**CSC 555: Mining Big Data**

Project, Phase 1 (due Wednesday, February 20th)

In this part of the project (which will also serve as our take-home midterm), you will 1) Set up a 4-node cluster and 2) perform data warehousing and transformation queries using Hive, Pig and Hadoop streaming. The modified Hive-style schema is at:

http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM\_schema\_hive.sql

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 (not csv).

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/dwdate.tbl>

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/lineorder.tbl>

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/part.tbl>

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/supplier.tbl>

<http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/customer.tbl>

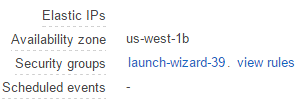
Please be sure to submit all code (pig, python and SQL).

# Part 1: Multi-node cluster

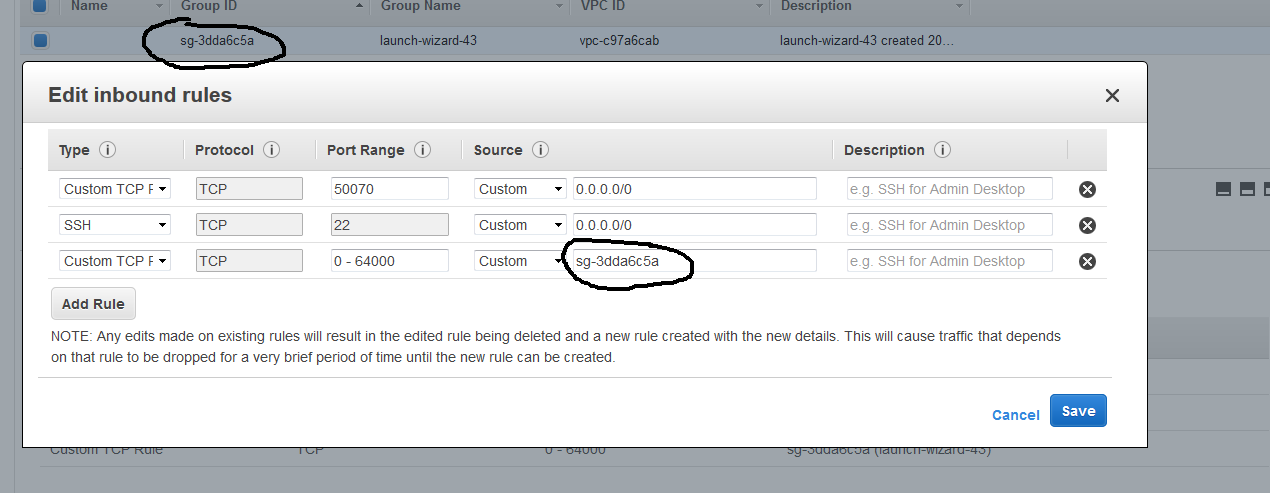
1. Your first step is to setup a multi-node cluster and re-run a simple wordcount. For this part, you will create a 4-node cluster (with a total of 1 master + 3 worker nodes). Include your master node in the “slaves” file, to make sure all 4 nodes are working.

You need to perform the following steps:

1. Create a new node of a medium size (you can always switch the size of the node). It is possible, but I do not recommend trying to reconfigure your existing Hadoop into this new cluster (it is much easier to make 4 new nodes for a total of 5 in your AWS account).
   1. **When creating a node I recommend changing the default 8G hard drive to 30G on all nodes.**
   2. 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.

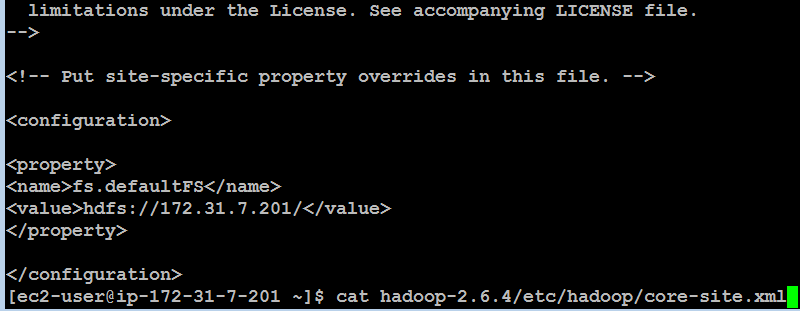
In order to make changes, you need to do the following. Access the cluster security group (launch-wizard-xx). Right click on the security group and choose Edit inbound rules

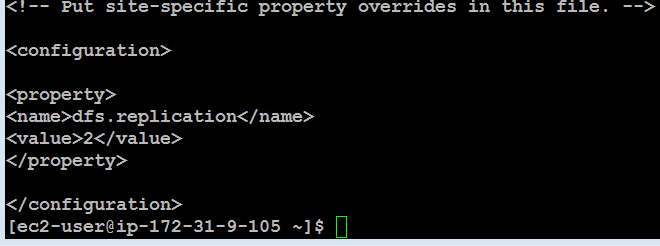
Note that the first line below is opening port 50070. The second line below is the default (port 22 is required for regular SSH connections). The third line opens all ports but ONLY for the same security group (assuming that all of your nodes in the cluster share the same security group – that will happen automatically if you use the “create more like this” option when creating instances as specified in part 1-c below). We had some issues with machines being hacked and disabled without that last limitation, so make sure you include it.



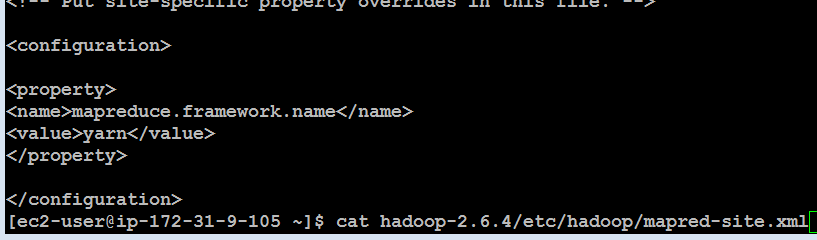
* 1. Right click on the Master node and choose “create more like this” to create 3 more nodes with same settings. If you configure the network settings on master first, security group information will be copied.

NOTE: Hard drive size will not be copied and default to 8G unless you change it.

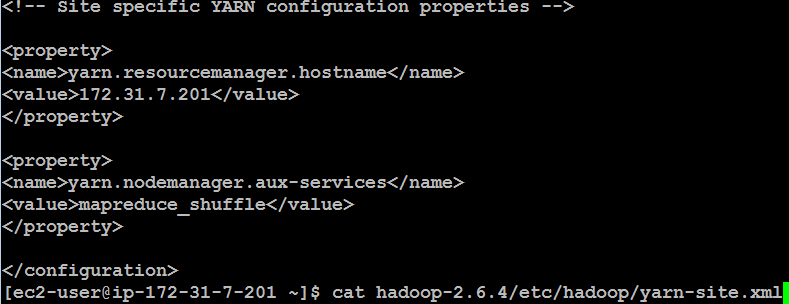
1. Connect to the master and set up Hadoop similarly to what you did previously. Do not attempt to repeat these steps on workers yet – you will only need to set up Hadoop once.
   1. Configure core-site.xml, adding the **PrivateIP** (do not use public IP) of the master. 
   2. Configure hdfs-site and set replication factor to 2.



* 1. 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



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



Finally, edit the slaves file and list your 4 nodes (master and 3 workers) using Private IPs

[ec2-user@ip-172-31-7-201 ~]$ cat hadoop-2.6.4/etc/hadoop/slaves

172.31.7.201

172.31.5.246

…

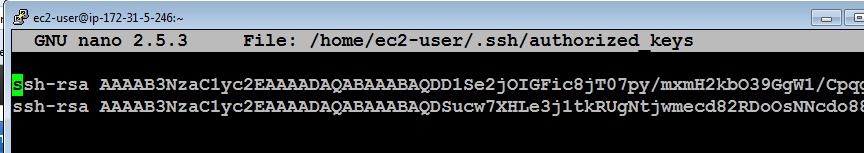
Make sure that you use private IP (private DNS is also ok) for your configuration files (such as conf/masters and conf/slaves or the other 3 config files). The advantage of the Private IP is that it does not change after your instance is stopped (if you use the Public IP, the cluster would need to be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

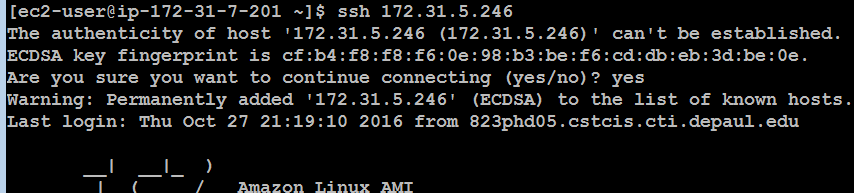
Note that the example below is NOT the master, but one of the workers (ip-172-31-5-246). The first public key is the .pem Amazon half and the 2nd public key is the master’s public key copied in as one line.



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)

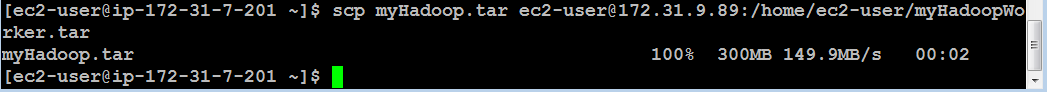
tar cvf myHadoop.tar hadoop-2.6.4

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 non-master node and unpack the .tar file.

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

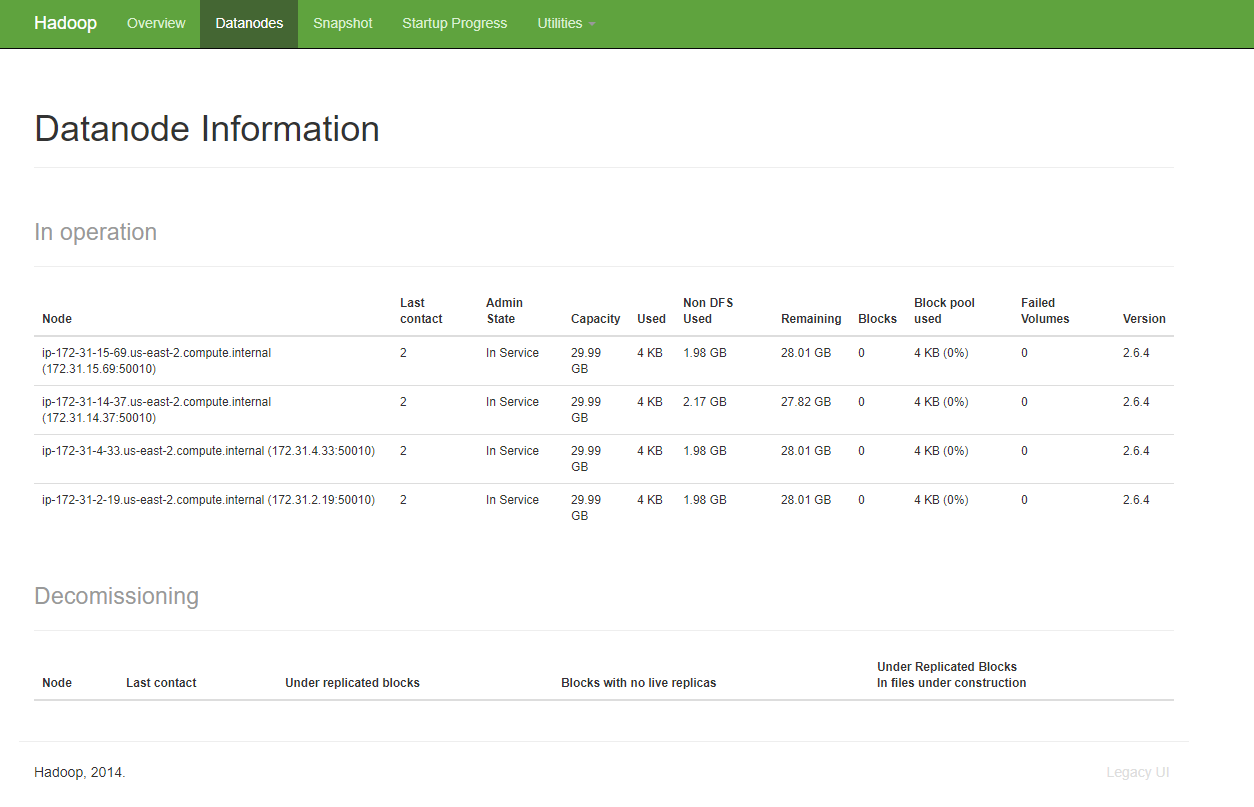
Once you have successfully completed the previous steps, you should can start and use your new cluster by going to the master node and running the start-dfs.sh and start-yarn.sh scripts (you do not need to explicitly start anything on worker nodes – the master will do that for you).

You should verify that the cluster is running by pointing your browser to the link below.

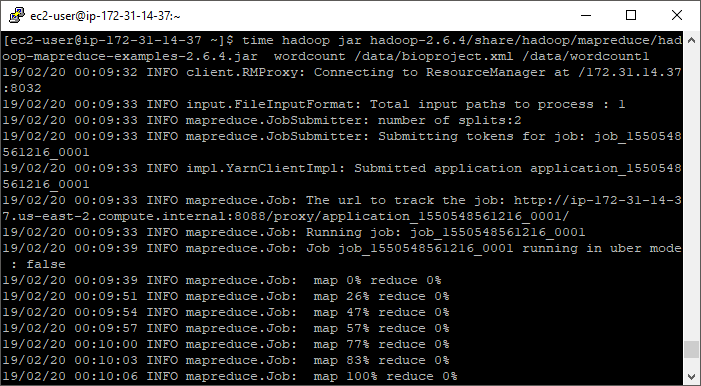
http://[insert-the-public-ip-of-master]:50070/

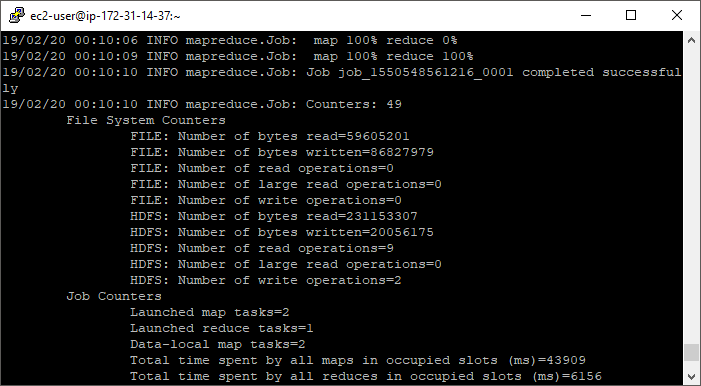
Make sure that the cluster is operational (you can see the 4 nodes under Datanodes tab).

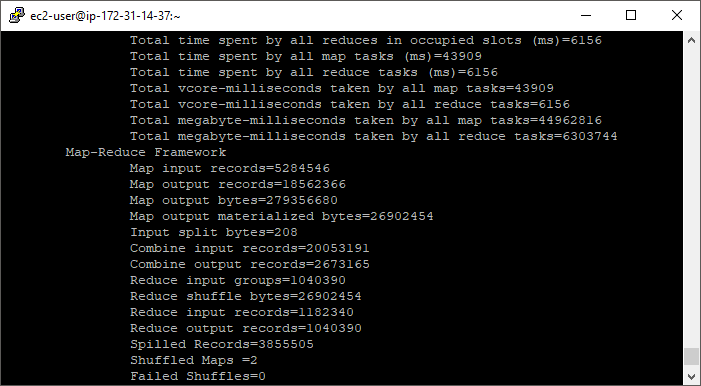
Submit a screenshot of your cluster status view.

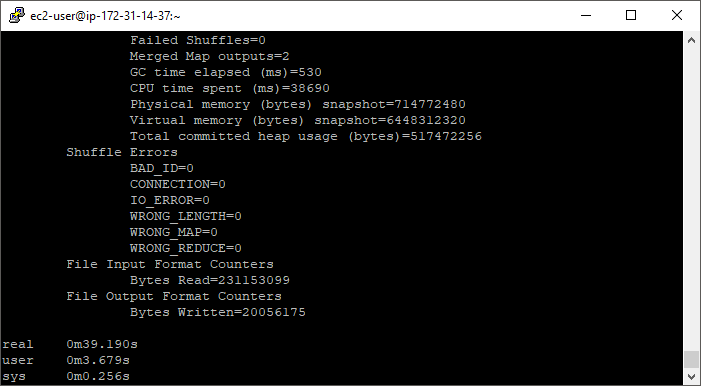


Repeat the steps for wordcount using bioproject.xml from Assignment 1 and submit screenshots of running it.









[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -du /data/wordcount1/

0 /data/wordcount1/\_SUCCESS

20056175 /data/wordcount1/part-r-00000

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -cat /data/wordcount1/part-r-00000 | grep subarctic

subarctic 21

Submit a short paragraph with a discussion about how the results compare (faster? slower? How much faster/slower? Due to what?)

The single-node Hadoop instance from assignment 1 ran the word count job in 1 minute 10.229 seconds. The four-node instance ran the job 39.190 seconds which is roughly twice as fast. The speed increase is due to the additional nodes, however, I would have expected a more than doubling of speed, based just on the number of nodes. It’s likely that network speed and block distribution have impacted speed.

Running the following command shows that the file was only split into two blocks and where they are located.

hdfs fsck /data/bioproject.xml -files -blocks -locations

Connecting to namenode via http://ip-172-31-14-37.us-east-2.compute.internal:50070

FSCK started by ec2-user (auth:SIMPLE) from /172.31.14.37 for path /data/bioproject.xml at Wed Feb 20 00:30:53 UTC 2019

/data/bioproject.xml 231149003 bytes, 2 block(s): OK

0. BP-806142183-172.31.14.37-1550548147742:blk\_1073741825\_1001 len=134217728 repl=2 [172.31.14.37:50010, 172.31.4.33:50010]

1. BP-806142183-172.31.14.37-1550548147742:blk\_1073741826\_1002 len=96931275 repl=2 [172.31.14.37:50010, 172.31.4.33:50010]

Status: HEALTHY

Total size: 231149003 B

Total dirs: 0

Total files: 1

Total symlinks: 0

**Total blocks (validated): 2 (avg. block size 115574501 B)**

Minimally replicated blocks: 2 (100.0 %)

Over-replicated blocks: 0 (0.0 %)

Under-replicated blocks: 0 (0.0 %)

Mis-replicated blocks: 0 (0.0 %)

Default replication factor: 2

Average block replication: 2.0

Corrupt blocks: 0

Missing replicas: 0 (0.0 %)

Number of data-nodes: 4

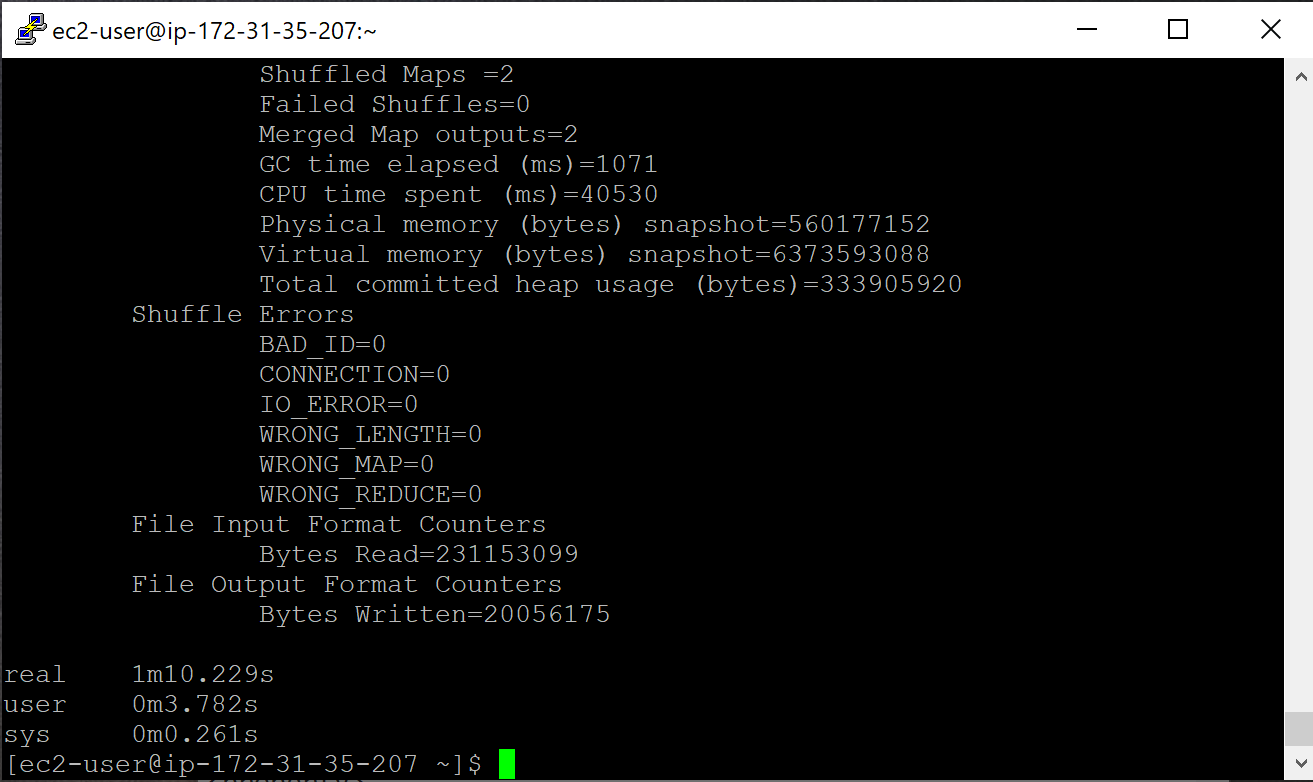
Number of racks: 1

FSCK ended at Wed Feb 20 00:30:53 UTC 2019 in 1 milliseconds

The filesystem under path '/data/bioproject.xml' is HEALTHY

This would explain the half-time only gain in speed.

Here are the times from assignment 1:



# Part 2: Hive

Run the following three (1.2, 1.3 and 2.1) queries in Hive and record the time they take to execute: <http://rasinsrv07.cstcis.cti.depaul.edu/CSC555/SSBM1/SSBM_queries.sql>

Create and load tables:

CREATE TABLE lineorder (

lo\_orderkey INT,

lo\_linenumber INT,

lo\_custkey INT,

lo\_partkey INT,

lo\_suppkey INT,

lo\_orderdate INT,

lo\_orderpriority VARCHAR(15),

lo\_shippriority VARCHAR(1),

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 VARCHAR(10)

)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/lineorder.tbl' OVERWRITE INTO TABLE lineorder;

CREATE TABLE 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 INT,

d\_daynuminmonth INT,

d\_daynuminyear INT,

d\_monthnuminyear INT,

d\_weeknuminyear INT,

d\_sellingseason VARCHAR(13),

d\_lastdayinweekfl VARCHAR(1),

d\_lastdayinmonthfl VARCHAR(1),

d\_holidayfl VARCHAR(1),

d\_weekdayfl VARCHAR(1)

)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/dwdate.tbl' OVERWRITE INTO TABLE dwdate;

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;

LOAD DATA LOCAL INPATH '/home/ec2-user/part.tbl' OVERWRITE INTO TABLE part;

CREATE TABLE supplier (

s\_suppkey INT,

s\_name VARCHAR(25),

s\_address VARCHAR(25),

s\_city VARCHAR(10),

s\_nation VARCHAR(15),

s\_region VARCHAR(12),

s\_phone VARCHAR(15)

)ROW FORMAT DELIMITED FIELDS

TERMINATED BY '|' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/ec2-user/supplier.tbl' OVERWRITE INTO TABLE supplier;

**1.2 Time to execute = 29.337 seconds**

hive> select sum(lo\_extendedprice) as revenue

> from lineorder, dwdate

> where lo\_orderdate = d\_datekey

> and d\_yearmonth = 'Jan1993'

> and lo\_discount between 5 and 6

> and lo\_quantity between 25 and 35;

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\_20190220020132\_4923f23d-e5cb-45d3-82e0-4a510a01bcb9

Total jobs = 1

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/ec2-user/apache-hive-2.0.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/home/ec2-user/hadoop-2.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Execution log at: /tmp/ec2-user/ec2-user\_20190220020132\_4923f23d-e5cb-45d3-82e0-4a510a01bcb9.log

2019-02-20 02:01:37 Starting to launch local task to process map join; maximum memory = 477626368

2019-02-20 02:01:39 Dump the side-table for tag: 1 with group count: 31 into file: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-01-32\_259\_3060710722325099777-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile01--.hashtable

2019-02-20 02:01:39 Uploaded 1 File to: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-01-32\_259\_3060710722325099777-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile01--.hashtable (945 bytes)

2019-02-20 02:01:39 End of local task; Time Taken: 1.174 sec.

Execution completed successfully

MapredLocal task succeeded

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\_1550548561216\_0002, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0002/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0002

Hadoop job information for Stage-2: number of mappers: 3; number of reducers: 1

2019-02-20 02:01:45,505 Stage-2 map = 0%, reduce = 0%

2019-02-20 02:01:54,167 Stage-2 map = 67%, reduce = 0%, Cumulative CPU 8.12 sec

2019-02-20 02:01:56,269 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 13.35 sec

2019-02-20 02:01:59,480 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 14.74 sec

MapReduce Total cumulative CPU time: 14 seconds 740 msec

Ended Job = job\_1550548561216\_0002

MapReduce Jobs Launched:

Stage-Stage-2: Map: 3 Reduce: 1 Cumulative CPU: 14.74 sec HDFS Read: 594366950 HDFS Write: 12 SUCCESS

Total MapReduce CPU Time Spent: 14 seconds 740 msec

OK

14215822897

Time taken: 29.337 seconds, Fetched: 1 row(s)

**1.3 Time to execute = 26.664 seconds**

hive> select sum(lo\_extendedprice) as revenue

> from lineorder, dwdate

> where lo\_orderdate = d\_datekey

> and d\_weeknuminyear = 6 and d\_year = 1994

> and lo\_discount between 5 and 8

> and lo\_quantity between 36 and 41;

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\_20190220020641\_17598f2d-e1e8-4c46-b1b9-257308183cf5

Total jobs = 1

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/ec2-user/apache-hive-2.0.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/home/ec2-user/hadoop-2.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Execution log at: /tmp/ec2-user/ec2-user\_20190220020641\_17598f2d-e1e8-4c46-b1b9-257308183cf5.log

2019-02-20 02:06:46 Starting to launch local task to process map join; maximum memory = 477626368

2019-02-20 02:06:47 Dump the side-table for tag: 1 with group count: 7 into file: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-06-41\_907\_505512871082648851-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile11--.hashtable

2019-02-20 02:06:47 Uploaded 1 File to: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-06-41\_907\_505512871082648851-1/-local-10005/HashTable-Stage-2/MapJoin-mapfile11--.hashtable (414 bytes)

2019-02-20 02:06:47 End of local task; Time Taken: 1.124 sec.

Execution completed successfully

MapredLocal task succeeded

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\_1550548561216\_0003, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0003/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0003

Hadoop job information for Stage-2: number of mappers: 3; number of reducers: 1

2019-02-20 02:06:52,992 Stage-2 map = 0%, reduce = 0%

2019-02-20 02:07:01,340 Stage-2 map = 33%, reduce = 0%, Cumulative CPU 5.46 sec

2019-02-20 02:07:02,373 Stage-2 map = 67%, reduce = 0%, Cumulative CPU 8.67 sec

2019-02-20 02:07:04,438 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 14.48 sec

2019-02-20 02:07:07,516 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 15.95 sec

MapReduce Total cumulative CPU time: 15 seconds 950 msec

Ended Job = job\_1550548561216\_0003

MapReduce Jobs Launched:

Stage-Stage-2: Map: 3 Reduce: 1 Cumulative CPU: 15.95 sec HDFS Read: 594367056 HDFS Write: 11 SUCCESS

Total MapReduce CPU Time Spent: 15 seconds 950 msec

OK

4435791464

Time taken: 26.664 seconds, Fetched: 1 row(s)

**2.1 Time to execute = 95.876 seconds**

hive> select sum(lo\_revenue), d\_year, p\_brand1

> from lineorder, dwdate, part, supplier

> where lo\_orderdate = d\_datekey

> and lo\_partkey = p\_partkey

> and lo\_suppkey = s\_suppkey

> and p\_category = 'MFGR#12'

> and s\_region = 'AMERICA'

> group by d\_year, p\_brand1

> order by d\_year, p\_brand1;

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\_20190220023831\_8e9649ac-0603-44d5-90b3-d56361ef1153

Total jobs = 6

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/ec2-user/apache-hive-2.0.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/home/ec2-user/hadoop-2.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Execution log at: /tmp/ec2-user/ec2-user\_20190220023831\_8e9649ac-0603-44d5-90b3-d56361ef1153.log

2019-02-20 02:38:36 Starting to launch local task to process map join; maximum memory = 477626368

2019-02-20 02:38:37 Dump the side-table for tag: 1 with group count: 2556 into file: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10014/HashTable-Stage-13/MapJoin-mapfile51--.hashtable

2019-02-20 02:38:37 Uploaded 1 File to: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10014/HashTable-Stage-13/MapJoin-mapfile51--.hashtable (67039 bytes)

2019-02-20 02:38:37 End of local task; Time Taken: 1.24 sec.

Execution completed successfully

MapredLocal task succeeded

Launching Job 1 out of 6

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_1550548561216\_0004, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0004/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0004

Hadoop job information for Stage-13: number of mappers: 3; number of reducers: 0

2019-02-20 02:38:43,105 Stage-13 map = 0%, reduce = 0%

2019-02-20 02:38:50,331 Stage-13 map = 33%, reduce = 0%, Cumulative CPU 4.96 sec

2019-02-20 02:38:53,421 Stage-13 map = 67%, reduce = 0%, Cumulative CPU 20.54 sec

2019-02-20 02:38:57,529 Stage-13 map = 100%, reduce = 0%, Cumulative CPU 29.47 sec

MapReduce Total cumulative CPU time: 29 seconds 470 msec

Ended Job = job\_1550548561216\_0004

Stage-15 is selected by condition resolver.

Stage-16 is filtered out by condition resolver.

Stage-2 is filtered out by condition resolver.

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/ec2-user/apache-hive-2.0.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/home/ec2-user/hadoop-2.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Execution log at: /tmp/ec2-user/ec2-user\_20190220023831\_8e9649ac-0603-44d5-90b3-d56361ef1153.log

2019-02-20 02:39:02 Starting to launch local task to process map join; maximum memory = 477626368

2019-02-20 02:39:04 Dump the side-table for tag: 1 with group count: 7883 into file: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10010/HashTable-Stage-10/MapJoin-mapfile31--.hashtable

2019-02-20 02:39:04 Uploaded 1 File to: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10010/HashTable-Stage-10/MapJoin-mapfile31--.hashtable (249337 bytes)

2019-02-20 02:39:04 End of local task; Time Taken: 1.581 sec.

Execution completed successfully

MapredLocal task succeeded

Launching Job 3 out of 6

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_1550548561216\_0005, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0005/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0005

Hadoop job information for Stage-10: number of mappers: 2; number of reducers: 0

2019-02-20 02:39:09,882 Stage-10 map = 0%, reduce = 0%

2019-02-20 02:39:19,133 Stage-10 map = 50%, reduce = 0%, Cumulative CPU 3.98 sec

2019-02-20 02:39:24,267 Stage-10 map = 75%, reduce = 0%, Cumulative CPU 13.12 sec

2019-02-20 02:39:29,394 Stage-10 map = 100%, reduce = 0%, Cumulative CPU 18.06 sec

MapReduce Total cumulative CPU time: 18 seconds 60 msec

Ended Job = job\_1550548561216\_0005

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/home/ec2-user/apache-hive-2.0.1-bin/lib/log4j-slf4j-impl-2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/home/ec2-user/hadoop-2.6.4/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Execution log at: /tmp/ec2-user/ec2-user\_20190220023831\_8e9649ac-0603-44d5-90b3-d56361ef1153.log

2019-02-20 02:39:34 Starting to launch local task to process map join; maximum memory = 477626368

2019-02-20 02:39:35 Dump the side-table for tag: 1 with group count: 378 into file: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile21--.hashtable

2019-02-20 02:39:35 Uploaded 1 File to: file:/tmp/ec2-user/11570b8a-22b0-40e2-a5d5-b2bc82e91a84/hive\_2019-02-20\_02-38-31\_761\_2546730059735407168-1/-local-10008/HashTable-Stage-4/MapJoin-mapfile21--.hashtable (7792 bytes)

2019-02-20 02:39:35 End of local task; Time Taken: 1.144 sec.

Execution completed successfully

MapredLocal task succeeded

Launching Job 4 out of 6

Number of reduce tasks not specified. Estimated from input data size: 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\_1550548561216\_0006, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0006/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0006

Hadoop job information for Stage-4: number of mappers: 1; number of reducers: 1

2019-02-20 02:39:41,250 Stage-4 map = 0%, reduce = 0%

2019-02-20 02:39:46,390 Stage-4 map = 100%, reduce = 0%, Cumulative CPU 3.1 sec

2019-02-20 02:39:51,510 Stage-4 map = 100%, reduce = 100%, Cumulative CPU 4.47 sec

MapReduce Total cumulative CPU time: 4 seconds 470 msec

Ended Job = job\_1550548561216\_0006

Launching Job 5 out of 6

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\_1550548561216\_0007, Tracking URL = http://ip-172-31-14-37.us-east-2.compute.internal:8088/proxy/application\_1550548561216\_0007/

Kill Command = /home/ec2-user/hadoop-2.6.4/bin/hadoop job -kill job\_1550548561216\_0007

Hadoop job information for Stage-5: number of mappers: 1; number of reducers: 1

2019-02-20 02:39:57,270 Stage-5 map = 0%, reduce = 0%

2019-02-20 02:40:01,460 Stage-5 map = 100%, reduce = 0%, Cumulative CPU 0.91 sec

2019-02-20 02:40:06,582 Stage-5 map = 100%, reduce = 100%, Cumulative CPU 2.22 sec

MapReduce Total cumulative CPU time: 2 seconds 220 msec

Ended Job = job\_1550548561216\_0007

MapReduce Jobs Launched:

Stage-Stage-13: Map: 3 Cumulative CPU: 29.47 sec HDFS Read: 594357772 HDFS Write: 184733291 SUCCESS

Stage-Stage-10: Map: 2 Cumulative CPU: 18.06 sec HDFS Read: 184744581 HDFS Write: 8750831 SUCCESS

Stage-Stage-4: Map: 1 Reduce: 1 Cumulative CPU: 4.47 sec HDFS Read: 8763049 HDFS Write: 9913 SUCCESS

Stage-Stage-5: Map: 1 Reduce: 1 Cumulative CPU: 2.22 sec HDFS Read: 15499 HDFS Write: 6937 SUCCESS

Total MapReduce CPU Time Spent: 54 seconds 220 msec

OK

567838207 1992 MFGR#121

610663790 1992 MFGR#1210

550769662 1992 MFGR#1211

649205856 1992 MFGR#1212

624031241 1992 MFGR#1213

670488468 1992 MFGR#1214

633152470 1992 MFGR#1215

674846781 1992 MFGR#1216

675093435 1992 MFGR#1217

600202070 1992 MFGR#1218

538043594 1992 MFGR#1219

655326672 1992 MFGR#122

540262882 1992 MFGR#1220

556120633 1992 MFGR#1221

590762777 1992 MFGR#1222

535448651 1992 MFGR#1223

703752611 1992 MFGR#1224

570832868 1992 MFGR#1225

614061593 1992 MFGR#1226

581759388 1992 MFGR#1227

644642592 1992 MFGR#1228

640858430 1992 MFGR#1229

789755835 1992 MFGR#123

468535087 1992 MFGR#1230

592436656 1992 MFGR#1231

664275152 1992 MFGR#1232

613885100 1992 MFGR#1233

667399281 1992 MFGR#1234

640290070 1992 MFGR#1235

501892561 1992 MFGR#1236

591481503 1992 MFGR#1237

477423770 1992 MFGR#1238

638259374 1992 MFGR#1239

572354196 1992 MFGR#124

740479248 1992 MFGR#1240

478777095 1992 MFGR#125

592174616 1992 MFGR#126

706151632 1992 MFGR#127

542306646 1992 MFGR#128

581987352 1992 MFGR#129

823087702 1993 MFGR#121

648160706 1993 MFGR#1210

634743898 1993 MFGR#1211

785639283 1993 MFGR#1212

638255029 1993 MFGR#1213

616837237 1993 MFGR#1214

634687975 1993 MFGR#1215

638353900 1993 MFGR#1216

663372951 1993 MFGR#1217

683985855 1993 MFGR#1218

646950033 1993 MFGR#1219

622532984 1993 MFGR#122

530830127 1993 MFGR#1220

543346337 1993 MFGR#1221

756921203 1993 MFGR#1222

533544350 1993 MFGR#1223

915916085 1993 MFGR#1224

473007381 1993 MFGR#1225

739036124 1993 MFGR#1226

592178887 1993 MFGR#1227

583507058 1993 MFGR#1228

617453491 1993 MFGR#1229

637863868 1993 MFGR#123

625534310 1993 MFGR#1230

580327635 1993 MFGR#1231

697373098 1993 MFGR#1232

515571416 1993 MFGR#1233

651935758 1993 MFGR#1234

575779480 1993 MFGR#1235

591878667 1993 MFGR#1236

609618576 1993 MFGR#1237

444614010 1993 MFGR#1238

595256327 1993 MFGR#1239

660586237 1993 MFGR#124

788730059 1993 MFGR#1240

616224539 1993 MFGR#125

617126754 1993 MFGR#126

654438324 1993 MFGR#127

731657001 1993 MFGR#128

548048395 1993 MFGR#129

564405648 1994 MFGR#121

645404849 1994 MFGR#1210

631620635 1994 MFGR#1211

568332348 1994 MFGR#1212

678785857 1994 MFGR#1213

534002330 1994 MFGR#1214

654400242 1994 MFGR#1215

558646341 1994 MFGR#1216

687845641 1994 MFGR#1217

546674347 1994 MFGR#1218

567272942 1994 MFGR#1219

659884062 1994 MFGR#122

562582172 1994 MFGR#1220

598618997 1994 MFGR#1221

601016441 1994 MFGR#1222

555134404 1994 MFGR#1223

737422302 1994 MFGR#1224

570745955 1994 MFGR#1225

746302245 1994 MFGR#1226

651707481 1994 MFGR#1227

573693547 1994 MFGR#1228

647918373 1994 MFGR#1229

580449592 1994 MFGR#123

493270412 1994 MFGR#1230

603546148 1994 MFGR#1231

719865331 1994 MFGR#1232

638982238 1994 MFGR#1233

743247677 1994 MFGR#1234

598680959 1994 MFGR#1235

615726097 1994 MFGR#1236

542569815 1994 MFGR#1237

573510781 1994 MFGR#1238

579855853 1994 MFGR#1239

684573322 1994 MFGR#124

873735737 1994 MFGR#1240

560488304 1994 MFGR#125

657036514 1994 MFGR#126

622571183 1994 MFGR#127

586845664 1994 MFGR#128

534541525 1994 MFGR#129

706469511 1995 MFGR#121

602892803 1995 MFGR#1210

645166092 1995 MFGR#1211

613289283 1995 MFGR#1212

599586479 1995 MFGR#1213

562570804 1995 MFGR#1214

672528755 1995 MFGR#1215

669000972 1995 MFGR#1216

725362449 1995 MFGR#1217

657026635 1995 MFGR#1218

519659003 1995 MFGR#1219

724727741 1995 MFGR#122

517956131 1995 MFGR#1220

635741351 1995 MFGR#1221

564368410 1995 MFGR#1222

600665149 1995 MFGR#1223

762700351 1995 MFGR#1224

671669586 1995 MFGR#1225

572568748 1995 MFGR#1226

530361300 1995 MFGR#1227

633357085 1995 MFGR#1228

547960244 1995 MFGR#1229

660711077 1995 MFGR#123

602735858 1995 MFGR#1230

499852146 1995 MFGR#1231

715300753 1995 MFGR#1232

557149571 1995 MFGR#1233

710023059 1995 MFGR#1234

622425239 1995 MFGR#1235

634565501 1995 MFGR#1236

572847270 1995 MFGR#1237

549318912 1995 MFGR#1238

593851712 1995 MFGR#1239

585421815 1995 MFGR#124

707207888 1995 MFGR#1240

538246872 1995 MFGR#125

605799021 1995 MFGR#126

665978112 1995 MFGR#127

646960956 1995 MFGR#128

508749401 1995 MFGR#129

523879145 1996 MFGR#121

643645053 1996 MFGR#1210

595065339 1996 MFGR#1211

674626440 1996 MFGR#1212

496297087 1996 MFGR#1213

583249505 1996 MFGR#1214

702184857 1996 MFGR#1215

601809334 1996 MFGR#1216

704898387 1996 MFGR#1217

528843086 1996 MFGR#1218

586246330 1996 MFGR#1219

712110492 1996 MFGR#122

518444215 1996 MFGR#1220

499319414 1996 MFGR#1221

679469356 1996 MFGR#1222

628762754 1996 MFGR#1223

724844856 1996 MFGR#1224

660620587 1996 MFGR#1225

667674729 1996 MFGR#1226

483838085 1996 MFGR#1227

609855391 1996 MFGR#1228

658959557 1996 MFGR#1229

566217852 1996 MFGR#123

528879998 1996 MFGR#1230

589481194 1996 MFGR#1231

702805896 1996 MFGR#1232

663679947 1996 MFGR#1233

571149450 1996 MFGR#1234

478648074 1996 MFGR#1235

568249365 1996 MFGR#1236

592616167 1996 MFGR#1237

466676148 1996 MFGR#1238

670693719 1996 MFGR#1239

560667719 1996 MFGR#124

821167950 1996 MFGR#1240

476864333 1996 MFGR#125

558030884 1996 MFGR#126

635873891 1996 MFGR#127

551010618 1996 MFGR#128

560570630 1996 MFGR#129

587013207 1997 MFGR#121

616287892 1997 MFGR#1210

548588761 1997 MFGR#1211

589593892 1997 MFGR#1212

424306670 1997 MFGR#1213

511971910 1997 MFGR#1214

631772246 1997 MFGR#1215

692135140 1997 MFGR#1216

777994957 1997 MFGR#1217

707053720 1997 MFGR#1218

561169527 1997 MFGR#1219

664916245 1997 MFGR#122

594466157 1997 MFGR#1220

588848171 1997 MFGR#1221

528988960 1997 MFGR#1222

537098211 1997 MFGR#1223

674763166 1997 MFGR#1224

450402292 1997 MFGR#1225

701360722 1997 MFGR#1226

506011570 1997 MFGR#1227

585578737 1997 MFGR#1228

622744016 1997 MFGR#1229

646503168 1997 MFGR#123

571800941 1997 MFGR#1230

502601790 1997 MFGR#1231

677924656 1997 MFGR#1232

534455976 1997 MFGR#1233

714934715 1997 MFGR#1234

767151420 1997 MFGR#1235

618877179 1997 MFGR#1236

639638057 1997 MFGR#1237

401953419 1997 MFGR#1238

610756714 1997 MFGR#1239

543248087 1997 MFGR#124

675132692 1997 MFGR#1240

479099365 1997 MFGR#125

570696568 1997 MFGR#126

583074592 1997 MFGR#127

695133104 1997 MFGR#128

655638776 1997 MFGR#129

344575925 1998 MFGR#121

417152416 1998 MFGR#1210

317068168 1998 MFGR#1211

374341516 1998 MFGR#1212

332740903 1998 MFGR#1213

304873002 1998 MFGR#1214

366101132 1998 MFGR#1215

379133898 1998 MFGR#1216

359508497 1998 MFGR#1217

320623334 1998 MFGR#1218

346182862 1998 MFGR#1219

312440027 1998 MFGR#122

348123961 1998 MFGR#1220

339845398 1998 MFGR#1221

355416161 1998 MFGR#1222

344889822 1998 MFGR#1223

396906691 1998 MFGR#1224

290208878 1998 MFGR#1225

419415707 1998 MFGR#1226

358466340 1998 MFGR#1227

251549955 1998 MFGR#1228

383138860 1998 MFGR#1229

296330561 1998 MFGR#123

437181243 1998 MFGR#1230

398944492 1998 MFGR#1231

424062455 1998 MFGR#1232

406967188 1998 MFGR#1233

428867240 1998 MFGR#1234

352277781 1998 MFGR#1235

361827086 1998 MFGR#1236

341618569 1998 MFGR#1237

244739231 1998 MFGR#1238

414151803 1998 MFGR#1239

330082371 1998 MFGR#124

415312453 1998 MFGR#1240

360289624 1998 MFGR#125

341657580 1998 MFGR#126

377507061 1998 MFGR#127

361416497 1998 MFGR#128

318769573 1998 MFGR#129

Time taken: 95.876 seconds, Fetched: 280 row(s)

Perform the following transform operation using SELECT TRANSFORM on the customer table by creating a new table:

For the c\_address column, shorten it to 8 characters (i.e., if the value is longer, remove extra characters, but otherwise keep it as-is). For c\_city, add a space and a # to indicate the digit at the end (e.g., UNITED KI2 => UNITED KI #2, or INDONESIA4 => INDONESIA #4). Make sure to modify the columns of the target table accordingly (since you are introducing longer columns).

# customer.py

#!/usr/bin/python

import sys

for line in sys.stdin:

line = line.strip().split(',')

line[2] = line[2][:8]

line[3] = line[3][:len(line[3])-1] + ' #' + line[3][len(line[3])-1]

print '\t'.join(line)

# create and load customer table

# CREATE TABLE customer (

# c\_custkey INT,

# c\_name VARCHAR(25),

# c\_address VARCHAR (25),

# c\_city VARCHAR (10),

# c\_nation VARCHAR (15),

# c\_region VARCHAR (12),

# c\_phone VARCHAR (15),

# c\_mktsegment VARCHAR (10)

# )

# ROW FORMAT DELIMITED FIELDS

# TERMINATED BY '|' STORED AS TEXTFILE;

# LOAD DATA LOCAL INPATH '/home/ec2-user/customer.tbl' OVERWRITE INTO TABLE customer;

# Create customer2 table

create table customer2 (

c\_custkey INT,

c\_name VARCHAR(25),

c\_address VARCHAR (8),

c\_city VARCHAR (12),

c\_nation VARCHAR (15),

c\_region VARCHAR (12),

c\_phone VARCHAR (15),

c\_mktsegment VARCHAR (10)

)

ROW FORMAT DELIMITED FIELDS

TERMINATED BY '\t' STORED AS TEXTFILE;

**Select transform**

INSERT OVERWRITE TABLE customer2 SELECT TRANSFORM (c\_custkey, c\_name, c\_address, c\_city, c\_nation, c\_region, c\_phone, c\_mktsegment) USING 'python customer.py' AS (c\_custkey, c\_name, c\_address, c\_city, c\_nation, c\_region, c\_phone, c\_mktsegment) FROM customer;

# test query:

# hive> select \* from customer2 limit 5;

# OK

# 1 Customer#000000001 j5JsirBM MOROCCO #0 MOROCCO AFRICA 25-989-741-2988 BUILDING

# 2 Customer#000000002 487LW1do JORDAN #1 JORDAN MIDDLE EAST 23-768-687-3665 AUTOMOBILE

# 3 Customer#000000003 fkRGN8n ARGENTINA #7 ARGENTINA AMERICA 11-719-748-3364 AUTOMOBILE

# 4 Customer#000000004 4u58h f EGYPT #4 EGYPT MIDDLE EAST 14-128-190-5944 MACHINERY

# 5 Customer#000000005 hwBtxkoB CANADA #5 CANADA AMERICA 13-750-942-6364 HOUSEHOLD

# Time taken: 0.051 seconds, Fetched: 5 row(s)

# Part 3: Pig

Convert and load the data into Pig, implementing only queries 0.1, 0.2, 0.3. Do not implement all queries.

Check disk storage space in HDFS, if your disk usage is over 90% Pig may hang without an error or a warning.

One easy way to time Pig is as follows: put your sequence of pig commands 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 inform you how long the pig script took to run).

**0.1**

lineorder = 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:chararray,

lo\_shippriority:chararray,

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::chararray

);

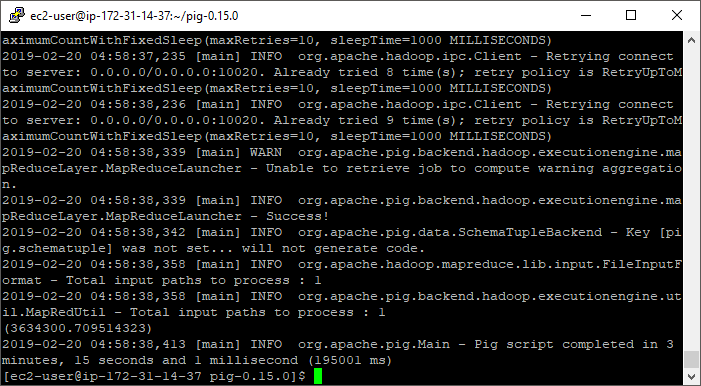
# grouped = group lineorder all;

# avg = foreach grouped generate AVG(lineorder.lo\_revenue);

# DUMP avg;

Answer = 3634300.709514323

Execution time of 3 minutes, 15 seconds, 1 millisecond



# 0.2

grouped = group lineorder by lo\_discount;

counted = FOREACH grouped GENERATE group as lo\_discount, COUNT(lineorder.lo\_extendedprice) as cnt;

DUMP counted;

2019-02-20 05:20:36,325 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(0,544886)

(1,545834)

(2,546173)

(3,545293)

(4,545545)

(5,546395)

(6,544970)

(7,546192)

(8,544803)

(9,545309)

(10,545815)

2019-02-20 05:20:36,402 [main] INFO org.apache.pig.Main - Pig script completed in 3 minutes, 30 seconds and 49 milliseconds (210049 ms)

# 0.3

filtered = FILTER lineorder BY lo\_discount < 3;

grouped = GROUP filtered BY lo\_quantity;

summed = FOREACH grouped GENERATE group as lo\_quantity, SUM(filtered.lo\_revenue) as rev;

DUMP summed;

2019-02-20 06:04:22,932 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1

(1,4879019020)

(2,9644127315)

(3,14575887127)

(4,19360189865)

(5,24073923574)

(6,29125189531)

(7,33982891466)

(8,38671565454)

(9,43381602619)

(10,48619780003)

(11,53159489411)

(12,58264291629)

(13,62920595763)

(14,67451818069)

(15,73414895616)

(16,78360133885)

(17,82320521791)

(18,86995495639)

(19,93016668045)

(20,97258062753)

(21,100684344044)

(22,107454001560)

(23,112984674102)

(24,116527702603)

(25,123160894092)

(26,126451771059)

(27,132113291310)

(28,135413154368)

(29,141357789043)

(30,145181046794)

(31,149937771539)

(32,157770330201)

(33,161774040572)

(34,164150363629)

(35,170173151151)

(36,175712858188)

(37,178733976488)

(38,186428562667)

(39,187696104837)

(40,196345645204)

(41,199250645070)

(42,204966410590)

(43,209016181876)

(44,213245636104)

(45,217565230742)

(46,223784510215)

(47,229077142619)

(48,234125822088)

(49,236641410613)

(50,243791122644)

2019-02-20 06:04:22,987 [main] INFO org.apache.pig.Main - Pig script completed in 3 minutes, 5 seconds and 71 milliseconds (185071 ms)

# Part 4: Hadoop Streaming

Implement queries **0.2 and 0.3** using Hadoop streaming with python.

**Set up commands to move lineorder.tbl into Hadoop directory.**

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -mkdir /user/ec2-user/ssbm

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -put lineorder.tbl ssbm

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -ls ssbm

Found 1 items

-rw-r--r-- 2 ec2-user supergroup 594313001 2019-02-21 21:34 ssbm/lineorder.tbl

--Q0.2 Added simple test query

SELECT lo\_discount, COUNT(lo\_extendedprice)

FROM lineorder

GROUP BY lo\_discount;

For this query, I wrote my own mapper and thought I could use the reducer example provided.

**myMapper.0.2.py**

#!/usr/bin/python

import sys

for line in sys.stdin:

line = line.strip()

vals = line.split("|")

discount = vals[11]

# vals[9] = lo\_extendedprice, but any value would work for count

if discount == "0":

print "%s\t%s" % (discount, vals[9])

**myReducer.py (provided example)**

#!/usr/bin/python

import sys

curr\_id = None

curr\_cnt = 0

id = None

# The input comes from standard input (line by line)

for line in sys.stdin:

line = line.strip()

# parse the line and split it by '\t'

ln = line.split('\t')

# grab the key (int)

id = int(ln[0])

if curr\_id == id:

curr\_cnt += 1

else:

if curr\_id: # output the count, single key completed

# NOTE: Change this to '%s\t%d' if your key is a string

print '%d\t%d' % (curr\_id, curr\_cnt)

curr\_id = id

curr\_cnt = 0

# output the last key

if curr\_id == id:

print '%s\t%d' % (curr\_id, curr\_cnt)

However, when I used this reducer, I found that the “0” discount wasn’t output.

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -cat /data/output5/part-00000

1 545833

10 545814

2 546172

3 545292

4 545544

5 546394

6 544969

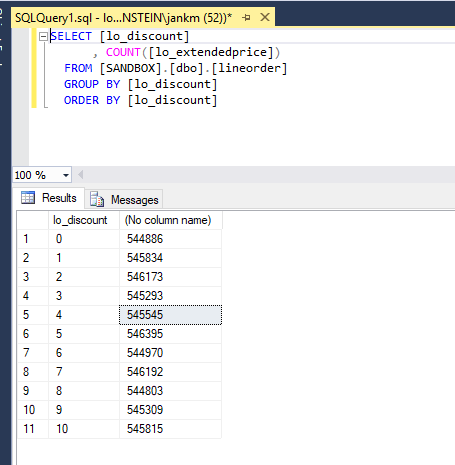
7 546191

8 544802

9 545308

I attributed this to the conversion of id to an integer, removing this resolved the issue.

Also, as a means to check my work, I loaded the lineorder table into a local SQL Server instance and compared the counts. The streaming results were all lower by 1.



The SQL Server results match the Pig results.

I believe this is because curr\_cnt should be initialized to 1 in myReducer and not 0. The first time an id passes through the code, curr\_cnt isn’t incremented because curr\_id == id is not true. Curr\_cnt isn’t incremented until the second iteration.

Here is the modified reducer code and final output.

**myReducer.0.2.py**

#!/usr/bin/python

import sys

curr\_id = None

curr\_cnt = 1

id = None

# The input comes from standard input (line by line)

for line in sys.stdin:

line = line.strip()

# parse the line and split it by '\t'

ln = line.split('\t')

# grab the key (int)

id = ln[0]

if curr\_id == id:

curr\_cnt += 1

else:

if curr\_id: # output the count, single key completed

# NOTE: Change this to '%s\t%d' if your key is a string

print '%s\t%d' % (curr\_id, curr\_cnt)

curr\_id = id

curr\_cnt = 1

# output the last key

if curr\_id == id:

print '%s\t%d' % (curr\_id, curr\_cnt)

**Command:**

[ec2-user@ip-172-31-14-37 ~]$ hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/ssbm -output /data/output9 -mapper myMapper.0.2.py -reducer myReducer.0.2.py -file myReducer.0.2.py -file myMapper.0.2.py

**Results:**

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -cat /data/output9/part-00000 0 544886

1 545834

10 545815

2 546173

3 545293

4 545545

5 546395

6 544970

7 546192

8 544803

9 545309

--Q0.3 Added simple test query

SELECT lo\_quantity, SUM(lo\_revenue)

FROM lineorder

WHERE lo\_discount < 3

GROUP BY lo\_quantity;

**myMapper.0.3.py**

#!/usr/bin/python

import sys

for line in sys.stdin:

line = line.strip()

vals = line.split("|")

discount = int(vals[11])

quantity = vals[8]

revenue = vals[12]

if discount < 3:

print "%s\t%s" % (quantity, revenue)

**myReducer.0.3.py**

#!/usr/bin/python

import sys

curr\_id = None

curr\_tot = 0

id = None

# The input comes from standard input (line by line)

for line in sys.stdin:

line = line.strip()

# parse the line and split it by '\t'

ln = line.split('\t')

# grab the key

id = ln[0]

# grab the value (int)

val = int(ln[1])

if curr\_id == id:

curr\_tot += val

else:

if curr\_id: # output the sum, single key completed

print '%s\t%d' % (curr\_id, curr\_tot)

curr\_id = id

curr\_tot = val

# output the last key

if curr\_id == id:

print '%s\t%d' % (curr\_id, curr\_tot)

**command**

[ec2-user@ip-172-31-14-37 ~]$ hadoop jar hadoop-streaming-2.6.4.jar -input /user/ec2-user/ssbm -output /data/output16 -mapper myMapper.0.3.py -reducer myReducer.0.3.py -file myReducer.0.3.py -file myMapper.0.3.py

**Results**

[ec2-user@ip-172-31-14-37 ~]$ hadoop fs -cat /data/output16/part-00000

1 4879019020

10 48619780003

11 53159489411

12 58264291629

13 62920595763

14 67451818069

15 73414895616

16 78360133885

17 82320521791

18 86995495639

19 93016668045

2 9644127315

20 97258062753

21 100684344044

22 107454001560

23 112984674102

24 116527702603

25 123160894092

26 126451771059

27 132113291310

28 135413154368

29 141357789043

3 14575887127

30 145181046794

31 149937771539

32 157770330201

33 161774040572

34 164150363629

35 170173151151

36 175712858188

37 178733976488

38 186428562667

39 187696104837

4 19360189865

40 196345645204

41 199250645070

42 204966410590

43 209016181876

44 213245636104

45 217565230742

46 223784510215

47 229077142619

48 234125822088

49 236641410613

5 24073923574

50 243791122644

6 29125189531

7 33982891466

8 38671565454

9 43381602619

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

Submit a single document containing your written answers. Be sure that this document contains your name and “CSC 555 Project Phase 1” at the top.