DSC 333 and CSC 555 Take-home midterm
Nikhil Bhalala
Dsc-333

In this part of the project, you will 1) Set up a 3-node cluster and 2) perform data warehousing and transformation queries using Hive, Pig and Hadoop streaming on that cluster. The modified Hive-style schema is:
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/SSBM_schema_hive.sql
(**you still have to add the delimiter to table definitions**)
It is based on SSBM benchmark (derived from industry standard TPCH benchmark). The data is at Scale1, or the smallest unit – lineorder is the largest table at about 0.6GB. You can use wget to download the following links. Keep in mind that data is |-separated.
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/dwdate.tbl
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/lineorder.tbl
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/part.tbl
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/supplier.tbl
http://cdmgcsarprd01.dpu.depaul.edu/CSC555/SSBM1/customer.tbl
Please be sure to submit all code (pig, python and HiveQL) and all command lines you executed.

Part 1: Multi-node cluster

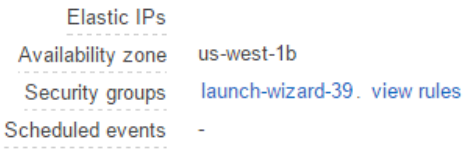
1) Your first step is to setup a multi-node cluster and re-run wordcount. For this part, you will create a 3-node cluster (with a total of 1 master + 2 worker nodes). Include your master node in the workers file, to make sure **all 3** nodes are working.
   You need to perform the following steps:

   1. Create a medium machine on AWS (which will serve as your master). It is possible, but I do not recommend trying to reconfigure your existing Hadoop setup into this new cluster (it is much easier to make 3 new nodes for a total of 4).

      a. When creating a node I recommend changing the default 8G hard drive to 20G.

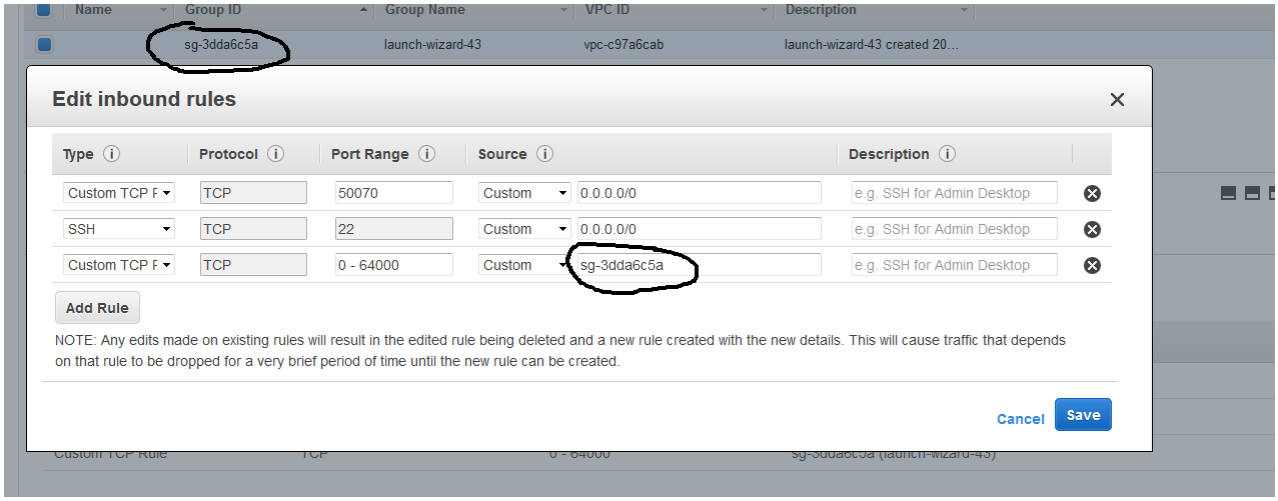
      b. Change your security group setting to open firewall access. We need to open the ports in two different ways. We will open port 50070 for the web interface in order to be able to see the cluster status in a browser. We will also set 0-64000 range opening up all ports. However, we will ensure that the ports are open only **within** the cluster and not to the world.

In order to make changes, you need to do the following. Access the cluster security group (launch-wizard-xx).



Right click on the security group and choose Edit inbound rules

Note that the first line below is opening port 50070. The second line below is the default (port 22 is required for regular SSH connections). The third line opens all ports but ONLY for the same security group (assuming that all of your nodes in the cluster share the same security group). Please note that we previously had some issues with machines being hacked without that last limitation, so don't skip this step



c. Create two new small machines and make sure they are using the same security group that you have configured on the master. You would need to change the security group settings so that both of the workers are sharing the same security group. For that, you can go to "Networking", "Change Security Groups" and check the security group you want.
NOTE: Please make sure to label the machines so that they are easy to find, as it may get a little cluttered.

2. Connect to the master and set up Hadoop similarly to what you did previously. Use the following link:

   http://dbgroup.cdm.depaul.edu/Courses/CSC555/hadoop-2.6.4.tar.gz

   Do not set up Hadoop on the workers – you will only need to configure up Hadoop once.

   a. Configure core-site.xml, adding the **PrivateIP** (do not use public IP) of the master.

```
  limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
<name>fs.defaultFS</name>
<value>hdfs://172.31.7.201/</value>
</property>

</configuration>
[ec2-user@ip-172-31-7-201 ~]$ cat hadoop-2.6.4/etc/hadoop/core-site.xml
```

b. Configure hdfs-site and set replication factor to 2.

```
<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
<name>dfs.replication</name>
<value>2</value>
</property>

</configuration>
[ec2-user@ip-172-31-9-105 ~]$
```

c. cp hadoop-2.6.4/etc/hadoop/mapred-site.xml.template hadoop-2.6.4/etc/hadoop/mapred-site.xml and then configure mapred-site.xml

```
<!-- Put site-specific property overrides in this file. -->

<configuration>

<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>

</configuration>
[ec2-user@ip-172-31-9-105 ~]$ cat hadoop-2.6.4/etc/hadoop/mapred-site.xml
```

d. Configure yarn-site.xml (once again, use PrivateIP of the master)

```
<!-- Site specific YARN configuration properties -->

<property>
<name>yarn.resourcemanager.hostname</name>
<value>172.31.7.201</value>
</property>

<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>

</configuration>
[ec2-user@ip-172-31-7-201 ~]$ cat hadoop-2.6.4/etc/hadoop/yarn-site.xml
```

Finally, edit the workers file and list your 3 nodes (master and 2 workers) using Private IPs

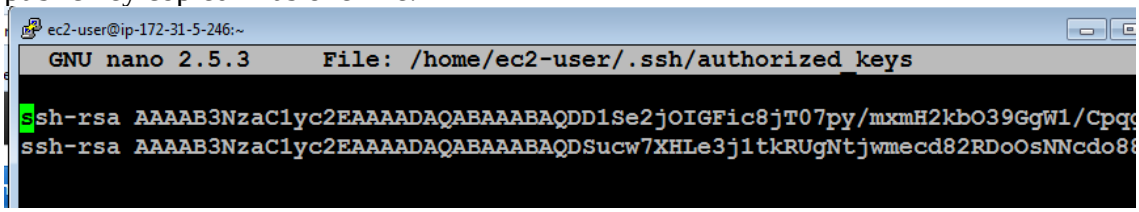
[ec2-user@ip-172-31-7-201 ~]$ cat hadoop-2.6.4/etc/hadoop/workers
172.31.7.201
172.31.5.246
…

Make sure that you use private IP (private DNS is also ok) for your configuration files (such as conf/masters and conf/workers or the other 3 config files). The advantage of the Private IP is that it does not change after your instance is stopped (if you use the Public IP, the cluster would need to be reconfigured every time it is stopped). The downside of the Private IP is that it is only

meaningful within the Amazon EC2 network. So all nodes in EC2 can talk to each other using Private IP, but you cannot connect to your instance from the outside (e.g., from your laptop) because Private IP has no meaning for your laptop (since your laptop is not part of the Amazon EC2 network).

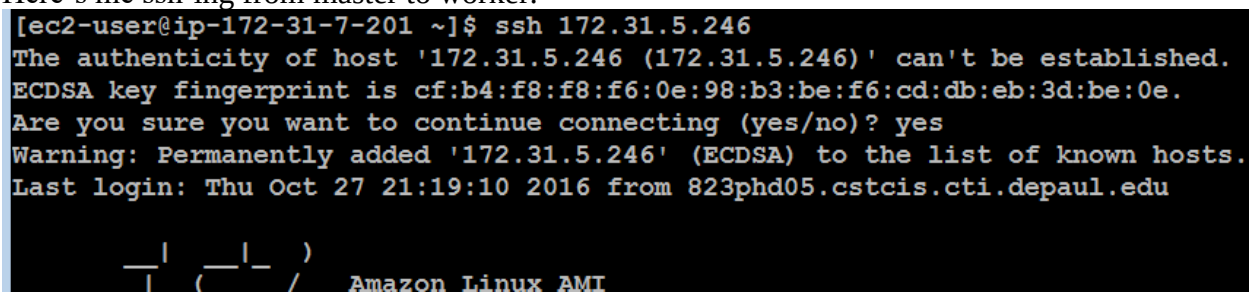
Now, we will pack up and move Hadoop to the workers. All you need to do is to generate and then copy the public key to the worker nodes to achieve passwordless access across your cluster.

1. Run ssh-keygen -t rsa (and enter empty values for the passphrase) on the master node. That will generate .ssh/id_rsa and .ssh/id_rsa.pub (private and public key). You now need to manually copy the .ssh/id_rsa.pub and append it to ~/.ssh/authorized_keys **on each worker.**
   Keep in mind that this is a single-line public key and accidentally introducing a line break (like discussed in class) would prevent the key from matching it's private key pair.
   Note that the example below is NOT the master, but one of the workers (ip-172-31-5-246). The first public key is the .pem Amazon half and the 2nd public key is the master's public key copied in as one line.



   You can add the public key of the master to the master by running this command:
   cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

Make sure that you can ssh to all of the nodes from the master node (by running ssh 54.186.221.92, where the IP address is your worker node) from the master and ensuring that you were able to login. You can exit after successful ssh connection by typing exit (the command prompt will tell you which machine you are connected to, e.g., ec2-user@ip-172-31-37-113). Here's me ssh-ing from master to worker.



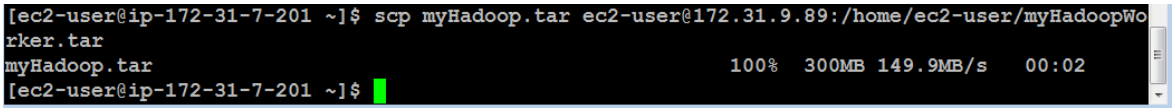
Once you have verified that you can ssh from the master node to every cluster member including the master itself (ssh localhost), you are going to return to the master node (**exit** until your prompt shows the IP address of the master node) and pack the contents of the hadoop directory there. Make sure your Hadoop installation is configured correctly (because from now on, you will have 4 copies of the Hadoop directory and all changes need to be applied in 4 places).

cd (go to root home directory, i.e. /home/ec2-user/)

(pack up the entire Hadoop directory into a single file for transfer. You can optionally compress the file with gzip)
tar cvf myHadoop.tar hadoop-2.6.4
ls -al myHadoop.tar (to verify that the .tar file had been created)

Now, you need to copy the myHadoop.tar file to every non-master node in the cluster. If you had successfully setup public-private key access in the previous step, this command (for each worker node) will do that:

(copies the myHadoop.tar file from the current node to a remote node into a file called myHadoopWorker.tar. Don't forget to replace the IP address with that your worker nodes. By the way, since you are on the Amazon EC2 network, either Public or Private IP will work just fine.)
scp myHadoop.tar ec2-user@54.187.63.189:/home/ec2-user/myHadoopWorker.tar



Once the tar file containing your Hadoop installation from master node has been copied to each worker node, you need to login to each worker node and unpack the .tar file.
You also need to install Java using **sudo yum install ant**. Without Java on the worker nodes, Hadoop will not start.

Run the following command (on each worker node, not on the master) to untar the hadoop file. We are purposely using a different tar archive name (i.e., **myHadoopWorker.tar**), so if you get "file not found" error, that means you are running this command on the master node or have not yet successfully copied myHadoopWorker.tar file to the worker.
tar xvf myHadoopWorker.tar

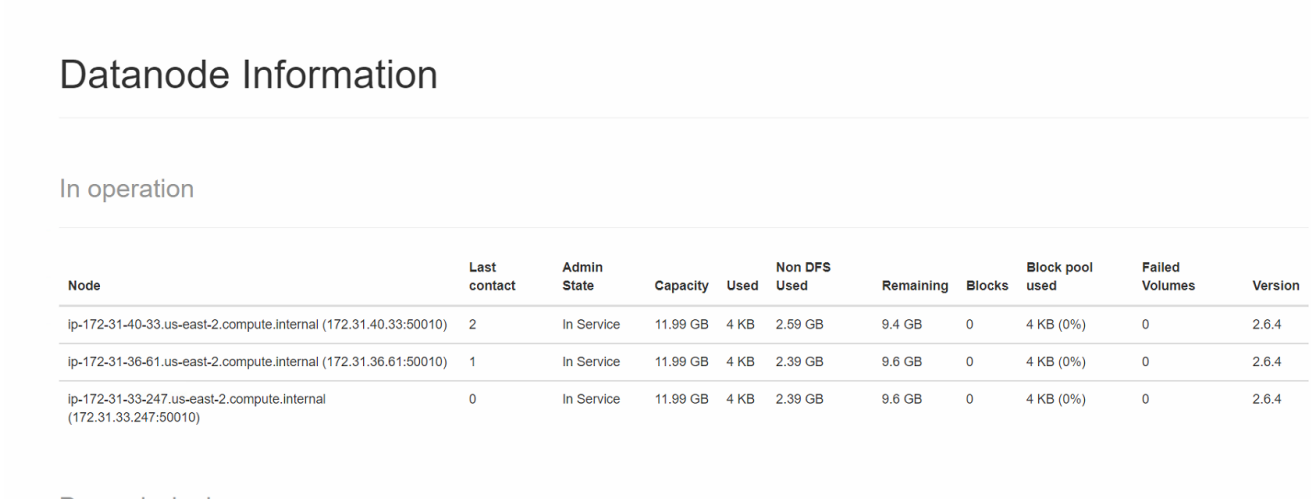
Once you are done, run this on the master (nothing needs to be done on the workers to format the cluster unless you are re-formatting, in which case you'll need to delete the dfs directory).
hadoop namenode -format

Once you have successfully completed the previous steps, you should can start and use your new cluster by going to the master node and running the start-dfs.sh and start-yarn.sh scripts (you do not need to explicitly start anything on worker nodes – the master will do that for you).

You should verify that the cluster is running by pointing your browser to the link below.
http://[insert-the-public-ip-of-master]:50070/

Make sure that the cluster is operational (you can see the 3 nodes under Datanodes tab).

Answer 1-A) Submit a screenshot of your cluster status view.



Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-40-33.us-east-2.compute.internal (172.31.40.33:50010)	2	In Service	11.99 GB	4 KB	2.59 GB	9.4 GB	0	4 KB (0%)	0	2.6.4
ip-172-31-36-61.us-east-2.compute.internal (172.31.36.61:50010)	1	In Service	11.99 GB	4 KB	2.39 GB	9.6 GB	0	4 KB (0%)	0	2.6.4
ip-172-31-33-247.us-east-2.compute.internal (172.31.33.247:50010)	0	In Service	11.99 GB	4 KB	2.39 GB	9.6 GB	0	4 KB (0%)	0	2.6.4

Answer 1-B) Repeat the steps for wordcount using bioproject.xml from Assignment 2 and submit screenshots of running it.

```
22/10/28 00:32:05 INFO mapreduce.Job:  map 100% reduce 100%
22/10/28 00:32:06 INFO mapreduce.Job: Job job_1666915259814_0001 completed successf
22/10/28 00:32:06 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=59605201
                FILE: Number of bytes written=86827979
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=231153307
                HDFS: Number of bytes written=20056175
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=55818
                Total time spent by all reduces in occupied slots (ms)=11365
                Total time spent by all map tasks (ms)=55818
                Total time spent by all reduce tasks (ms)=11365
                Total vcore-milliseconds taken by all map tasks=55818
                Total vcore-milliseconds taken by all reduce tasks=11365
                Total megabyte-milliseconds taken by all map tasks=57157632
                Total megabyte-milliseconds taken by all reduce tasks=11637760
        Map-Reduce Framework
                Map input records=5284546
                Map output records=18562366
                Map output bytes=279356680
                Map output materialized bytes=26902454
                Input split bytes=208
                Combine input records=20053191
                Combine output records=2673165
                Reduce input groups=1040390
                Reduce shuffle bytes=26902454
                Reduce input records=1182340
                Reduce output records=1040390
                Spilled Records=3855505
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=947
                CPU time spent (ms)=40580
                Physical memory (bytes) snapshot=577392640
                Virtual memory (bytes) snapshot=6323806208
                Total committed heap usage (bytes)=334499840
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=231153099
        File Output Format Counters
                Bytes Written=20056175

real    0m55.719s
user    0m3.972s
sys     0m0.223s
[ec2-user@ip-172-31-40-33 ~]$
```

```
subarctic       21
[ec2-user@ip-172-31-40-33 ~]$ hadoop fs -du /data/wordcount1/
0         /data/wordcount1/_SUCCESS
20056175  /data/wordcount1/part-r-00000
[ec2-user@ip-172-31-40-33 ~]$
```

Answer 1-C) Submit a short paragraph with a discussion about how the results compare (faster? slower? How much faster/slower?)

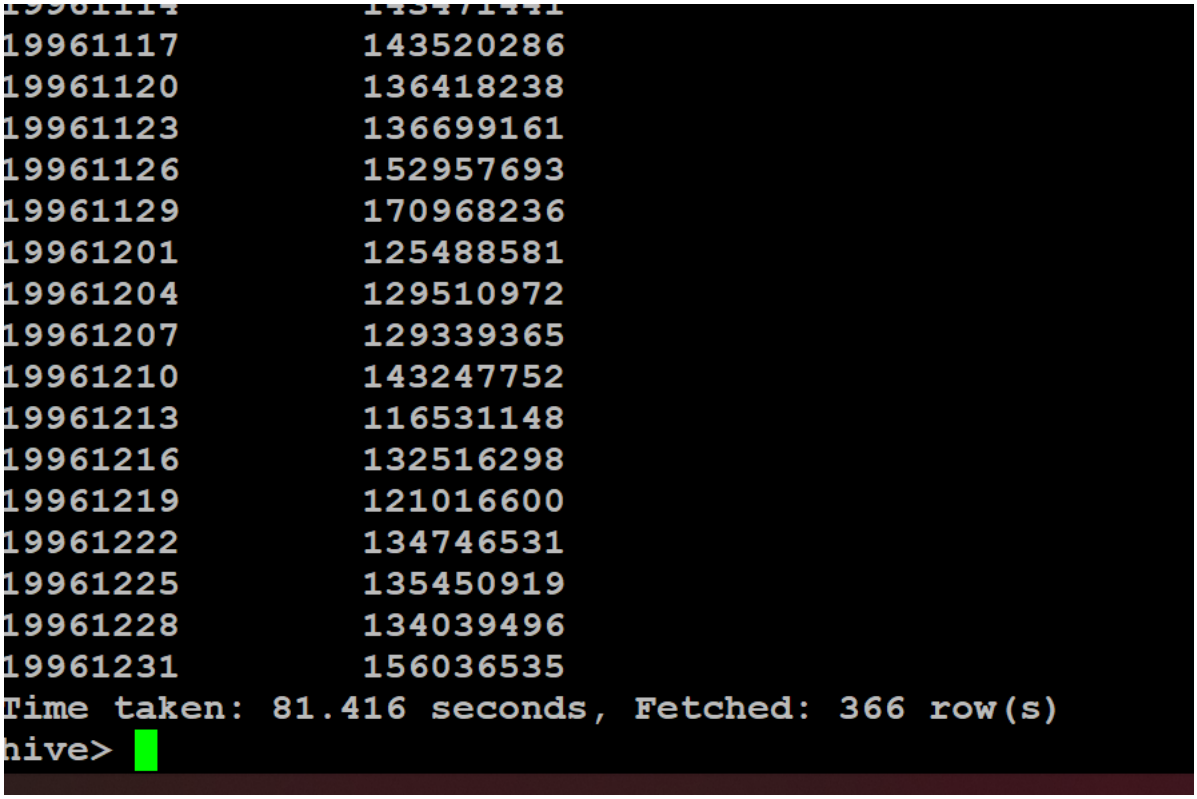
- ⇨ As we can see, the two methods' timelines differ significantly.
- ⇨ The time was 55 seconds in the multi-node cluster and 1 minute, 21 seconds in the single-node.
- ⇨ There is a difference of 26 seconds in both the methods.
- ⇨ At least one worker node is required in multi-node clusters.
- ⇨ Each task is completed by a worker node in a cluster, and the outcomes are stored in HDFS.
- ⇨ The division of tasks enables time savings at work.

Part 2: Hive

1) Run the following query in Hive and report the time it takes to execute:

```
select lo_orderdate, sum(lo_extendedprice) as revenue
from lineorder, dwdate
where lo_orderdate = d_datekey
  and d_year = 1996
  and lo_discount between 5 and 7
  and lo_quantity < 12
GROUP BY lo_orderdate;
```

```
hive> select * from lineorder limit 5;
OK
1       1       7381    155190  828     19960102        5-LOW   0       17      2116823 17366547        4       2032150 74711   2 1
9960212 TRUCK
1       2       7381    67310   163     19960102        5-LOW   0       36      4598316 17366547        9       4184467 76638   6 1
9960228 MAIL
1       3       7381    63700   71      19960102        5-LOW   0       8       1330960 17366547        10      1197864 99822   2 1
9960305 REG AIR
1       4       7381    2132    943     19960102        5-LOW   0       28      2895564 17366547        9       2634963 62047   6 1
9960330 AIR
1       5       7381    24027   1625    19960102        5-LOW   0       24      2282448 17366547        10      2054203 57061   4 1
9960314 FOB
Time taken: 1.398 seconds, Fetched: 5 row(s)
hive> LOAD DATA LOCAL INPATH '/home/ec2-user/dwdate.tbl'
    > OVERWRITE INTO TABLE dwdate;
Loading data to table default.dwdate
OK
Time taken: 0.362 seconds
hive> select * from dwadate limit 5;
FAILED: SemanticException [Error 10001]: Line 1:14 Table not found 'dwadate'
hive> select * from dwdate limit 5;
OK
19920101        January 1, 1992 Thursday        January 1992    199201  Jan1992 5       1       1       1       1       Winter  0 1
1       1
19920102        January 2, 1992 Friday  January 1992    199201  Jan1992 6       2       2       1       1       Winter  0       1 0
1
19920103        January 3, 1992 Saturday        January 1992    199201  Jan1992 7       3       3       1       1       Winter  1 1
0       0
19920104        January 4, 1992 Sunday  January 1992    199201  Jan1992 1       4       4       1       1       Winter  0       1 0
0
19920105        January 5, 1992 Monday  January 1992    199201  Jan1992 2       5       5       1       1       Winter  0       1 0
1
Time taken: 0.158 seconds, Fetched: 5 row(s)
hive> show tables;
OK
dwdate
lineorder
Time taken: 0.021 seconds, Fetched: 2 row(s)
```



⇨ query time it takes execute : 81.416 seconds

2) Perform the following transform operation using SELECT TRANSFORM on the dwdate table by creating a new table. The new dwdate table will combine d_daynuminweek, d_daynuminmonth, and d_daynuminyear into a single column in the new table using a '&' as the delimiter character. You should also eliminate the following 1 column: d_lastdayinmonthfl. The final table will have fewer columns than the original table because you merge 3 columns into 1 and remove 1 column

⇨

```
#!/usr/bin/python

import sys

for line in sys.stdin:

    line=line.strip()

    v= line.split('\t')

    val7 = str(v[7].split(' '))

    val8= str(v[8].split(' '))

    val9 =str(v[9].split(' '))
```

```
        d =(val7,str(val8),str(val9))
        t= '&'.join(d)
        v[7]=t
        v[8]=v[10]
        v[9]=v[11]
        v[10]=v[12]
        v[11]=v[13]
        v[12]=v[15]
        v[13]=v[16]
        #v[14]=v[16]
        del v[14:]
        #v[12]=v[13]
        #v[13]=v[14]
        print('\t'.join(str(i)for i in v))
```

```
create table New_dwdate (
 d_datekey            int,
 d_date               varchar(19),
 d_dayofweek          varchar(10),
 d_month              varchar(10),
 d_year               int,
 d_yearmonthnum       int,
 d_yearmonth          varchar(8),
 d_daynuminweek       varchar(30),
 d_monthnuminyear     int,
 d_weeknuminyear      int,
```

d_sellingseason varchar(13),

d_lastdayinweekfl varchar(1),

d_holidayfl varchar(1),

d_weekdayfl varchar(1)

)ROW FORMAT DELIMITED FIELDS

TERMINATED BY '\t' STORED AS TEXTFILE;

ADD FILE /home/ec2-user/my_dwdate.py;

INSERT OVERWRITE TABLE New_dwdate

SELECT TRANSFORM (d_datekey,d_date,d_dayofweek,d_month,d_year,d_yearmonthnum,d_yearmonth,d_daynuminweek,d_daynuminmonth,d_daynuminyear,d_monthnuminyear,d_weeknuminyear,d_sellingseason,d_lastdayinweekfl,d_lastdayinmonthfl,d_holidayfl,d_weekdayfl) USING'my_dwdate.py'AS(d_datekey,d_date,d_dayofweek,d_month,d_year,d_yearmonthnum,d_yearmonth,d_daynuminweek,d_monthnuminyear,d_weeknuminyear,d_sellingseason,d_lastdayinweekfl,d_holidayfl,d_weekdayfl) FROM dwdate;

```
Time taken: 0.32 seconds, Fetched: 2 row(s)
hive> ADD FILE /home/ec2-user/my_dwdate.py;
Added resources: [/home/ec2-user/my_dwdate.py]
hive> INSERT OVERWRITE TABLE New_dwdate
    > SELECT TRANSFORM (d_datekey,d_date,d_dayofweek,d_month,d_year,d_yearmonthnum,d_yearmo
nth,d_daynuminweek,d_daynuminmonth,d_daynuminyear,d_monthnuminyear,d_weeknuminyear,d_sellin
gseason,d_lastdayinweekfl,d_lastdayinmonthfl,d_holidayfl,d_weekdayfl) USING'my_dwdate.py'AS
(d_datekey,d_date,d_dayofweek,d_month,d_year,d_yearmonthnum,d_yearmonth,d_daynuminweek,d_mo
nthnuminyear,d_weeknuminyear,d_sellingseason,d_lastdayinweekfl,d_holidayfl,d_weekdayfl) FRO
M dwdate;
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_20221030011225_e3aef9d2-1e3c-4b56-80db-f344e4354666
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_1667085546139_0001, Tracking URL = http://ip-172-31-40-33.us-east-2.comp
ute.internal:8088/proxy/application_1667085546139_0001/
Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job  -kill job_1667085546139_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2022-10-30 01:12:41,903 Stage-1 map = 0%,  reduce = 0%
2022-10-30 01:12:50,684 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.07 sec
MapReduce Total cumulative CPU time: 2 seconds 70 msec
Ended Job = job_1667085546139_0001
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://172.31.40.33/user/hive/warehouse/new_dwdate/.hive-staging_hive_2022-
10-30_01-12-25_888_8368438460942741254-1/-ext-10000
Loading data to table default.new_dwdate
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 2.07 sec   HDFS Read: 240422 HDFS Write: 253049 SUC
CESS
Total MapReduce CPU Time Spent: 2 seconds 70 msec
OK
Time taken: 26.683 seconds
hive> select *from New_dwdate limit 10;
OK
19920101        January 1, 1992 Thursday        January 1992    199201  Jan1992 ['5']&['1']&['1']       1       1       Winter  0       1       1
19920102        January 2, 1992 Friday  January 1992    199201  Jan1992 ['6']&['2']&['2']       1       1       Winter  0       1       1
19920103        January 3, 1992 Saturday        January 1992    199201  Jan1992 ['7']&['3']&['3']       1       1       Winter  1       0       0
19920104        January 4, 1992 Sunday  January 1992    199201  Jan1992 ['1']&['4']&['4']       1       1       Winter  0       0       0
19920105        January 5, 1992 Monday  January 1992    199201  Jan1992 ['2']&['5']&['5']       1       1       Winter  0       0       1
19920106        January 6, 1992 Tuesday January 1992    199201  Jan1992 ['3']&['6']&['6']       1       1       Winter  0       0       1
19920107        January 7, 1992 Wednesday       January 1992    199201  Jan1992 ['4']&['7']&['7']       1       2       Winter  0       0       1
19920108        January 8, 1992 Thursday        January 1992    199201  Jan1992 ['5']&['8']&['8']       1       2       Winter  0       0       1
19920109        January 9, 1992 Friday  January 1992    199201  Jan1992 ['6']&['9']&['9']       1       2       Winter  0       0       1
19920110        January 10, 1992        Saturday        January 1992    199201  Jan1992 ['7']&['10']&['10']     1       2       Winter  1       0       0
Time taken: 0.958 seconds, Fetched: 10 row(s)
```

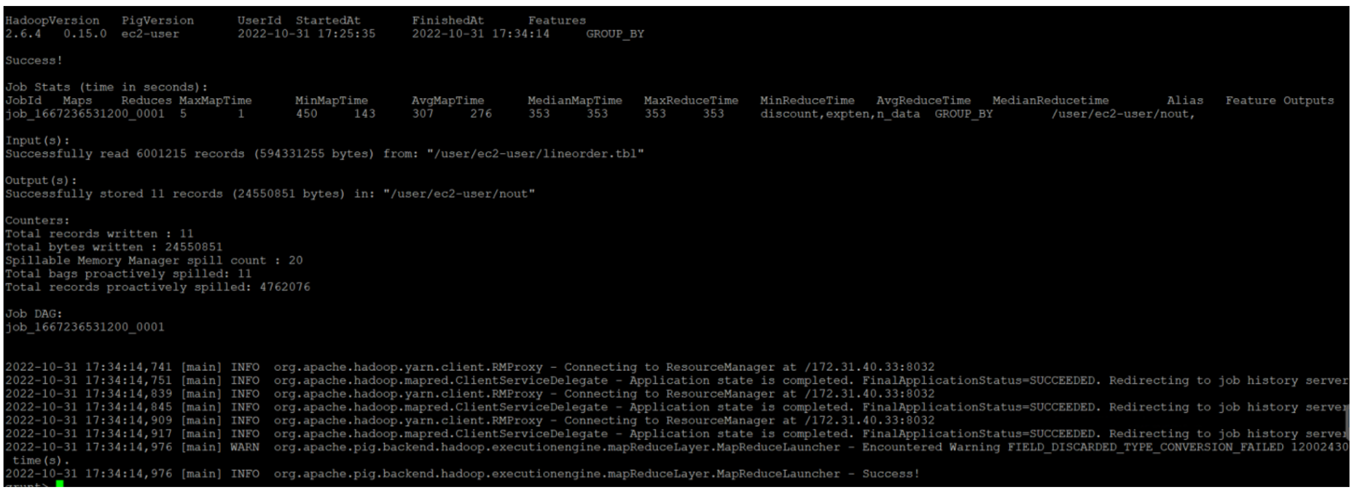
Part 3: Pig

Convert and load the data into Pig, implementing and timing the following queries:

SELECT lo_discount, SUM(lo_extendedprice)
FROM lineorder
GROUP BY lo_discount;

n_data = LOAD '/user/ec2-user/lineorder.tbl' USING PigStorage('|') As (lo_orderkey:int,lo_linenumber:int,lo_custkey:int,lo_partkey:int,lo_suppkey:int,lo_orderdate:int,lo_orderpriority:int,lo_shippriority:int,lo_quantity:int,lo_extendedprice:int,lo_ordertotalprice:int,lo_discount:int,lo_revenue:int,lo_supplycost:int,lo_tax:int,lo_commitdate:int,lo_shipmode:int);

discount = Group n_data By lo_discount;
e= FOREACH discount GENERATE n_data.lo_discount ,SUM(n_data.lo_extendedprice);
dump e;

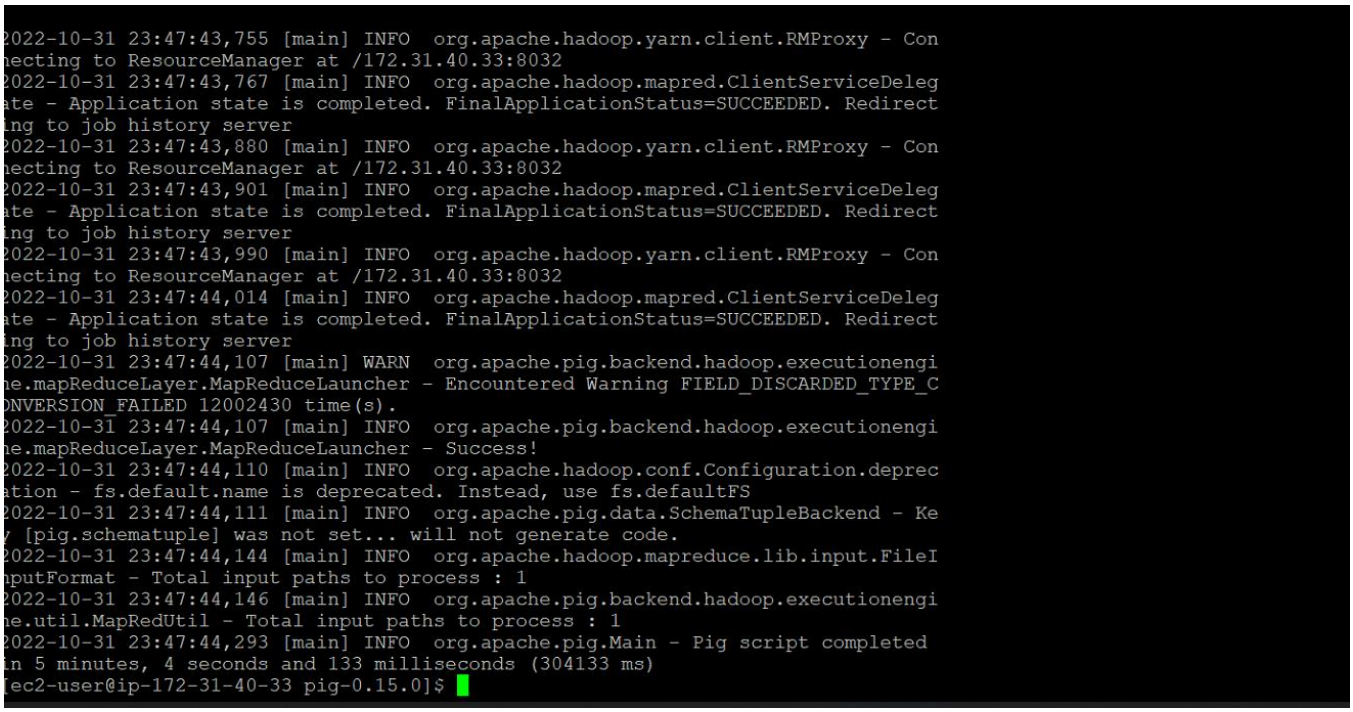


```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
2.6.4   0.15.0  ec2-user        2022-10-31 17:25:35     2022-10-31 17:34:14     GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        Alias   Feature Outputs
job_1667236531200_0001  5       1       450     143     307     276     353     353     353     353     discount,expten,n_data  GROUP_BY        /user/ec2-user/nout,

Input(s):
Successfully read 6001215 records (594331255 bytes) from: "/user/ec2-user/lineorder.tbl"

Output(s):
Successfully stored 11 records (24550851 bytes) in: "/user/ec2-user/nout"

Counters:
Total records written : 11
Total bytes written : 24550851
Spillable Memory Manager spill count : 20
Total bags proactively spilled: 11
Total records proactively spilled: 4762076

Job DAG:
job_1667236531200_0001


2022-10-31 17:34:14,741 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /172.31.40.33:8032
2022-10-31 17:34:14,751 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2022-10-31 17:34:14,839 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /172.31.40.33:8032
2022-10-31 17:34:14,845 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2022-10-31 17:34:14,909 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /172.31.40.33:8032
2022-10-31 17:34:14,917 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStatus=SUCCEEDED. Redirecting to job history server
2022-10-31 17:34:14,976 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning FIELD_DISCARDED_TYPE_CONVERSION_FAILED 12002430
 time(s).
2022-10-31 17:34:14,976 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
```

```
,(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10),(10)},2085020008418)
022-10-31 19:22:12,701 [main] INFO  org.apache.pig.Main - Pig script completed in 5 minutes, 44 seconds and 530 milliseconds (344530 ms)
ec2-user@ip-172-31-40-33 pig-0.15.0]$
```

⇨ 5mim 44sec 530milliseconds

SELECT lo_quantity, AVG(lo_revenue)
FROM lineorder
WHERE lo_discount > 11 AND lo_quantity < 22
GROUP BY lo_quantity;



One easy way to time Pig is as follows: put your sequence of pig commands, including LOAD, into a text file and then run, from command line in pig directory (e.g., [ec2-user@ip-172-31-6-39 pig-0.15.0]$), **bin/pig -f pig_script.pig** (which will report how long the pig script took to run).
Store the results of the 2nd Pig query into HDFS and report the size of the output.

⇨ Store will be Zero

Part 4: Hadoop Streaming

Implement, run and time the following query using Hadoop streaming with python.

```
SELECT lo_quantity, MAX(lo_revenue)
FROM (SELECT lo_revenue, MIN(lo_quantity) as lo_quantity,
                         MIN(lo_discount) as lo_discount
      FROM lineorder
      WHERE lo_orderpriority LIKE '%URGENT'
```

```
        GROUP BY lo_revenue)
    WHERE lo_discount BETWEEN 6 AND 9
    GROUP BY lo_quantity;
```

This requires running two different map reduce jobs. First, you would write a job that executes the subquery and produces an output in HDFS. Then you would write a second job that uses output of the first job as the input.

Don’t forget to submit your python code, and the command line you used to run Hadoop streaming jobs.

NOTE: You may implement this part in Java if you prefer.

Inter quarry

```
(8, 81360, 0)
(8, 81360, 0)
(14, 81360, 0)
(42, 81360, 0)
(9, 81360, 0)
(11, 81360, 0)
(12, 81360, 0)
(44, 81360, 0)
(45, 81360, 0)
(1, 81360, 0)
(31, 81360, 0)
(8, 81360, 0)
(1, 81360, 0)
(8, 81360, 0)
(26, 81360, 0)
(40, 81360, 0)
(11, 81360, 0)
(21, 81360, 0)
(31, 81360, 0)
(29, 81360, 0)
(36, 81360, 0)
(21, 81360, 0)
(45, 81360, 0)
(1, 81360, 0)
(21, 81360, 0)
(19, 81360, 0)
(15, 81360, 0)
(1, 81360, 0)
(15, 81360, 0)
(31, 81360, 0)
(14, 81360, 0)
(10, 81360, 0)
(42, 81360, 0)
(36, 81360, 0)
(31, 81360, 0)
(6, 81360, 0)
(20, 81360, 0)
(45, 81360, 0)
(21, 81360, 0)
(28, 81360, 0)
(29, 81360, 0)
(3, 81360, 0)
(49, 81360, 0)
(10, 81360, 0)
(44, 81360, 0)
(1, 81360, 0)
(36, 81360, 0)
(36, 81360, 0)
^CTraceback (most recent call last):
  File "reducer1.py", line 23, in <module>
    print(rev_lst[i], min(qua_lst), min(dis_lst))
KeyboardInterrupt
[ec2-user@ip-172-31-40-33 ~]$ nano reducer1.py
```

Mapper1.py

```
#!/usr/bin/python
import sys
for line in sys.stdin:
        line = line.strip()
        val = line.split("|")
        lo_quantity = val[8]
        lo_revenue = val[12]
        lo_discount = val[11]
        if val[6].endswith("URGENT"):
                print  (lo_quantity + '|' + lo_revenue + '|' +  lo_discount)
```

Reducer

```
#!/usr/bin/python
import sys
curr_id = None
id = None
dis = None
rev = None
qua = None
qua_lst = []
dis_lst = []
rev_lst =[]
#min_qua = 0 min_dis = 0
for line in sys.stdin:
        line = line.strip()
```

```
        ln = line.split('|')
        id = int(ln[0])
        qua  = int(ln[1])
        dis = int(ln[2])
        #qua = int(ln[8])
        qua_lst.append(qua)
        dis_lst.append(dis)
        rev_lst.append(id)
for i in rev_lst:
        print(rev_lst[i], min(qua_lst), min(dis_lst))
```

- ⇨ outer query

```
13      9855853
7       9855853
12      9855853
26      9855853
38      9855853
17      9855853
27      9855853
46      9855853
39      9855853
37      9855853
25      9855853
12      9855853
20      9855853
28      9855853
7       9855853
26      9855853
40      9855853
34      9855853
25      9855853
36      9855853
37      9855853
12      9855853
12      9855853
15      9855853
9       9855853
41      9855853
25      9855853
^X7     9855853
34      9855853
17      9855853
41      9855853
34      9855853
50      9855853
31      9855853
15      9855853
17      9855853
37      9855853
21      9855853
41      9855853
21      9855853
40      9855853
40      9855853
32      9855853
42      9855853
12      9855853
13      9855853
40      9855853
12      9855853
32      9855853
26      9855853
26      9855853
```

Mapper2

```
#!/usr/bin/python

import sys
```

```
for line in sys.stdin:
        ln = line.strip()
        vals = ln.split("|")
        lo_quantity = vals[8]
        lo_discount = vals[11]
        lo_revenue = vals[12]
        if 6 <= int(lo_discount) <=9:
                print  (lo_quantity + '|' + lo_revenue)
```

reducer2

```
#!/usr/bin/python
import sys
id = None
dis = None
qua = None
rev_all = []
qua_all = []
for line in sys.stdin:
        line = line.strip()
        ln = line.split('|')
        rev =int(ln[1])
        qua = int(ln[0])
        rev_all.append(rev)
        qua_all.append(qua)
for i in qua_all:
```

```
    print '%d\t%d' % (qua_all[i], max(rev_all))
```