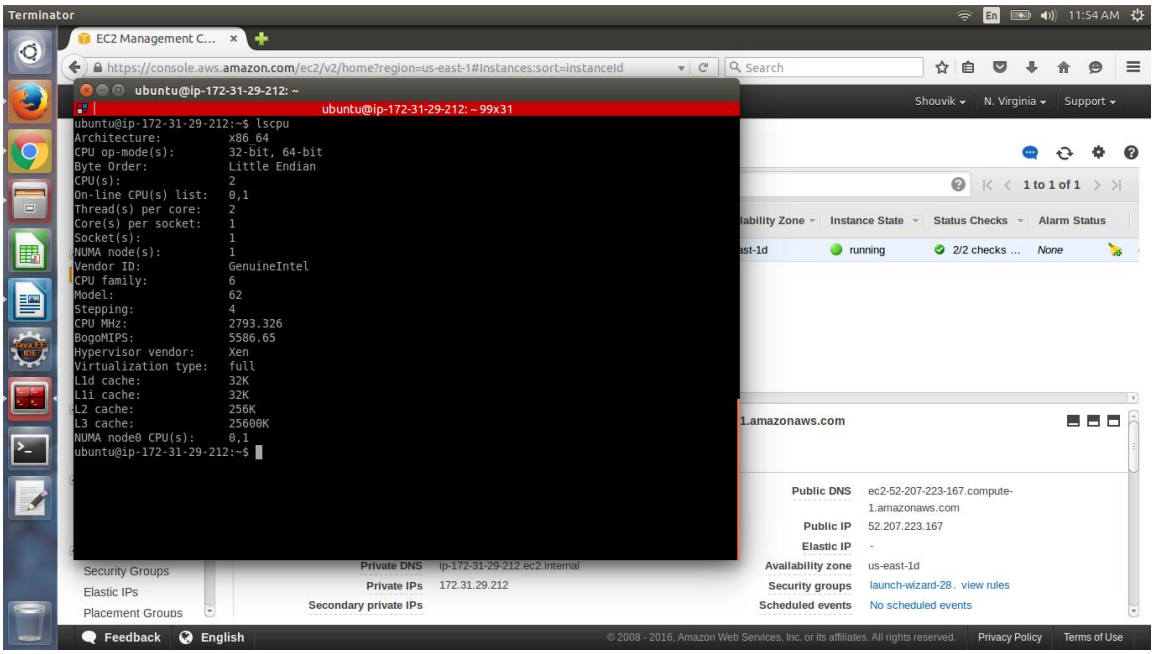
CS 553 Programming Assignment#2 Sort on Hadoop/Spark and Shared-Memory Sort

This document lists the various aspects which needed to be accomplished for successful completion of assignment:

- Setting Virtual Cluster (1 node and 16 nodes)
- Implementing Shared-Memory Sort for 10GB in Java
- Install and Configure Hadoop
- Implementing 10 and 100 GB sort in Hadoop
- Install and Configure Spark
- Implementing 10 and 100 GB sort in Spark
- Measuring the performance of the above task.

1 EXPERIMENTAL ENVIRONMENT

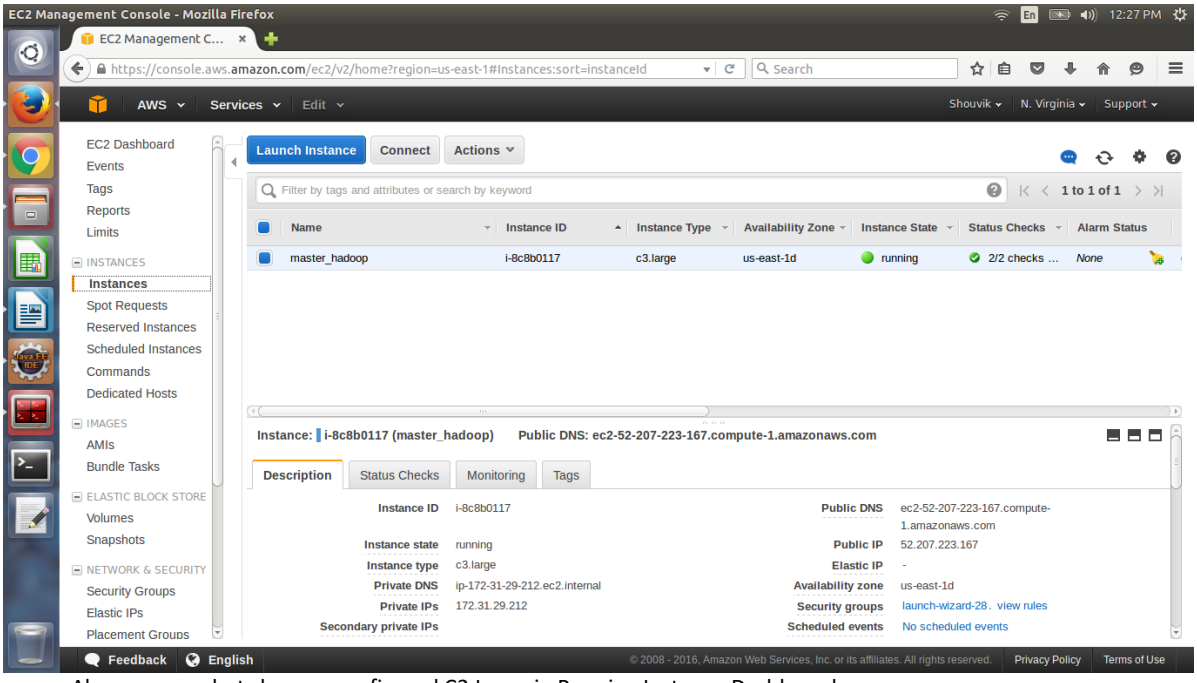
I would be running my experiments of on Amazon EC2 C3.Large cloud instance. Below screenshot provides the details of our Experimental Environment (1 node).
Java Version: 1.7.0_95
Hadoop Version: 2.7.2
Spark Version: 1.6.1



2 SETTING VIRTUAL CLUSTER

For setting up virtual cluster follow the below steps:

1. Log into Amazon AWS service Website (http://aws.amazon.com)
2. Select EC2 from the compute section.
3. From EC2 dashboard select Instance (You can also go for spot requests).
4. Inside Instance select Launch Instance.
5. In instance type selection screen select “C3.Large” instance. Here if you need a spot instance then you select appropriate checkbox.
6. Add storage to our selected instance.
7. In Security group, we add and configure TCP and ICMP protocol to use public IP address.



Above screenshot shows a configured C3.Large in Running Instance Dashboard.

Steps to mount unmounted disk space and renaming it to /mnt/raid:

1. Type below command to see the size of unmounted partitions:
   >>lsblk
2. Merge and join two unmounted partitions using below commands:
   >>sudo apt-get install mdadm
   >>sudo mdadm --create --verbose /dev/md0 --level=0 --name=hadoop --raid-devices=2 /dev/xvdb /dev/xvdc

3. Label the joined partition with a specific name using below command:
   >>sudo mkfs.ext4 -L hadoop /dev/md0
4. Create a directory which needs to be mapped with the mounted disk:
   >>sudo mkdir -p /mnt/raid
5. Mount the newly created space to the directory that we created in Step4:
   >>sudo mount LABEL=hadoop /mnt/raid
6. To check the mounted path enter the below command:
   >>df -h
7. Grant permissions to the “ubuntu” user using below command:
   >>sudo chown -R ubuntu:ubuntu mnt

Screenshots for steps for mounting unmounted disk space and renaming it to /mnt/raid:

```
ubuntu@ip-172-31-21-202:~$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
xvda    202:0    0     8G  0 disk
└─xvda1 202:1    0     8G  0 part /
xvdb    202:16   0    30G  0 disk
xvdc    202:32   0  15.3G  0 disk
ubuntu@ip-172-31-21-202:~$ sudo mdadm --create --verbose /dev/md0 --level=0 --name=hadoop --raid-devices=2 /dev/xvdb /dev/xvdc
mdadm: chunk size defaults to 512K
mdadm: /dev/xvdc appears to contain an ext2fs file system
    size=15996416K  mtime=Thu Jan  1 00:00:00 1970
Continue creating array? yes
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
ubuntu@ip-172-31-21-202:~$ sudo mkfs.ext4 -L hadoop /dev/md0
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=hadoop
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=128 blocks, Stripe width=256 blocks
2967888 inodes, 11863168 blocks
593158 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
363 block groups
32768 blocks per group, 32768 fragments per group
8176 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
        4096000, 7962624, 11239424

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

ubuntu@ip-172-31-21-202:~$ sudo mkdir -p /mnt/raid
ubuntu@ip-172-31-21-202:~$ sudo mount LABEL=hadoop /mnt/raid
```

```
ubuntu@ip-172-31-21-202:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            1.9G   12K  1.9G   1% /dev
tmpfs           377M  352K  377M   1% /run
/dev/xvda1      7.8G  1.8G  5.6G  25% /
none            4.0K     0  4.0K   0% /sys/fs/cgroup
none            5.0M     0  5.0M   0% /run/lock
none            1.9G     0  1.9G   0% /run/shm
none            100M     0  100M   0% /run/user
/dev/md0         45G   52M   43G   1% /mnt/raid
ubuntu@ip-172-31-21-202:~$ cd ..
ubuntu@ip-172-31-21-202:/home$ cd ..
ubuntu@ip-172-31-21-202:/$ sudo su
root@ip-172-31-21-202:/# chown -R ubuntu:ubuntu mnt
root@ip-172-31-21-202:/#
```

3 IMPLEMENTING SHARED-MEMORY SORT FOR 10GB

In this task I created a multithreaded Java program that sorts a 10GB file. I have printed the output into a file based on the varying number of threads.

Program Logic:

- The main computation in my program is done by two functions namely, "*readAndPartition*" and "*mergeTempFiles*"
- The "*readAndPartition*" function reads the raw input file line by line and creates a arraylist in the memory. I allow this arraylist to grow for 20000 lines. Once the arraylist size reaches 20000, I pass the arraylist to "mySortClass" which extends from thread. This class sorts the arraylist in parallel. So if I have 2 threads as input then each thread gets 10000 records from arraylist, sorts it and writes it into a temporary file. Thus, after completion of this function, I get small sorted chuck of files. This function returns the total number of intermediate files that gets created.
- The "*mergeTempFiles*" function takes the total number of intermediate files, the temporary directory path and final output file name. This function, based on the total number of intermediate files, creates an array of BufferReader objects for each file. Also, I create a TreeMap datastructure which reads and stores the first line of all the BufferReader objects created from the intermediate files. The advantage of putting the first line of all intermediate files into TreeMap is that it internally sorts them while storing. So the first element of the TreeMap would be the smallest line among all intermediate files. I remove this element from the tree and write it into the output file. I read the next line of the particular file that got removed and add it up in the TreeMap object.
- At the end of "*mergeTempFiles*" function I have a new output file that is in sorted order. This function after creating the output file deletes all the intermediate files similar to Map Reduce.

Program Flow:

1. Install gensort to generate the input file of 10 GB using:

   ```
   >>wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
   >>tar -xvzf gensort-linux-1.5.tar.gz
   ```

2. Create a 10GB input file using gensort program.

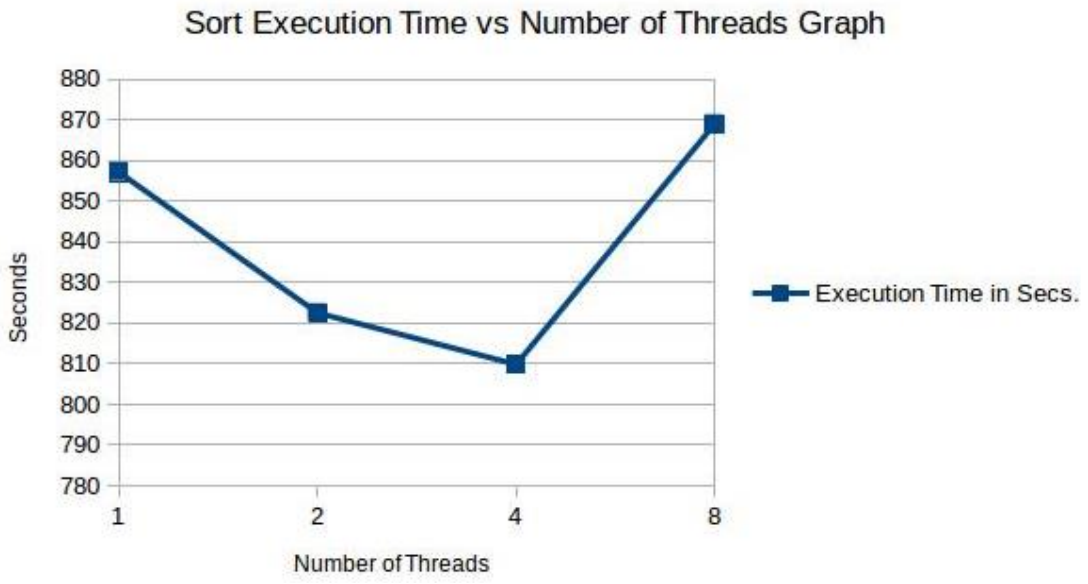
   ```
   >> cd 64/
   >>./gensort -a 100000000 [inputfileName]
   ```

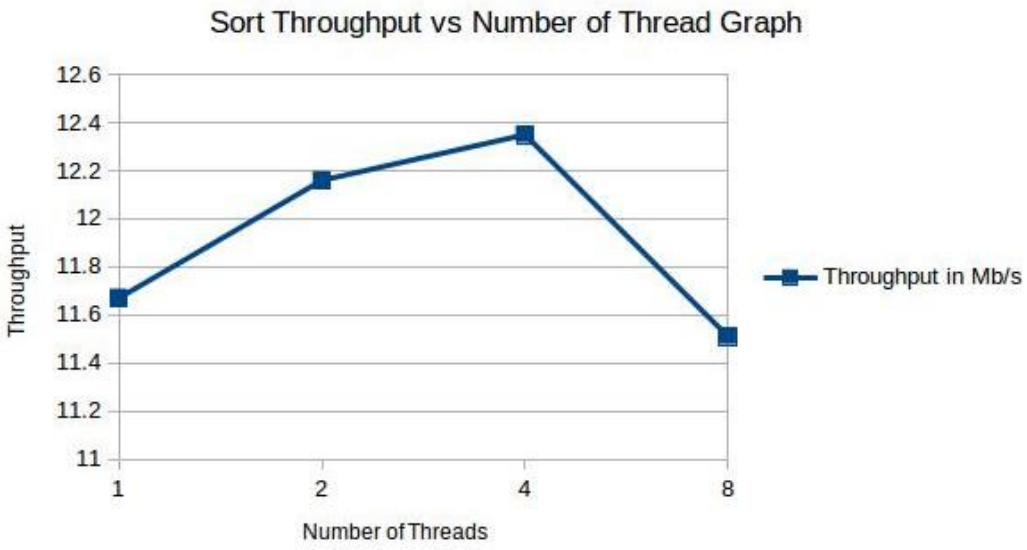
3. I have hard coded the input file, temporary file directory and output file name in my program.
4. I take the number of threads as a command line argument for the program.

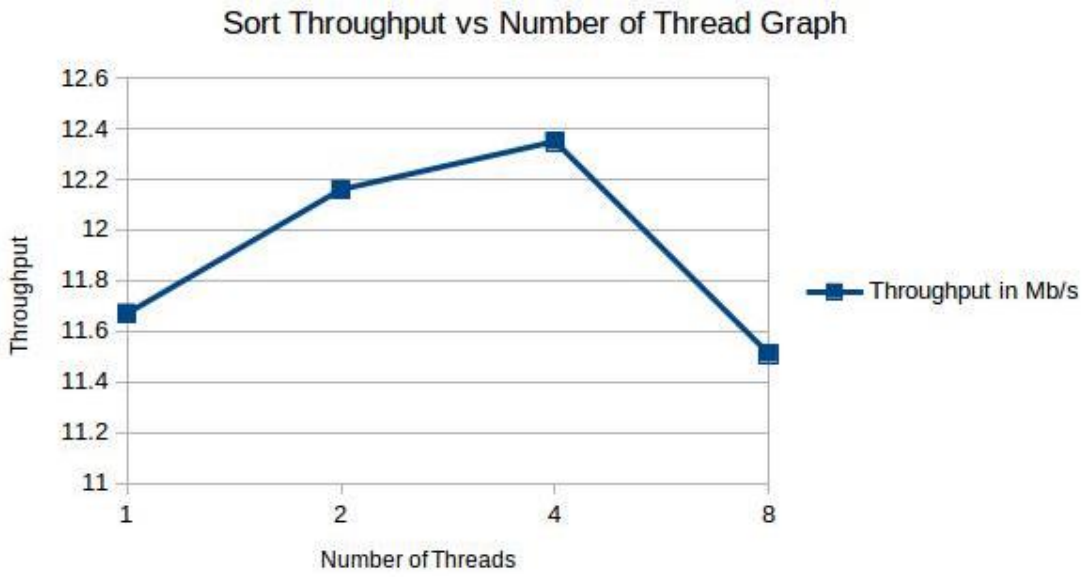
5. First I call a user defined function named ”*readAndPartition*” which takes source file name, temporary files directory path and number of threads as input.
6. The “*readAndPartition*” function returns the total number of intermediate files that got created by this function.
7. I then call “*mergeTempFiles*” function which takes the total count of intermediate files, temporary files directory path and output file name, merges all the intermediate files present in the temporary file folder and creates a fills in data into the output file.
8. Once the output file is created completely by “*mergeTempFiles*” function, it deletes all the intermediate files.
9. After the program has we get an output file which is in unix format (ASCII text). We need to convert this file into DOS format (ASCII text, with CRLF line terminators) before passing to valsort for final checkup.
10. The command to convert from Unix to Dos format is as below:
    >>unix2dos <outputFileName>
11. Later we validate the correctness of our output file by running the valsort program.
    >>./valsort <outputFileName>

Outputs (10 GB Input File):

No. of Threads	*Execution Time in Secs.*	*Throughput in Mb/s*	*Speed up (w.r.t 1thread throughput)*
1	857.141	11.67	1
2	822.525	12.16	1.042
4	809.854	12.35	1.058
8	869.015	11.51	0.986







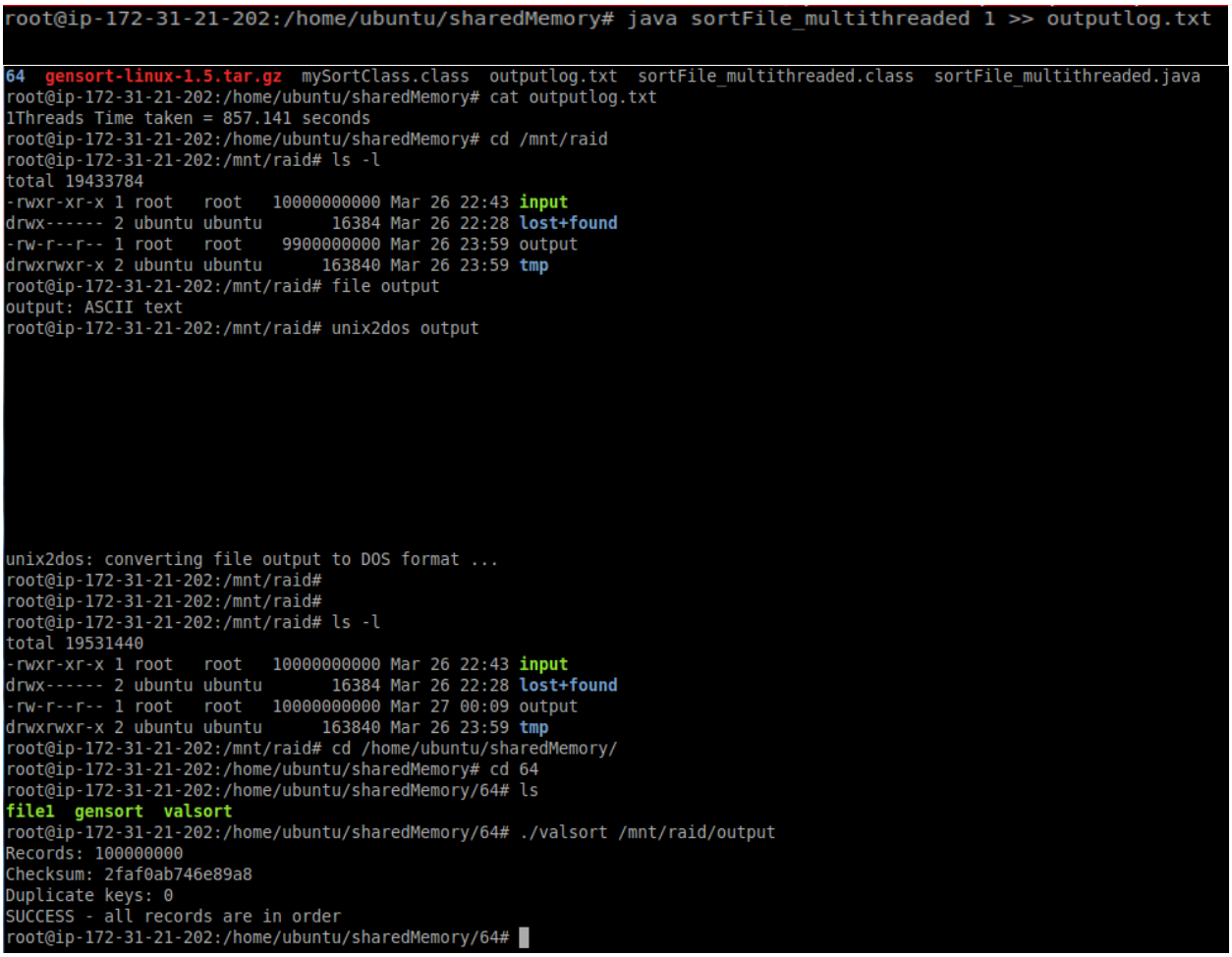
Result Interpretation:

The program performs optimally upto 4 threads as visible through the plotted graphs and tables. The programs performance drops drastically with 8 threads. As per the program logic, with the increase in number or threads there is an increase in the number of intermediate file of smaller size. This results in increases the overhead of read and write which adversely affects the programs performance as disk read writes are resource intensive.

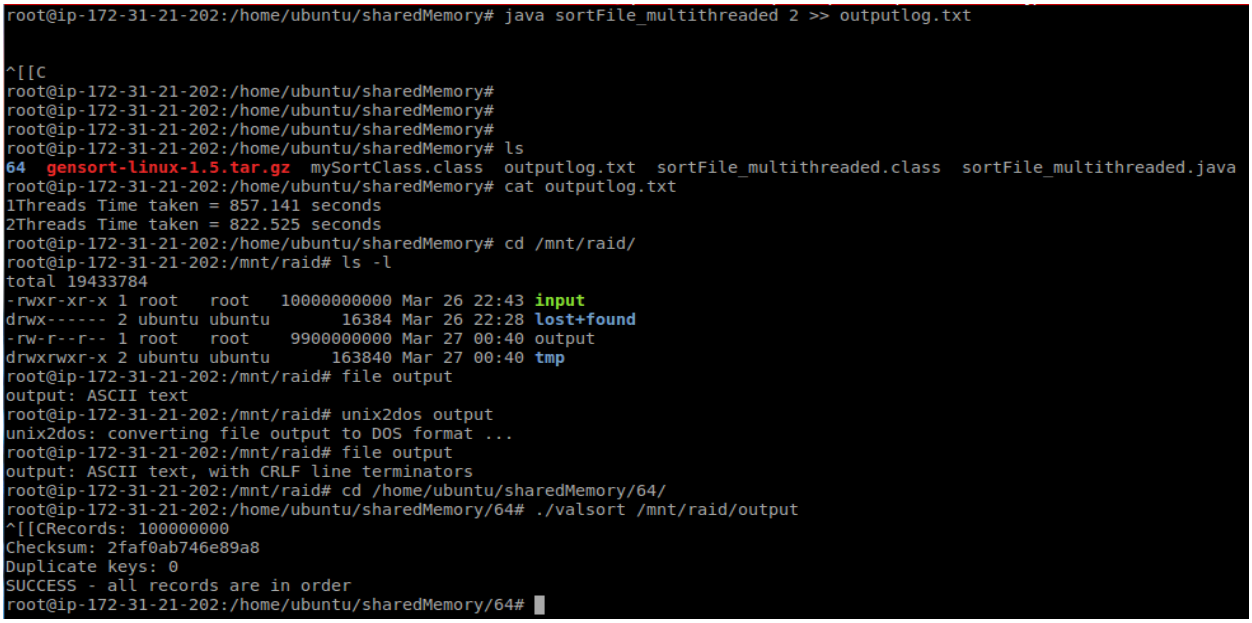
Formula:

Throughput = Sorting File Size (in Mb's)/ Time Taken to sort (in Secs)

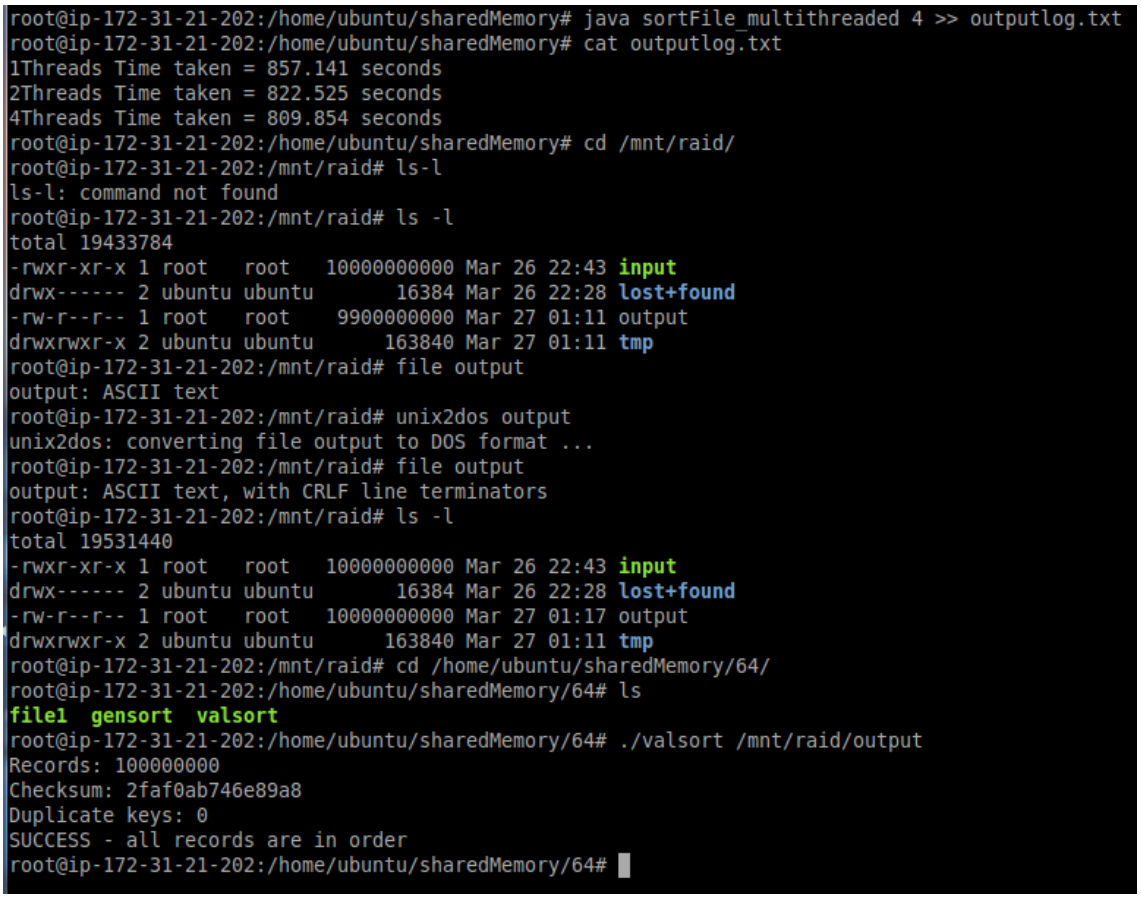
Screenshots for Shared-Memory Sort for 1 Thread:



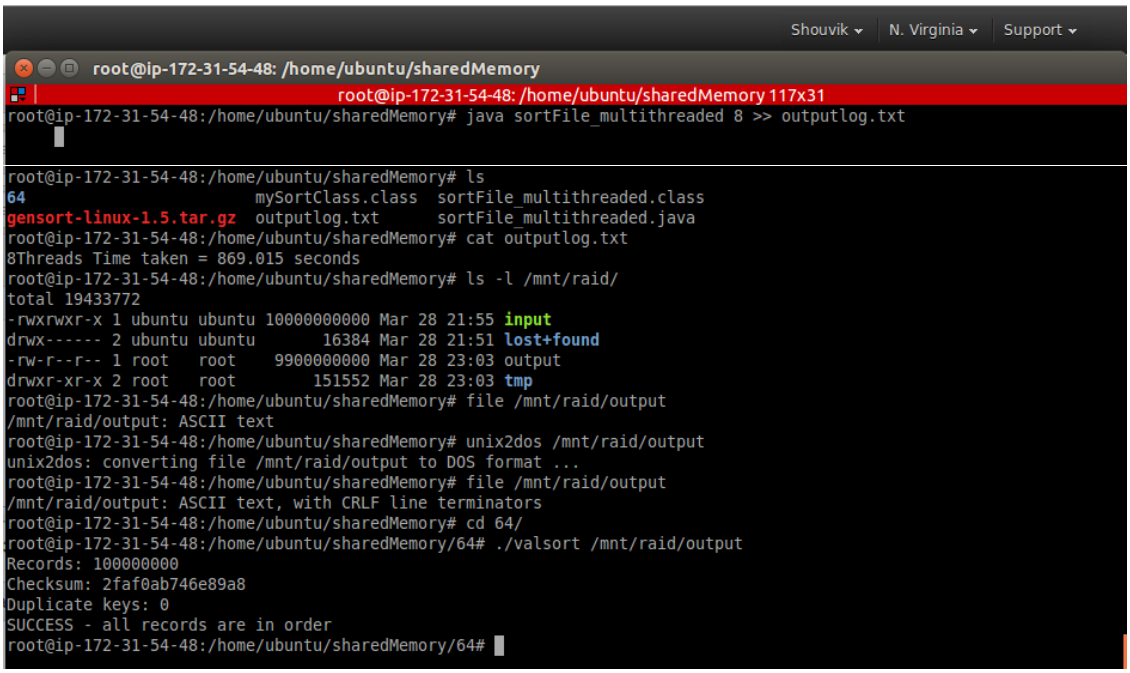
Screenshots for Shared-Memory Sort for 2 Thread:



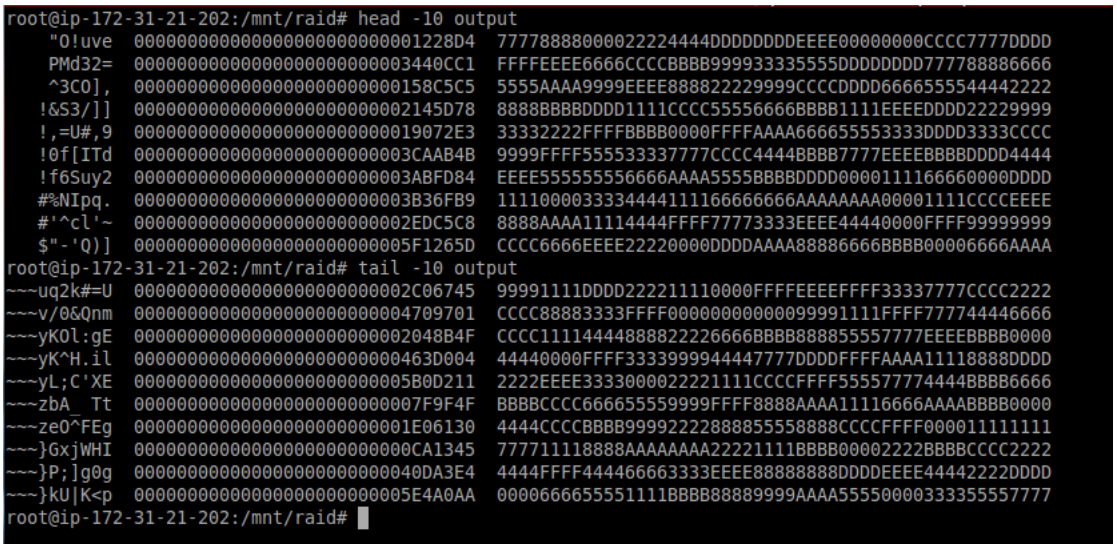
Screenshots for Shared-Memory Sort for 4 Thread:



Screenshots for Shared-Memory Sort for 8 Thread:



Screenshots for top 10 and bottom 10 lines of sorted output file:



```
root@ip-172-31-21-202:/mnt/raid# head -10 output
    "O!uve  00000000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
    PMd32=  00000000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DDDDDDDD7777888866666
    ^3CO],  00000000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
   !&S3/]]  00000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
   !,=U#,9  00000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
   !0f[ITd  00000000000000000000000003CAAB4B  9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
   !f6Suy2  00000000000000000000000003ABFD84  EEEE555555556666AAAA5555BBBBDDDD00001111666660000DDDD
   #%NIpq.  00000000000000000000000003B36FB9  11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
   #'^cl'~  00000000000000000000000002EDC5C8  8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
   $"-'Q)]  00000000000000000000000005F1265D  CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
root@ip-172-31-21-202:/mnt/raid# tail -10 output
~~~uq2k#=U  00000000000000000000000002C06745  99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm  00000000000000000000000004709701  CCCC88883333FFFF0000000000099991111FFFF777744446666
~~~yKOl:gE  00000000000000000000000002048B4F  CCCC1111444488822226666BBBB888855557777EEEEBBBB0000
~~~yK^H.il  00000000000000000000000000463D004  44440000FFFF3333999944447777DDDDFFFFAAAA11118888DDDD
~~~yL;C'XE  00000000000000000000000005B0D211  2222EEEE3333000022221111CCCCFFFF5555777744444BBBB6666
~~~zbA_ Tt  00000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg  00000000000000000000000001E06130  4444CCCCBBBB999922228888555588888CCCCFFFF000011111111
~~~}GxjWHI  00000000000000000000000000CA1345  777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g  00000000000000000000000040DA3E4  4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p  00000000000000000000000005E4A0AA  0000666655551111BBBB88889999AAAA555500003333555777 7
root@ip-172-31-21-202:/mnt/raid#
```

4 INSTALL AND CONFIGURE HADOOP

Steps to install Hadoop:

1. Start EC2 C3.Large instance and log into it.
2. Update all the installed applications using

   >> sudo apt-get update

3. Install Java in the instance using

   >>sudo apt-get install openjdk-7-jdk

4. Download and Extract hadoop packages by using below command

   >>wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.2/hadoop-2.7.2.tar.gz

5. Extract hadoop packages from the tar and rename the folder using
   >>tar -xvzf hadoop-2.7.2.tar.gz
   >>mv hadoop-2.7.2 hadoop

6. Copy .pem file from your local filesystem into the instance using

   >> scp -i "hadoop.pem" hadoop.pem ubuntu@ec2-52-87-176-251.compute-1.amazonaws.com:~/

7. Make pem file available to ssh using

   >> eval `ssh-agent -s`
   >>ssh-add hadoop.pem

8. Update .bashrc file to include hadoop variables as below:
   >>vi .bashrc
   (Add below variables at the end of .bashrc file)
   # Set Hadoop-related environment variables
   export HADOOP_HOME=$HOME/hadoop
   export HADOOP_CONF_DIR=$HOME/hadoop/etc/hadoop
   export HADOOP_MAPRED_HOME=$HOME/hadoop
   export HADOOP_COMMON_HOME=$HOME/hadoop
   export HADOOP_HDFS_HOME=$HOME/hadoop
   export YARN_HOME=$HOME/hadoop
   # Set JAVA_HOME
   export JAVA_HOME=/usr/lib/jvm/ java-1.7.0-openjdk-1.7.0.99.x86_64/jre
   export PATH=$PATH:$JAVA_HOME/bin
   export HADOOP_CLASSPATH=/usr/lib/jvm/ java-1.7.0-openjdk-1.7.0.99.x86_64lib/tools.jar
   # Add Hadoop bin/ directory to PATH
   export PATH=$PATH:$HOME/hadoop/bin
   (Save .bashrc file)
   source .bashrc

9. Update hadoop-env.sh file inside /hadoop/etc/hadoop/ as below:

```
>> vi /hadoop/etc/hadoop/hadoop-env.sh
(update the Java Home path)
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.99.x86_64/jre
(Save hadoop-env.sh)
```

10. Add master nodes private ip in etc/hosts file using

```
>> sudo vi /etc/hosts
(Add private IP of master)
172.31.8.62 master
(Save the hosts file)
```

11. Mount and join unmounted disk partition and name it /mnt/raid (See Section2: Setting Virtual Cluster for instructions on how to mount disk)

12. Create a hadoop-ec2-user file in the mounted partition using below command:

```
>> mkdir /mnt/raid/temp/hadoop-ec2-user
```

13. Add configurations in core-site.xml file using below commands:

```
>>vi hadoop/etc/hadoop/core-site.xml
(Add below entries between the configuration tags)
<property>
        <name>fs.default.name</name>
        <value>hdfs://master:9000</value>
</property>
<property>
        <name>hadoop.tmp.dir</name>
        <value>/mnt/raid/temp/Hadoop-ec2-user</value>
</property>
(Save the core-site.xml file)
```

14. Create a HDFS directory in the mounted disk and then create Namenode and Datanode directories inside HDFS directory using below command:

```
>>mkdir -p /mnt/raid/hdfs
>>mkdir -p /mnt/raid/hdfs/namenode
>>mkdir -p /mnt/raid/hdfs/datanode
```

15. Add configurations in hdfs-site.xml file using below commands:

```
>>vi hadoop/etc/hadoop/hdfs-site.xml
(Add below entries between the configuration tags)
<property>
        <name>dfs.replication</name>
        <value>1</value>
</property>
<property>
```

```
        <name>dfs.permissions</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>
```
(Save the hdfs-site.xml file)

16. Create a mapred-site.xml file using below command:

```
>>cp mapred-site.xml.template mapred-site.xml
```

17. Add configurations in mapred-site.xml file using below commands:

```
>>vi hadoop/etc/hadoop/mapred-site.xml
```
(Add below entries between the configuration tags)
```
<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>
```
(Save the mapred-site.xml file)

18. Add configurations in yarn-site.xml file using below commands:

```
>>vi hadoop/etc/hadoop/yarn-site.xml
```
(Add below entries between the configuration tags)
```
<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>
```

(Save the yarn-site.xml file)

19. Add localhost in hadoop/etc/hadoop/slaves file as below:

```
>> vi hadoop/etc/hadoop/slaves
```

Add localhost within the slave file and save it (This would be changed when we would create master node out of this AMI)

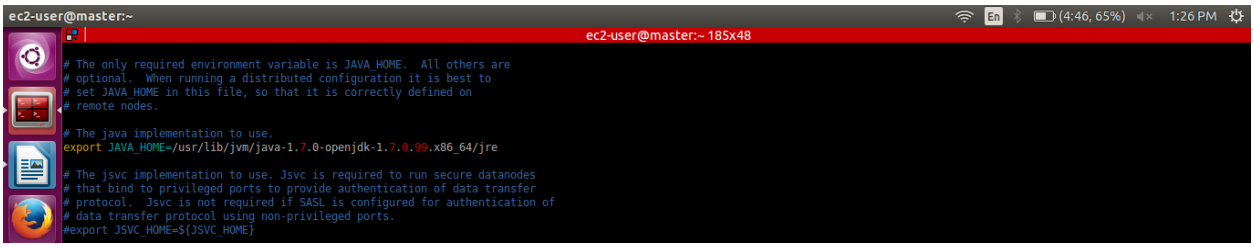
20. For password less login generate Keygen using below commands:

```
>> 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'
```

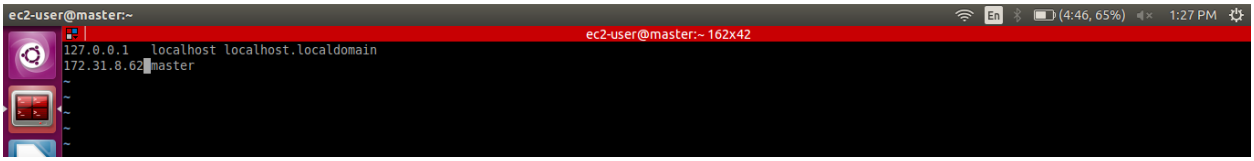
21. Create an AMI of the instance so that it can be used to create either a 1 node cluster or 16 node cluster.

Screenshots to configure Hadoop:

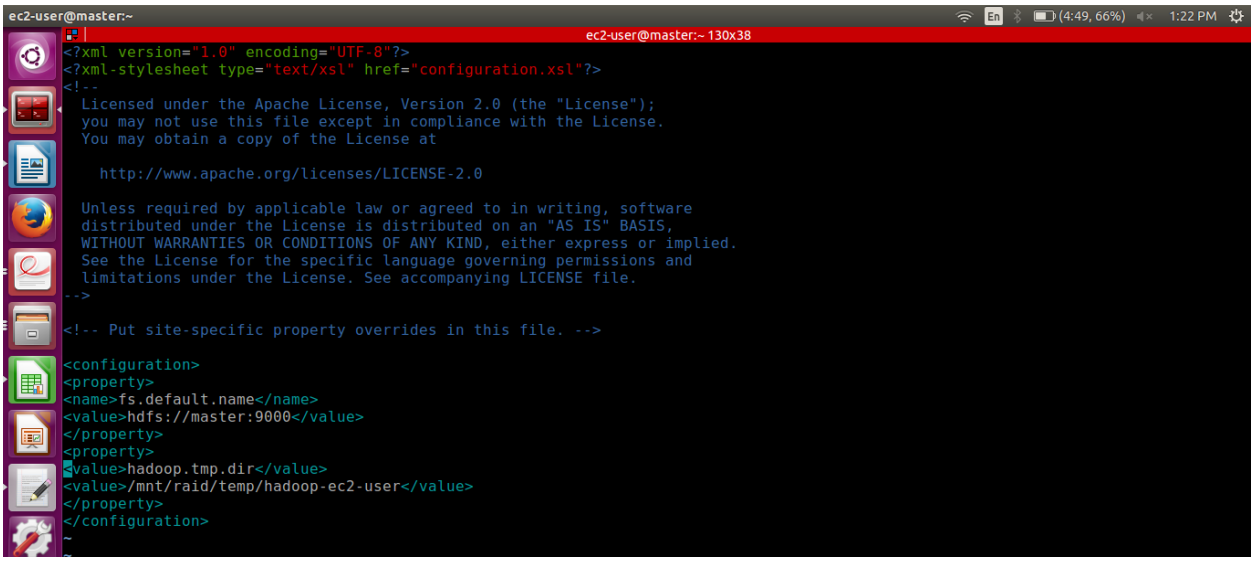
Screen-shot showing variables to be added in .bashrc file for Hadoop



Screen-shot showing Java Home variable set in hadoop-env.sh file for Hadoop

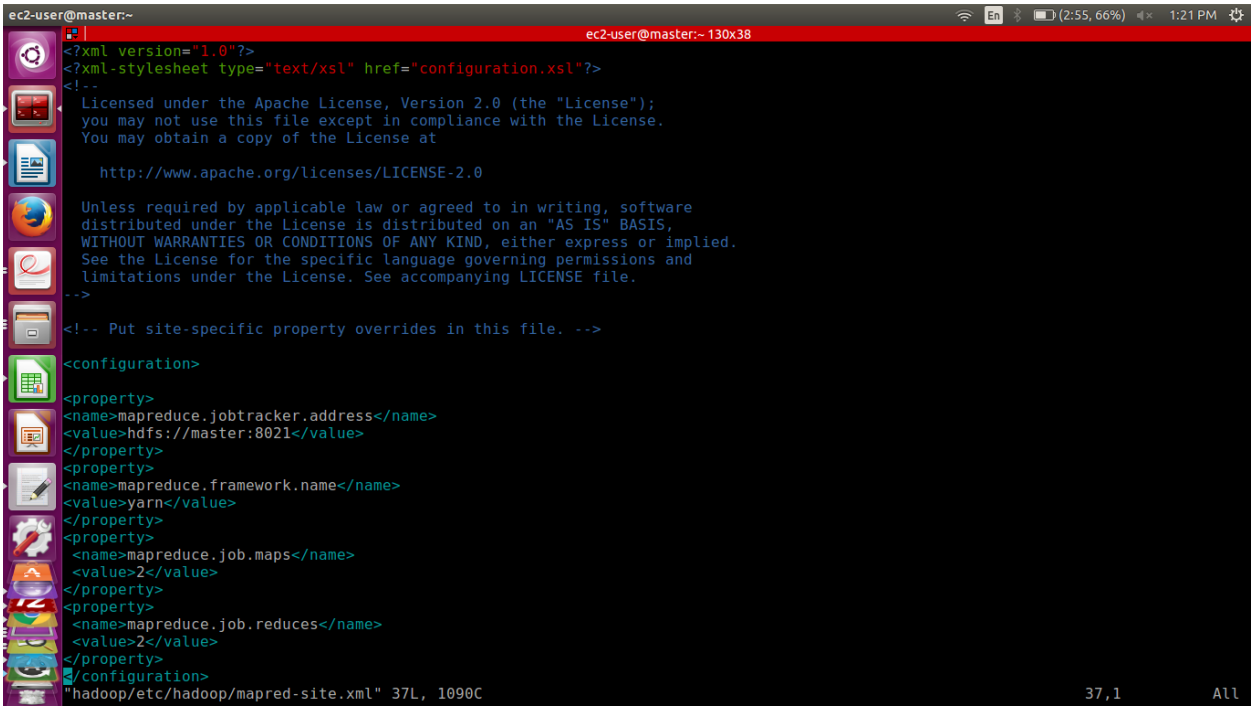


Screen-shot showing Master alias set in etc/hosts file for Hadoop

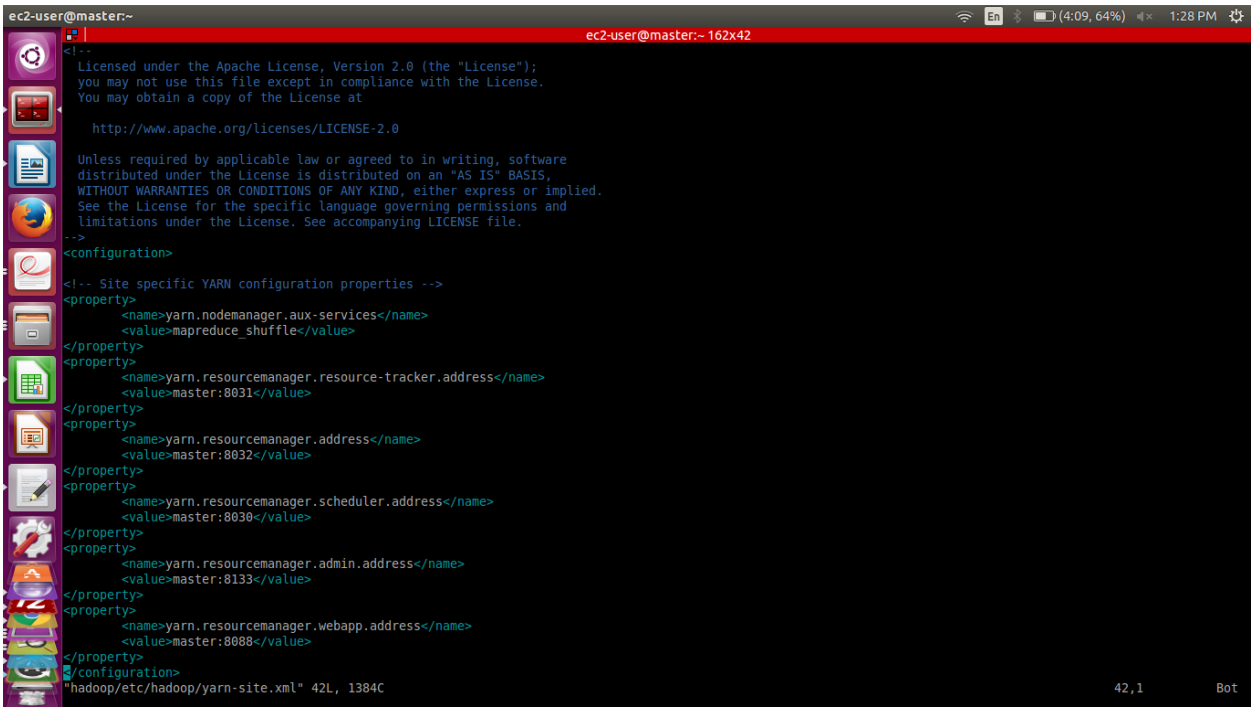


Screen-shot showing changes to be made in core-site.xml file for Hadoop

Screen-shot showing changes to be made in hdfs-site.xml file for Hadoop



Screen-shot showing changes to be made in mapred-site.xml file for Hadoop



Screen-shot showing changes to be made in yarn-site.xml file for Hadoop

5 IMPLEMENTING 10 AND 100 GB SORT IN HADOOP

Steps to start 1 node and perform 10GB Sort in Hadoop:

1. Start an instance using the Hadoop AMI which we created in above "Configuring Hadoop" section.
2. Make separate config file in ssh folder using below command

   >> cd ~/.ssh
   >>sudo vi config
   (Inside config file enter Hostname, Public DNS and identity of .pem file)
   (Save the config file)
3. Add private IPs of master and slave in etc/hosts using below command:

   >> sudo vi /etc/hosts
   (Add private IP of master & slave in master node)
   172.31.51.163 master
   172.31.20.146 slave

   (Add private IP of master in slave node)
   172.31.51.163 master
   (Save the hosts file)
4. Format the namenode from the master node using below command:

   >>hadoop namenode -format
5. Once the namenode has been formatted we can start Hadoop services using below command:

   >> cd hadoop/sbin
   >> ./start-all.sh
   >>./yarn-deamon.sh start nodemanager
6. Check whether Hadoop services have started using below command:

   >>jps
7. Now we need to run our Hadoop sort program.
8. Install gensort to generate the input file of 10 GB using:

   >>wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
   >>tar -xvzf gensort-linux-1.5.tar.gz
9. Check disk space using below command:

   >>lsblk
10. Generate input file of 10GB size using below command:

    >> cd 64/

>>./gensort -a 100000000 *[inputfileName]*

11. We now need to create an input directory within Hadoop file system using below command:

    >> hadoop fs -mkdir /user
    >>hadoop fs -mkdir /user/shouvik
    >>hadoop fs -mkdir /user/shouvik/input

12. Now we need to put input file inside hadoop file system using below command:

    >>hadoop fs -put [inputFileName] /user/shouvik/input/input10gb

13. You can confirm that input file has been placed within the Hadoop file system using below command:

    >>hadoop fs -ls /user/shouvik/input/

14. Compile the source java file using below command:

    >>hadoop com.sun.tools.javac.Main TeraSort_hadoop.java

15. After compilation is done we need to create a jar file using below command:

    >>jar -cf ts.jar TeraSort_hadoop*.class

16. To run the jar file use below command:

    >> hadoop jar ts.jar TeraSort_hadoop /user/shouvik/input /user/shouvik/output

17. Output file get generated in /user/shouvik/output directory. To check use below command:

    >>hadoop fs -ls /user/shouvik/ts/output/

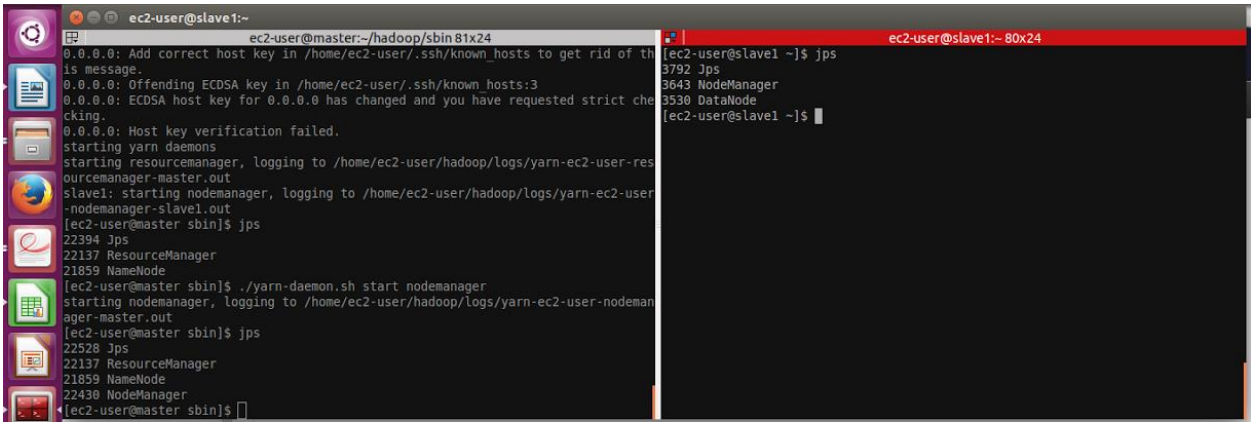
18. Copy the generated output file to your local instance disk using below command:

    >> hadoop fs -get /user/shouvik/output/part-r-00000 /mnt/raid

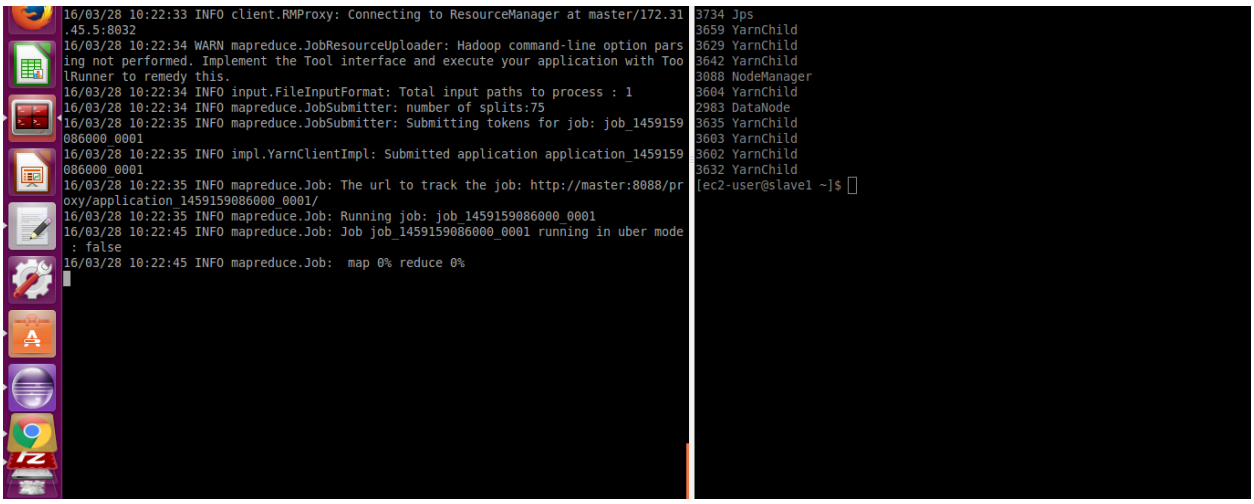
19. You can validate the output file using valsort using below command:

    >> ./valsort /mnt/raid/part-r-00000

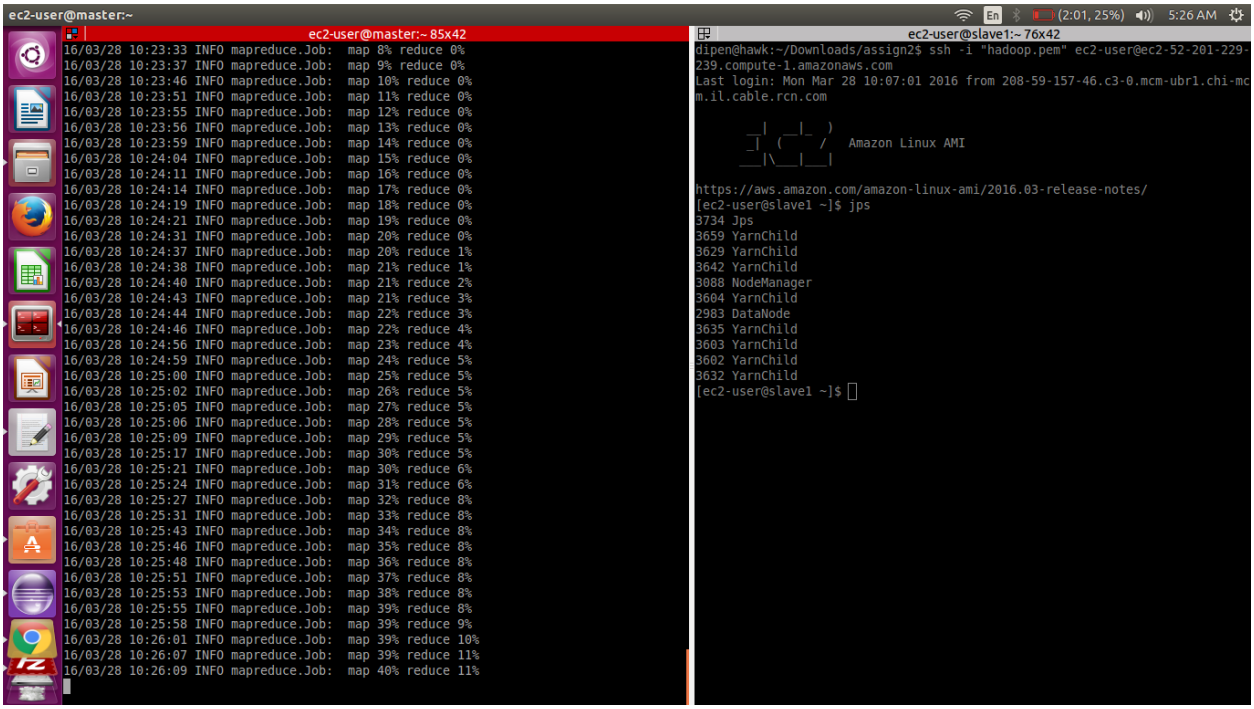
Screenshots for 1 node cluster to perform 10GB sort in Hadoop:



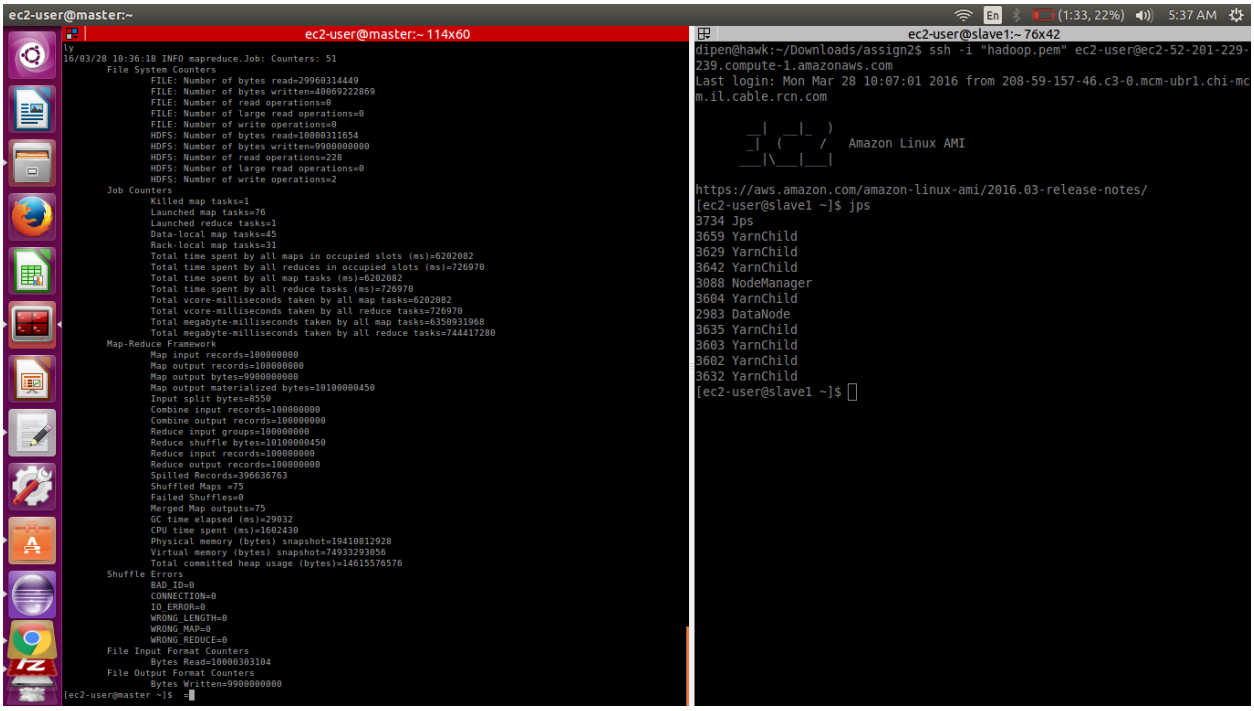
Screen-shot showing JPS command on master and slave for Hadoop



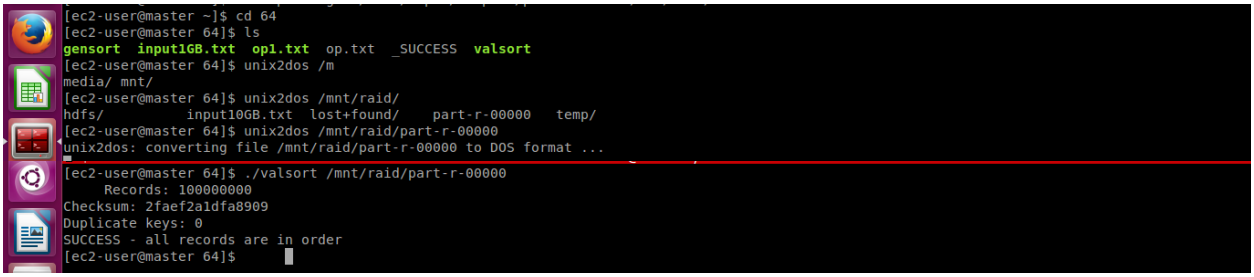
Screen-shot showing start of Hadoop program and processes in Slave node



Screen-shot showing running Hadoop program and processes in Slave node



Screen-shot showing completion of Hadoop program and processes in Slave node



Screen-shot showing valsort result for 10gb sort of Hadoop program.

Steps to start 16 nodes and perform 100GB Sort in Hadoop:

1. Start 1 master and 16 slaves instances using the Hadoop AMI which we created in above “Configuring Hadoop” section.

2. Make separate config file in ssh folder of master using below command

   >> cd ~/.ssh
   >>sudo vi config
   (Inside config file for each slave enter Hostname, Public DNS and identity of .pem file)
   (Save the config file)

3. Add private IPs of master and slave in etc/hosts using below command:

   >> sudo vi /etc/hosts
   (Add private IP of master & all slaves in master node)

Eg.172.31.51.163 master
172.31.20.146 slave1
172.31.45.20 slave2

(Add private IP of master in slave node)
172.31.51.163 master
(Save the hosts file)

4. Mount disk in master and all slaves. (For instructions on how to mount disk see "Steps to mount unmounted disk space and renaming it to /mnt/raid" topic in "Setting virtual cluster section".)

5. Update the hadoop/etc/Hadoop/slaves using below command:

   >> vi hadoop/etc/hadoop/slaves
   Add all slave aliases as defined in the etc/hosts file and save it.

6. Format the namenode from the master node using below command:

   >>hadoop namenode -format

7. Once the namenode has been formatted we can start Hadoop services using below command:

   >> cd hadoop/sbin
   >> ./start-all.sh
   >>./yarn-deamon.sh start nodemanager

8. Check whether Hadoop services have started using below command:

   >>jps

9. Now we need to run our Hadoop sort program.

10. Install gensort to generate the input file of 100 GB using:

    >>wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
    >>tar -xvzf gensort-linux-1.5.tar.gz

11. Check disk space using below command:

    >>lsblk

12. Generate input file of 100GB size using below command:

    >> cd 64/
    >>./gensort -a 1000000000 *[inputfileName]*

13. We now need to create an input directory within Hadoop file system using below command:

    >> hadoop fs -mkdir /user
    >>hadoop fs -mkdir /user/shouvik
    >>hadoop fs -mkdir /user/shouvik/input

14. Now we need to put input file inside hadoop file system using below command:

>>hadoop fs -put [inputFileName] /user/shouvik/input/input100gb

15. You can confirm that input file has been placed within the Hadoop file system using below command:

>>hadoop fs -ls /user/shouvik/input/

16. Compile the source java file using below command:

>>hadoop com.sun.tools.javac.Main TeraSort_hadoop.java

17. After compilation is done we need to create a jar file using below command:

>>jar -cf ts.jar TeraSort_hadoop*.class

18. To run the jar file use below command:

>> hadoop jar ts.jar TeraSort_hadoop /user/shouvik/input /user/shouvik/output

19. Output file get generated in /user/shouvik/output directory. To check use below command:

>>hadoop fs -ls /user/shouvik/ts/output/

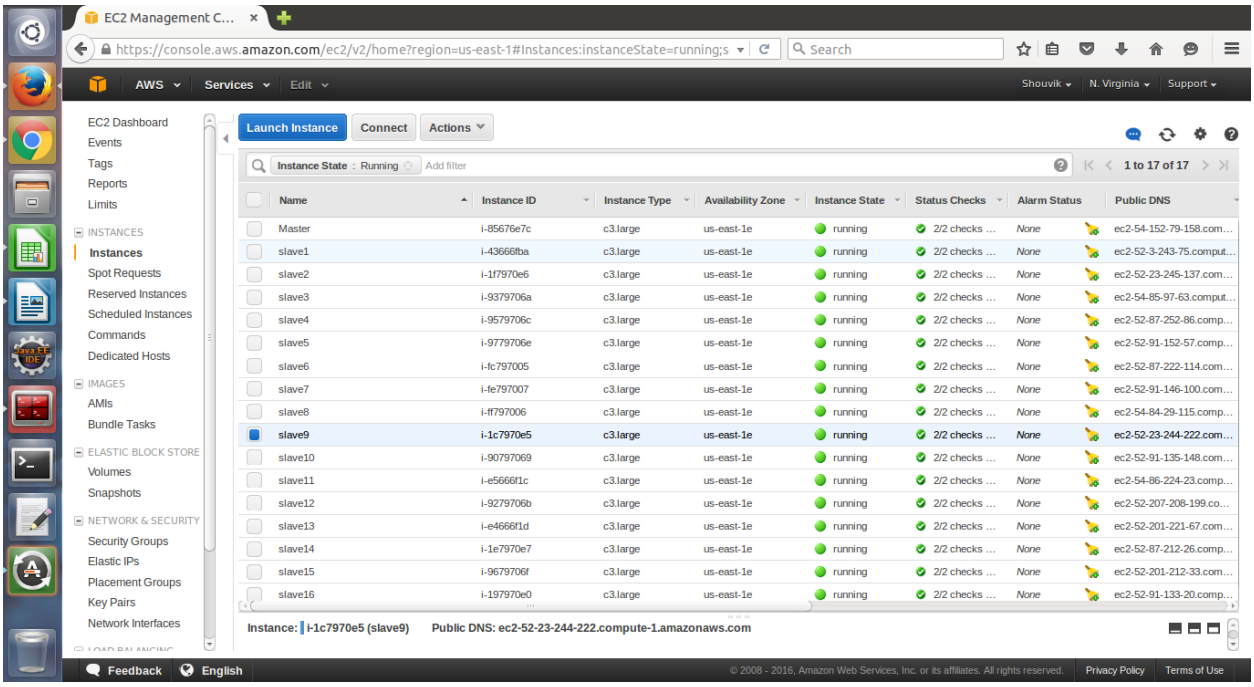
20. Copy the generated output file to your local instance disk using below command:

>>hadoop fs -get /user/shouvik/output/part-r-00000 /mnt/raid

21. You can validate the output file using valsort using below command:

>>./valsort /mnt/raid/part-r-00000

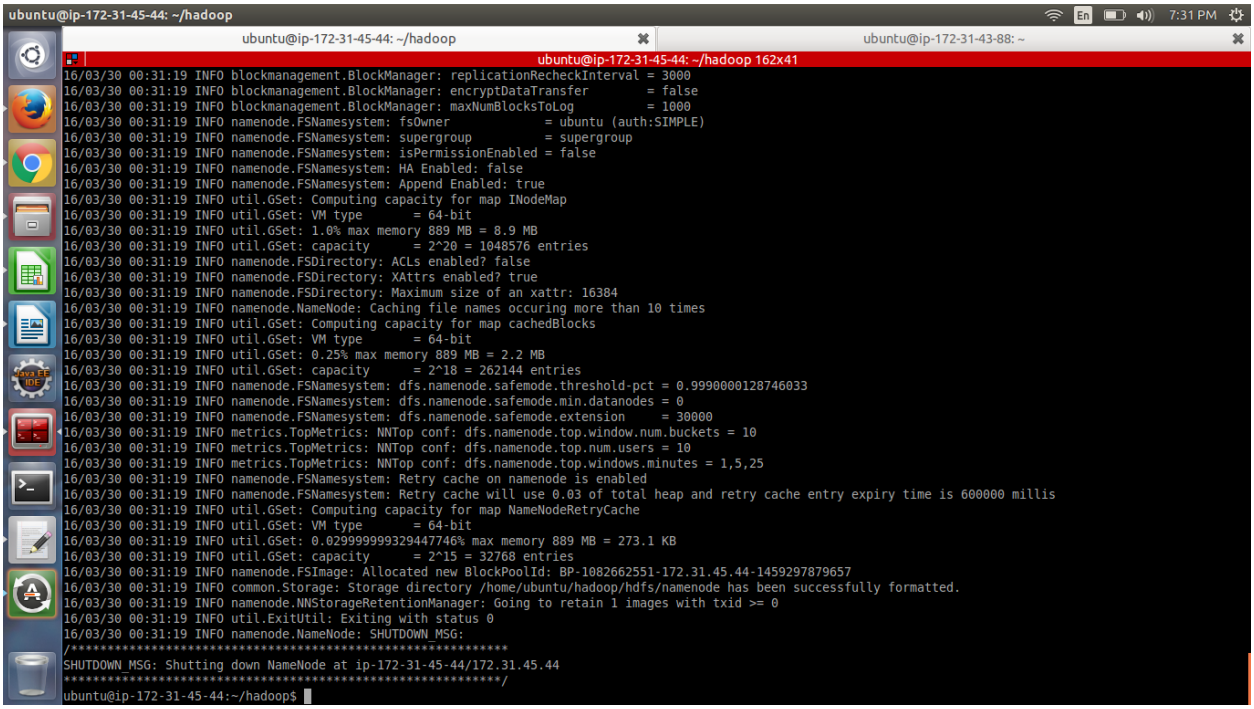
Screenshots for 16 node cluster to perform 100GB sort in Hadoop:



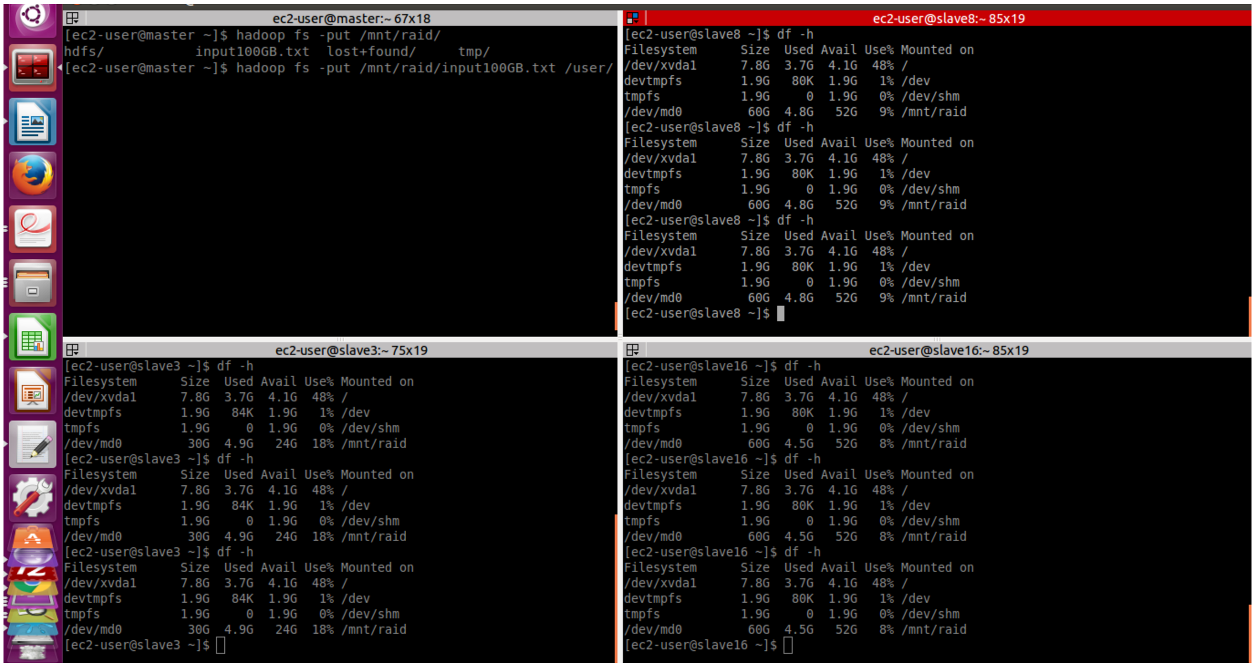
Screen-shot showing 1 master and 16 slaves running in AWS.

Screen-shot showing 1 master and 16 slaves IP added to /etc/hosts.

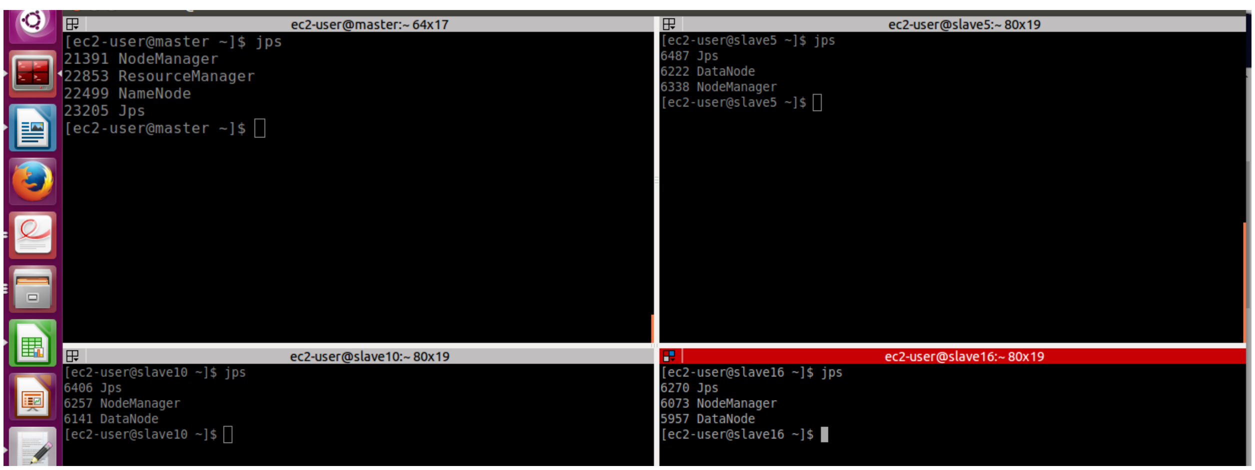
Screen-shot showing 16 slave aliases added in Hadoop/etc/Hadoop/slaves file.



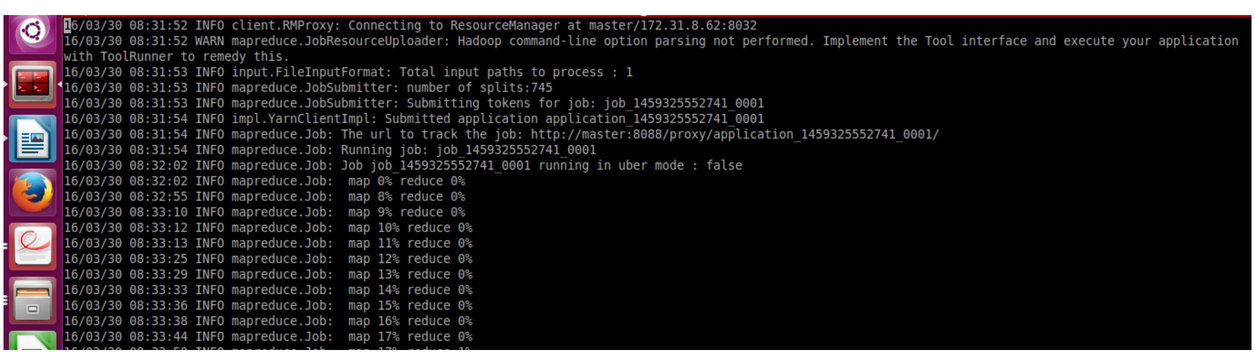
Screen-shot showing namenode being formatted in Hadoop master node.



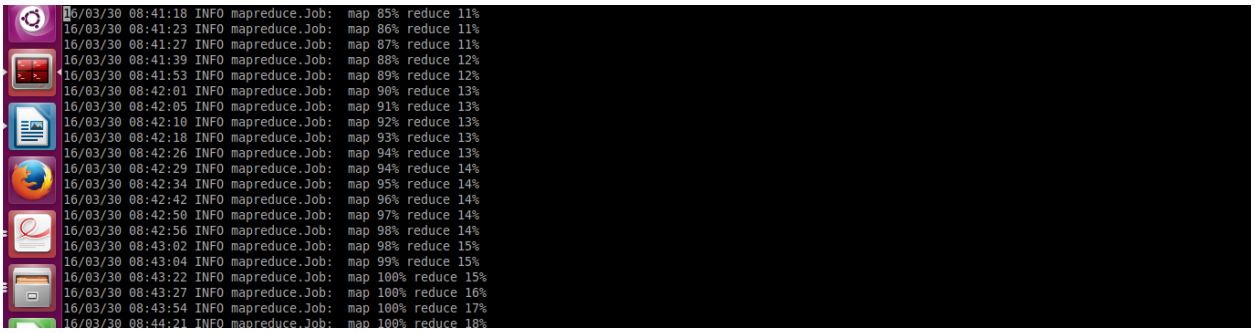
Screen-shot showing 100gb input file being copied into HDFS file system.



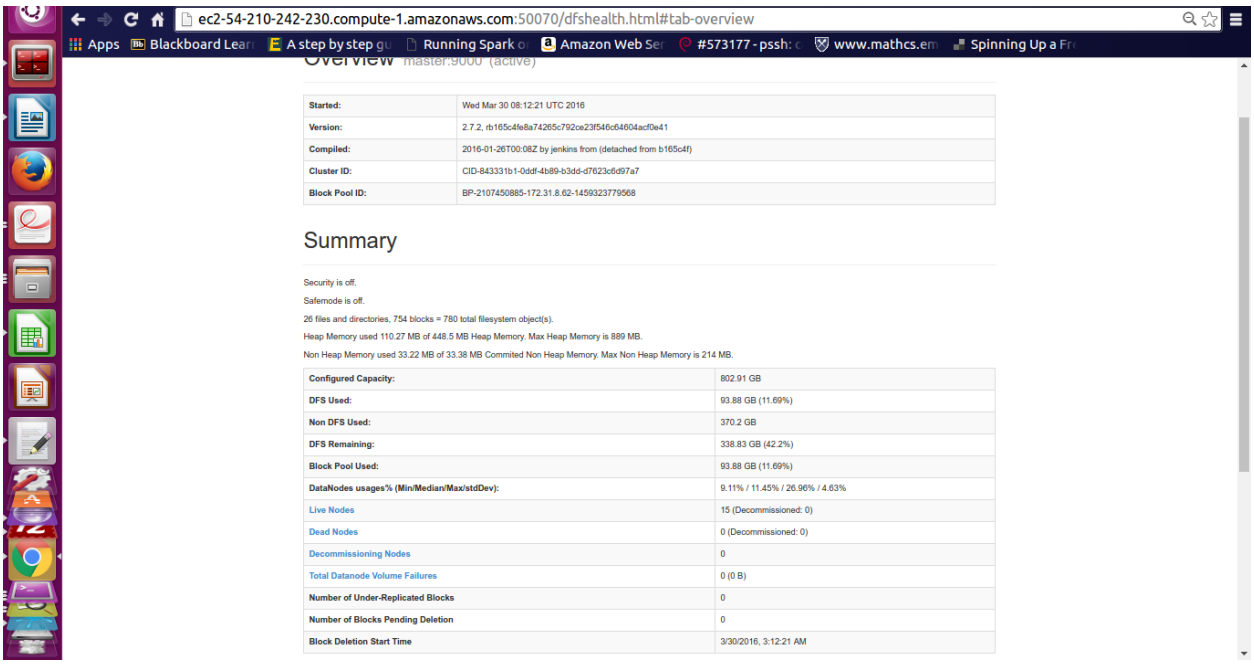
Screen-shot showing processes running in master and slaves after staring Hadoop using JPS command.



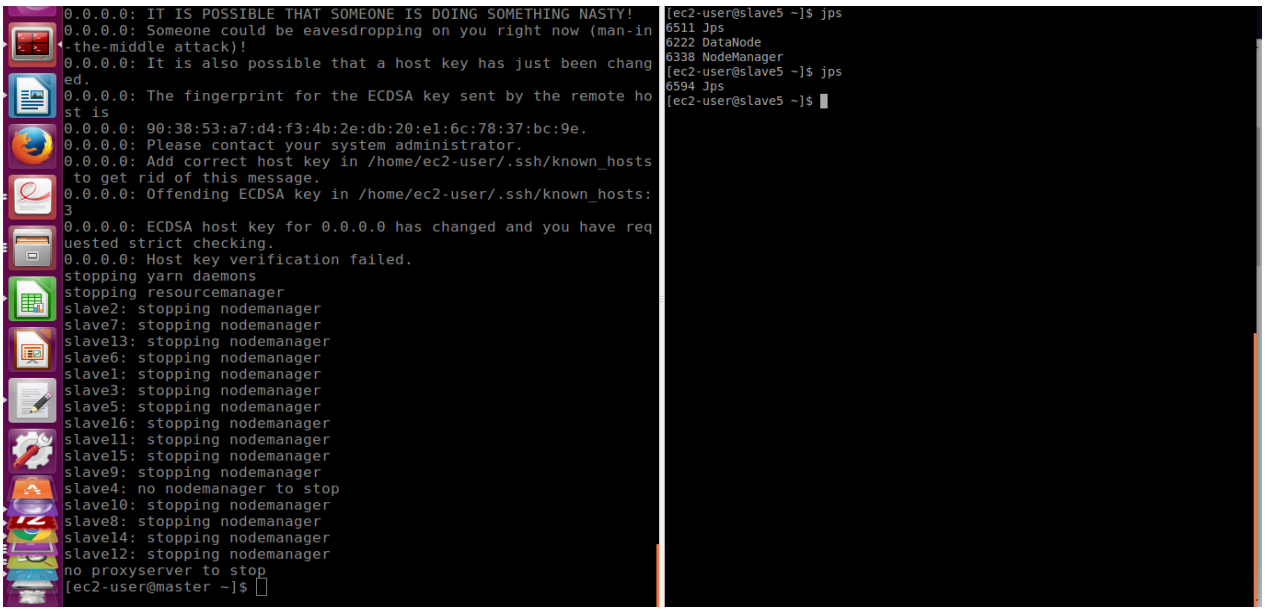
Screen-shot showing start of Hadoop program



Screen-shot showing running hadoop program and processes in Slave node



Screen-shot showing summary page for Hadoop in browser.



Screen-shot showing Hadoop processes stopping in all slave instance.

Questions and Answers:

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

   Ans: Master node manages the whole process like opening, closing input file, partitioning the input file, assigning the partitioned files to workers, and so on.

   Slave node are the workers which are assigned to execute branch to finish assigned input part on which processing needs to be done. Basically slaves work on the actual data whereas master's work is to schedule and synchronize this work.

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 uses different port to uniquely identify itself. If two services are given the same port either of the service would work at a single time. Both services won't be able to work simultaneously. Thus there would be port collision between different function modules if we use same port number.

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

   Ans: We can change the number of mappers and reducers by adding two properties in "mapred-site.xml" file as below:

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

6 INSTALL AND CONFIGURE SPARK

Steps to get an Identity and Access Management Key:

1. Click on your Account Name on top right corner and then select “Security Credentials”.
2. Expand the Access Keys tab.
3. Click on “Create New Access Key”
4. A popup will show up. Click on the “Download key file” and store the key(.csv file) in a secure location.

Steps to install Spark:

1. Start EC2 C3.Large instance and log into it.
2. Update all the installed applications using

   >> sudo apt-get update

3. Install Java in the instance using

   >>sudo apt-get install openjdk-7-jdk

4. Download Spark packages by using below command

   >>wget http://apache.arvixe.com/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz

5. Extract Spark packages from the tar and rename the folder using

   >>tar -xvzf spark-1.6.1-bin-hadoop2.6.tgz
   >>mv spark-1.6.1-bin-hadoop2.6 spark

6. Download Scala packages by using below command

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

7. Create a folder in usr/local for Scala using below command

   >> sudo mkdir /usr/local/src/scala

8. Update .bashrc file with Scala variables

   >> vi .bashrc
   (Inside .bashrc file enter below 2 entries)
   export SCALA_HOME=/usr/local/src/scala/scala-2.10.4
   export PATH=$SCALA_HOME/bin:$PATH
   (Save the .bashrc file)
   >> source .bashrc

9. Copy .pem file from your local filesystem into the instance using

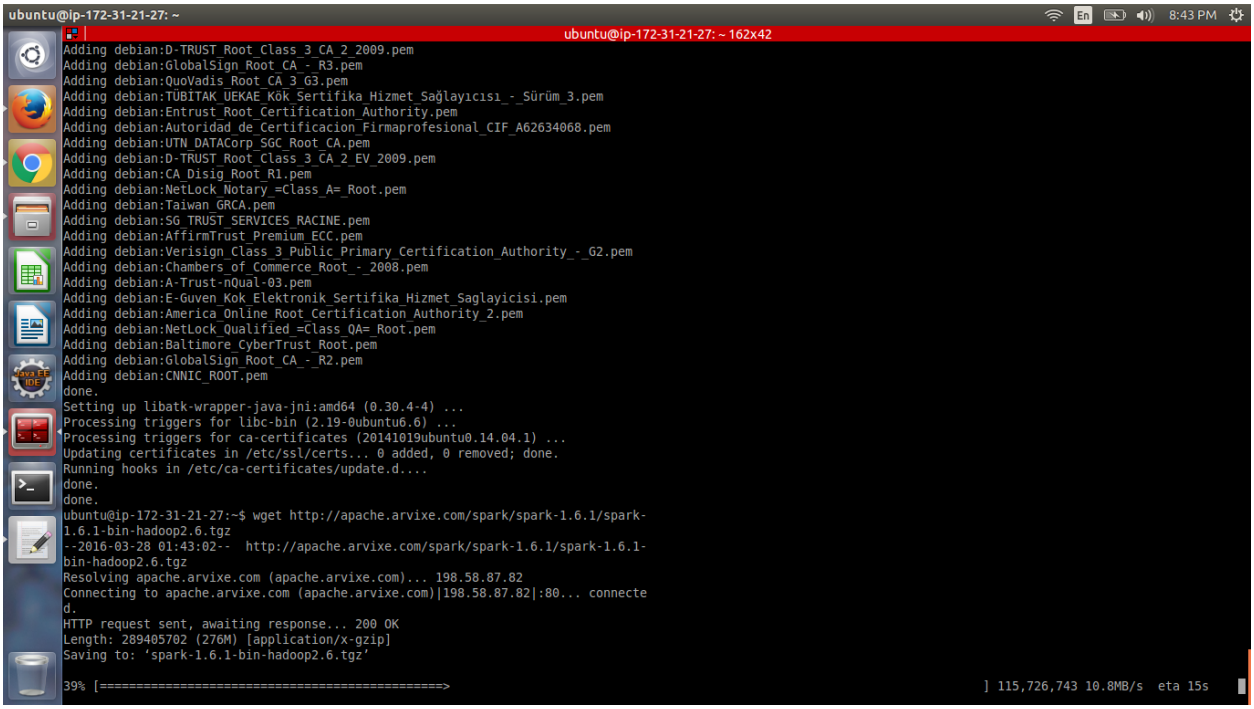
   >> scp -i “spark.pem” spark.pem ubuntu@ec2-54-164-89-172.compute-1.amazonaws.com:~/

10. Export your IAM AccessKeyID and SecurityAccess Key as below

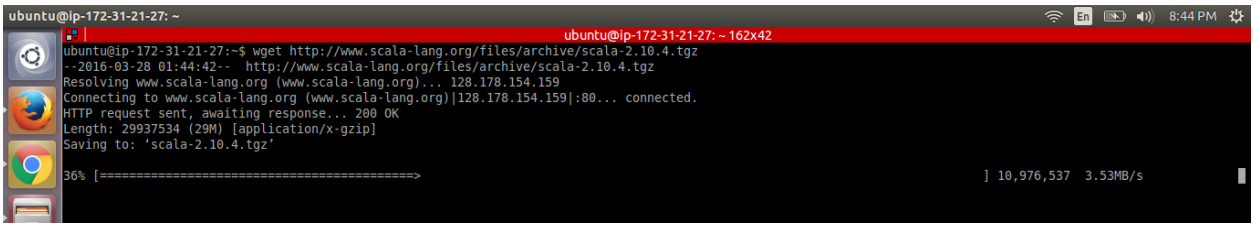
    >>export AWS_ACCESS_KEY_ID=[Your Access Key]
    >>export AWS_SECRET_ACCESS_KEY=[Your Secret Access Key]

11. Create an AMI of the instance so that it can be used to create either a 1 node cluster or 16 node cluster

Screenshots to configure Spark:



Screenshot showing Spark Installation



Screenshot showing Scala Installation

7 IMPLEMENTING 10 AND 100 GB SORT IN SPARK

Steps to start 1 node and perform 10GB Sort in Spark:

1. Start an instance using the spark AMI which we creating in above section.
2. To start a cluster of 1 node write below command (Below command will create a 1 master and 1 slave cluster):

   >> cd spark/ec2/
   >>./spark-ec2 -k spark -i spark.pem -s 1 -t c3.large --spot-price=0.025 --ebs-vol-size=30 --ebs-vol-num=1 launch spark_tsort

3. Login into the cluster using the below command (By default we login into Master node):

   >>./spark-ec2 -k spark -i spark.pem login spark_tsort

4. Below all commands are now written in the master node of Spark
5. Install gensort to generate the input file of 10 GB using:

   >>wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
   >>tar -xvzf gensort-linux-1.5.tar.gz

6. Check disk space using below command:

   >>lsblk

7. Generate input file of 10GB size using below command:

   >> cd 64/
   >>./gensort -a 100000000 *[inputfileName]*

8. Modify hdfs-site.xml to using below command:

   >> cd ephimeral-hdfs/conf/
   >> vi hdfs-site.xml
   (Inside hdfs-site.xml modify the dfs.replication to make its value to 1 as shown below)

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

   (Save the hdfs-site.xml file)

9. Now we need to copy this configuration into the slave node. This is done by below command:

   >> cd spark/ec2/
   >>./spark-ec2/copy-dir ephemeral-hdfs/conf/

10. Now we need to stop SPARK and HDFS service and start again as configuration file was modified using

    >>./stop-all.sh (inside spark/sbin/ directory)

>>./stop-dfs.sh (inside ephemeral-hdfs/bin/ directory)
>>./hadoop namenode -format
>> ./start-all.sh (inside spark/sbin/ directory)
>>./start-dfs.sh (inside ephemeral-hdfs/bin/ directory)

11. Now we need to put input file inside HDFS file system using below command:

    >>ephemeral-hdfs/bin/hadoop fs -put [inputFileName] /user/shouvik/input/input10gb

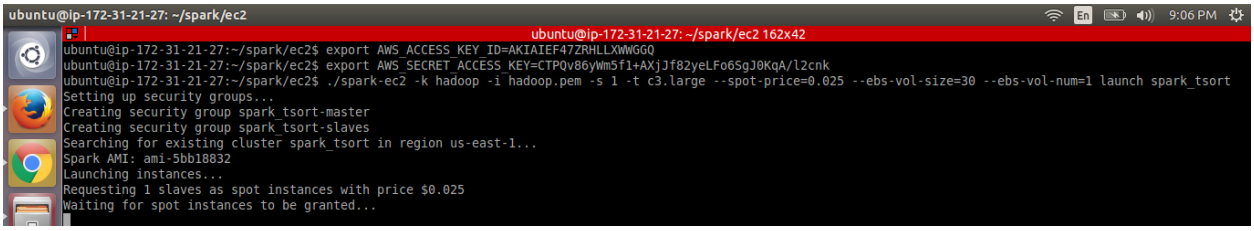
12. Now Run Program of Sort written in Scala into spark shell.

    >>cd spark/bin
    >>./spark-shell

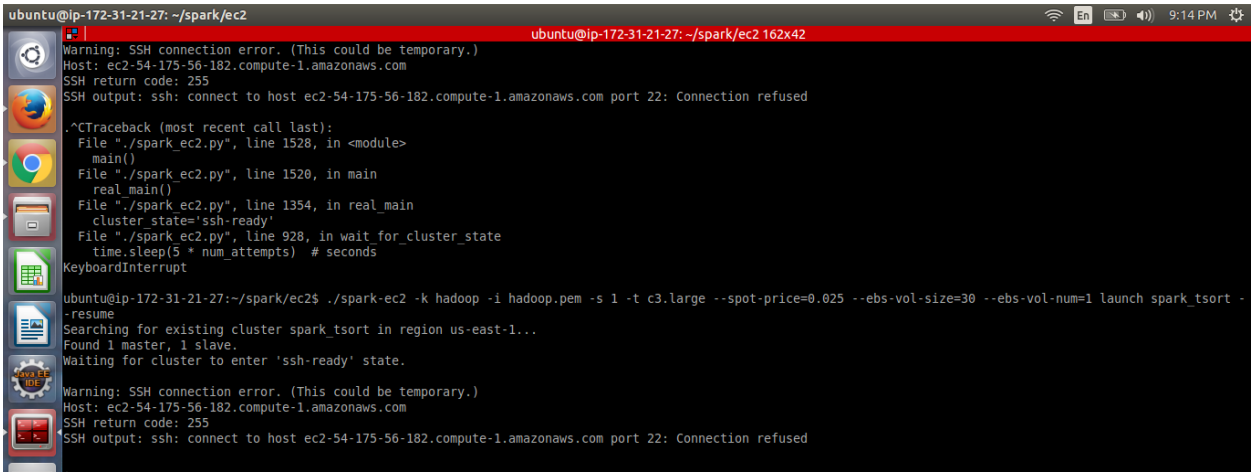
13. Copy and paste following command into shell. (Note: Run one by one)

    >>val sortFile = sc.textFile("hdfs://ec2-52-207-231-47.compute-1.amazonaws.com:9000/user/shouvik/input/input10gb")
    >>val sortedobj = sortFile.flatMap(line=>line.split("\n")).map(word=>(word.substring(0,10),word.substring(10))).sortByKey()
    >>sortedobj.saveAsTextFile("hdfs://ec2-52-207-231-47.compute-1.amazonaws.com:9000/user/shouvik/output10gb")

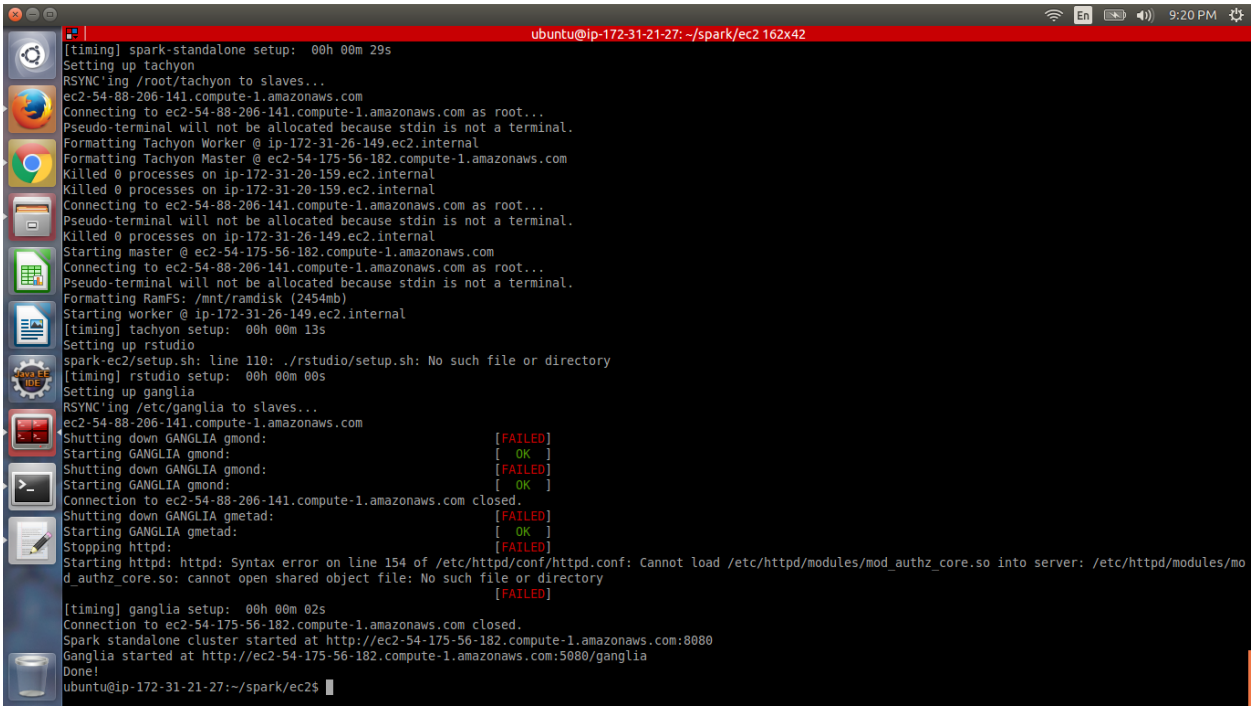
Screenshots for 1 node cluster to perform 10GB sort in Spark:



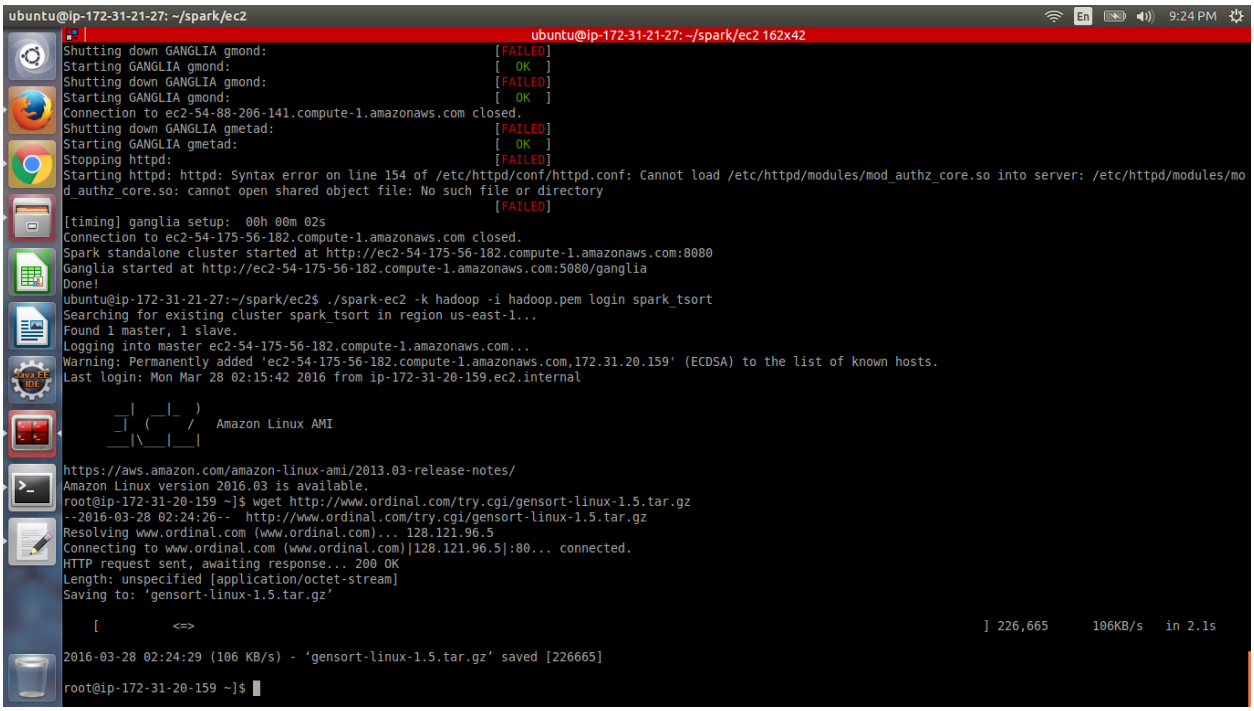
Screenshot showing start of 1 node cluster in Spark



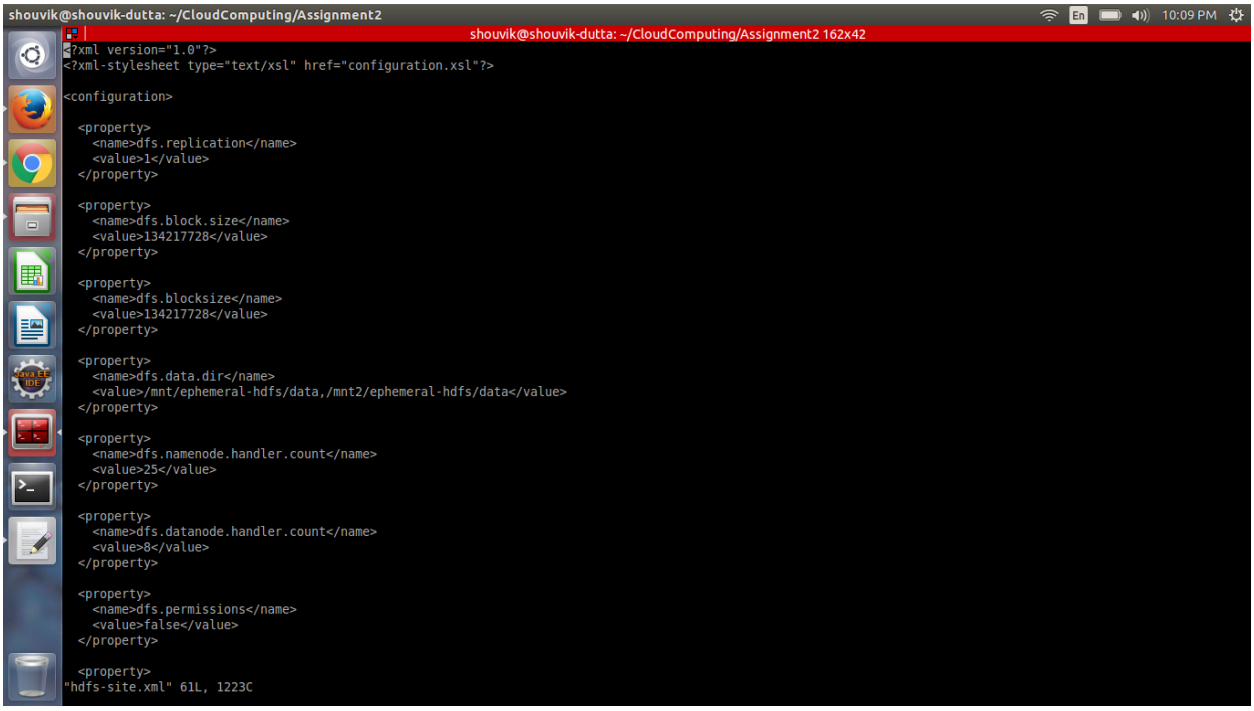
Screenshot showing resume of 1 node cluster in Spark



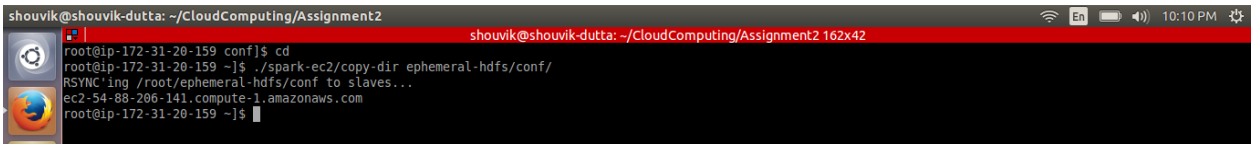
Screenshot showing successful completion of creation of 1 node cluster in Spark



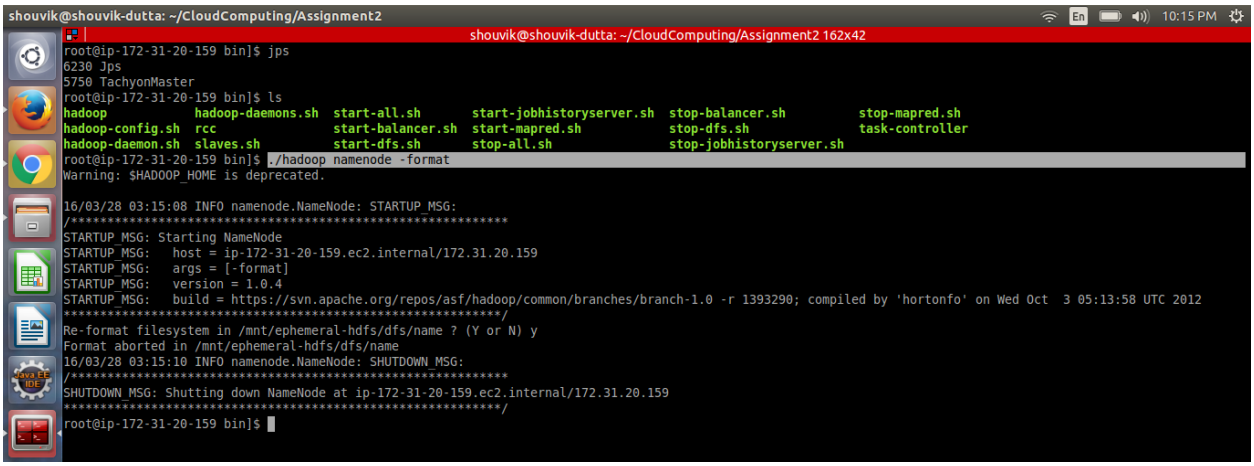
Screenshot showing login to master node in Spark



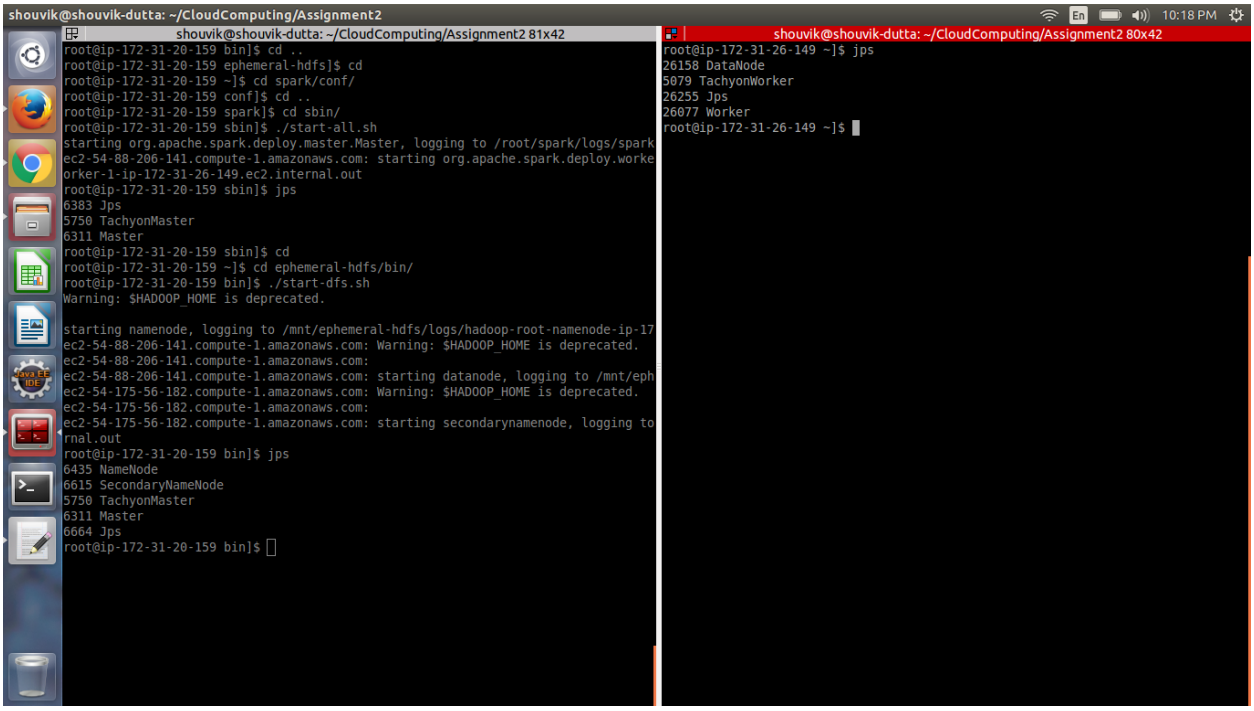
Screenshot showing modification to be made in hdfs-site.xml file in Spark



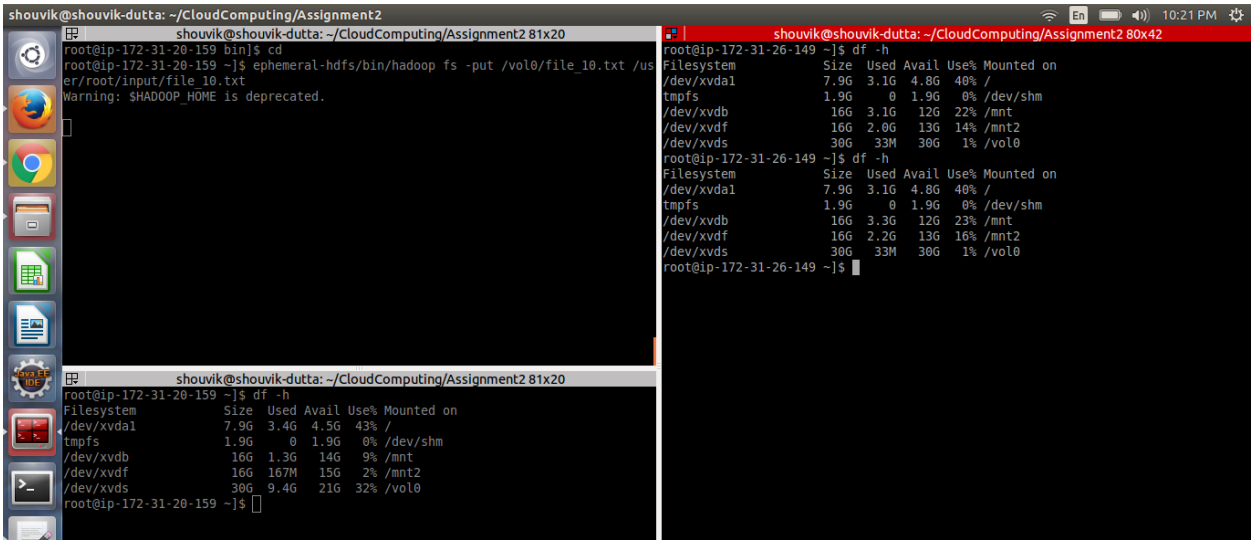
Screenshot showing config file being copied to all slaves in Spark



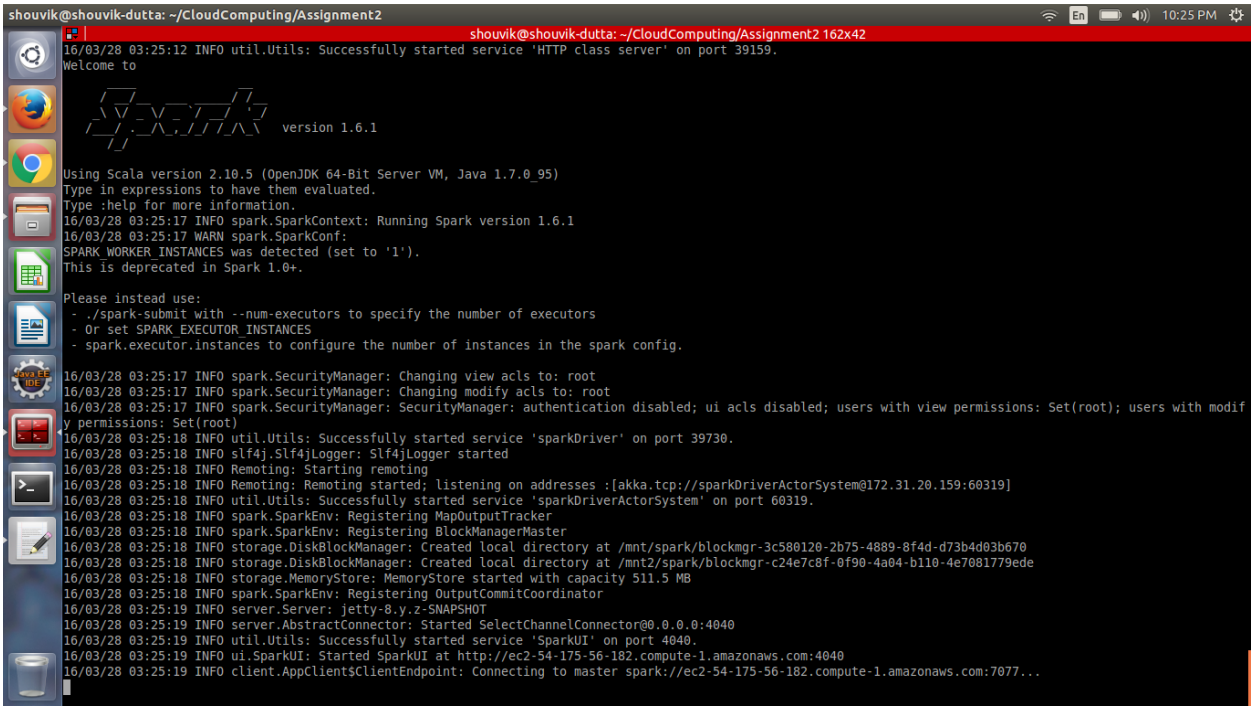
Screenshot showing namenode formatted in Spark



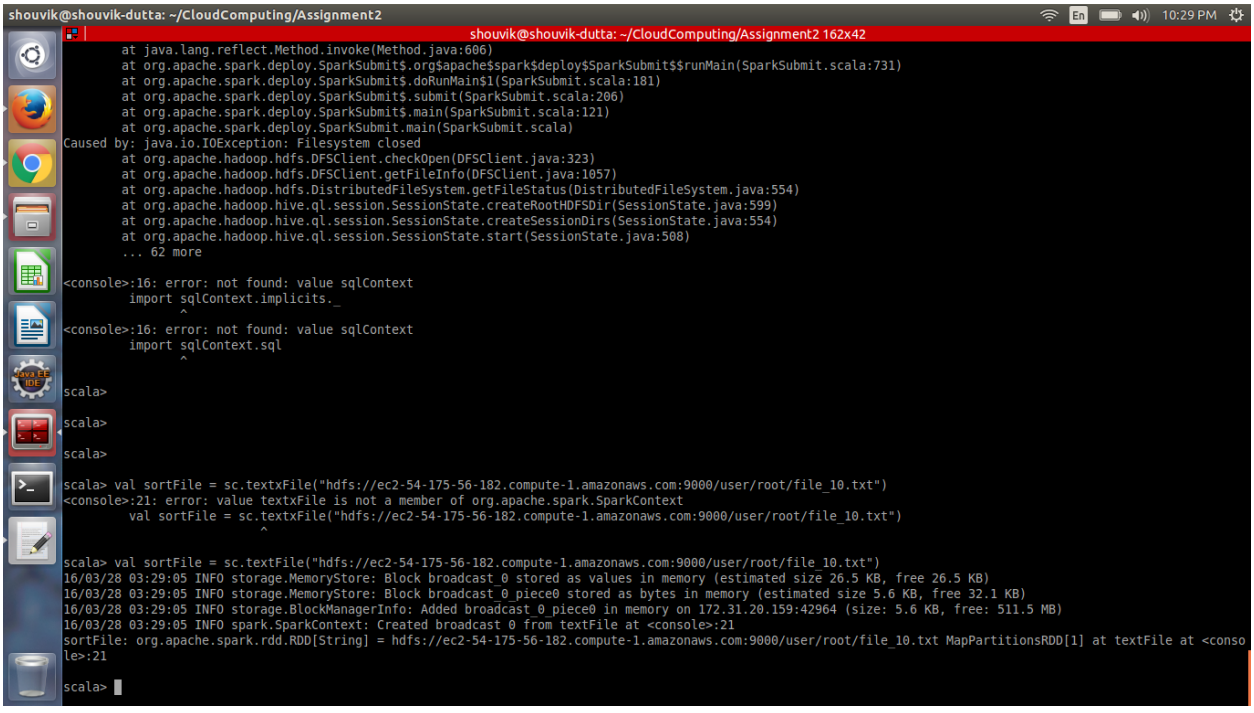
Screenshot showing master and slave running in Spark



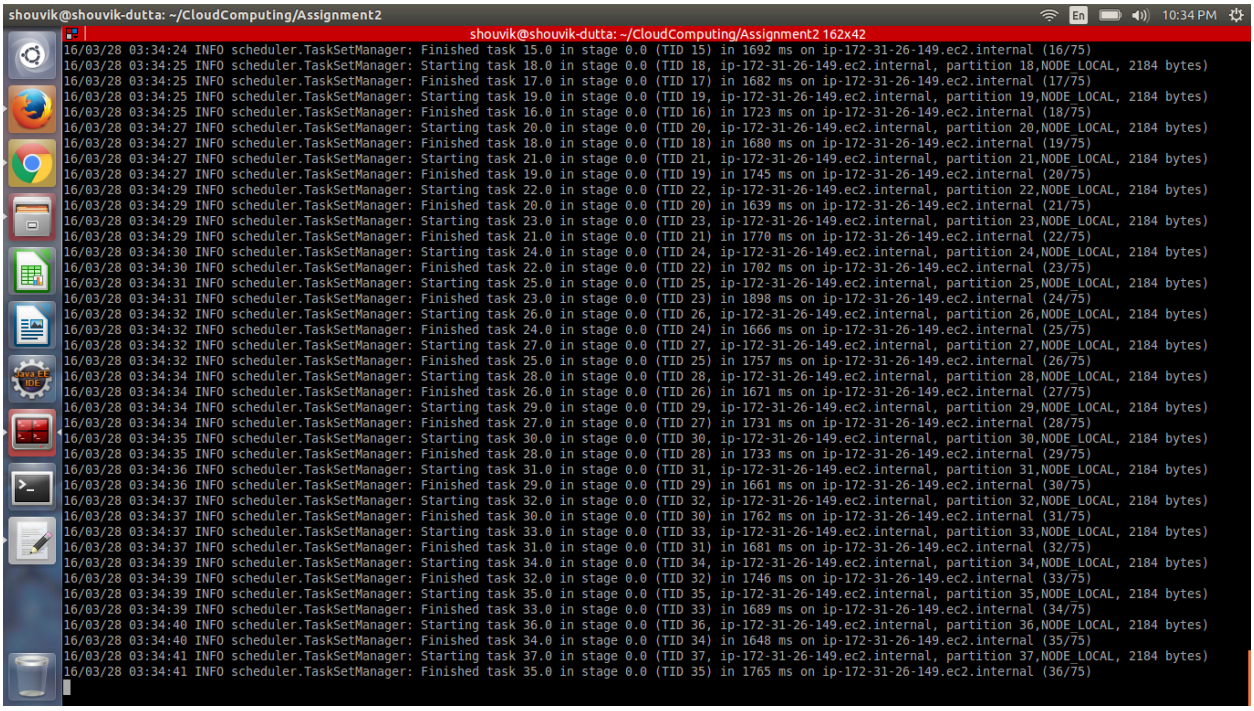
Screenshot showing input data being put in HDFS in Spark



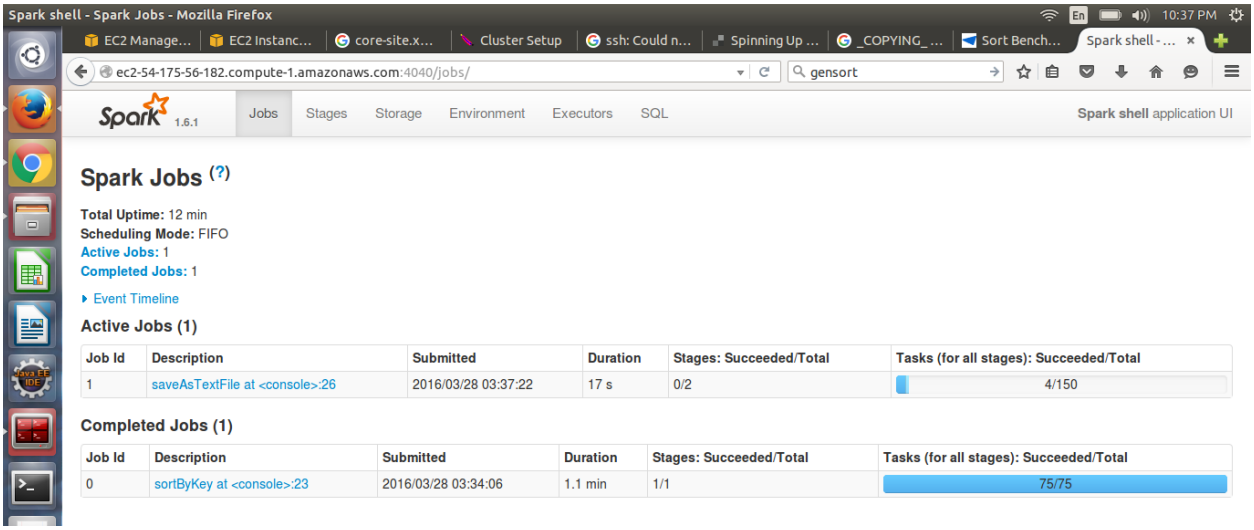
Screenshot showing shell running in Spark



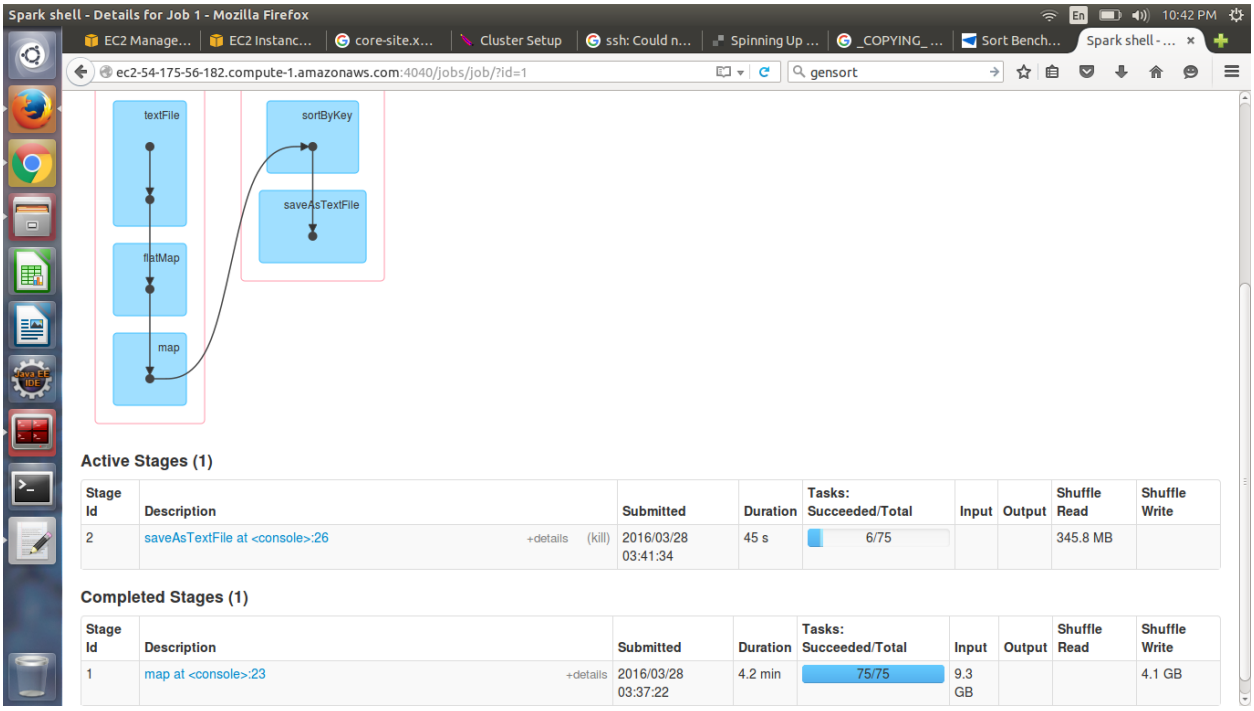
Screenshot showing scala sort program written in Spark Shell



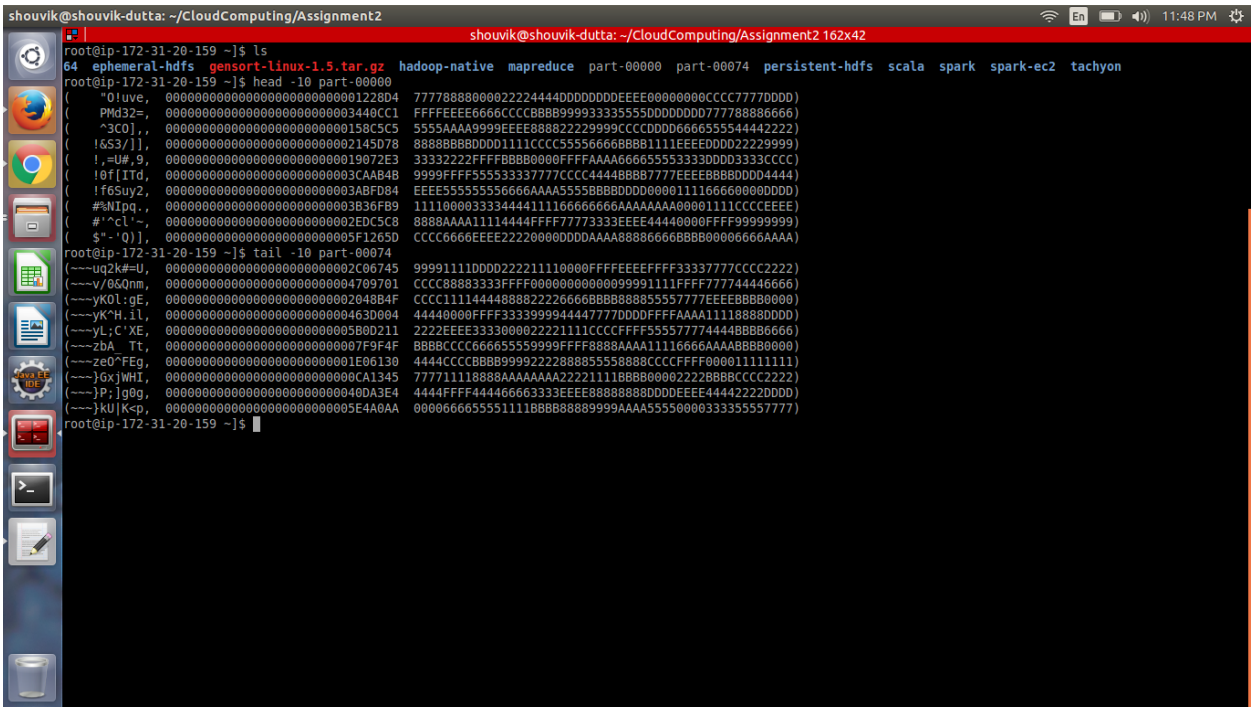
Screenshot showing scala sort program running in Spark Shell



Screenshot showing spark jobs in browser



Screenshot showing visualization for scala sort program in browser



Screenshots for top 10 and bottom 10 lines of sorted output file

Steps to start 16 nodes and perform 100GB Sort in Spark:

1. Start an instance using the spark AMI which we creating in above section.
2. To start a cluster of 1 node write below command (Below command creates a C3.4xlarge master instance and 16 C3.large slave instances)

   >> cd spark/ec2/
   >>./spark-ec2 -k spark -i spark.pem -s 16 --instance-type=c3.large -m c3.4xlarge --spot-price=0.14 launch spark_tsort_100gb

3. Login into the cluster using the below command (By default we login into Master node):

   >>./spark-ec2 -k spark -i spark.pem login spark_tsort_100gb

4. Below all commands are now written in the master node of Spark
5. Install gensort to generate the input file of 100 GB using:

   >>wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
   >>tar -xvzf gensort-linux-1.5.tar.gz

6. Check disk space using below command:

   >>lsblk

7. Generate input file of 100GB size using below command:

   >> cd 64/
   >>./gensort -a 1000000000 *[inputfileName]*

8. Modify hdfs-site.xml to using below command:

   >> cd ephimeral-hdfs/conf/
   >> vi hdfs-site.xml
   (Inside hdfs-site.xml modify the dfs.replication to make its value to 1 as shown below)
   <property>
   <name>dfs.replication</name>
   <value>1</value>
   </property>
   (Save the hdfs-site.xml file)

9. Now we need to copy this configuration into all 16 slave nodes. This is done by below command:

   >> cd spark/ec2/
   >>./spark-ec2/copy-dir ephemeral-hdfs/conf/

10. Now we need to stop SPARK and HDFS service and start again as configuration file was modified using

    >>./stop-all.sh (inside spark/sbin/ directory)
    >>./stop-dfs.sh (inside ephemeral-hdfs/bin/ directory)
    >>./hadoop namenode -format

>> ./start-all.sh (inside spark/sbin/ directory)
>>./start-dfs.sh (inside ephemeral-hdfs/bin/ directory)

11. Now we need to put input file inside HDFS file system using below command:

    >>ephemeral-hdfs/bin/hadoop fs -put [inputFileName] /user/shouvik/input/input100gb

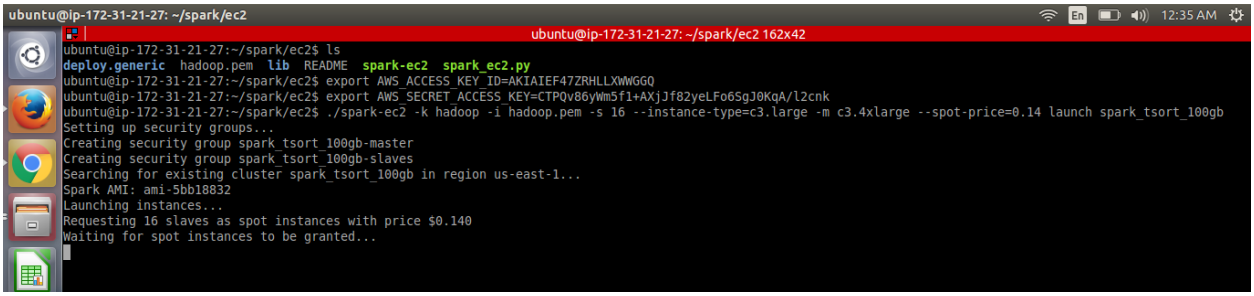
12. Now Run Program of Sort written in Scala into spark shell.

    >>cd spark/bin
    >>./spark-shell

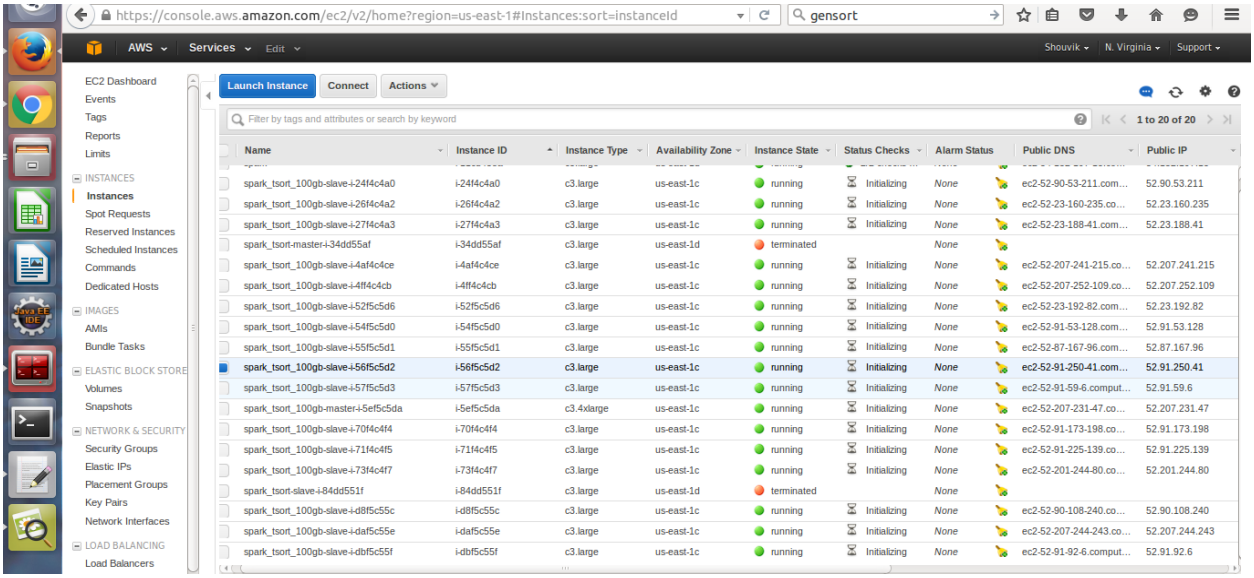
13. Copy and paste following command into shell. (Note: Run one by one)

    >>val sortFile = sc.textFile("hdfs://ec2-52-207-231-47.compute-1.amazonaws.com:9000/user/shouvik/input/input100gb")
    >>val sortedobj = sortFile.flatMap(line=>line.split("\n")).map(word=>(word.substring(0,10),word.substring(10))).sortByKey()
    >>sortedobj.saveAsTextFile("hdfs://ec2-52-207-231-47.compute-1.amazonaws.com:9000/user/shouvik/output10gb")

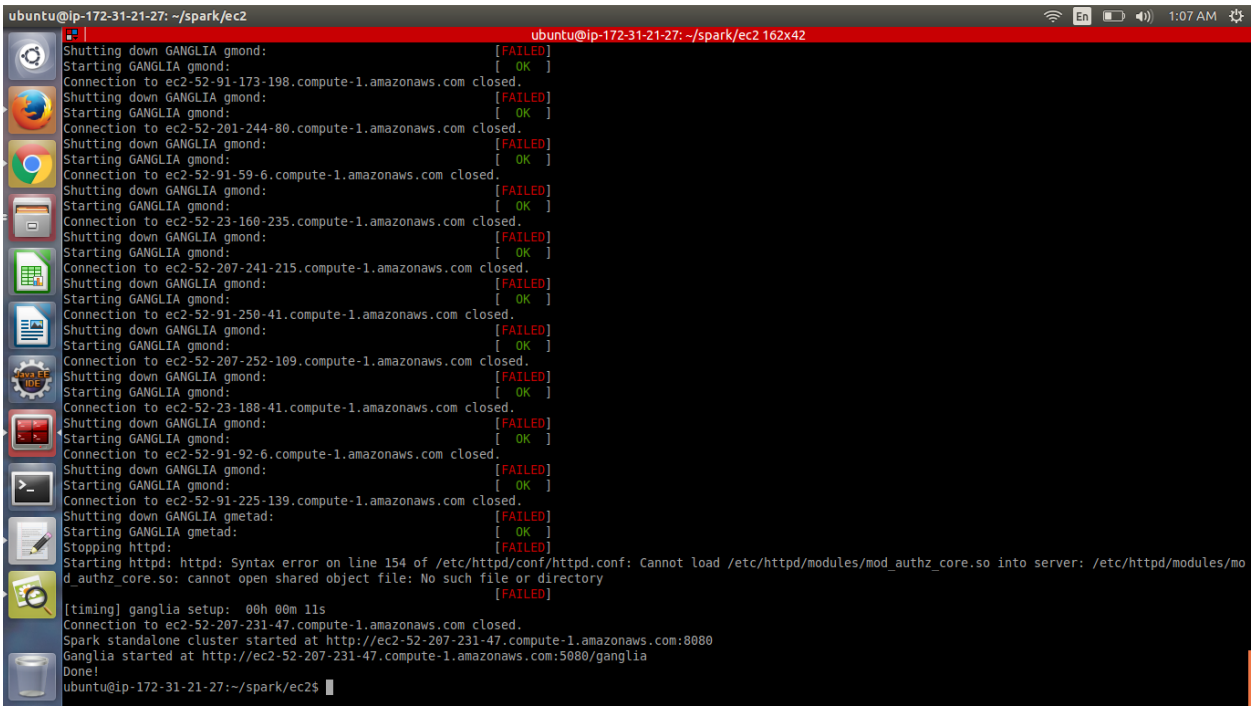
Screenshots for 16 node cluster to perform 100GB sort in Spark:



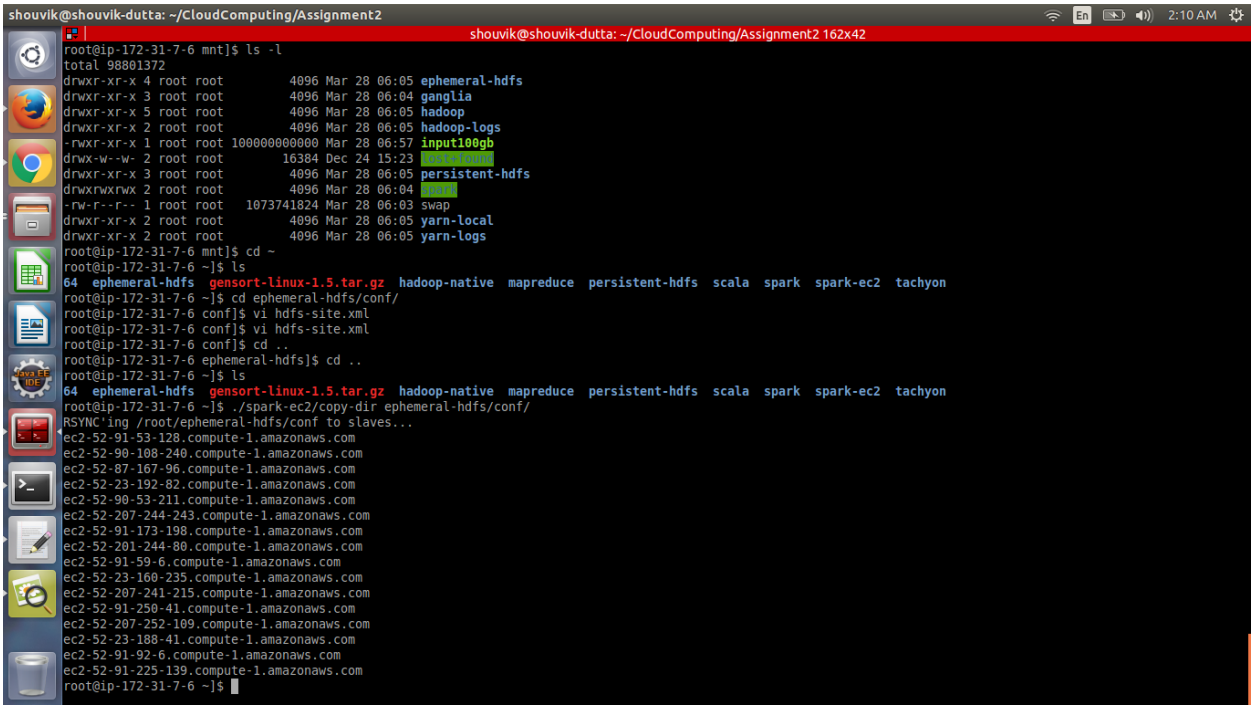
Screenshot showing start of 16 nodes cluster in Spark



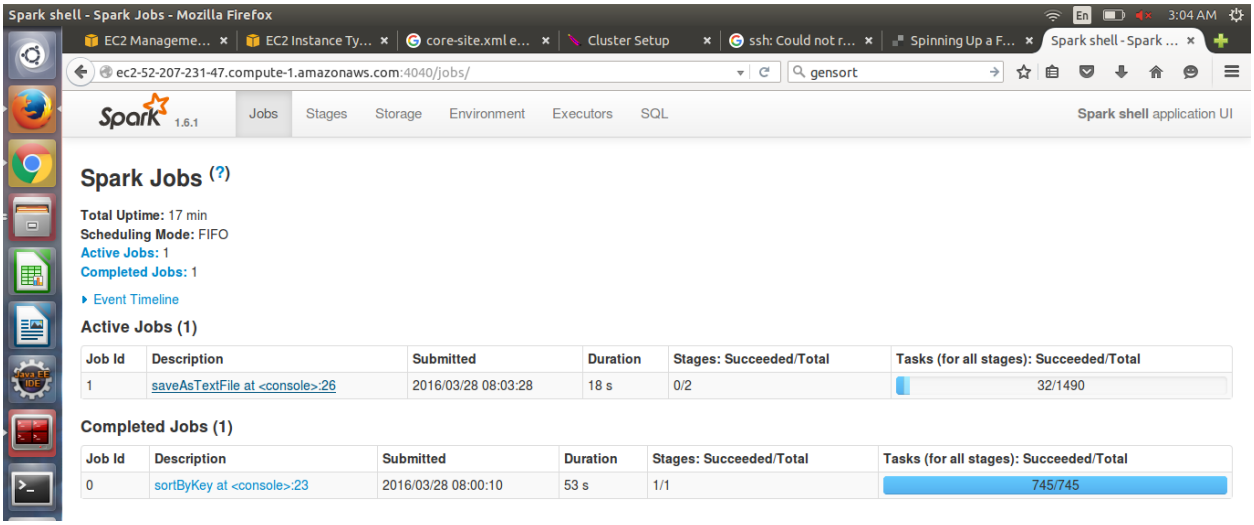
Screenshot showing 16 instance running in AWS dashboard



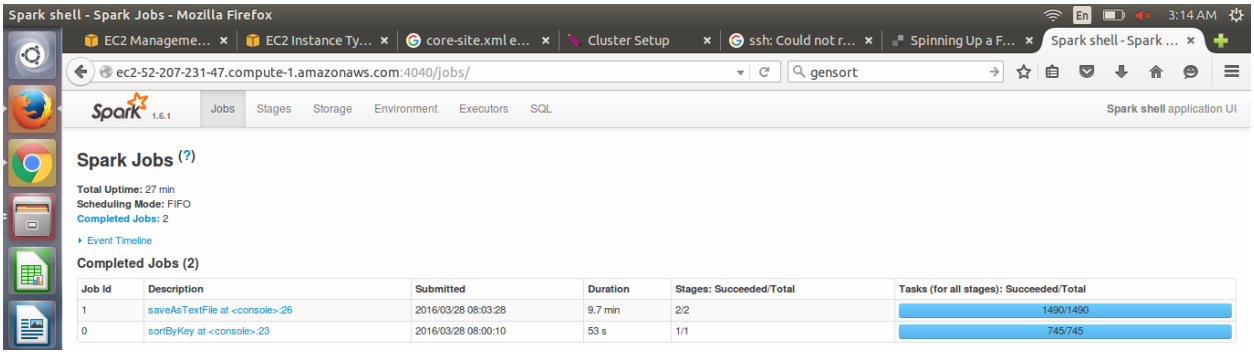
Screenshot showing 16 node cluster installation done



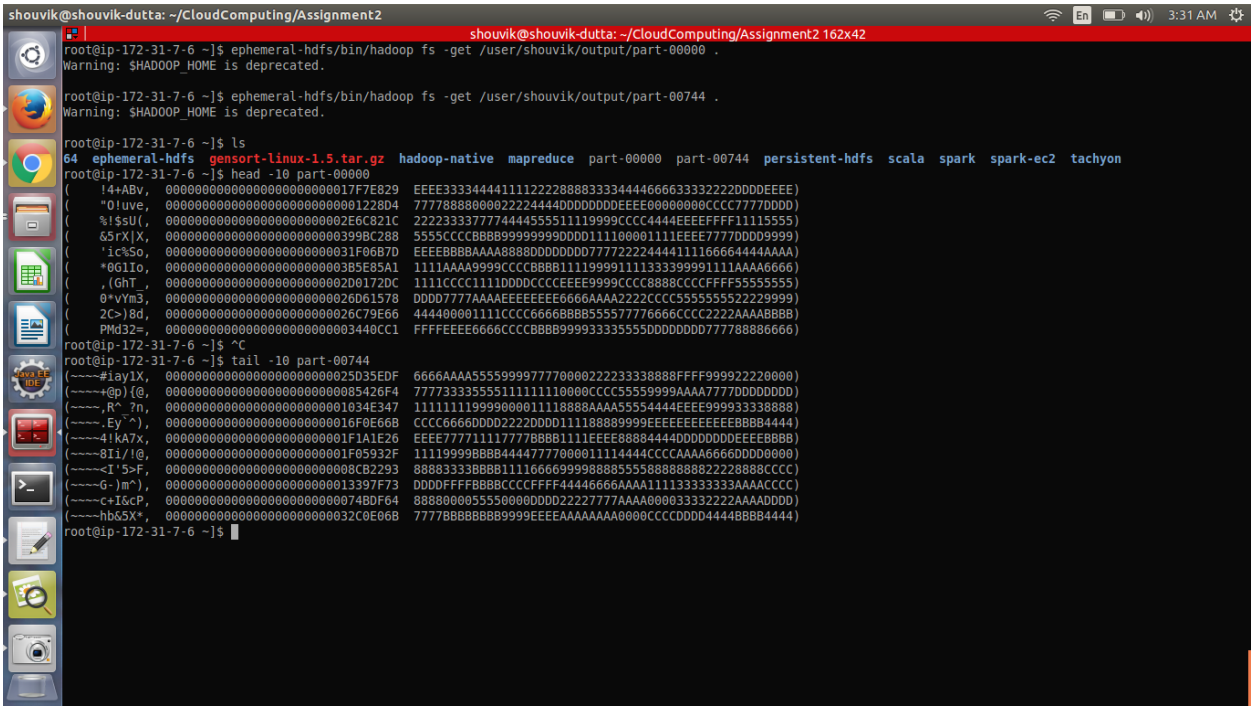
Screenshot showing config file being copied to all slaves in Spark



Screenshot showing spark jobs in browser



Screenshot showing completion of 100GB sort in browser

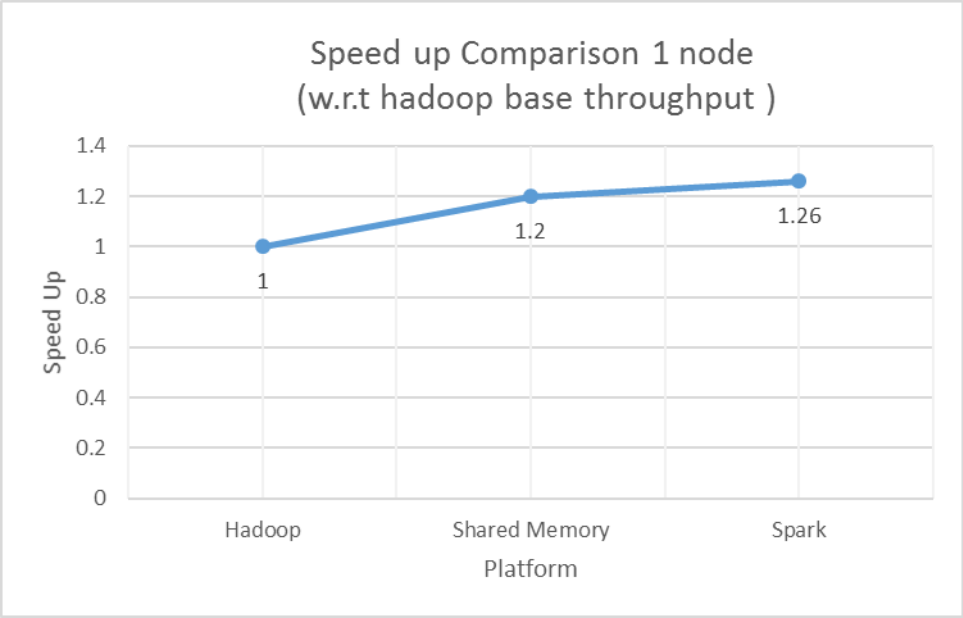


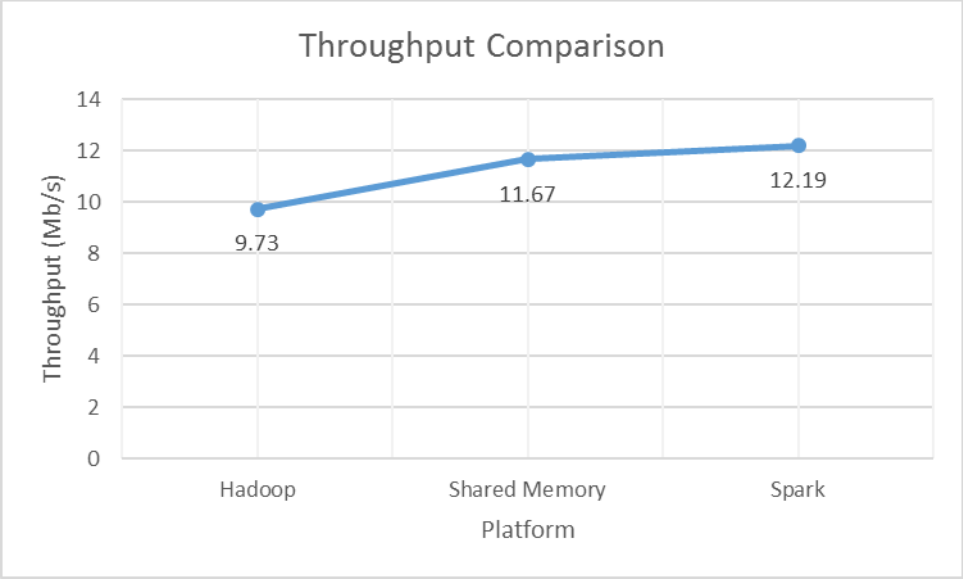
Screenshots for top 10 and bottom 10 lines of sorted output file

8 PERFORMANCE EVALUATION

Data Size = 10GB
Number of Nodes/ Threads = 1

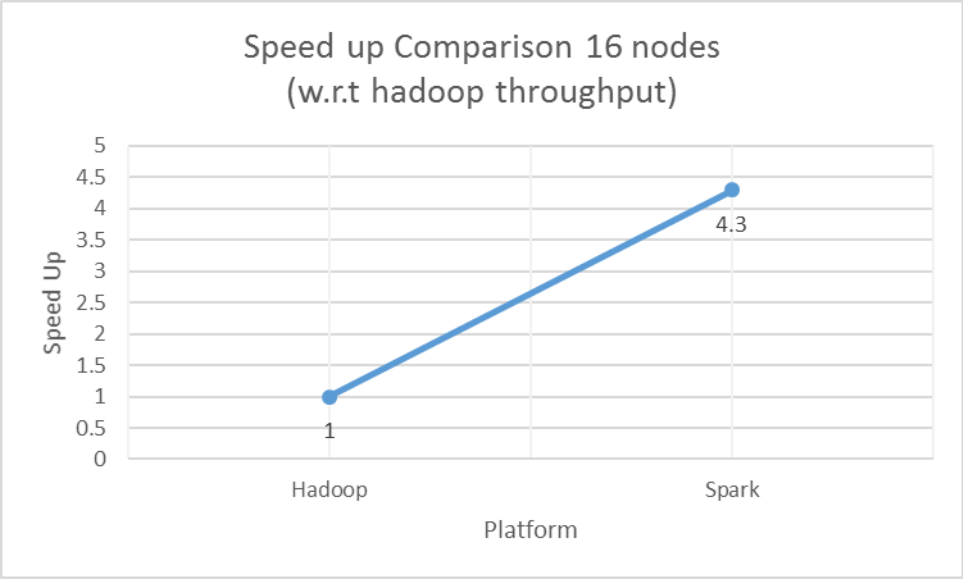
Platform	*Execution Time in Secs.*	*Throughput in Mb/s*	*Speed up (w.r.t hadoop base throughput)*
Hadoop	1027.882	9.73	1
Shared Memory	857.141	11.67	1.2
Spark	820.437	12.19	1.26





Data Size = 100GB
Number of Nodes/ Threads = 16

Platform	*Execution Time in Secs.*	*Throughput in Mb/s*	*Speed up (w.r.t hadoop throughput)*
Hadoop	2735	36.57	1
Spark	635	157.48	4.3



Result Interpretation:

For 1 node/1 thread, spark does a fair job and is the fastest to sort 10GB of data. My shared memory program does better than Hadoop. As Hadoop has a lot of overheads it is only suitable for doing data intensive task when data size is quite large. For 100GB data, spark was significantly faster than Hadoop. One reason might be as Spark keeps its final output in several files instead of merging into a single output file like Hadoop. This saves Spark some of the computation cycle which Hadoop lacks.

For 100 node scale, my bets would be on Spark based on the results that we got by performing 16 nodes Hadoop and Spark experiment.

For 1000 nodes, I guess Hadoop would fare better as it doesn't store the intermediate data. Spark might run out of resources in terms of disk space while performing a 1000 node based sort.

References:

- http://javarevisited.blogspot.com/2014/08/quicksort-sorting-algorithm-in-java-in-place-example.html
- http://insightdataengineering.com/blog/hadoopdevops/
- https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- http://spark.apache.org/docs/latest/ec2-scripts.html