**CSC 555: Mining Big Data**

Final (due November 22nd)

Nikhil Bhalala

In the final, you will execute queries using Pig and Hadoop streaming and develop a custom version of KMeans clustering. The schema and data is available at:

<http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/>

You can use the 1-node or 3-node cluster for the final, just indicate what you have used. Please be sure to submit all code. You should also submit the command lines you use and a screenshot of a completed run (just the last page, do not worry about capturing the entire output).

**I highly recommend creating a small sample input** (e.g., by running head -n 10000 lineorder.tbl > lineorder.tbl.sample, you can create a small version of lineorder with 10000 lines) and testing your code with a smaller file until you can verify that it works.

# Part 1: Pig

Implement the following query using Pig:

select c\_nation, SUM(lo\_revenue) as SUM

from customer, lineorder

where lo\_custkey = c\_custkey

and c\_region = 'ASIA'

and (lo\_discount < 6 OR lo\_discount = 8)

group by c\_nation

order by SUM;

ANS:

wget http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/customer.tbl

hadoop fs -put customer.tbl

head -n 10000 lineorder.tbl > lineorder.tbl.sampl

hadoop fs -put lineorder.tbl.sampl

cd pig-0.15.0

bin/pig

line\_data = LOAD '/user/ec2-user/lineorder.tbl.sampl' USING PigStorage('|') As (lo\_orderkey:int,lo\_linenumber:int,lo\_custkey:int,lo\_partkey:int,lo\_suppkey:int,lo\_orderdate:int,lo\_orderpriority:int,lo\_shippriority:int,lo\_quantity:int,lo\_extendedprice:int,lo\_ordertotalprice:int,lo\_discount:int,lo\_revenue:int,lo\_supplycost:int,lo\_tax:int,lo\_commitdate:int,lo\_shipmode:int);

cust\_data = LOAD '/user/ec2-user/customer.tbl' USING PigStorage('|') As(c\_custkey:int,c\_name:chararray, c\_address:chararray,c\_city:chararray,c\_nation:chararray,c\_region:chararray,c\_phone:chararray,c\_mktsegment:chararray);

cust\_filter= FILTER cust\_data BY c\_region == 'ASIA';

liner\_filter= FILTER line\_data BY (lo\_discount < 6 OR lo\_discount == 8);

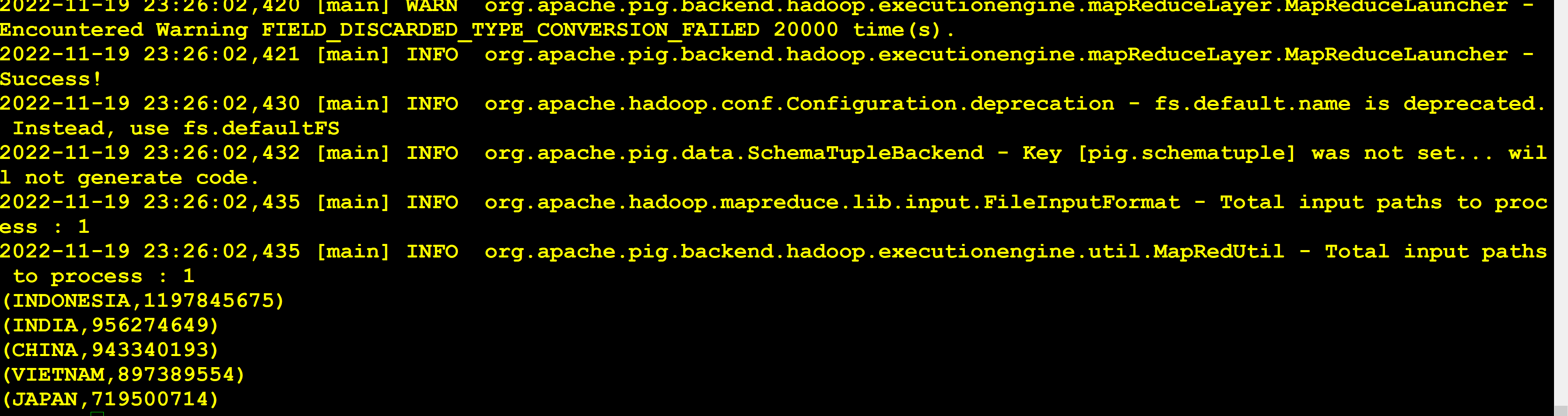
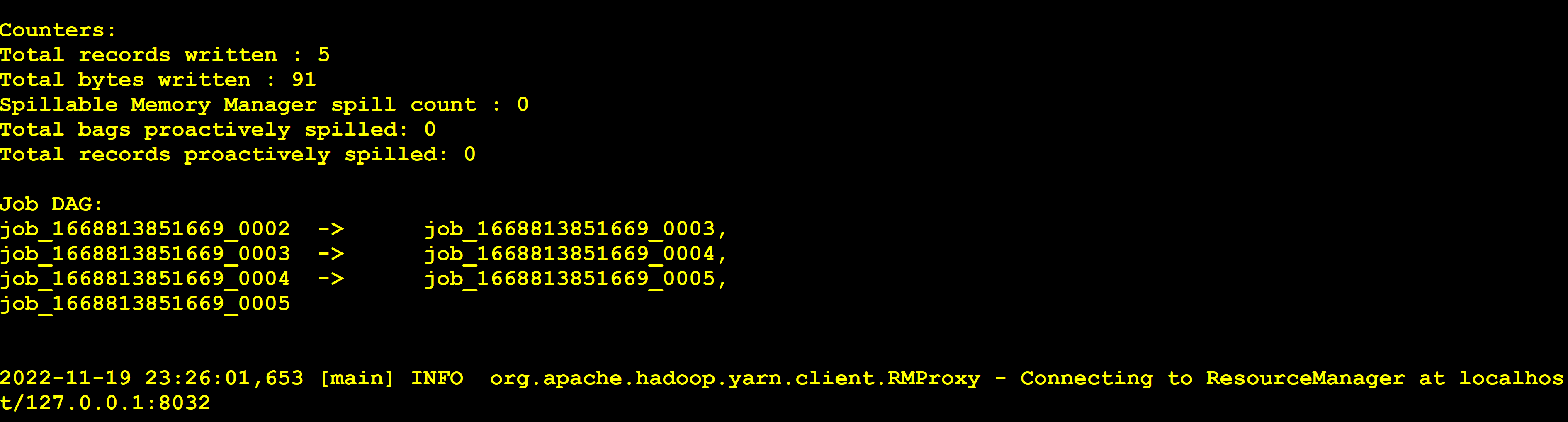
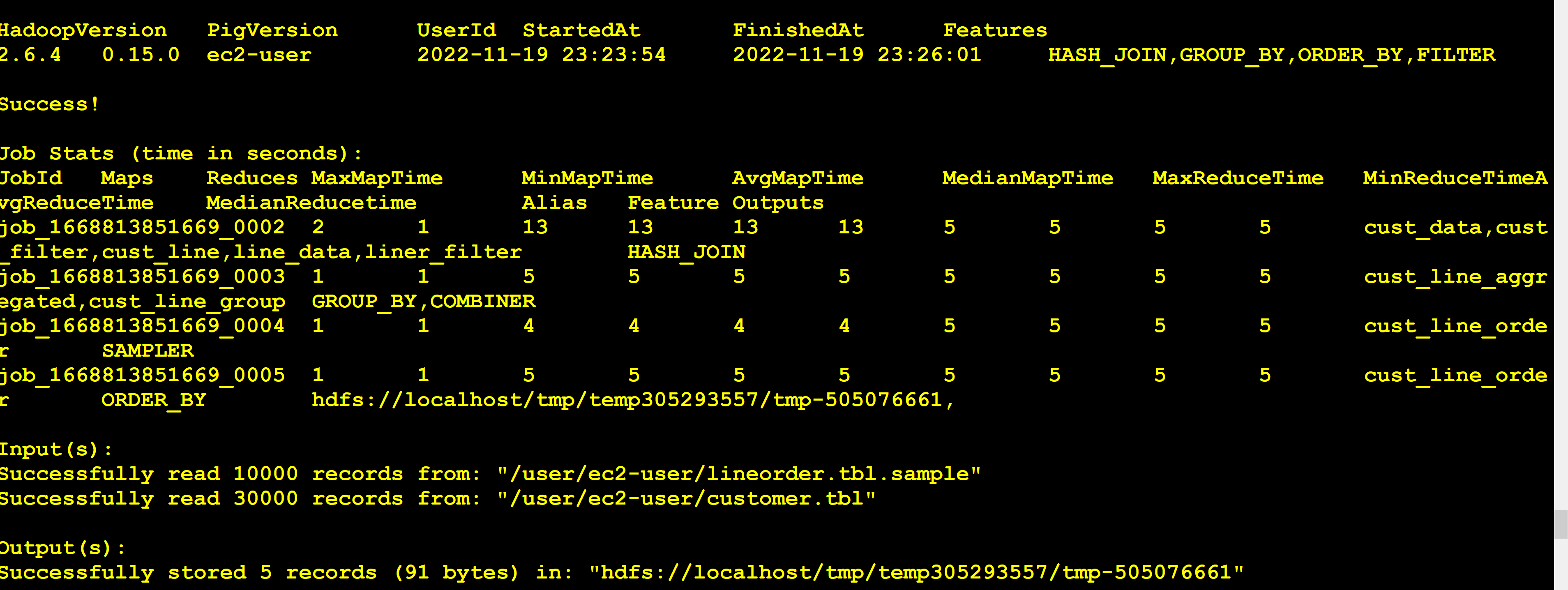
cust\_line = JOIN cust\_filter BY c\_custkey,liner\_filter BY lo\_custkey;

cust\_line\_group =GROUP cust\_line BY c\_nation;

cust\_line\_aggregated = FOREACH cust\_line\_group GENERATE group, SUM(cust\_line.lo\_revenue);

cust\_line\_order = ORDER cust\_line\_aggregated BY $1 DESC;

dump cust\_line\_order;



# Part 2: Hadoop streaming

Implement the following query using Hadoop streaming:

select sum(lo\_revenue), count(\*), d\_year, p\_category

from lineorder, dwdate, part

where lo\_orderdate = d\_datekey

and lo\_partkey = p\_partkey

and d\_sellingseason = 'Fall'

and p\_brand1 = 'MFGR#2123'

group by d\_year, p\_category;

In Hadoop streaming, normally such query will use a total of 3 passes (two joins and another one for GROUP BY). However, you have to implement it as a map- join with dwdate, which would only require a total of 2 MapReduce passes (one for join of lineorder and part, and one for GROUP BY). You would join the dwdate table like we did in class using “-file dwdate.tbl” option. This query requires at least two passes -- Part table is not a “small” table and cannot be joined in the mapper.

Mapper 1

#!/usr/bin/python

import sys

fd = open("dwdate.tbl")

lines =fd.readlines()

fd.close()

dwdate ={}

for line in lines:

vals=line.split('|')

d\_datekey = vals[0]

d\_year =vals[4]

if vals[12] =="Fall":

dwdate[d\_datekey]=d\_year

# input comes from STDIN (standard input)

for line in sys.stdin:

line = line.strip()

split = line.split('|')

lo\_key=split[5]

if split[2].find('#')>=0:

if split[4] == 'MFGR#2123' :

print (split[0] + '\t'+ split[3] + '\t' + 'part')

else:

if lo\_key in dwdate:

print (split[3]+ '\t'+ split[12] + '\t'+ dwdate[lo\_key]+ '\t' +'lineorder')

reducer 1 #!/usr/bin/python

import sys

currentKey = None

valspr = None

valsLO = None

for line in sys.stdin:

split = line.strip().split('\t')

key = split[0]

values = split[1].split(' ')

# if currentKey == key: # Same key

if values[-1].endswith('part'):

valspr.append(values) # Remove the 'part' entry

print('value')

print(values)

else:

valsLO.append(values)

print('vaulelo')

print(values)

'''

else:

if currentKey:

for pr in valspr:

for lo in valsLO:

pr1 =valspr[pr].split('\t')

lo1 =valsLO[lo].split('\t')

print(pr1[0] +'\t'+ pr1[1] + '\t' + lo1[0] +'\t'+ lo1[1] + '\t' + lo1[2])

currentKey = key

if values[-1].endswith('part'):

valspr = [values] # Remove the 'part' entry

valsLO = []

else:

valspr = []

valsLO = [values]

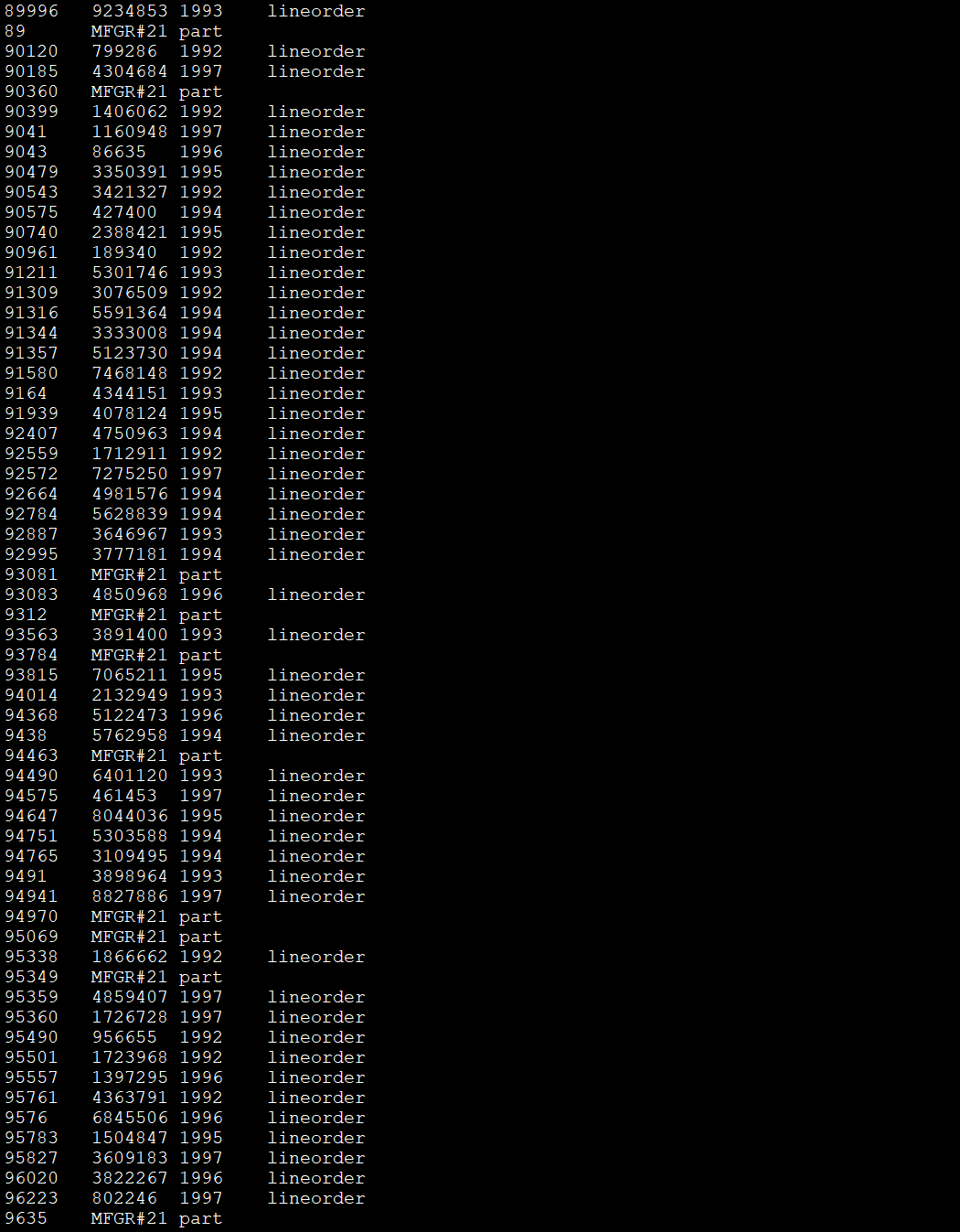
for pr in valspr:

for lo in valsLO:

pr1 =valspr[pr].split('\t')

lo1 =valsLO[lo].split('\t')

print(pr1[0] +'\t'+ pr1[1] + '\t' + lo1[0] +'\t'+ lo1[1] + '\t' + lo1[2])



Hadoop fs -ls liner\_part

hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -input /user/ec2-user/liner\_part input file dwdate -mapper finalMapper3.py -file finalMapper3.py -reducer finalReducertest22.py -file finalReducertest22.py -output output1

# Part 3: Clustering

Using Hadoop streaming and randomly generated data (similar to what you did in Assignment6, but generate 1,600,000 rows and 7 columns of numeric data) perform four KMeans iterations manually, using 11 centers**.** You can randomly choose the initial centers, such as by picking 11 random points from your data. For each of four KMeans iterations, include the centers produced by your code.

This would require passing a text file with cluster centers using -file option as discussed in class, opening the centers.txt in the mapper with open(‘centers.txt’, ‘r’) and assigning a key to each point based on which center is the closest to each particular point. Your reducer would then compute the new centers by averaging the points, which would conclude the iteration. At that point, the output of the reducer with new centers can be given to the next pass of the same map reduce code using the -file option (you would need to get the output from HDFS into a local file for that).

The only difference between first and subsequent iterations is that in first iteration you have to pick the initial centers. Starting from the 2nd iteration, the centers will be given to you by a previous pass of KMeans, and so on. Include the centers you computed at each iteration in your answer.

Ans:-

Python file dataset

#!/usr/bin/python

import random

fd = open('Num\_dataset.txt', 'w')

num = int(random.random() \* 1000 + 100)

for x in range(1600000):

for y in range(7):

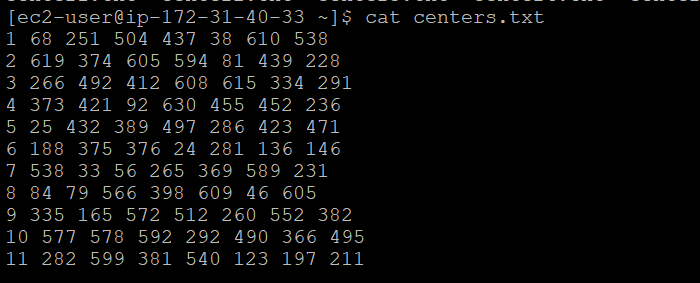
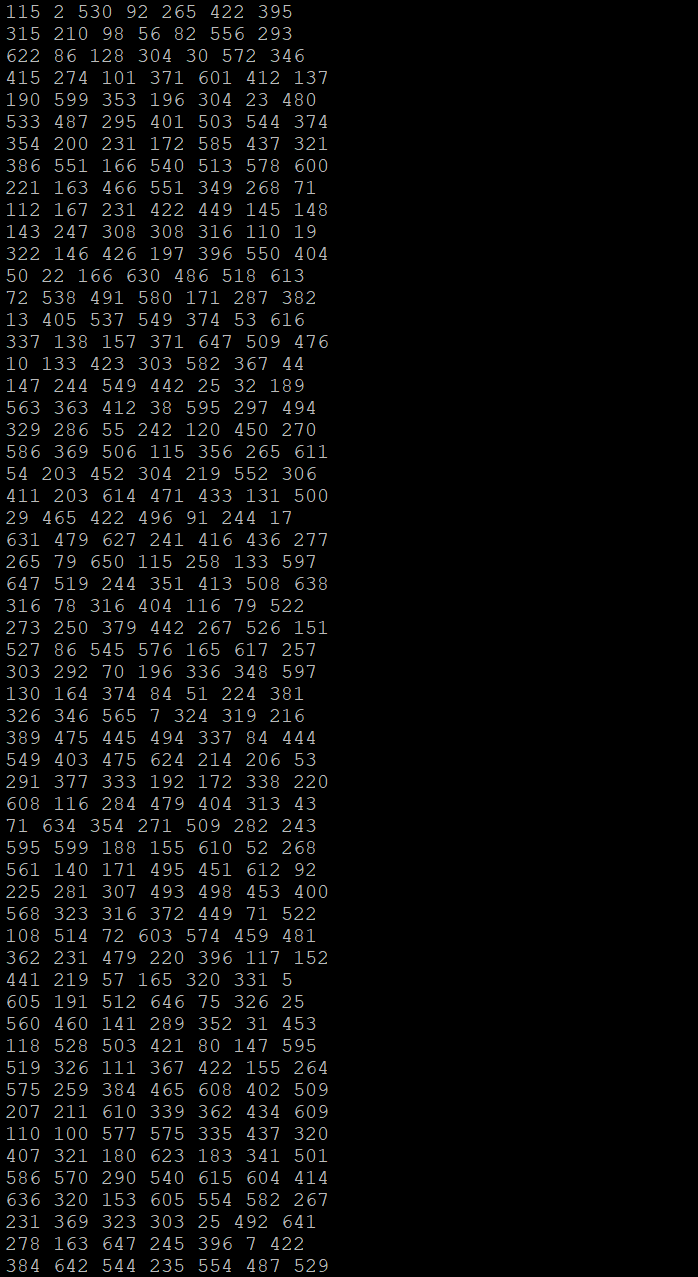
r = str(int(random.random() \* num))

fd.write(r)

fd.write(" ")

fd.write("\n")

fd.close()



KmMapper:

#!/usr/bin/python

import sys

import math

centers\_point = {}

fd = open('centers.txt', 'r')

centers = map(lambda x:x.strip().split(' '), fd.readlines())

for center in centers:

c = center[0]

p = [float(i) for i in center[1:]]

centers\_point[c] = p

fd.close()

for line in sys.stdin:

line = line.strip()

split = line.split(' ')

min\_distance = 99999.99

min\_ind = -1

for cp, point in centers\_point.items():

distance = math.sqrt(((float(split[0]) - point[0]) \*\* 2) + ((float(split[1]) - point[1]) \*\* 2) + ((float(split[2]) - point[2]) \*\* 2) + ((float(split[3]) - point[3]) \*\* 2) + ((f$

if distance < min\_distance:

min\_distance = distance

min\_ind = cp

print('%s\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t' % (min\_ind, float(split[0]), float(split[1]), float(split[2]), float(split[3]), float(split[4]), float(split[5]), float(split$

Reduer

#!/usr/bin/python

import sys

curr\_cluster = None

curr\_x1\_sum = 0

curr\_x2\_sum = 0

curr\_x3\_sum = 0

curr\_x4\_sum = 0

curr\_x5\_sum = 0

curr\_x6\_sum = 0

curr\_x7\_sum = 0

curr\_array\_count = 0

for line in sys.stdin:

line = line.strip()

ln = line.split('\t')

cluster = int(ln[0])

x1 = float(ln[1])

x2 = float(ln[2])

x3 = float(ln[3])

x4 = float(ln[4])

x5 = float(ln[5])

x6 = float(ln[6])

x7 = float(ln[7])

if curr\_cluster == cluster:

curr\_x1\_sum += x1

curr\_x2\_sum += x2

curr\_x3\_sum += x3

curr\_x4\_sum += x4

curr\_x5\_sum += x5

curr\_x6\_sum += x6

curr\_x7\_sum += x7

curr\_array\_count += 1

else:

if curr\_cluster:

p1 = curr\_x1\_sum / curr\_array\_count

p2 = curr\_x2\_sum / curr\_array\_count

p3 = curr\_x3\_sum / curr\_array\_count

p4 = curr\_x4\_sum / curr\_array\_count

p5 = curr\_x5\_sum / curr\_array\_count

p6 = curr\_x6\_sum / curr\_array\_count

p7 = curr\_x7\_sum / curr\_array\_count

curr\_cluster = cluster

curr\_x1\_sum += x1

curr\_x2\_sum += x2

curr\_x3\_sum += x3

curr\_x4\_sum += x4

curr\_x5\_sum += x5

curr\_x6\_sum += x6

curr\_x7\_sum += x7

curr\_array\_count += 1

else:

if curr\_cluster:

p1 = curr\_x1\_sum / curr\_array\_count

p2 = curr\_x2\_sum / curr\_array\_count

p3 = curr\_x3\_sum / curr\_array\_count

p4 = curr\_x4\_sum / curr\_array\_count

p5 = curr\_x5\_sum / curr\_array\_count

p6 = curr\_x6\_sum / curr\_array\_count

p7 = curr\_x7\_sum / curr\_array\_count

curr\_cluster = cluster

curr\_x1\_sum += x1

curr\_x2\_sum += x2

curr\_x3\_sum += x3

curr\_x4\_sum += x4

curr\_x5\_sum += x5

curr\_x6\_sum += x6

curr\_x7\_sum += x7

curr\_array\_count += 1

if curr\_cluster == cluster:

p1 = curr\_x1\_sum / curr\_array\_count

p2 = curr\_x2\_sum / curr\_array\_count

p3 = curr\_x3\_sum / curr\_array\_count

p4 = curr\_x4\_sum / curr\_array\_count

p5 = curr\_x5\_sum / curr\_array\_count

p6 = curr\_x6\_sum / curr\_array\_count

p7 = curr\_x7\_sum / curr\_array\_count

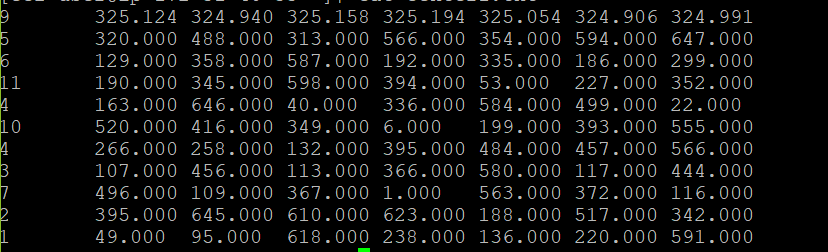
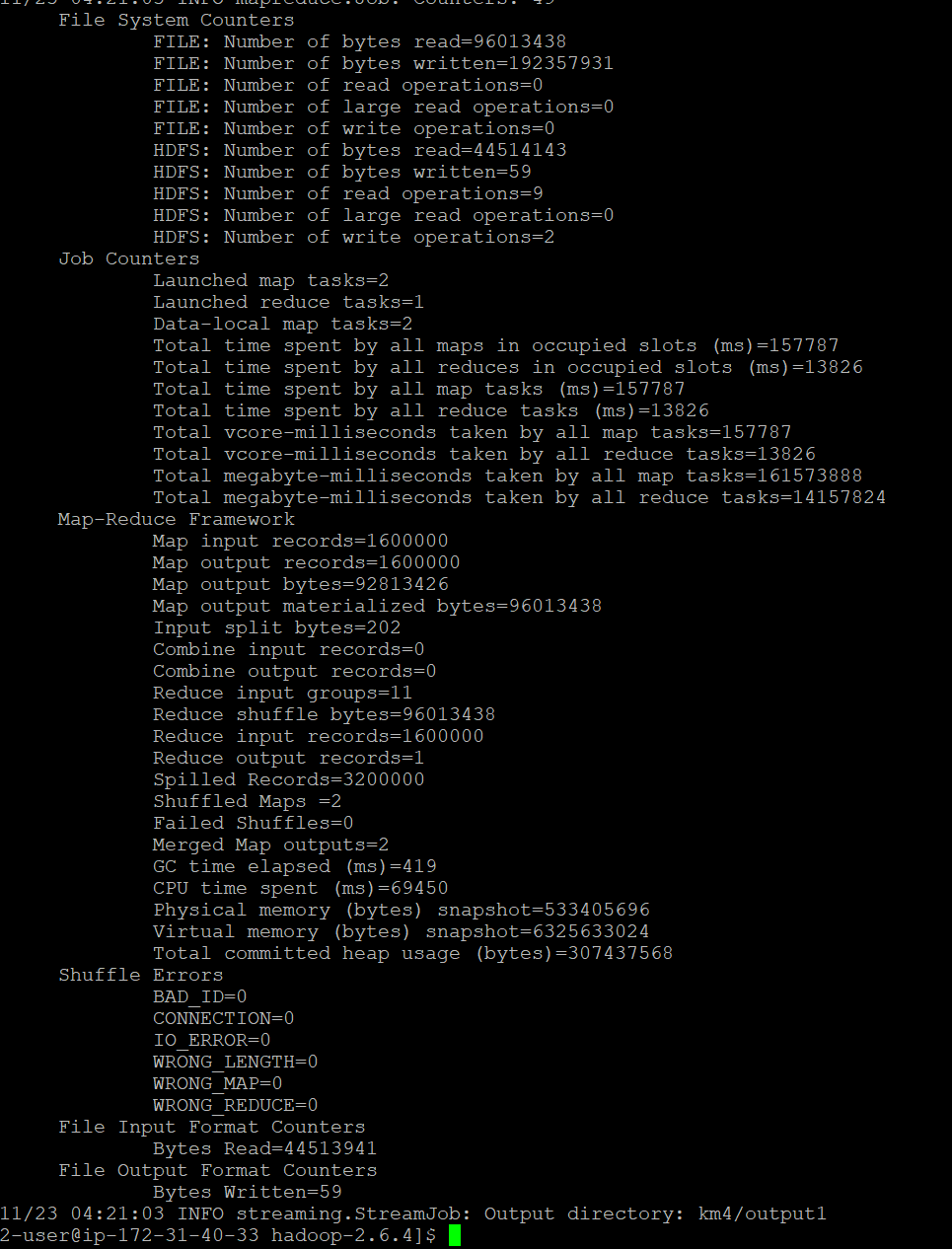
print('%d\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t' % (curr\_cluster, p1, p2, p3, p4, p5, p6, p7))

1St iteration

hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/Num\_dataset.txt -file centers.txt -mapper kmMapper3.py -file kmMapper3.py -reducer kmReducer3.py -file kmReducer3.py -output km4/output1

hadoop fs -ls km4/output1

hadoop fs -copyToLocal km4/output1/part-00000 /home/ec2-user/center1.txt

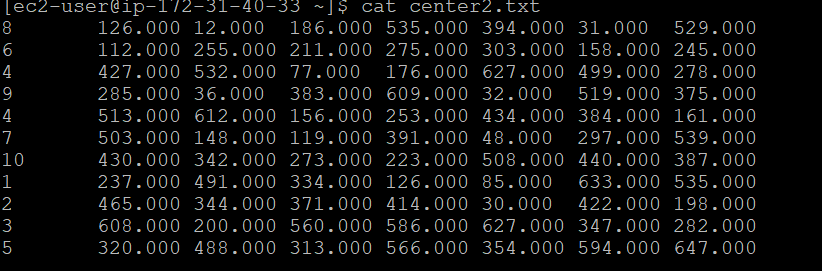
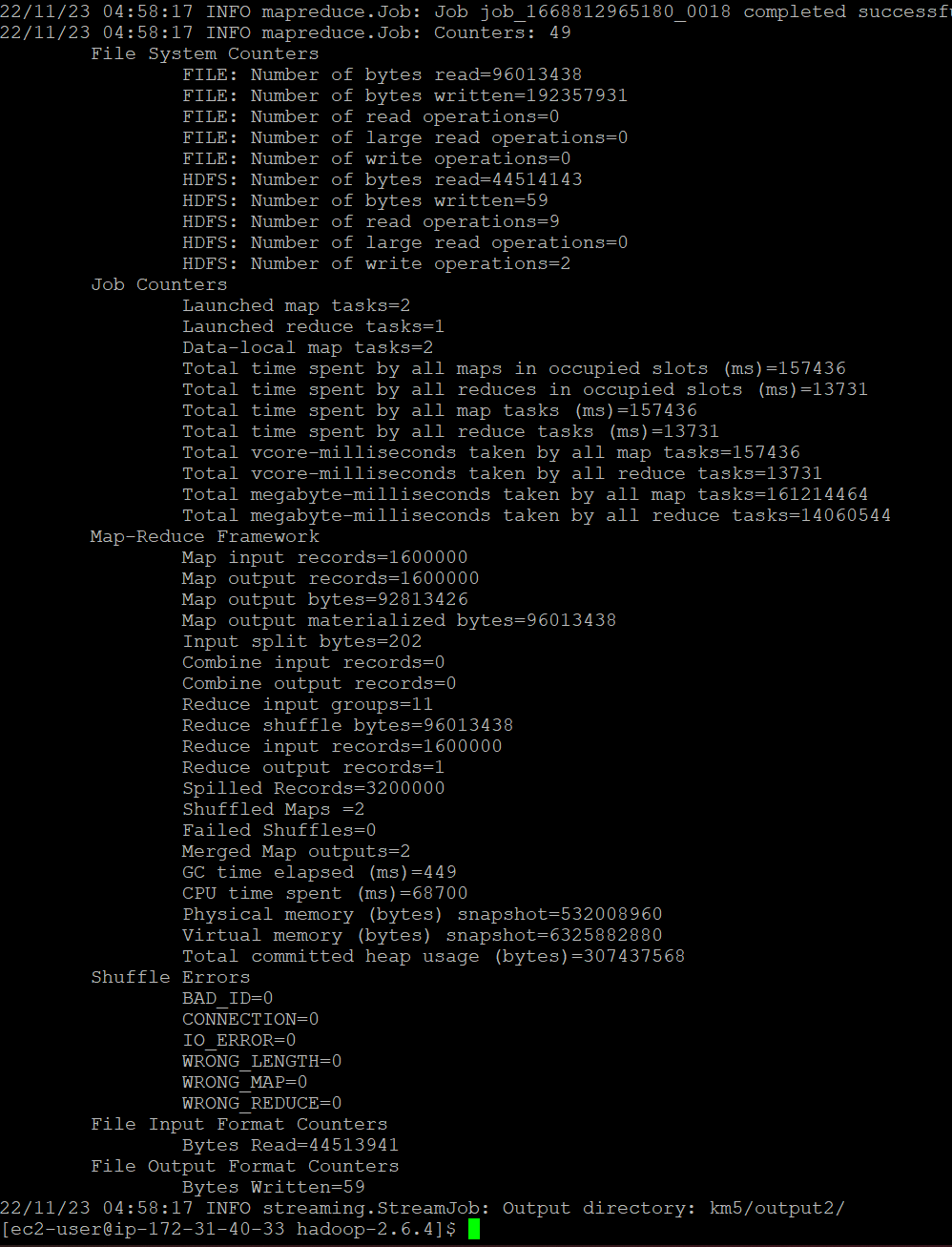


2 iteration

hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/Num\_dataset.txt -file centers.txt -mapper kmMapper3.py -file kmMapper3.py -reducer kmReducer3.py -file kmReducer3.py -output km5/output2

hadoop fs -ls km5/output2

hadoop fs -copyToLocal km5/output2/part-00000 /home/ec2-user/center2.txt

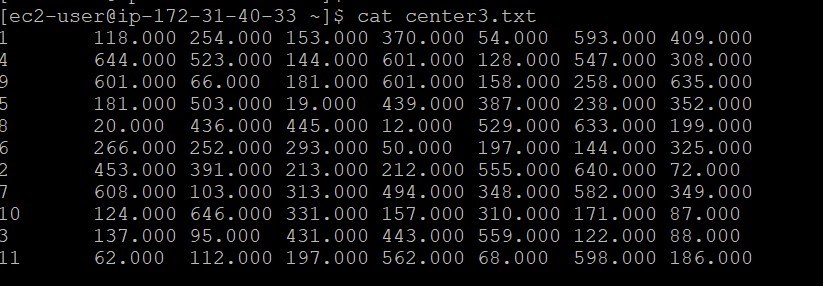
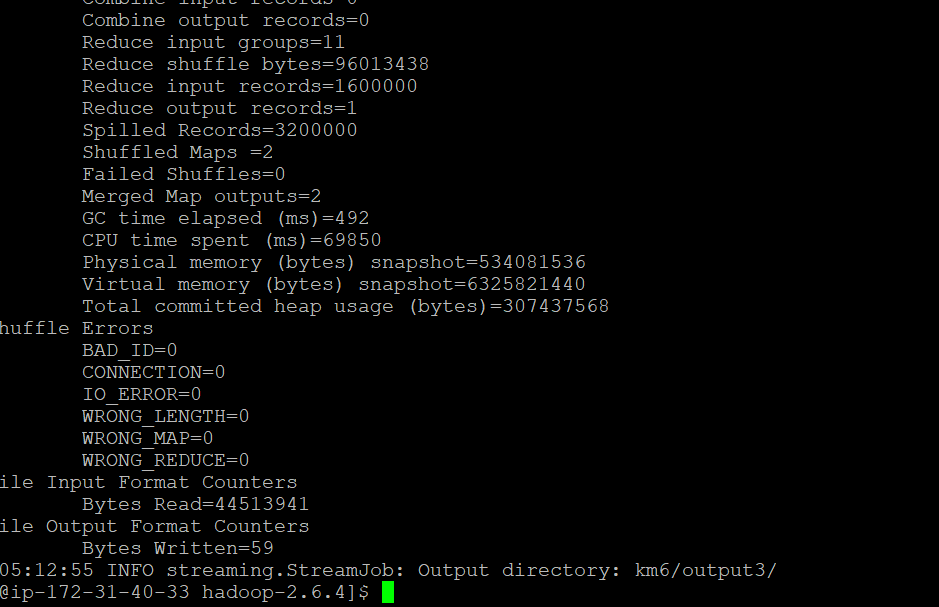


3 iteration

hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/Num\_dataset.txt -file centers.txt -mapper kmMapper3.py -file kmMapper3.py -reducer kmReducer3.py -file kmReducer3.py -output km6/output3

hadoop fs -ls km6/output3

hadoop fs -copyToLocal km6/output3/part-00000 /home/ec2-user/center3.txt

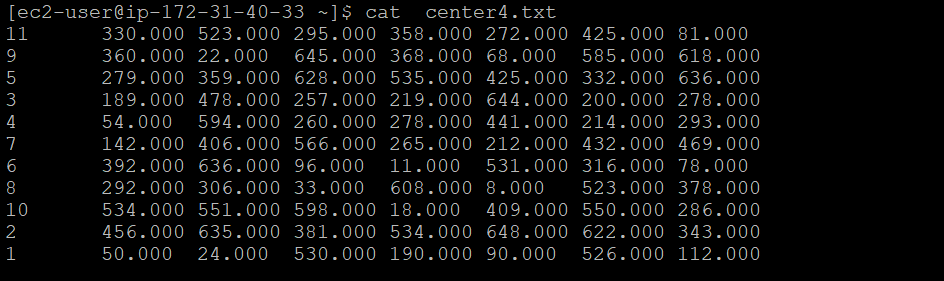


4 iteration

hadoop jar ./share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -input /user/ec2-user/data/Num\_dataset.txt -file centers.txt -mapper kmMapper3.py -file kmMapper3.py -reducer kmReducer3.py -file kmReducer3.py -output km7/output4

hadoop fs -ls km7/output4

hadoop fs -copyToLocal km7/output4/part-00000 /home/ec2-user/center4.txt



Submit a single document containing your written answers. Be sure that this document contains your name and “CSC 555 Final” at the top.