**CSC 555 Mining Big Data**

Assignment 2 (due Sunday, September 30th)

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 can assume that you know how to parse the file and extract numbers/names.

* 1. For a Student table (ID, FullName, Address, Phone, Year, Month), convert

SELECT Year, Month, COUNT(FullName)

FROM Student

GROUP BY Year, Month;

For each block of data, the mapper will produce key value pairs consisting of Year and Month as the key and FullName as the value. The reducer will perform a count of all of the occurrences of each key. Because the FullName value itself is not actually needed to determine the result, the mapper could assign a value of 1 for each key and the reducer would simply sum all of the records for each key.

* 1. 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;

Blocks will come from either table Employee or table Agent. The mapper will produce keys containing First and Last from each block. The values will depend upon which table the block came from. For Employee, the values will be EID, Phone, and a table identifier (could be the name or some assigned id). From Agent, the values will be AID and a table identifier.

The reducer will combine values where the keys from Employee match the keys from Agent.

* 1. Same tables:

SELECT Age, COUNT(DISTINCT a.Last)

FROM Employee, Agent

WHERE EID = AID

GROUP BY Age;

Blocks will come from either table Employee or table Agent. The mapper will produce keys containing either EID or AID from each block, depending upon which table the block is from. The values will depend upon which table the block came from. For Employee, the values will be Age and a table identifier (could be the name or some assigned id). From Agent, the values will be Last\* and a table identifier.

There reducer will have multiple steps. First the reducer will combine values from Employee with values from Agent where the keys match, in this case EID and AID. Next the reducer will remove duplicate values of Last and Age, meaning, for a given Age, a specific Last value should only occur once. Finally, the reducer will count each of the remaining records for each age value.

Note, the record matching and deduplication could also be performed by using a combiner prior to sending to the reducer.

\* I’ve made the assumption that Last is coming from Agent based upon the alias reference a.Last, however, the alias isn’t actually declared with the table in the query provided.

1. 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 10 different attributes (columns). You regularly run a Hadoop job to perform analysis pertaining to 3 specific attributes – TimeOfAccess, OriginOfAccess and FileName.
   1. If a Mapper task fails while processing a block of data – what is the best location to restart it?

The Master node, aka Name node, is responsible for detecting failures and will restart all of the Map tasks assigned to that Mapper.

* 1. If the job is executed with 5 Reducers
     1. How many files does the output generate?

Five

* + 1. Suggest one possible hash function that may be used to assign keys to reducers.

From the TimeOfAccess attribute, apply the following function to the minutes value.

Minutes % 5

Assuming that the possible minutes values have a relatively equal rate of occurrence, this hash function will evenly distribute keys to the 5 reducers.

* 1. True or False?
     1. A message that was encrypted with a public key can be decrypted with a corresponding private key

True

* + 1. A message that was encrypted with a private key can be decrypted with a corresponding public key

Technically true, you may do this as a means of signing the message for authenticity. However, you would not do this in practice to secure a message.

* + 1. A message that was encrypted with a public key can only be read by its intended recipient, the holder of the private key

True

1. Consider a Hadoop job that processes an input data file of size equal to 38 disk blocks (38 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.
   1. Approximately how long will it take to process the file if you only had one Hadoop worker node? You can assume that that only one mapper is created on every node.

38 Minutes

* 1. 20 Hadoop worker nodes?

2 minutes

* 1. 50 Hadoop worker nodes?

1 minute

* 1. 75 Hadoop worker nodes?

1 minute

* 1. 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 38 blocks. Which of the answers (if any) in a)-c) above will have to change?

There will be no change to 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.

An assumption is that blocks are distributed equally among nodes.

1. 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**

* 1. 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.cstcis.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.cstcis.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.)

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

*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\_20190203212532\_91021965-3283-44d1-a483-fd49f35ddc7b*

*Total jobs = 1*

*Launching Job 1 out of 1*

*Number of reduce tasks determined at compile time: 1*

*In order to change the average load for a reducer (in bytes):*

*set hive.exec.reducers.bytes.per.reducer=<number>*

*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\_1549226838708\_0001, Tracking URL = http://ip-172-31-35-207.us-east-2.compute.internal:8088/proxy/application\_1549226838708\_0001/*

*Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1549226838708\_0001*

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

*2019-02-03 21:25:45,169 Stage-1 map = 0%, reduce = 0%*

*2019-02-03 21:25:51,927 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.19 sec*

*2019-02-03 21:26:00,624 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.4 sec*

*MapReduce Total cumulative CPU time: 2 seconds 400 msec*

*Ended Job = job\_1549226838708\_0001*

*MapReduce Jobs Launched:*

*Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.4 sec HDFS Read: 11775010 HDFS Write: 6 SUCCESS*

*Total MapReduce CPU Time Spent: 2 seconds 400 msec*

*OK*

*34175*

*Time taken: 29.636 seconds, Fetched: 1 row(s)*

34,175 rows

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)

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

*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\_20190203212808\_ac49941f-bd66-492a-a250-a3d09fa5099d*

*Total jobs = 1*

*Launching Job 1 out of 1*

*Number of reduce tasks determined at compile time: 1*

*In order to change the average load for a reducer (in bytes):*

*set hive.exec.reducers.bytes.per.reducer=<number>*

*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\_1549226838708\_0002, Tracking URL = http://ip-172-31-35-207.us-east-2.compute.internal:8088/proxy/application\_1549226838708\_0002/*

*Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1549226838708\_0002*

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

*2019-02-03 21:28:15,304 Stage-1 map = 0%, reduce = 0%*

*2019-02-03 21:28:22,840 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 1.5 sec*

*2019-02-03 21:28:30,283 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 2.73 sec*

*MapReduce Total cumulative CPU time: 2 seconds 730 msec*

*Ended Job = job\_1549226838708\_0002*

*MapReduce Jobs Launched:*

*Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 2.73 sec HDFS Read: 11777415 HDFS Write: 37 SUCCESS*

*Total MapReduce CPU Time Spent: 2 seconds 730 msec*

*OK*

***0.059892 17.820177449476272 47.06831***

*Time taken: 23.33 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: 2.852 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;**

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)

Report the size of the newly created file.

*[ec2-user@ip-172-31-35-207 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user/ThreeColExtract*

*Found 1 items*

*-rwxr-xr-x 1 ec2-user supergroup 627873 2019-02-03 21:49 /user/ec2-user/ThreeColExtract/000000\_0*

Size = 627,873 bytes

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.

*[ec2-user@ip-172-31-35-207 apache-hive-2.0.1-bin]$ hadoop fs -ls /user/ec2-user/TwoColExtract*

*Found 1 items*

*-rwxr-xr-x 1 ec2-user supergroup 287749 2019-02-03 22:06 /user/ec2-user/TwoColExtract/000000\_0*

*[ec2-user@ip-172-31-35-207 apache-hive-2.0.1-bin]$*

Size = 287,749 bytes

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