CS 553:Cloud Computing
Assignment-2

Name:Priyank Shah **CWID:A20344797**

1. Design Report

In Assignment-2 following are the parts that I have implemented:

- Setting up Virtual Cluster of c3.large contains 1 master and 1 slave, mounting EBS volume on master and slave node.
- Setting up Virtual Cluster of c3.large contains 1 master and 16 slave, mounting EBS volume on master and slave node.
- Implementing Hadoop Sort for 1 node and 16 nodes.
- Implementing Spark Sort for 1 node and 16 nodes.
- Implementing Shared Memory sort for Single Thread and Multiple thread.
- Comparison between Shared memory, Hadoop and Spark on 1 Node.
- Measuring Speed up and Performance evaluation of 1 node and 16 node cluster.

In Hadoop, I have generated input with gensort for 10 GB and 100 GB and run them on 1 and 16 nodes respectively. I have implemented the program in java. For Hadoop you need to generate jar file from java file, add class file to jar. Put generated data on hadoop file system and run the program with jar file. Get the output from hadoop file system to local file system and check with val sort for the sake of sorted output validity.

In Spark, I have generated input with gensort for 10 GB and 100 GB and run them on 1 and 16 nodes respectively. I have implemented the program in scala, which cam be run in almost similar fashion as java. Put the scala file in the same folder where spark-shell resides. Open the spark-shell and run the program, it will generate the output.

In Shared Memory, I have generated input with gensort for 10 GB. I am taking user input for threads. Once program start, make sure to have enough space for intermediate files(temporary files). It will cut data in 20 MB Chunks. And perform quick sort. On temporary generated file it will imply merge sort and builds single output file.

Configuration and Installation:

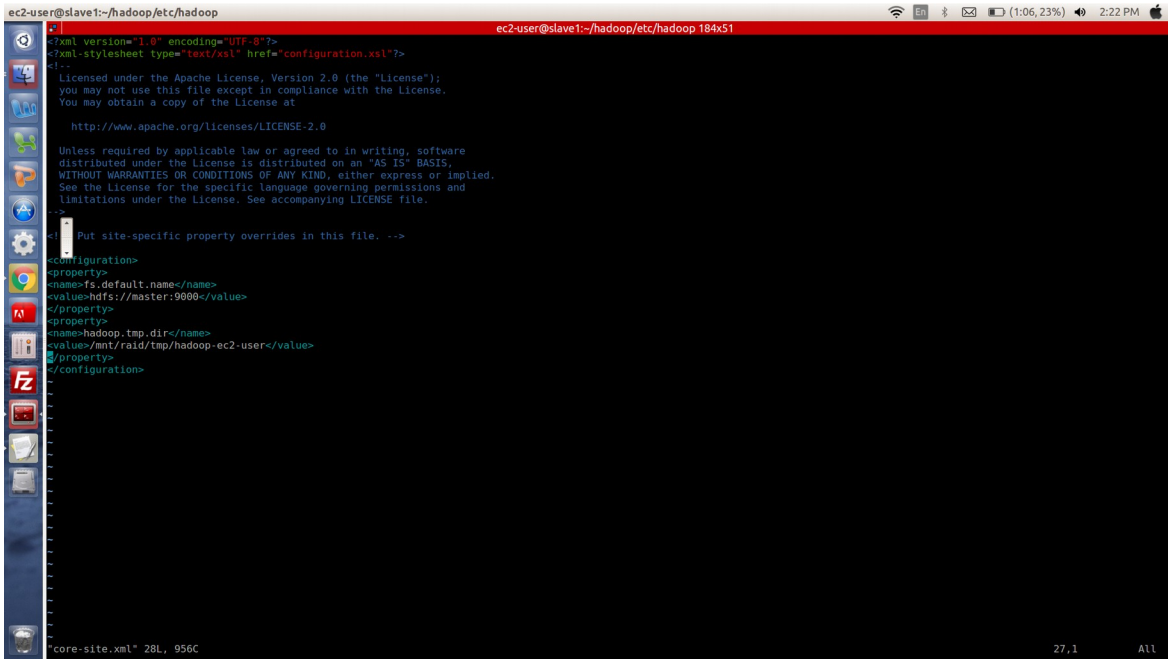
1.Hadoop:

Instance Type: c3.large
RAM: 3.75 GB
# of Instance: 1(for 10 GB) and 17(1 master and 16 slave)
Storage: 32 GB
OS: Amazon Linux AMI
Java version: 1.7.0
Hadoop Version: 2.7.2

Following are Files Changes which resides in /hadoop/etc/hadoop:

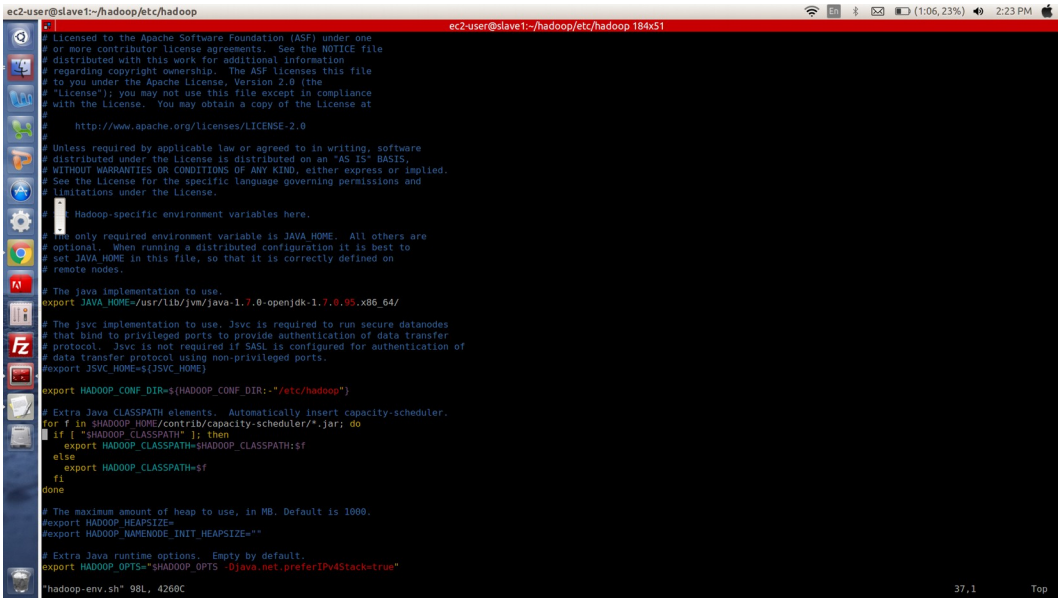
1. core-site.xml:

```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://master:9000</value>
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/mnt/raid/tmp/hadoop-ec2-user</value>
</property>
</configuration>
```

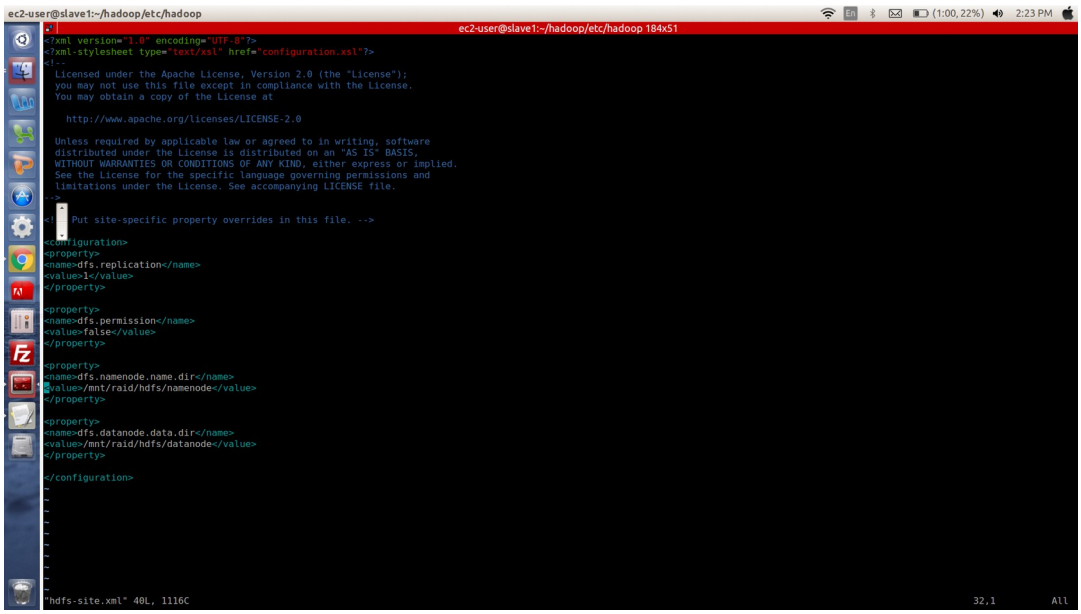


2.hadoop-env.sh:

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.95.x86_64/
```



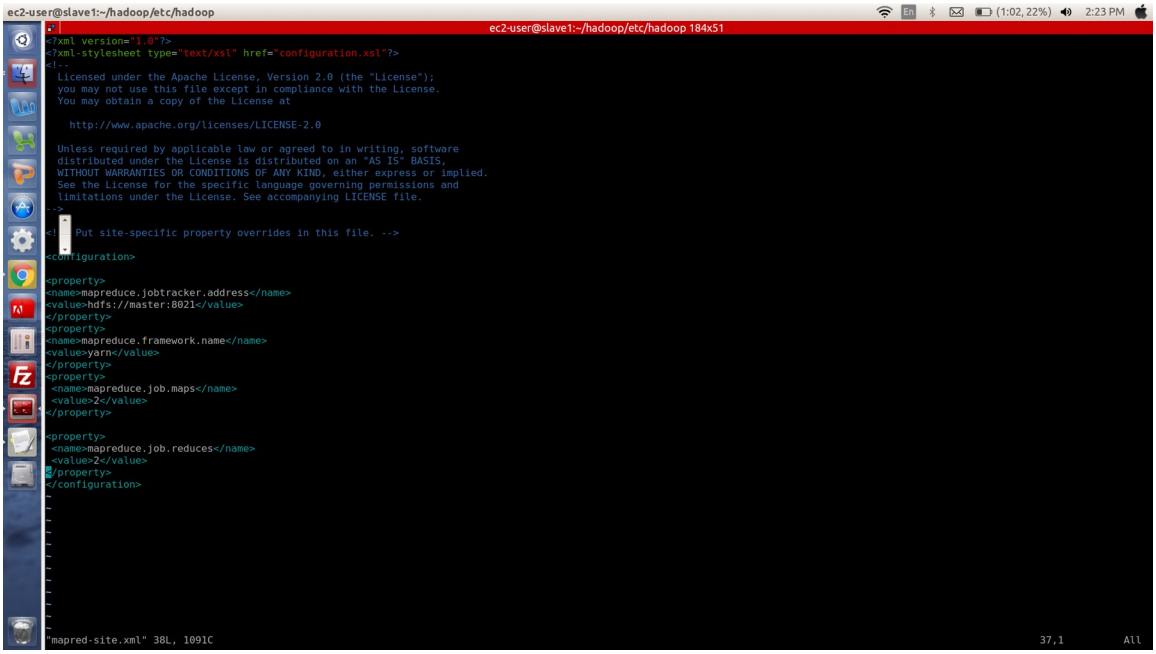
3.hdfs-site.xml:




```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
```

```
<property>
<name>dfs.permission</name>
<value>false</value>
</property>

<property>
<name>dfs.namenode.name.dir</name>
<value>/mnt/raid/hdfs/namenode</value>
</property>

<property>
<name>dfs.datanode.data.dir</name>
<value>/mnt/raid/hdfs/datanode</value>
</property>
</configuration>
```

4. mapred-site.xml:

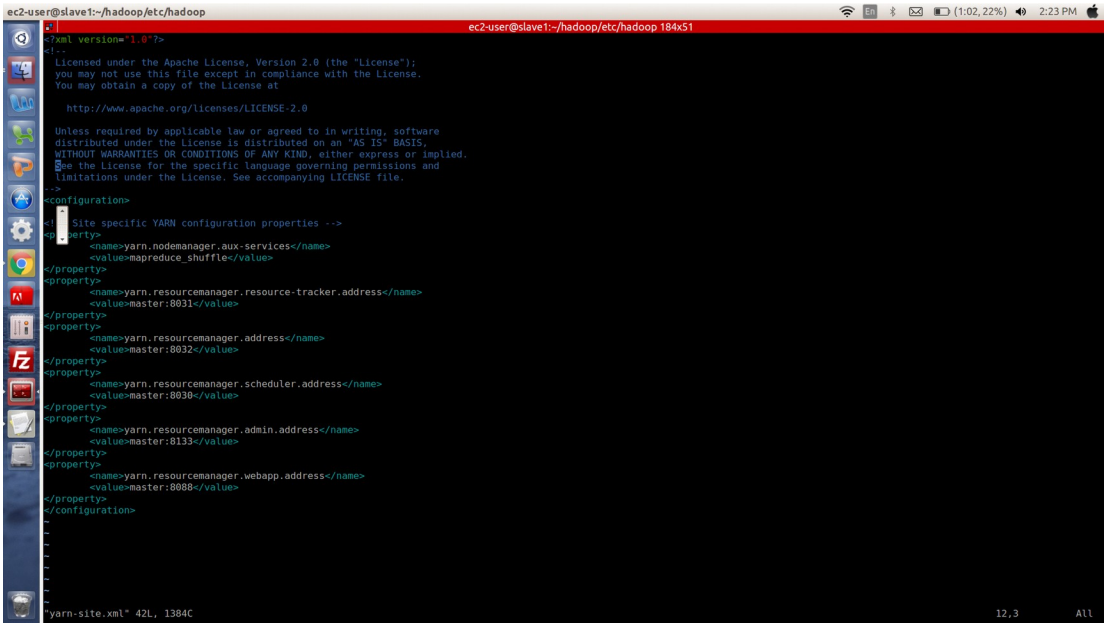



```
<configuration>
<property>
<name>mapreduce.jobtracker.address</name>
<value>hdfs://master:8021</value>
</property>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
```

note: following properties are only application to 100 GB

```
experiments. Value is decided by number of threads available in
instance to determine the optimum performance.

<property>
 <name>mapreduce.job.maps</name>
 <value>2</value>
</property>
<property>
 <name>mapreduce.job.reduces</name>
 <value>2</value>
</property>

</configuration>
```

5.yarn-site.xml

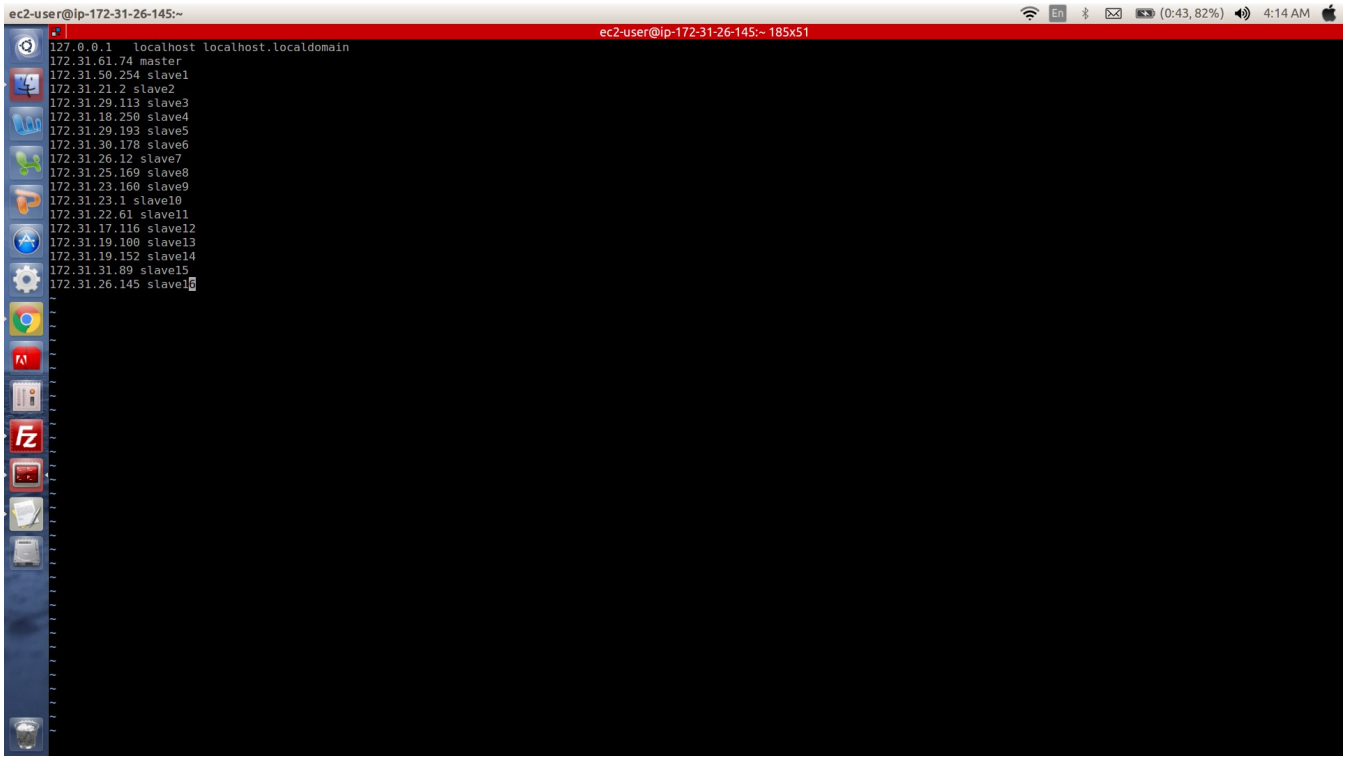



```
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
</property>
<property>
                                <name>yarn.resourcemanager.resource-
tracker.address</name>
        <value>master:8031</value>
</property>
<property>
        <name>yarn.resourcemanager.address</name>
        <value>master:8032</value>
```

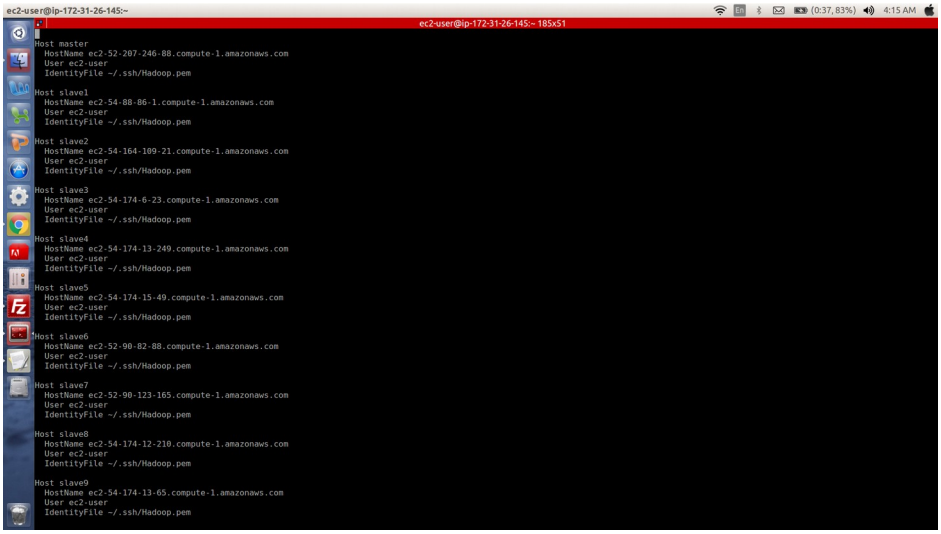
```
</property>

<property>
        <name>yarn.resourcemanager.scheduler.address</name>
        <value>master:8030</value>
</property>
<property>
        <name>yarn.resourcemanager.admin.address</name>
        <value>master:8133</value>
</property>
<property>
        <name>yarn.resourcemanager.webapp.address</name>
        <value>master:8088</value>
</property>
</configuration>
```

Changes in system files are as follow:

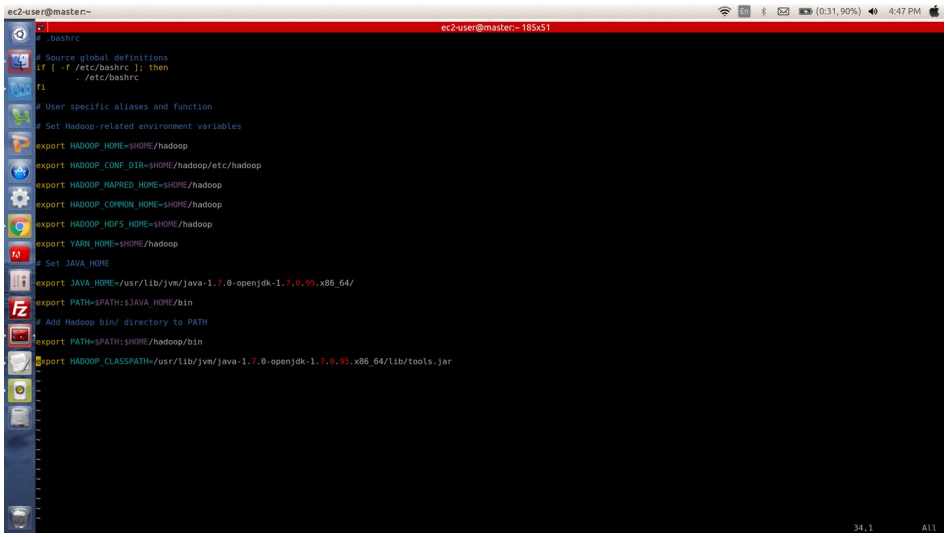
6. etc/hosts:



7. ~/.ssh/config



8. /.bashrc:



Installation Procedure:

1.Choose Amazon Machine Image from the Given list, preferably Amazon Linux AMI.

2.Select Instance Type, In our case choose c3.large

3.Select Number of instances and do "spot Request" if you are launching 16 instances. Bid proper price for instance and select the subnet that has the least price mentioned.

4. Add Storage as per requirement. For 10 GB experiment – 30 GB EBS volume is enough while for 100 GB experiment take at least 300 GB worth of space and on slave keep 60 GB EBS.

5. Configure Security Group, Select **All Traffic** to ensure successful connection between server and client.

6. Once Instance is launched follow the procedure:

7. Download java with following link:
   sudo apt-get install java-1.7.0-openjdk
   sudo apt-get install java-devel

8. Download Hadoop from the following link:
   wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.2/hadoop-2.7.2.tar.gz

9. Unzip Hadoop with:
   tar -xvf hadoop-2.7.2.tar.gz
   rename it:
   mv hadoop-2.7.2 hadoop

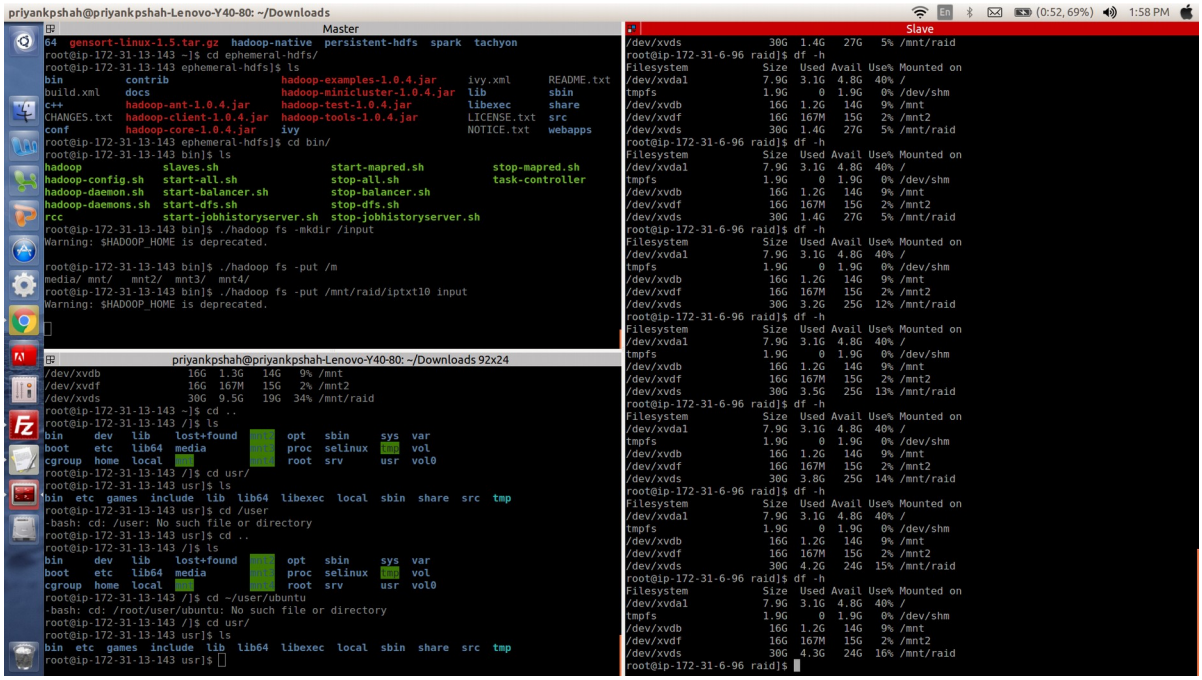
10. Download gensort from following link:
    wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz

11. Follows the Configuaration procedure from report file.

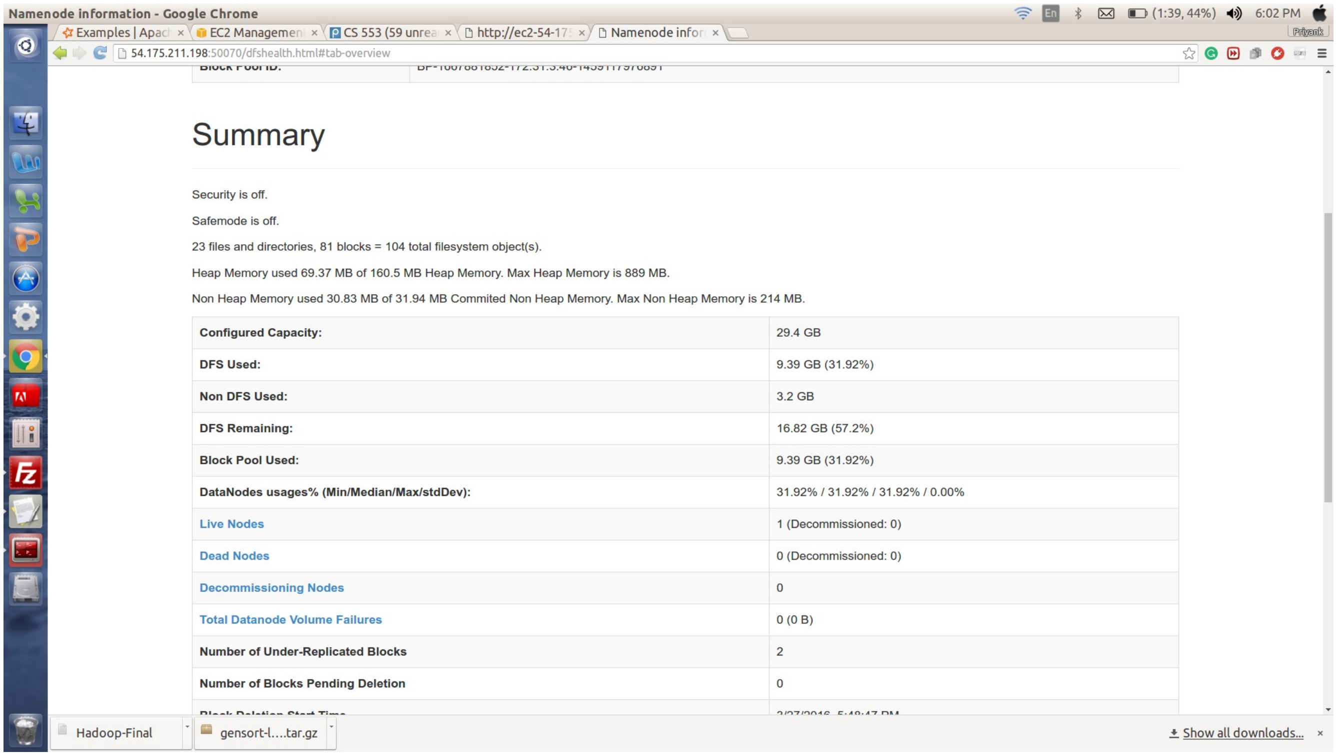
ScreenShots:

10 GB Experiment:

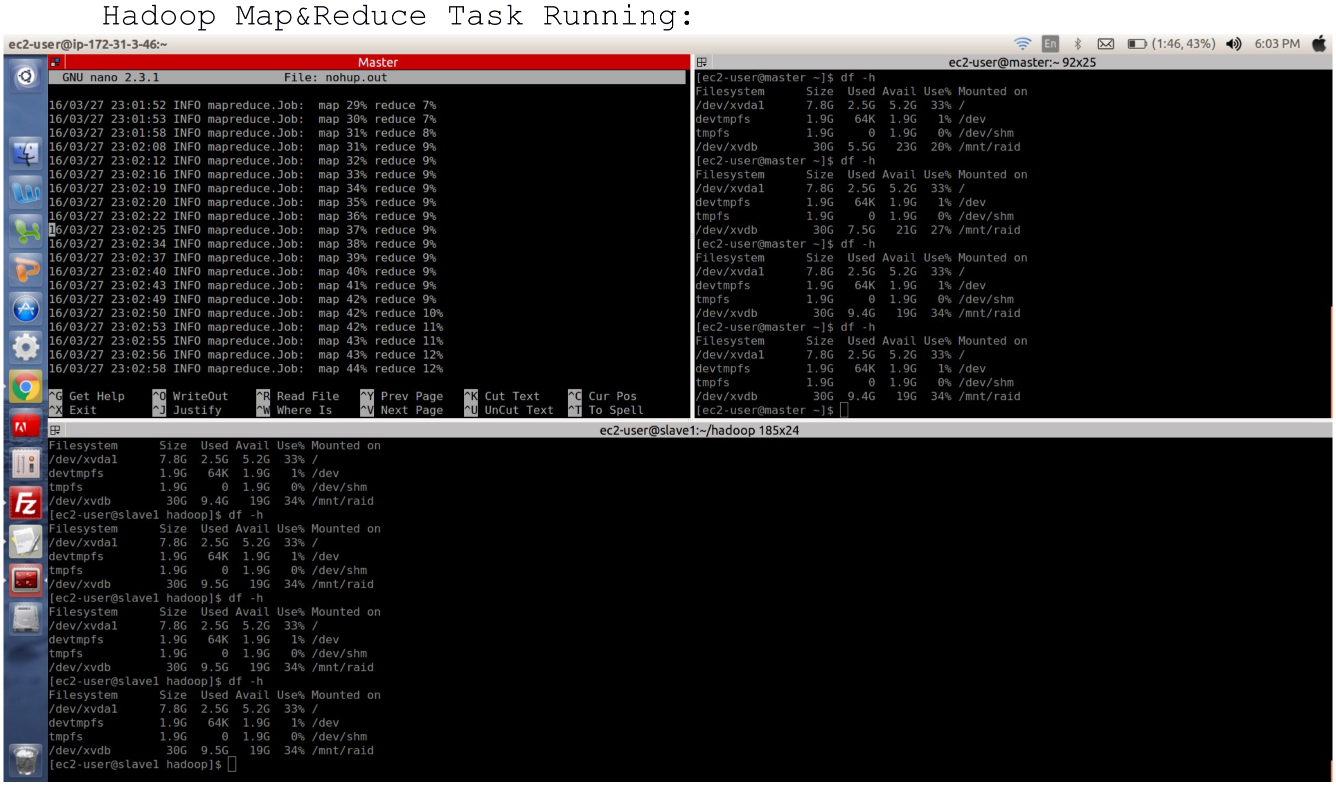
After Configuration done and data generated, Put data on hadoop file system.



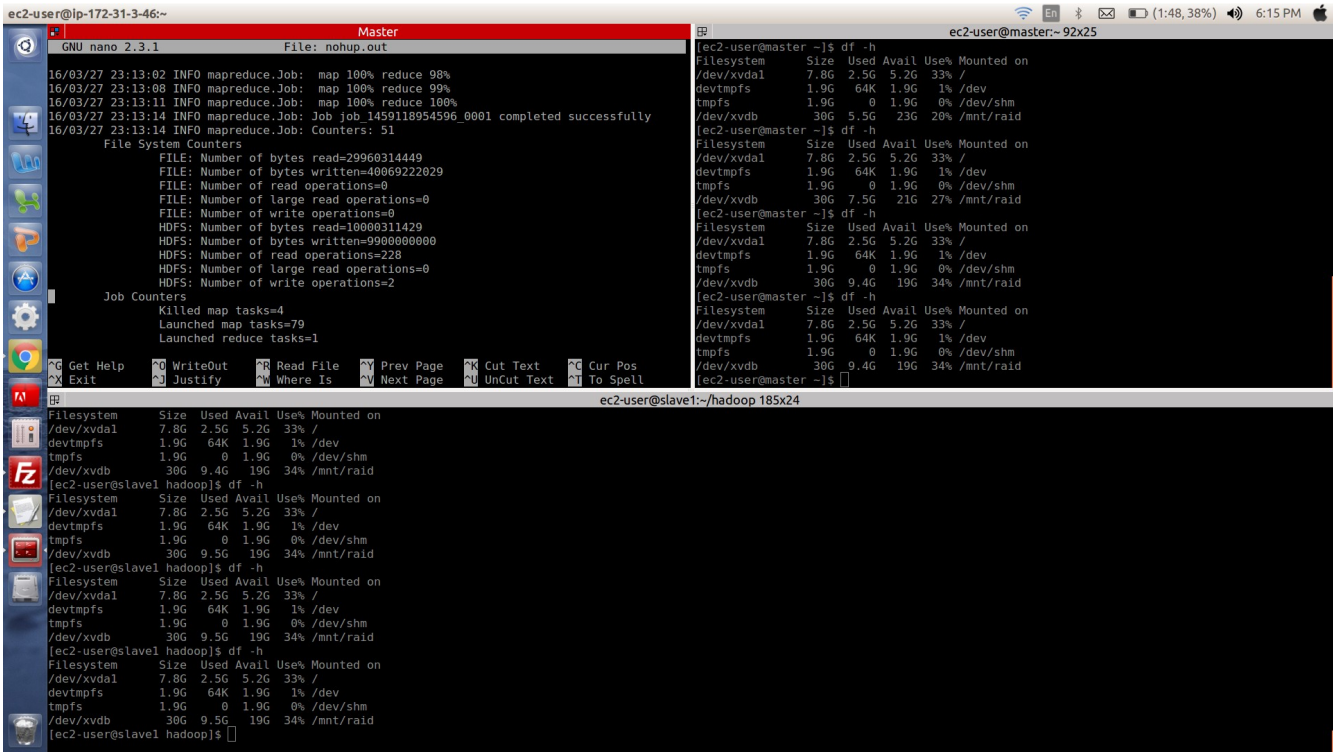
Once data is on hadoop file system,open you browser and type, master_public_dns:50070. You can see that data is on the datanode(slave) and it is running.



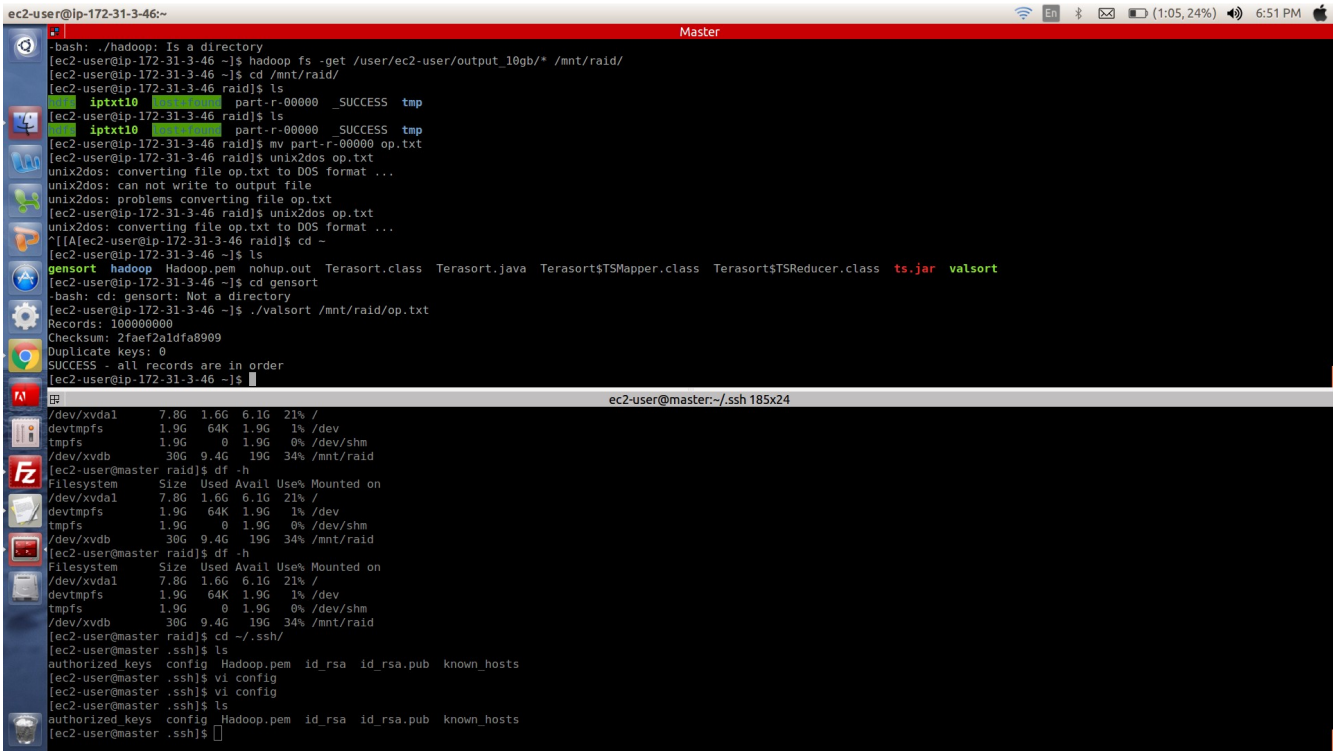
Hadoop Map&Reduce Task Running:



Hadoop Map&Reduce Task done:

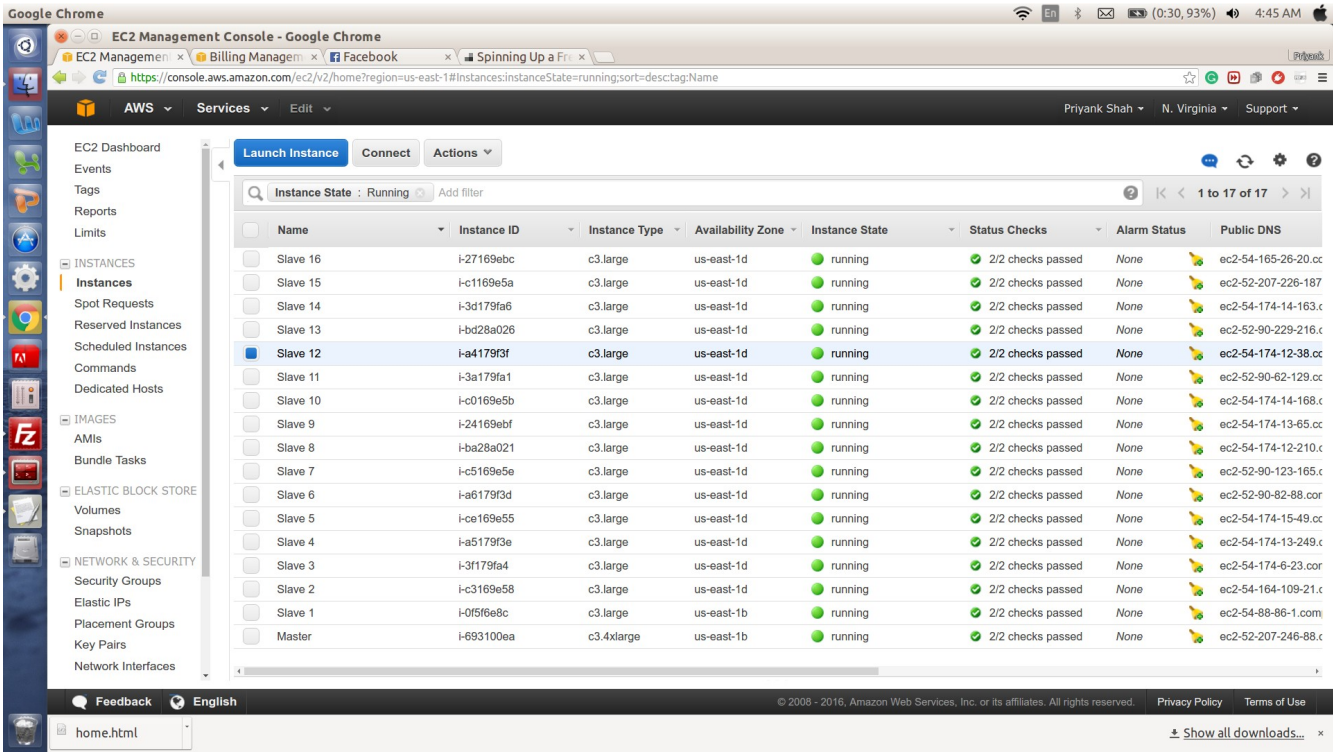


Hadoop Valsort:

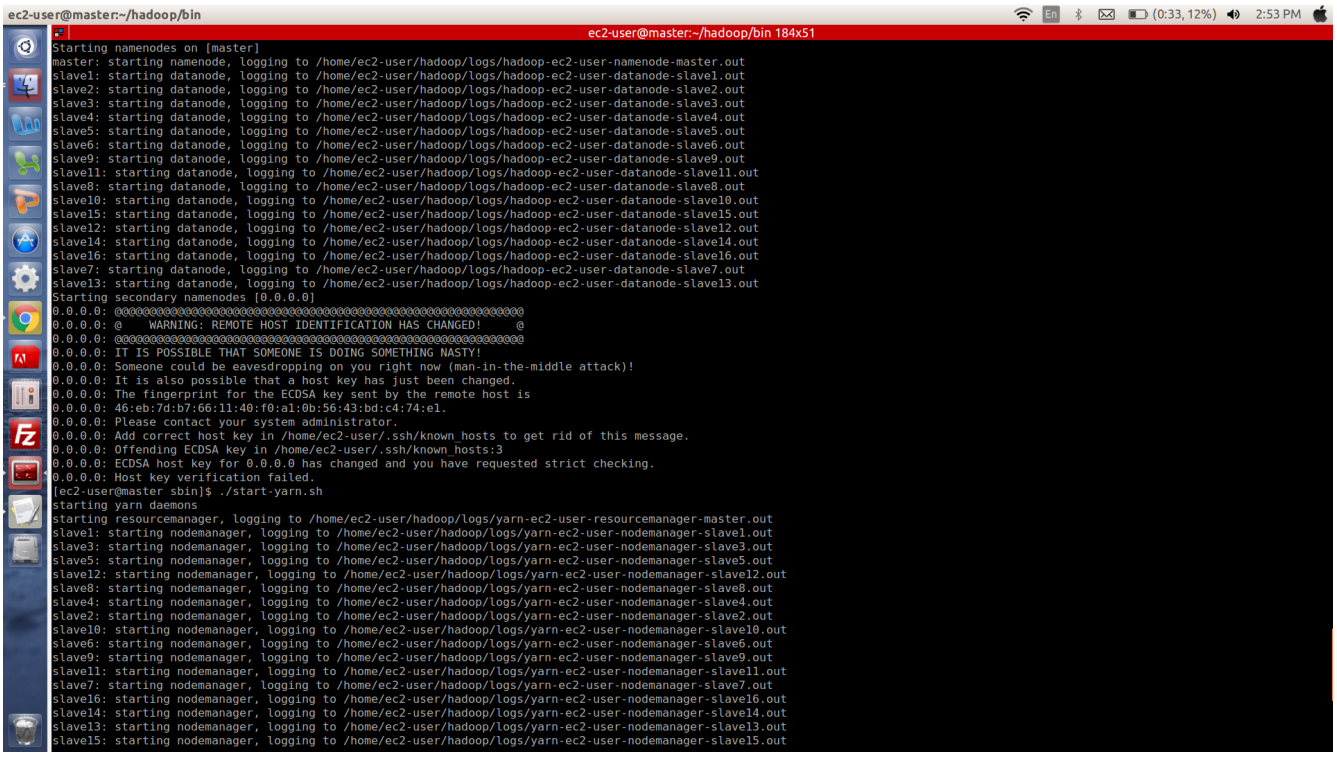


100 GB Experiment:

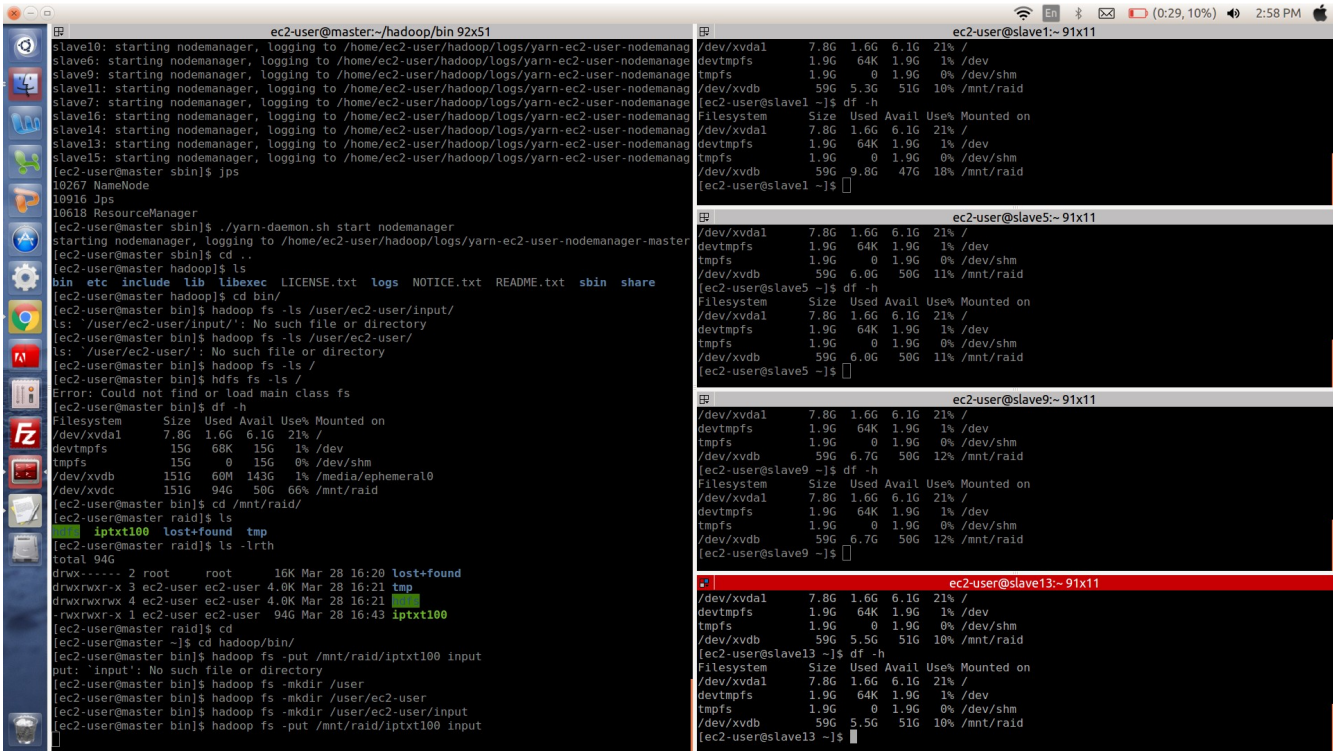
16 nodes Running:



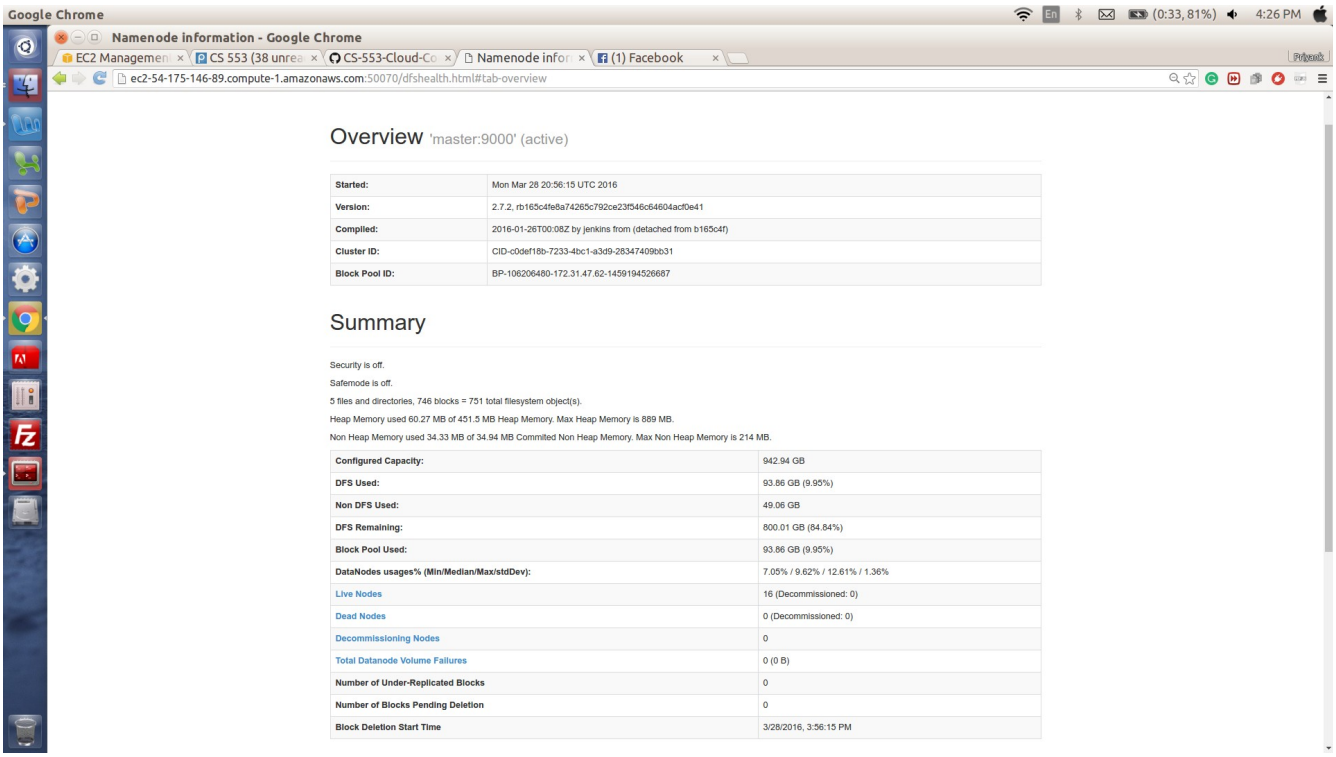
Datanode Started:



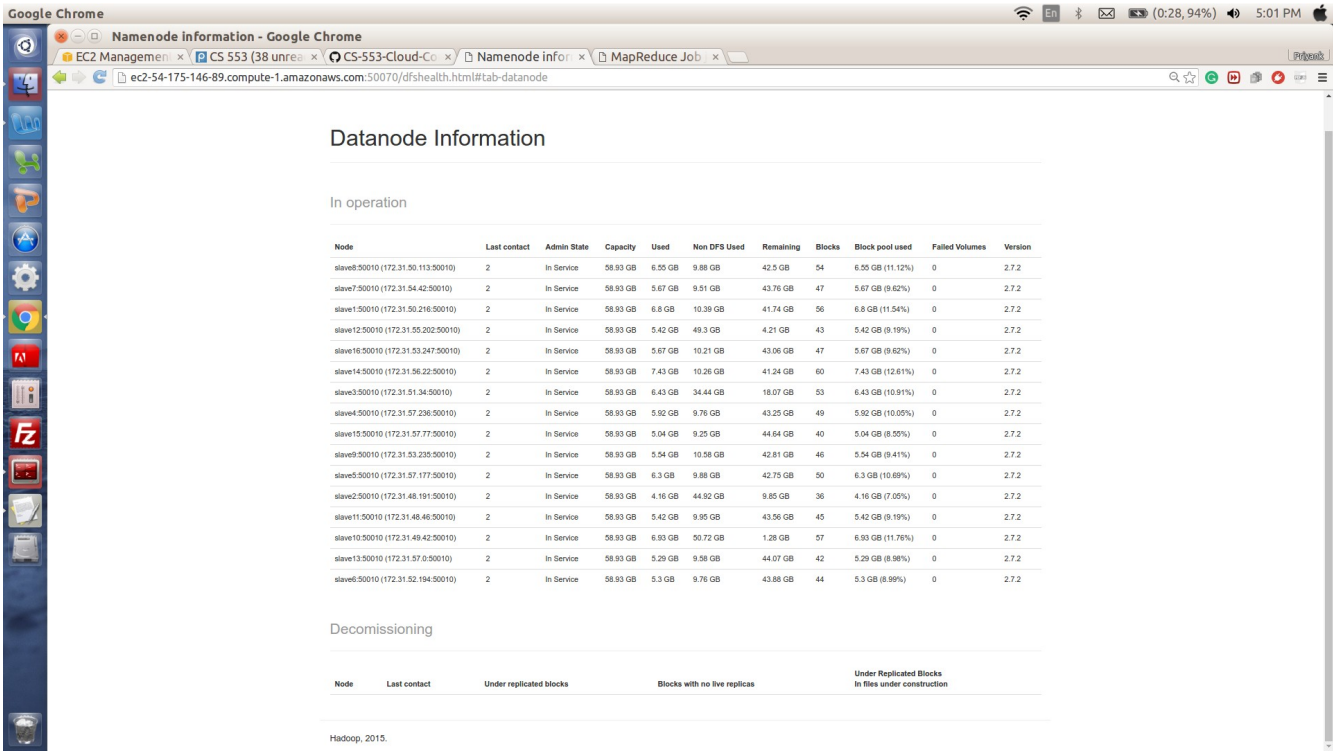
Put Data on Hadoop filesystem:



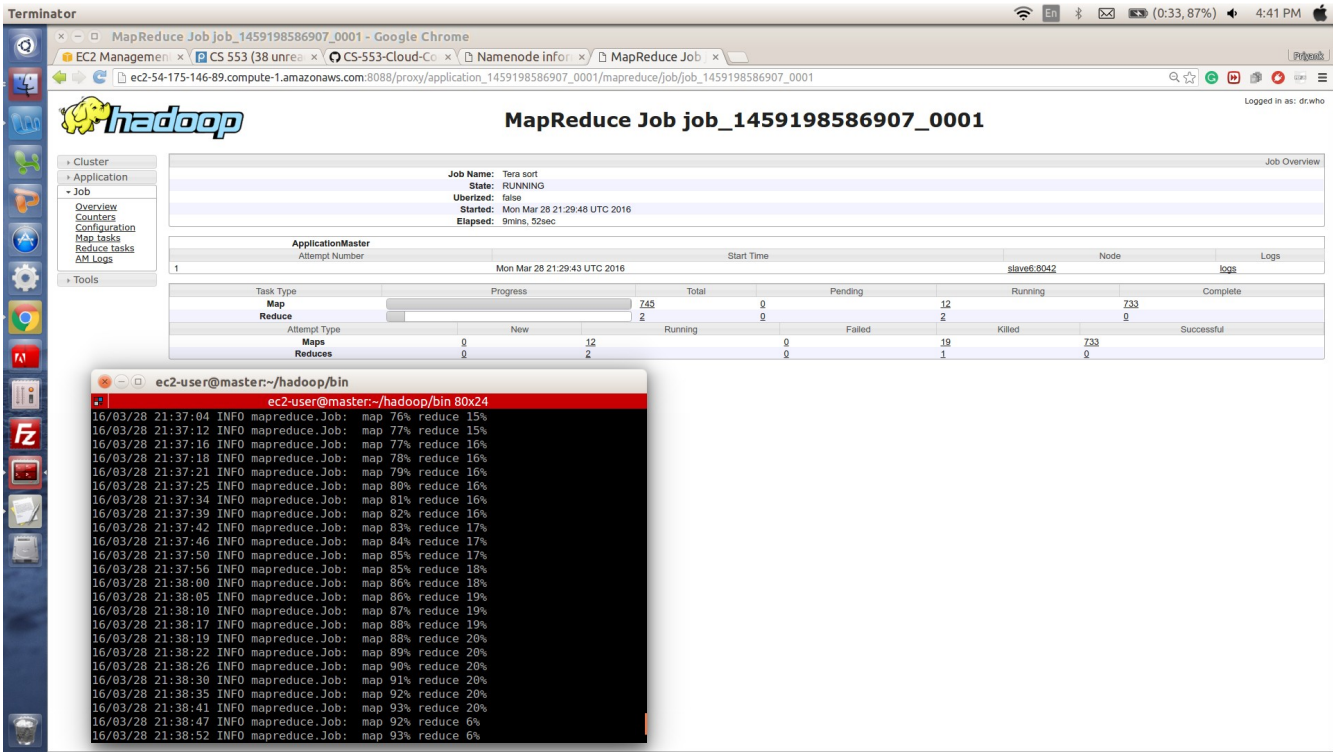
Data Set on Datanode:



DataNodes:



Hadoop Job Running:



Problem Faced:

1. Datanode was not generating on slave nodes, For this problem first delete the any current data in datanode. If problem persist,check path in hdfs-site.xml file. Check if sufficient permission is given to datanode directory to write datanode directory.

2. During the early phase of assignment, I didn't created alias for master and slave IP(s). So every time, I have to make changes to the hdfs-site file and yarn-site file for Master public DNS. Then I came to know about the alias in /etc/hosts file and ~/.ssh/config file which replaced to **master** and **slave** with their respective public DNS and private ips, in turns it decrease the amount of time taken to re-configure the master and slave node.

3. While doing 100 GB experiment, I attached 30 GB EBS volume to slaves, soon cluster ran out of space. At that time instead of terminating it and creating and configuring whole cluster, I found that we can attach EBS volume on running instance.

4. It was difficult to set 16 node cluster, especially manually configuring the files on each slave, I made a script which will copy different script and copy it to all the slaves and execute them remotely.

Methodology:

- First of all, Program will take input from the input file and store it as value.
- We need to sort the generated data on the base of key so, we will fetch first 10 char from the string and store it as key in Text data type field and rest in the another field as value.
- Mapper class sort the data which is key in our case before passing it to reducer class, so in second class that is TSReducer we will fetch the key(0,10 - char) with its respective value(10,end - char) and store it in Text. We will iterate one by one value and store.
- We will write it back as (key , value). That is our output. Here Reducer class will work as combiner too.

Questionnaires:

- **conf/masters** that is resided in hadoop/etc/hadoop, stores the hostname of master node.
- **conf/slaves** that is resided in hadoop/etc/hadoop, stores the hostname of all the slave nodes.
- **conf/core-site.xml** that is resided in hadoop/etc/hadoop, it

overrides the default core parameters. It informs the Hadoop deamon where namenode runs in cluster. Contains common I/O settings related to HDFS and MapReduce.

- **conf/hdfs-site.xml** that is resided in hadoop/etc/hadoop, contains the setting of Namenode , Secondary Namenode and Datanode like number of replication, permission checking on HDFS.
- **conf/mapred-site.xml** that is resided in hadoop/etc/hadoop, contains the configuration settings of MapReduce Daemons, job tracker and task-tracker.
- **What is Master node ? What is Data node?**

  Master node hosts the various storage and processing management services, described in list for entire Hadoop cluster, Namenode service are created on the top of Master Node.

  Data node is responsible for storing actual data in HDFS and processing them, Datanode service are created on the top of Data node.
- **Why we need to set unique port in shared environment service? Error or side-effect of using same port?**

  Each service uses different port to uniquely identified by callee. So it will not throw error , but it will not able to perform either of the service that has the same port numbers.
- **How can we change the number of reducers and mappers from the configuration file?**

  In hadoop/etc/hadoop there is a file called mapred-site.xml . We can add the value as number of mappers and reducers for **mapreduce.job.maps** and **mapreduce.job.reduces**.

- **Changes made while scaling from 1 node to 16 Node:**

  Particularly for this experiment we are using 10 GB dataset for 1 slave and 100 GB dataset for 16 slaves. So we need more space on both Master and Slave for 16 node Experiments. I would suggest at least 250 GB of space on Master Node. That is once you put data on Hadoop file system, delete the generated data to let output store.

  Otherwise, We need to mention all the 16 nodes private ip in/etc/hosts file and public dns in ~/.ssh/config file.

  Add slaves public dns in known_host file resides in same location as config file.

  Mention hostname/alias as per /etc/hosts in hadoop/etc/hadoop/slaves file.

  Change as mentioned above in configuration file in etc/hadoop/mapred-site.xml.

2.Spark Benchmarking:

Instance Type: c3.large
RAM: 3.75 GB
# of Instance: 1(for 10 GB) and 17(1 master and 16 slave)
Storage: 32 GB
OS: Amazon Linux AMI
Java version: 1.7.0
Spark Version: 1.6.1

Installation Procedure:

1.Choose Amazon Machine Image from the Given list, preferably Amazon Linux AMI.
2.Select Instance Type, In our case choose c3.large
3.Select Number of instances and do "spot Request" if you are launching 16 instances.Bid proper price for instance and select the subnet that has the least price mentioned.
4.Add Storage as per requirement. For 10 GB experiment – 30 GB EBS volume is enough, for 100 GB experiment have at least 250 GB space and on slave take 60 GB EBS.
5.Configure Security Group, Select **All Traffic** to ensure successful connection between server and client.
6.Once Instance is launched follow the procedure to download necessary files:
Download Java:

```
sudo apt-get update
sudo apt-get install openjdk-7-jre-headless
sudo apt-get install openjdk-7-jdk
```

Download Spark and unzip it:
wget http://mirrors.ocf.berkeley.edu/apache/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz

Download Scala:

```
wget http://www.scala-lang.org/files/archive/scala-2.10.4.tgz
```

7.Create local folder and unzip scala:

```
sudo mkdir /usr/local/src/scala
sudo tar xvf scala-2.10.4.tgz -C /usr/local/src/scala/
```

8.Open **bashrc** and add following parameter:

```
vi .bashrc
export SCALA_HOME=/usr/local/src/scala/scala-2.10.4
export PATH=$SCALA_HOME/bin:$PATH
```

source the bashrc:

```
. ./bashrc
```

Check if Scala is properly configured with
Scala -version

9. Export your Access Key and Access ID:

```
export AWS_ACCESS_KEY_ID=<Access Key ID>
export AWS_SECRET_ACCESS_KEY=<Secrete Access Key>
```

These keys can be generated by following steps:
go to your aws account, Click on your name and click on Security Credential.Click on Access Keys, Generate a New one and Store it somewhere safe.

10. Once Spark is Configured as mentioned in report, do the following steps:

for 1 Node Run following command:

```
./spark-1.6.1/ec2/spark-ec2 -k Spark -i Spark.pem --slaves=1 --instance-type=c3.large --ebs-vol-size=30 --spot-price=0.04 launch spark_10gb
```

for 16 Node Run following command:

```
./spark-1.6.1/ec2/spark-ec2 -k Spark -i Spark.pem --slaves=16 --instance-type=c3.large -m c3.4xlarge --spot-price=0.15 launch spark_100gb
```

11. Change the configuration files as following in Master node launched from Main node:

1.ephermal/conf/core-site.xml

```
<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/mnt/raid</value>
</property>

<property>
  <name>fs.default.name</name>
<value>hdfs://MASTER's_PUBLIC_DNS:9000</value>
</property>

<property>
<name>fs.defaultFS</name>
<value>hdfs://MASTER's_PUBLIC_DNS:9000</value>
 </property>
</configuration>
```

2.ephermal/conf/hdfs-site.xml
Changes are only in following properties:

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
 </property>

<property>
  <name>dfs.data.dir</name>
  <value>/mnt/raid/</value>
</property>
```

3. spark/conf/core-site.xml
Changes are only in following properties:

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/mnt/raid/</value>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://MASTER's PUBLIC_DNS:9000</value>
</property>
```

12. Once Configuration are done, copy it in all the slaves:
```
./copy-dir ~/ephemeral-hdfs/conf/
./copy-dir ~/spark/conf/
```

13. Stop all the services with following command:
```
ephermal-hdfs/bin/./stop-dfs.sh
spark/sbin/./stop-all.sh
```

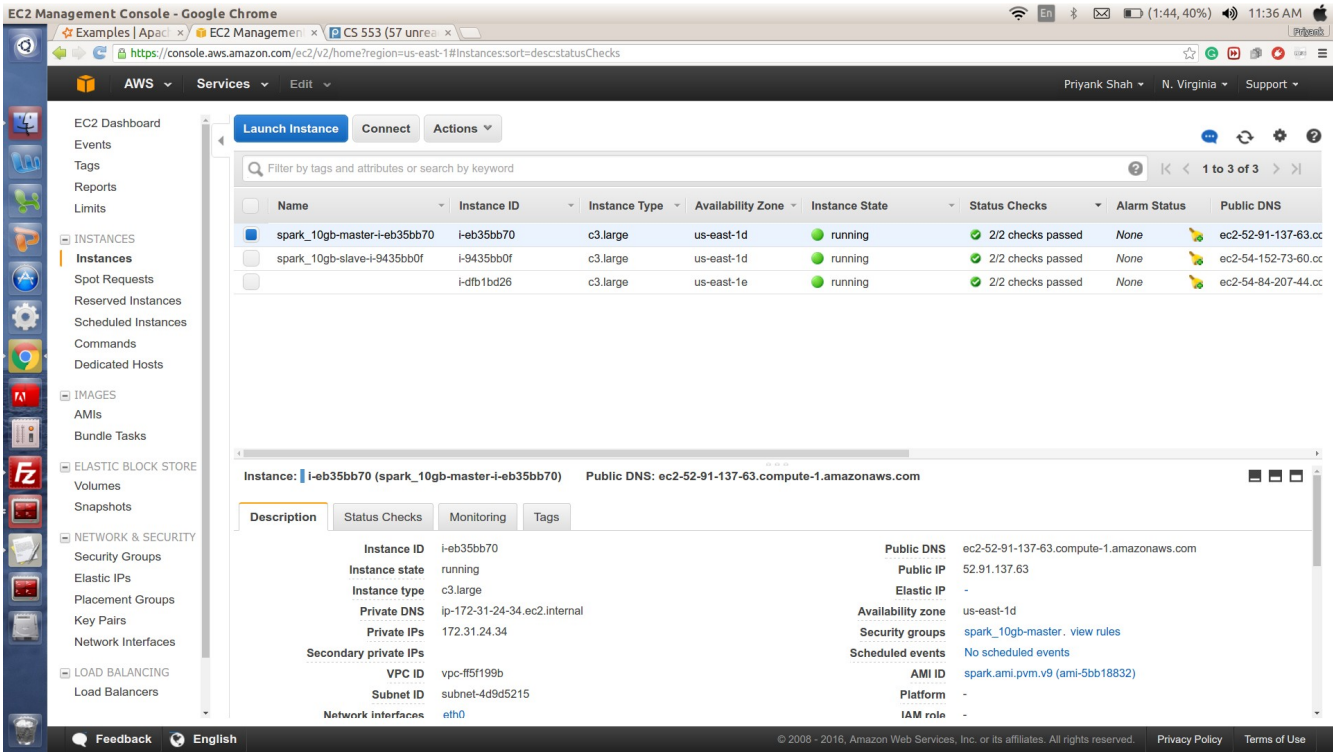
14. Format the namenode:
```
ephermal-hdfs/bin/.hadoop namenode -format
```

15. Start all the Service with following commands:
```
cd spark/sbin/ → ./start-all.sh
cd ephermal-hdfs/bin/ → ./start-dfs.sh
```

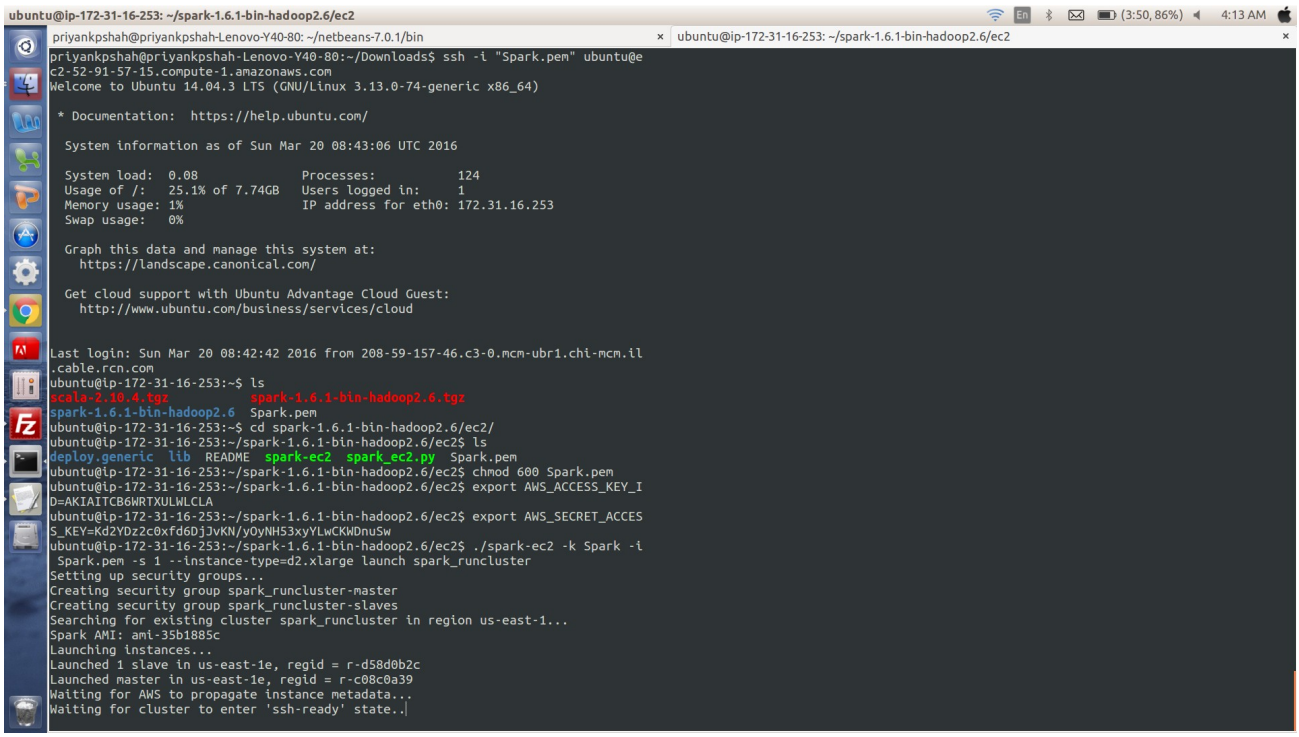
ScreenShots:

10 GB Experiments:

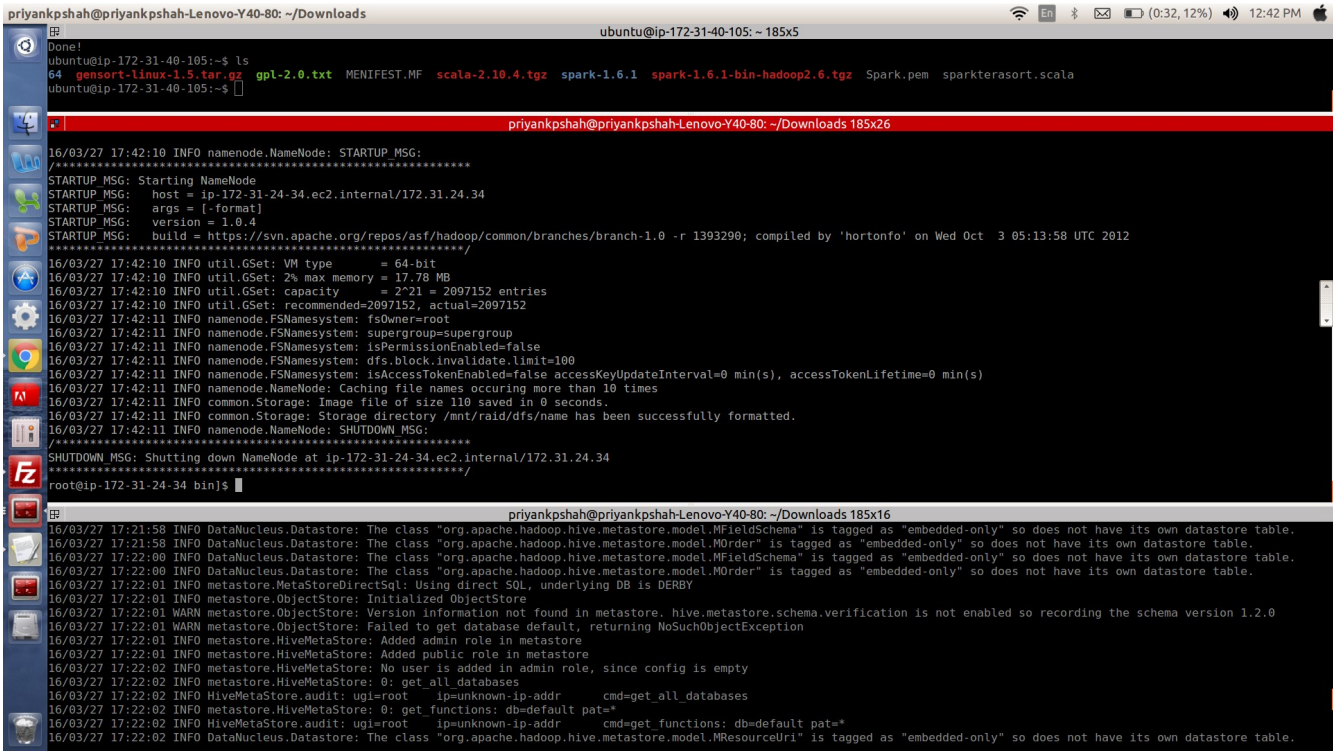
Spark Cluster on AWS:



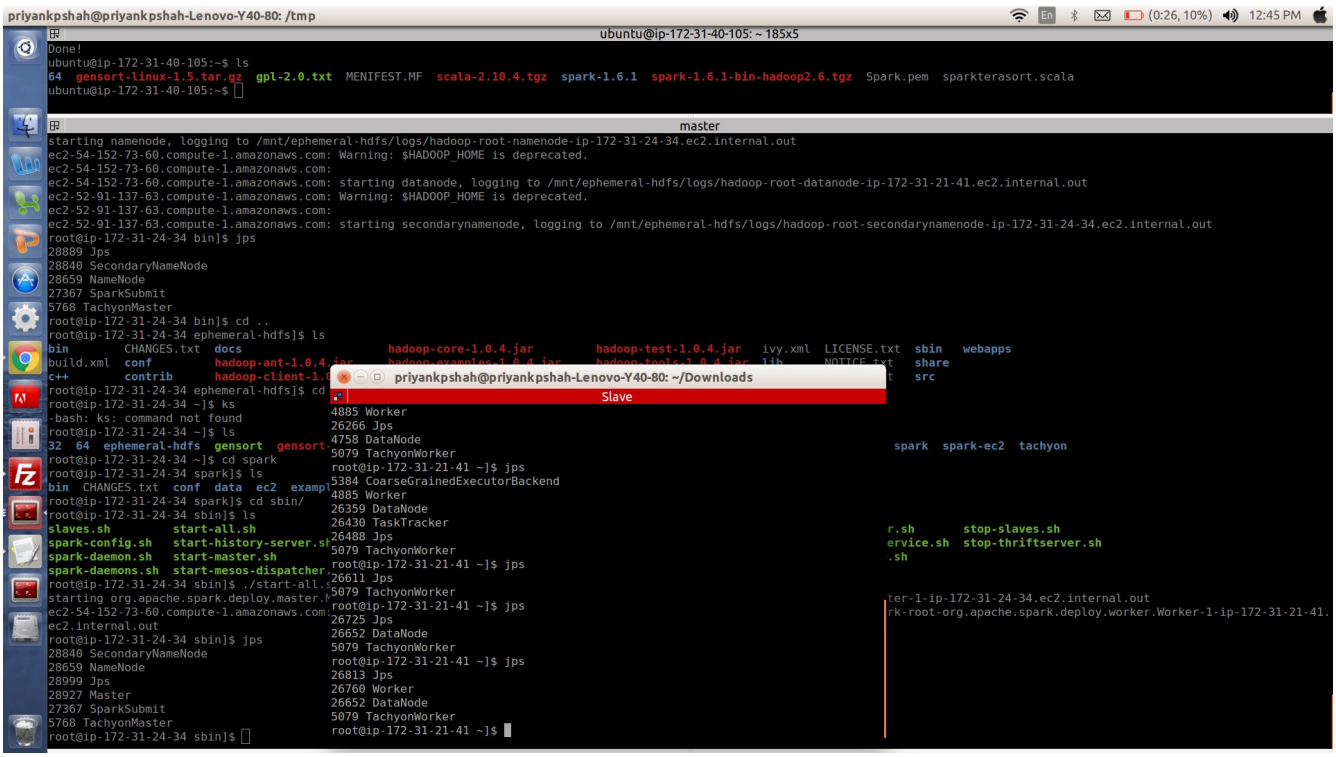
Cluster Formation Started:



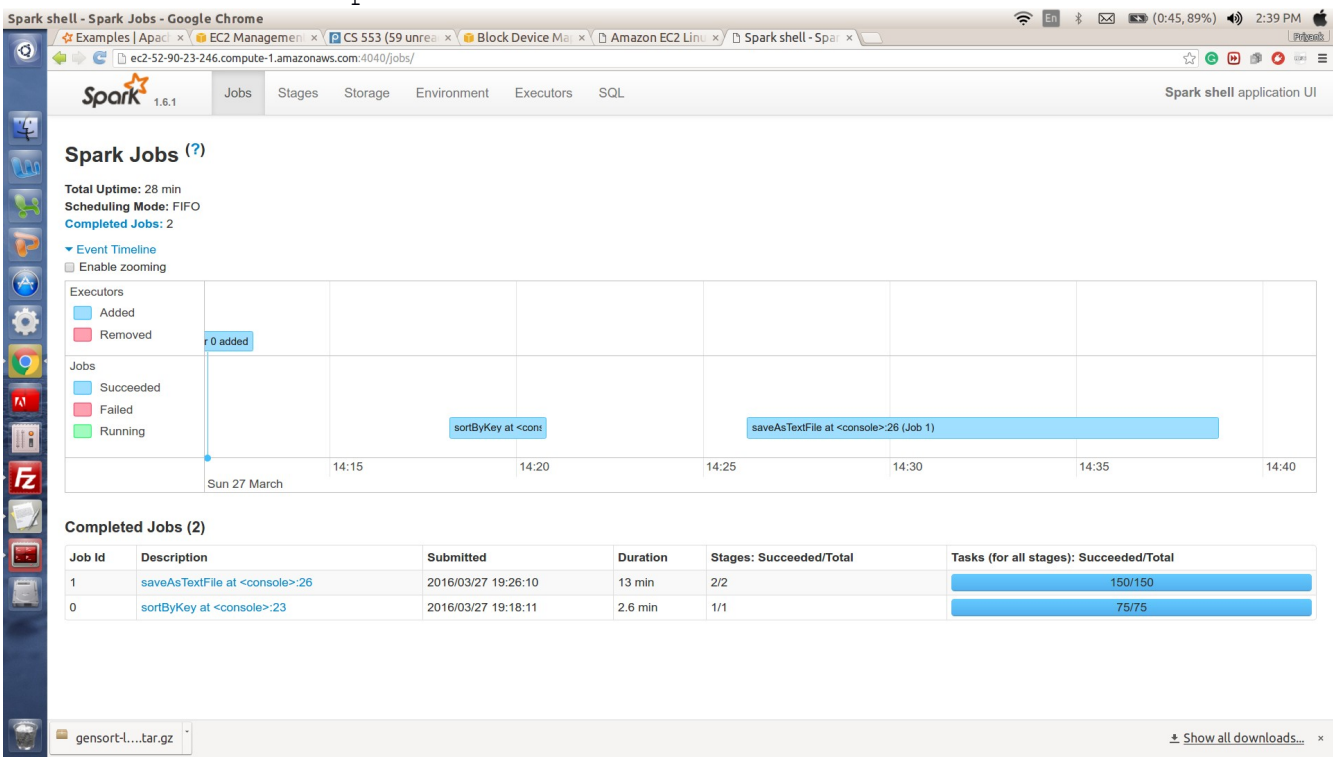
Namenode Format:



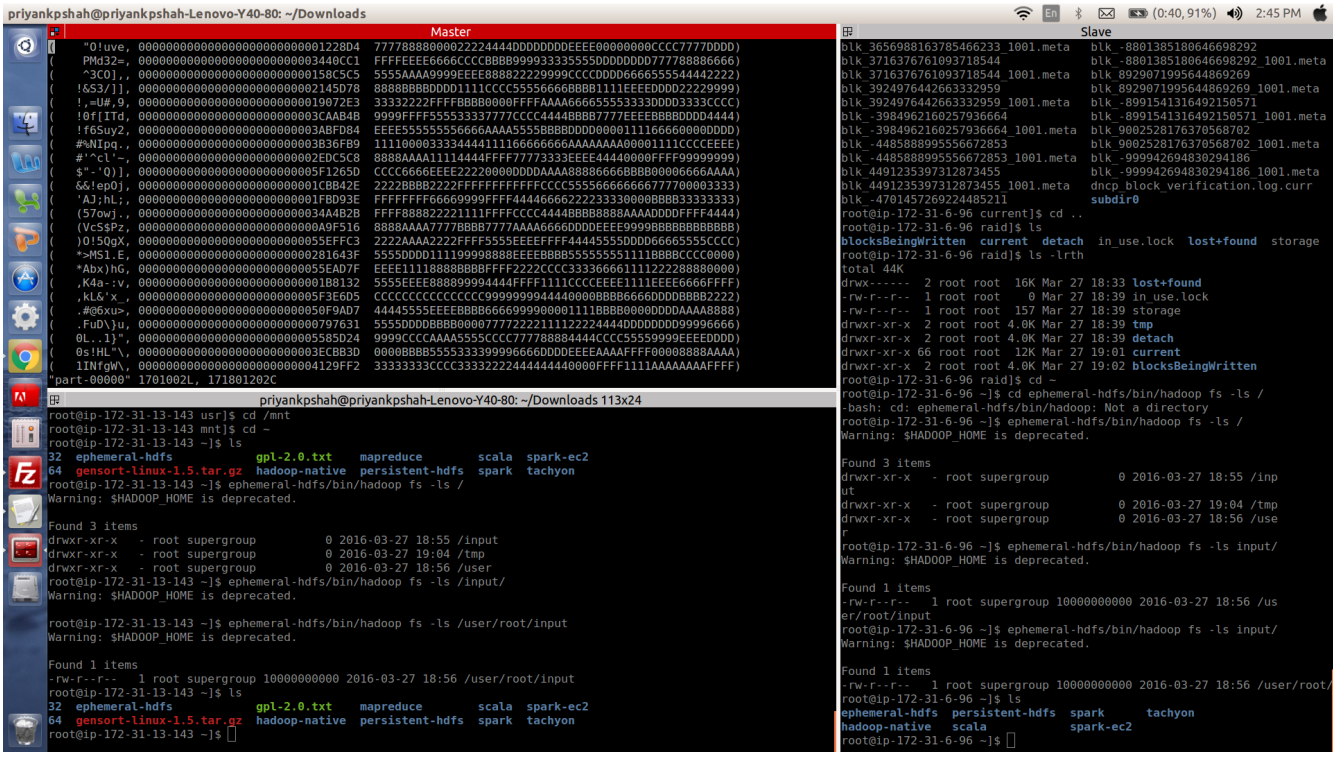
Service Started on master/slave:



10 GB Job Completed:

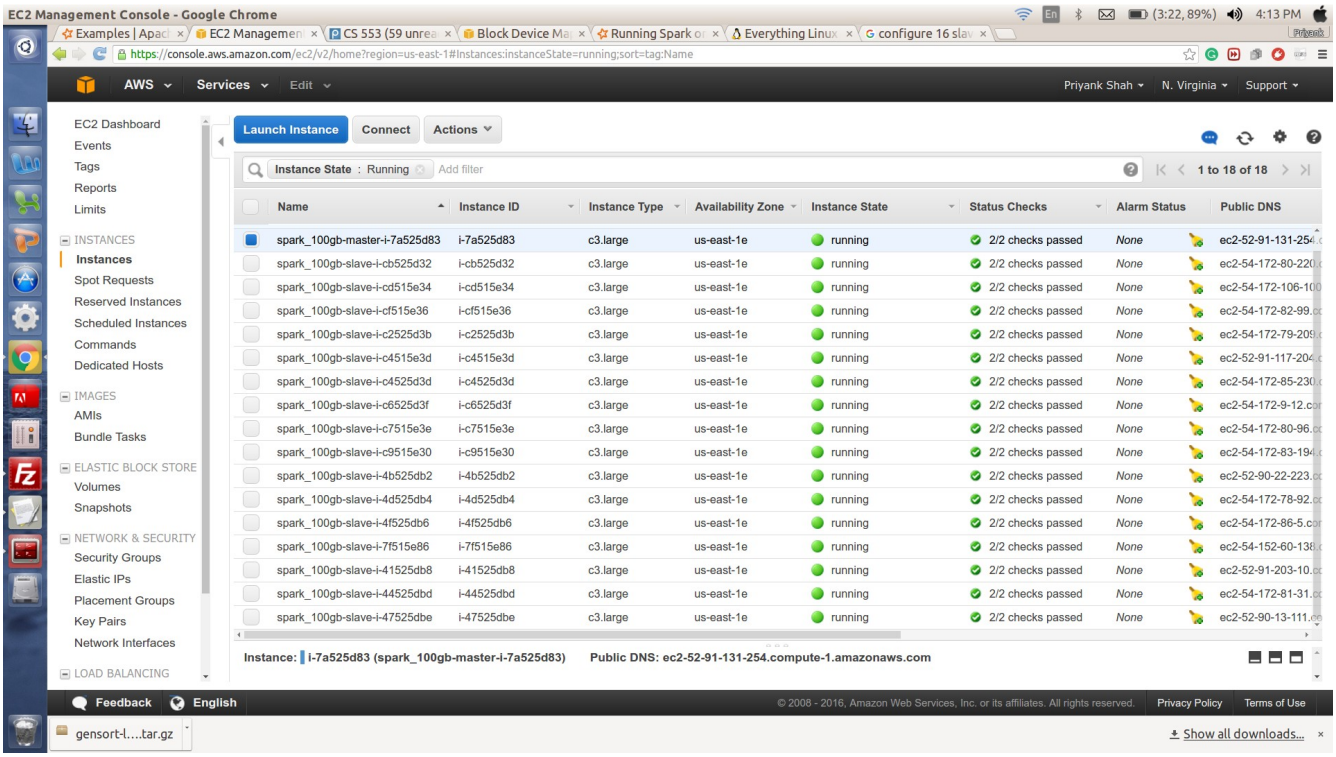


10 GB output:



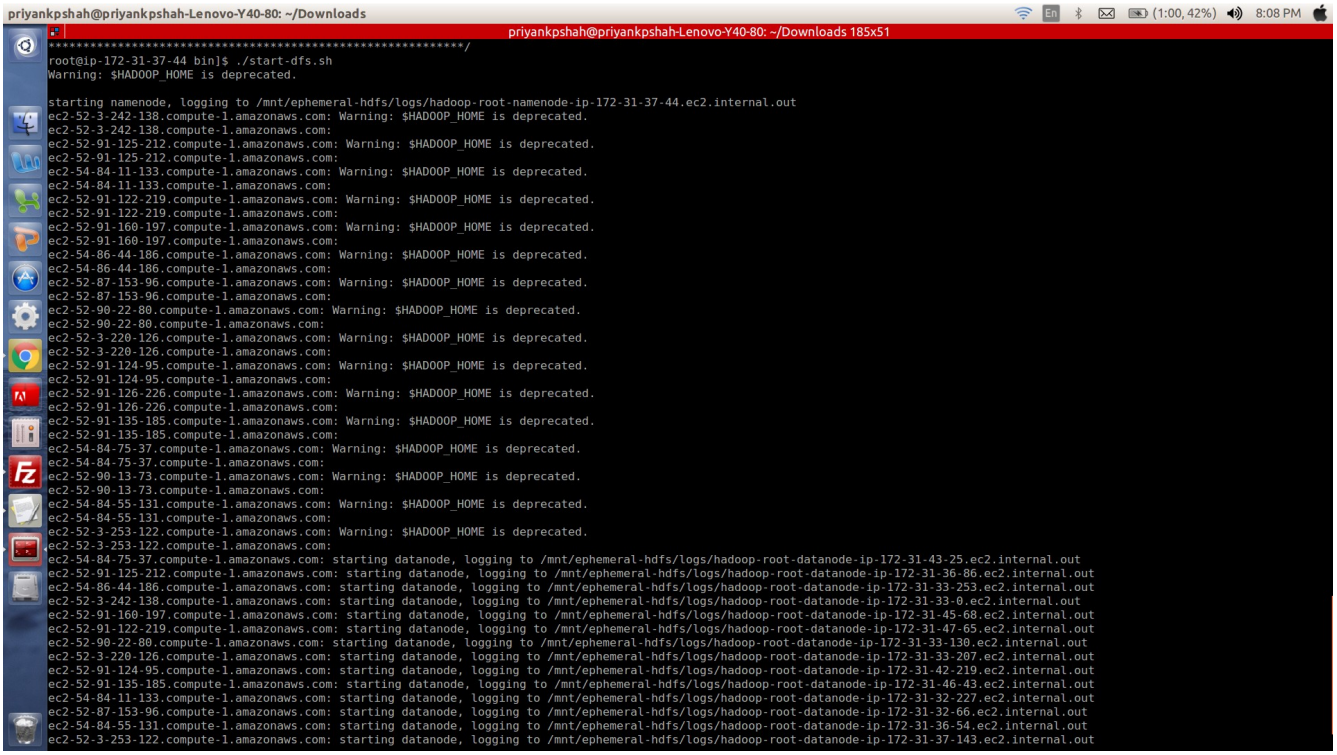
100 GB Experiment:

Amazon AWS Cluster:

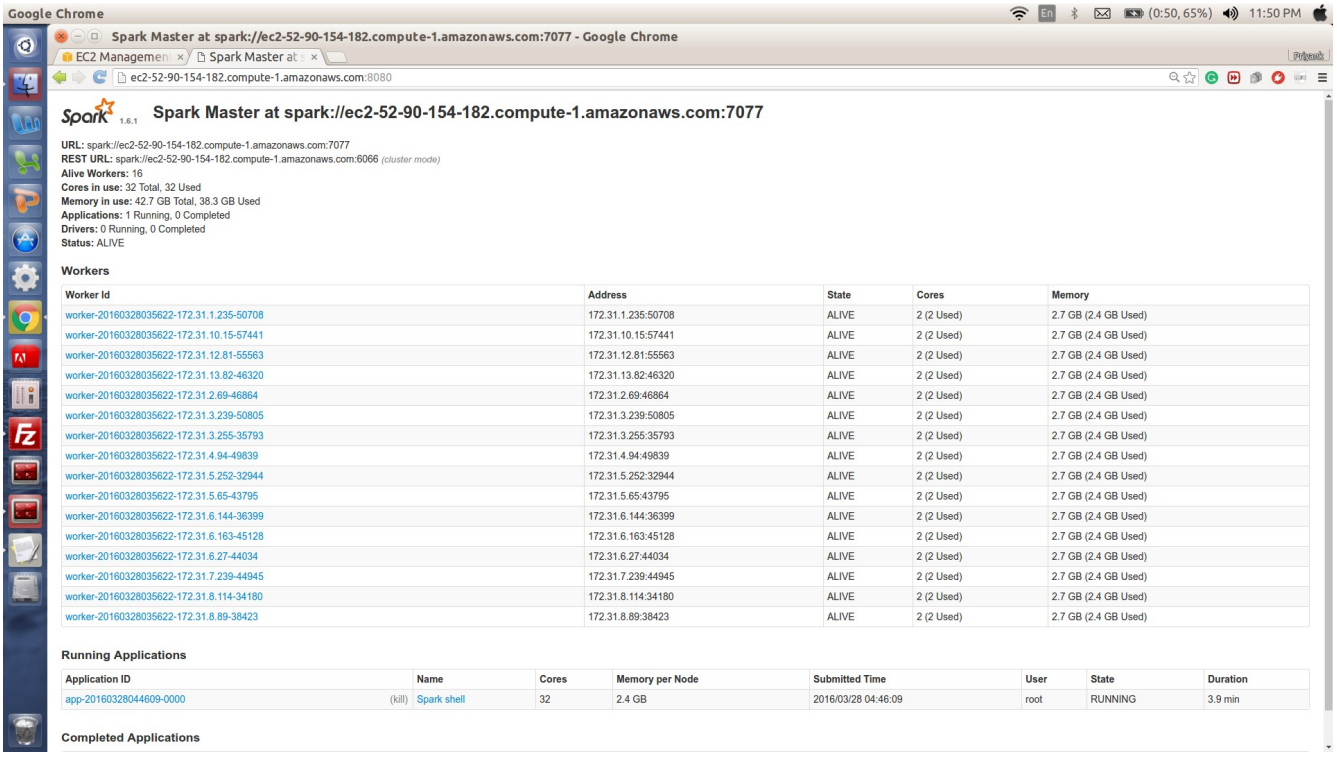


Cluster Configured and Started:

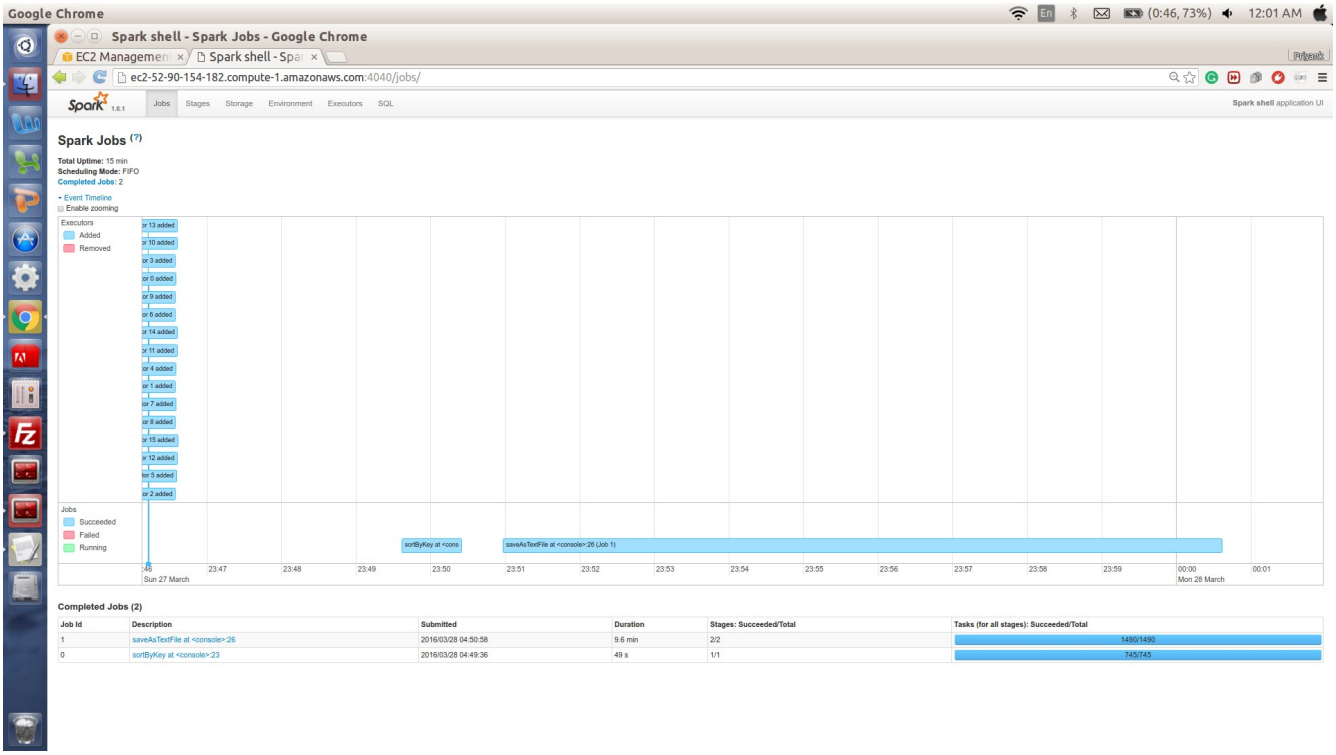
Datanode Running on 16 Slaves:



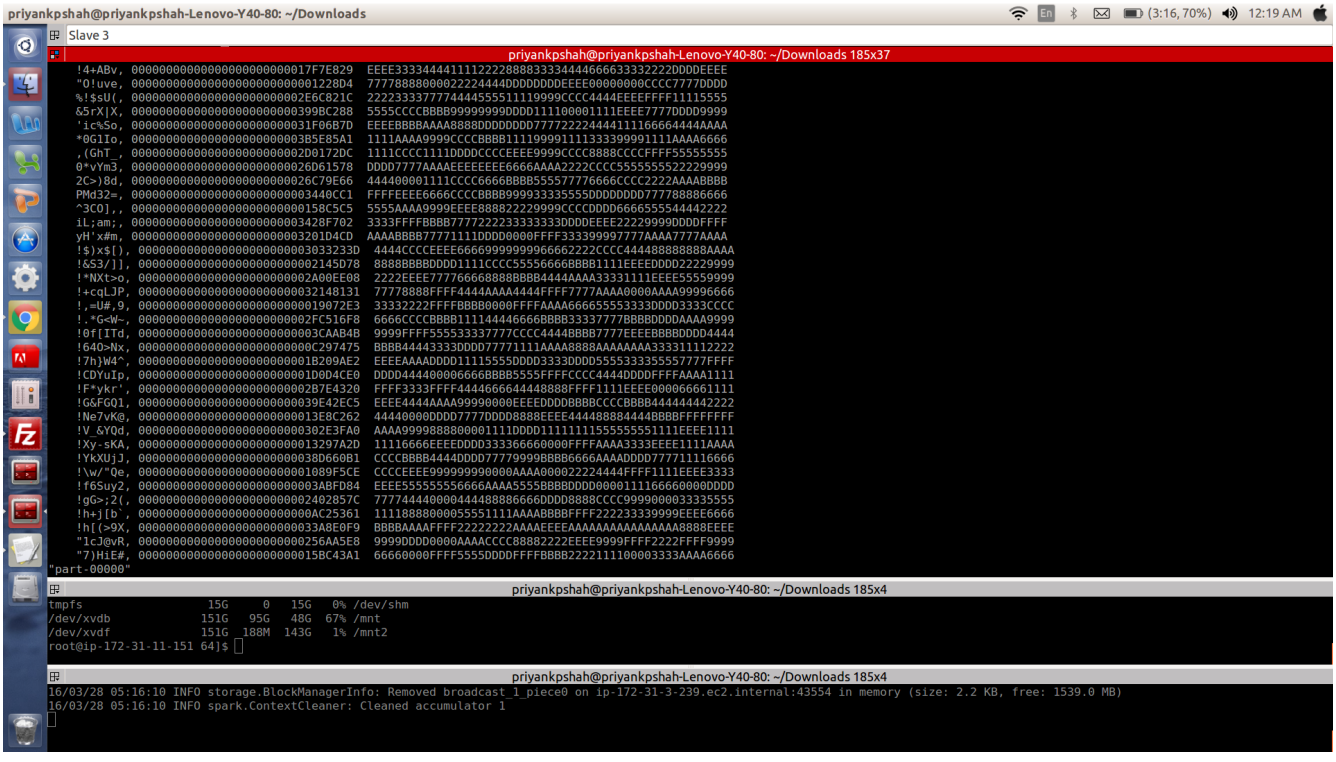
Workers:



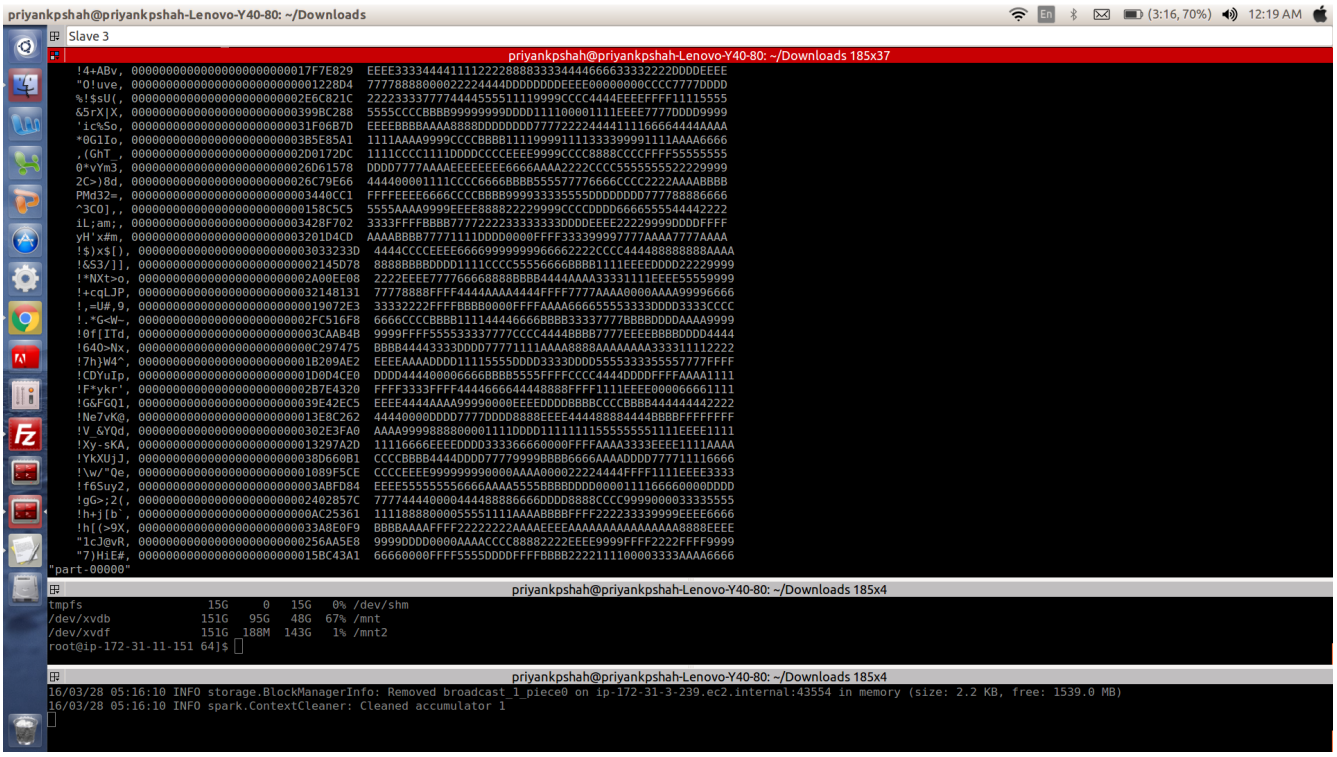
Job Finished on 16 nodes:



Output from first File:



Output from Last File-Last Lines:



Problem Faced in Spark:

1.Not as a problem but, While launching the cluster , I was not aware of how to add EBS volume, how to launch spot instance instead of on demand instance.

2. Scala language is new,so It was bit difficult to implement logics. By referring examples from its website, found work flow and implemented.

3. Uploading file on hadoop file system was bit tough for me, classmates suggestions help me to understand how spark works.

Methodology:

- In Scala, we will read the file and and store in val data type, basically it is used for fixed data type,whose value can't be change at any point while executing.
- We will differentiate key and value with substring function.
- Map function will store data as (Key,Value) pair, which we can sort by sortByKey() function.
- We will write it back in the output file at desired path.

3.Shared Memory Benchmarking:

Shared Memory is basically program written in java for sorting the data generated by gensort. Purpose of writing and executing this program is to understand the concept of multi-threading and how execution of large files affected w.r.t. time complexity and space complexity. Goal of developing this program is to optimize it as much as possible, to achieve least execution time.

Methodology:

- In this program, I am taking value of threads as an input from user. User of this program have to manually generate the data in side /mnt/raid/ path with the name of file as "input" and empty file as "output" on given path. Temporary files will be generated on system path.
- I am using quick sort approach for sorting the key. First of all program will fetch the data in chunks of 20 Mb and pass it to quicksort function which applies quick sort.
- After performing quick sort on chunk it will be stored as temp file inside system's tmp folder. Once quick sort is performed and tmp files are generated, it will gather tmp files and sort it with merge sort which check one by one line of each file and put the in ascending order.
- In order to run the program for 10 GB dataset, I have increased to system limit of total number of open files to 10000 for safer side. Which can be check with "ulimit -a" command and can be set with "ulimit -n LIMIT" command, where LIMIT is the number of files you want to open at a time.
- Temp files will be automatically deleted when system exits the program run.

2. Performance Evaluation:

1 node Comparison:
Data Size: 10 GB

	1 Node Time	Speed Up	Throughput
Hadoop	836 Sec	1.00	97.990 Mbps
Spark	936 Sec	1.11	87.521 Mbps
Shared Memory	940 Sec	1.12	87.149 Mbps

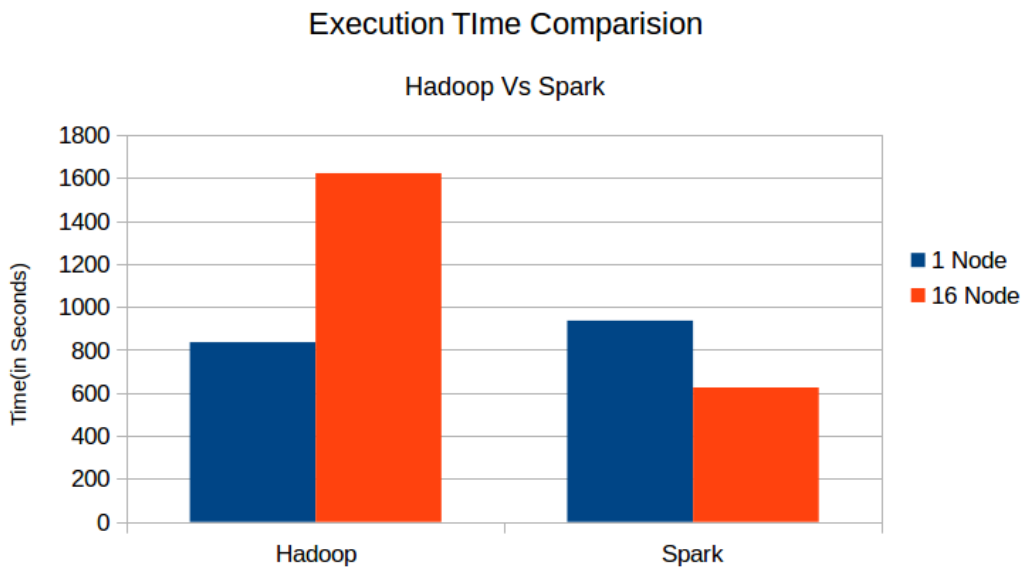
16 node Comparison:
Data Size: 100 GB

	Node Time	Speed Up	Throughput
Hadoop	1622 Sec	1.00	505.055 Mbps
Spark	625 Sec	0.38	1310.720 Mbps

Shared Memory:

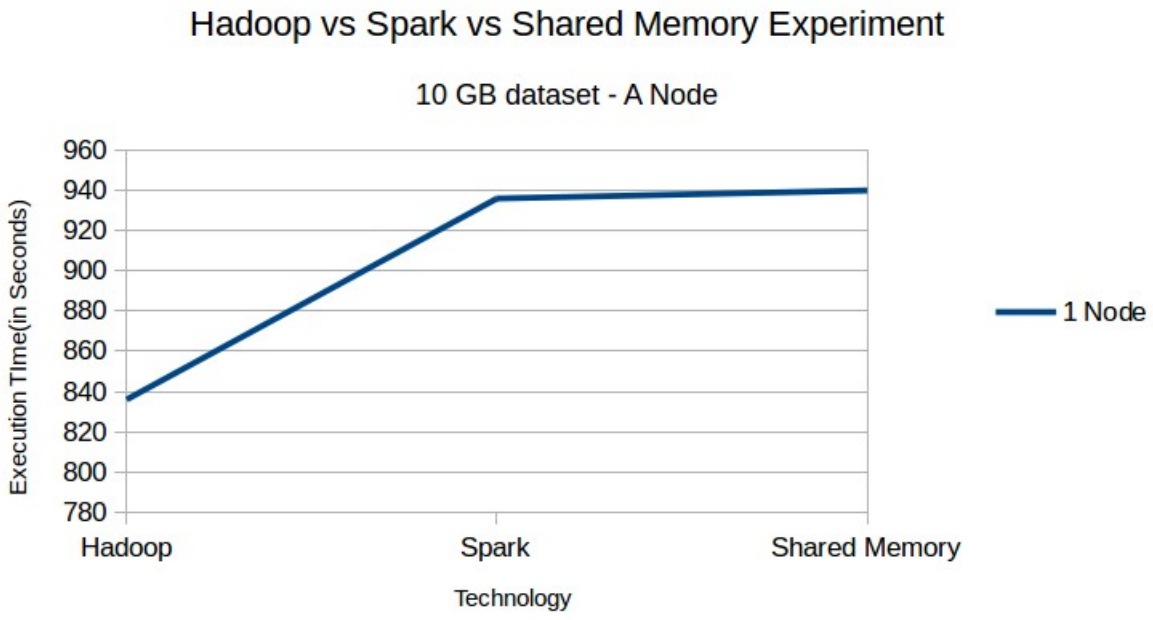
	Node Time	Speed Up	Throughput
1 Thread	940 Sec	1.00	87.149 Mbps
2 Thread	907 Sec	0.95	90.320 Mbps
4 Thread	946 Sec	1.08	86.598 Mbps
8 Thread	970 Sec	1.10	84.454 Mbps

Spark Vs Hadoop:

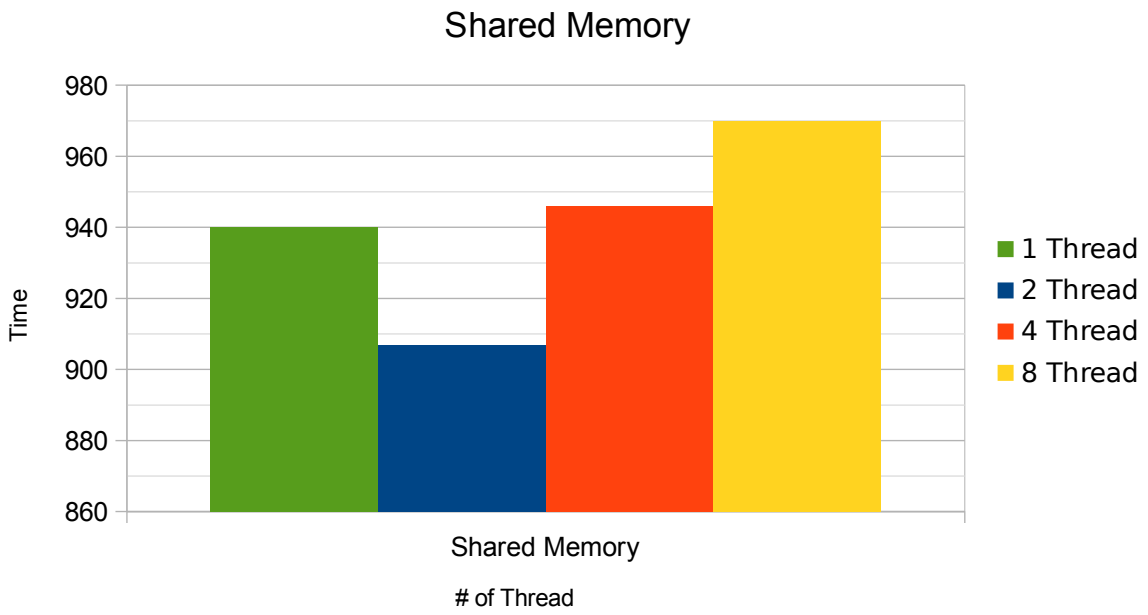


Hadoop Vs Spark Vs Shared Memory:

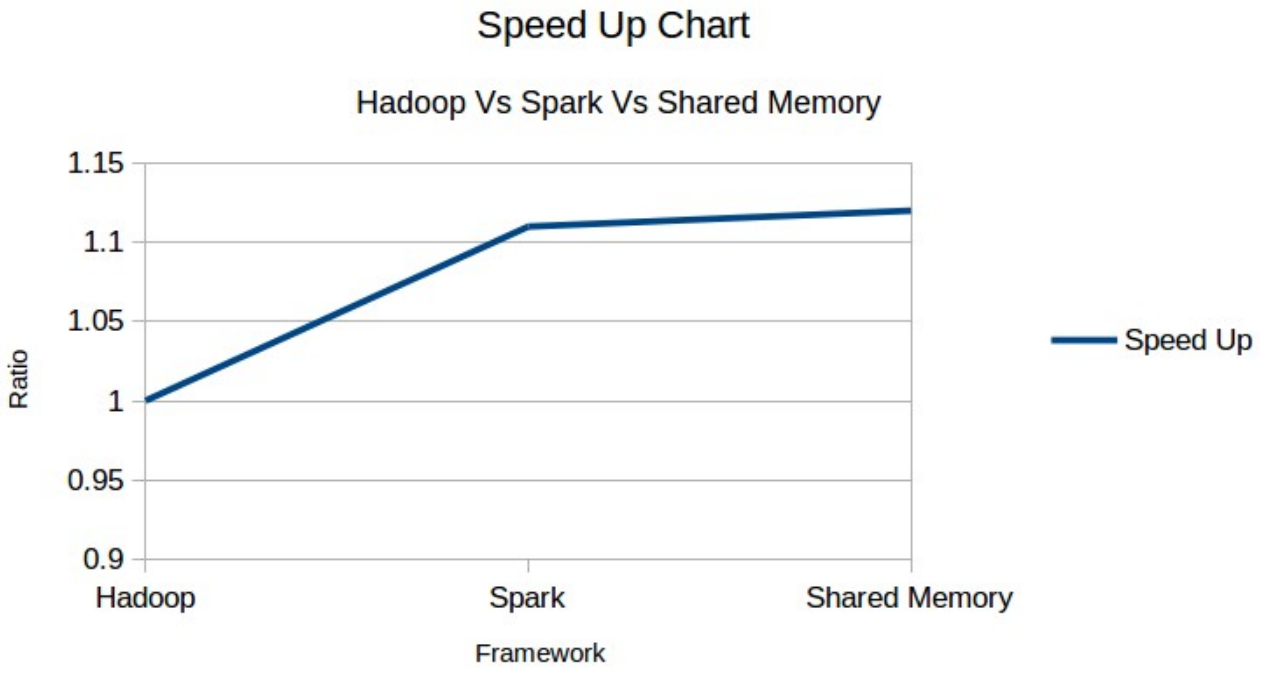
Below Chart shows the comparison between Hadoop , Spark and Shared Memory for 10 GB Data Set:



Shared Memory Performance Comparison:



Speed Up Comparison:



Conclusion:

- From this experiment I conclude that, In Spark Program runs on 16 Node much faster than 1 node.
- Initially, in 10 GB experiment Spark program was bit slower but over the time Spark results outnumbered the Hadoop Results.
- In Shared Memory, program with 2 thread runs faster than the program for 1 thread. Program for 4 node runs slower than the 2 nodes because c3.large have only 2 cores. Program for 8 thread runs even more slower than the 4 thread programs.
- Overall, Hadoop took least time for 10 GB Experiment. Spark took least time for 100 GB experiment. Shared Memory with 2 thread took the least time in 1,2,4 and 8 thread experiments.
- Overall Shared Memory is the best in case of 10 GB. It has the highest speed up ratio.
- Spark runs 100 times faster than the Hadoop on memory and 10 times faster when running with the disk. So at 100 Node Spark will be more efficient in management and execution.
- At 1000 nodes, Same will happen, Spark will perform better than the hadoop but required much more memory for intermediate data. So its better to with Hadoop for 1000 node.

References:

1)https://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/mapred/Mapper.html
2)https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html
3)http://www.itadmintools.com/2012/10/throughput-calculator.html
4)http://tldp.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/x4733.html
5)http://stackoverflow.com/questions/1318980/how-to-iterate-over-a-treemap
6)http://tldp.org/LDP/solrhe/Securing-Optimizing-Linux-RH-Edition-v1.3/x4733.html