**CSC 555 Mining Big Data**

Assignment 3

**Due Monday, February 11th**

1. MapReduce:
   1. Describe how to implement the following query in MapReduce

SELECT SUM(lo\_revenue)

FROM lineorder, dwdate

WHERE lo\_orderdate = d\_datekey

AND d\_yearmonth = 'Jan1994'

AND lo\_discount BETWEEN 5 AND 7;

Mapper1(lineorder): (key:lo\_orderdate, value: lo\_revenue, sourceid)

For an input block of data from lineorder as the source, for every lineorder record that the code identifies, set lo\_orderdate as the key and lo\_revenue and the source table identifier as the value, but only if the record has a lo\_discount value between 5 and 7.

Mapper2(dwdate): (key:d\_datekey, value: sourceid)

For an input block of data from dwdate as the source, for every dwdate record that the code identifies, set d\_datekey as the key and the source table identifier as the value (no other values are needed). The mapper will only output a key-value pair where the record has a d\_yearmonth value equaling Jan1994.

Reducer1: For each lo\_orderdate received from Mapper1, pair it with a d\_datekey received from Mapper2. If no match is found, discard the Mapper1 key-value. After all keys from Mapper1 are matched or eliminated, sum all of the remaining lo\_revenue values and output; there will be a single value written. Note, after pairing is complete, the key-values from Mapper2 can be discarded.

* 1. SELECT d\_month, COUNT(DISTINCT d\_sellingseason)

FROM dwdate

GROUP BY d\_month

ORDER BY COUNT(DISTINCT d\_sellingseason)

Mapper1: (key: d\_month, value: d\_sellingseason)

For an input block of data, for every dwdate records that the code identifies, set d\_month as the key and set d\_sellingseason as a value.

Reducer1: For each d\_month received, output d\_month and the count of unique values of d\_sellingseason.

Second pass for sorting, applied to output of previous pass.

Mapper2: (key: count\_d\_sellingseason, value: d\_month)

For an input block of data, for each record with the count of distinct d\_sellingseason and d\_month, set the count of distinct d\_sellingseasaon as the key and the corresponding d\_month as the value.

Modify the partitioner to a custom range function in order to enable key-based sorting.

Reducer2: For each count\_d\_sellingseason received, output the d\_month values as a list (e.g., 2, June, July). Note that the result is not going to be exactly as the SQL query output because of duplicate count\_d\_sellingseason entries.

\*Answer largely borrowed for Assignment3 example

1. Consider a Hadoop job that processes an input data file of size equal to 120 disk blocks (120 different blocks, you can assume that HDFS replication factor is set to 1). The mapper in this job requires 1 minute to read and process a single block of data. Reducer requires 1 second (not 1 minute) to produce an answer for one key worth of values and there are a total of 4000 **distinct** keys (mappers generate a lot more key-value pairs, but there 4000 unique keys). Assume that each node has a reducer and that the keys are distributed evenly.
   1. 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 one mapper and one reducer are created on every node.

3 hours, 6 minutes, 40 seconds

* 1. 10 Hadoop worker nodes?

18 minutes, 40 seconds\*

* 1. 30 Hadoop worker nodes?

6 minutes, 13.3 seconds\*

* 1. 50 Hadoop worker nodes?

4 minutes, 20 seconds\*

* 1. Why (or why not) would the introduction of the combiner affect the runtime of this job?

Although we are not considering network transfer costs for our calculations here, the Combiner performs aggregation on the output from the mappers on the same nodes as the Mappers. As a result of aggregation, fewer records are transferred over the network to the Reducers, reducing network transfer cost.

Also, the amount of data to be processed by the Reducers will be reduced.

* 1. How would changing the replication factor affect your answers for a-d?

The write time of the reduce tasks results will increase.

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

\*This assumes that each node runs in equal time.

* 1. Suppose you have a 6-node cluster with replication factor of 3. Describe what MapReduce has to do after it determines that a node has crashed while a job was being processed. For simplicity, assume that the failed node is not replaced and your cluster is reduced to 5 nodes. Specifically:
     1. What does HDFS (the storage layer/NameNode) have to do in response to node failure in this case?

The NameNode will direct the replication of the blocks that were on the dead node to the remaining nodes. In this case, replication factor is 3, so there are still two copies of the blocks remaining that were previously on the failed node.

* + 1. What does MapReduce execution 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 to address a failure.

Any failed Map tasks must be restarted. These failed tasks will be set to idle by the Master and will be run on one of the remaining Workers once available. For a failed Reducer, its currently executing Reduce tasks are set to idle and rescheduled to run on another reducer.

* 1. Where does the Mapper store output key-value pairs before they are sent to Reducers?

On the that Mapper’s node.

* 1. Why can’t Reducers begin processing before Mapper phase is complete?

The Reducers need the entire dataset before they can begin processing. For example, the Reducers for a word count job cannot count the words until all of the words are present.

1. 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 or this won’t work.

Use Hive user defined function (i.e., SELECT TRANSFORM with 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 transformed table): for 2nd and 7th columns, split it into individual columns. That is, a value in 2nd column, ‘blush maroon’ now becomes 2nd and 3rd column with ‘blush’ and ‘maroon’ respectively. Similarly, the 7th column will be transformed from ‘STANDARD BURNISHED NICKEL’ into three columns with values ‘STANDARD’, ‘BURNISHED’, and ‘NICKEL‘.

Please be sure that you create the entire new table, not just the transformed column. You will **add a total of 3 new columns** to the original part table as a result.

Remember that your transform python code (split/join) should always use tab (‘\t’) between fields even if the source data is |-separated.

**Create Table Commands:**

CREATE TABLE part (

p\_partkey INT,

p\_name STRING,

p\_mfgr STRING,

p\_category STRING,

p\_brand1 STRING,

p\_color STRING,

p\_type STRING,

p\_size INT,

p\_container STRING)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

CREATE TABLE part2 (

p\_partkey INT,

p\_name1 STRING,

p\_name2 STRING,

p\_mfgr STRING,

p\_category STRING,

p\_brand1 STRING,

p\_color STRING,

p\_type1 STRING,

p\_type2 STRING,

p\_type3 STRING,

p\_size INT,

p\_container STRING)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '\t' STORED AS TEXTFILE;

**assignment3.py**

#!/usr/bin/python

import sys

for line in sys.stdin:

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

name = line[1].strip().split(' ')

type = line[6].strip().split(' ')

print '\t'.join([line[0], name[0], name[1], \

line[2], line[3], line[4], line[5], \

type[0], type[1], type[2], line[7], \

line[8]])

**Insert Command and Output:**

hive> INSERT OVERWRITE TABLE part2 SELECT TRANSFORM (p\_partkey, p\_name, p\_mfgr, p\_category, p\_brand1, p\_color, p\_type, p\_size, p\_container) USING 'assignment3.py' AS (p\_partkey, p\_name1, p\_name2 ,p\_mfgr, p\_category, p\_brand1, p\_color, p\_type1, p\_type2, p\_type3, p\_size, p\_container) FROM part;

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = ec2-user\_20190211042254\_014220ec-65ec-4391-a7fb-3ad582b8b481

Total jobs = 3

Launching Job 1 out of 3

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_1549842415735\_0039, Tracking URL = http://ip-172-31-35-207.us-east-2.compute.internal:8088/proxy/application\_1549842415735\_0039/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1549842415735\_0039

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0

2019-02-11 04:23:02,577 Stage-1 map = 0%, reduce = 0%

2019-02-11 04:23:37,775 Stage-1 map = 100%, reduce = 0%

2019-02-11 04:23:38,802 Stage-1 map = 0%, reduce = 0%

2019-02-11 04:23:50,491 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.22 sec

MapReduce Total cumulative CPU time: 5 seconds 220 msec

Ended Job = job\_1549842415735\_0039

Stage-4 is selected by condition resolver.

Stage-3 is filtered out by condition resolver.

Stage-5 is filtered out by condition resolver.

Moving data to: hdfs://localhost/user/hive/warehouse/part2/.hive-staging\_hive\_2019-02-11\_04-22-54\_855\_8906110457229521876-1/-ext-10000

Loading data to table default.part2

MapReduce Jobs Launched:

Stage-Stage-1: Map: 1 Cumulative CPU: 5.22 sec HDFS Read: 17146260 HDFS Write: 16939338 SUCCESS

Total MapReduce CPU Time Spent: 5 seconds 220 msec

OK

Time taken: 57.142 seconds

**Test Queries:**

hive> SELECT \* FROM part2 limit 1;

OK

1 lace spring MFGR#1 MFGR#11 MFGR#1121 goldenrod PROMO BURNISHED COPPER 7 JUMBO PKG

Time taken: 0.136 seconds, Fetched: 1 row(s)

hive> SELECT \* FROM part limit 1;

OK

1 lace spring MFGR#1 MFGR#11 MFGR#1121 goldenrod PROMO BURNISHED COPPER 7 JUMBO PKG

Time taken: 0.092 seconds, Fetched: 1 row(s)

1. 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:

**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)

34,174

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;

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)

File size = 627,867 bytes

[ec2-user@ip-172-31-35-207 ~]$ hadoop fs -ls ThreeColExtract.csv/part-m-00000

-rw-r--r-- 1 ec2-user supergroup 627867 2019-02-11 01:22 ThreeColExtract.csv/part-m-00000

Note, I used comma to separate the columns

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