CSC 555 Mining Big Data

Assignment 3

Due Tuesday, February 11th

1) MapReduce:

   a) Describe how to implement the following query in MapReduce

```
SELECT SUM(lo_extendedprice)
FROM lineorder, dwdate
WHERE lo_orderdate = d_datekey
  AND d_yearmonth = 'Jan1995'
  AND lo_discount BETWEEN 4 AND 7;
```

Map lineorder : { lo_orderdate: lo_discount, lo_extendedprice }
Map dwdate : { d_datekey : d_yearmonth }

WHERE lo_orderdate = d_datekey::::::::Join::::::: Reducer{ lineorder: d_yearmonth, lo_discount, lo_extendedprice }

Map 3: { d_yearmonth: lo_discount, lo_extendedprice }

Reducer d_yearmonth = 'Jan1995' …….. Reducer{ d_yearmonth = 'Jan1995' : lo_discount, lo_extendedprice }

Map 4: { d_yearmonth , lo_discount : lo_extendedprice }

Reducer d_yearmonth = 'Jan1995' …….. Reducer{ d_yearmonth = 'Jan1995' , lo_discount BETWEEN 4 AND 7 : SUM(lo_extendedprice)}

   b)

```
SELECT d_month, COUNT(d_sellingseason)
FROM dwdate
GROUP BY d_month
ORDER BY COUNT(d_sellingseason)
```

Mapper 1 dwdate: { d_month : d_sellingseason}

GROUP BY d_month::::::: Mapper 2: { d_month: d_sellingseason }

Reduce: (key: d_month, value: COUNT(d_sellingseason))

2) Consider a Hadoop job that processes an input data file of size equal to 88 disk blocks (88 different blocks, you can assume that HDFS replication factor is set to 1). The mapper in this job requires 2 minutes to read and fully process a single block of data. Reducer requires 1 second (**not** minute) to produce an answer for one key worth of values and there are a total of 6000 **distinct** keys (mappers generate a lot of key-value pairs, but keys only occur in the 1-6000 range for a total of 6000 unique entries). Assume that each node has a reducer and that the keys are distributed evenly.

   a) How long will it take to complete the job if you only had one Hadoop worker node? For the sake of simplicity, assume that that only one mapper and only one reducer are created on every node.

      The only mapper has to go to every block to do mapping so its 88*2 = 176 mins, then the only one reducer so it's 6000 sec cost equal to 100 mins total is 276 mins.

   b) 30 Hadoop worker nodes?

      30 nodes workers, 88 blocks divide 30 nodes is 2.93. In no any failure situation, workers have to do three rounds so is 2*3= 6 mins. For 30 reducers, which is 6000/30 sec = 200 sec is equal to 3 mins and 20 sec. To sum up total time spend will be 9 min 20 sec.

   c) 50 Hadoop worker nodes?

      50 nodes workers, 88 blocks divide 50 nodes is 1.76. In no any failure situation, workers have to do twos rounds so is 2*2= 4 mins. For 50 reducers, which is 6000/50 sec = 120 sec is equal to 2 mins. To sum up total time spend will be 6 min.

   d) 100 Hadoop worker nodes?

      100 nodes workers, 88 blocks divide 100 nodes is 0.88. In no any failure situation, workers have to do one rounds so is 2*1= 2 mins. For 100 reducers, which is 6000/100sec = 60 sec is equal to 1 min. To sum up total time spend will be 3 min.

   e) Would changing the replication factor have any affect your answers for a-d?

      No, because the replication factor just makes sure if the node failure still keep the data and can sign worker into other nodes.

   You can ignore the network transfer costs as well as the possibility of node failure.

3)

a) Suppose you have a 7-node cluster with replication factor of 3. Describe what MapReduce has to do after it determines that a node has crashed while a job is being processed. For simplicity, assume that the failed node is not replaced and your cluster is reduced to 6 nodes. Specifically:

   i) What does HDFS (the storage layer) have to do in response to node failure in this case?

      If name node doesn't receive any respond from data node, then the name node, which is master node will replicate again into 6 nodes.

   ii) What does MapReduce engine have to do to respond to the node failure? Assume that there was a job in progress because otherwise MapReduce does not need to do anything.

      Any Mapper or the reducer are completed their task, they will reset back to idle and eligible for schedule to other node that has the block.

b) Where does the Mapper store output key-value pairs before they are sent to Reducers?

   Mapper stored the data on local disks, or we called Intermediate files.

c) Can Reducers begin processing before Mapper phase is complete? **Why or why not?**

   The reducer have 3 phase, shuffle, sort, reduce so for shuffle, which is collecting data from mapper so this part might start when other node is working, but for sort, and reduce, reducer will only start when all the mapper done.

4) Using the SSBM schema (http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM_schema_hive.sql) load the Part table into Hive (data available at http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/part.tbl)

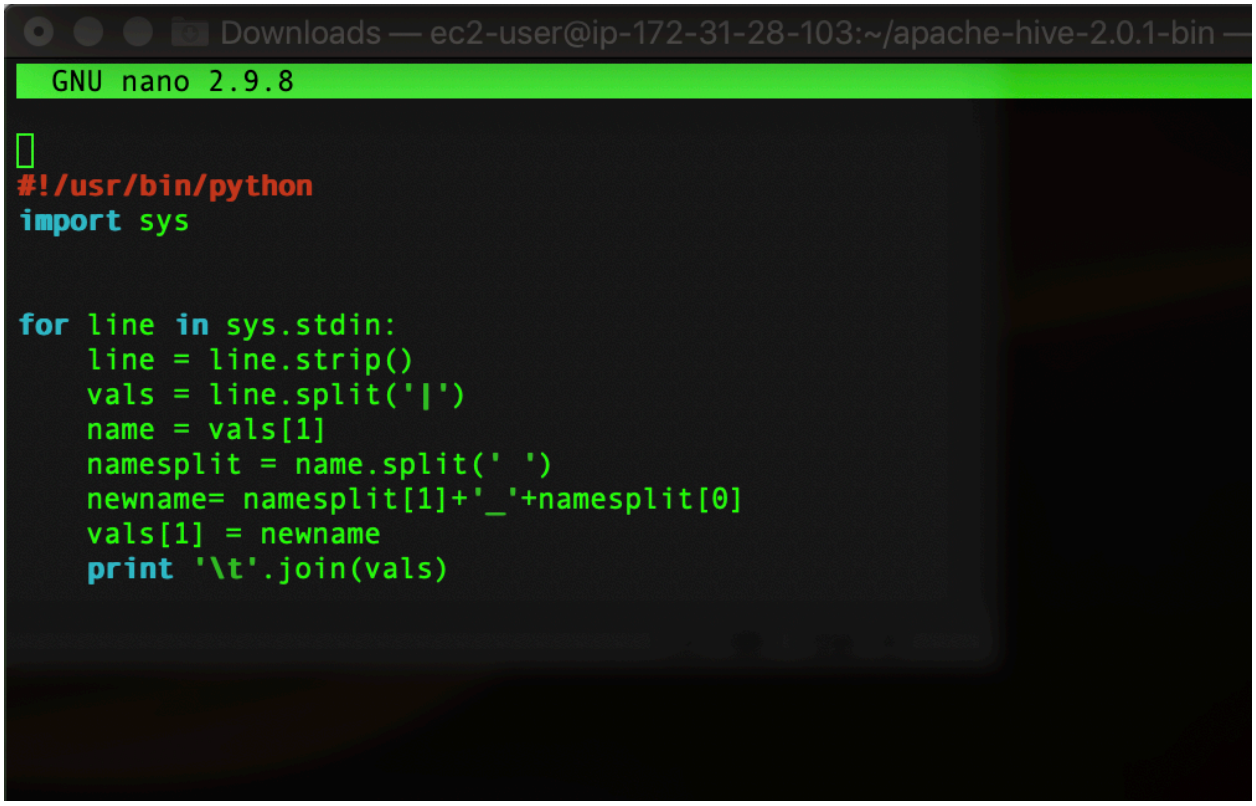
   NOTE: The schema above is made for Hive, but by default Hive assumes '\t' separated content. You will need to modify your CREATE TABLE statement to account for '|' delimiter in the data.

```
stop-dfs.sh; stop-yarn.sh; mr-jobhistory-daemon.sh stop historyserver
rm -rf /tmp/hadoop-ec2-user/dfs/

hdfs namenode -format

start-dfs.sh; start-yarn.sh; mr-jobhistory-daemon.sh start historyserver

$HADOOP_HOME/bin/hadoop fs -chmod g+w /tmp

$HADOOP_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse
```

```
cd $HIVE_HOME

bin/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 '\t' STORED AS TEXTFILE;

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

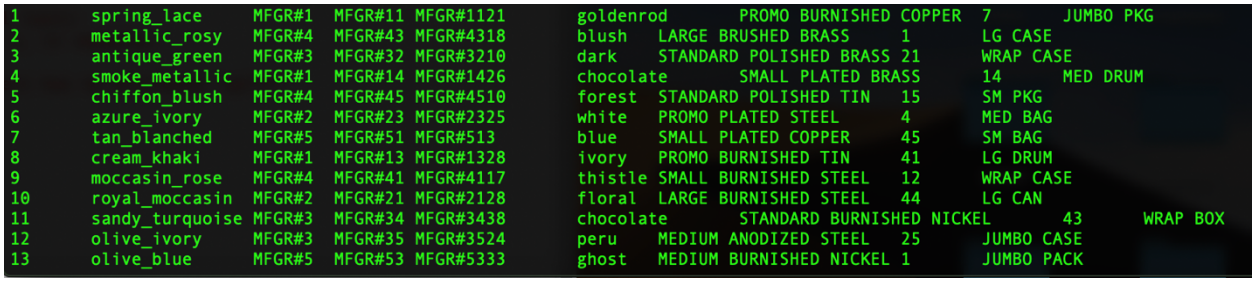
```
hive> select * from part;
OK
1       lace spring     MFGR#1  MFGR#11 MFGR#1121       goldenrod       PROMO BURNISHED COPPER  7   JUMBO PKG
2       rosy metallic   MFGR#4  MFGR#43 MFGR#4318       blush   LARGE BRUSHED BRASS     1       LG CASE
3       green antique   MFGR#3  MFGR#32 MFGR#3210       dark    STANDARD POLISHED BRASS 21      WRAP CASE
4       metallic smoke  MFGR#1  MFGR#14 MFGR#1426       chocolate       SMALL PLATED BRASS      14  MED DRUM
5       blush chiffon   MFGR#4  MFGR#45 MFGR#4510       forest  STANDARD POLISHED TIN   15      SM PKG
6       ivory azure     MFGR#2  MFGR#23 MFGR#2325       white   PROMO PLATED STEEL      4       MED BAG
7       blanched tan    MFGR#5  MFGR#51 MFGR#513        blue    SMALL PLATED COPPER     45      SM BAG
8       khaki cream     MFGR#1  MFGR#13 MFGR#1328       ivory   PROMO BURNISHED TIN     41      LG DRUM
9       rose moccasin   MFGR#4  MFGR#41 MFGR#4117       thistle SMALL BURNISHED STEEL   12      WRAP CASE
10      moccasin royal  MFGR#2  MFGR#21 MFGR#2128       floral  LARGE BURNISHED STEEL   44      LG CAN
```



Use Hive user defined function (i.e., SELECT TRANSFORM from our example, weekday mapper is available here: http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/weekday_mapper.py) to perform the following transformation on Part table (creating a new PartSwap table): in the 2nd column/p_name swap the two words in the column and replace the space by _. For example, rose moccasin would become moccasin_rose or honeydew dim would be dim_honeydew.

```
create table partswap (
 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 '\t' STORED AS TEXTFILE;

ADD FILE /home/ec2-user/apache-hive-2.0.1-bin/namemapperr.py;
```

```
SELECT TRANSFORM (p_partkey, p_name, p_mfgr, p_category, p_brand1, p_color, p_type, p_size,
p_container) USING 'python namemapperr.py' AS (p_partkey, p_name, p_mfgr, p_category, p_brand1,
p_color, p_type, p_size, p_container) FROM part;

select * from partswap;
```



```
1       spring_lace     MFGR#1  MFGR#11 MFGR#1121       goldenrod       PROMO BURNISHED COPPER  7       JUMBO PKG
2       metallic_rosy   MFGR#4  MFGR#43 MFGR#4318       blush   LARGE BRUSHED BRASS     1       LG CASE
3       antique_green   MFGR#3  MFGR#32 MFGR#3210       dark    STANDARD POLISHED BRASS 21      WRAP CASE
4       smoke_metallic  MFGR#1  MFGR#14 MFGR#1426       chocolate       SMALL PLATED BRASS      14      MED DRUM
5       chiffon_blush   MFGR#4  MFGR#45 MFGR#4510       forest  STANDARD POLISHED TIN   15      SM PKG
6       azure_ivory     MFGR#2  MFGR#23 MFGR#2325       white   PROMO PLATED STEEL      4       MED BAG
7       tan_blanched    MFGR#5  MFGR#51 MFGR#513        blue    SMALL PLATED COPPER     45      SM BAG
8       cream_khaki     MFGR#1  MFGR#13 MFGR#1328       ivory   PROMO BURNISHED TIN     41      LG DRUM
9       moccasin_rose   MFGR#4  MFGR#41 MFGR#4117       thistle SMALL BURNISHED STEEL   12      WRAP CASE
10      royal_moccasin  MFGR#2  MFGR#21 MFGR#2128       floral  LARGE BURNISHED STEEL   44      LG CAN
11      sandy_turquoise MFGR#3  MFGR#34 MFGR#3438       chocolate       STANDARD BURNISHED NICKEL       43      WRAP BOX
12      olive_ivory     MFGR#3  MFGR#35 MFGR#3524       peru    MEDIUM ANODIZED STEEL   25      JUMBO CASE
13      olive_blue      MFGR#5  MFGR#53 MFGR#5333       ghost   MEDIUM BURNISHED NICKEL 1       JUMBO PACK
```

Keep in mind that your transform python code (split/join) should **always** use tab ('\t') between fields even if the source data is |-separated. You can also take a look at the transform example included with this assignment for your reference (Examples_Assignment3.doc)

5) Download and install Pig:

```
cd
wget http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/pig-0.15.0.tar.gz
gunzip pig-0.15.0.tar.gz
tar xvf pig-0.15.0.tar
```

set the environment variables (this can also be placed in ~/.bashrc to make it permanent)

```
export PIG_HOME=/home/ec2-user/pig-0.15.0
export PATH=$PATH:$PIG_HOME/bin
```

Use the same vehicles file. Copy the vehicles.csv file to the HDFS if it is not already there.

Now run pig (and use the pig home variable we set earlier):

```
cd $PIG_HOME
bin/pig
```

Create the same table as what we used in Hive, assuming that vehicles.csv is in the home directory on HDFS:

```
hadoop dfs -copyFromLocal /home/ec2-user/vehicles.csv
/user/ec2-user
```

```
VehicleData = LOAD '/user/ec2-user/vehicles.csv' USING PigStorage(',')
AS (barrels08:FLOAT, barrelsA08:FLOAT, charge120:FLOAT, charge240:FLOAT, city08:FLOAT);
```

You can see the table description by

DESCRIBE VehicleData;
Verify that your data has loaded by running:

VehicleG = GROUP VehicleData ALL;
Count = FOREACH VehicleG GENERATE COUNT(VehicleData);
DUMP Count;

How many rows did you get? (if you get an error here, it is likely because vehicles.csv is not in HDFS)

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime      Alias   Featu
re Outputs
job_1581455066244_0024  1       1       5       5       5       5       4       4       4       4       Count,VehicleData,VehicleG      GROUP_BY,COMBINER       hdfs://localh
ost/tmp/temp140918457/tmp1223200921,

Input(s):
Successfully read 34175 records (11766951 bytes) from: "/user/ec2-user/vehicles.csv"

Output(s):
Successfully stored 1 records (9 bytes) in: "hdfs://localhost/tmp/temp140918457/tmp1223200921"

Counters:
Total records written : 1
Total bytes written : 9
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
```

Create the same ThreeColExtract file that you have in the previous assignment, by placing barrels08, city08 and charge120 into a new file using PigStorage .You want the STORE command to record output in HDFS. (discussed in p457, Pig Chapter, "Data Processing Operator section)
NOTE: You can use this to get one column:

OneCol = FOREACH VehicleData GENERATE barrels08, city08, charge120;
Store OneCol INTO ThreeColExtract1 USING PigStorage(',');

Verify that the new file has been created and report the size of the newly created file. (you can use **quit** to exit the grunt shell)

```
Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime      Alias   Fea
ture    Outputs
job_1581455066244_0036  1       0       5       5       5       5       0       0       0       0       VehicleData,a   MAP_ONLY        hdfs://localhost/user/ec2-user/Thre
eColExtract1,

Input(s):
Successfully read 34175 records (11766951 bytes) from: "/user/ec2-user/vehicles.csv"

Output(s):
Successfully stored 34175 records (627867 bytes) in: "hdfs://localhost/user/ec2-user/ThreeColExtract1"

Counters:
Total records written : 34175
Total bytes written : 627867
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0

Job DAG:
job_1581455066244_0036
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$  hadoop fs -ls /user/ec2-user
Found 5 items
drwxr-xr-x   - ec2-user supergroup          0 2020-02-12 01:46 /user/ec2-user/ThreeColExtract
drwxr-xr-x   - ec2-user supergroup          0 2020-02-12 02:32 /user/ec2-user/ThreeColExtract1
drwxr-xr-x   - ec2-user supergroup          0 2020-02-12 02:20 /user/ec2-user/ThreeColExtract_new
drwxr-xr-x   - ec2-user supergroup          0 2020-02-12 01:36 /user/ec2-user/data
-rw-r--r--   1 ec2-user supergroup   11766581 2020-02-12 01:39 /user/ec2-user/vehicles.csv
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$  hadoop fs -ls /user/ec2-user/ThreeColExtract1
Found 2 items
-rw-r--r--   1 ec2-user supergroup          0 2020-02-12 02:32 /user/ec2-user/ThreeColExtract1/_SUCCESS
-rw-r--r--   1 ec2-user supergroup     627867 2020-02-12 02:32 /user/ec2-user/ThreeColExtract1/part-m-00000
```

Submit a single document containing your written answers. Be sure that this document contains your name and "CSC 555 Assignment 3" at the top.