**CSC 555: Mining Big Data**

Project, Phase 2 (due Sunday March 24th)

In this part of the project, you will execute queries using Hive, Pig and Hadoop streaming and develop a custom version of KMeans clustering. The schema is available below, but don’t forget to apply the correct delimiter:

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM_schema_hive.sql>

The data is available at (this is Scale1, the smallest denomination of this benchmark)

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/>

In your submission, please note what cluster you are using. Please be sure to submit all code (pig, python and Hive). 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 whole output). An answer without code will not receive credit.

**I highly recommend creating a small sample input** (e.g., by running head lineorder.tbl > lineorder.tbl.sample) and testing your code with it. You can run **head -n 500 lineorder.tbl** to get a specific number of lines.

NOTE: the total number of points adds up to 70 because Phase I is worth 30 of the project.

# Part 1: Data Transformation (15 pts)

Transform part.tbl table into a \*-separated (‘\*’) file: Use Hive, MapReduce with HadoopStreaming and Pig (i.e. 3 different solutions).

In all solutions you must switch odd and even columns (i.e., switch the positions of columns 1 and 2, columns 3 and 4, etc.). You do not need to transform the columns in any way, just a new data file.

Using my multi-node cluster

**Hive**

CREATE TABLE part (

p\_partkey INT,

p\_name VARCHAR(22),

p\_mfgr VARCHAR(6),

p\_category VARCHAR(7),

p\_brand1 VARCHAR(9),

p\_color VARCHAR(11),

p\_type VARCHAR(25),

p\_size INT,

p\_container VARCHAR(10)

)ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/part.tbl' OVERWRITE INTO TABLE part;

**Python code (colSwitcher.py):**

#!/usr/bin/python

import sys

for line in sys.stdin:

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

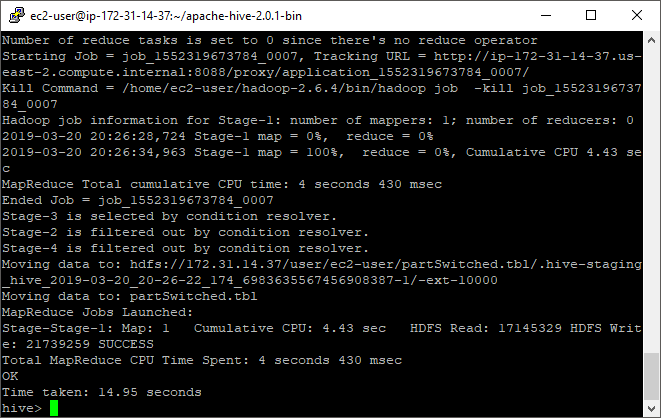
print '\*'.join([line[1], line[0], line[3], line[2], line[5], line[4], line[7], line[6], line[8]])

**Commands:**

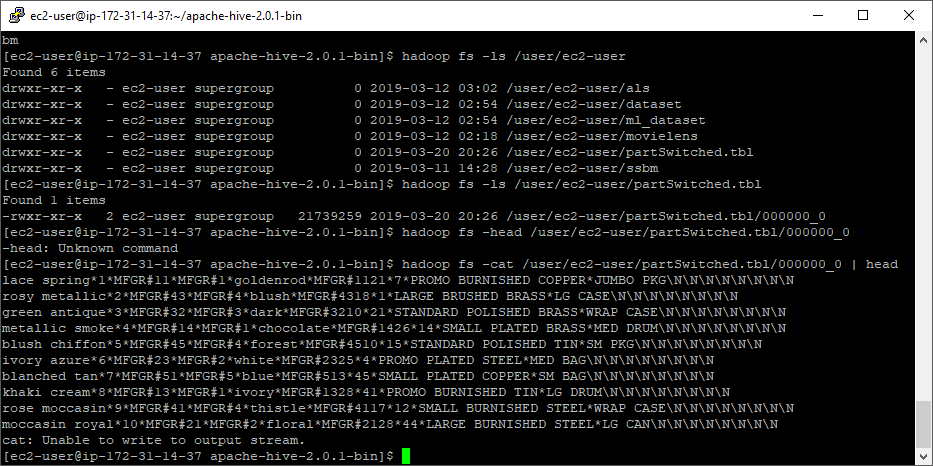
ADD FILE /home/ec2-user/colSwitcher.py;

INSERT OVERWRITE DIRECTORY 'partSwitched.tbl' SELECT TRANSFORM (p\_partkey, p\_name, p\_mfgr, p\_category, p\_brand1, p\_color, p\_type, p\_size, p\_container) USING 'colSwitcher.py' AS (p\_name, p\_partkey, p\_category, p\_mfgr, p\_color, p\_brand1, p\_size, p\_type, p\_container) FROM part;

# Completed Run:



**Output, first ten rows:**



**Hadoop Streaming**

There is no need for a custom mapper for this exercise, so I used the linux cat function as the mapper. The reducer code is:

**colSwitcherReducer.py**

#!/usr/bin/python

import sys

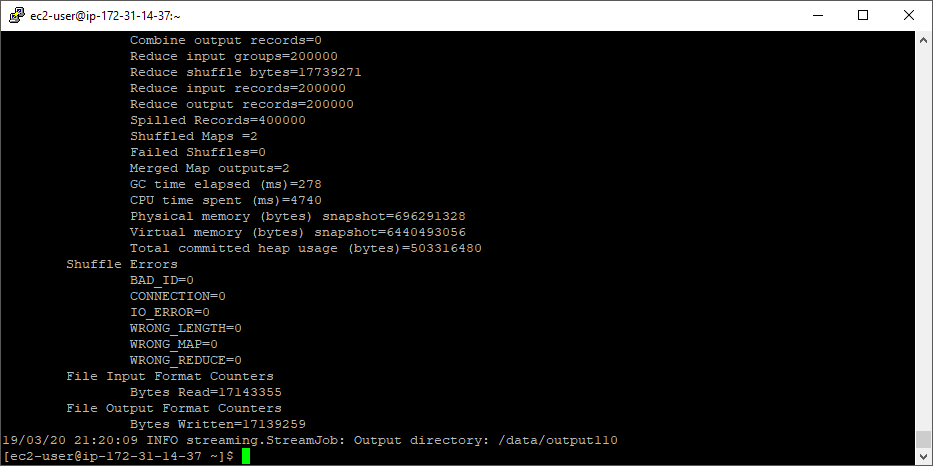
for line in sys.stdin:

line = line.strip().split('|')

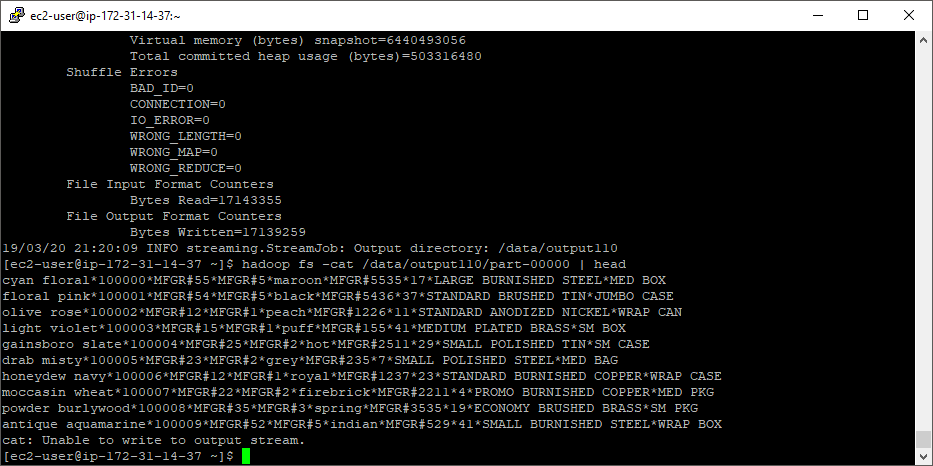
print "%s\*%s\*%s\*%s\*%s\*%s\*%s\*%s\*%s" % (line[1],line[0],line[3],line[2],line[5],line[4],line[7],line[6],line[8])

**Command:**

hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/ssbm/part.tbl -output /data/output110 -mapper /bin/cat -reducer colSwitcherReducer.py -file colSwitcherReducer.py



**Output, first ten rows:**



**Pig**

**Load the Data:**

PartData = LOAD '/user/ec2-user/ssbm/part.tbl' USING PigStorage('|') AS (p\_partkey:int, p\_name:chararray, p\_mfgr:chararray, p\_category:chararray, p\_brand1:chararray, p\_color:chararray, p\_type:chararray, p\_size:int, p\_container:chararray);

**Verify Data Loaded:**

PartG = GROUP PartData ALL;

Count = FOREACH PartG GENERATE COUNT(PartData);

DUMP Count;

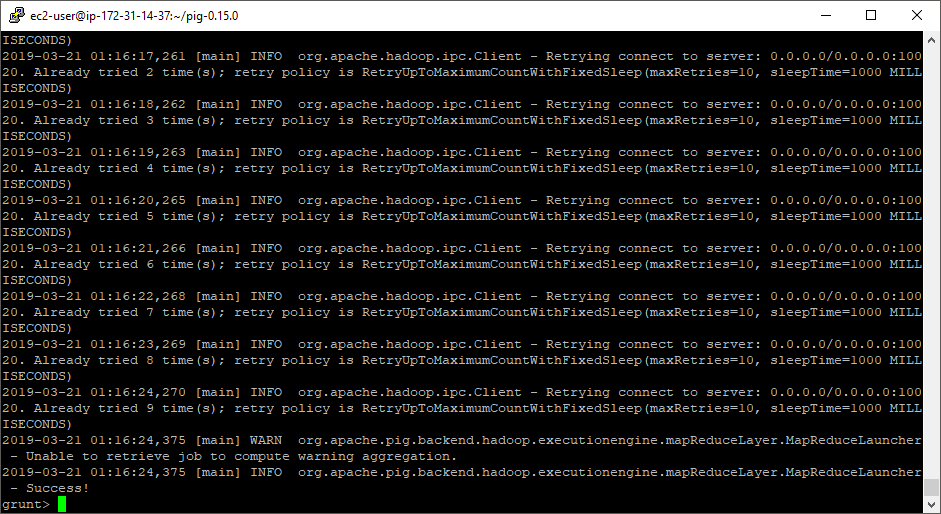
# 

**Switch Columns:**

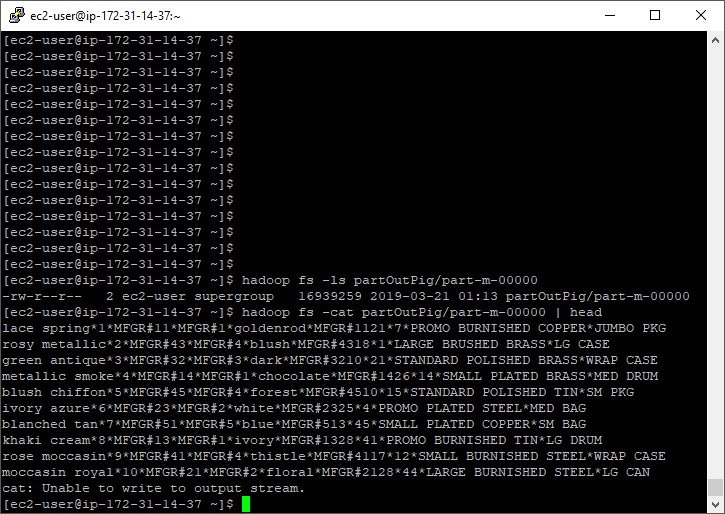
PartSwitchedPig = FOREACH PartData GENERATE p\_name, p\_partkey, p\_category, p\_mfgr, p\_color, p\_brand1, p\_size, p\_type, p\_container;

**Write to file:**

STORE PartSwitchedPig INTO 'partOutPig' USING PigStorage('\*');



**Output, first ten rows:**



# Part 2: Querying (25 pts)

Implement the following query:

select lo\_quantity, c\_nation, sum(lo\_revenue)

from customer, lineorder

where lo\_custkey = c\_custkey

and c\_region = 'AMERICA'

and lo\_discount BETWEEN 3 and 5

group by lo\_quantity, c\_nation;

using Hive, MapReduce with HadoopStreaming and Pig (i.e. 3 different solutions). I Hive, this merely requires pasting the query into the Hive prompt and timing it. In Hadoop streaming, this will require a total of 2 passes (one for join and another one for GROUP BY).

**Hive:**

**Create and load tables:**

CREATE TABLE lineorder (

lo\_orderkey INT,

lo\_linenumber INT,

lo\_custkey INT,

lo\_partkey INT,

lo\_suppkey INT,

lo\_orderdate INT,

lo\_orderpriority VARCHAR(15),

lo\_shippriority VARCHAR(1),

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 VARCHAR(10)

)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/lineorder.tbl' OVERWRITE INTO TABLE lineorder;

# CREATE TABLE customer (

# c\_custkey INT,

# c\_name VARCHAR(25),

# c\_address VARCHAR (25),

# c\_city VARCHAR (10),

# c\_nation VARCHAR (15),

# c\_region VARCHAR (12),

# c\_phone VARCHAR (15),

# c\_mktsegment VARCHAR (10)

# )

# ROW FORMAT DELIMITED FIELDS

# TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/customer.tbl' OVERWRITE INTO TABLE customer;

**Execute sql statement:**

select lo\_quantity, c\_nation, sum(lo\_revenue)

from customer, lineorder

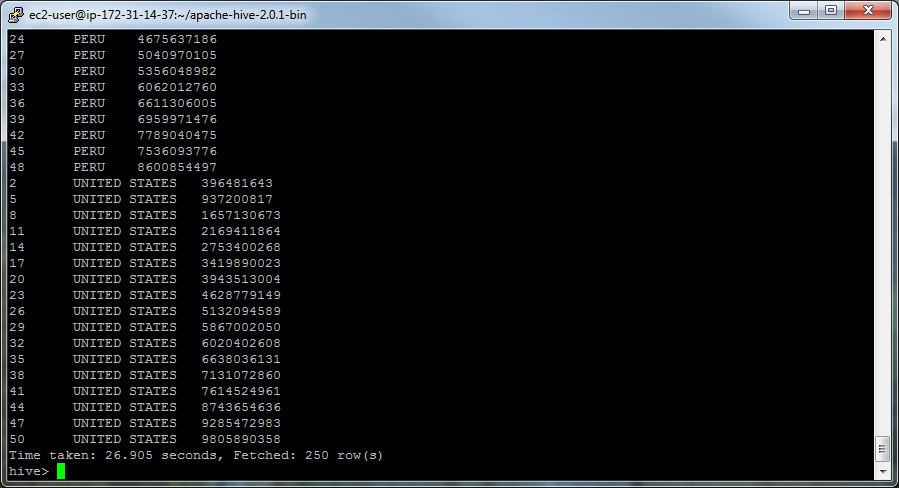
where lo\_custkey = c\_custkey

and c\_region = 'AMERICA'

and lo\_discount BETWEEN 3 and 5

group by lo\_quantity, c\_nation;

**End of output with time taken:**



**Hadoop Streaming:**

**Join**

**lineCustMapJoin.py**

#!/usr/bin/python

import sys

# input comes from STDIN (standard input)

for line in sys.stdin:

line = line.strip().split('|')

if line[1].startswith('Customer#'):

if line[5] == 'AMERICA': # Return on matching records

print line[0], '\t', line[4], '\t', 'customer'

# lineorder

else:

if 3 <= int(line[11]) <= 5: # Return on matching records

print line[2], '\t', line[8], '\t', line[12], '\t', 'lineorder'

**lineCustReduceJoin.py**

#!/usr/bin/python

import sys

currentKey = None

quantity = []

revenue = []

nation = ''

# input comes from STDIN

for line in sys.stdin:

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

key = split[0] # key is customer id

value = '\t'.join(split[1:])

if currentKey == key: # Same key

if value.endswith('lineorder'):

quantity.extend([split[1]])

revenue.extend([split[2]])

if value.endswith('customer'):

nation = split[1]

else:

# Do not print anything until all records

# for a key have been seen, this is signaled

# by currentKey != key

# Check for values and then iterate results

lenQuantity = len(quantity)

lenNation = len(nation)

if (lenQuantity\*lenNation > 0):

i = 0

while i < lenQuantity:

print quantity[i], '\t', nation, '\t', revenue[i]

i += 1

# reset values

quantity = []

revenue = []

nation = ''

if value.endswith('lineorder'):

quantity.extend([split[1]])

revenue.extend([split[2]])

if value.endswith('customer'):

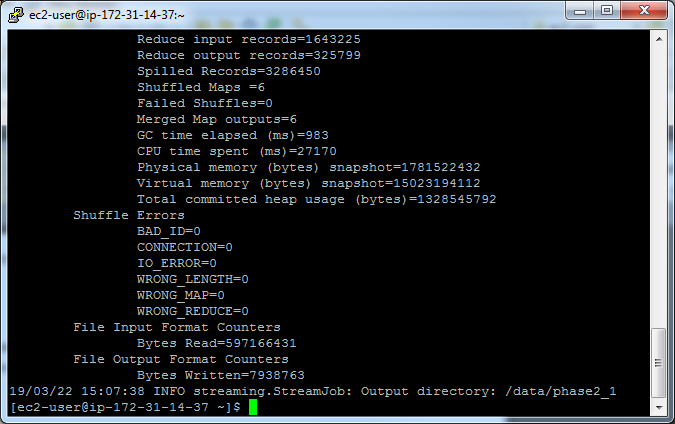
nation = split[1]

# set the current key at the end of each iteration

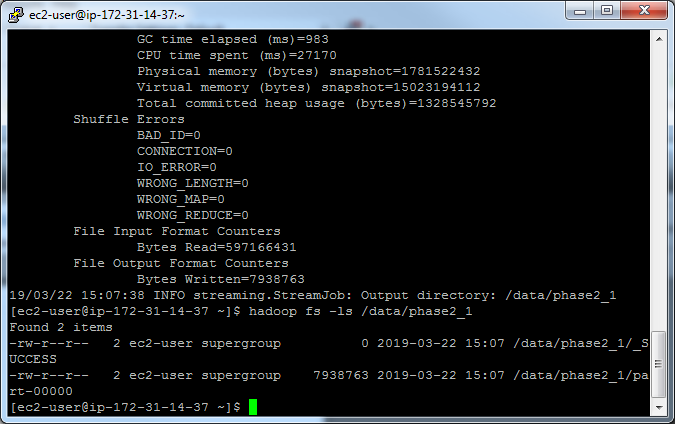
currentKey = key

**Commands**:

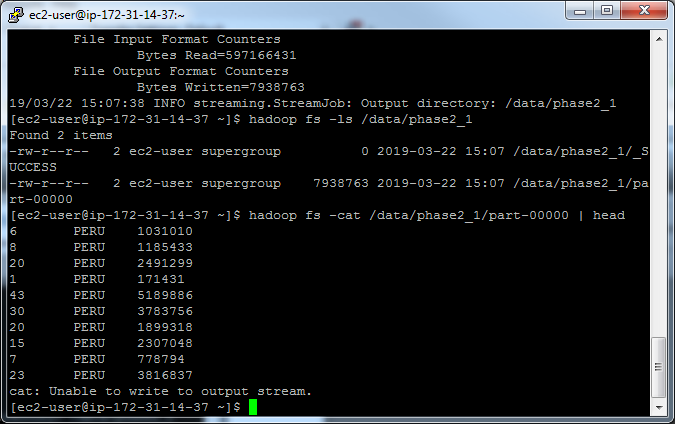
hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/phase2 -output /data/phase2\_1 -mapper lineCustMapJoin.py -reducer lineCustReduceJoin.py -file lineCustMapJoin.py -file lineCustReduceJoin.py



hadoop fs -ls /data/phase2\_1



hadoop fs -cat /data/phase2\_1/part-00000 | head



**Group:**

**lineCustReduceGroup.py**

#!/usr/bin/python

import sys

curr\_id = None

curr\_tot = 0

id = None

# The input comes from standard input (line by line)

for line in sys.stdin:

# parse the line and split it by '\t'

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

# grab the key

# values include some whitespace, removing here

id = line[0].strip() + '\t' + line[1].strip()

# grab the value (int)

val = int(line[2])

if curr\_id == id:

curr\_tot += val

else:

if curr\_id: # output the sum, single key completed

print '%s\t%d' % (curr\_id, curr\_tot)

curr\_tot = val

# set curr\_id to id at end of each iteration

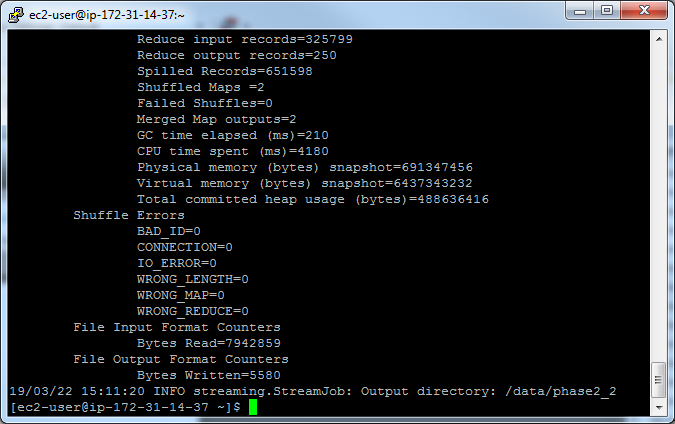
curr\_id = id

# output the last key

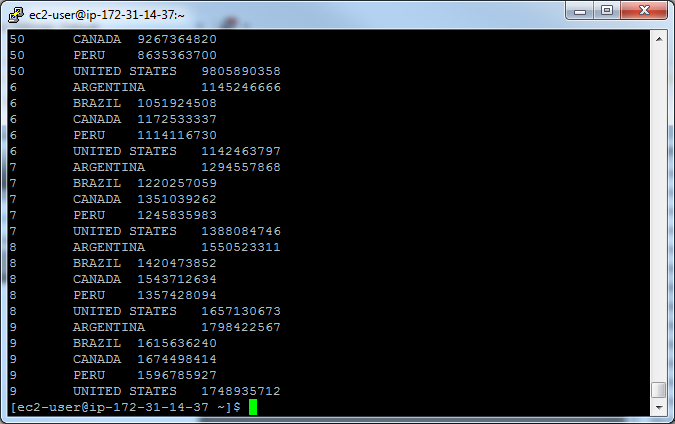
if curr\_id == id:

print '%s\t%d' % (curr\_id, curr\_tot)

hadoop jar hadoop-streaming-2.6.4.jar -D stream.num.map.output.key.fields=2 -input /data/phase2\_01/part-00000 -output /data/phase2\_06 -mapper /bin/cat -reducer lineCustReduceGroup.py -file lineCustReduceGroup.py



hadoop fs -cat /data/phase2\_2/part-00000



**Pig:**

**Load Tables:**

lineorder = LOAD '/user/ec2-user/ssbm/lineorder.tbl' USING PigStorage('|')

AS (lo\_orderkey:int,

lo\_linenumber:int,

lo\_custkey:int,

lo\_partkey:int,

lo\_suppkey:int,

lo\_orderdate:int,

lo\_orderpriority:chararray,

lo\_shippriority:chararray,

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:chararray

);

customer = LOAD '/user/ec2-user/ssbm/customer.tbl' USING PigStorage('|')

AS (c\_custkey:int,

c\_name:chararray,

c\_address:chararray,

c\_city:chararray,

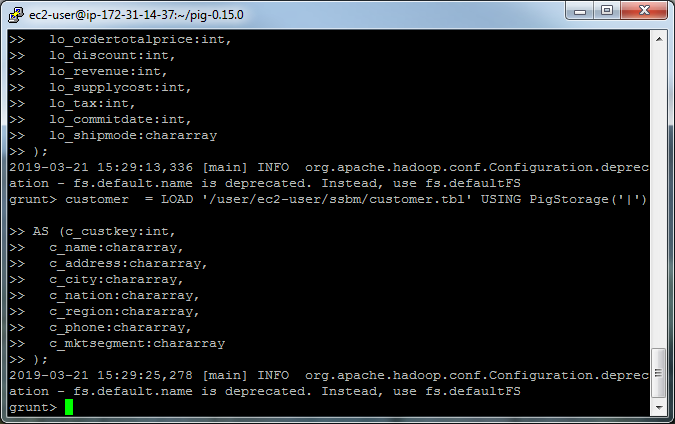
c\_nation:chararray,

c\_region:chararray,

c\_phone:chararray,

c\_mktsegment:chararray

);



**Execution steps:**

FilteredLineorder = FILTER lineorder BY lo\_discount >= 3 AND lo\_discount <= 5;

FilteredCustomer = FILTER customer BY c\_region == 'AMERICA';

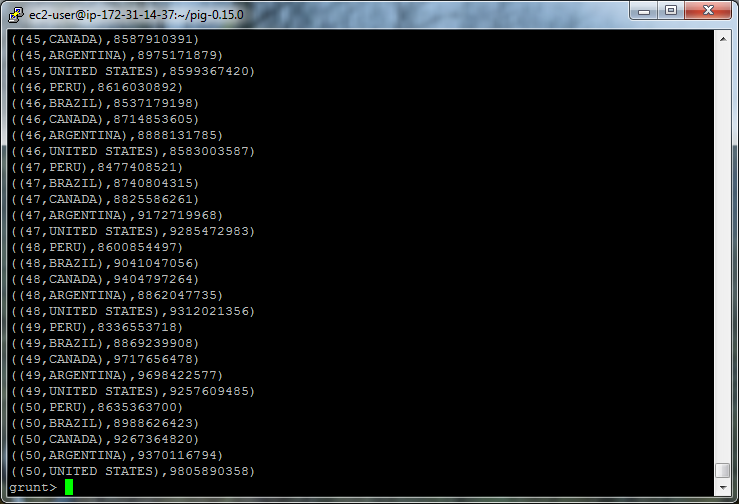
JoinedData = JOIN FilteredLineorder BY (lo\_custkey), FilteredCustomer BY (c\_custkey);

GroupedData = GROUP JoinedData BY (lo\_quantity, c\_nation);

Result = FOREACH GroupedData GENERATE group, SUM(JoinedData.lo\_revenue) as rev;

DUMP Result;

I had difficulty with displaying the non-summed columns so I left them out of the command. What’s displayed still includes the grouped columns. Moving on in the interest of time.



# Part 3: Clustering (30 pts)

Create a new numeric file with 25,000 rows and 3 columns, separated by space – you can generate numeric data as you prefer, but submit whatever code that you have used.

1. (5 pts) Using Mahout synthetic clustering as you have in a previous assignment on sample data. This entails running the **same** clustering command, but substituting your own input data instead of the sample.

For this exercise, I used the wine data set from UCI: <https://archive.ics.uci.edu/ml/datasets/wine>

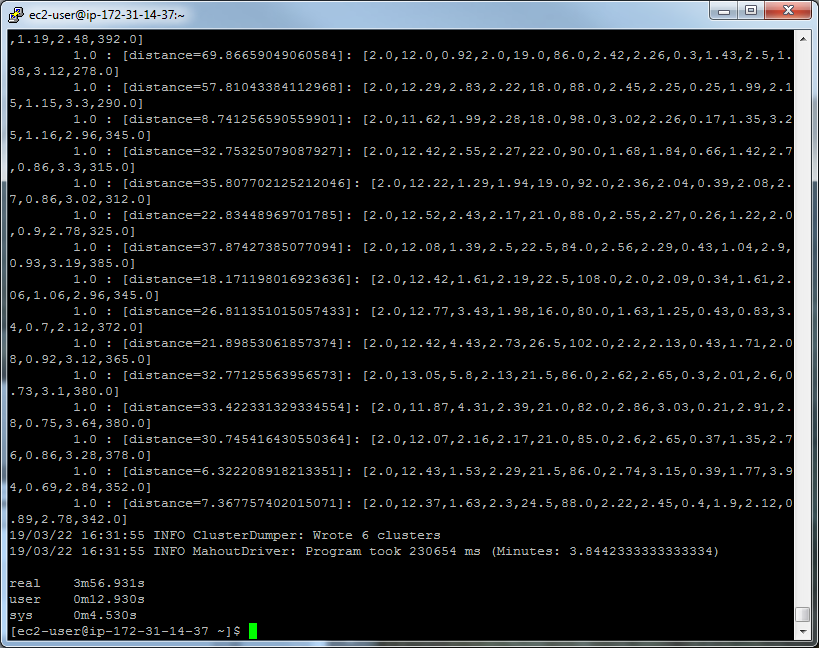
**Commands:**

wget <https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data>

**Note**: mahout did not like commas to separate the variables, so I used nano to replace commas with a space.

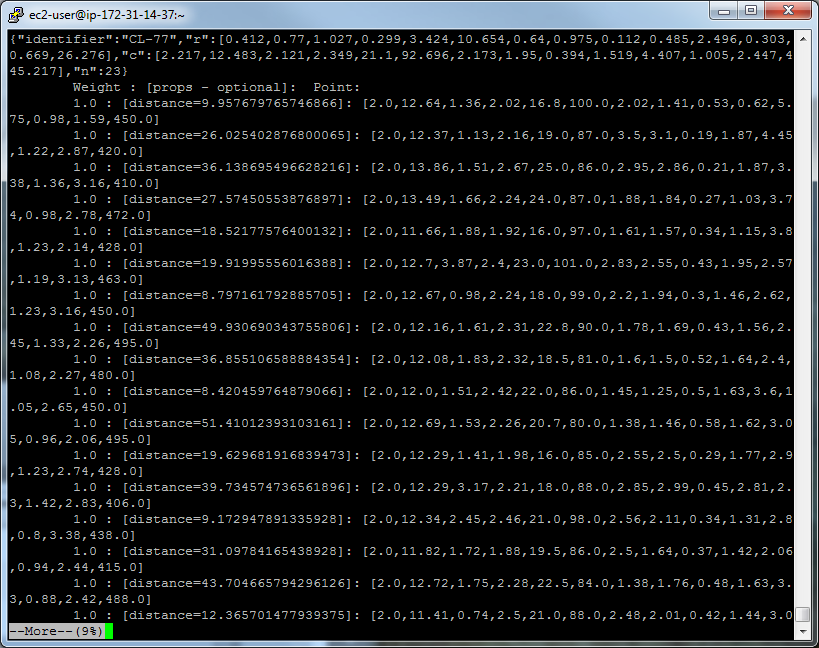
hadoop fs -put wine.data testdata/

time mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job



mahout clusterdump --input output/clusters-10-final --pointsDir output/clusteredPoints --output clusteranalyze.txt

more clusteranalyze.txt

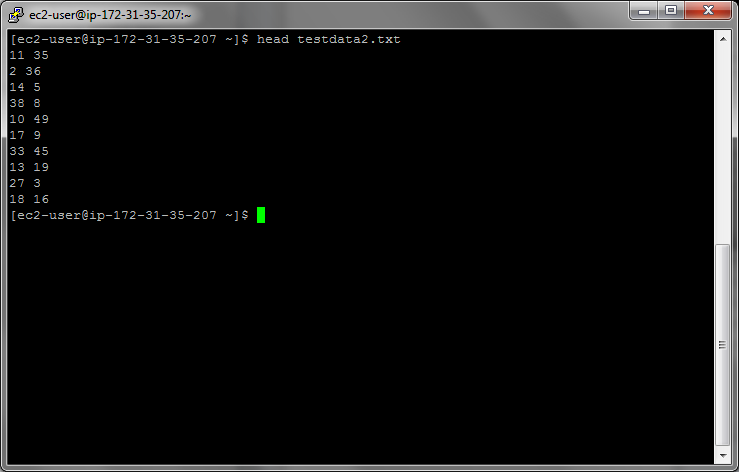


1. (25 pts) Using Hadoop streaming perform four iterations manually **using 6 centers** (initially with randomly chosen centers). This would require passing a text file with cluster centers using -file option, opening the centers.txt in the mapper with open(‘centers.txt’, ‘r’) and assigning a key to each point based on which center is the closest to each particular point. Your reducer would then compute the new centers, and at that point the iteration is done and the output of the reducer with new centers can be given to the next pass of the same code.

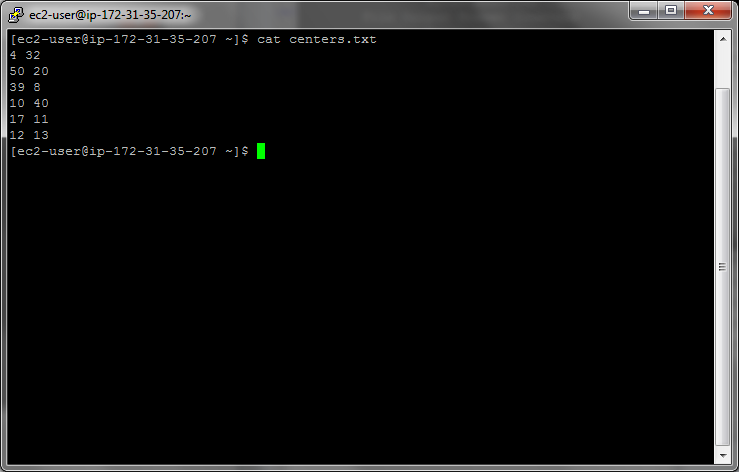
The only difference between first and subsequent iteration is that in first iteration you have to pick the initial centers. Starting from 2nd iteration, the centers will be given to you by a previous pass of KMeans.

I used the single-node Hadoop instance for this exercise.

First, I used an online random sequence generator (<https://www.random.org/sequences/?mode=advanced>) to generate 100 x and y variables. Here’s a screen shot. The full list is at the end of this document.



Then I randomly picked six starting centers from the list:



**Code:**

**kmeansMapper.py**

#!/usr/bin/python

import sys

import math

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

centers = []

for line in fd:

line = line.strip()

vals = line.split(' ')

centers.extend([vals])

fd.close()

for line in sys.stdin:

line = line.strip()

vals = line.split(' ')

clusterNum = None

distance = None

i = 0

#compare to each center and store the smallest distance

for center in centers:

euclidDist = math.sqrt( (float(vals[0])-float(center[0]))\*\*2 + (float(v$

if clusterNum:

if euclidDist < distance:

clusterNum = i+1

distance = euclidDist

else: #always record the first cluster

clusterNum = i+1

distance = euclidDist

i += 1

print clusterNum, '\t', vals[0], '\t', vals[1]

**kmeansReducer.py**

#!/usr/bin/python

import sys

currId = None # this is the "current" key

currXs = []

currYs = []

id = None

# The input comes from standard input (line by line)

for line in sys.stdin:

line = line.strip()

ln = line.split('\t')

id = ln[0]

if currId == id:

currXs.append(float(ln[1]))

currYs.append(float(ln[2]))

else:

if currId:

#calculate center

centerX = sum(currXs)/len(currXs)

centerY = sum(currYs)/len(currYs)

print '%s %s %s %s' % (centerX, centerY, currId, zip(currXs, cu$

currXs = []

currYs = []

currId = id

currXs.append(float(ln[1]))

currYs.append(float(ln[2]))

# output the last key

if currId == id:

#calculate center

centerX = sum(currXs)/len(currXs)

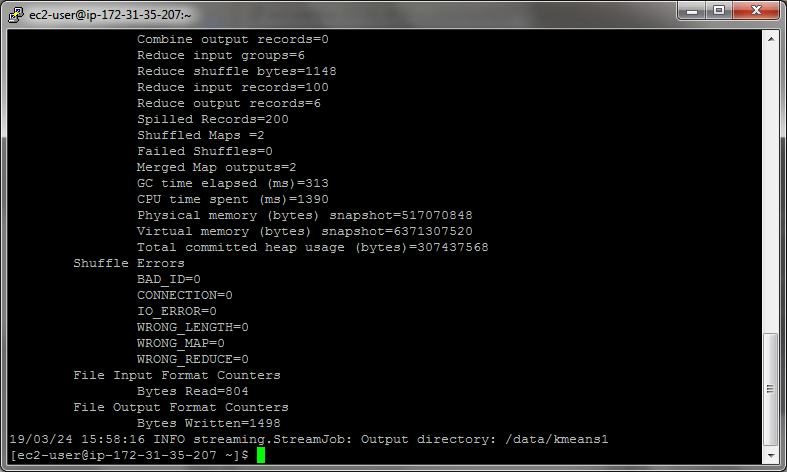
centerY = sum(currYs)/len(currYs)

print '%s %s %s %s' % (centerX, centerY, currId, zip(currXs, currYs))

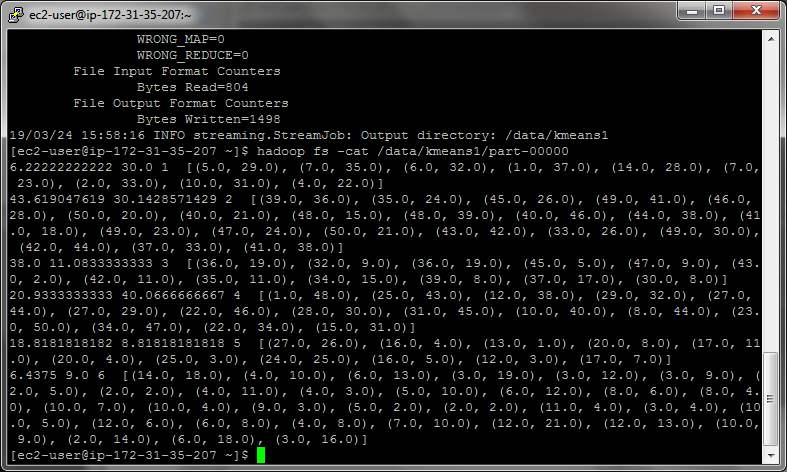
**Executions** (note cluster output text is also at the end of this file)**:**

**Execution 1:**

hadoop jar hadoop-streaming-2.6.4.jar -input /data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans1



hadoop fs -cat /data/kmeans1/part-00000



The first two values are the new center points, the third value is the cluster number and the sets of pairs are the points belonging to those clusters.

**Execution 2:**

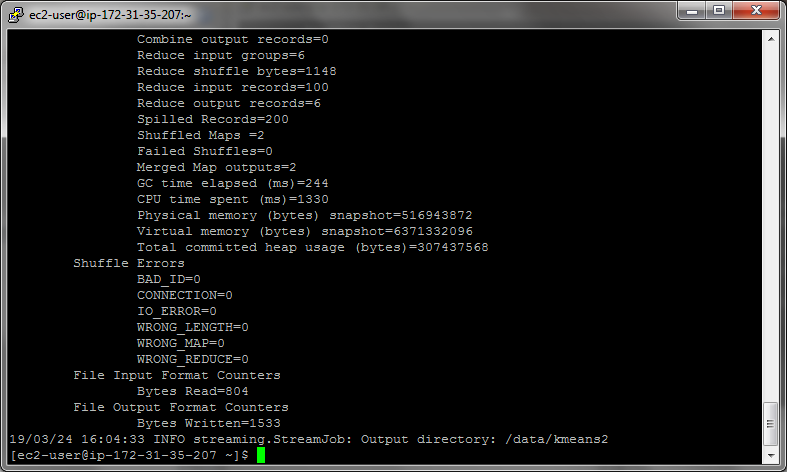
Replace the centers file:

rm centers.txt

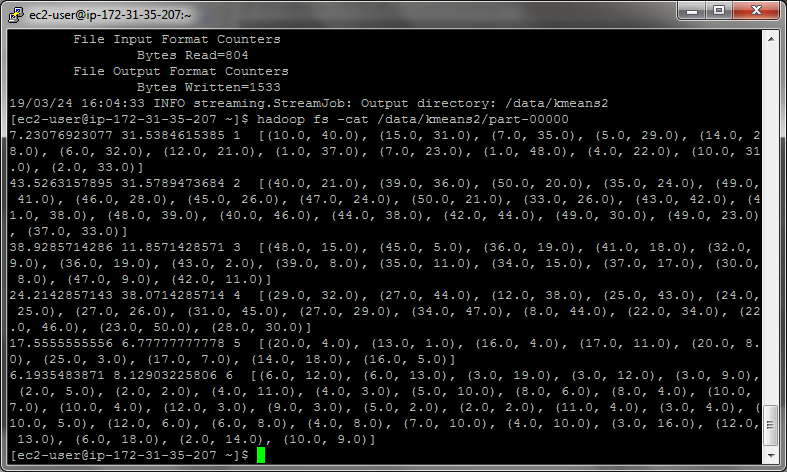
hadoop fs -get /data/kmeans1/part-00000 centers.txt

Run with new centers:

hadoop jar hadoop-streaming-2.6.4.jar -input /data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans2



hadoop fs -cat /data/kmeans2/part-00000



**Execution 3:**

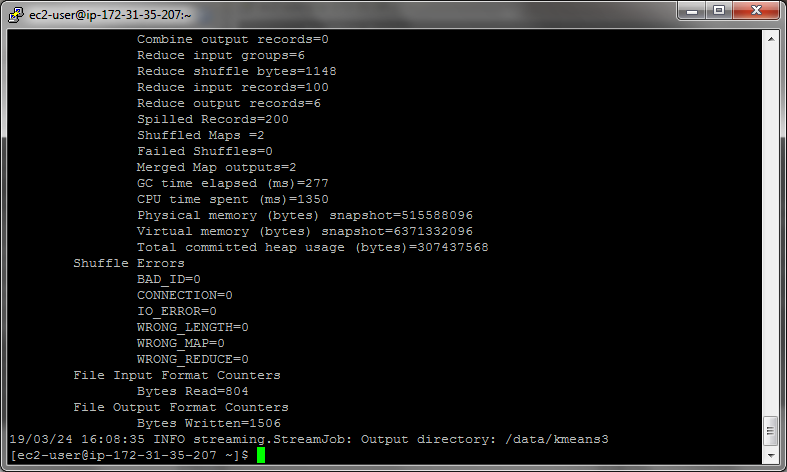
Replace the centers file:

rm centers.txt

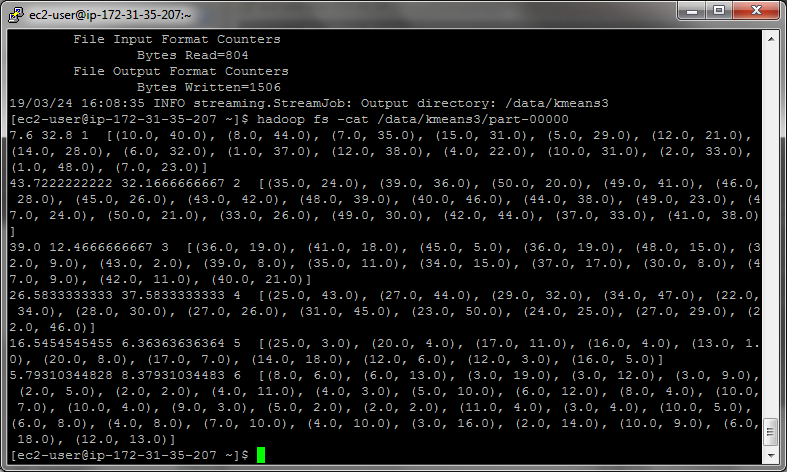
hadoop fs -get /data/kmeans2/part-00000 centers.txt

Run with new centers:

hadoop jar hadoop-streaming-2.6.4.jar -input /data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans3



hadoop fs -cat /data/kmeans3/part-00000



**Execution 4:**

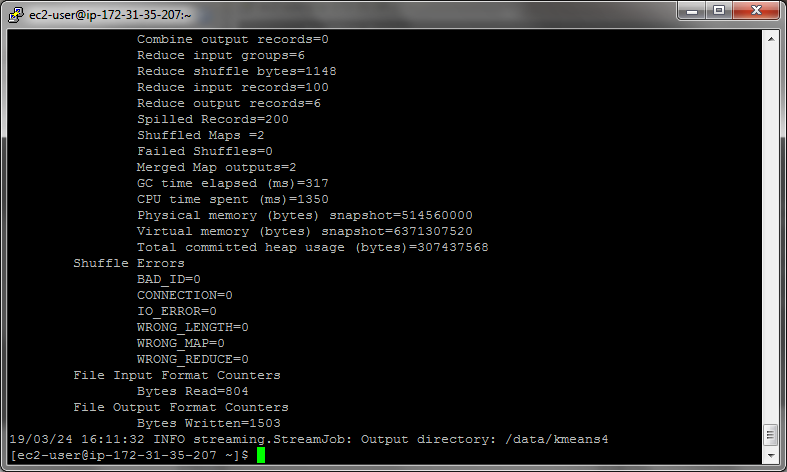
Replace the centers file:

rm centers.txt

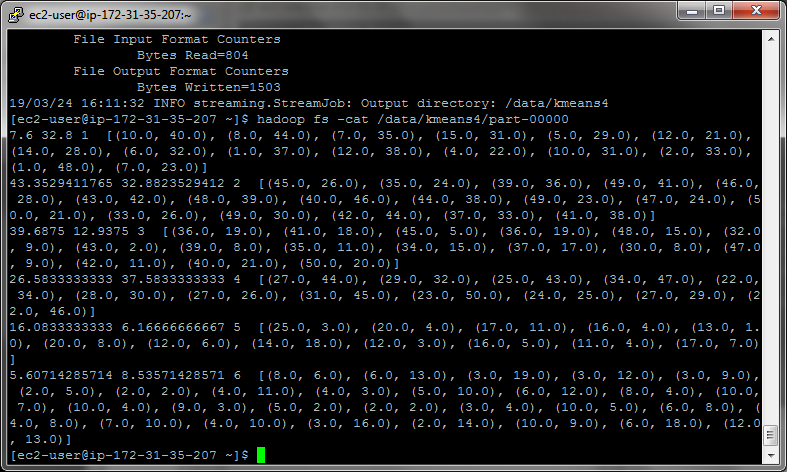
hadoop fs -get /data/kmeans3/part-00000 centers.txt

Run with new centers:

hadoop jar hadoop-streaming-2.6.4.jar -input /data/testdata2.txt -file centers.txt -mapper kmeansMapper.py -file kmeansMapper.py -reducer kmeansReducer.py -file kmeansReducer.py -output /data/kmeans4



hadoop fs -cat /data/kmeans4/part-00000



**Extra credit (7 pts)**: Create the equivalent of KMeans driver from Mahout. That is, write a python script that will automatically execute the hadoop streaming command, then get the new centers from HDFS and repeat the command. This will be easiest to do if you write your reducer to output just the centers (without the key) to HDFS. This way, all you have to do is to execute the get command to get the new centers (you can hard-code the locations of output in HDFS into your script).

Submit a single document containing your written answers. Be sure that this document contains your name and “CSC 555 Project Phase 2” at the top.

**testdata2.txt**

11 35

2 36

14 5

38 8

10 49

17 9

33 45

13 19

27 3

18 16

28 40

4 32

46 25

21 15

29 7

43 22

37 48

44 34

20 12

31 6

39 26

41 23

50 47

24 30

1 42

45 26

46 28

1 48

27 44

35 24

29 32

7 23

16 5

14 18

50 20

39 36

40 21

2 33

10 31

4 22

42 11

6 13

47 9

30 8

37 17

3 19

12 38

34 15

25 43

49 41

17 7

35 11

39 8

43 2

48 15

24 25

23 50

41 38

3 16

12 21

22 46

13 1

20 4

37 33

36 19

5 29

34 47

14 28

42 44

49 30

6 18

10 40

31 45

27 26

32 9

28 30

33 26

43 42

16 4

50 21

47 24

27 29

7 35

15 31

49 23

41 18

17 11

2 14

10 9

25 3

22 34

20 8

44 38

40 46

36 19

48 39

12 13

1 37

6 32

45 5

Clustering Output:

**Execution 1:**

6.22222222222 30.0 1 [(5.0, 29.0), (7.0, 35.0), (6.0, 32.0), (1.0, 37.0), (14.0, 28.0), (7.0, 23.0), (2.0, 33.0), (10.0, 31.0), (4.0, 22.0)]

43.619047619 30.1428571429 2 [(39.0, 36.0), (35.0, 24.0), (45.0, 26.0), (49.0, 41.0), (46.0, 28.0), (50.0, 20.0), (40.0, 21.0), (48.0, 15.0), (48.0, 39.0), (40.0, 46.0), (44.0, 38.0), (41.0, 18.0), (49.0, 23.0), (47.0, 24.0), (50.0, 21.0), (43.0, 42.0), (33.0, 26.0), (49.0, 30.0), (42.0, 44.0), (37.0, 33.0), (41.0, 38.0)]

38.0 11.0833333333 3 [(36.0, 19.0), (32.0, 9.0), (36.0, 19.0), (45.0, 5.0), (47.0, 9.0), (43.0, 2.0), (42.0, 11.0), (35.0, 11.0), (34.0, 15.0), (39.0, 8.0), (37.0, 17.0), (30.0, 8.0)]

20.9333333333 40.0666666667 4 [(1.0, 48.0), (25.0, 43.0), (12.0, 38.0), (29.0, 32.0), (27.0, 44.0), (27.0, 29.0), (22.0, 46.0), (28.0, 30.0), (31.0, 45.0), (10.0, 40.0), (8.0, 44.0), (23.0, 50.0), (34.0, 47.0), (22.0, 34.0), (15.0, 31.0)]

18.8181818182 8.81818181818 5 [(27.0, 26.0), (16.0, 4.0), (13.0, 1.0), (20.0, 8.0), (17.0, 11.0), (20.0, 4.0), (25.0, 3.0), (24.0, 25.0), (16.0, 5.0), (12.0, 3.0), (17.0, 7.0)]

6.4375 9.0 6 [(14.0, 18.0), (4.0, 10.0), (6.0, 13.0), (3.0, 19.0), (3.0, 12.0), (3.0, 9.0), (2.0, 5.0), (2.0, 2.0), (4.0, 11.0), (4.0, 3.0), (5.0, 10.0), (6.0, 12.0), (8.0, 6.0), (8.0, 4.0), (10.0, 7.0), (10.0, 4.0), (9.0, 3.0), (5.0, 2.0), (2.0, 2.0), (11.0, 4.0), (3.0, 4.0), (10.0, 5.0), (12.0, 6.0), (6.0, 8.0), (4.0, 8.0), (7.0, 10.0), (12.0, 21.0), (12.0, 13.0), (10.0, 9.0), (2.0, 14.0), (6.0, 18.0), (3.0, 16.0)]

**Execution 2:**

7.23076923077 31.5384615385 1 [(10.0, 40.0), (15.0, 31.0), (7.0, 35.0), (5.0, 29.0), (14.0, 28.0), (6.0, 32.0), (12.0, 21.0), (1.0, 37.0), (7.0, 23.0), (1.0, 48.0), (4.0, 22.0), (10.0, 31.0), (2.0, 33.0)]

43.5263157895 31.5789473684 2 [(40.0, 21.0), (39.0, 36.0), (50.0, 20.0), (35.0, 24.0), (49.0, 41.0), (46.0, 28.0), (45.0, 26.0), (47.0, 24.0), (50.0, 21.0), (33.0, 26.0), (43.0, 42.0), (41.0, 38.0), (48.0, 39.0), (40.0, 46.0), (44.0, 38.0), (42.0, 44.0), (49.0, 30.0), (49.0, 23.0), (37.0, 33.0)]

38.9285714286 11.8571428571 3 [(48.0, 15.0), (45.0, 5.0), (36.0, 19.0), (41.0, 18.0), (32.0, 9.0), (36.0, 19.0), (43.0, 2.0), (39.0, 8.0), (35.0, 11.0), (34.0, 15.0), (37.0, 17.0), (30.0, 8.0), (47.0, 9.0), (42.0, 11.0)]

24.2142857143 38.0714285714 4 [(29.0, 32.0), (27.0, 44.0), (12.0, 38.0), (25.0, 43.0), (24.0, 25.0), (27.0, 26.0), (31.0, 45.0), (27.0, 29.0), (34.0, 47.0), (8.0, 44.0), (22.0, 34.0), (22.0, 46.0), (23.0, 50.0), (28.0, 30.0)]

17.5555555556 6.77777777778 5 [(20.0, 4.0), (13.0, 1.0), (16.0, 4.0), (17.0, 11.0), (20.0, 8.0), (25.0, 3.0), (17.0, 7.0), (14.0, 18.0), (16.0, 5.0)]

6.1935483871 8.12903225806 6 [(6.0, 12.0), (6.0, 13.0), (3.0, 19.0), (3.0, 12.0), (3.0, 9.0), (2.0, 5.0), (2.0, 2.0), (4.0, 11.0), (4.0, 3.0), (5.0, 10.0), (8.0, 6.0), (8.0, 4.0), (10.0, 7.0), (10.0, 4.0), (12.0, 3.0), (9.0, 3.0), (5.0, 2.0), (2.0, 2.0), (11.0, 4.0), (3.0, 4.0), (10.0, 5.0), (12.0, 6.0), (6.0, 8.0), (4.0, 8.0), (7.0, 10.0), (4.0, 10.0), (3.0, 16.0), (12.0, 13.0), (6.0, 18.0), (2.0, 14.0), (10.0, 9.0)]

**Execution 3:**

7.6 32.8 1 [(10.0, 40.0), (8.0, 44.0), (7.0, 35.0), (15.0, 31.0), (5.0, 29.0), (12.0, 21.0), (14.0, 28.0), (6.0, 32.0), (1.0, 37.0), (12.0, 38.0), (4.0, 22.0), (10.0, 31.0), (2.0, 33.0), (1.0, 48.0), (7.0, 23.0)]

43.7222222222 32.1666666667 2 [(35.0, 24.0), (39.0, 36.0), (50.0, 20.0), (49.0, 41.0), (46.0, 28.0), (45.0, 26.0), (43.0, 42.0), (48.0, 39.0), (40.0, 46.0), (44.0, 38.0), (49.0, 23.0), (47.0, 24.0), (50.0, 21.0), (33.0, 26.0), (49.0, 30.0), (42.0, 44.0), (37.0, 33.0), (41.0, 38.0)]

39.0 12.4666666667 3 [(36.0, 19.0), (41.0, 18.0), (45.0, 5.0), (36.0, 19.0), (48.0, 15.0), (32.0, 9.0), (43.0, 2.0), (39.0, 8.0), (35.0, 11.0), (34.0, 15.0), (37.0, 17.0), (30.0, 8.0), (47.0, 9.0), (42.0, 11.0), (40.0, 21.0)]

26.5833333333 37.5833333333 4 [(25.0, 43.0), (27.0, 44.0), (29.0, 32.0), (34.0, 47.0), (22.0, 34.0), (28.0, 30.0), (27.0, 26.0), (31.0, 45.0), (23.0, 50.0), (24.0, 25.0), (27.0, 29.0), (22.0, 46.0)]

16.5454545455 6.36363636364 5 [(25.0, 3.0), (20.0, 4.0), (17.0, 11.0), (16.0, 4.0), (13.0, 1.0), (20.0, 8.0), (17.0, 7.0), (14.0, 18.0), (12.0, 6.0), (12.0, 3.0), (16.0, 5.0)]

5.79310344828 8.37931034483 6 [(8.0, 6.0), (6.0, 13.0), (3.0, 19.0), (3.0, 12.0), (3.0, 9.0), (2.0, 5.0), (2.0, 2.0), (4.0, 11.0), (4.0, 3.0), (5.0, 10.0), (6.0, 12.0), (8.0, 4.0), (10.0, 7.0), (10.0, 4.0), (9.0, 3.0), (5.0, 2.0), (2.0, 2.0), (11.0, 4.0), (3.0, 4.0), (10.0, 5.0), (6.0, 8.0), (4.0, 8.0), (7.0, 10.0), (4.0, 10.0), (3.0, 16.0), (2.0, 14.0), (10.0, 9.0), (6.0, 18.0), (12.0, 13.0)]

**Execution 4:**

7.6 32.8 1 [(10.0, 40.0), (8.0, 44.0), (7.0, 35.0), (15.0, 31.0), (5.0, 29.0), (12.0, 21.0), (14.0, 28.0), (6.0, 32.0), (1.0, 37.0), (12.0, 38.0), (4.0, 22.0), (10.0, 31.0), (2.0, 33.0), (1.0, 48.0), (7.0, 23.0)]

43.3529411765 32.8823529412 2 [(45.0, 26.0), (35.0, 24.0), (39.0, 36.0), (49.0, 41.0), (46.0, 28.0), (43.0, 42.0), (48.0, 39.0), (40.0, 46.0), (44.0, 38.0), (49.0, 23.0), (47.0, 24.0), (50.0, 21.0), (33.0, 26.0), (49.0, 30.0), (42.0, 44.0), (37.0, 33.0), (41.0, 38.0)]

39.6875 12.9375 3 [(36.0, 19.0), (41.0, 18.0), (45.0, 5.0), (36.0, 19.0), (48.0, 15.0), (32.0, 9.0), (43.0, 2.0), (39.0, 8.0), (35.0, 11.0), (34.0, 15.0), (37.0, 17.0), (30.0, 8.0), (47.0, 9.0), (42.0, 11.0), (40.0, 21.0), (50.0, 20.0)]

26.5833333333 37.5833333333 4 [(27.0, 44.0), (29.0, 32.0), (25.0, 43.0), (34.0, 47.0), (22.0, 34.0), (28.0, 30.0), (27.0, 26.0), (31.0, 45.0), (23.0, 50.0), (24.0, 25.0), (27.0, 29.0), (22.0, 46.0)]

16.0833333333 6.16666666667 5 [(25.0, 3.0), (20.0, 4.0), (17.0, 11.0), (16.0, 4.0), (13.0, 1.0), (20.0, 8.0), (12.0, 6.0), (14.0, 18.0), (12.0, 3.0), (16.0, 5.0), (11.0, 4.0), (17.0, 7.0)]

5.60714285714 8.53571428571 6 [(8.0, 6.0), (6.0, 13.0), (3.0, 19.0), (3.0, 12.0), (3.0, 9.0), (2.0, 5.0), (2.0, 2.0), (4.0, 11.0), (4.0, 3.0), (5.0, 10.0), (6.0, 12.0), (8.0, 4.0), (10.0, 7.0), (10.0, 4.0), (9.0, 3.0), (5.0, 2.0), (2.0, 2.0), (3.0, 4.0), (10.0, 5.0), (6.0, 8.0), (4.0, 8.0), (7.0, 10.0), (4.0, 10.0), (3.0, 16.0), (2.0, 14.0), (10.0, 9.0), (6.0, 18.0), (12.0, 13.0)]