Alex Teboul
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;
```

- One way would be to use 2 mappers and 1 reducer for this query in MapReduce. The first mapper could take lo_orderdate as key and lo_extendedprice as value from what I assume would be lineorder as the source table. Adding in this source . If possible only take records for which lo_discount is between 4 and 7, because this is from the same table.
  - M1(lo_orderdate, lo_extendedprice + lineorder_sourcetable_id)
- The second mapper could take d_datekey as key and the sourcetable_id in dwdate (which I assume it has as well) as the values. Then select only the records from dwdate where d_yearmonth='Jan1995'.
  - M2(d_datekey, dwdate_sourcetable_id)
- Then the reducer could join every lo_orderdate from M1 with the d_datekey records of M2. From this group of matching keys the lo_extendedprice could get summed up. This should only output one row/record in the end which is that sum.

   b)

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

- M1 could take d_month as key and d_sellingseason as value from dwdate. Then send this to a R1 reducer which would take that d_month key and output the pair of the d_month and count of of d_sellingseason. This should accomplish the group by.
- Then to get the order by working (second pass for sorting like in the A3 example doc). M2 mapper takes the count of d_sellingseason and sets the value to d_month. Example shows that you can modify the partitioner to a custom range function that will enable the sorting to take place. The R2 goes for every count of d_sellingseason and outputs the d_month values as a list. From here send the output like the query except that you could get multiple d_month rows which could get cleaned up in a next step.

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.
    i) **Number of Blocks:** 88 Blocks
    ii) **Mapper:** 2min / 1 Block
    iii) **Reducer:** 1 sec / 1 key = 6000 sec / 6000key = 100 min / 6000 key

b) 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.

- 88 Blocks * (2 min / 1 Blocks) = 176min
- 100min/6000keys = 100min
- **1 Node Total = 276 minutes**

c) 30 Hadoop worker nodes?

- 30 blocks in 2 min + 30 blocks in 2 min + 15 blocks in 2 min = 6min
- 100min/6000keys = 100min/30nodes = 3.33min
- **30 Node Total = 9.33 minutes**

d) 50 Hadoop worker nodes?

- 50 blocks in 2min + 38 blocks in 2 min = 4 min
- 100min/6000keys = 100min/50nodes = 2min
- **50 Node Total = 6 minutes**

e) 100 Hadoop worker nodes?

- 88 blocks in 2 min = 2min
- 100min/6000keys = 100min/100nodes = 1min
- **100 Node Total = 3 minutes**

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

- I believe it could have a negative impact on performance to increase the replication factor. If you increased the factor to the point where all the blocks are at all nodes, then you end up with the same runtimes as in part a. In a real world situation, a balance would have to be struck between the runtime issues and availability of blocks, network transfer costs, and node failure – but we're supposed to ignore those for this question.

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?

- Well there are 2 copies of the blocks from the failed node distributed amongst the other nodes, so those blocks would need to get processed in the other nodes. It might replicate some blocks to the other nodes if you still needed to keep the replication factor of 3.

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.

- Map and reduce tasks get cancelled or set to idle and then they must be run/rescheduled to be completed by another worker node. This will take place when the workers have completed the tasks they are currently involved in. Also they would have to take place at a node that has the block that was being worked on.

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

- On that node where the job is run.

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

- No, the reducer needs the whole output from the mapper to do its work so it can't begin until it receives that. For example, to get an average word length or count of words you need all those lengths or all those words.

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)

```
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 '|';
OK
Time taken: 0.365 seconds
hive>
```

- Part table going in^

```
hive> LOAD DATA LOCAL INPATH 'part.tbl'
    > OVERWRITE INTO TABLE part;
Loading data to table default.part
OK
Time taken: 2.8 seconds
```

- Getting the data into the part table^

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.

```
hive> select p_name from part limit 2;
OK
lace spring
rosy metallic
Time taken: 1.047 seconds, Fetched: 2 row(s)
hive>
```

- Checking that the data loaded into the part table ^

```
hive> select count(*) from part;
WARNING: Hive-on-MR is deprecated in Hive 2 and
onsider using a different execution engine (i.e.
Query ID = ec2-user_20200215025826_2879de14-f64a
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile tim
In order to change the average load for a reduce
  set hive.exec.reducers.bytes.per.reducer=<numb
In order to limit the maximum number of reducers
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1581732433853_0003, Tracking
8/proxy/application_1581732433853_0003/
Kill Command = /home/ec2-user/hadoop-2.6.4/bin/h
Hadoop job information for Stage-1: number of ma
2020-02-15 02:58:34,468 Stage-1 map = 0%,  reduc
2020-02-15 02:58:41,995 Stage-1 map = 100%,  red
2020-02-15 02:58:49,532 Stage-1 map = 100%,  red
MapReduce Total cumulative CPU time: 2 seconds 7
Ended Job = job_1581732433853_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CP
7 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 770 ms
OK
200000
Time taken: 24.561 seconds, Fetched: 1 row(s)
hive>
```

- Confirming all the rows loaded in as well.

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.

```
ec2-user@ip-172-31-41-200:~
  GNU nano 2.9.8                                    partswap.py

#!/usr/bin/python
import sys

for line in sys.stdin:
    line = line.strip()
    vals = line.split('\t')
    name = vals[1]
    first, last = name.split(' ')
    name = '_'.join([last, first])
    vals[1] = name
    print '\t'.join(vals)
```

- Python code to try and swap the first and last words in that column.

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)

```
hive> 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';
OK
Time taken: 1.88 seconds
hive>
```

- Making the partswap table ^

```
hive> SELECT TRANSFORM (p_partkey, p_name, p_mfgr, p_category, p_brand1, p_color, p_type, p_size, p_container) USING 'python p
me, p_mfgr, p_category, p_brand1, p_color, p_type, p_size, p_container) FROM part;
```

- Getting the partswap table filled with transformed p_name column using partswap.py^

```
hive> select p_name from partswap limit 4;
OK
spring_lace
metallic_rosy
antique_green
smoke_metallic
Time taken: 0.116 seconds, Fetched: 4 row(s)
hive>
```

- Confirm that the partswap table has the appropriate last_first transformation applied.

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.

```
[ec2-user@ip-172-31-41-200 ~]$ jps
4530 NameNode
5191 JobHistoryServer
4682 DataNode
5132 NodeManager
9804 Jps
5005 ResourceManager
4863 SecondaryNameNode
[ec2-user@ip-172-31-41-200 ~]$ hadoop fs -mkdir /data
[ec2-user@ip-172-31-41-200 ~]$ hadoop fs -put vehicles.csv /data/
[ec2-user@ip-172-31-41-200 ~]$ hadoop fs -ls /data
Found 1 items
-rw-r--r--   1 ec2-user supergroup   11766581 2020-02-15 03:36 /data/vehicles.csv
[ec2-user@ip-172-31-41-200 ~]$
```

- Making sure the vehicles file is in HDFS^

```
[ec2-user@ip-172-31-41-200 ~]$ hadoop fs -put vehicles.csv /user/ec2-user/
[ec2-user@ip-172-31-41-200 ~]$ hadoop fs -ls /user/ec2-user
Found 1 items
-rw-r--r--   1 ec2-user supergroup   11766581 2020-02-15 03:53 /user/ec2-user/vehicles.csv
[ec2-user@ip-172-31-41-200 ~]$
```

- ***Corrected ^FILE IN HDFS to work below

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;

```
grunt> DESCRIBE VehicleData;
VehicleData: {barrels08: float,barrelsA08: float,charge120: float,charge240: float,city08: float}
grunt>
```

Verify that your data has loaded by running:

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

```
d, use fs.defaultFS
2020-02-15 03:55:55,637 [main] INFO
enerate code.
2020-02-15 03:55:55,661 [main] INFO
2020-02-15 03:55:55,664 [main] INFO
cess : 1
(34174)
grunt>
```

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

- 34,174 rows ^

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;

```
grunt> ThreeColExtract = FOREACH VehicleData GENERATE barrels08, city08, charge120;
grunt> DUMP ThreeColExtract;
(11.75609,24.0,0.0)
(13.1844,21.0,0.0)
(14.964294,19.0,0.0)
(14.327048,20.0,0.0)
(15.689436,18.0,0.0)
(15.689436,18.0,0.0)
(18.304342,16.0,0.0)
grunt>
```

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)

```
grunt> STORE ThreeColExtract INTO 'ThreeColExtract'
>> USING PigStorage ('_');
2020-02-15 04:35:09,793 [main] INFO  org.apache.hadoop.conf.Co
ultFS
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/ThreeColExtract"
```

- The new file is 627867 bytes large.

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