the output data.



***Result***

Now output is stored in output file. As per required user IDs and there friends suggestion.

"User\_ID: 924" ["11860", "15416", "2409", "43748", "439", "45881", "6995"]

"User\_ID: 8941" ["8938", "8938", "8942", "8938", "8939", "8942", "8946", "8945", "8938", "8943"]

"User\_ID: 8942" ["8938", "8941", "8938", "8941", "8939", "8938", "8939", "8945", "8946", "8939"]

"User\_ID: 9019" ["320", "320", "9018", "320", "9018", "320", "9016", "320", "9017", "9020"]

"User\_ID: 9020" ["320", "9018", "9021", "320", "9016", "9017", "9019", "9021", "9022", "317"]

"User\_ID: 9021" ["320", "9018", "9020", "320", "9016", "9017", "9019", "9020", "9022", "317"]

"User\_ID: 9022" ["320", "9019", "9020", "9021", "317", "9016", "9017", "9018", "9019", "9020"]

"User\_ID: 9990" ["9987", "9988", "9989", "9993", "35667", "9987", "9988", "9989", "9994", "35667"]

"User\_ID: 9992" ["9988", "9990", "9993", "9987", "9988", "9989", "9990", "9994", "35667", "9987"]

"User\_ID: 9993" ["9990", "9987", "9988", "9989", "35667", "9990", "9994", "9987", "9988", "9989"]

**References**

1. Hadoop.apache.org. (2019). Hadoop Streaming. [online] Available at: <https://hadoop.apache.org/docs/r1.2.1/streaming.html>.
2. Geeksforgeeks a Computer science portal [2014] Available at: <https://www.geeksforgeeks.org/find-the-element-that-appears-once/>
3. Andresromero.github.io. (2019). Loading Effects for Grid Items | Demo 1.

Available at : <http://andresromero.github.io/People-you-may-know/>

1. Hadoop tutorials basic Available at : <https://www.tutorialspoint.com/hadoop/index.htm>
2. Hortonworks. (2019). Learning the Ropes of the HDP Sandbox - Hortonworks. [online] Available at: <https://www.cloudera.com/tutorials/getting-started-with-hdp-sandbox.html>