Robert Kaszubski

CSC555 – Assignment 3

**1)**

a)

The mapper will have to process both files (employee, agent). Keys will contain First, Last no matter which file the data is coming from as both tables, employee and agent have a First and Last column and that will serve as the key. The value depends on which table the data is coming from. From the employee table, the values will be EID, and AGE. From the Age table the value will be AID.

The reducer will then perform the join. It will return Key: First\_Last and Value: EID, AID, Age

Mapper:

Mapper1e – KEY: First\_Last VALUE: EID, Age

Mapper2a – KEY: First\_Last VALUE: AID

Reducer – KEY: First\_Last Value: EID, AID, Age

b)

The mapper will have to process both files (lineorder, dwdate).

Mapper1 lineorder – KEY: lo\_orderdate VALUE: lo\_extendedprice, lo\_discount

Mapper2 dwdate – KEY: d\_datekey VALUE: d\_yearmonth

Then the reducer will match lo\_orderdate and d\_datekey, check if lo\_discount = 6, and d\_yearmonth = Feb1996. For the pairs that match, it’ll keep. The rest would be discarded. Then it would calculate the sum(lo\_extendedprice), and return the single value

Reducer: KEY: sum(lo\_extendedprice) VALUE: 1

Value can be anything. If we wanted to also group by orderdate, then we could have

KEY: lo\_orderdate/d\_datekey VALUE: sum(lo\_extendedprice)

But by my understanding of the query its simply the sum of all the prices.

c)

You would have to do two MapReduce jobs

Mapper1 : KEY: d\_month, VALUE: d\_year

Reducer1: KEY: d\_month, VALUE: count(d\_year)

So for the first MapReduce job, the Mapper key is d\_month as it is what we are grouping by, and the Value is d\_year which is what we’re trying to find the count of. Then for our first reducer the key is d\_month and our value is the count of d\_year.

Mapper2: KEY: Count(d\_year) VALUE: d\_month

Custom Partitioner

Reducer2: KEY: Count(d\_year) VALUE: d\_month

So, for the second MapReduce job, we take the Count(d\_year) the first reducer returned and use it as the KEY, then Value becomes d\_month. Now we can use a custom partitioner to split the data between the reducers. The reducer then automatically sorts the data based on the key. If we are only using one reducer then we wouldn’t need the custom partitioner as all the data is fed into the same reducer. However, if we have blocks of data and it has to be dividing to multiple different reducers then we would want to set a Count(d\_year) value to evenly partition the data. Then the output files would be properly sorted.

**2)**

a)

Mapper: 72\*1 = 72 minutes

Reducer: 7000 \* 1 / 60 = 116.67

Total: **about 188.67 minutes**

b)

30 nodes:

Mapper: First round: 30, Second: 30, Third: 12

3 \* 1 min = 3 minutes

So we can divide the total number of blocks by the nodes and round up to the nearest integer. Because even though the third round only has 12 blocks of data it’ll still take the full minute.

72/30 = 2.4, rounds up to 3 (rounds) \* 1 (minute each) = 3 minutes

Same for the reducer except in seconds:

7000 / 30 = 233.33, rounds up to 234 seconds

234/60 = 3.9 minutes

Total: **about 6.9 minutes**

c)

50 nodes

Mapper:

72/50 = 1.44, rounds up to 2

2 minutes

Reducer:

7000 / 50 = 140 seconds

140 / 60 = 2.33 minutes

Total: **about 4.33 minutes**

d)

100 nodes:

Mapper:

72/100 = .72, rounds up to 1

1 minute

Reducer:

7000/100 = 70

70/60 = 1.17

Total: **about 2.17 minutes**

e)

Replication factor shouldn’t affect these answers. The only thing that may change is the write speed when the output files from each reducer have to be stored x number of times each instead of just one. But the MapReduce job itself would run at the same speed no matter the replication factor.

**3)**

a)

i) The NameNode which manages the system would identify the files that were on the lost node, find the two other copies of each file on the other nodes, and replicate them on different nodes to maintain a replication factor of 3.

ii)

MapReduce jobs would be set to idle and run on a different node when it is available.

b)

They are stored on the local disk of the node the mapper is running on

c)

No they can’t begin processing before the Mapper phase is complete. They require all blocks to be processed by the mapper as reducers require all data before they can begin.

**4)**

Updated Schema:

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 '|' STORED AS TEXTFILE;

create table partswapped (

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,

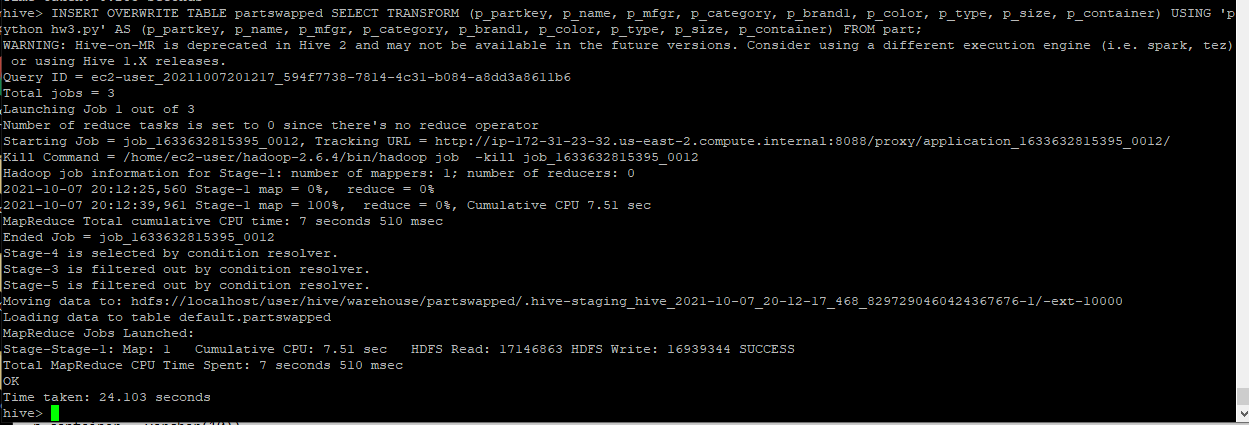
p\_container varchar(10))

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '\t' STORED AS TEXTFILE;



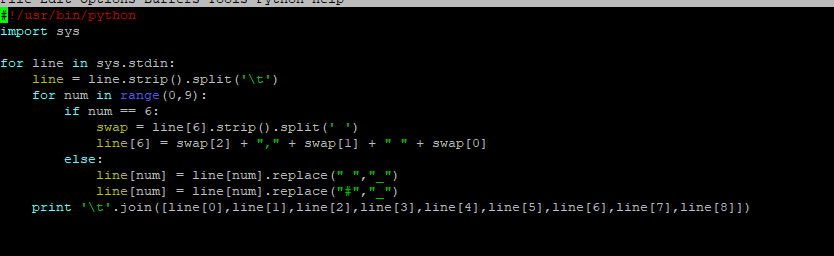
INSERT OVERWRITE TABLE partswapped SELECT TRANSFORM (p\_partkey, p\_name, p\_mfgr, p\_category, p\_brand1, p\_color, p\_type, p\_size, p\_container) USING 'python hw3.py' AS (p\_partkey, p\_name, p\_mfgr, p\_category, p\_brand1, p\_color, p\_type, p\_size, p\_container) FROM part;



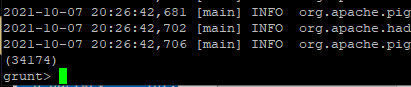
Sample Output:



Python Code (File also attached to submission):



**5)**



34174 ROWS





File was created:



627867 Bytes