

# CSC 555 Mining Big Data

## Assignment 2 (due Sunday, 2/2)

Suggested reading: Mining of Massive Datasets: Chapter 2.

Hadoop: The Definitive Guide: Chapter 17 (Hive).

- 1) Describe how you would implement a MapReduce job consisting of Map and Reduce description. You can describe it in your own words or as pseudo-code. Keep in mind that map task reads the input file and produces (key, value) pairs. Reduce task takes a list of (key, value) pairs for each key and combines all values for each key. Remember that Map operates on individual blocks and Reduce on individual keys with a set of values. Thus, for Mapper you need to state what your code does given a block of data and for Reduce you need to state what your reducer does for each key. You don't have to explain how to parse the file and extract numbers/names.

- a) For a Student table (ID, Name, Address, Phone, City, State), convert

```
SELECT Year, Month, COUNT(Name)
FROM Student
GROUP BY Year, Month;
```

**Map: {Year \_Month: Student}**

**Mapper will read it into Key "Year \_Month" and Value Student**

**Reduce: same Year, Month in to reducer, {Year \_Month: Value: COUNT(\*)}**

- b) For Employee (EID, First, Last, Phone, Age) and Agent(AID, First, Last, Address), find everyone with the same name using MapReduce:

```
SELECT a.First, a.Last, EID, AID, Phone
FROM Employee as e, Agent as a
WHERE e.Last = a.Last AND e.First = a.First;
```

**Map Employee: {e.First\_e.Last: EID, Phone }**

**Map Agent: {a.First\_a.Last: AID }**

**Reducer will read it into Key "e.First\_e.Last" and "a.First\_a.Last" to combine them if they are both same.**

**Reduce: same Year, Month in to reducer, { a.First\_a.Last : EID, AID, Phone }**

- c) Same tables:

```
SELECT Age, COUNT(DISTINCT a.Last)
FROM Employee, Agent
WHERE EID = AID
GROUP BY Age;
```

**Map Employee: {EID: Age }**

**Map Agent: { AID: a.Last }**

**Map3: {Age: a.Last }**

**Reduce: {Age, COUNT(DISTINCT a.Last)}**

- 2) Suppose you are tasked with analysis of the company's web server logs. The log dump contains a large amount of information with up to 8 different attributes (columns). You regularly run a Hadoop job to perform analysis pertaining to 3 specific attributes – TimeOfAccess, OriginOfAccess and FileName.

- a) How would you attempt to speed up the repeated execution of the query? (this is an intentionally open-ended question, there are several acceptable answers)

**Make the reducer size small, so they will have more reducer work in the same time.**

- b) If a Mapper task fails while processing a block of data – what is the location (which node) where MapReduce framework will prefer to restart it?

**If a Mapper task fails while processing a block of data, the master Node will schedule a Worker when one becomes available to the block.**

- c) If the job is executed with 4 Reducers
- How many files does the output generate?  
**One files**
  - Suggest one possible hash function that may be used to assign keys to reducers.

**Aggregation, to aggregate the value in attributes.**

- d) True or False?

- i) A message that was encrypted with a public key can be decrypted with a corresponding private key

**True**

- ii) A message that was encrypted with a private key can be decrypted with a corresponding public key

**True**

- iii) A message that was encrypted with a public key can only be read by its intended recipient, the holder of the private key

**False**

- 3) Consider a Hadoop job that processes an input data file of size equal to 45 disk blocks (45 different blocks, you can assume that HDFS replication factor is set to 1). The mapper in this job requires 1 minute to read and fully process a single block of data. For the purposes of this assignment, you can assume that the reduce part of this job takes zero time.

- a) Approximately how long will it take to process the file if you only had one Hadoop worker node? You can assume that only one mapper is created on every node.

**If there is no failure in mapping process and every node working time is same, then the only node will be mapping every block one by one so it 45 min.**

- b) 20 Hadoop worker nodes?

**If there is no failure in mapping process and every node working time is same, then the 20 nodes will take  $45/20 = 2.25$ , so 3 rounds for 20 nodes is 3 min in mapping.**

- c) 50 Hadoop worker nodes?

**If there is no failure in mapping process and every node working time is same, then the 50 nodes will take 1 min in mapping, they might have 5 nodes doesn't work.**

- d) 75 Hadoop worker nodes?

**If there is no failure in mapping process and every node working time is same, then the 75 nodes will take 1 min in mapping, they might have 30 nodes doesn't work.**

- e) Now suppose you were told that the replication factor has been changed to 3? That is, each block is stored in triplicate, but file size is still 45 blocks. Which of the answers (if any) in a)-c) above will have to change?

**No. If there is no failure in mapping process and every node working time is same, and it just triplicate to protect when the node fail. Not changing the time with the node processing time.**

You can ignore the network transfer costs and other potential overheads as well as the possibility of node failure. If you feel some information is missing please be sure to state your assumptions.

- 4) In this section we are going to use Hive to run a few queries over the Hadoop framework. These instructions assume that you are starting from a working Hadoop installation. It should be sufficient to start your instance and the Hadoop framework on it.

Hive commands are listed in **Calibri bold font**

- a) Download and install Hive:

**cd**

(this command is there to make sure you start from home directory, on the same level as where hadoop is located)

**wget <http://rasinsrv07.esteis.cti.depaul.edu/CSC555/apache-hive-2.0.1-bin.tar.gz>**

**gunzip apache-hive-2.0.1-bin.tar.gz**

**tar xvf apache-hive-2.0.1-bin.tar**

set the environment variables (can be automated by adding these lines in ~/.bashrc). If you don't, you will have to set these variables every time you use Hive.

**export HIVE\_HOME=/home/ec2-user/apache-hive-2.0.1-bin**

**export PATH=\$HIVE\_HOME/bin:\$PATH**

**\$HADOOP\_HOME/bin/hadoop fs -mkdir /tmp**

**\$HADOOP\_HOME/bin/hadoop fs -mkdir /user/hive/warehouse**

(if you get an error here, it means that /user/hive does not exist yet. Fix that by running

**\$HADOOP\_HOME/bin/hadoop fs -mkdir -p /user/hive/warehouse instead)**

**\$HADOOP\_HOME/bin/hadoop fs -chmod g+w /tmp**

**\$HADOOP\_HOME/bin/hadoop fs -chmod g+w /user/hive/warehouse**

We are going to use Vehicle data (originally from <http://www.fueleconomy.gov/feg/download.shtml>)

You can get the already unzipped, comma-separated file from here:

**wget <http://rasinsrv07.esteis.cti.depaul.edu/CSC555/vehicles.csv>**

You can take a look at the data file by either

**nano vehicles.csv** or

**more vehicles.csv** (you can press space to scroll and q or Ctrl-C to break out)

Note that the first row in the data is the list of column names. What follows after commands that start Hive, is the table that you will create in Hive loading the first 5 columns. Hive is not particularly sensitive about invalid or partial data, hence if we only define the first 5 columns, it will simply load the first 5 columns and ignore the rest.

You can see the description of all the columns here (atvtype was added later)

<http://www.fueleconomy.gov/feg/ws/index.shtml#vehicle>

Create the ec2-user directory on the HDFS side (absolute path commands should work anywhere and not just in Hadoop directory as bin/hadoop does). Here, we are creating the user “home” directory on the HDFS side.

**hadoop fs -mkdir /user/ec2-user/**

Run hive (from the hive directory because of the first command below):

**cd \$HIVE\_HOME**

**\$HIVE\_HOME/bin/schematool -initSchema -dbType derby**

(NOTE: This command initializes the database metastore. If you need to restart/reformat or see errors related to meta store, run **rm -rf metastore\_db/** and then repeat the above initSchema command)

**bin/hive**

You can now create a table by pasting this into the Hive terminal:

```
CREATE TABLE VehicleData (  
barrels08 FLOAT, barrelsA08 FLOAT,  
charge120 FLOAT, charge240 FLOAT,  
city08 FLOAT)  
ROW FORMAT DELIMITED FIELDS  
TERMINATED BY ',' STORED AS TEXTFILE;
```

You can load the data (from the local file system, not HDFS) using:

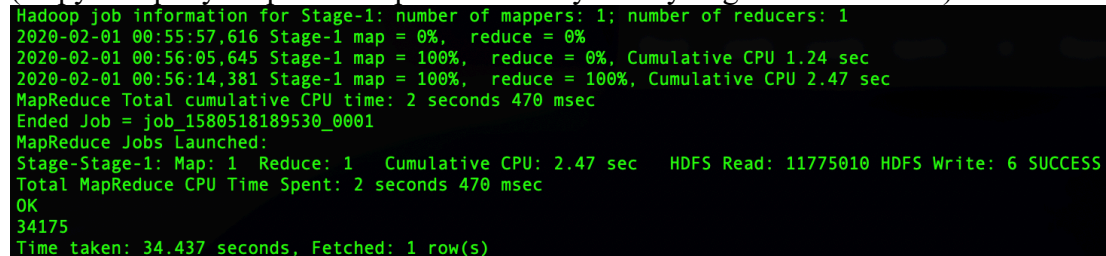
```
LOAD DATA LOCAL INPATH '/home/ec2-user/vehicles.csv'  
OVERWRITE INTO TABLE VehicleData;
```

(NOTE: If you downloaded vehicles.csv file into the hive directory, you have to change file name to /home/ec2-user/apache-hive-2.0.1-bin/vehicles.csv instead)

Verify that your table had successfully loaded by running

**SELECT COUNT(\*) FROM VehicleData;**

(Copy the query output and report how many rows you got as an answer.)



```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2020-02-01 00:55:57,616 Stage-1 map = 0%, reduce = 0%  
2020-02-01 00:56:05,645 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.24 sec  
2020-02-01 00:56:14,381 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.47 sec  
MapReduce Total cumulative CPU time: 2 seconds 470 msec  
Ended Job = job_1580518189530_0001  
MapReduce Jobs Launched:  
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.47 sec HDFS Read: 11775010 HDFS Write: 6 SUCCESS  
Total MapReduce CPU Time Spent: 2 seconds 470 msec  
OK  
34175  
Time taken: 34.437 seconds, Fetched: 1 row(s)
```

Run a couple of HiveQL queries to verify that everything is working properly:

**SELECT MIN(barrels08), AVG(barrels08), MAX(barrels08) FROM VehicleData;**

(copy the output from that query)

```

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2020-02-01 00:57:42,581 Stage-1 map = 0%, reduce = 0%
2020-02-01 00:57:50,341 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.49 sec
2020-02-01 00:57:57,853 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.83 sec
MapReduce Total cumulative CPU time: 2 seconds 830 msec
Ended Job = job_1580518189530_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.83 sec HDFS Read: 11777415 HDFS Write: 37 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 830 msec
OK
0.059892      17.820177449476272      47.06831
Time taken: 24.652 seconds, Fetched: 1 row(s)

```

**SELECT (barrels08/city08) FROM VehicleData;**

(you do not need to report the output from that query, but report “Time taken”)

**Time taken: 0.193 seconds, Fetched: 34175 row(s)**

Next, we are going to output three of the columns into a separate file (as a way to transform data for further manipulation that you may be interested in)

**INSERT OVERWRITE DIRECTORY 'ThreeColExtract'**

**SELECT barrels08, city08, charge120**

**FROM VehicleData;**

```

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2020-02-01 00:59:11,344 Stage-1 map = 0%, reduce = 0%
2020-02-01 00:59:18,987 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.98 sec
MapReduce Total cumulative CPU time: 1 seconds 980 msec
Ended Job = job_1580518189530_0003
Stage-3 is selected by condition resolver.
Stage-2 is filtered out by condition resolver.
Stage-4 is filtered out by condition resolver.
Moving data to: hdfs://localhost/user/ec2-user/ThreeColExtract/.hive-staging_hive_2020-02-01_00-59-03_389_69980543847
29834463-1/-ext-10000
Moving data to: ThreeColExtract
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 1.98 sec HDFS Read: 11770539 HDFS Write: 627873 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 980 msec
OK
Time taken: 17.786 seconds

```

You can now exit Hive by running **exit;**

And verify that the new output file has been created (the file will be called 000000\_0)

The file would be created in HDFS in user home directory (/user/ec2-user/ThreeColExtract)

```
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user/ThreeColExtract
```

Found 1 items

```
-rwxr-xr-x  1 ec2-user supergroup      627873 2020-02-01
00:59 /user/ec2-user/ThreeColExtract/000000_0
```

Report the size of the newly created file and include the screenshot.

```

[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user/ThreeColExtract
Found 1 items
-rwxr-xr-x  1 ec2-user supergroup      627873 2020-02-01 00:59 /user/ec2-user/ThreeColExtract/000000_0
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$

```

Next, you should go back to the Hive terminal, create a new table that is going to load 8 columns instead of 5 in our example (i.e. create and load a new table that defines 8 columns by including columns city08U,cityA08,cityA08U) and use Hive to generate a new output file containing only the city08U and cityA08U columns from the vehicles.csv file. Report the size of that output file as well.

```
CREATE TABLE NewVehicleData (
barrels08 FLOAT, barrelsA08 FLOAT,
charge120 FLOAT, charge240 FLOAT,
city08 FLOAT, city08U FLOAT,
cityA08 FLOAT, cityA08U FLOAT)
ROW FORMAT DELIMITED FIELDS
TERMINATED BY ',' STORED AS TEXTFILE;
```

```
LOAD DATA LOCAL INPATH '/home/ec2-user/ThreeColExtract'
OVERWRITE INTO TABLE NewVehicleData;
```

```
INSERT OVERWRITE DIRECTORY 'NewFile'
SELECT cityA08U, cityA08U
FROM NewVehicleData;
```

```
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$ hadoop fs -ls
/user/ec2-user/NewFile
Found 1 items
-rwxr-xr-x  1 ec2-user supergroup    276710 2020-02-01
01:18 /user/ec2-user/NewFile/000000_0
```

```
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user
Found 3 items
drwxr-xr-x  - ec2-user supergroup    0 2020-02-01 01:18 /user/ec2-user/NewFile
drwxr-xr-x  - ec2-user supergroup    0 2020-02-01 00:59 /user/ec2-user/ThreeColExtract
drwxr-xr-x  - ec2-user supergroup    0 2020-01-29 23:43 /user/ec2-user/instead
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user/NewFile
Found 1 items
-rwxr-xr-x  1 ec2-user supergroup    276710 2020-02-01 01:18 /user/ec2-user/NewFile/000000_0
[ec2-user@ip-172-31-28-103 apache-hive-2.0.1-bin]$
```

12

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