CLOUD COMPUTING

PROGRAMMING ASSIGNMENT – 2

PROJECT REPORT

1. Introduction:

This programming assignment is basically made to get used to with the Amazon Web Services (AWS) virtual cluster creation, deployment and use. Besides this main aspect, I had setup virtual cluster for hadoop and spark for single as well as multiple nodes (16 nodes). After setting up these environment, created sorting program for hadoop and spark for 10GB and 100GB of data. Next task was to implement the 10GB data sorting algorithm using any sorting method.

2. Runtime Environment Setup:

Amazon Web Services EC2

AMI – Amazon Linux (HVM), SSD Volume Type – ami-3d50120d , 2 x 16 GB

Hadoop – 2.7.2

Spark - 1.6.1

Java - 1.7.0_95

Shared-Memory:

EC2 instance type : c3-large

Hadoop part :

EC2 instance type : (master node) c3.4xlarge, (slaves node)c3.large

Spark park:

EC2 instance type : (master node) c3.4xlarge, (slaves node)c3.large

- **Details of EC2 Instances Used** :

1. Instance type : c3.large Linux
   RAM : 3.75
   Cores : 2
   Storage : 32 GB

2. Instance type : c3.4xlarge Linux
   RAM : 30
   Cores : 16
   Storage : 320 GB

3. Programming and Designing:

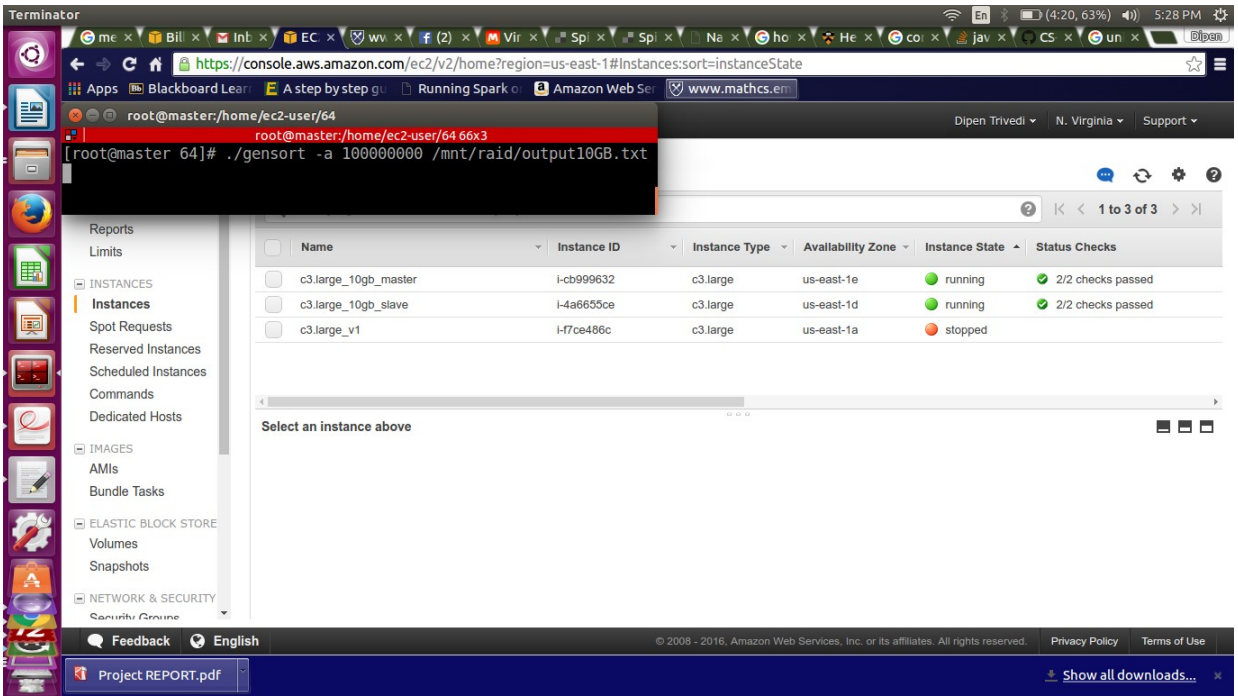
For programming and building, JAVA and Python are used for hadoop and spark respectively. As JAVA provides libraries to perform the map-reduce functionality easily on hadoop cluster and Python is good for the scripting, both were good fit for the hadoop and spark. Working in the Linux Environment made the work more reliable and efficient, as the terminal commands and methodology are same on the EC2 instances also. Used c3.large and c3.4xlarge instances to perform the tasks. Used Gensort to generate the 10GB/100GB data and valsort to check the sorted data.

4. Data Generation and Validation :

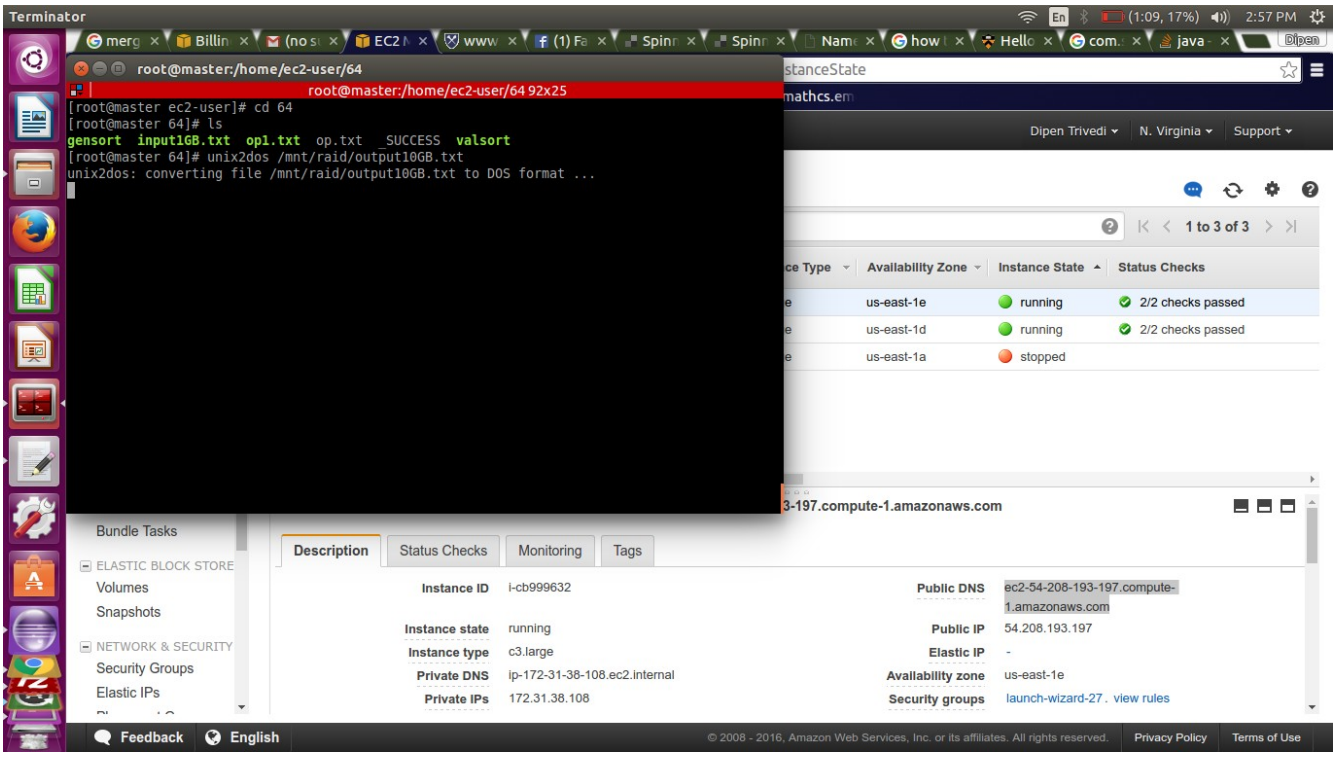
To generate 10GB and 100GB data, Gensort is used. Gensort generates the data as per the input value of number of line and the format of the file. '-a' creates the ascii file.

To validate the sorted data Valsort is used. To make the data operated by valsort, first step is to convert it the dos format as valsort only accept the input in dos format. Then, just run the valsort on the output file.

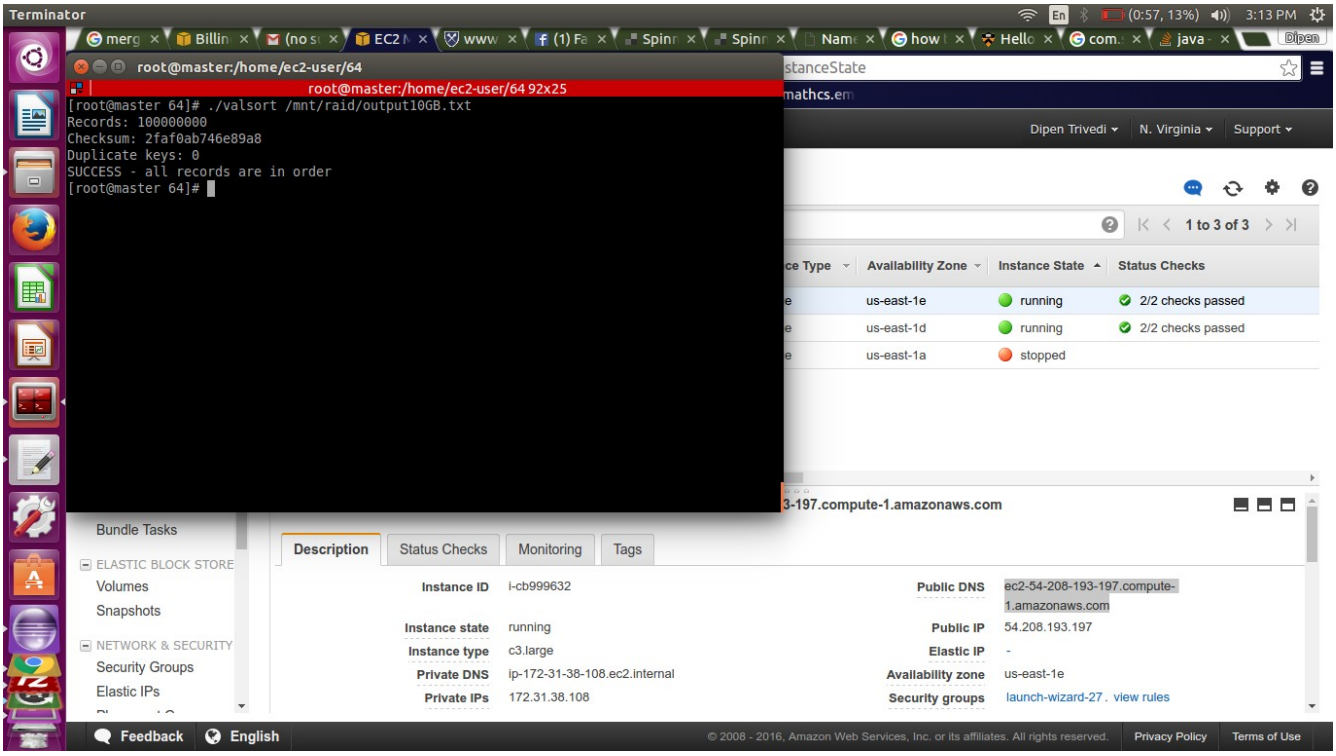
- **10GB data generation with Gensort :**



- **10GB data unix2dos :**

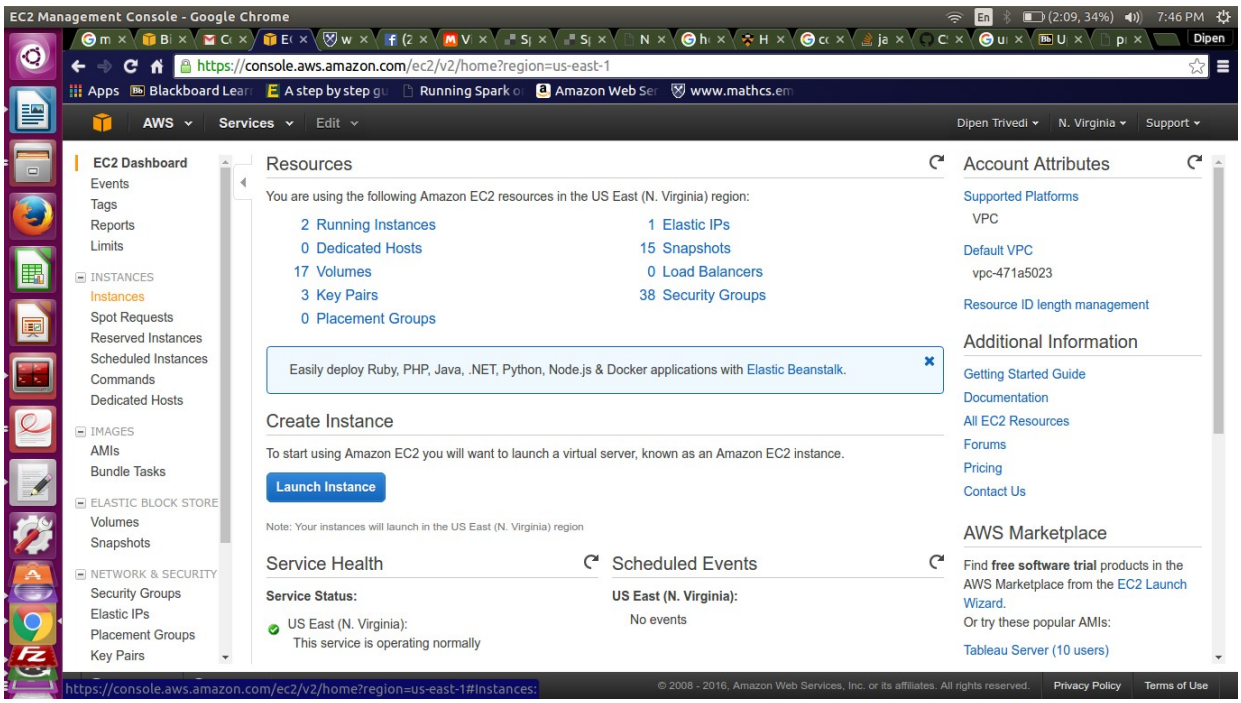


- **10GB data validation with valsort :**

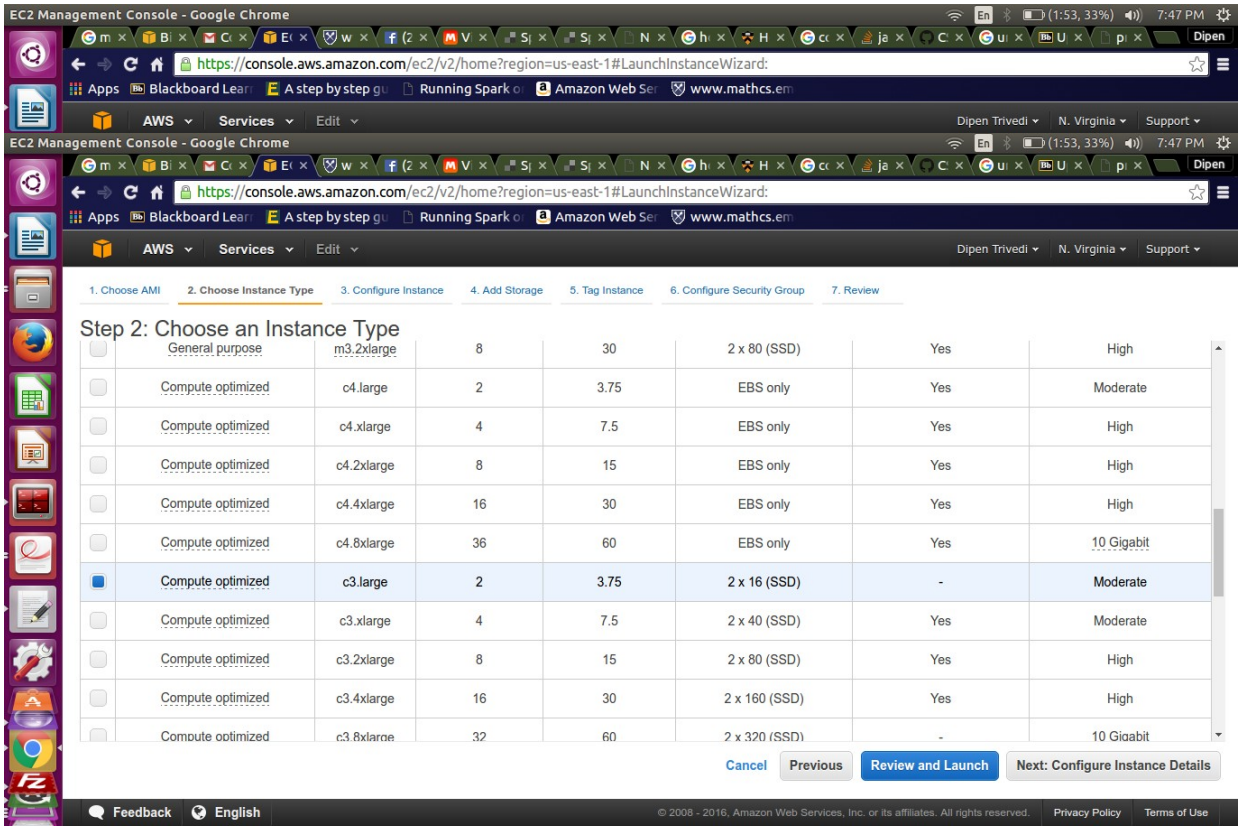


5. Virtual Cluster Launch (1-node) :

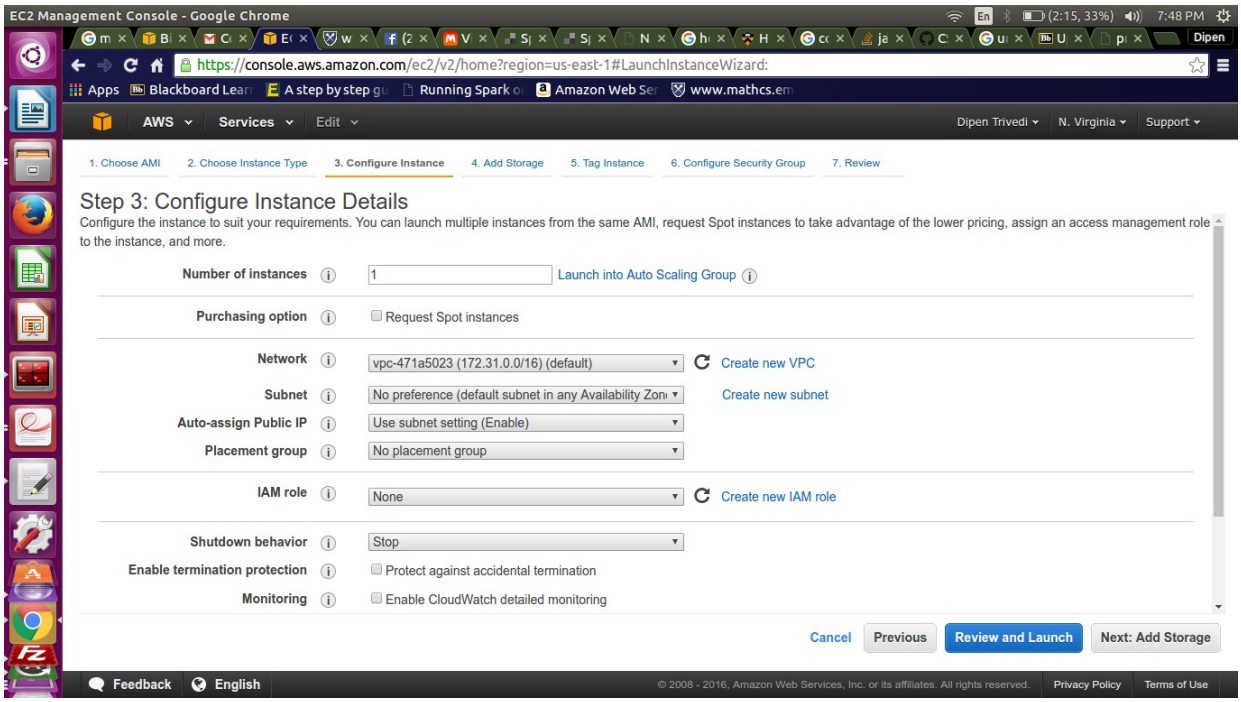
- Go to Instances in the EC2 Instances tab.

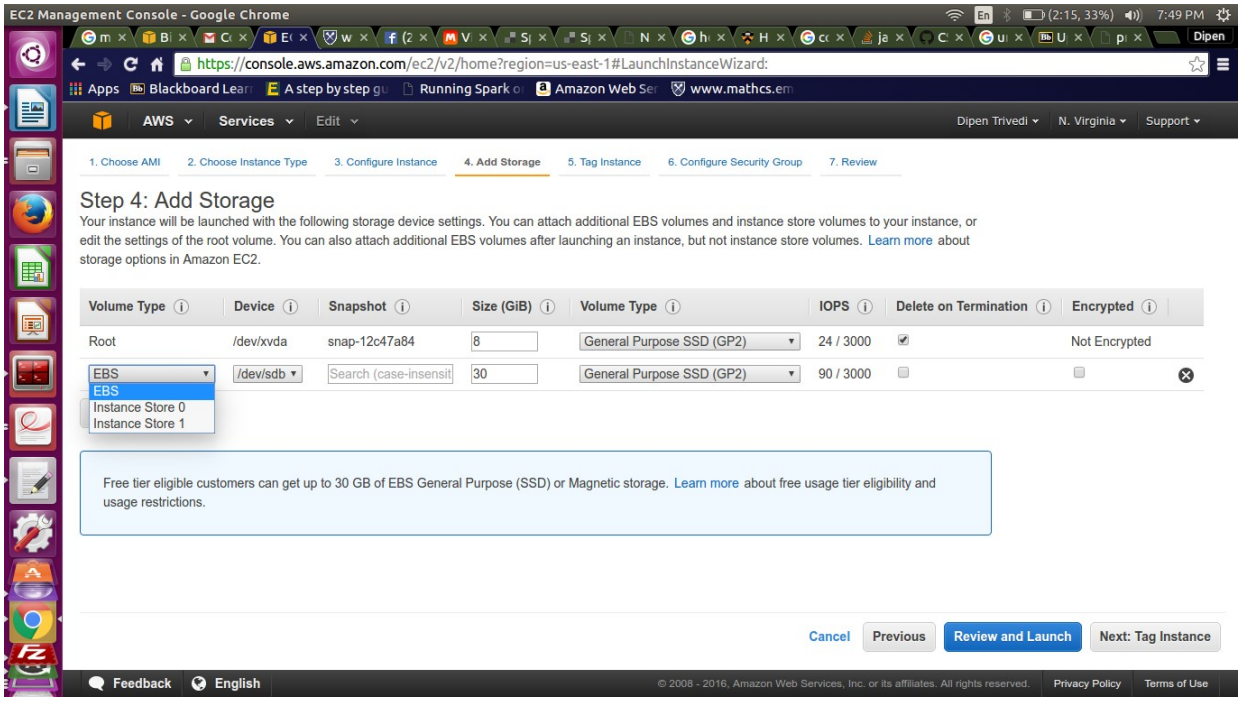


- Select launch instance. Select any instance(c3.large here) and then configure it.

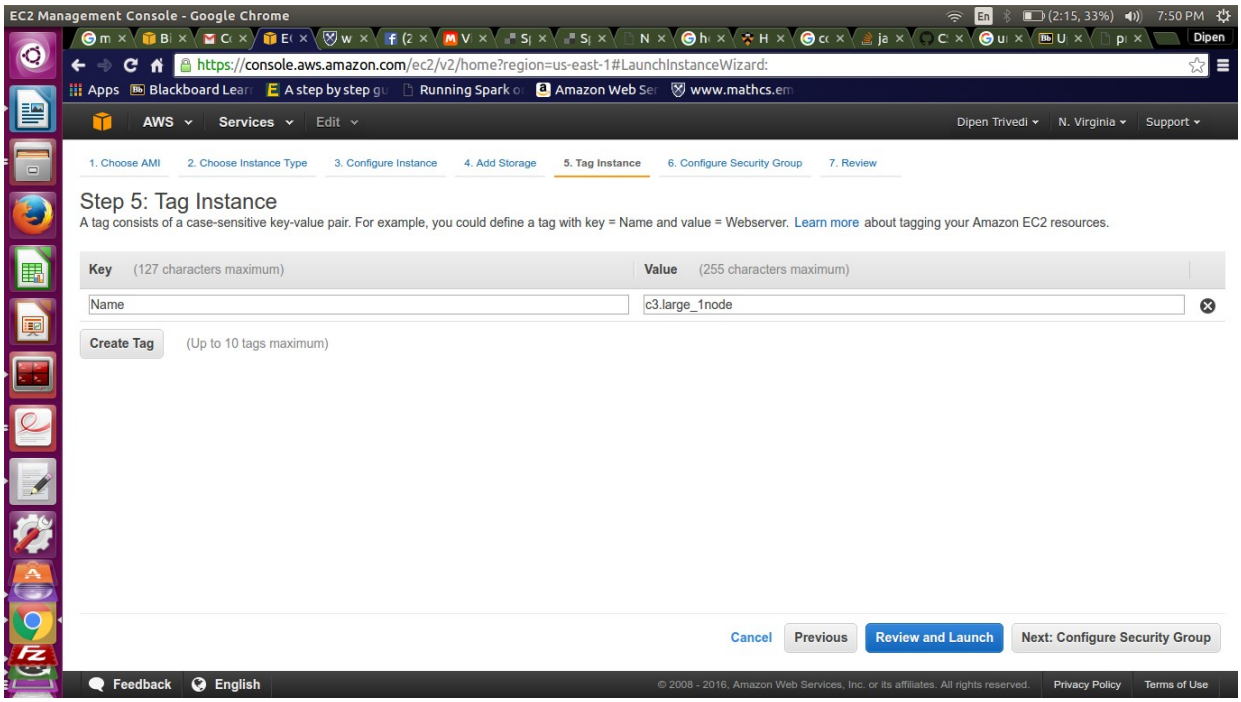


- Select instance on-demand or on-spot.

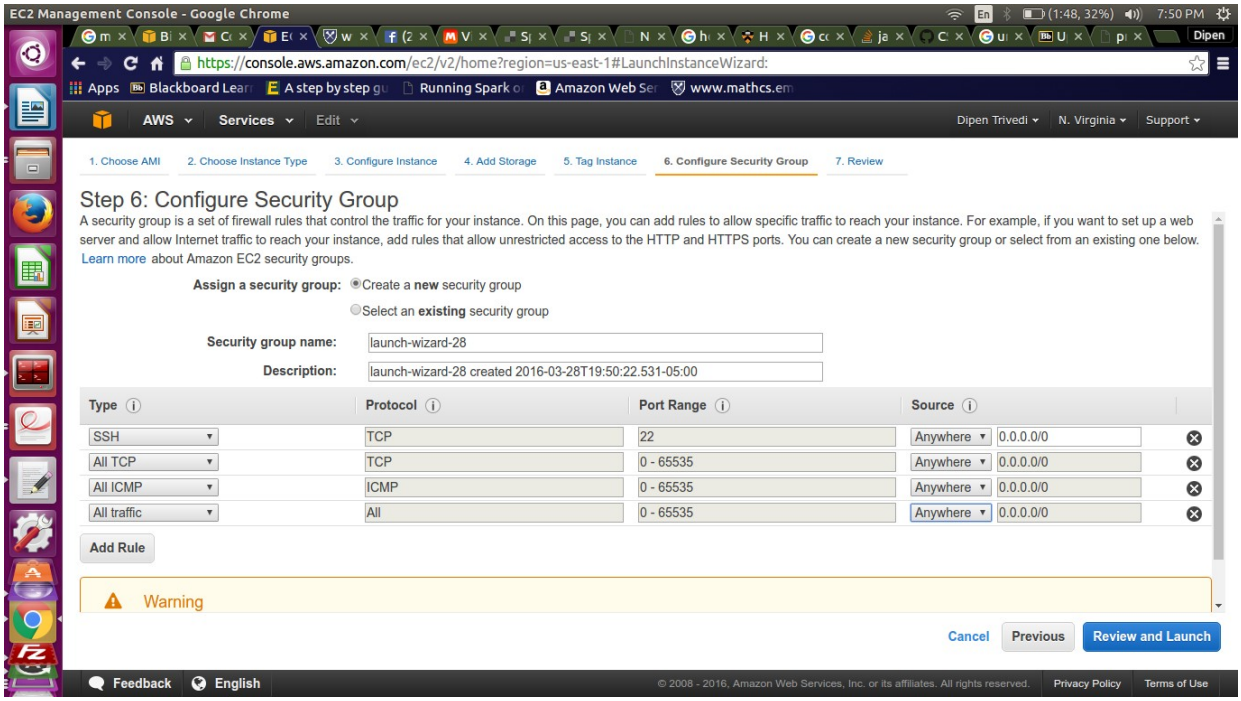




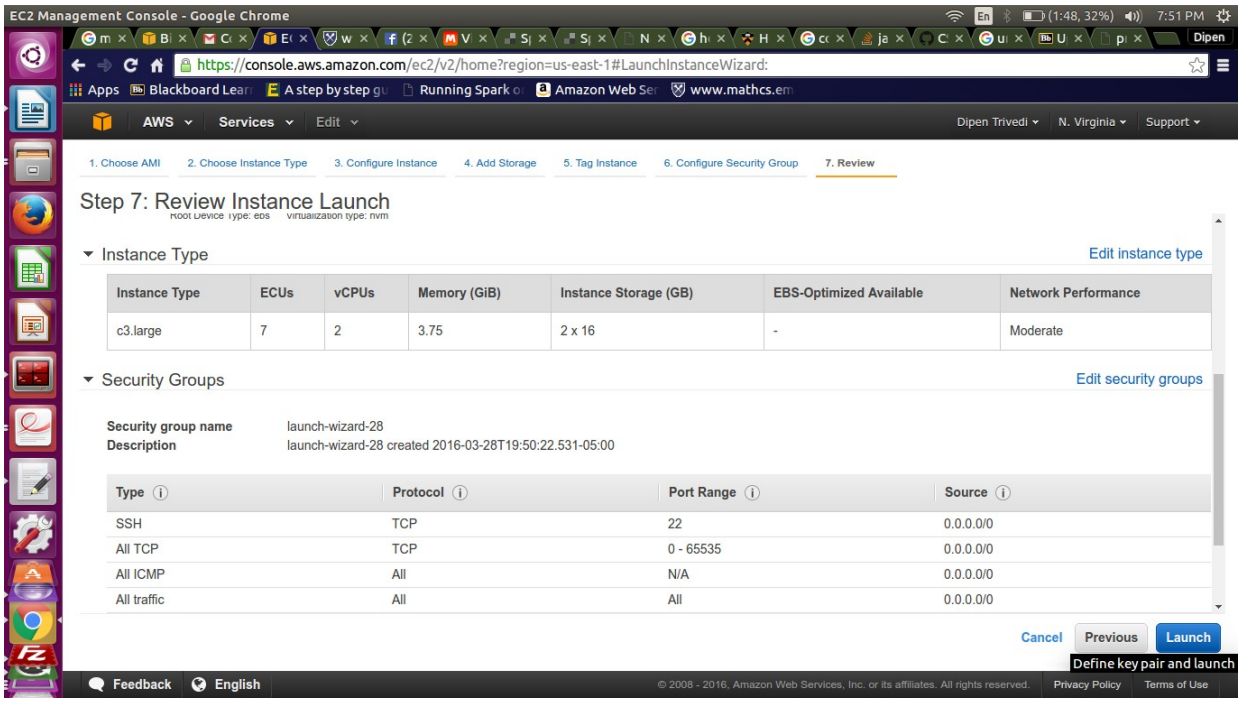
- After give label to the instance.

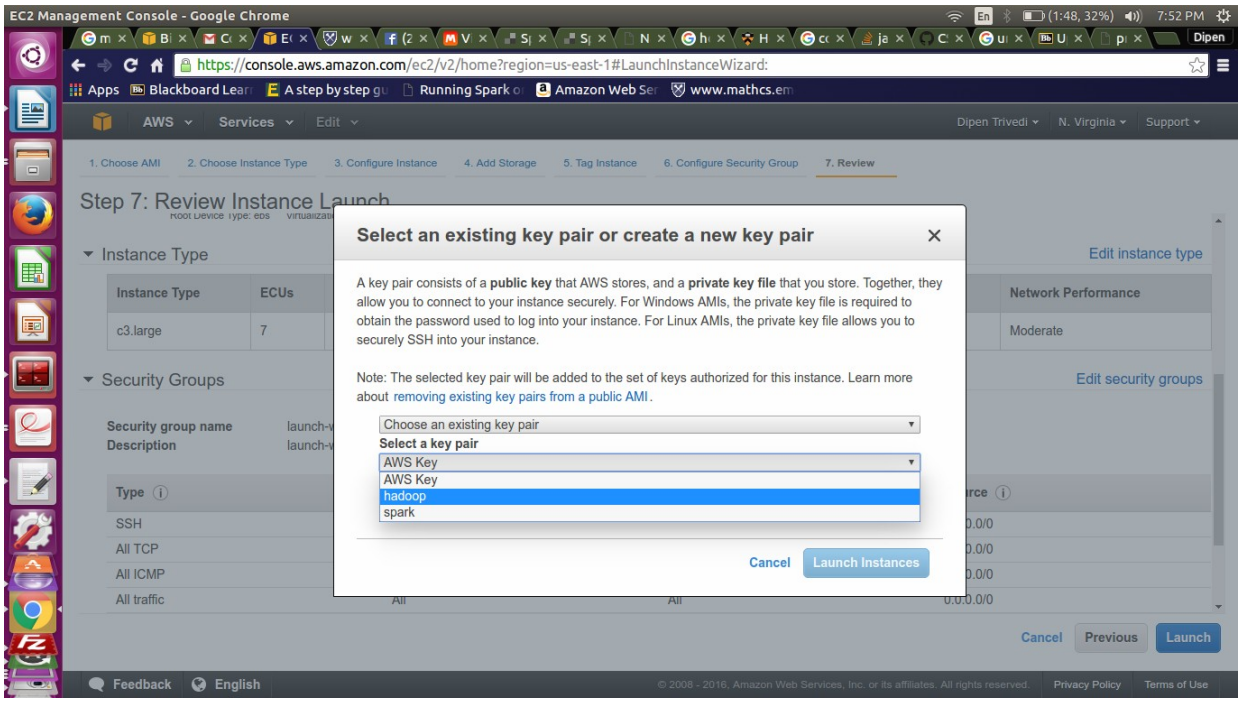


- Configure the security group to allow different traffic in network. Here. All TCP, All ICMP, SSH and All trafic rules are given to the instance.

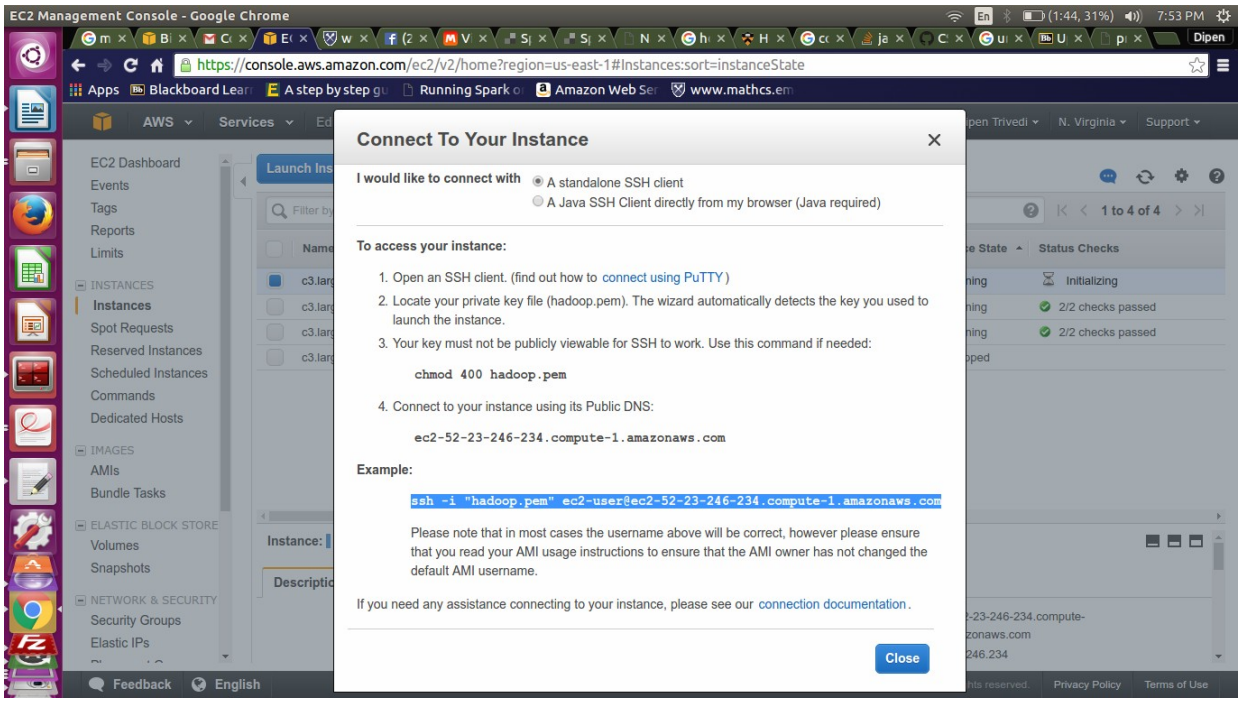


- Review and launch the instance.

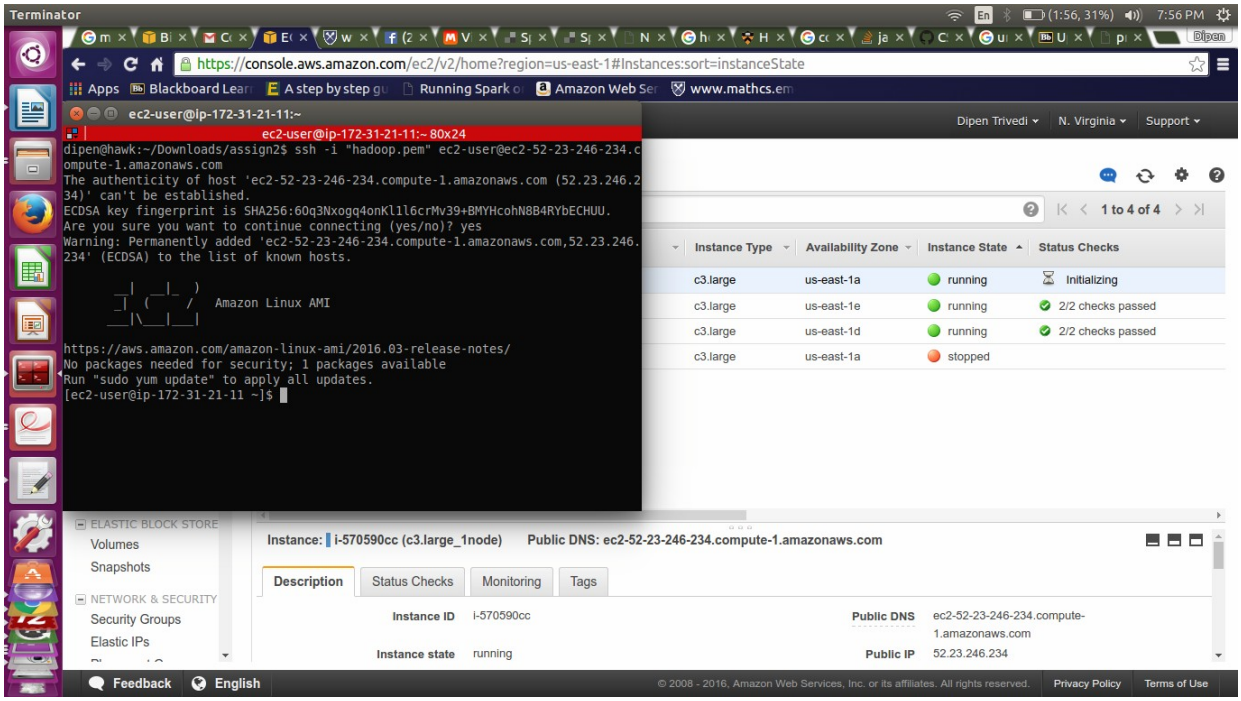




- To access the instance, take action connect and copy the SSH command to the terminal.



- Instance it connected now.



6. Hadoop Sort:

- **Hadoop Sorting Algorithm:**

  Created one java program for hadoop file system that sorts the input data.In Hadoop, Map-Reduce function takes the data in <key,value> format. To get key and value, substring the input data line by line, and assign it to the key and value respectively. Sorting is done by the frame work and after data sorting is done, Reduce function comes in the action. The main work of the reduce function is to combine the same key data, and then to add them in one <key,value> format. After that, write the data in same format as the input, so that it will be easy to validate using the valsort.

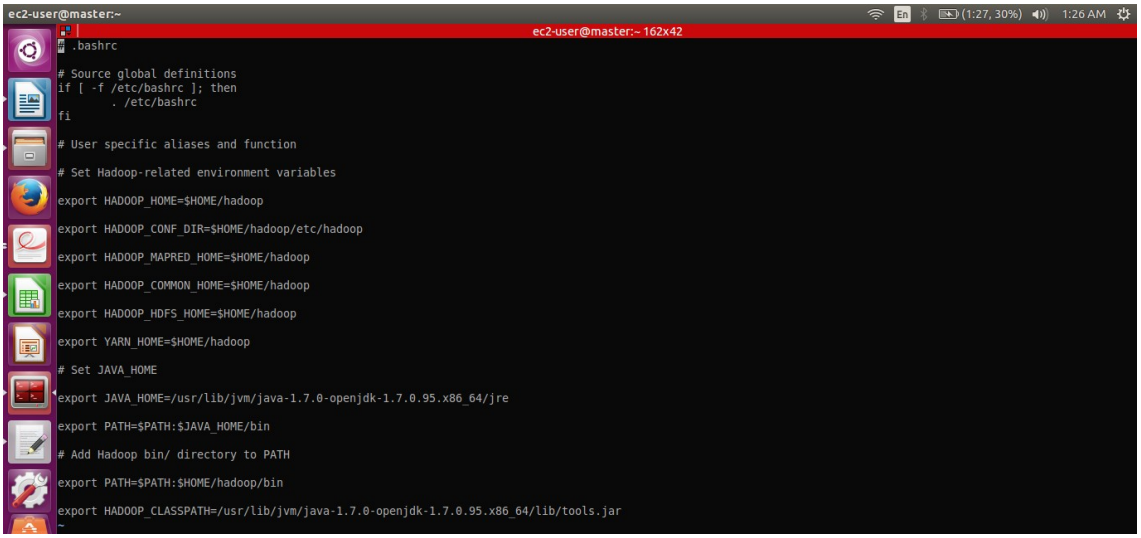
- **Hadoop Configuration:**

After starting the instance we have to run following commands to install the hadoop in the system:

- *sudo apt-get update*
- *sudo apt-get install openjdk-7-jdk*
- *wget https://archive.apache.org/dicst/hadoop/core/hadoop-2.7.2/hadoop-2.7.2.tar.gz*
- *sudo tar -xvf hadoop-2.7.2.tar.gz*

Now add hadoop and java environment variables in *.bashrc* file.

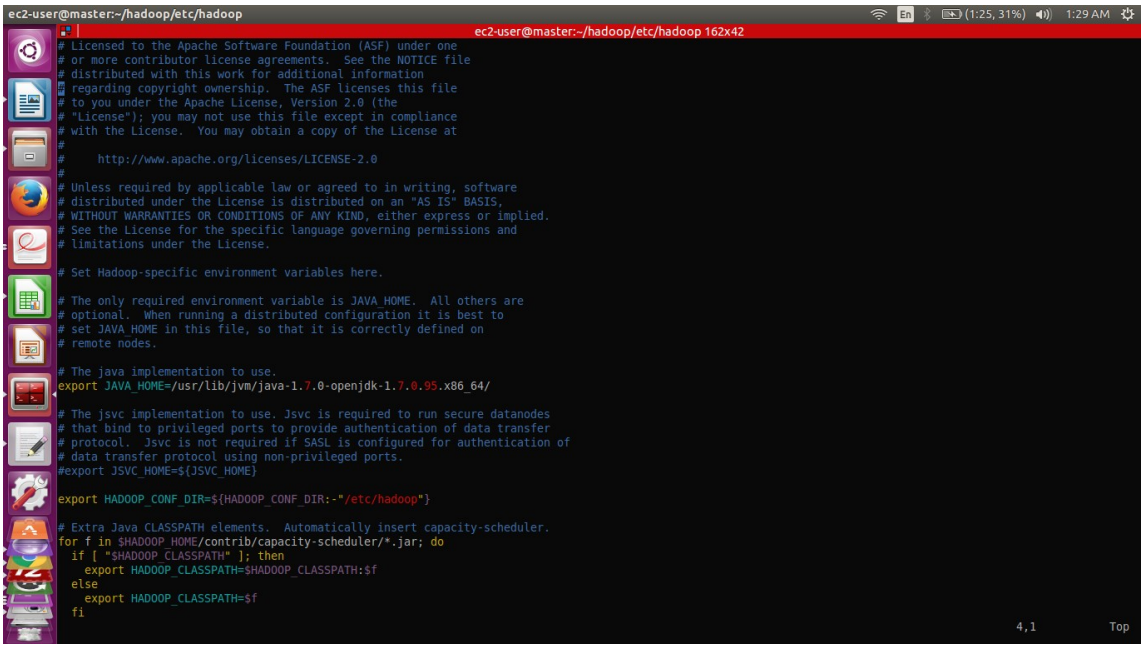
- *Sudo vi .bashrc*



- *source .bashrc*

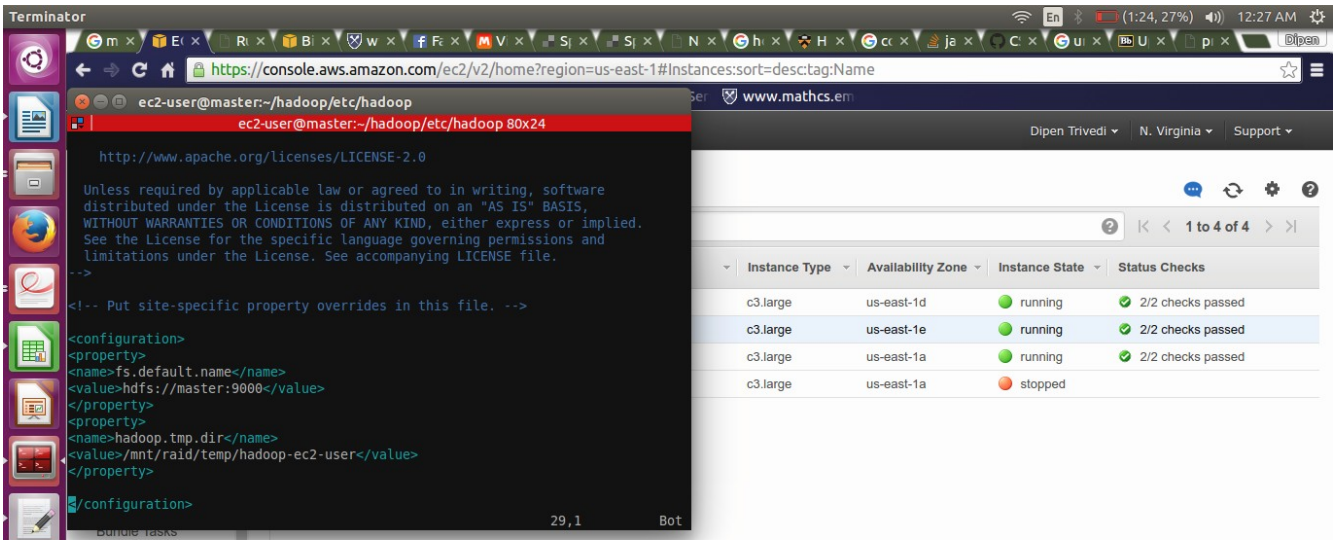
Add java home path in *hadoop-env.sh*

- *sudo vi /hadoop/etc/hadoop/hadoop-env.sh*



Now, edit core-site.xml, which is configuration file for hadoop Core.

- *sudo vi /hadoop/etc/hadoop/core-site.xml*

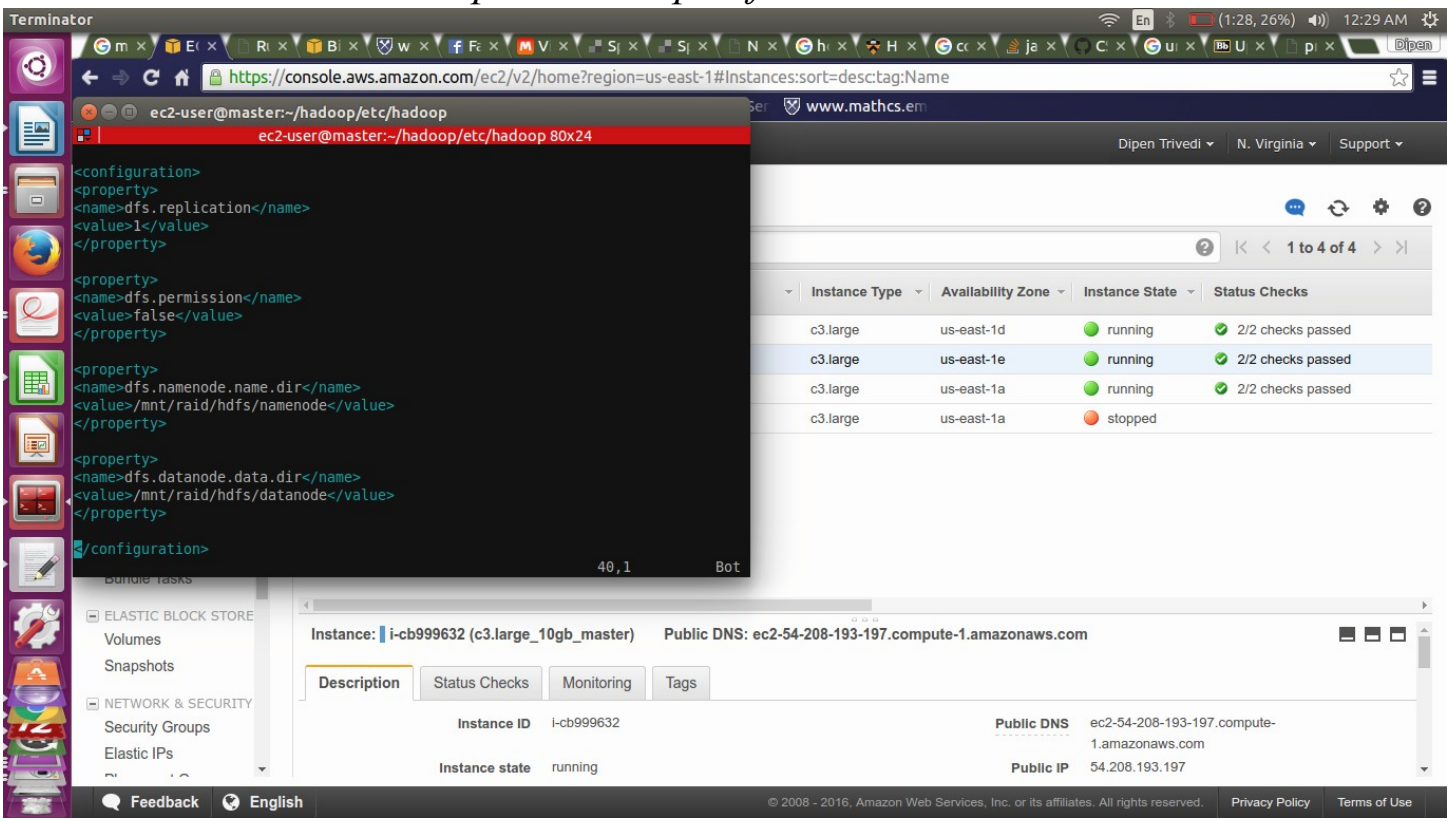


Create namenode and datanode:

- *mkdir -p /hadoop/hdfs/namenode*
- *mkdir -p /hadoop/hdfs/datanode*

Now, edit hdfs-site.xml, which contains the configuration settings for HDFS daemons.

- *Vi /hadoop/etc/hadoop/hdfs-site.xml*

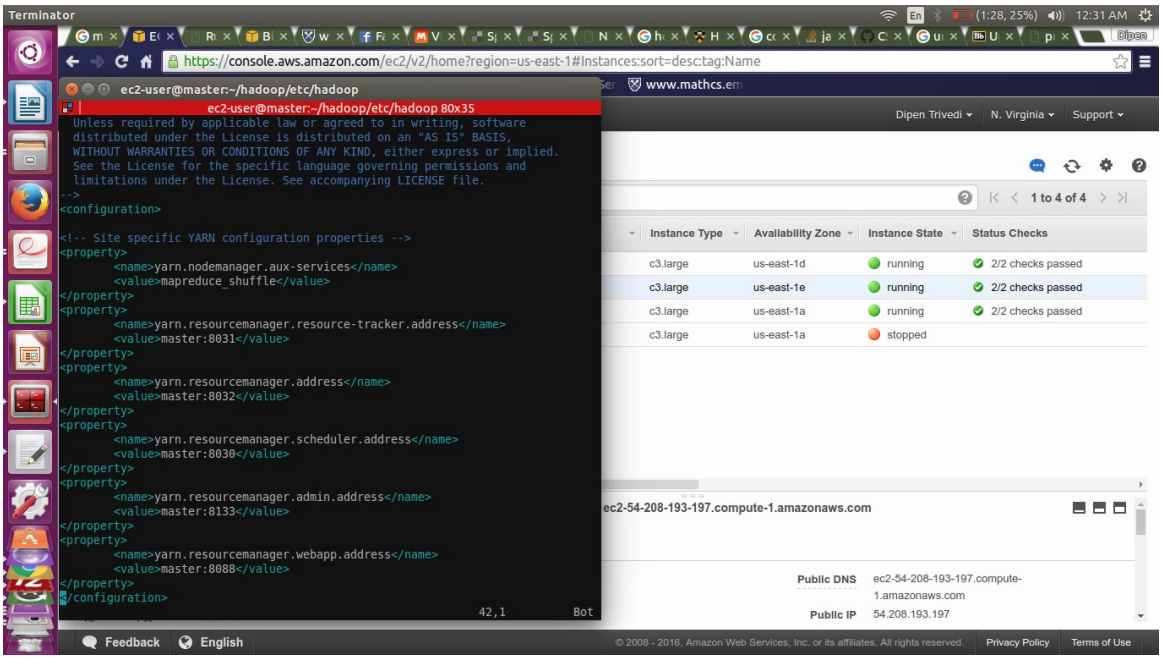


Edit mapred-site.xml, which contains the configuration settings for Map-Reduce daemons.

- *cp /hadoop/etc/hadoop/mapred-site.xml.template /hadoop/etc/hadoop/mapred-site.xml*
- *vi /hadoop/etc/hadoop/mapred-site.xml*

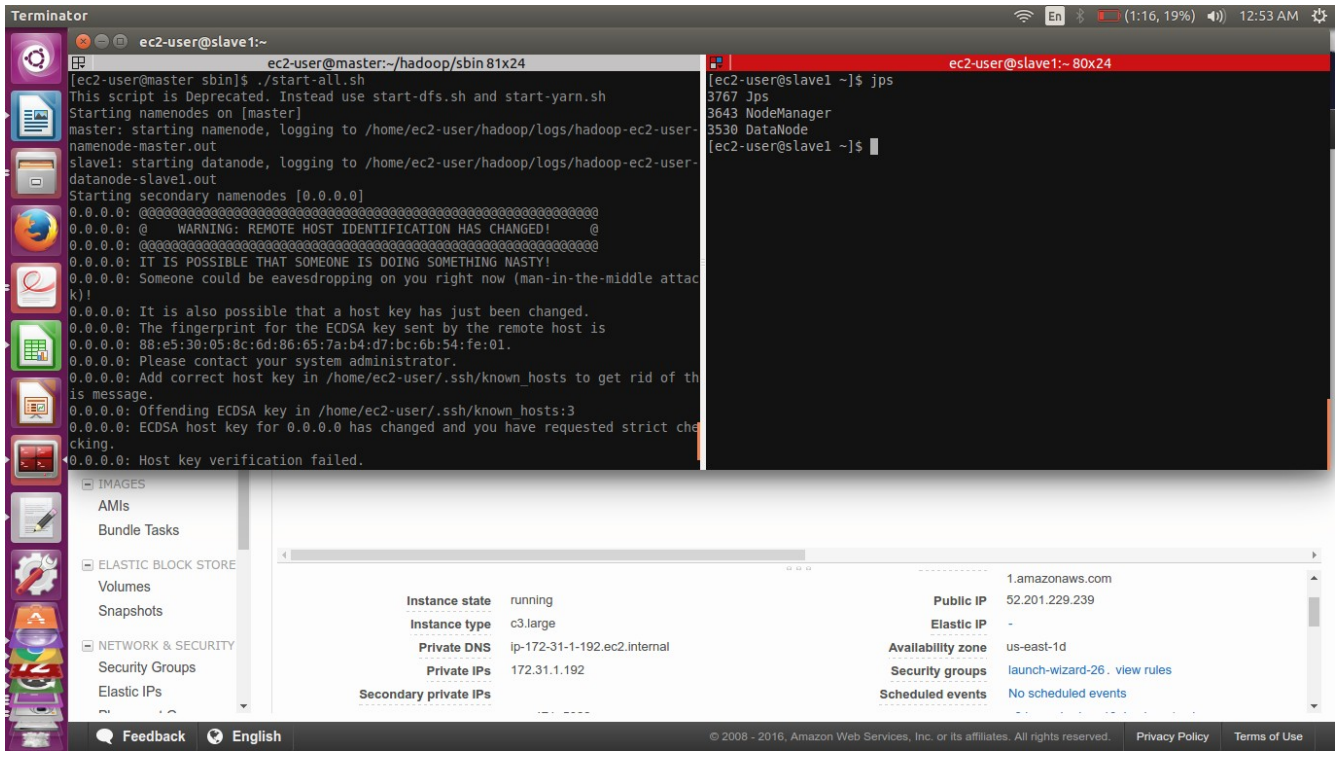
Edit yarn-site.xml, which contains settings for YARN.

- *vi /hadoop/etc/hadoop/yarn-site.xml*



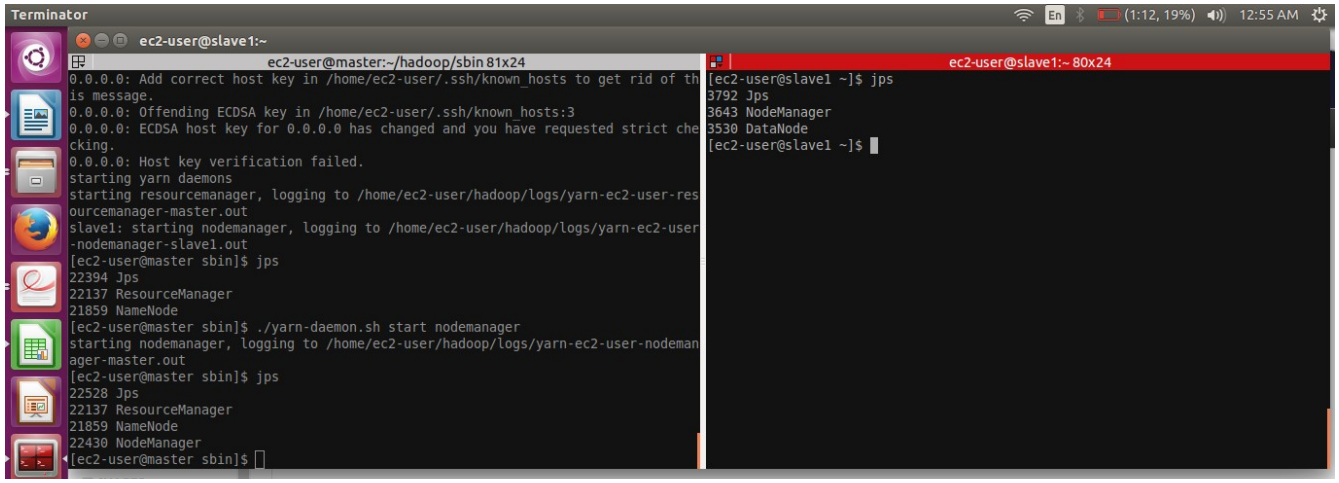
start all the daemons:

- *hadoop namenode -format*
- *./hadoop/sbin/start-all.sh*
- *./hadoop/sbin/yarn-daemon.sh start nodemanager*



check if the all the daemons are running :

- *jps*



Mount the unmounted data drive commands :

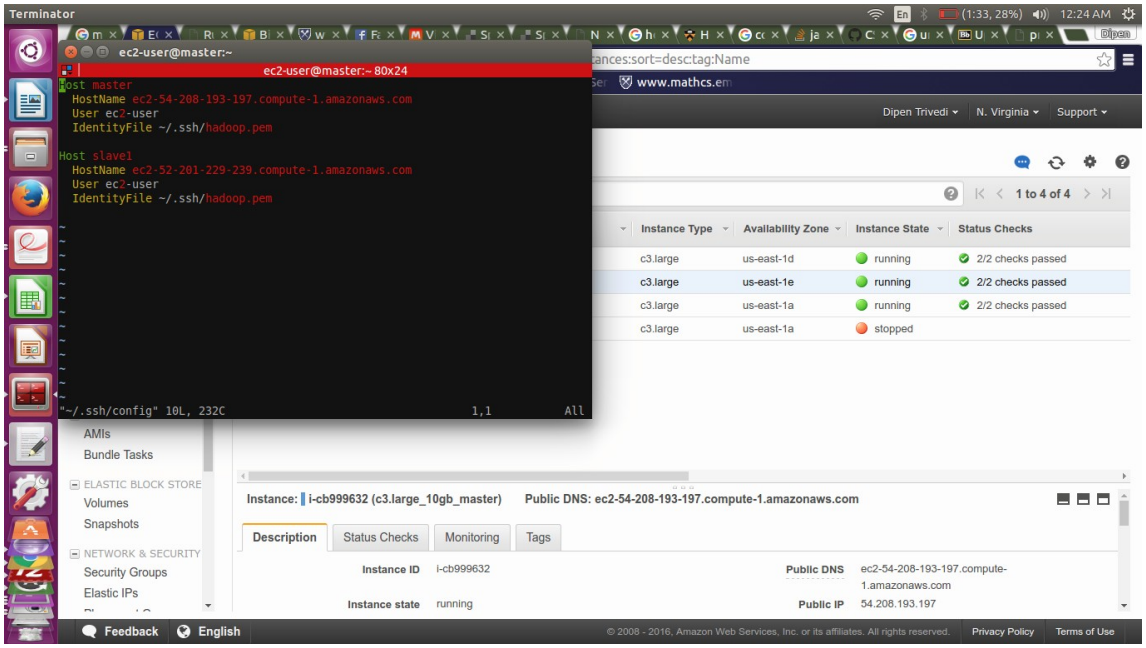
The following is the command to mount the unmounted drives into the hdfs:

- *sudo mdadm --create --verbose /dev/md0 --level=0 --name=hadoop --raid-devices=3 /dev/xvdb /dev/xvdc /dev/xvdd*
- *sudo mkfs.ext4 -L hadoop /dev/md0*
- *sudo mkdir -p /mnt/raid*
- *sudo mount LABEL=hadoop /mnt/raid*
- *sudo chmod 777 /mnt/raid*

SSH Configuration :

To make stop specifying the pem key and host address every time we SSH from local to node, we have to config the ~/.ssh/config file.

- *~/.ssh/config*



- *ssh-keygen -f ~/.ssh/id_rsa -t rsa -P “”*
- *cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys*
- *cat ~/.ssh/id_rsa.pub | ssh slave1 'cat >> ~/.ssh/authorized_keys'*

■ Create AMI of this configuration.

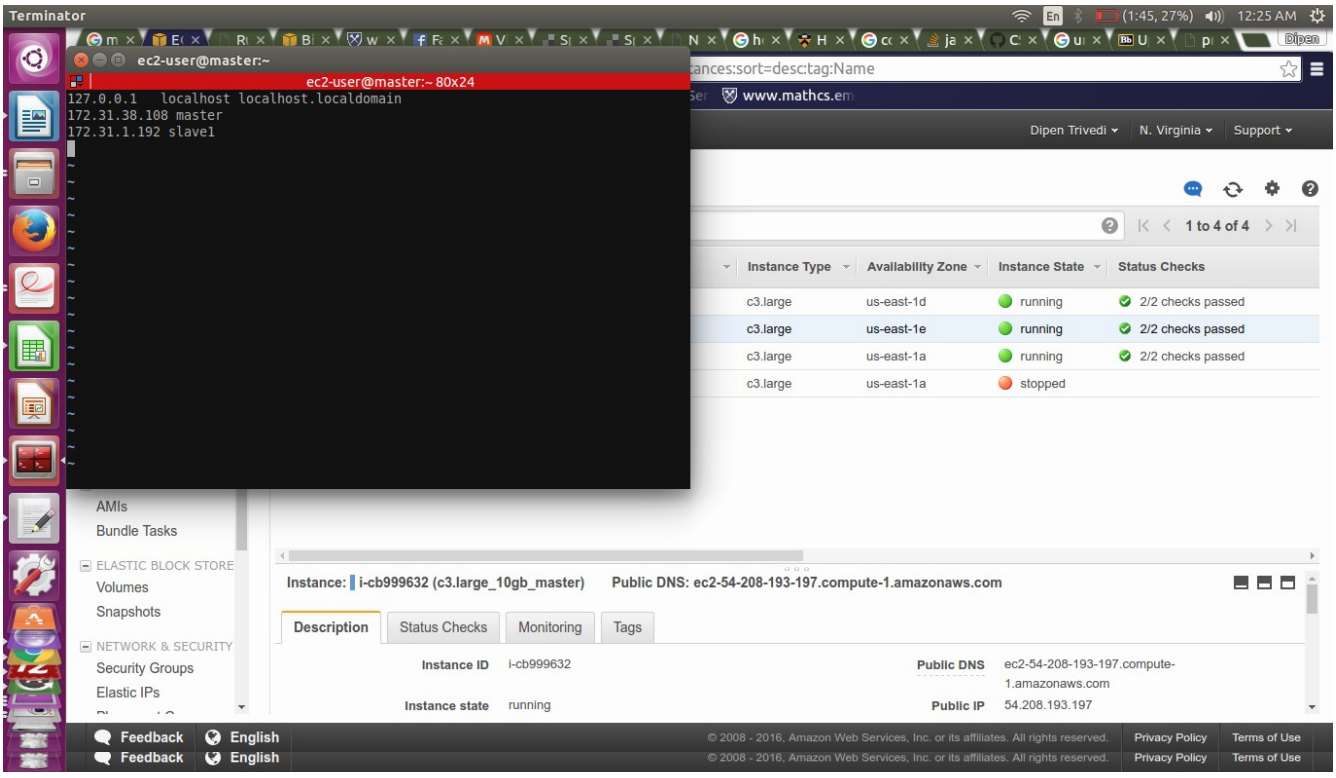
➢ **10GB data sort(1 master, 1 slave) :**

To run 1 master and 1 slave, change following files after launching the above all commands to setup the hadoop:

Run the previously configure AMI:

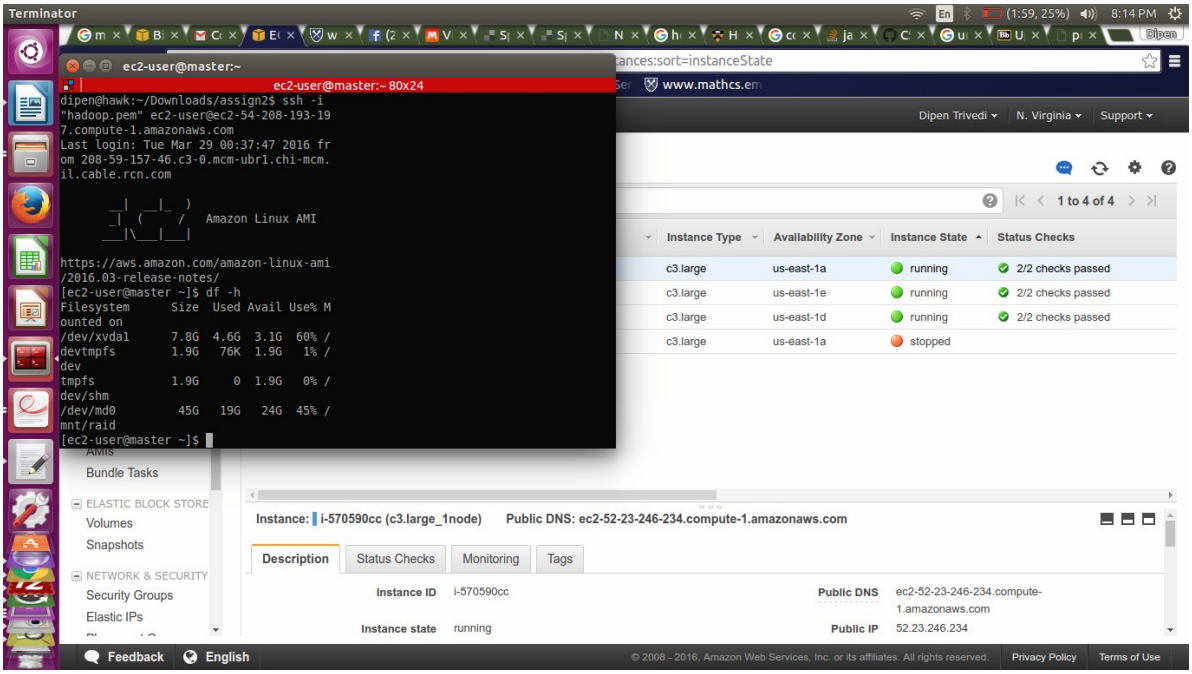
change the host and slave in /etc/hosts:

- sudo vi /etc/hosts

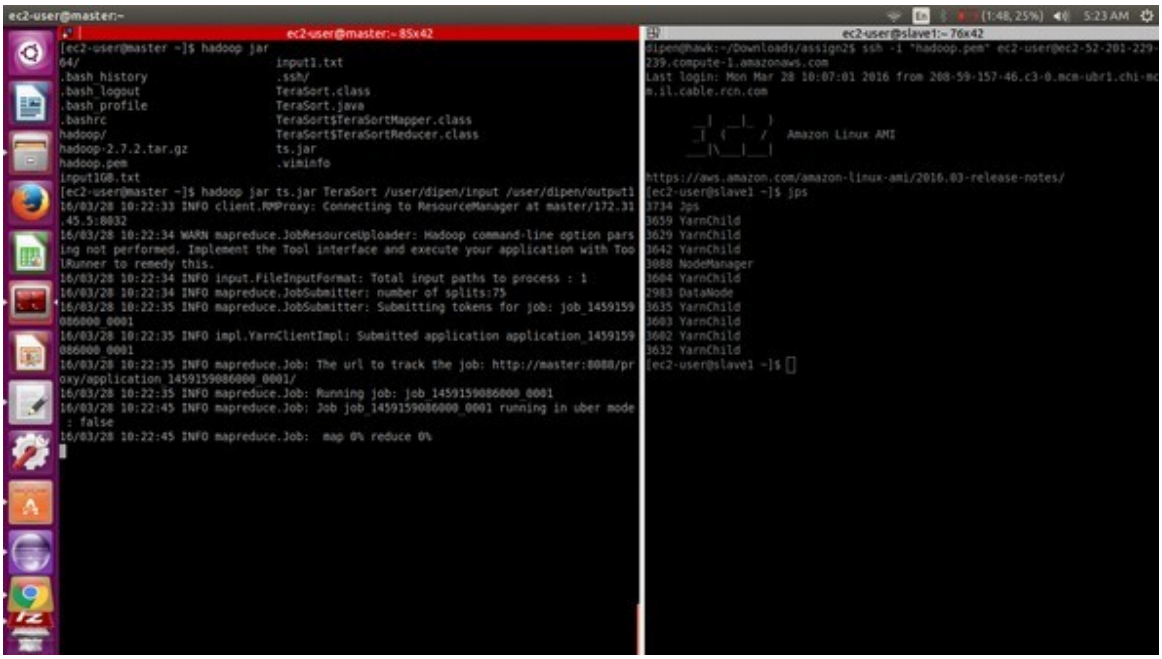


Now, run all daemons as suggested in hadoop configuration.

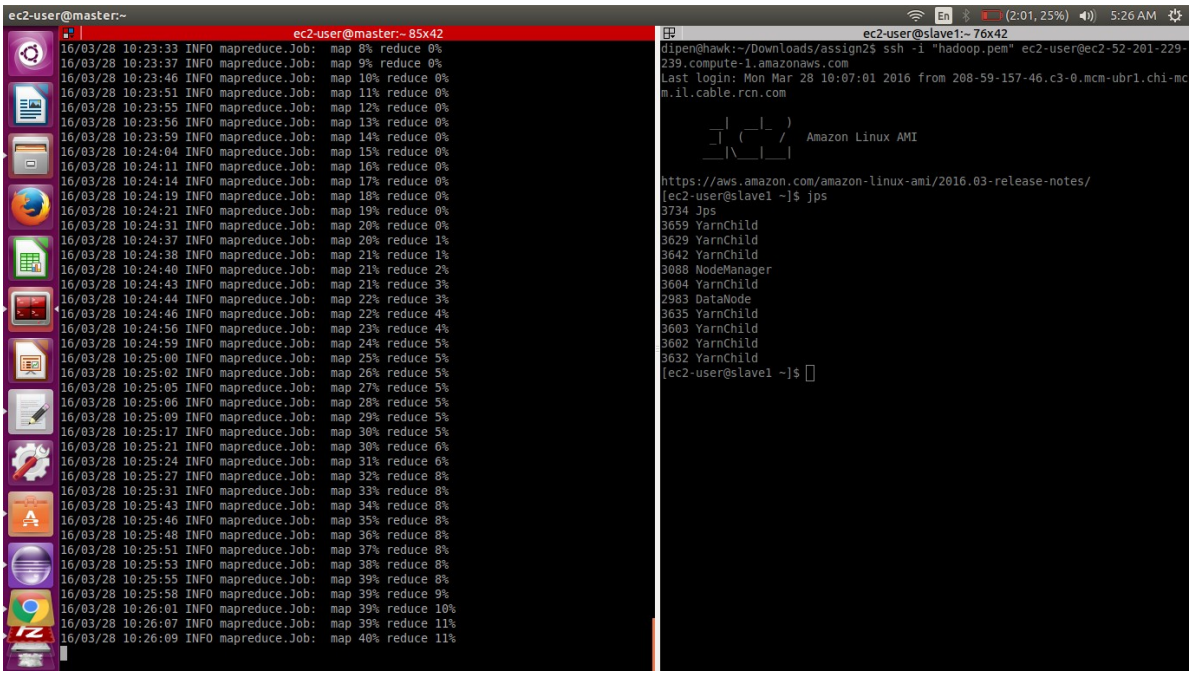
- **Mounted drive's size: after sort :**



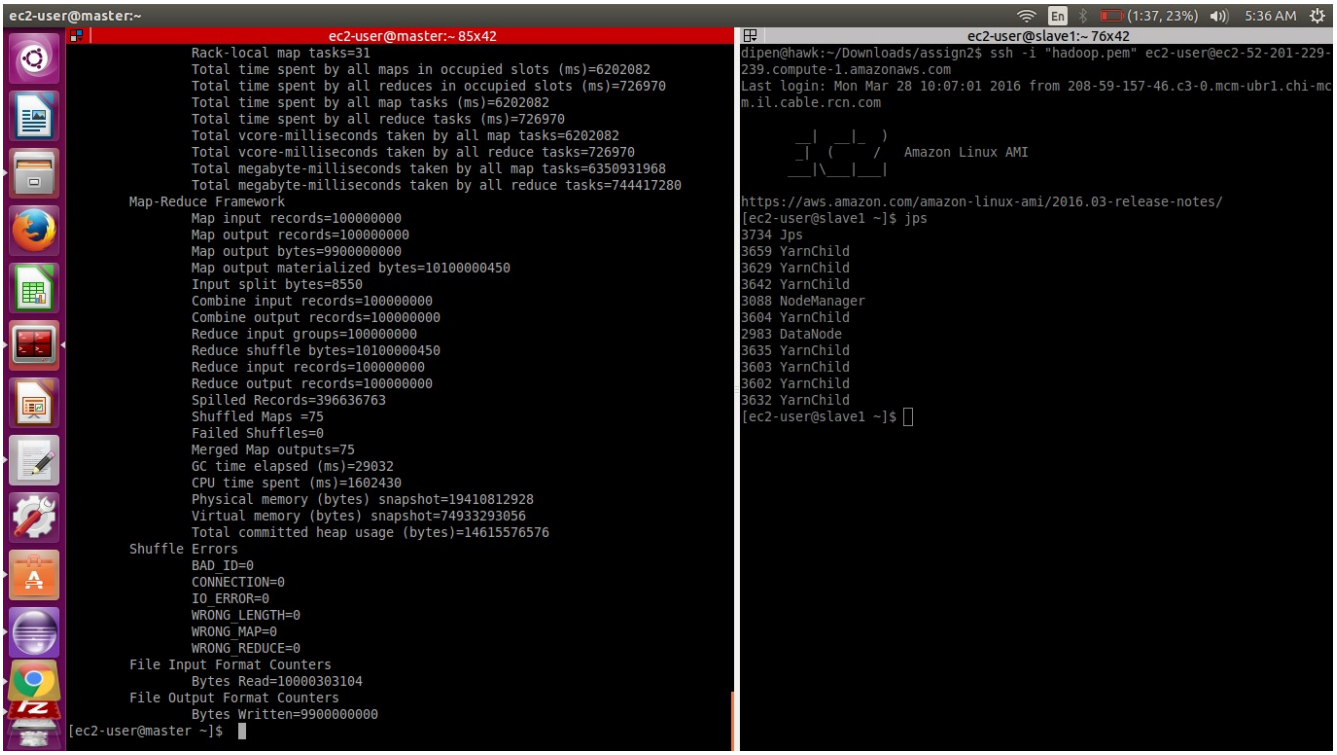
- **Compile Java File TeraSort.java:**
  hadoop com.sun.tools.javac.Main TeraSort.java
- **Create Jar file of TeraSort.java:**
  jar cf ts.jar TeraSort*.class
- **Run Jar File ts.jar :**
  jar ts.jar TeraSort /user/dipen/input /user/dipen/output
- **Sorting : Map 0% & Reduce 0% :**



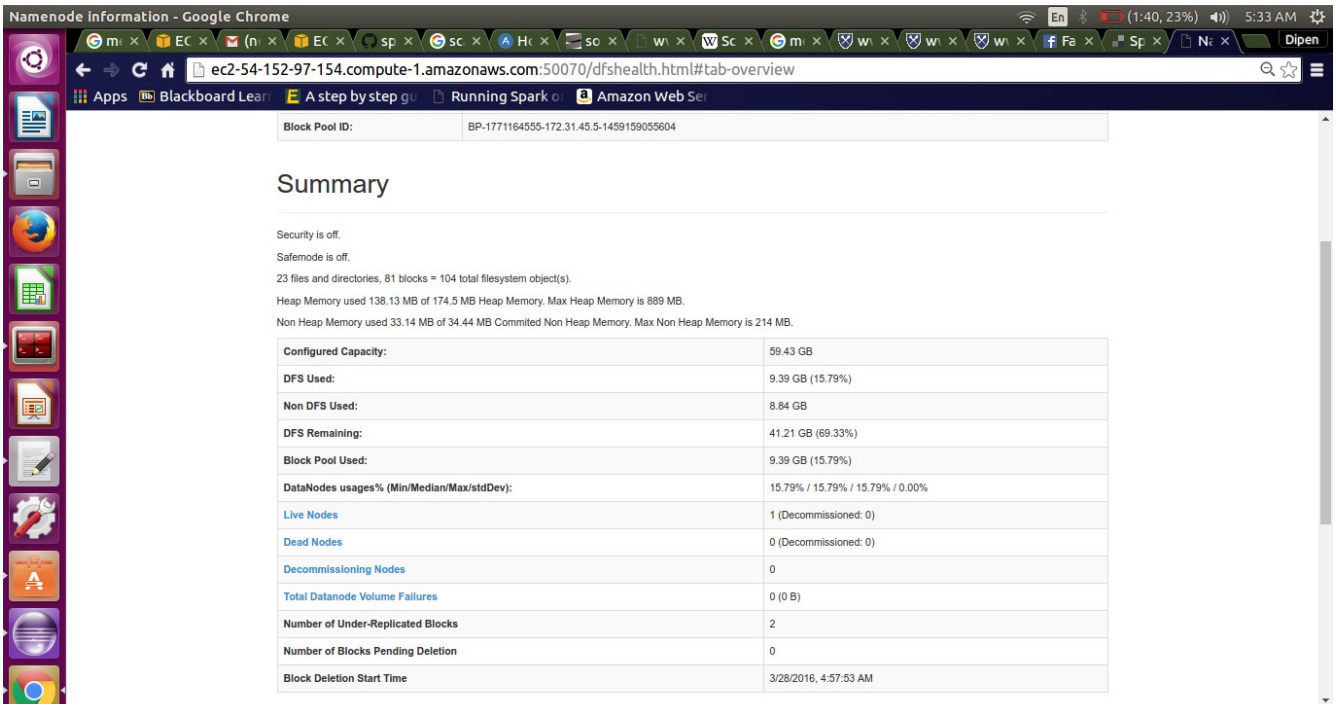
- **sorting : Map 40% & Reduce 11% :**



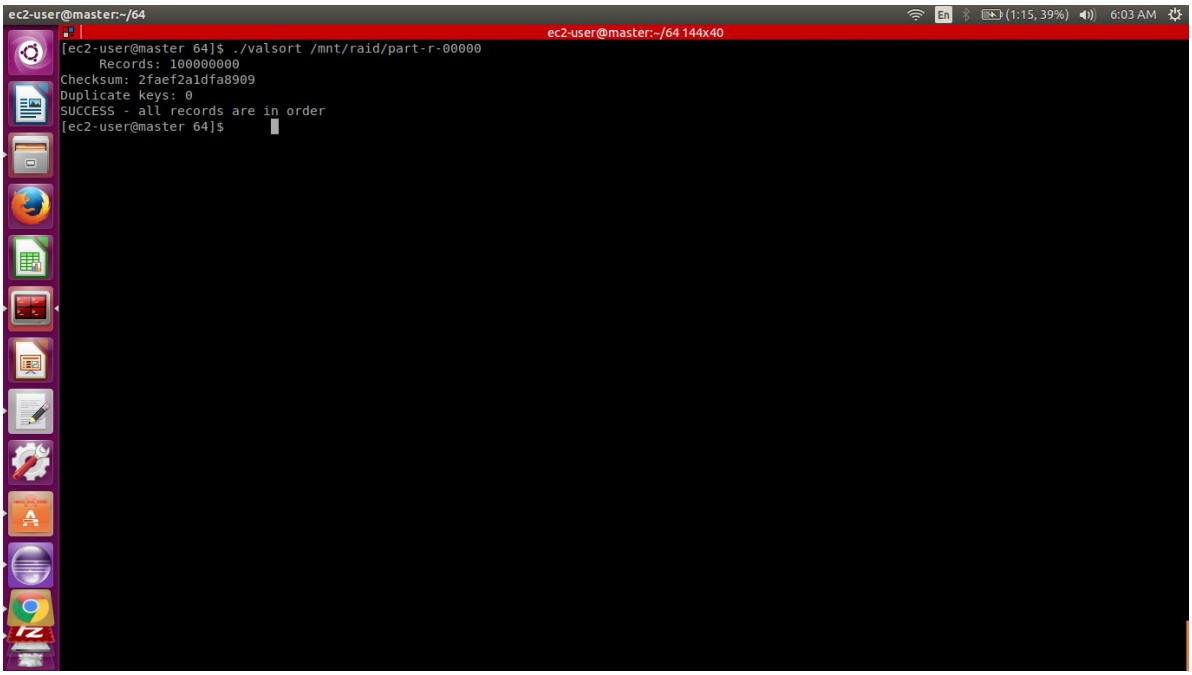
- **sorting : Map 100% & Reduce 100% :**



- **hadoop web interface overview after 100% map-reduce :**



- **validate the data using Valsort :**

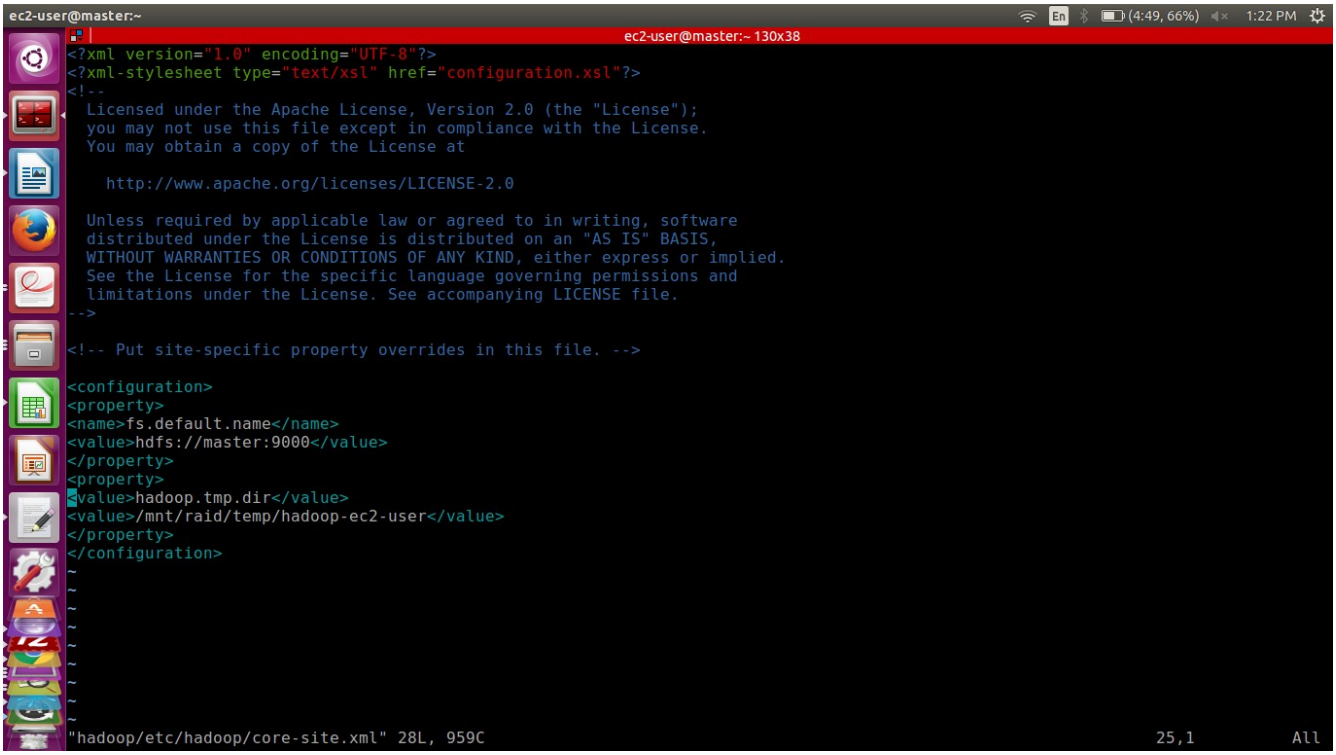


- **100GB data sort(1 master, 16 slave) :**
  To run 1 master and 1 slave, change following files after launching the above all commands to setup the hadoop:

■ Run the previously configure AMI.

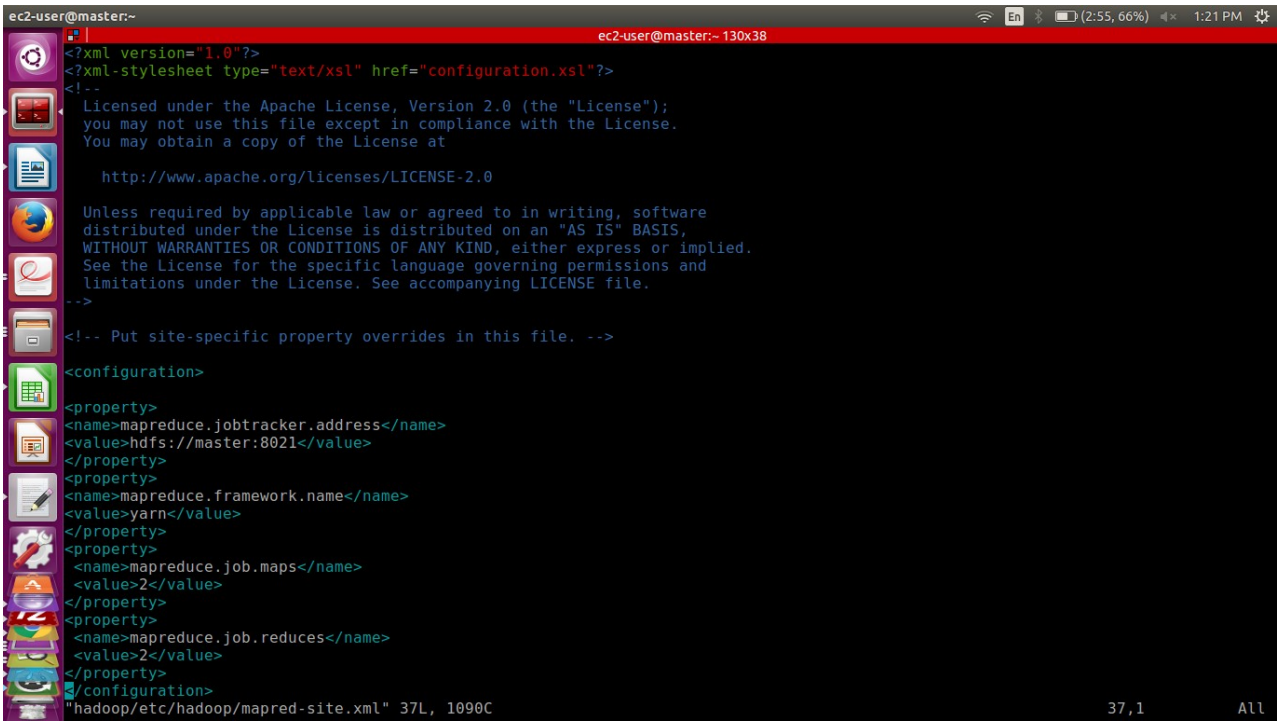
■ Now, following are the changes that we have to made to configure the Multiple slave in the Master Cluster:

1. change in core-site.xml :
   Add the hadoop.tmp.dir property.



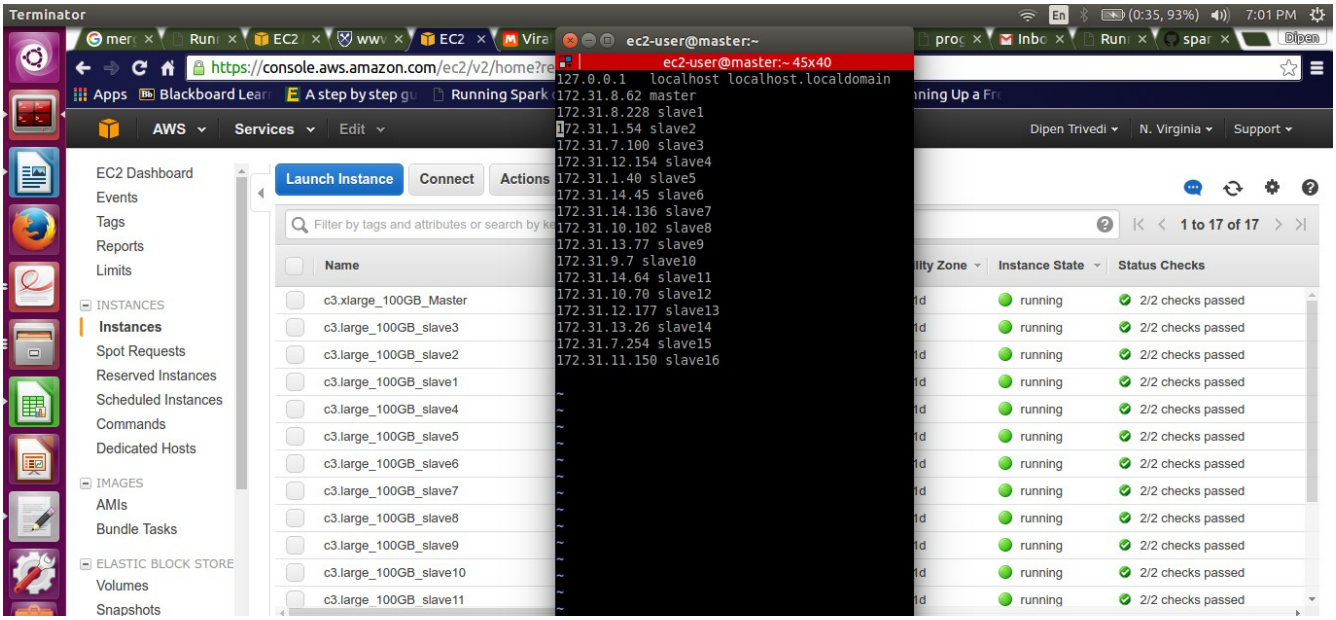
2. change in mapred-site.xml

Add mapreduce.job.maps and mapreduce.job.reduces properties.

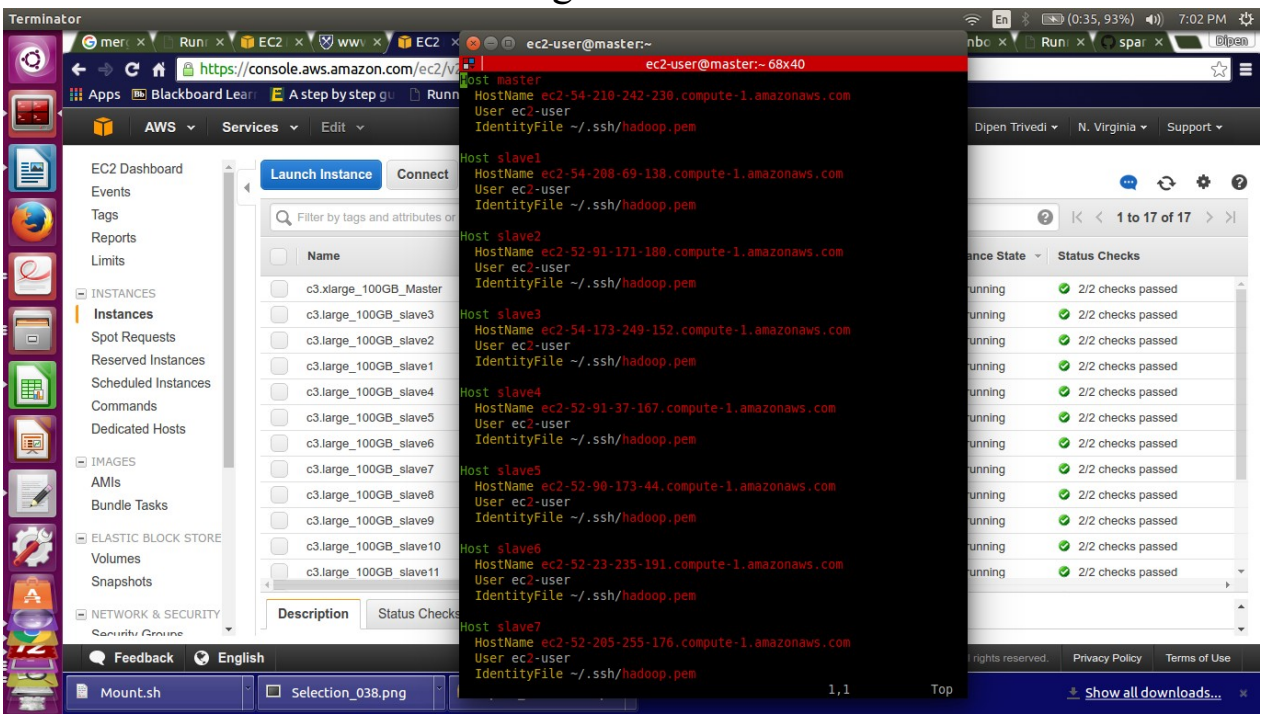


3. change the /etc/hosts in master:

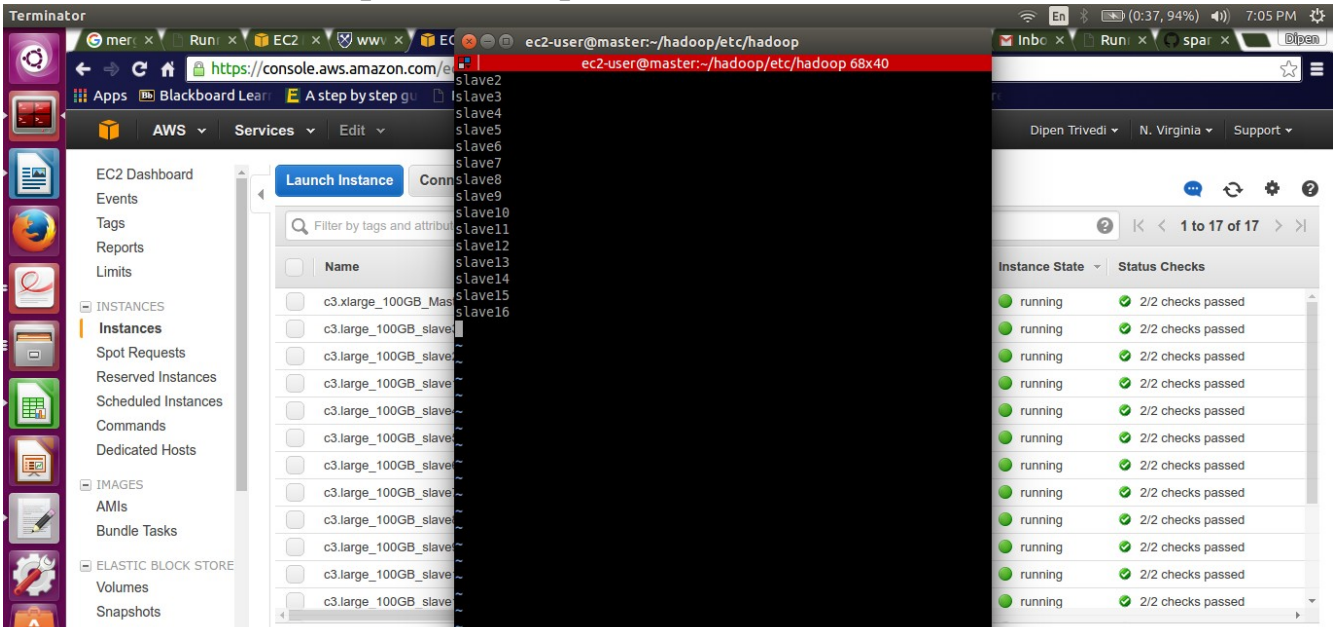
- sudo vi /etc/hosts



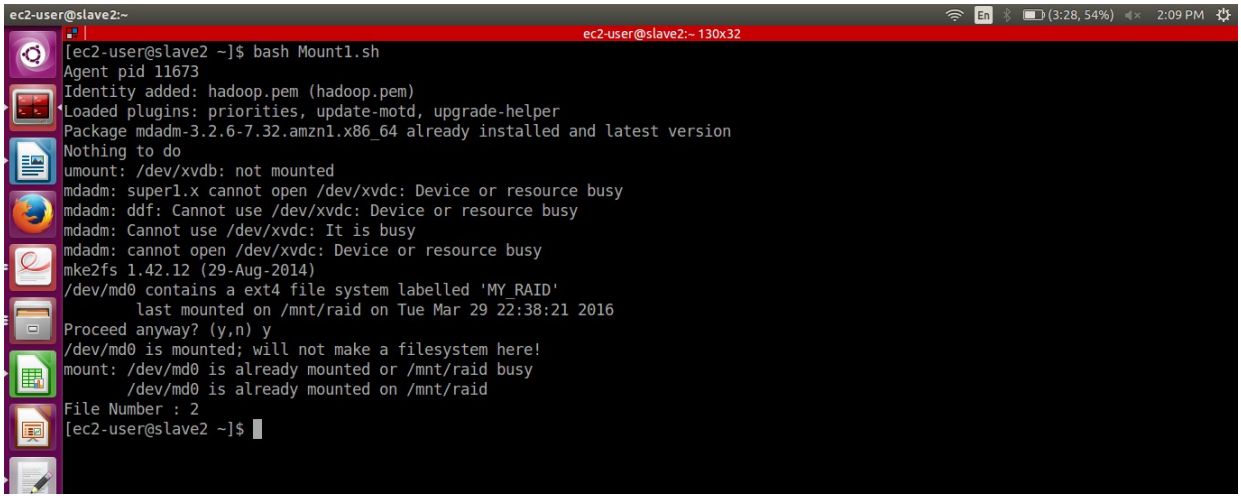
4. change the ~/.ssh/config in master:
   - sudo vi ~/.ssh/config



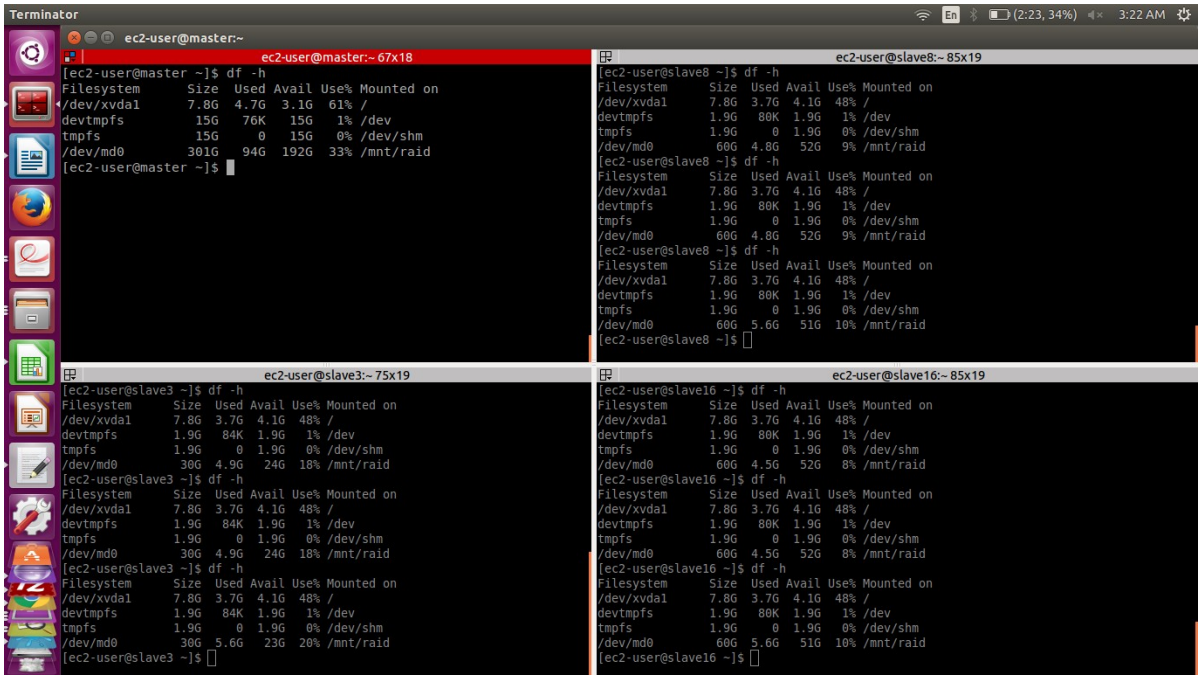
5. change the /hadoop/etc/hadoop/slaves file in master:
   - sudo vi /hadoop/etc/hadoop/slaves



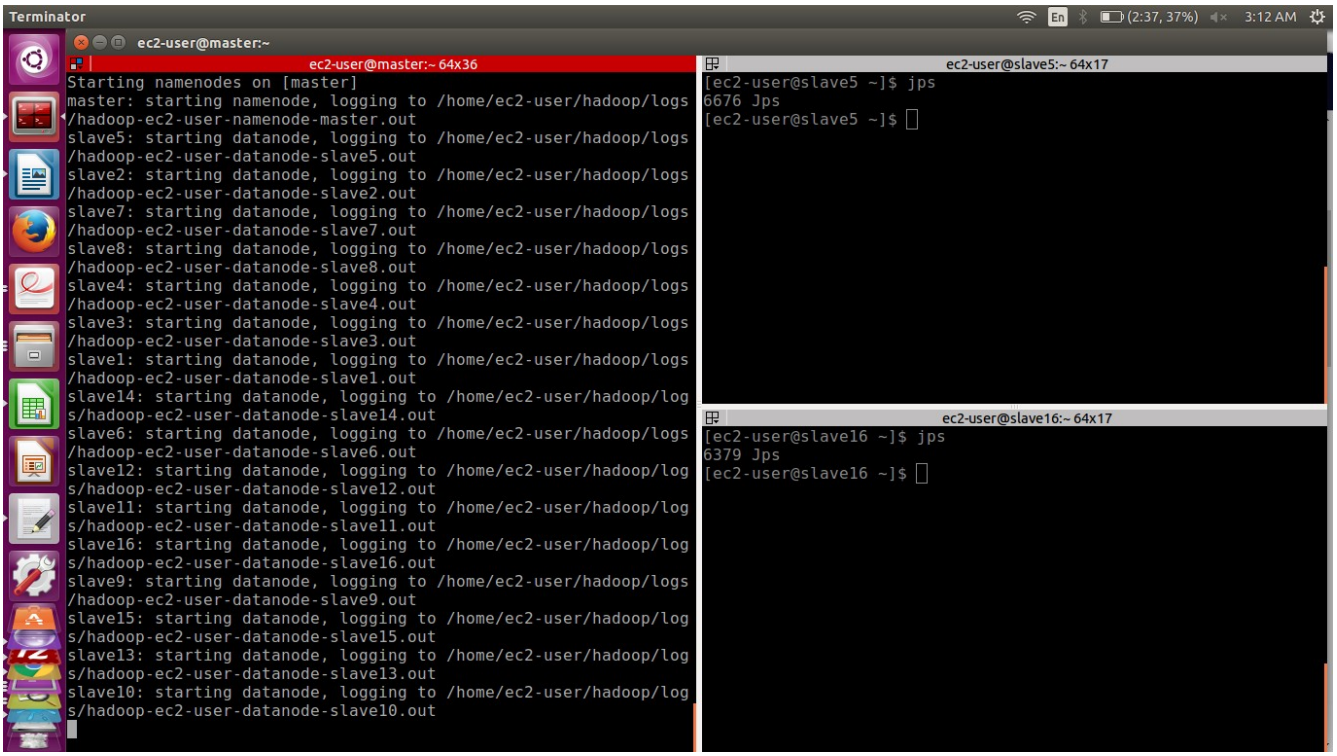
- Mount the unmounted drives in the master and slaves. For that run the bash file created for mounting drives, formating name nodes and changing the /etc/hosts and /hadoop/etc/slaves.
  - *bash Mount1.sh*



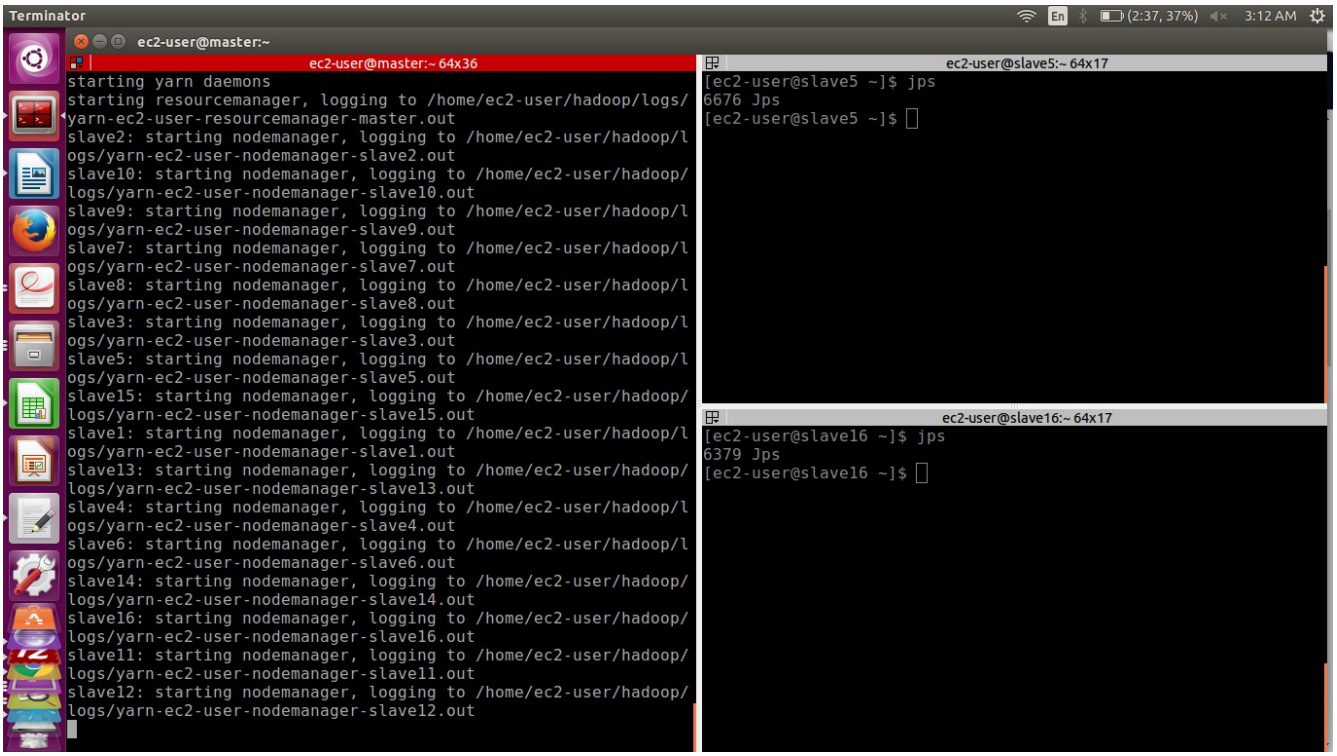
- Now, put the 100GB input file into the hdfs file system.
- Create Output folder in hdfs file system.
- Following screen shot shows the file transfer between master and slaves hdfs file system.



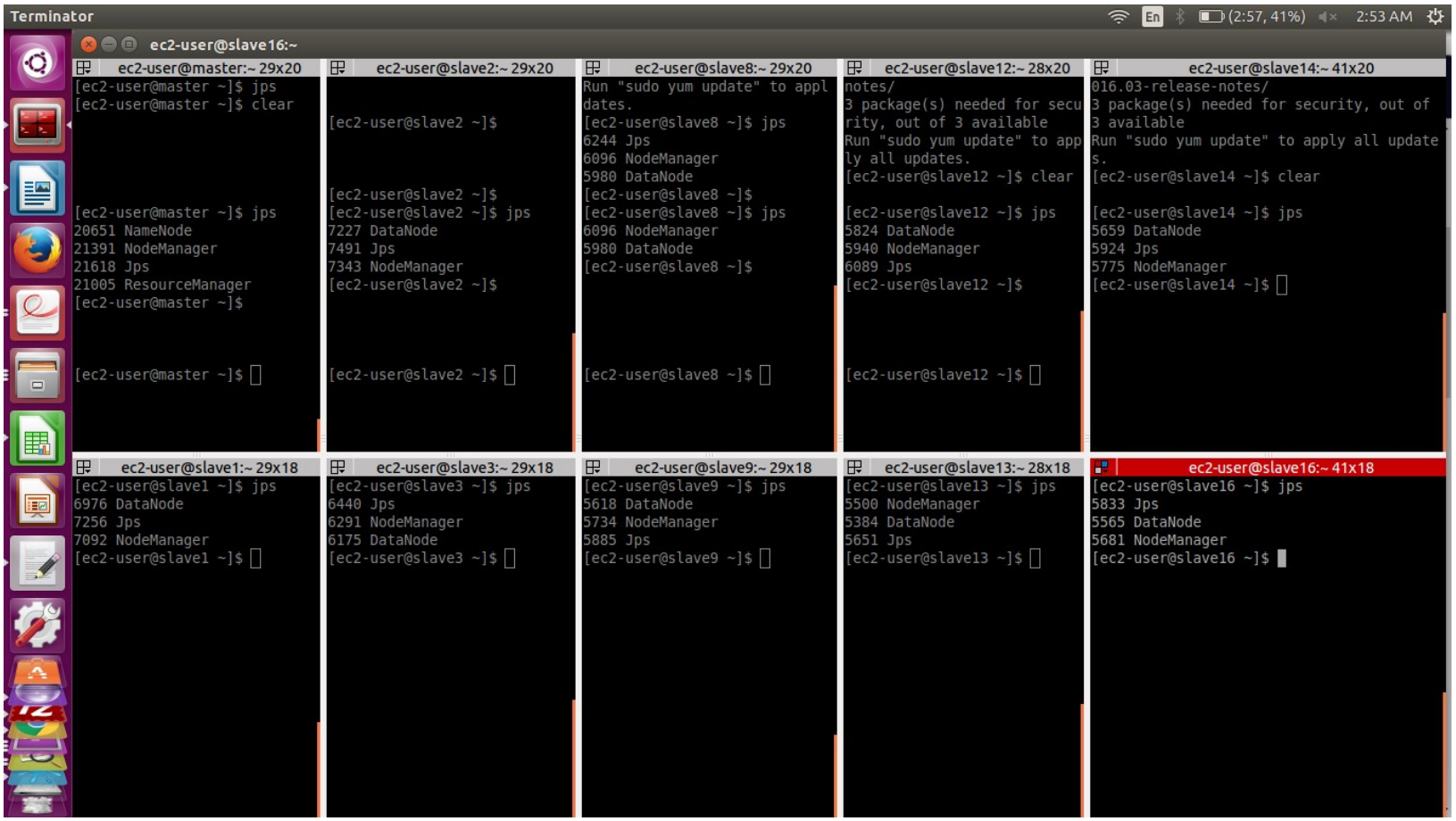
- start all the daemons:
  - *hadoop namenode -format*
  - *./hadoop/sbin/start-all.sh*



  - *./hadoop/sbin/yarn-daemon.sh start nodemanager*

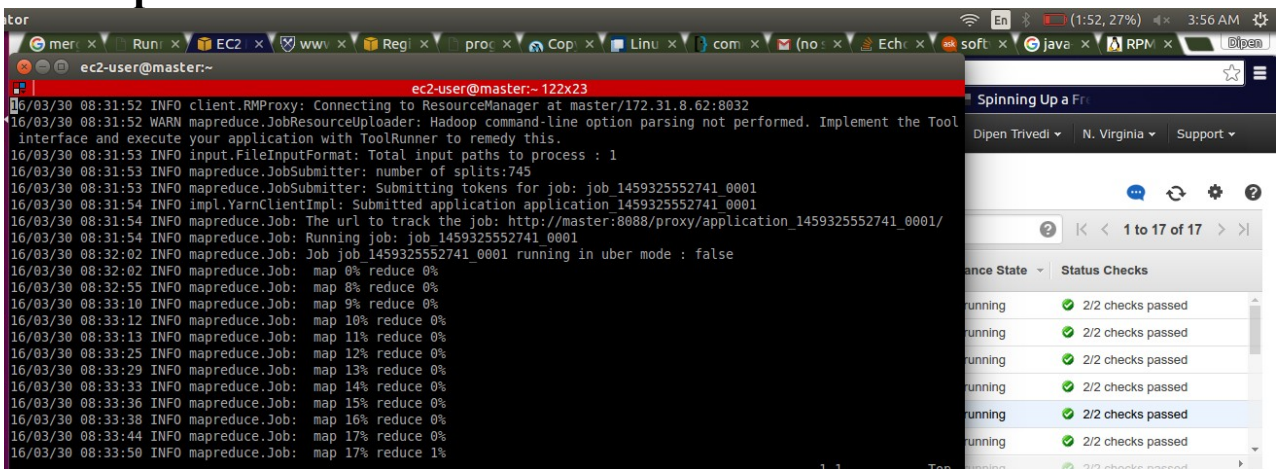


- *jps*

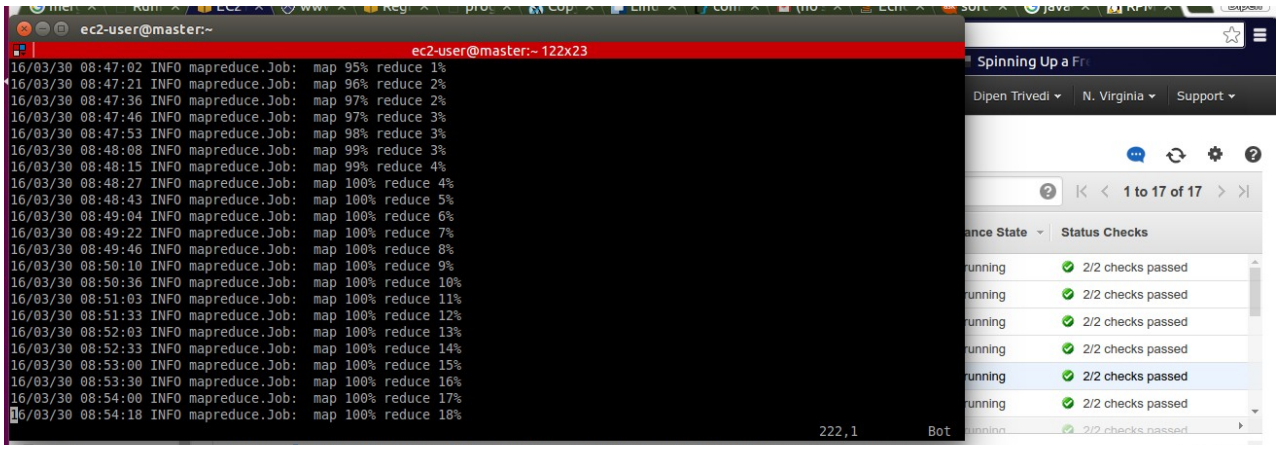

Terminator
(2:57, 41%)
2:53 AM
ec2-user@slave16:~
ec2-user@master:~ 29x20
[ec2-user@master ~]$ jps
[ec2-user@master ~]$ clear
[ec2-user@master ~]$ jps
20651 NameNode
21391 NodeManager
21618 Jps
21005 ResourceManager
[ec2-user@master ~]$
ec2-user@slave2:~ 29x20
[ec2-user@slave2 ~]$
[ec2-user@slave2 ~]$
[ec2-user@slave2 ~]$ jps
7227 DataNode
7491 Jps
7343 NodeManager
[ec2-user@slave2 ~]$
ec2-user@slave8:~ 29x20
Run "sudo yum update" to appl
dates.
[ec2-user@slave8 ~]$ jps
6244 Jps
6096 NodeManager
5980 DataNode
[ec2-user@slave8 ~]$
[ec2-user@slave8 ~]$ jps
6096 NodeManager
5980 DataNode
[ec2-user@slave8 ~]$
ec2-user@slave12:~ 28x20
notes/
3 package(s) needed for secu
rity, out of 3 available
Run "sudo yum update" to app
ly all updates.
[ec2-user@slave12 ~]$ clear
[ec2-user@slave12 ~]$ jps
5824 DataNode
5940 NodeManager
6089 Jps
[ec2-user@slave12 ~]$
ec2-user@slave14:~ 41x20
016.03-release-notes/
3 package(s) needed for security, out of
3 available
Run "sudo yum update" to apply all update
s.
[ec2-user@slave14 ~]$ clear
[ec2-user@slave14 ~]$ jps
5659 DataNode
5924 Jps
5775 NodeManager
[ec2-user@slave14 ~]$
ec2-user@slave1:~ 29x18
[ec2-user@slave1 ~]$ jps
6976 DataNode
7256 Jps
7092 NodeManager
[ec2-user@slave1 ~]$
ec2-user@slave3:~ 29x18
[ec2-user@slave3 ~]$ jps
6440 Jps
6291 NodeManager
6175 DataNode
[ec2-user@slave3 ~]$
ec2-user@slave9:~ 29x18
[ec2-user@slave9 ~]$ jps
5618 DataNode
5734 NodeManager
5885 Jps
[ec2-user@slave9 ~]$
ec2-user@slave13:~ 28x18
[ec2-user@slave13 ~]$ jps
5500 NodeManager
5384 DataNode
5651 Jps
[ec2-user@slave13 ~]$
ec2-user@slave16:~ 41x18
[ec2-user@slave16 ~]$ jps
5833 Jps
5565 DataNode
5681 NodeManager
[ec2-user@slave16 ~]$

- Run the ts.jar file and to take the log details run the jar command in the nohub. Following screen shot shows the starting of Map-Reduce and the command to run jar.
  - *nohup hadoop jar ts.jar TeraSort /user/dipen/input /user/dipen/output &*
  - *nano nohup.out*

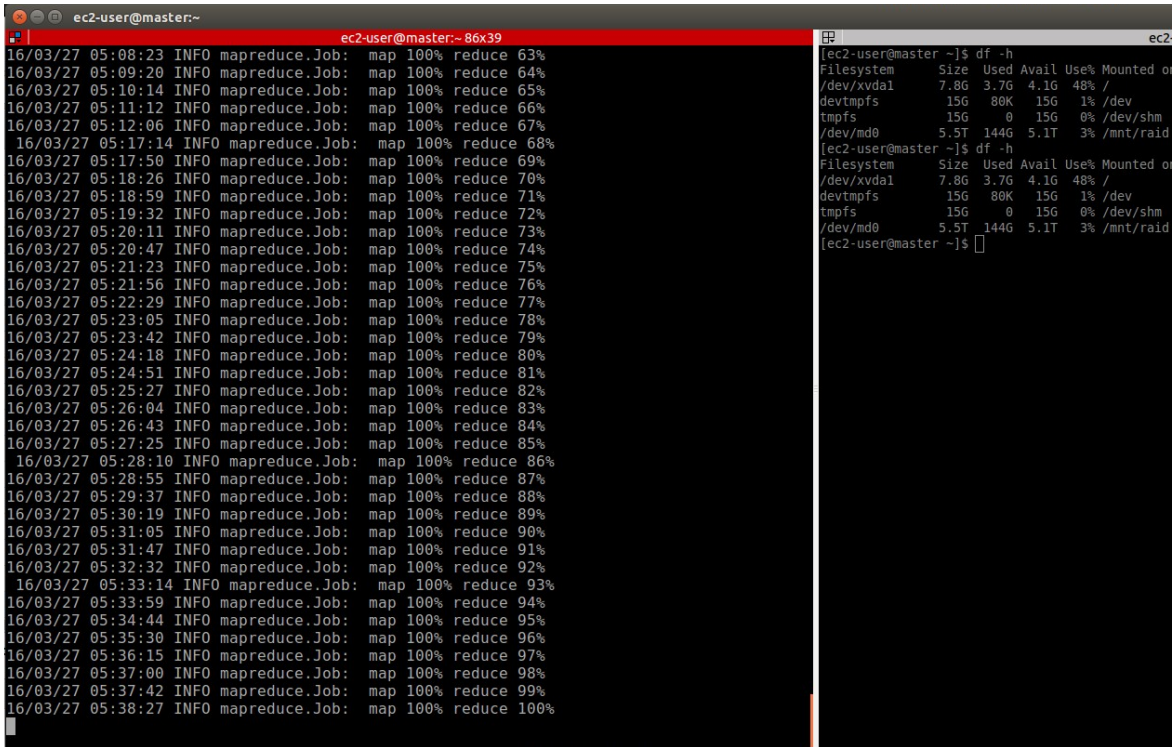
Map 0% - Reduce 0%



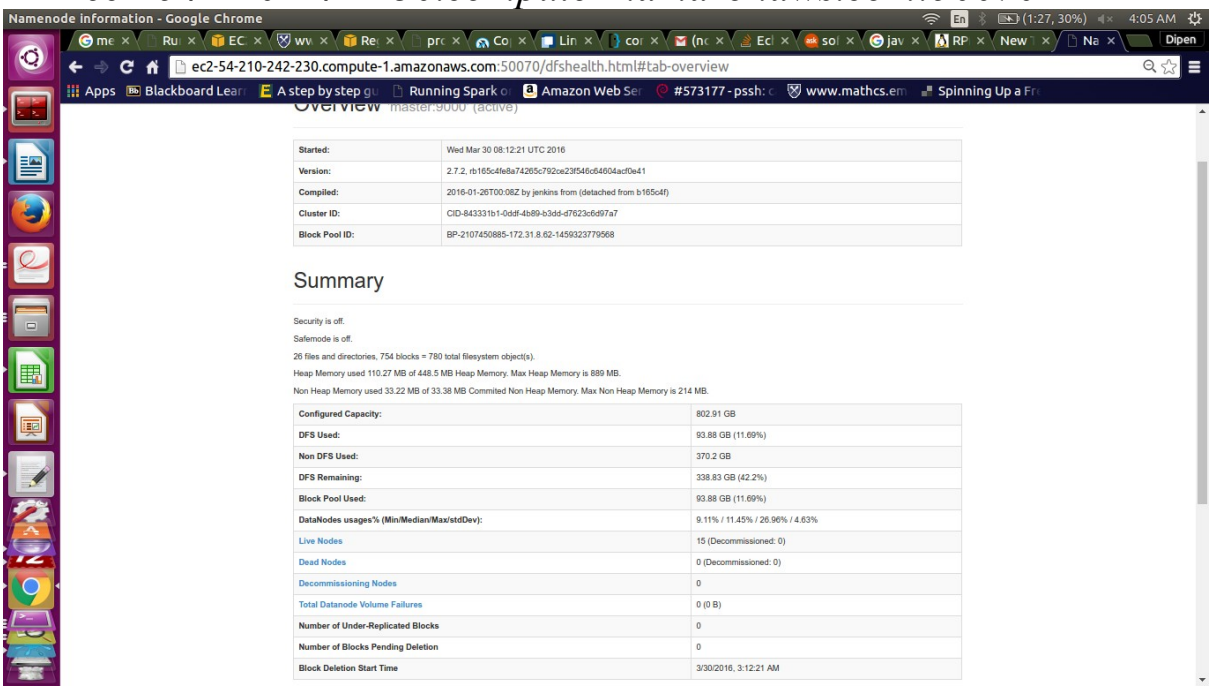
Map 100% - Reduce 18%



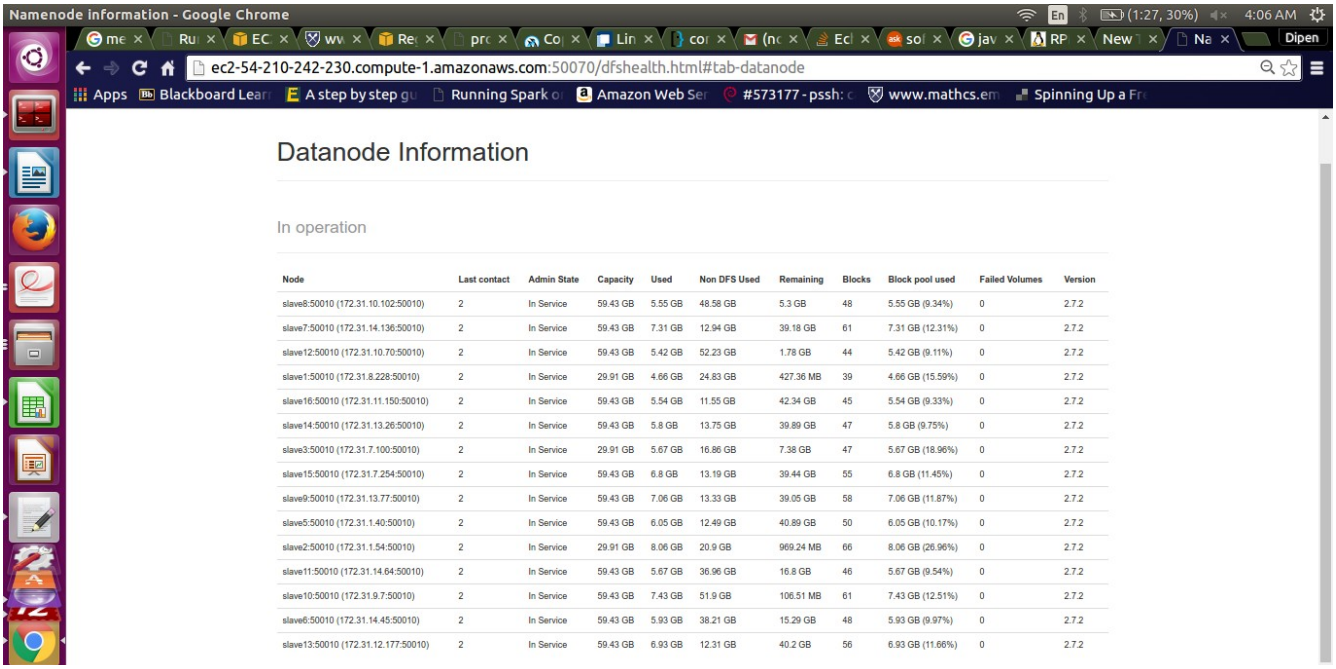
Map 100% - Reduce 100%



- hadoop web-interface overview :
  - open web browser and write : *ec2-54-210-242-230.compute-1.amazonaws.com:50070*



- hadoop web-interface Live Nodes :



Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
slave8:50010 (172.31.10.102:50010)	2	In Service	59.43 GB	5.55 GB	48.58 GB	5.3 GB	48	5.55 GB (9.34%)	0	2.7.2
slave7:50010 (172.31.14.136:50010)	2	In Service	59.43 GB	7.31 GB	12.94 GB	39.18 GB	61	7.31 GB (12.31%)	0	2.7.2
slave12:50010 (172.31.10.70:50010)	2	In Service	59.43 GB	5.42 GB	52.23 GB	1.78 GB	44	5.42 GB (9.11%)	0	2.7.2
slave1:50010 (172.31.8.228:50010)	2	In Service	29.91 GB	4.66 GB	24.83 GB	427.36 MB	39	4.66 GB (15.59%)	0	2.7.2
slave16:50010 (172.31.11.150:50010)	2	In Service	59.43 GB	5.54 GB	11.55 GB	42.34 GB	45	5.54 GB (9.33%)	0	2.7.2
slave14:50010 (172.31.13.26:50010)	2	In Service	59.43 GB	5.8 GB	13.75 GB	39.89 GB	47	5.8 GB (9.75%)	0	2.7.2
slave3:50010 (172.31.7.100:50010)	2	In Service	29.91 GB	5.67 GB	16.86 GB	7.38 GB	47	5.67 GB (18.96%)	0	2.7.2
slave15:50010 (172.31.7.254:50010)	2	In Service	59.43 GB	6.8 GB	13.19 GB	39.44 GB	55	6.8 GB (11.45%)	0	2.7.2
slave9:50010 (172.31.13.77:50010)	2	In Service	59.43 GB	7.06 GB	13.33 GB	39.05 GB	58	7.06 GB (11.87%)	0	2.7.2
slave5:50010 (172.31.1.40:50010)	2	In Service	59.43 GB	6.05 GB	12.49 GB	40.89 GB	50	6.05 GB (10.17%)	0	2.7.2
slave2:50010 (172.31.1.54:50010)	2	In Service	29.91 GB	8.06 GB	20.9 GB	969.24 MB	66	8.06 GB (26.96%)	0	2.7.2
slave11:50010 (172.31.14.64:50010)	2	In Service	59.43 GB	5.67 GB	36.96 GB	16.8 GB	46	5.67 GB (9.54%)	0	2.7.2
slave10:50010 (172.31.9.7:50010)	2	In Service	59.43 GB	7.43 GB	51.9 GB	106.51 MB	61	7.43 GB (12.51%)	0	2.7.2
slave6:50010 (172.31.14.45:50010)	2	In Service	59.43 GB	5.93 GB	38.21 GB	15.29 GB	48	5.93 GB (9.97%)	0	2.7.2
slave13:50010 (172.31.12.177:50010)	2	In Service	59.43 GB	6.93 GB	12.31 GB	40.2 GB	56	6.93 GB (11.66%)	0	2.7.2

- Valsort of the first 10 data of the output:

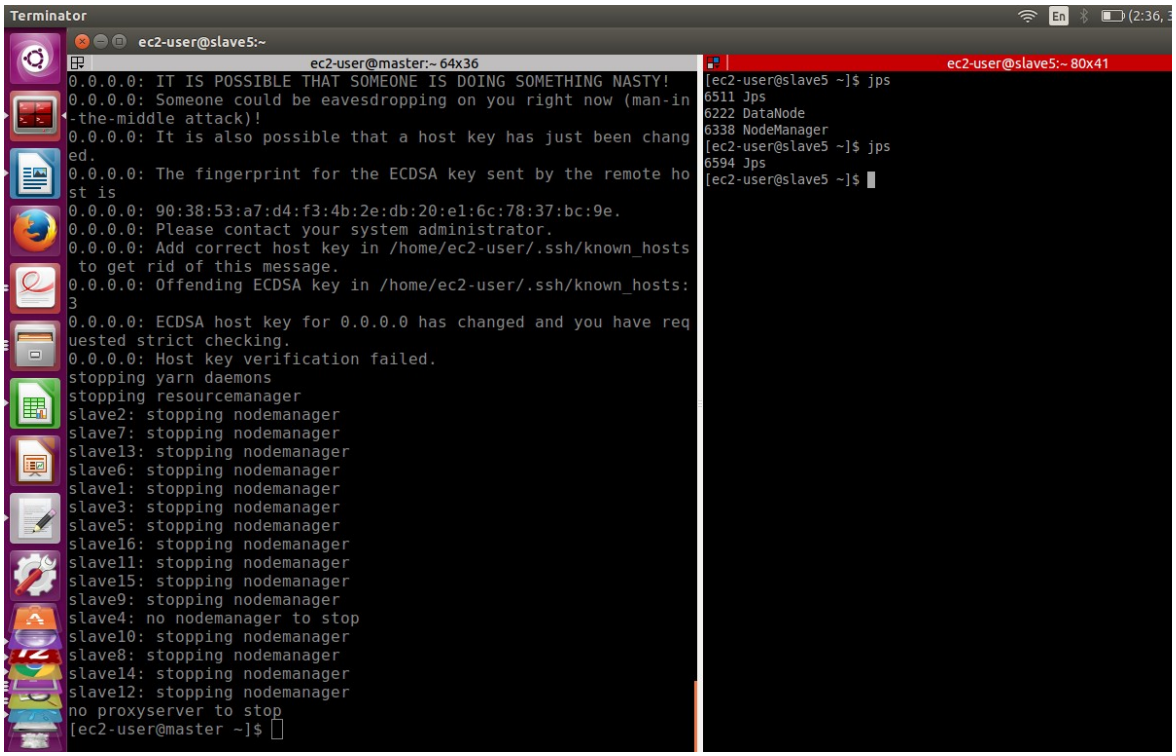
```
[ec2-user@master 64]$ ./valsort /mnt/raid/op1.txt
Records: 500012708
Checksum: ee6c27245251d89
Duplicate keys: 0
SUCCESS - all records are in order
[ec2-user@master 64]$

[ec2-user@master 64]$
```

- Valsort of the last 10 data of the output :

```
[ec2-user@master 64]$ ./valsort /mnt/raid/op2.txt
Records: 499987291
Checksum: ee683173c0538cf
Duplicate keys: 0
SUCCESS - all records are in order
[ec2-user@master 64]$
```

- Stop all daemons :
  - *./hadoop/etc/hadoop/stop-all.sh*



■ Performance Evaluation Table (10 Gb | 100 Gb):

➢ Execution time and Throughput :

Data Size (GB)	Execution Time	Throughput
10 GB	920	10.87
100 GB	4266	23.44

■ Problems Faced During the configuration and Running the program :

➢ Add hadoop.tmp.dir property in the core-site.xml in the master node. If it is not added, then when the program starts mapping and reducing it will stored the output data in the RAM rather than the mounted drive (/mnt/raid). So, after running the few percent of the reducing the program stops running and throws the error no more memory space left.

➢ Add mapreduce.job.maps and mapreduce.job.reduces properties in the mapered-sitx.xml in the master node. If it is not added the data works on the single cpu rather than the two cpu, which will slow down the reducing task and after some percentage of reducing task stop.

■ Configuration files Description :

Below are the description of the files and it explains what changes have to be done in these file to go from 1 node to 16 nodes cluster.

1)conf/master :

This is the master file of the HDFS system, which contains the private ip of the master. It differentiate master of the HDFS system with the slaves of the system. In slave nodes this file is not present.

2)conf/slaves :

This is the slave file of the HDFS system, which contains the private ip of the slave. Master node's conf/slaves file contains all the private ip of the slave nodoes. While slave node's conf/slaves file contains only the private ip of itself.

To go from 1 node to 16 nodes, only change required is to add all the new slave nodes' private ip, which were not there in the 1 node cluster.

3)conf/core-site.xml:

This file contains the configuration settings for the HDFS file sytem such as I/O settings like data storage, memory input data paths which are common to HDFS and Map-Reduce.

To go from 1 node to 16 nodes, only change required is to add the temporary data storage path, because in 16 nodes the data size to process is very large. So, the RAM of the master is not able to process such amount of large data.

4)conf/hdfs-site.xml :

This file contains the configuration settings for HDFS daemons, namenode, secondary name node, the data node and the data replication details.

The only change required to go from 1 node to 16 nodes is to change the path value of the data node and name node.

5)conf/mapred-site.xml :

This file contains the configuration settings for the Map-Reduce daemons and job tracker address.

To go from 1 node to 16 nodes, the changes have to be made is in the mapreduce.job.maps and mapreduce.job.reduces with the value of no of cores in the system(here,value 2).

■ **Questions:**

1) What is a Master node? What is a Slaves node?

Ans:

Master node is the heart of the HDFS system. The main task of the Master node is to schedule the task among the slaves and divide the input data into slaves in equal amount. It also keep track of all the slaves and their work. Master is unique for the entire HDFS system.

Slave node or Data Node is the worker node running under the master node, whose work is to complete the task assigned by the Master node. It also stored the data chunk given by the master. There can be multiple slave nodes in the HDFS system.

2) Why do we need to set unique available ports to those configuration files on a shared environment? What errors or side-effects will show if we use same port number for each user?

Ans:

Each service is joint with the uniquely identified port in the HDFS. So, we can not start 2 different services using the same port. Two services can not work simultaneously on the single port. This kind of conflict cause error by stopping both services in the system. So, it is better to use services with their unique ports.

3) How can we change the number of mappers and reducers from the configuration file?

Ans:

In HDFS, there is a file called mapred-site.xml in the hadoop/etc/hadoop forlder. To change the number of mappers and reducers we have change the value of the property **mapreduce.job.maps** and

mapreduce.job.reduces in the mapred-site.xml.

```
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License. See accompanying LICENSE file.
-->

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

<configuration>

<property>
<name>mapreduce.jobtracker.address</name>
<value>hdfs://master:8021</value>
</property>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
<property>
 <name>mapreduce.job.maps</name>
 <value>2</value>
</property>
<property>
 <name>mapreduce.job.reduces</name>
 <value>2</value>
</property>
</configuration>
```

3) How can we change the number of mappers and reducers from the configuration file?

Ans:

In HDFS, there is a file called mapred-site.xml in the hadoop/etc/hadoop forlder. To change the number of mappers and reducers we have change the value of the property **mapreduce.job.maps** and **mapreduce.job.reduces** in the mapred-site.xml.

7. Spark Sort:

- **Spark Configuration:**

After starting the instance we have to run following commands to install the spark in the system:

- *sudo apt-get update*
- *sudo apt-get install openjdk-7-jdk*

Install Spark :

- *wget http://d3kbcqa49mib13.cloudfront.net/spark-1.0.2.tgz*

Install hadoop :

- *wget* http://mirrors.ocf.berkeley.edu/apache/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz

install scala :

- *wget* http://www.scala-lang.org/files/archive/scala-2.10.4.tgz
- *sudo mkdir /usr/local/src/scala*
- *sudo tar xvf scala-2.10.4.tgz -C /usr/local/src/scala/*

Open bashrc file:

- vi .bashrc

  add following lines :
- *export SCALA_HOME=/usr/local/src/scala/scala-2.10.4*
  export PATH=$SCALA_HOME/bin:$PATH
- *. .bashrc*

Launching the Cluster :

Set environment variables for the AWS access key and secret access key:

- *export AWS_ACCESS_KEY_ID=AKPAICCB6WRTXULWLCL*
- *export AWS_SECRET_ACCESS_KEY_ID=KYDz2c0xfd6DvKN/yOyN53*

Give permission to the key:

- *chmod 600 spark.pem*

Now, to run the 1 master, 1slave and 1 master 16 slaves following steps should be followed:

➢ 10GB data sort(1 master, 1 slave) :

Run the "spark-ec2" file to create 1 master and 1 slave with c3.large instance type, extra data volume of 30GB and spot instance price=0.04:

- ./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_test

In case of connection refuse 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_test --resume

After Spark finish launching cluster, the following screen will be shown:

```
ec2-52-87-185-138.compute-1.amazonaws.com
Shutting down GANGLIA gmond:                               [FAILED]
Starting GANGLIA gmond:                                    [  OK  ]
Shutting down GANGLIA gmond:                               [FAILED]
Starting GANGLIA gmond:                                    [  OK  ]
Connection to ec2-52-87-185-138.compute-1.amazonaws.com closed.
Shutting down GANGLIA gmetad:                              [FAILED]
Starting GANGLIA gmetad:                                   [  OK  ]
Stopping httpd:                                            [FAILED]
Starting httpd:                                            [  OK  ]
[timing] ganglia setup:  00h 00m 02s
Connection to ec2-52-23-161-128.compute-1.amazonaws.com closed.
Spark standalone cluster started at http://ec2-52-23-161-128.compute-1.amazonaws
.com:8080
Ganglia started at http://ec2-52-23-161-128.compute-1.amazonaws.com:5080/ganglia
Done!
ubuntu@ip-172-31-45-14:~/spark-1.6.1-bin-hadoop2.6/ec2$ ls
deploy.generic  lib  README  spark-ec2  spark_ec2.py  spark.pem
ubuntu@ip-172-31-45-14:~/spark-1.6.1-bin-hadoop2.6/ec2$ cd
ubuntu@ip-172-31-45-14:~$ ls
sbt-0.13.5.deb    spark-1.6.1-bin-hadoop2.6      spark.pem
scala-2.10.4.tgz  spark-1.6.1-bin-hadoop2.6.tgz
ubuntu@ip-172-31-45-14:~$
```

Now, the spark setup is done. We have to do following change to hadoop's hdfs-site.xml. Following are the changes should be done in these files:

- *ephermal/conf/hdfs-site.xml*

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

Now, copy all these hdfs chages to slave with the following command:

- *./copy-dir ~/ephemeral-hdfs/conf/*

Now, go to the hdfs file system:

- *./ephemeral-hdfs/bin/hadoop/stop_dfs.sh*

Now, go to the spark file system:

- *./spark/sbin/stop_all.sh*

Now, go to the spark file system:

- *./spark/sbin/start_all.sh*

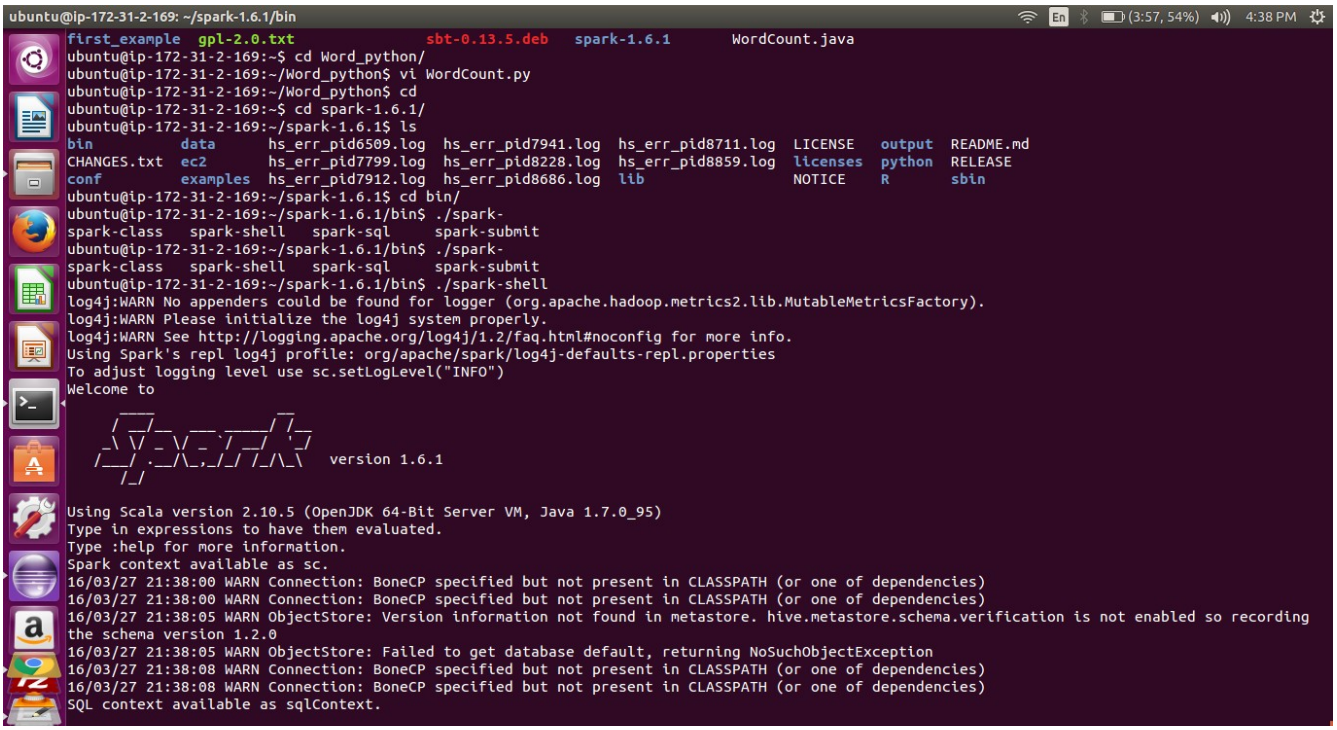
Now, go to the hdfs file system:

- *./ephemeral-hdfs/bin/hadoop/.start_dfs.sh*

Now, check of running daemons with following command:

- *jps*

Now, open spark shell to run the spark program :

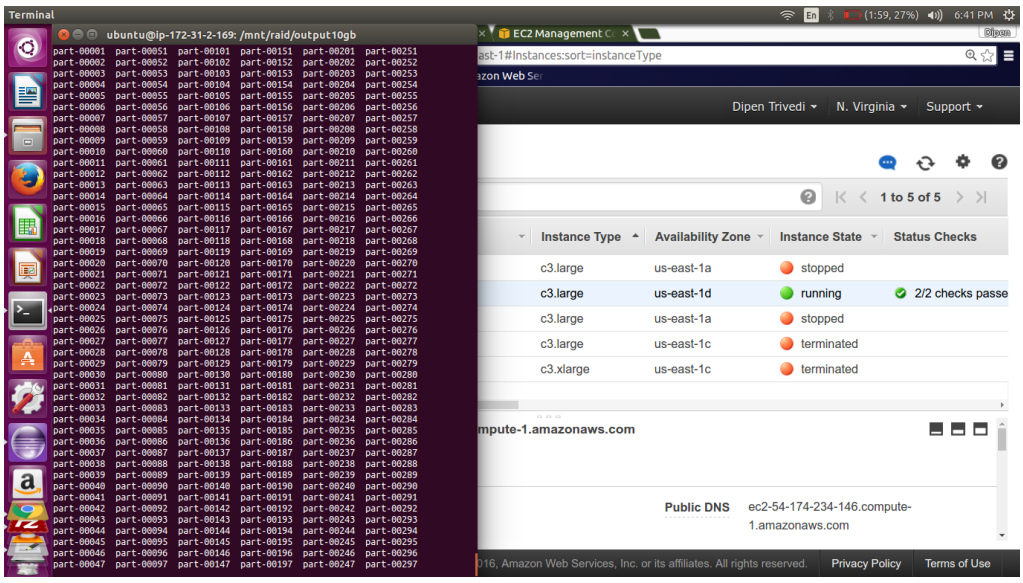


Now, run following spark program in the shell :

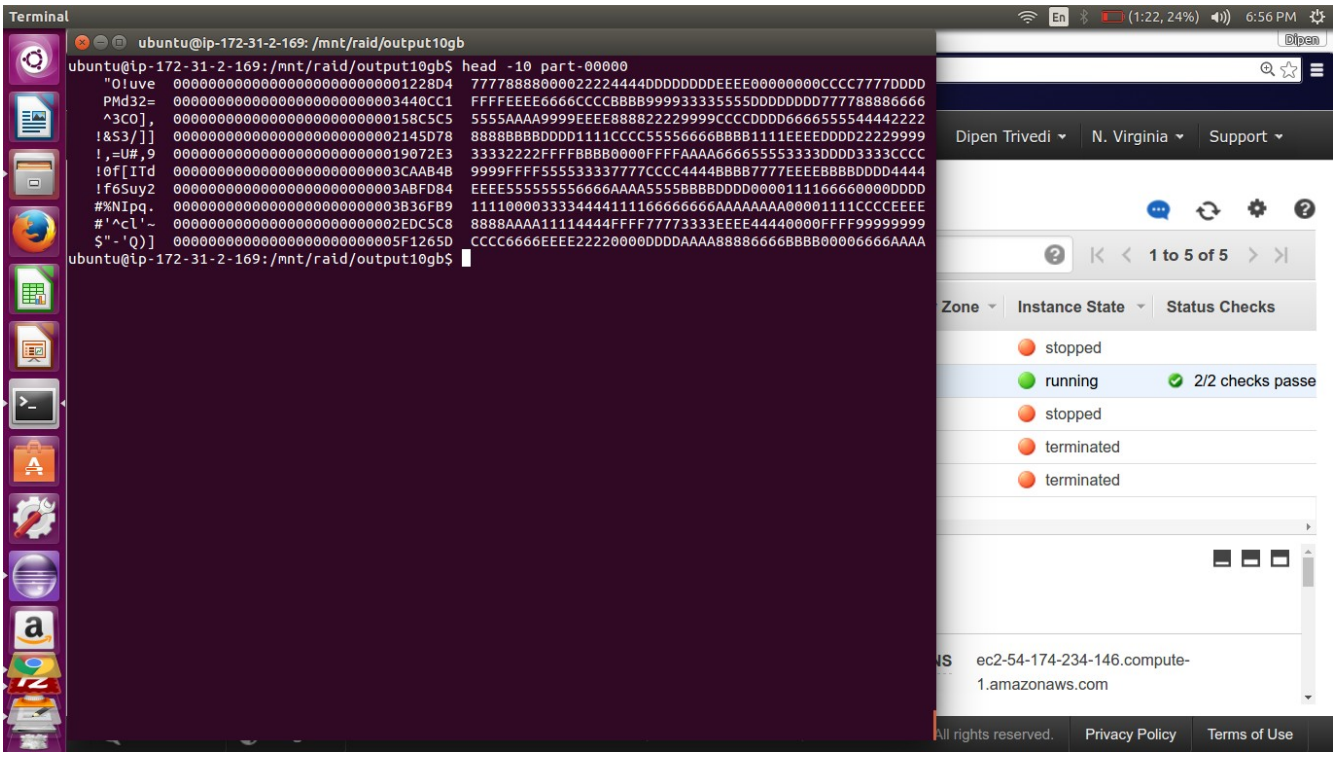
- *TerasortFile = sc.textFile("hdfs://PUBLIC DNS : 9000/user/hadooo/input/input_100GB.txt")*
- *TeraSortObj = TerasortFile.flatMap(lambda line:ine.split("\n")) .map(lambda dicto: (str(dicto[:10]),str(dicto[10:]))).sortByKey().map(lambda (a,b) : a+b)*

- *TeraSortObj.saveAsTextFile("hdfs://Public DNS :9000/user/hadoop/output_100GB")*

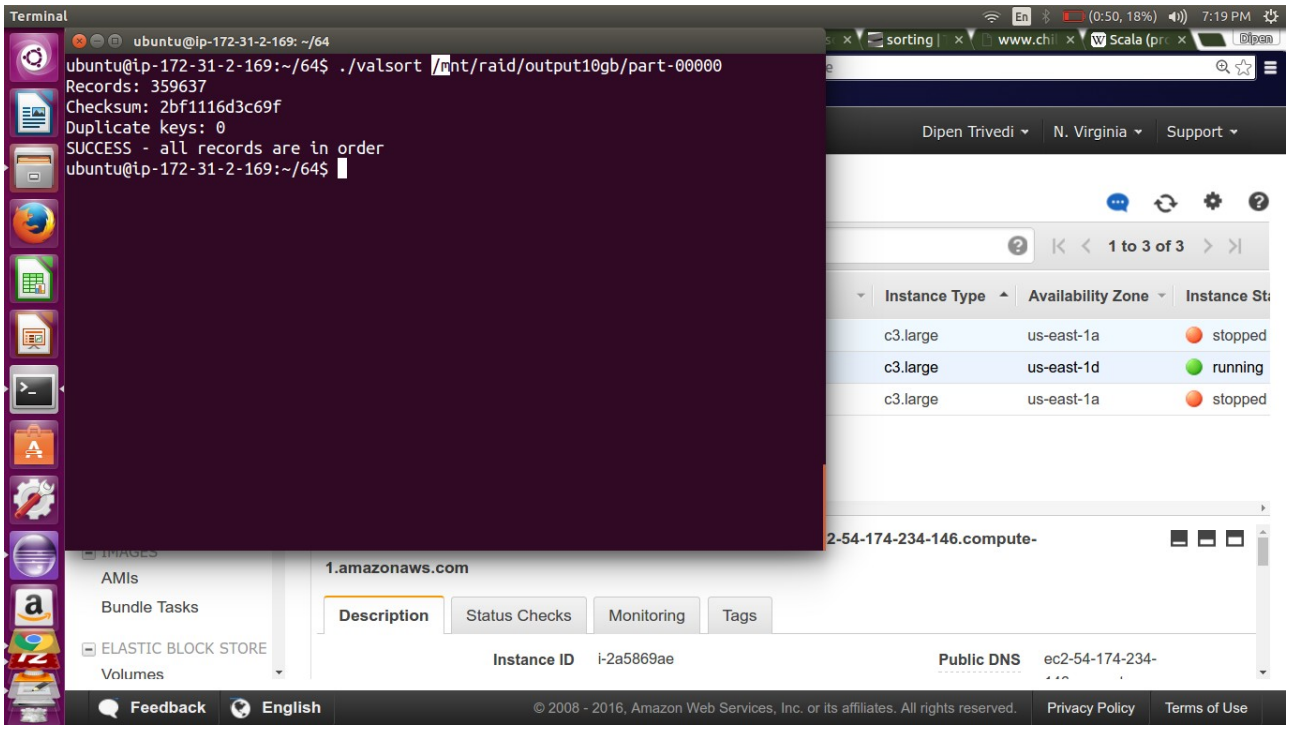
Following screen shot shows the number of files created after the program:



- First 10 records of Part-00000 file :



- Valsort of Part-00000 file :



➢ 100GB data sort(1 master, 16 slaves) :

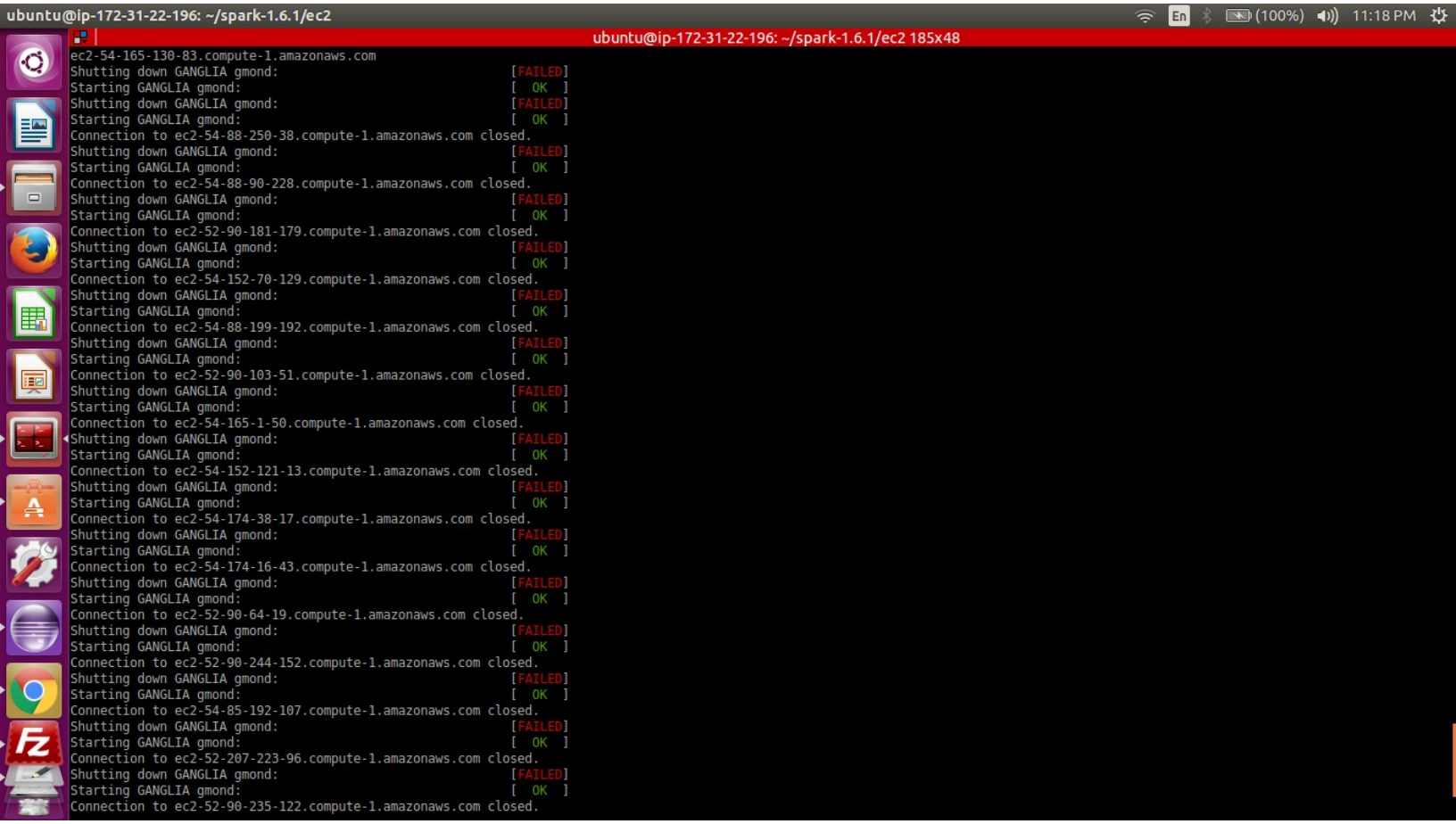
Run the "spark-ec2" file to create 1 master and 16 slave with c3.large instance type, extra data volume of 30GB and spot instance price=0.04:

- ./spark-1.6.1/ec2/spark-ec2 -k spark -i spark.pem --slaves=16 --instance-type=c3.large --ebs-vol-size=30 --spot-price=0.04 launch spark_test

In case of connection refuse run following command:

- ./spark-1.6.1/ec2/spark-ec2 -k spark -i spark.pem --slaves=16 --instance-type=c3.large --ebs-vol-size=30 --spot-price=0.04 launch spark_test --resume

After Spark finish launching clusters, the following screen will be shown:



Now, the spark setup is done. We have to do following change to hadoop's hdfs-site.xml. Following are the changes should be done in these files:

- *ephermal/conf/hdfs-site.xml*

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

Now, copy all these hdfs chages to slave with the following command:

- *./copy-dir ~/ephemeral-hdfs/conf/*

Now, go to the hdfs file system:

- *./ephemeral-hdfs/bin/hadoop/stop_dfs.sh*

Now, go to the spark file system:

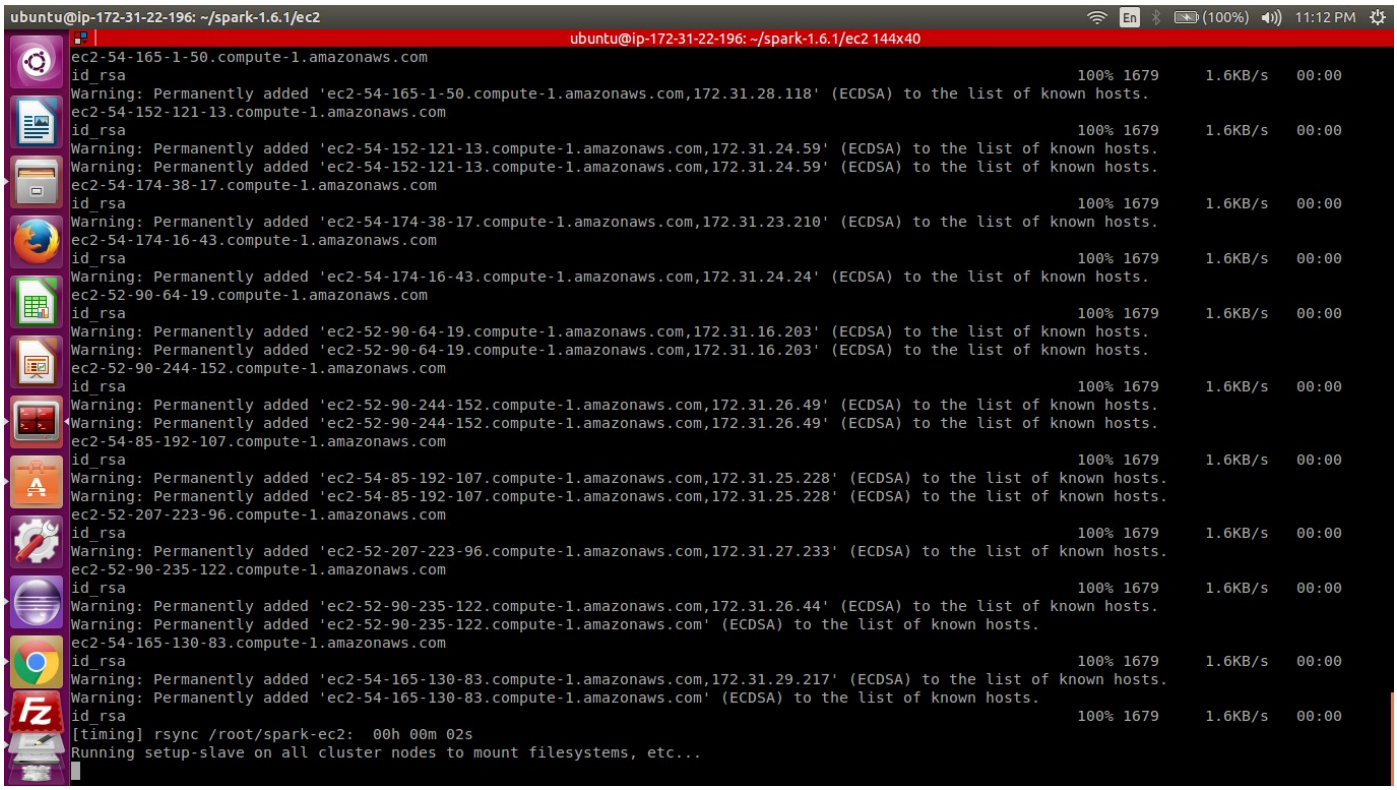
- *./spark/sbin/stop_all.sh*

Now, go to the spark file system:

- *./spark/sbin/start_all.sh*

Now, go to the hdfs file system:

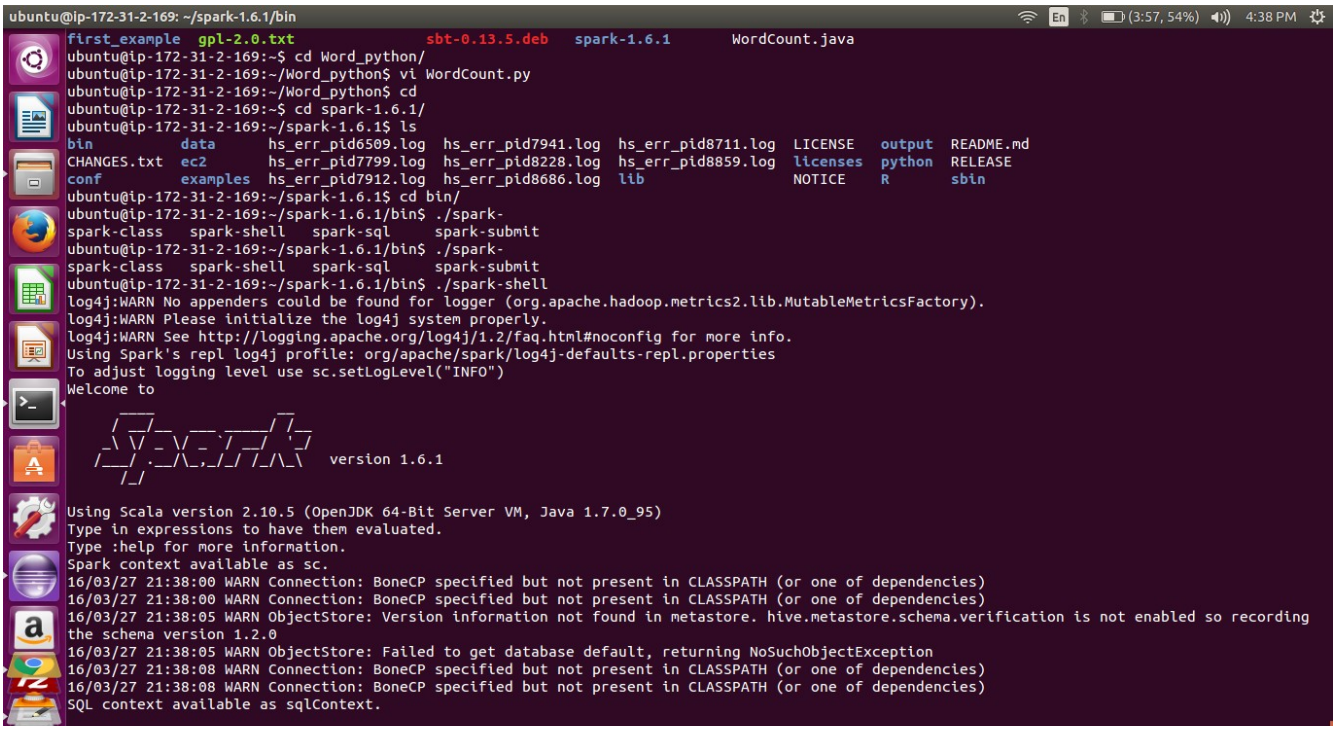
- *./ephemeral-hdfs/bin/hadoop/.start_dfs.sh*



Now, check of running daemons with following command:

- *jps*

Now, open spark shell to run the spark program :



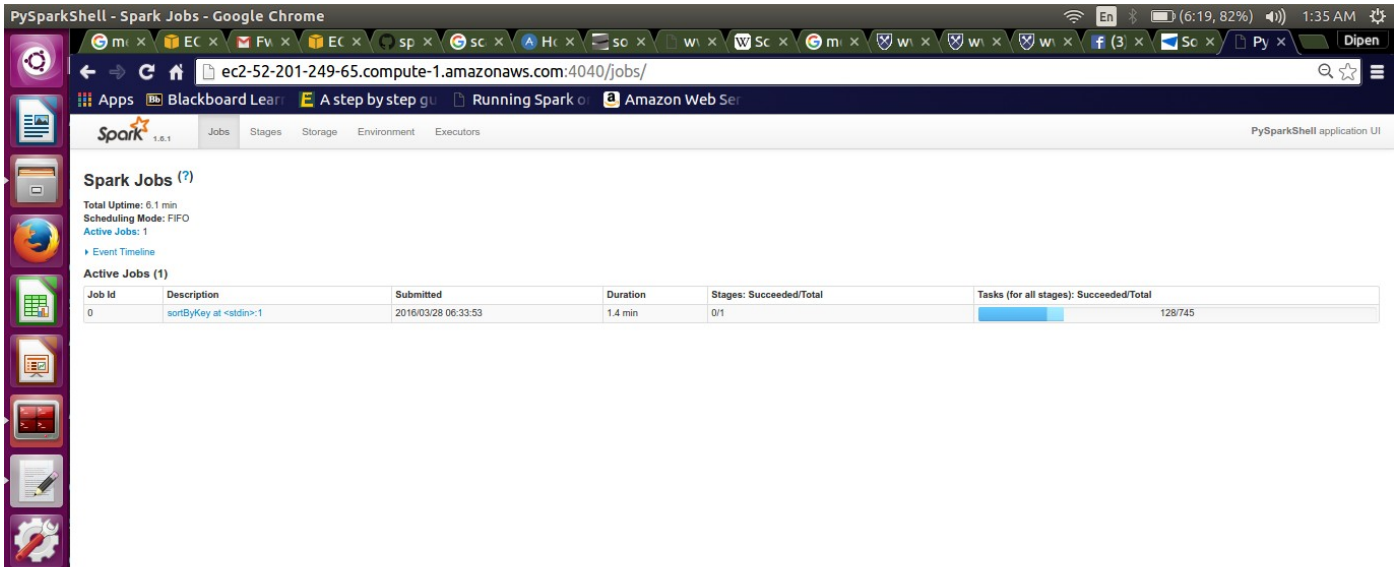
Now, run following spark program in the shell :

- *TerasortFile = sc.textFile("hdfs://PUBLIC DNS : 9000/user/hadooo/input/input_100GB.txt")*

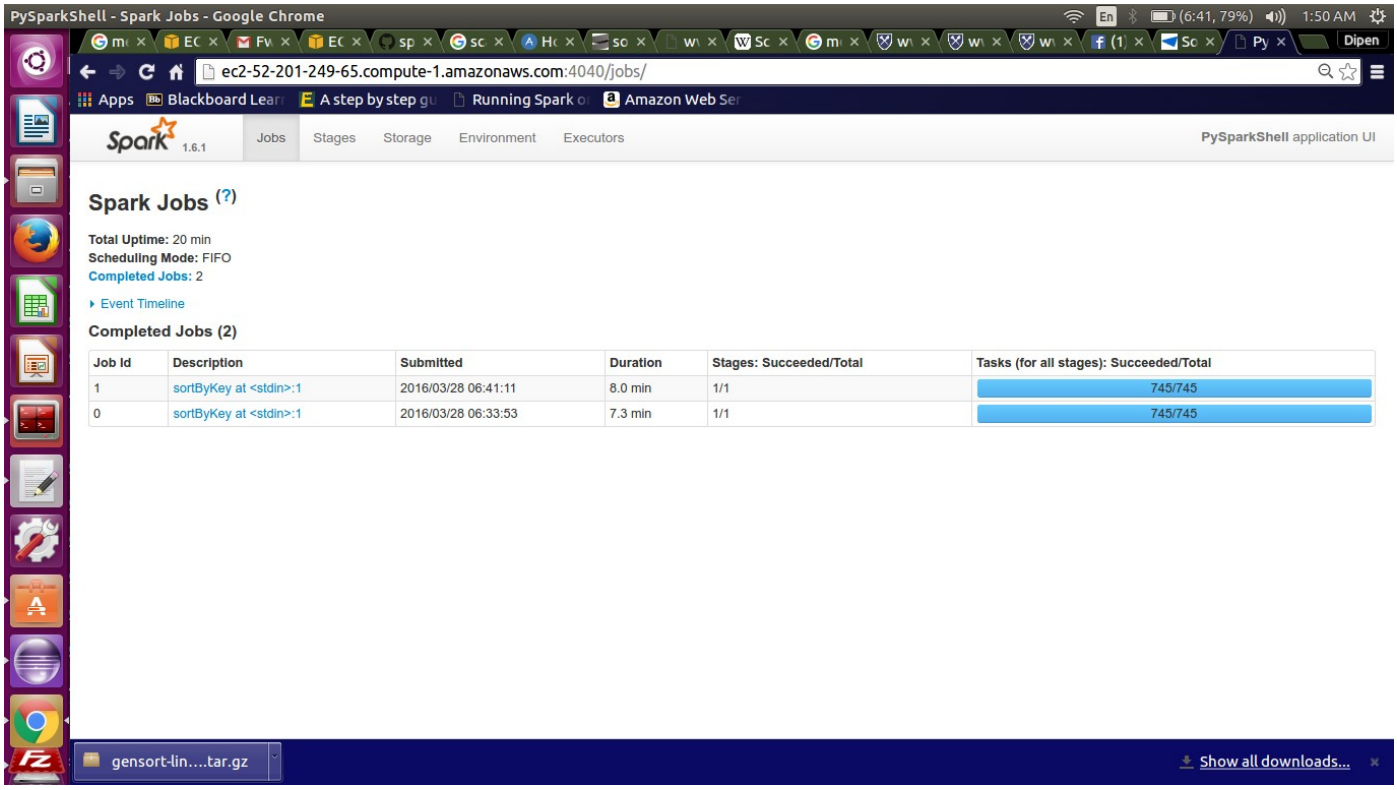
  TeraSortObj = TerasortFile.flatMap(lambda line:ine.split("\n"))
  .map(lambda dicto:
  (str(dicto[:10]),str(dicto[10:]))).sortByKey().map(lambda (a,b) : a+b)

  TeraSortObj.saveAsTextFile("hdfs://Public DNS :9000/user/hadoop/output_100GB")

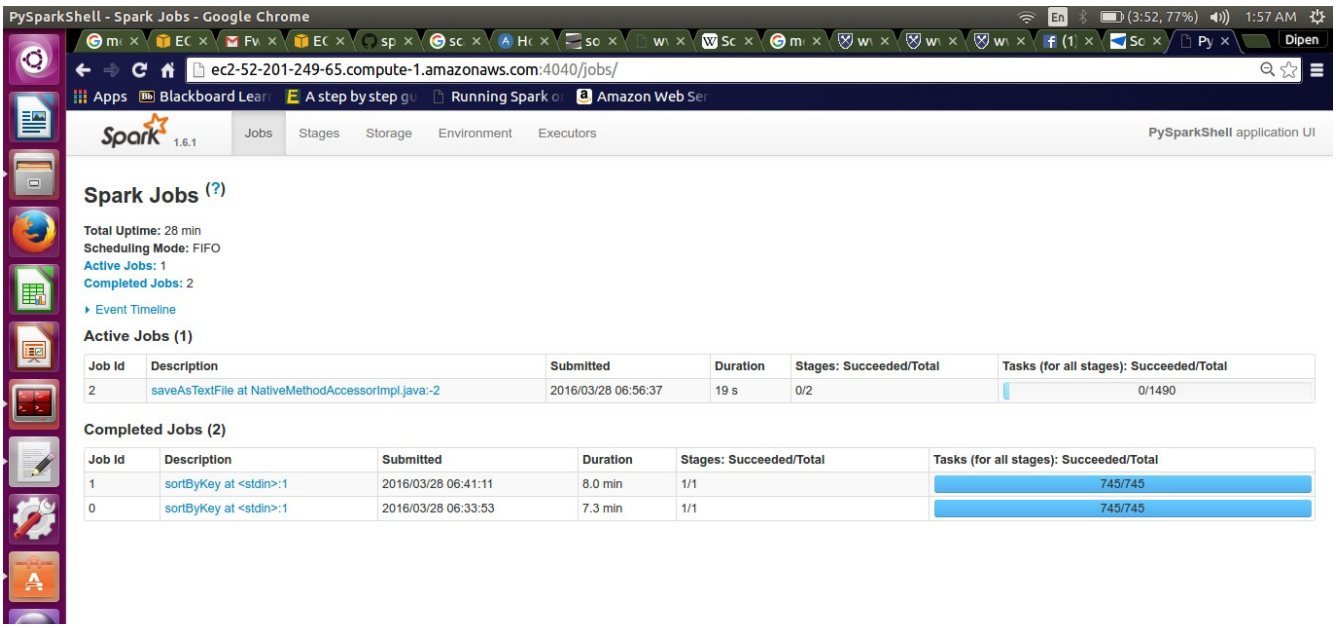
- Following Web interface shows the job task in progress:



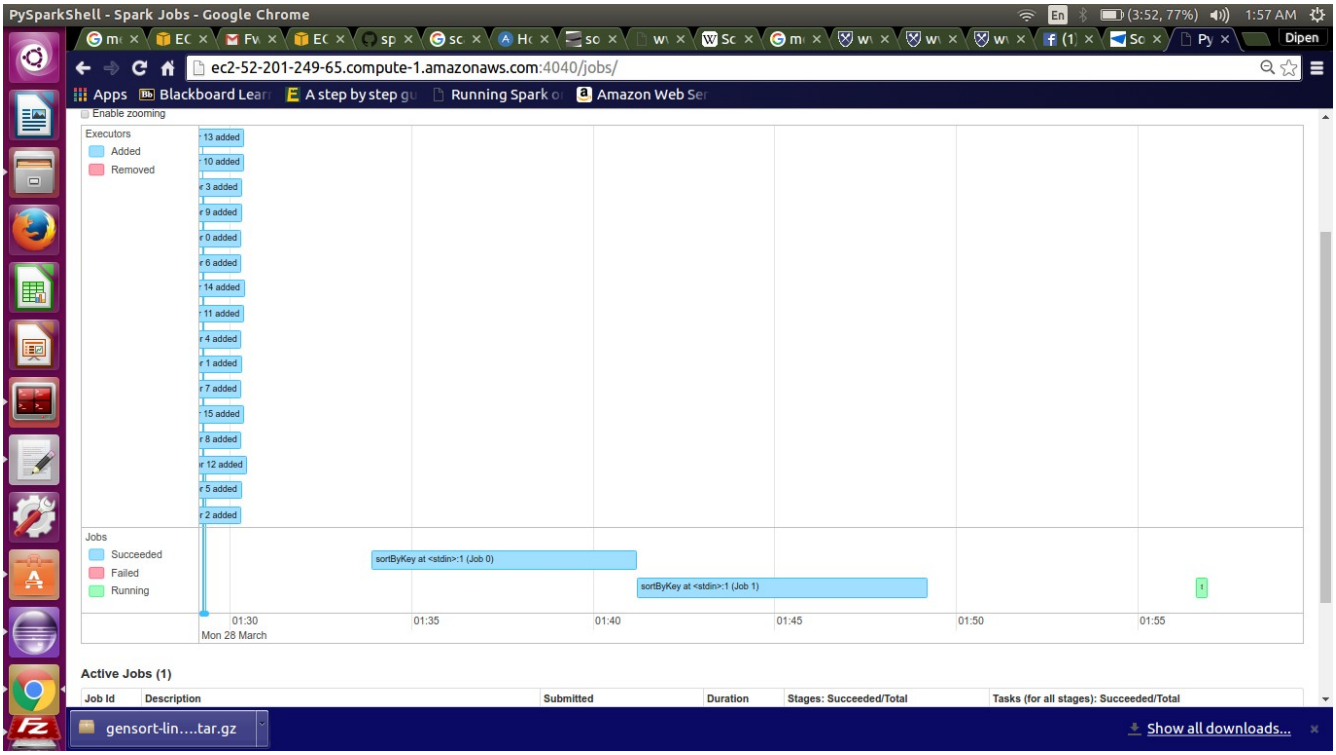
- Following Web interface shows the 100% sorting of the job task :



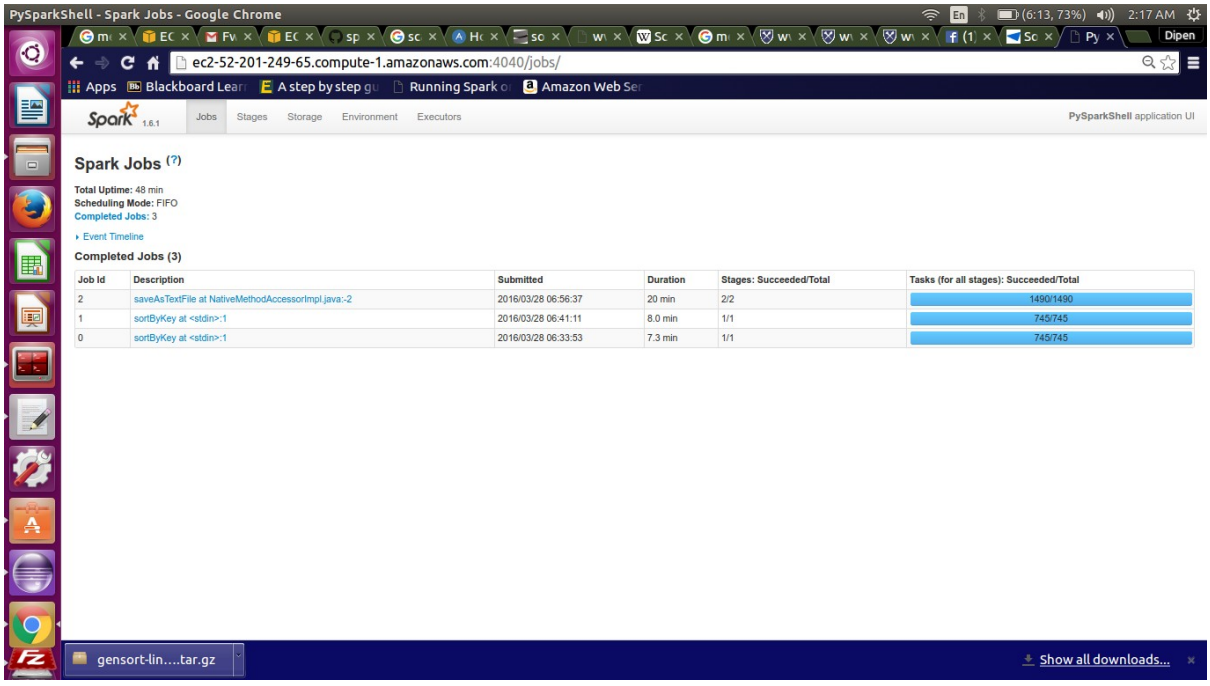
- Following Web interface shows writing in the file job task in progress:



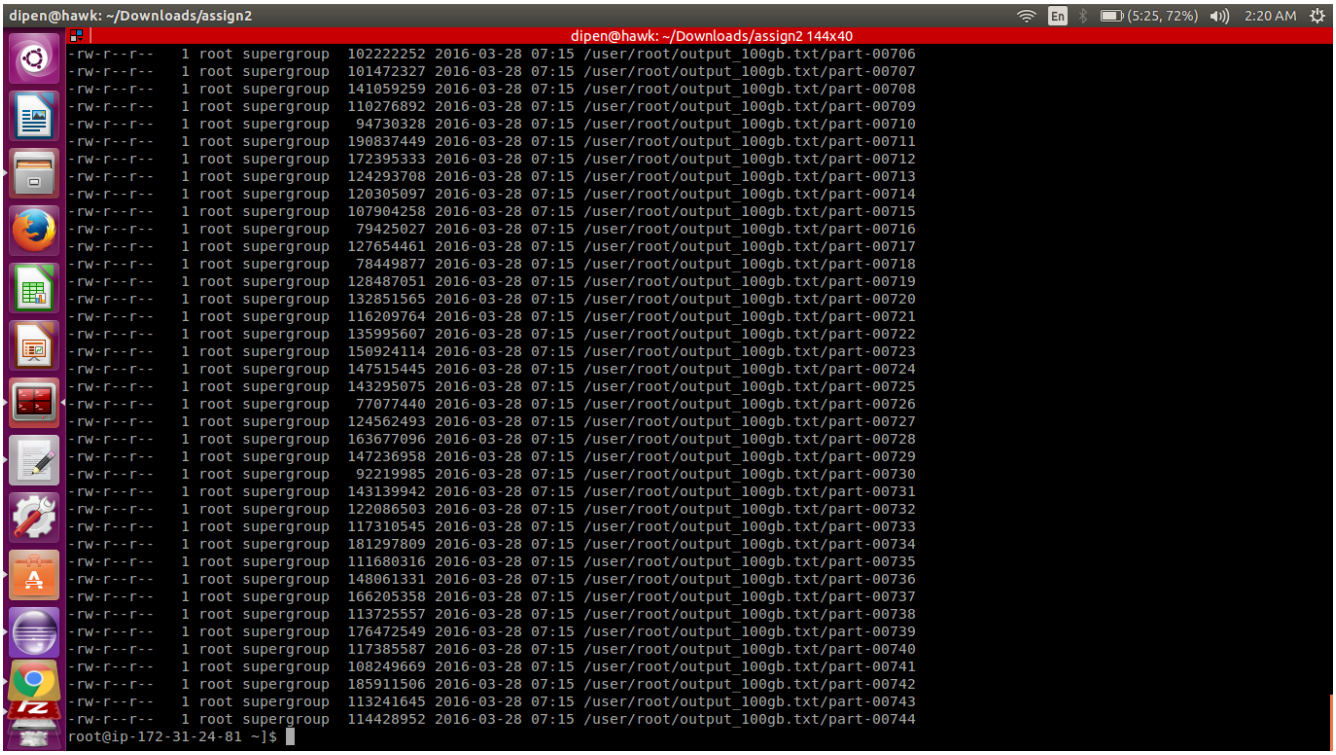
- Following Web interface shows the job task time line in progress:



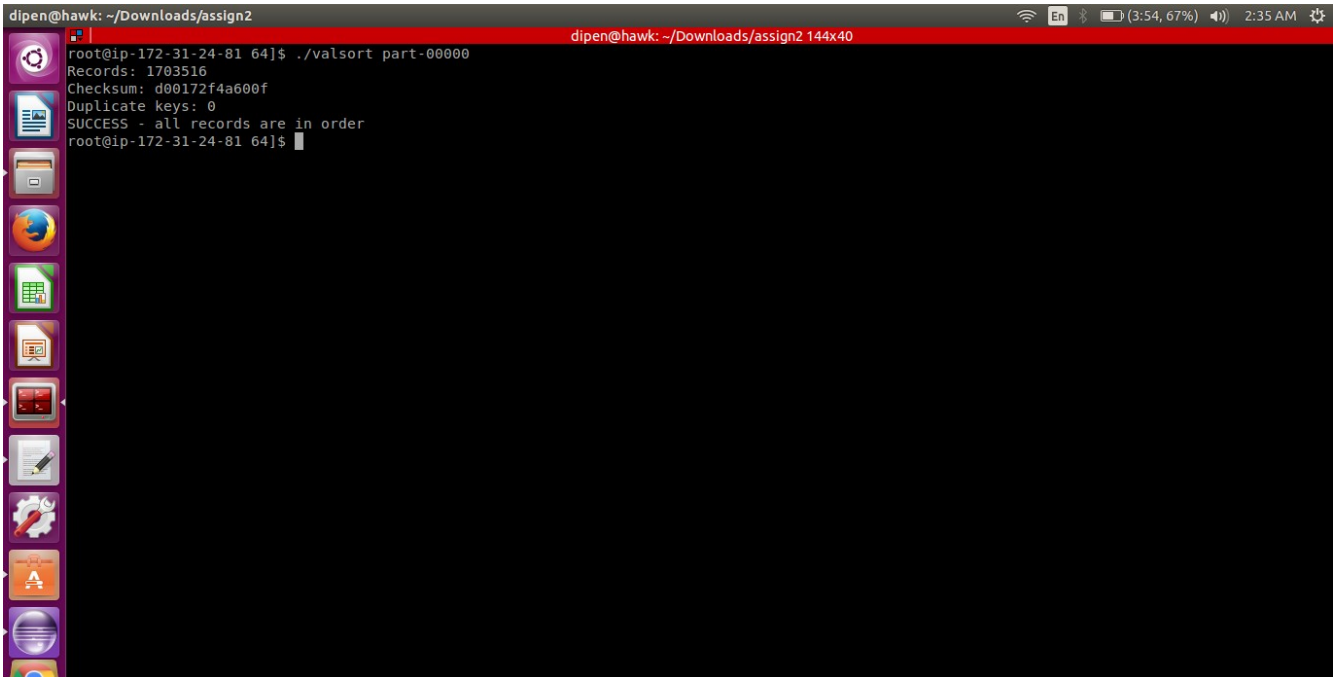
- Following Web interface shows the 100% job done and time taken to finish the job task:



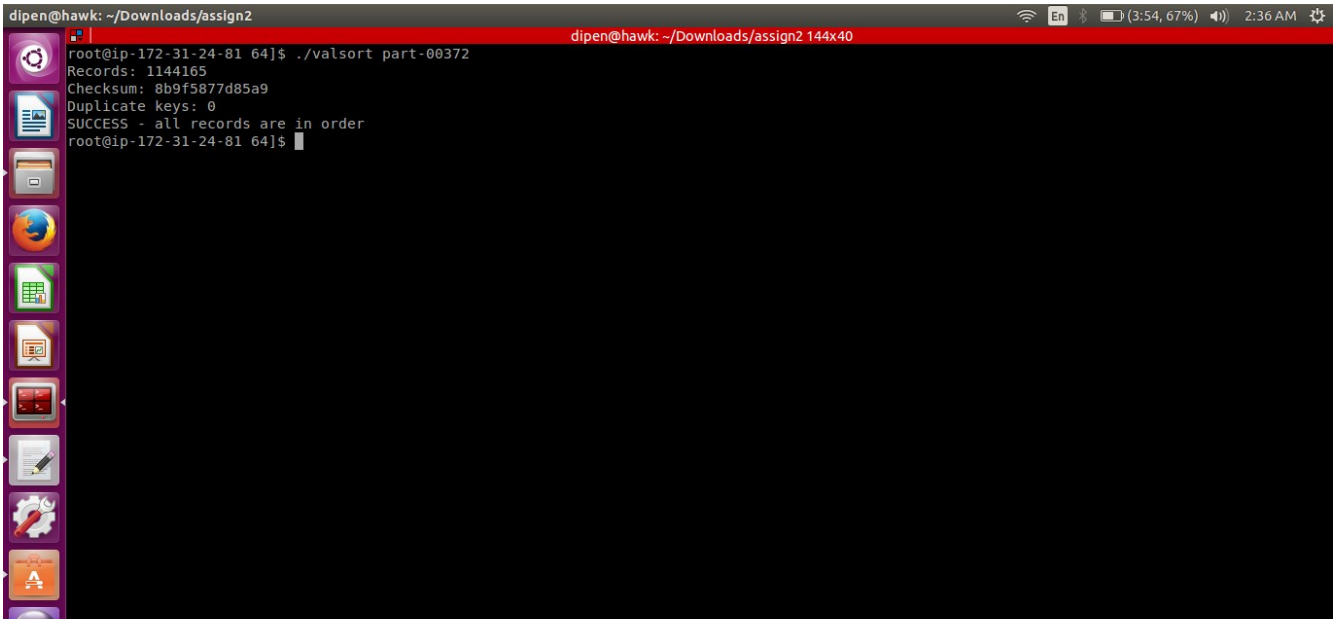
- Following screen shot shows the total 744 files created after the Job done :



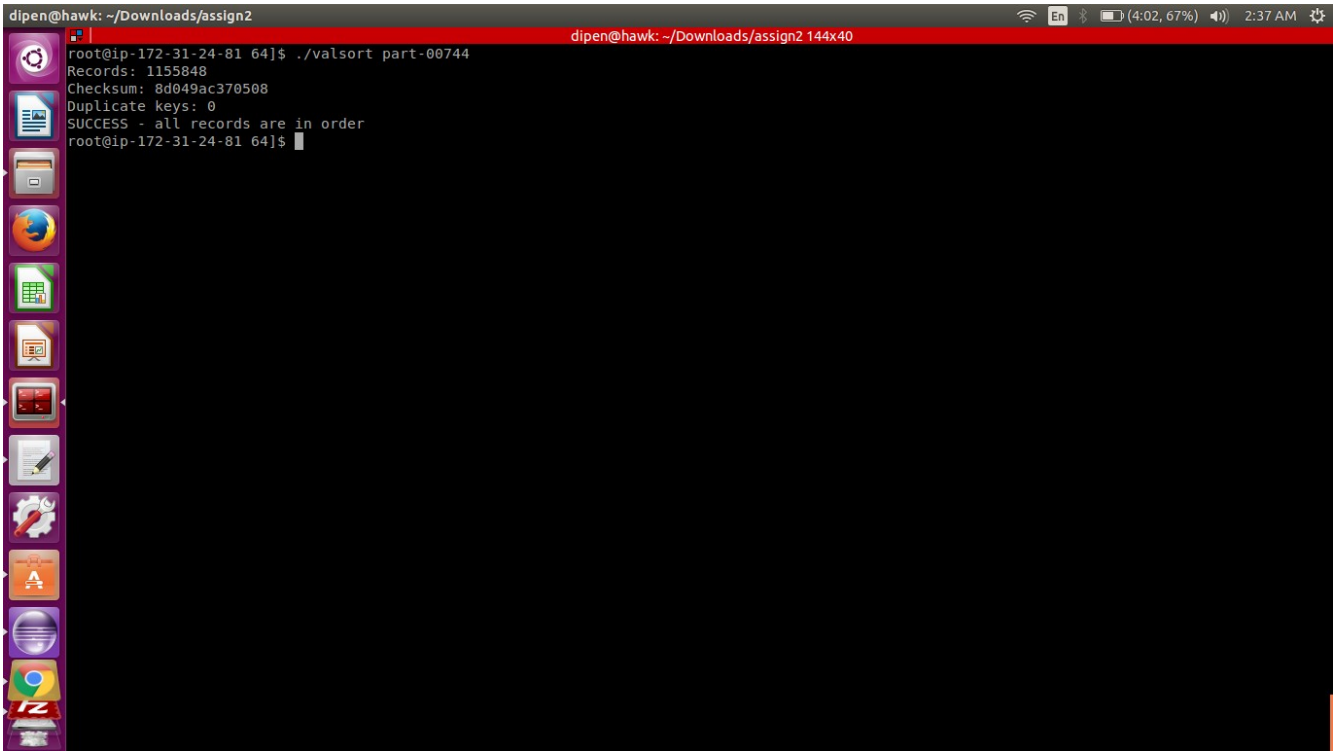
- Valsort of the output file part-00000:



- Valsort of the output file part-00372:



- Valsort of the output file part-00744:



■ Performance Evaluation Table (10GB | 100GB) :

Data size (GB)	Execution Time (seconds)	Throughput (MB/sec)
10	710	14.08
100	1640	60.59

■ Problems Faced during the Configuration and Running :

- In the **/hadoop/bin/hdfs-site.xml** change the property **dfs.replication** with default value of **3** to **1**. Because of this default value 3 replication of the data while reducing was 3 times, which was resulting in the 3 times more time to reduce the data. By changing it to one stops the data replication.

8. Shared Memory Sort:

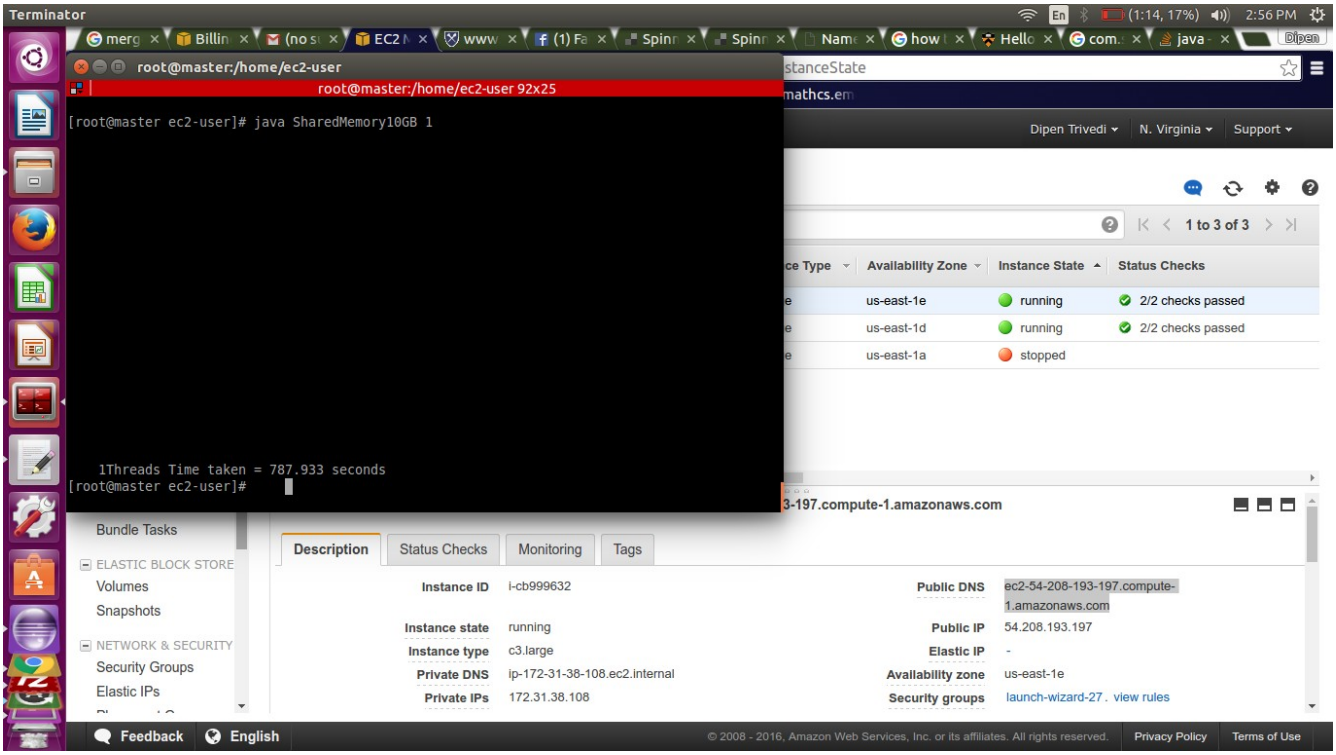
The shared memory program is created in java to perform the replication tasks of Mapping and Reducing done by the hadoop and spark. The program is performed on the 10GB data. Following is the Methodology/flow of the program:

➢ Methodology :

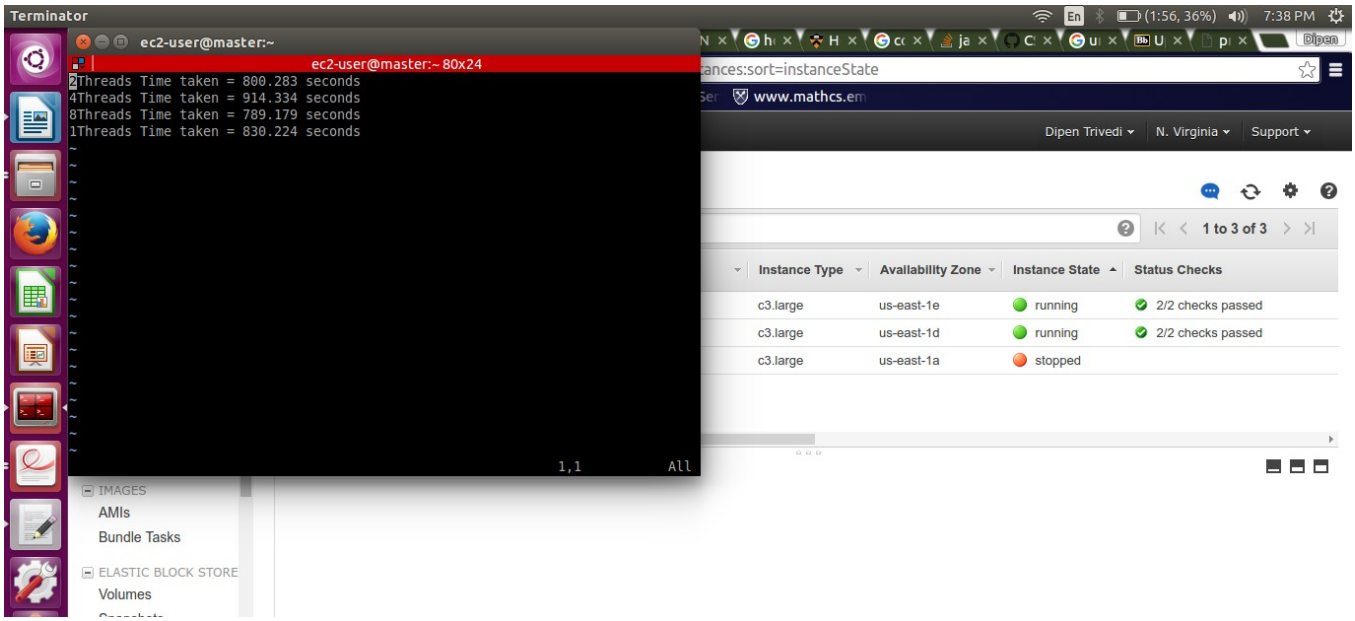
- Created input of size 10GB and call `inputFilePartion() method` to split the input into the key and value.
- Then the function will divide the input into small chunks and store the chunks into the separate temporary files.
- Created MergeSort() to sort the chunks respect to the keys and store them in the temporary files. This fuction will be called within the Thread.run() method.
- Multiple threads(1,2,4,8) are created to run the MergeSort(). For example, for two threads, the divided chunks will be sorted in the 2 separate parts and then they will be sorted in parallel.
- Store the temporary files's addresses into the ArrayList.
- Created a function `FileWrite()` to write the chunks into the temporary files.
- Created the function `outputTempFilesMerging()` to merge the sorted chunks.
- Before merging the chunks, we have to take the first line of the chunks and sort them. Take the sorted key, fetch the value according to the key from the hash map and write that value into the Output file.

➢ Program Flow :

- Use the configured instance(Page -) as mentioned above.
- Compile the java file.
  javac SharedMemory.java
- Run the java file SharedMemory file.
  java SharedMemory 1 >> output.txt
  - **Shared Memory 1-thread(10Gb data) :**

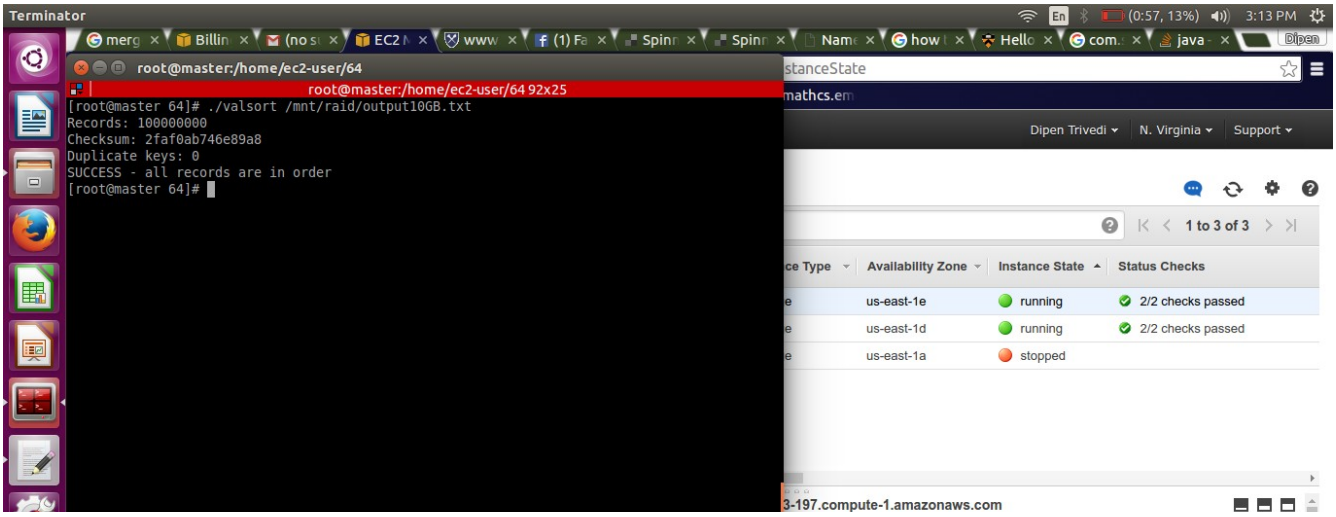


- Now, open the output.txt
  vi output.txt



- Now, run the valsort on the output file.
  .valsort /mnt/raid/output10GB.txt

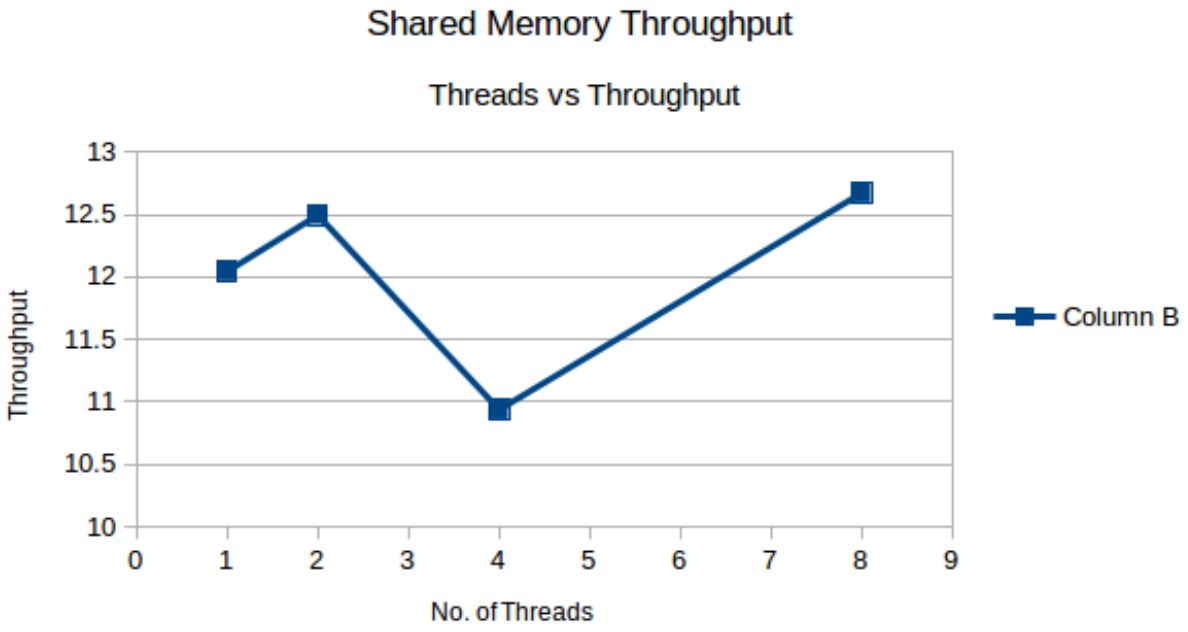
  - **<u>Shared Memory Valsort(10Gb data) :</u>**



➢ Performance Evaluation with Tables and Graph (1 GB data) :

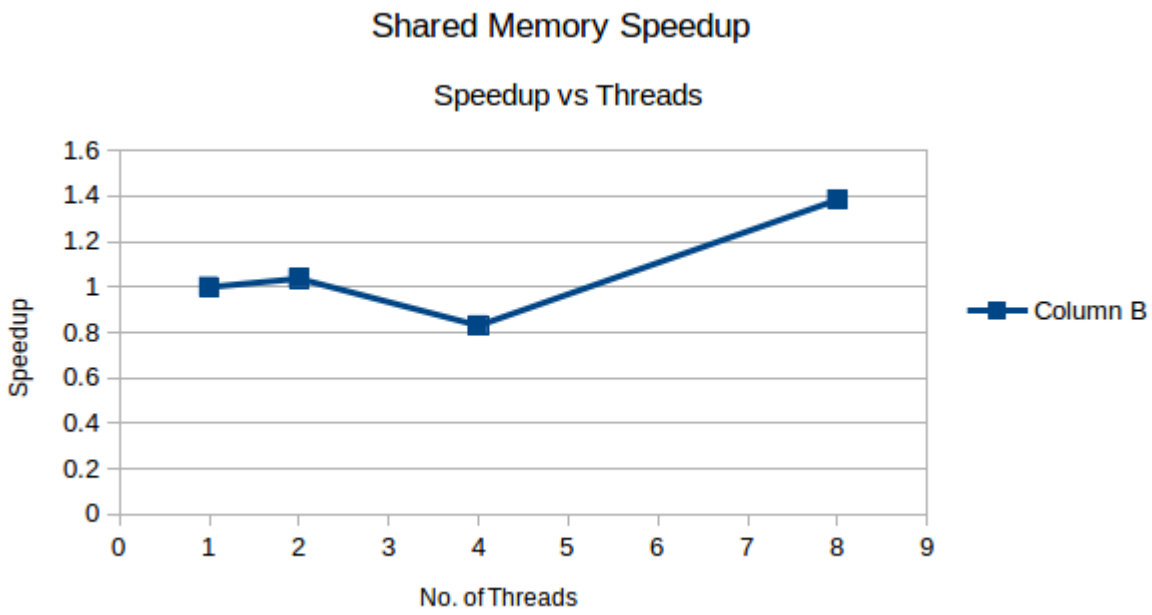
◆ Shared Memory : Throughput Table

Nubmer of Threads	Throughput
1	12.044
2	12.495
4	10.936
8	12.671



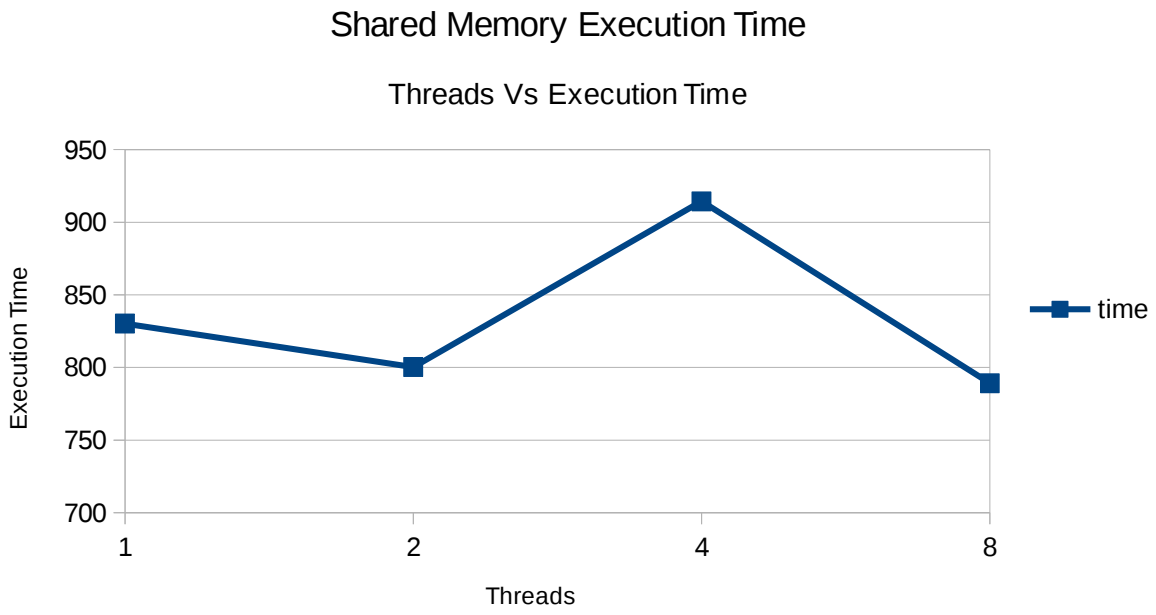
- **Speedup Table with respect to Execution Time of Thread 1:**

Nubmer of Threads	Time	Speedup
1	830.224	1
2	800.283	0.96
4	914.334	1.1
8	789.179	0.95



- **Execution Time Table (10 GB Data Size):**

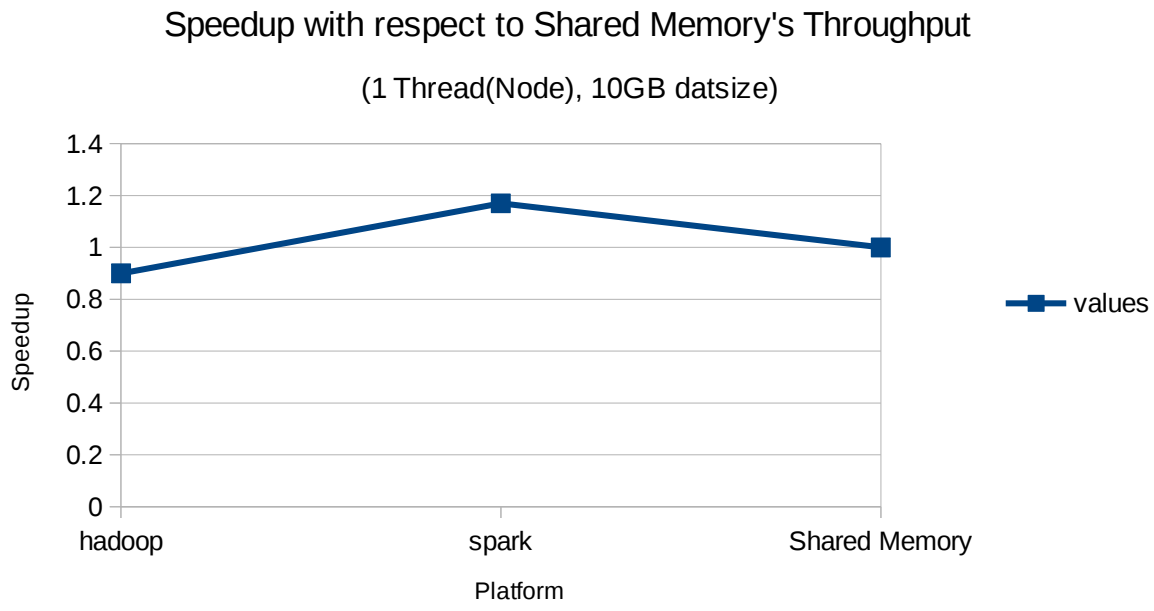
Nubmer of Threads	Execution Time (sec)
1	830.224
2	800.283
4	914.334
8	789.179



9. Performance Evaluation (Hadoop-Spark-Shared Memory) :

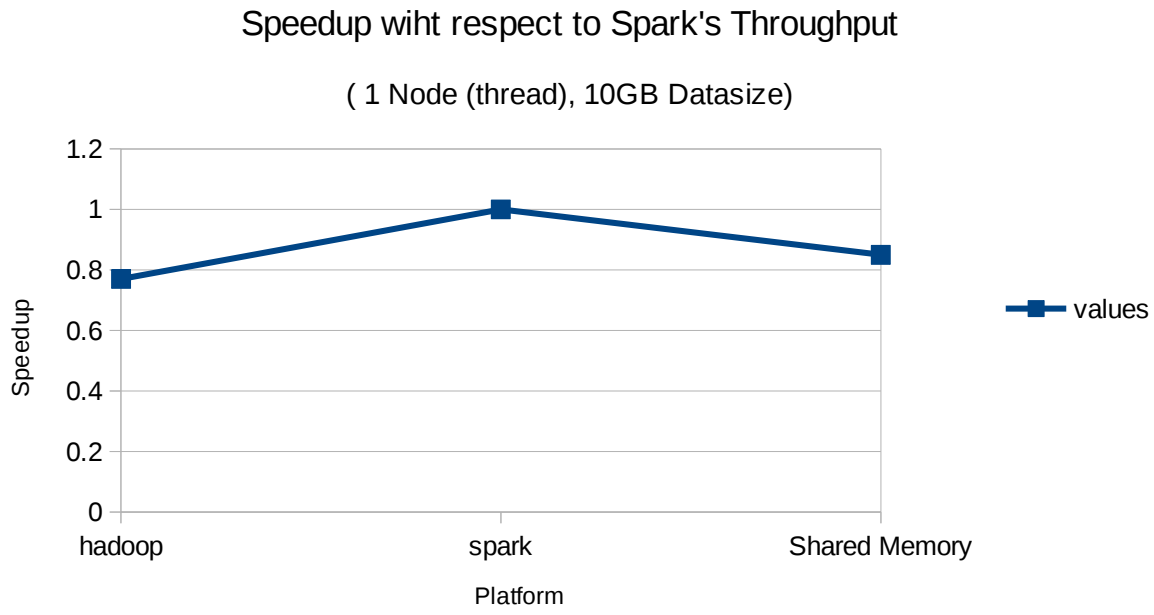
➢ Speedup with respect to Shared-Memory's Throughput : (1 Thread(node), 10GB)

Platform	Execution Time (sec)	Throughput	Speedup
hadoop	920	10.87	0.9
spark	710	14.08	1.17
Shared Memory	830.224	12.04	1



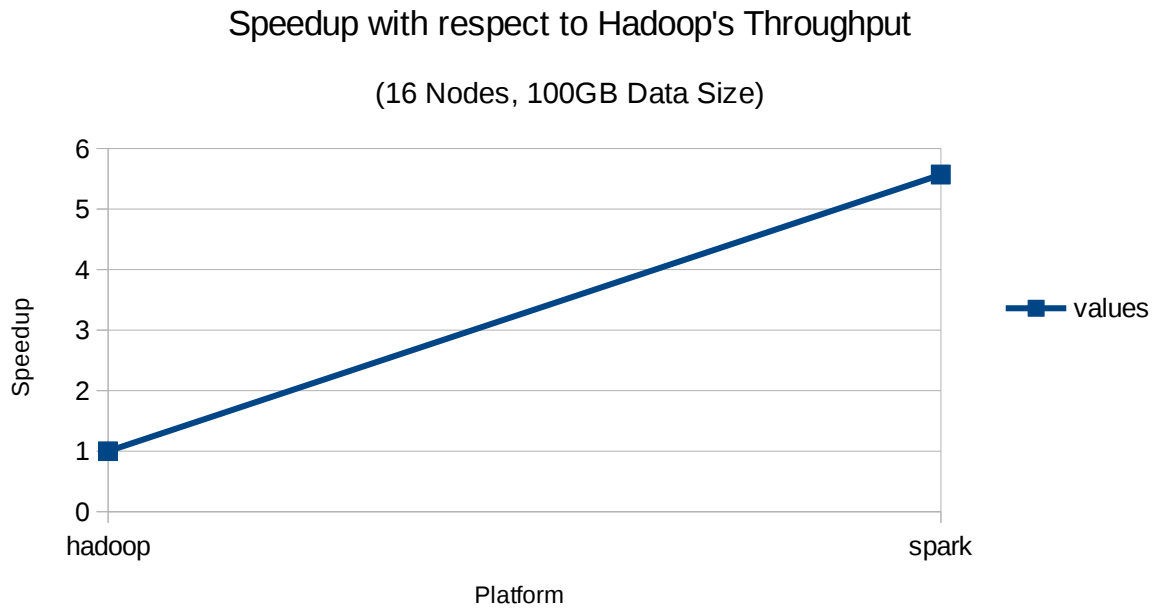
- **Speedup with respect to Spark's Throughput : (1 Thread(node), 10GB)**

Platform	Execution Time (sec)	Throughput	Speedup
hadoop	920	10.87	0.77
spark	710	14.08	1
Shared Memory	830.224	12.044	0.85



➢ Speedup with respect to Hadoop's Throughput : (16 nodes, 100GB)

Platform	Execution Time (sec)	Throughput	Speedup
hadoop	4266	10.87	1
spark	1640	60.59	5.57



■ Question :

1) Which seems to be best at 1 node scale?

Ans :

Spark performs best at 1 node compare to hadoop and shared memory which can be seen from the shared memory graph.

2) How about 16 nodes?

Ans :

From the speedup comparison graph of hadoop vs spark for 16 node, it is clear that spark is 5.57 time faster than the hadoop.

3) Can you predict which would be best at 100 node scale?

Ans :

From the above comparisons I can say that spark will perform better for the 100 node cluster, because spark uses the intermediate cashed data. Only problem will be the memory storing.

4) How about 1000 node scales?

Ans :

I guess it depend on what scale(cost or time) we are considering. If we consider cost than definitely spark is a bad option as we will have storage issue.

But if consider time than spark is much much faster compare to the hadoop. So, it depends on the user's perspective.

10. Conclusion :

By the end of this programing assignment and the above graphs, it was concluded that the shared memory for 1 thread take lesser time compare to Hadoop for (1 master, 1 slave) and more time than Spark (1 master, 1 slave). The performance of spark is better than the performance of hadoop in the Amazon AWS EC-2 instances.

We used d2.xlarge, c3.large and c3.4xlarge during this assignment. Which helped us to know more about these instances. We learned about the spot instances and on-demand instances. How and when to use EBS storage. I

In this assignment, we learned how to configure hadoop and spark on the single and multiple node cluster. There is no doubt that without script to perform such configuration is very hard as well as tedious specially for 16 node configuration. I made a script for hadoop which unmount the mounted drive, mounts the unmounted drive, changes the hosts and slaves and formats the whole HDFS namenode. But I was not able to make this script run for all the multiple node with a single run. I had to go to each node and run this bash file.

Yes, It was a lot of work that I had to for the configuration, but at the end I learned how the hadoop and spark works and how to manage the storage in the cloud.

11. REFERENCES:

[1] AMAZON-EC2 VIRTUAL SERVER HOSTING
[2] www.Eduonix.com/hadoop-cofiguration
[3] www.geeksforgeeks.org/mergeSort
[4] http://spark.apache.org/downloads.html
[5] http://spark.apache.org/docs/latest/cluster-overview.html
[6 spark/configuration
[7] http://hadoop.apache.org/