**CSC 555: Mining Big Data**

Project, Phase 2

In this part of the project, you will various queries using Hive, Pig and Hadoop streaming. The schema is available below, but do remember that schema should specify the delimiter:

http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM\_schema\_hive.sql

The data is available at (note that the data is |-separated, not comma separated):

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC553/data/> (this is Scale4)

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC553/data/Scale14/> (This is Scale14, larger version)

Please note what instance and what cluster you are using (you can reuse your existing cluster for most of the questions).

Please be sure to submit all code (pig and 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). You can use time command to record time of execution of anything you run.

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 use head -n 100 to get first 100 lines only).

# Part 1: Data Transformation

Using Scale4 data perform the following data processing.

1. Transform lineorder.tbl table into a csv (comma-separated file): Use Hive, MapReduce with HadoopStreaming and Pig (i.e. 3 different solutions)

Hive Steps (Solution 1):

1.

wget <http://rasinsrv07.cstcis.cti.depaul.edu/CSC553/data/lineorder.tbl>

2.

head –n100 lineorder.tbl > lineorder.tbl.sample (For code testing)

3.

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 '|';

4.

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

5.

ADD FILE /home/ec2-user/project\_final\_1a.py;

Python Code

#!/usr/bin/env python

import sys

#Reading from terminal

for line in sys.stdin:

words = line.strip()

vals = words.split('|')

#Result output to table

print ','.join(vals)

6.

create table lineorder2 (

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 ',';

7.

INSERT OVERWRITE TABLE lineorder2

SELECT TRANSFORM (lo\_orderkey,lo\_linenumber,lo\_custkey,lo\_partkey,lo\_suppkey,lo\_orderdate,lo\_orderpriority,lo\_shippriority,lo\_quantity,lo\_extendedprice,lo\_ordertotalprice,lo\_discount,lo\_revenue,lo\_supplycost,lo\_tax,lo\_commitdate,lo\_shipmode)

USING 'python project\_final\_1a.py'

AS (lo\_orderkey,lo\_linenumber,lo\_custkey,lo\_partkey,lo\_suppkey,lo\_orderdate,lo\_orderpriority,lo\_shippriority,lo\_quantity,lo\_extendedprice,lo\_ordertotalprice,lo\_discount,lo\_revenue,lo\_supplycost,lo\_tax,lo\_commitdate,lo\_shipmode)

from lineorder;

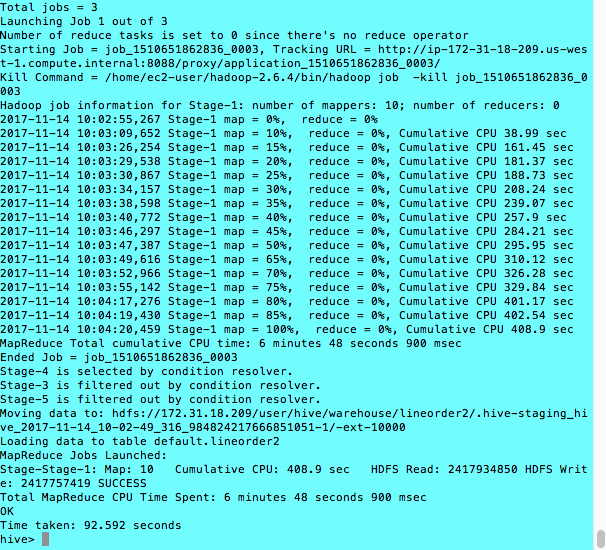
8.

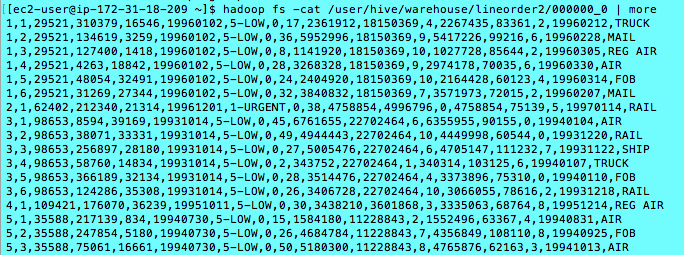
hadoop fs -get /user/hive/warehouse/lineorder2

9 (Execute in the lineorder2 directory)

cat 000000\_0 000001\_0 000002\_0 000003\_0 000004\_0 000005\_0 000006\_0 000007\_0 000008\_0 000009\_0 >> lineorder2.csv

Hive Output: (4- Node)





Hadoop Streaming Steps (Solution 2):

1.

hadoop fs -put linorder.tbl

2.

time hadoop jar /home/ec2-user/hadoop-2.6.4/share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator -input lineorder.tbl.1 -output /final1a\_hs/ -mapper project\_final\_1a.py -file project\_final\_1a.py

Python Code (project\_final\_1a.py) store in local drive:

#!/usr/bin/env python

import sys

#Reading from terminal

for line in sys.stdin:

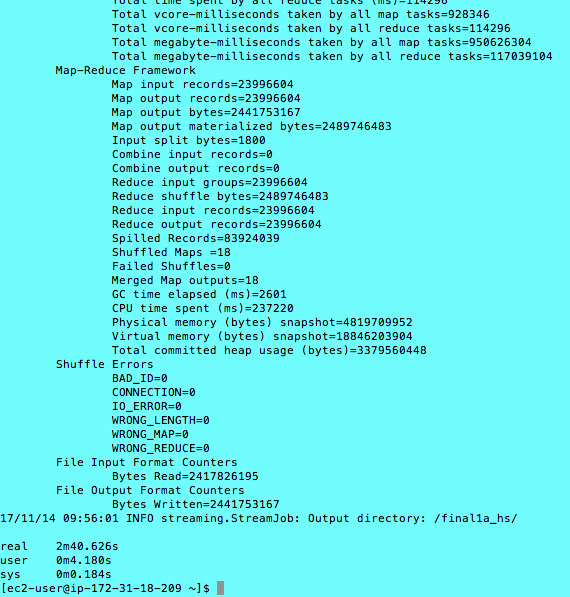
words = line.strip()

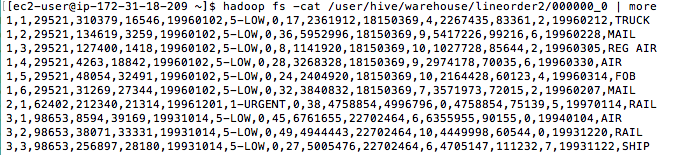
vals = words.split('|')

#Result output to table

print ','.join(vals)

Hadoop Streaming Output: (4- Node)





Pig Steps (Solution 3):

Steps:

1.

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

AS (lo\_orderkey :float,

lo\_linenumber :float,

lo\_custkey :float,

lo\_partkey :float,

lo\_suppkey :float,

lo\_orderdate :float,

lo\_orderpriority :chararray,

lo\_shippriority : chararray,

lo\_quantity : chararray,

lo\_extendedprice :float,

lo\_ordertotalprice :float,

lo\_discount :float,

lo\_revenue :float,

lo\_supplycost :float,

lo\_tax :float,

lo\_commitdate :float,

lo\_shipmode :chararray);

2.

DESCRIBE lod

3.

lod2 = FOREACH lod GENERATE lo\_orderkey,lo\_linenumber,lo\_custkey,lo\_partkey,lo\_suppkey,lo\_orderdate,lo\_orderpriority,lo\_shippriority,lo\_quantity,lo\_extendedprice,lo\_ordertotalprice,lo\_discount,lo\_revenue,lo\_supplycost,lo\_tax,lo\_commitdate,lo\_shipmode;

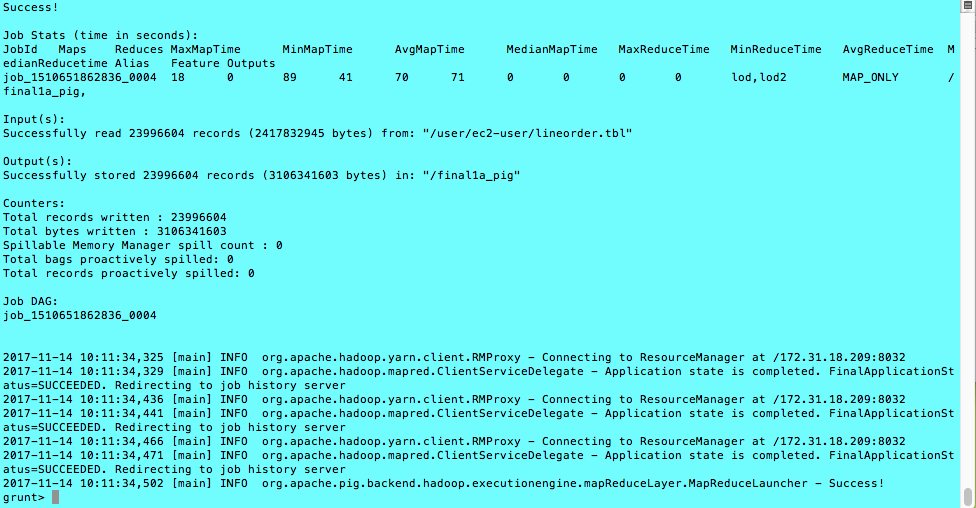
4.

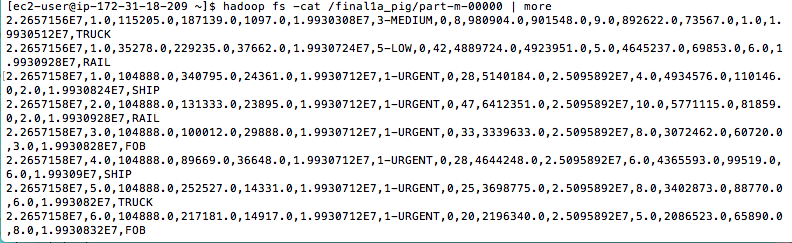
dump lod2;

5.

store lod2 into '/final1a\_pig' USING PigStorage (',');

Pig Output:





1. Extract five of the numeric columns that for rows where lo\_discount is between 4 and 6 into a space-separated text file (for K-Means clustering later). Use Hive and Pig (2 different solutions)

(NOTE: you do not need to use your code to identify what is a numeric column, just go by what the data types say. You should manually pick any 5 columns that contain only numbers)

Hive(Solution 1): (4-nodes)

1.

INSERT OVERWRITE DIRECTORY '/fiveCol1b\_hive'

row format delimited FIELDS TERMINATED BY ' '

select ((lo\_discount - min\_dis) / lo\_dis\_range) as dis,

((lo\_quantity - min\_qua) / lo\_qua\_range) as qua,

((lo\_extendedprice - min\_exprice) / lo\_exprice\_range) as exp,

((lo\_ordertotalprice - min\_otp) / lo\_otp\_range) as otp,

((lo\_revenue - min\_rev) / lo\_rev\_range) as rev

from (select lo\_discount ,

MIN(lo\_discount) over () as min\_dis,

(MAX(lo\_discount) over () - MIN(lo\_discount) over () ) as lo\_dis\_range,

lo\_quantity,

MIN(lo\_quantity) over () as min\_qua,

(MAX(lo\_quantity) over () - MIN(lo\_quantity) over () ) as lo\_qua\_range,

lo\_extendedprice,

MIN(lo\_extendedprice) over () as min\_exprice,

(MAX(lo\_extendedprice) over () - MIN(lo\_extendedprice) over () ) as lo\_exprice\_range,

lo\_ordertotalprice,

MIN(lo\_ordertotalprice) over () as min\_otp,

(MAX(lo\_ordertotalprice) over () - MIN(lo\_ordertotalprice) over () ) as lo\_otp\_range,

lo\_revenue,

MIN(lo\_revenue) over () as min\_rev,

(MAX(lo\_revenue) over () - MIN(lo\_revenue) over () ) as lo\_rev\_range

from lineorder

where lo\_discount between 4.0 and 6.0 ) x;

2.

hadoop fs -copyToLocal /fiveCol1b\_hive

3.

cd fiveCol1b\_hive

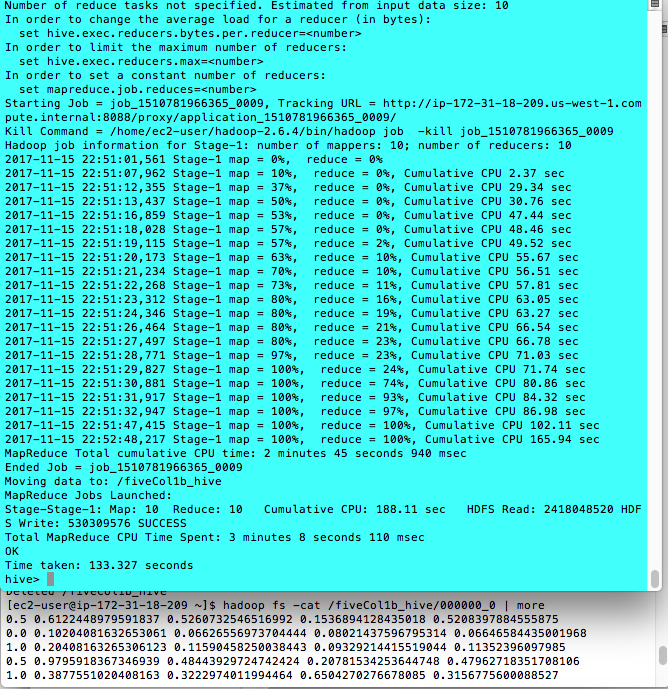
4.

cat 000000\_0 000001\_0 000002\_0 000003\_0 000004\_0 000005\_0 000006\_0 000007\_0 000008\_0 000009\_0 >> fiveCol1bhive\_one

5.

hadoop fs -put fiveCol1bhive\_one

Hive Output:



Pig(Solution 2): Note: in step 6, the function “PigStorage” must be input as this format, Can’t be edited in upper or lower case.

1.

Parse the followings on the script and then execute in the PIG directory

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

AS (lo\_orderkey :float,

lo\_linenumber :float,

lo\_custkey :float,

lo\_partkey :float,

lo\_suppkey :float,

lo\_orderdate :float,

lo\_orderpriority :chararray,

lo\_shippriority : chararray,

lo\_quantity : chararray,

lo\_extendedprice :float,

lo\_ordertotalprice :float,

lo\_discount :float,

lo\_revenue :float,

lo\_supplycost :float,

lo\_tax :float,

lo\_commitdate :float,

lo\_shipmode : chararray);

2.

describe lod

3.

lodDis = filter lod by ((lo\_discount>=4.0) and (lo\_discount<=6.0));

4.

fiveCol = foreach lodDis generate lo\_discount, lo\_quantity, lo\_extendedprice, lo\_ordertotalprice, lo\_revenue;

5

dump fiveCol

6.

store fiveCol into '/fiveCol1b\_pig' using PigStorage (' ');

7.

Hadoop fs -copyToLocal '/fiveCol1b\_pig'

8.

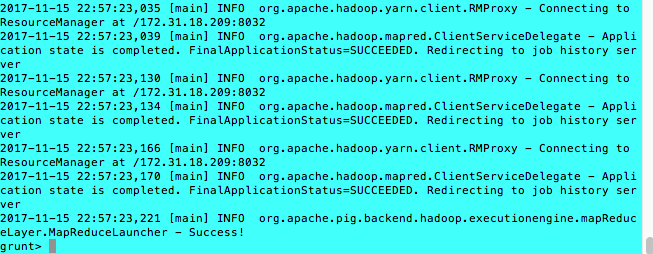
cat part-m-00000 part-m-00000 part-m-00001 part-m-00002 part-m-00003 part-m-00004 part-m-00005 part-m-00006 part-m-00007 part-m-00008 part-m-00009 part-m-00010 part-m-00011 part-m-00012 part-m-00013 part-m-00014 part-m-00015 part-m-00016 part-m-00017 >> fiveCol1bpig\_one

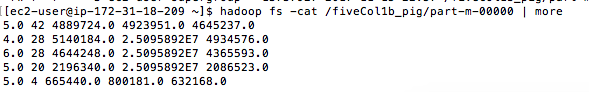
9. (Prepare for Question 3)

hadoop fs -put fiveCol1bpig\_one

Pig output:







1. Create a pre-join (i.e. a new data file) that corresponds to the following query below. You can think of it as a materialized view. What is the size of the new file? Use Hive and Pig (2 different solutions).

SELECT lo\_partkey, lo\_suppkey, lo\_discount, d\_year, lo\_revenue

FROM lineorder, dwdate

WHERE lo\_orderdate = d\_datekey;

Ans:

Hive (Solution 1):

wget http://rasinsrv07.cstcis.cti.depaul.edu/CSC553/data/dwdate.tbl

1.

create table dwdate (

d\_datekey int,

d\_date varchar(19),

d\_dayofweek varchar(10),

d\_month varchar(10),

d\_year int,

d\_yearmonthnum int,

d\_yearmonth varchar(8),

d\_daynuminweek int,

d\_daynuminmonth int,

d\_daynuminyear int,

d\_monthnuminyear int,

d\_weeknuminyear int,

d\_sellingseason varchar(13),

d\_lastdayinweekfl varchar(1),

d\_lastdayinmonthfl varchar(1),

d\_holidayfl varchar(1),

d\_weekdayfl varchar(1)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';

2.

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

3.

INSERT OVERWRITE DIRECTORY 'preJoin\_hive\_1c'

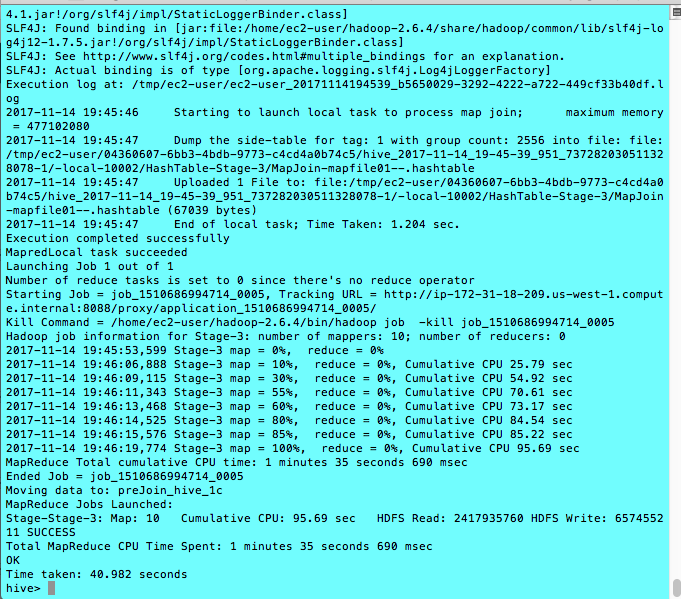
row format delimited FIELDS TERMINATED BY ','

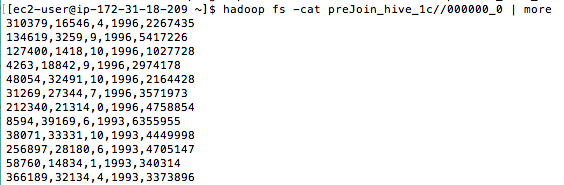
SELECT lo\_partkey, lo\_suppkey, lo\_discount, d\_year, lo\_revenue

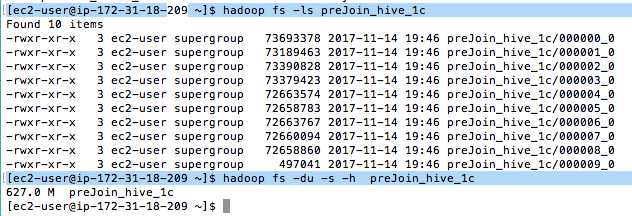
FROM lineorder, dwdate

WHERE lo\_orderdate = d\_datekey;

Hive Output:







The size of file(s) is 627M

Pig (Solution 2):

1.

Hadoop fs -put dwdate.tbl;

Hadoop fs -put lineorder.tbl;

2.

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

AS (lo\_orderkey :float,

lo\_linenumber :float,

lo\_custkey :float,

lo\_partkey :float,

lo\_suppkey :float,

lo\_orderdate :float,

lo\_orderpriority :chararray,

lo\_shippriority : chararray,

lo\_quantity : chararray,

lo\_extendedprice :float,

lo\_ordertotalprice :float,

lo\_discount :float,

lo\_revenue :float,

lo\_supplycost :float,

lo\_tax :float,

lo\_commitdate :float,

lo\_shipmode : chararray);

3.

dwd= LOAD '/user/ec2-user/dwdate.tbl' USING PigStorage('|')

AS(

d\_datekey :float,

d\_date :chararray,

d\_dayofweek :chararray,

d\_month :chararray,

d\_year :float,

d\_yearmonthnum :float,

d\_yearmonth :chararray,

d\_daynuminweek :float,

d\_daynuminmonth :float,

d\_daynuminyear :float,

d\_monthnuminyear :float,

d\_weeknuminyear :float,

d\_sellingseason :chararray,

d\_lastdayinweekfl :chararray,

d\_lastdayinmonthfl :chararray,

d\_holidayfl :chararray,

d\_weekdayfl :chararray);

4.

joinDate = JOIN lod BY lo\_orderdate, dwd BY d\_datekey;

dump joinDate;

5.

result = foreach joinDate generate lo\_partkey , lo\_suppkey , lo\_discount , d\_year , lo\_revenue;

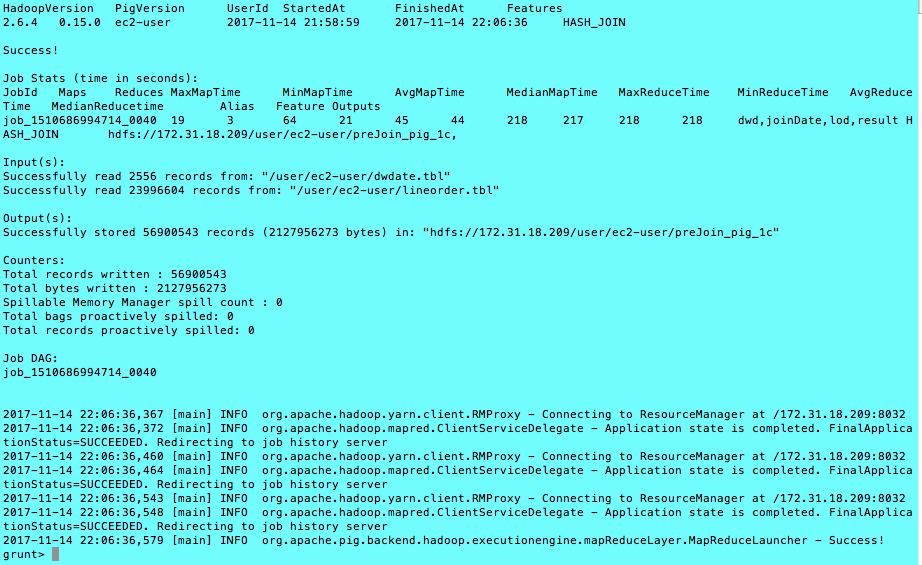
6.

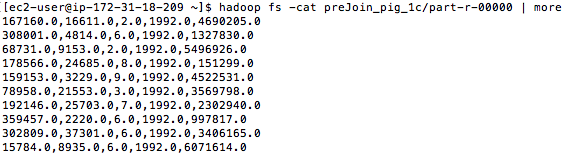
dump result;

7.

store result into 'preJoin\_pig\_1c' using PigStorage (',');

Pig Output







file size is 2.0G

Note

Based on the above reslt, the PIG output files are larger than the HIVE files. I would think that the reason is PIG Tables are created with ‘float’ numeric values , but HIVE table was created with ‘Integer’ Values.

# Part 2: Querying

All queries from SSBM benchmark are available here:

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

Using Scale4 data perform the following data processing and don’t forget to time your results.

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 '|';

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

 create table supplier (

s\_suppkey int,

s\_name varchar(25),

s\_address varchar(25),

s\_city varchar(10),

s\_nation varchar(15),

s\_region varchar(12),

s\_phone varchar(15)

) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';

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

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 '|';

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

1. Run SSBM queries 2.2, 3.2 and 4.2 using Hive only.

Ans:

2.2

select sum(lo\_revenue), d\_year, p\_brand1

from lineorder, dwdate, part, supplier

where lo\_orderdate = d\_datekey

and lo\_partkey = p\_partkey

and lo\_suppkey = s\_suppkey

and p\_brand1 between 'MFGR#2221'

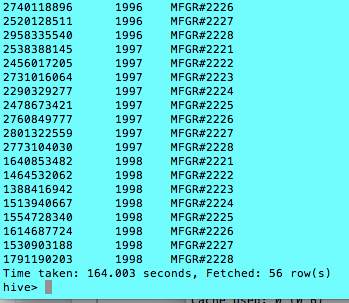
and 'MFGR#2228'

and s\_region = 'ASIA'

group by d\_year, p\_brand1

order by d\_year, p\_brand1;

Output: (4-nodes)



Time taken: 164.003 sec with 56 rows result.

3.2

select c\_city, s\_city, d\_year, sum(lo\_revenue) as revenue

from customer, lineorder, supplier, dwdate

where lo\_custkey = c\_custkey

and lo\_suppkey = s\_suppkey

and lo\_orderdate = d\_datekey

and c\_nation = 'UNITED STATES'

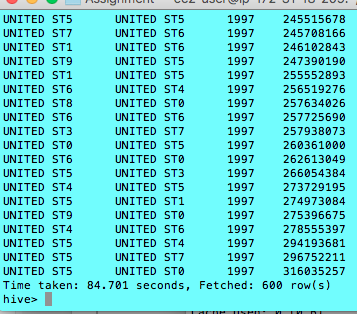
and s\_nation = 'UNITED STATES'

and d\_year between 1992 and 1997

group by c\_city, s\_city, d\_year

order by d\_year asc, revenue asc;

Output: (4-nodes)



Time taken is 84.701 sec with 600 rows result.

4.2

--Q4.2 Removed second match of OR conditions, expression in sum

select d\_year, s\_nation, p\_category, sum(lo\_revenue) as profit1

from lineorder , customer , supplier , part, dwdate

where lo\_custkey = c\_custkey

and lo\_suppkey = s\_suppkey

and lo\_partkey = p\_partkey

and lo\_orderdate = d\_datekey

and c\_region = 'AMERICA'

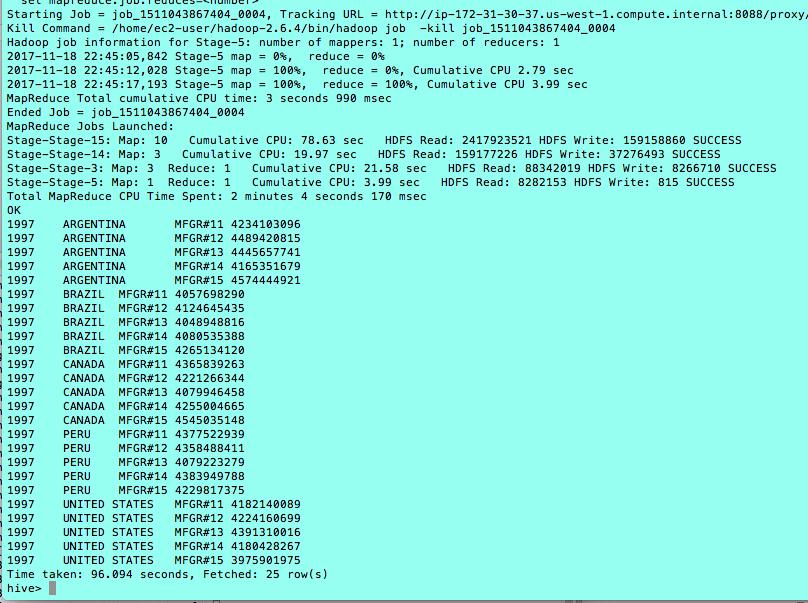
and s\_region = 'AMERICA'

and d\_year = 1997

and p\_mfgr = 'MFGR#1'

group by d\_year, s\_nation, p\_category;

Ans:



In this particular question, I have experienced that the sequence of calling the table in the query is really matter affecting the speed of the process. The sequence of calling the tables should be in the SAME order according to the conditions (“where” statement) statements.

1. For this part use Hive and Pig (two different solutions) to run Q2.1 using what you have created in 1-C (i.e. use PreJoin1 instead of lineorder and dwdate tables in the from clause). You would need to rewrite the query accordingly. (e.g. something like,

select sum(lo\_revenue), d\_year, p\_brand1

from MyNewStructureFrom1C, part, supplier

where lo\_partkey = p\_partkey

and lo\_suppkey = s\_suppkey

and p\_category = 'MFGR#12'

and s\_region = 'AMERICA'

group by d\_year, p\_brand1

order by d\_year, p\_brand1;**)**

HIVE: 4 –Nodes

1. From part 1c:

INSERT OVERWRITE DIRECTORY 'preJoin\_hive\_1c'

row format delimited FIELDS TERMINATED BY ','

SELECT lo\_partkey, lo\_suppkey, lo\_discount, d\_year, lo\_revenue

FROM lineorder, dwdate

WHERE lo\_orderdate = d\_datekey;

2.

Create Table preJoin1c(

lo\_partkey int,

lo\_suppkey int,

lo\_discount int,

d\_year int,

lo\_revenue int

) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

1. Copy the hdfs file to the local file system

hadoop fs -copyToLocal preJoin\_hive\_1c/000000\_0 > preJoin\_hive\_0

hadoop fs -copyToLocal preJoin\_hive\_1c/000001\_0 > preJoin\_hive\_1

hadoop fs -copyToLocal preJoin\_hive\_1c/000002\_0 > preJoin\_hive\_2

hadoop fs -copyToLocal preJoin\_hive\_1c/000003\_0 > preJoin\_hive\_3

hadoop fs -copyToLocal preJoin\_hive\_1c/000004\_0 > preJoin\_hive\_4

hadoop fs -copyToLocal preJoin\_hive\_1c/000005\_0 > preJoin\_hive\_5

hadoop fs -copyToLocal preJoin\_hive\_1c/000006\_0 > preJoin\_hive\_6

hadoop fs -copyToLocal preJoin\_hive\_1c/000007\_0 > preJoin\_hive\_7

hadoop fs -copyToLocal preJoin\_hive\_1c/000008\_0 > preJoin\_hive\_8

hadoop fs -copyToLocal preJoin\_hive\_1c/000009\_0 > preJoin\_hive\_9

1. Copy the file content to the local file

cp 000000\_0 preJoin\_hive\_0

cp 000000\_1 preJoin\_hive\_1

cp 000000\_2 preJoin\_hive\_2

cp 000000\_3 preJoin\_hive\_3

cp 000000\_4 preJoin\_hive\_4

cp 000000\_5 preJoin\_hive\_5

cp 000000\_6 preJoin\_hive\_6

cp 000000\_7 preJoin\_hive\_7

cp 000000\_8 preJoin\_hive\_8

cp 000000\_9 preJoin\_hive\_9

5. Insert the local data to the preJoin1c table

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_0' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_1' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_2' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_3' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_4' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_5' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_6' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_7' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_8' INTO TABLE preJoin1c;

LOAD DATA LOCAL INPATH '/home/ec2-user/preJoin\_hive\_9' INTO TABLE preJoin1c;

6.

select sum(lo\_revenue), d\_year, p\_brand1

from preJoin1c, part, supplier

where lo\_partkey = p\_partkey

and lo\_suppkey = s\_suppkey

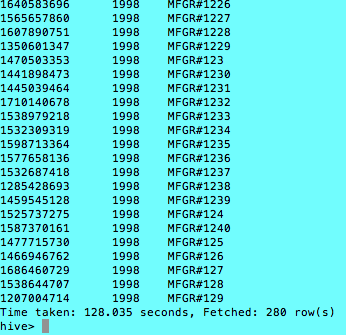
and p\_category = 'MFGR#12'

and s\_region = 'AMERICA'

group by d\_year, p\_brand1

order by d\_year, p\_brand1;

Hive Output: (4-node)

****

PIG: At least 4 –Nodes

1

hadoop fs -put part.tbl;

hadoop fs -put supplier.tbl;

2

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

AS (p\_partkey :float,

p\_name :chararray,

p\_mfgr :chararray,

p\_category :chararray,

p\_brand1 :chararray,

p\_color :chararray,

p\_type :chararray,

p\_size :float,

p\_container :chararray);

3

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

AS ( s\_suppkey :float,

s\_name :chararray,

s\_address :chararray,

s\_city :chararray,

s\_nation :chararray,

s\_region :chararray,

s\_phone :chararray);

4.

preJoin = LOAD 'preJoin\_pig\_1c' USING PigStorage(',')

AS (lo\_partkey :float

, lo\_suppkey :float

, lo\_discount :float

, d\_year :chararray

, lo\_revenue :float);

5.

pre\_supplier\_join= JOIN preJoin BY lo\_suppkey , supplier BY s\_suppkey;

6.

pre\_supplier\_part\_join = JOIN pre\_supplier\_join BY lo\_partkey, part BY p\_partkey;

7.

pre\_supplier\_part\_join\_filter = FILTER pre\_supplier\_part\_join BY ( p\_category == 'MFGR#12' ) and ( s\_region == 'AMERICA');

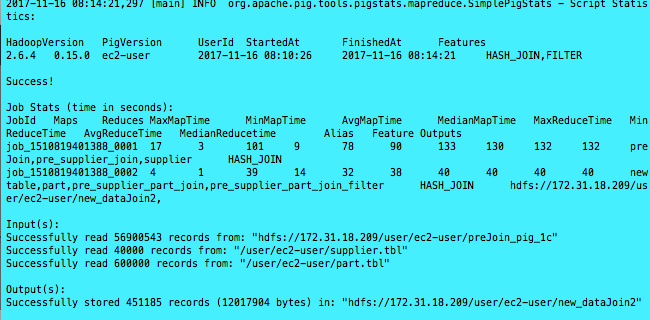
8.

newtable = FOREACH pre\_supplier\_part\_join\_filter generate lo\_revenue, d\_year, p\_brand1;

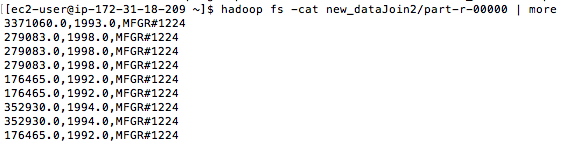
9.

STORE newtable INTO 'new\_dataJoin2' using PigStorage(',');

First Phase:



Time taken for phase 1 is about 03min 55sec



10.

newjoin2 = LOAD 'new\_dataJoin2' USING PigStorage (',')

AS (lo\_revenue :float,

d\_year :chararray ,

p\_brand1 :chararray);

11.

join\_group = GROUP newjoin2 BY (d\_year, p\_brand1);

12.

result = FOREACH join\_group GENERATE group, SUM(newjoin2.lo\_revenue);

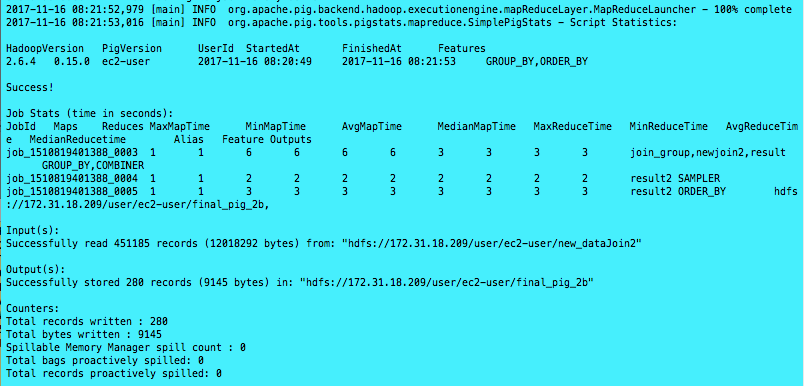
13.

result2 = ORDER result BY group;

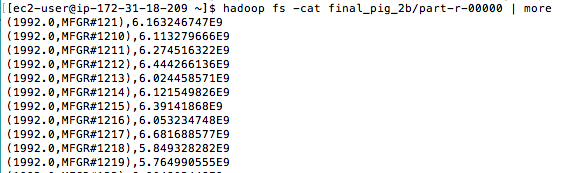
14.

STORE result2 into 'final\_pig\_2b' using PigStorage(',');

Second Phase



Time taken for 2nd phase is about 01min 04sec



The total time taken is about 04 min 59 sec

# Part 3: Clustering

Using the file you have created in 1-B, run KMeans clustering using 7 clusters.

1. Using Mahout synthetic clustering as you have in a previous assignment on sample data.

Command line input: the input file ‘fiveCol1bhive\_one’ was nomarlized into (0,1) sclae in part 2b

time mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job --maxIter 10 --numClusters 7 --t1 0.3 --t2 0.5 --input fiveCol1bhive\_one --output cluster\_output

Output:



1. Using Hadoop streaming perform one iteration manually with randomly chosen input 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 need to compute the new centers, and at that point the iteration is done.

Command line

time hadoop jar /home/ec2-user/hadoop-2.6.4/share/hadoop/tools/lib/hadoop-streaming-2.6.4.jar -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator -D mapred.text.key.comparator.options=-n -input fiveCol1bhive\_one -output 3b\_kmean\_output -mapper 3bmapper.py -reducer 3breducer.py -file 3breducer.py -file 3bmapper.py -file 3bcenters.txt

Mapper Code:

import math

import sys

center\_lst=[]

center\_dict={}

#Manually assign the first 7 clusters center

with open('3bcenters.txt','r') as ofile:

    lines=ofile.readlines()

    for line in lines:

        words=line.strip().split(' ')

        center\_lst.append(map(float,words))

#Read from the stdin

for record in sys.stdin:

    instance=record.strip().split(' ')

    cal\_lst=[]

    for c in center\_lst:

        num=0

        for i in range(0,len(c)):

            #print(c[i])

            a=float(c[i])

            b=float(instance[i])

            ab=a-b

            #Calculate the Euclidean Distance

            num+= math.pow(ab,2)

            #print(num)

        cal\_lst.append(round(math.sqrt(num),3))

    #Assign the key (Index of cluster,0-6) to the instance based on the shortest distance

    a=cal\_lst.index(min(cal\_lst))

    if a in center\_dict.keys():

        center\_dict[a].append(instance)

    else:

        center\_dict[a]=[instance]

#Print out ALL the instance with key

for i in range(len(center\_lst)):

    if i in center\_dict.keys():

        for val in center\_dict[i]:

            print '%d% 0.5f% 0.5f% 0.5f% 0.5f% 0.5f'% (i,float(val[0]),float(val[1]),float(val[2]),float(val[3]),float(val[4]))

    else:

        print '%d% 0.5f% 0.5f% 0.5f% 0.5f% 0.5f'%(i,center\_lst[i][0],center\_lst[i][1],center\_lst[i][2],center\_lst[i][3],center\_lst[i][4])

Reducer Code:

import math

import sys

#Initialize variables

new\_center=[]

c=[0,0,0,0,0]

key=-1

counter=0.0

for line in sys.stdin:

    words = line.strip().split(' ')

    if key == words[0]: #Collect all the points belong to the same center

        for i in range(1,len(words)):

            c[i-1] += float(words[i]) #Add all the columns values among the SAME cluster members

        counter += 1.0

    else: # Start here when countering the new clusters instance

        if counter != 0.0:

            aa=[0.0,0.0,0.0,0.0,0.0]

            aa = [c[i]/counter for i in range(len(c))]

            new\_center.append(aa) #Store all the clusters centers in the list

        c=[0,0,0,0,0] #reinitialize

        counter=0

        key=words[0]

        for i in range(1,len(words)):

            c[i-1] += float(words[i]) #Add all the columns values among the SAME cluster members

        counter+=1.0

aa = [c[i]/counter for i in range(len(c))]

new\_center.append(aa)

#The first column is the number of cluster

#The rest of columns are the centers

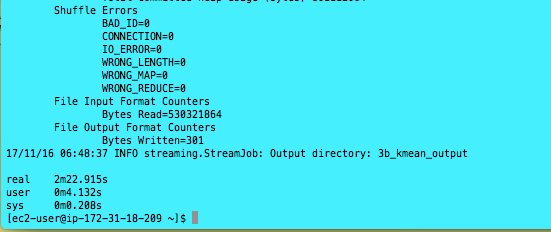
for i,val in enumerate(new\_center):

    print '%d% 0.5f% 0.5f% 0.5f% 0.5f% 0.5f'% (i,float(val[0]),float(val[1]),float(val[2]),float(val[3]),float(val[4]))

**Kmean Cluster output**

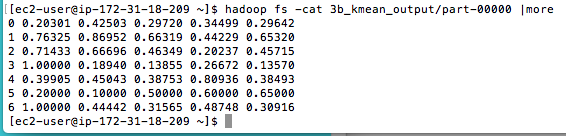
****

**Kmean cluster output**

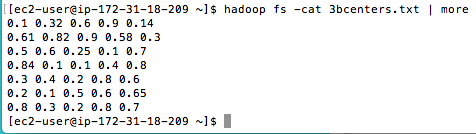
****

**Total time taken 2min 22.915sec**

**The New cluster center, the first column is the index of the 7 clusters**

****

**The initial assigned cluster center (Manually assigned in Center.txt). The dataset was normalized in (0,1) scale in Part 1b**

****

**NOTE:** if you get a java.lang.OutOfMemoryError error, you will need to reconfigure Hadoop to supply the java virtual machine with more memory. You can do this by editing the mapred-site.xml (Mapper should not need much RAM):

*<property>*

*<name>* *mapreduce.reduce.java.opts</name>*

*<value>-Xmx1024m</value>*

*</property>*

The amount of memory can be tweaked (you can go higher, but keep in mind how much physical memory your machine has). Do not forget to restart Hadoop after any configuration file change.

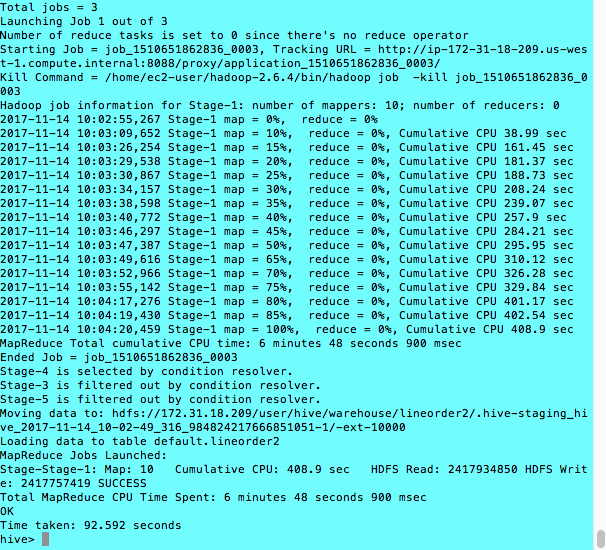
If you **still** run out of memory in 3-A submit the screenshot of that and you will get full credit for the question.

# Part 4: Performance

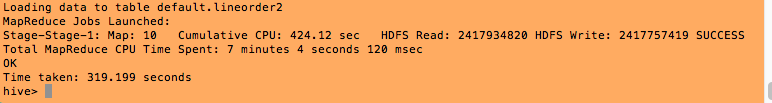
Compare the performance given following combinations. If you already ran that combination before it is sufficient to copy the runtime for comparison.

1. All three of your solutions to Part-1A with
   1. Scale4: single node and a cluster of at least 4 nodes

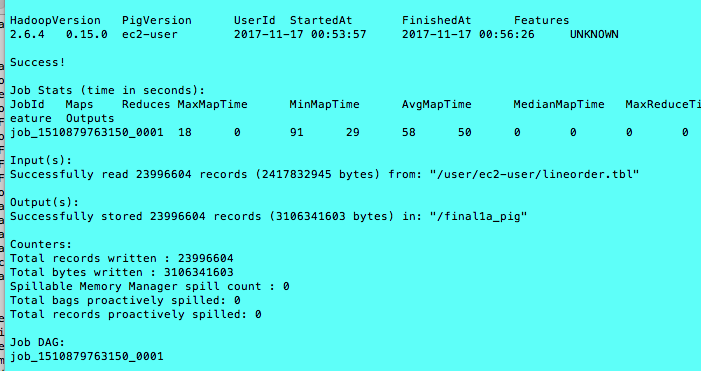
Ans: Hive – 4- node



Ans: Hive – Single node

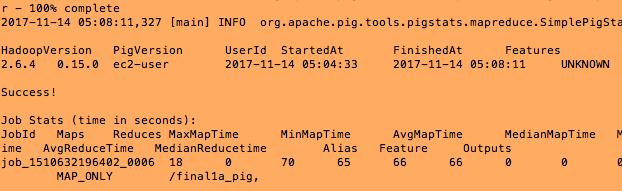


Ans: Pig – 4-node



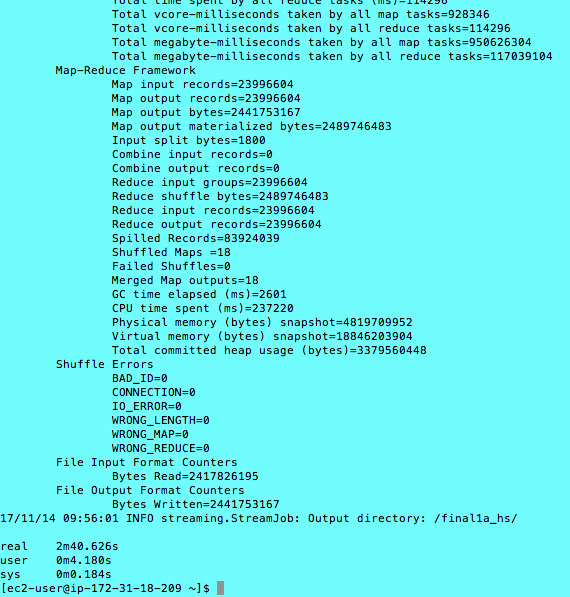
Time taken is 2min 29sec

Ans: Pig – Single node

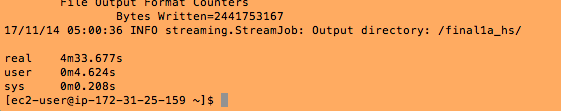


time taken : 03min 38sec

Ans: Hadoop Streaming – 4- node



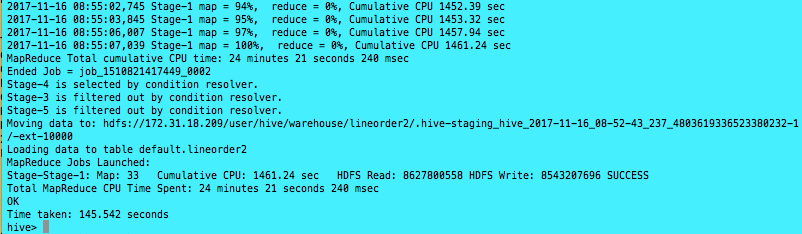
Ans: Hadoop Streaming – Single- node



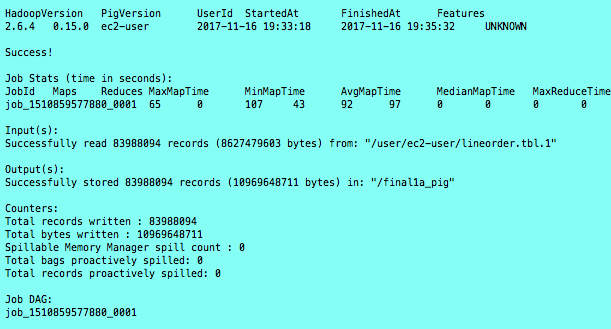
* 1. Scale14: a cluster of at least 4 nodes (Steps and coding are EXACTLY the same as Question #1)

Hive Output:



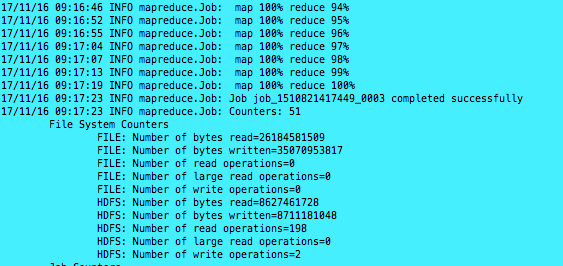


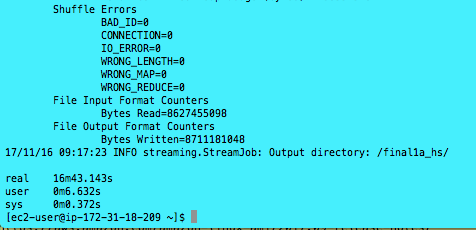
Pig Output:

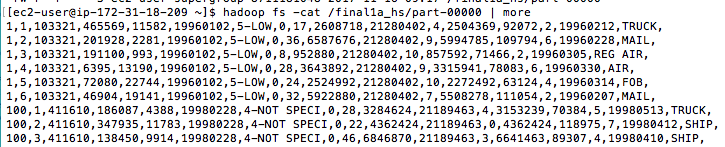


time taken is 02min 14sec

Hadoop streaming Output:



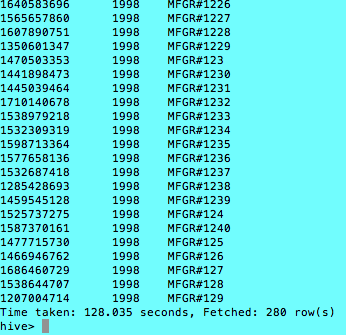




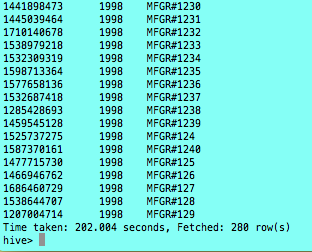
B. Both of your solution for your 2-B

Scale 4: Single and a cluster of at least 4nodes

Hive (At least 4-cluster)

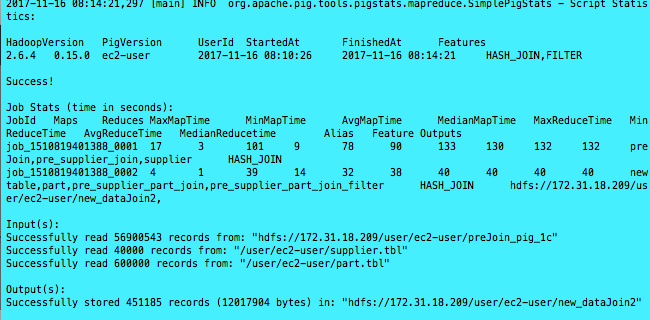


Hive : Single node:



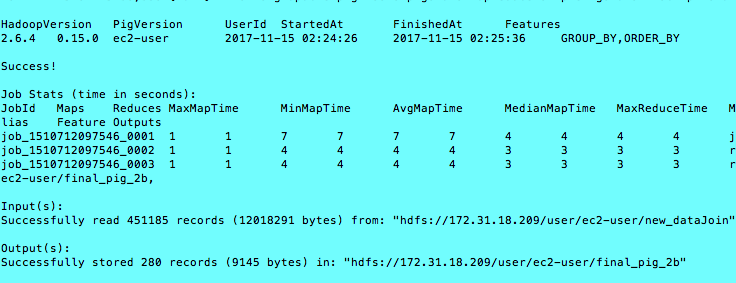
Pig: At Least 4 clusters:

First Phase:



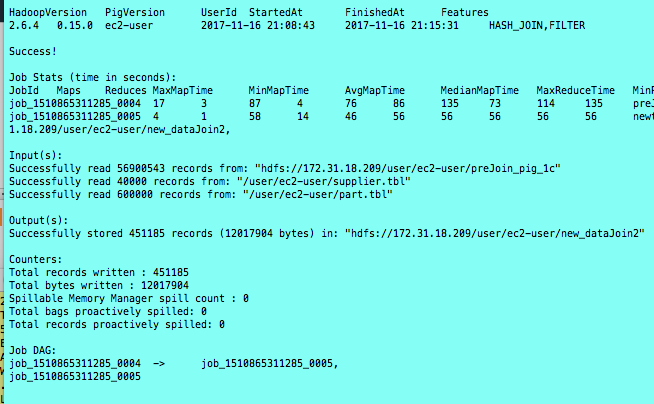
Time taken for phase 1 is about 03min 55sec

Second Phase:

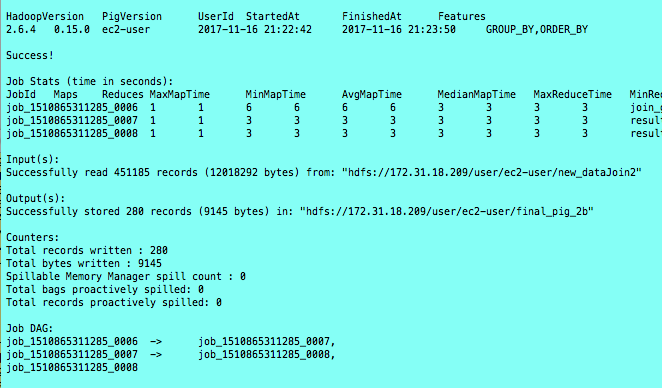


Time taken for phase 2 is about 01min 10sec

Pig -Single Cluster



Time taken for phase 1 is 06min 48sec



Time taken for phase 2 is 01min 08sec

Summarize the results and cluster performance/scaling in at least a paragraph.

Based on the above result, the performance of running task in different clusters set up achieve different performance. The multiple cluster setup apparently offers a better performance than the single node cluster in terms of process speed. In general speaking, the result above shows the time taken from the multi-nodes is about approximately 2-3 times faster than the Single node cluster. This result is very reasonable because the more worker clusters would share more works. However, I would say that the running speed also depends on the storage size and available space in the master and worker cluster. The limited size of storage size would hinder the cluster running performance.

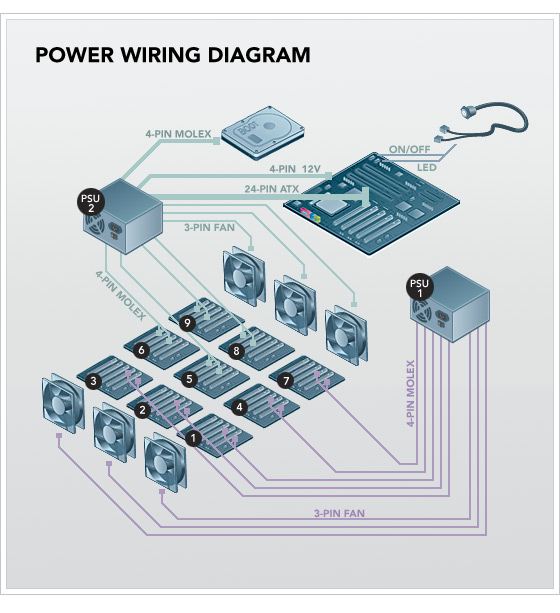
The size of dataset is also the matter to the cluster performance. Although the Scale14 dataset is about 4 times larger than the Scale4 dataset, the time consuming of Scale14 data is not 4 times slower than the Scale 4 dataset as shown above. As I said, the process speed also depends on the number of worker nodes, storage size and memory ram etc.

# Extra Credit

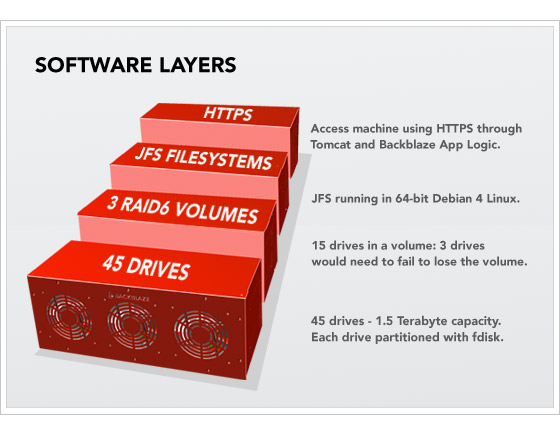
Research and describe the most affordable way to build a 1-Petabyte drive. Note that the setup has to be self-sufficient (i.e. easily usable) and include references. Buying 250 of 4TB drives is not enough because you still need a way to use it. The drive should be built to own, not to rent (Dropbox or similar services doesn’t count, even if it does say “unlimited” storage).

Ans

1-Petabyte drive which means 1000 Terabyte equals to 1 milion of Gigabyte. Based on the amazon general pricing, the cost of 1TB hard drive cost about $45, so that the only cost of hard drive is about $4500 which does not include other parts of the system. Based on the online research, there is a brand which is called BLACKBLAZE provides the design for the 1-Petabyte hard drive system for about $110000. The structure of the system includes 15 pod stack in a rack. Each pod includes the following main parts: “one pod contains one Intel Motherboard with four SATA cards plugged into it. The nine SATA cables run from the cards to nine port multiplier backplanes that each have five hard drives plugged directly into them (45 hard drives in total).” For each hard drive, there is nylon and rubber band to helps on dampen vibration problem. The following was the block parts diagram from BLACKBLAZE. The cost of the system is dominated by the hard drive , the detail list of the cost is listed in the provided website link.



The Storage Pod is configured with free software includes 64-bit Debian 4 Linux and the JFS file system. Also, “they are self-contained appliances, where all access to and from the pods is through HTTPS”. However, the pods functions does not include the iSCSI, no NFS, no SQL, no Fibre Channel.



By building stack of pod to have the 1-Petabyte hard drive is cheaper than other cloud provider e.g. Dell, Amazon. However, there is also some other cost (e.g. Space and Electrical and Maintenance fee etc) associated with this system that we need to take into account before building this system.

Reference: https://www.backblaze.com/blog/petabytes-on-a-budget-how-to-build-cheap-cloud-storage/

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.