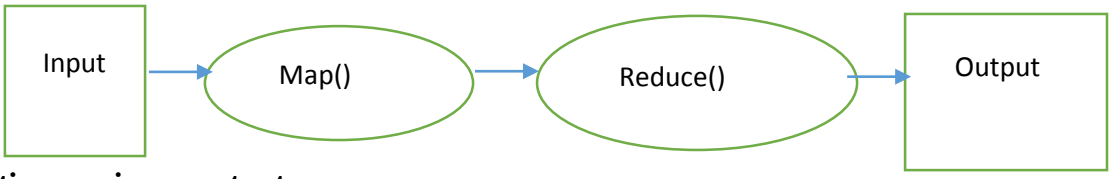
Prog-2_Report

SHARED MEMORY:

1. **Problem Description:**
   - To implement Shared Memory Sort on 10 GB file using multi-threading to take advantage of multi-cores and evaluate its performance in terms of time taken for sorting 10 GB file on Amazon EC2 c3.large instance.
   - Shared Memory is the program name for performing sorting in parallel on single machine by sharing memory using multiple threads to take advantage of multiple cores.
2. **Methodology:**
   - The large file of 10 GB was generated using gensort.
   - In order to sort a large file efficiently, file was divided into smaller chunks of size 1024 each.
   - Now to sort the file, each of these chunks are sorted and then merged back into original file.
   - For sorting each chunk, first 10 characters of each line are taken as key and rest of them as value.
   - These key value pairs are mapped into TreeMap, which maintains the key value pairs in a sorted manner.
   - Once we get sorted data, key and value are merged into line and written back to the file.
   - These chunks are then merged using MergeSort to get a sorted version of original 10 GB file.
   - To evaluate the performance of shared memory, the program was multi-threaded and results were noted for 1, 2, 4 and 8 threads.
3. **Runtime environment setup:**
   - This experiment was carried out on Amazon AWS.
   - Amazon EC2 c3.large instance was launched on spot request and was configured with 10 GB of storage.
   - Connection to the remote server was established using FileZilla for file transfer
   - All the program files were kept on Amazon EC2 host using FileZilla.
   - Instance was connected thru terminal of local host.
   - Java environment setup on instance thru terminal was carried out using following commands.
     - Sudo apt-get update
     - Sudo apt-get install openjdk-7-jdk -y
     - Sudo apt-get install ssh –y
   - To use ANT, it was downloaded using following command
     - Sudo apt-get install ant
   - Now we have environment ready to start execution of our program.

4. **Installation steps:**
    - Start Amazon Aws.
    - Launch Amazon Ec2 c3 large instance with on spot request and configure it with 40 GB data storage. EBS volume will be mounted by this.
    - Connect your instance with Local host using FileZilla to transfer the required files.
    - Put the genSort Folder on Server host thru FileZilla.
    - genSort Folder contains script that will download the gensort which will be used to generate 10 GB of data in a File named Input.txt.
    - To generate this data, give following command from terminal by setting path cd gensort/64:
      ./gensort –a 100000000 Input.txt.
    - Now we have Input.txt 10 GB file. This file will be given as input to shared memory program.
    - Put files SharedMemory.java, propertiescontroller.java files which contains the code in a folder named src and build.xml file which is ANT file on server's root.
    - Now give command run ant to run the shared memory program.
    - The output will be shown on screen i.e. time taken for sorting 10 GB data with various no of threads.
    - To check whether the output file generated was sorted or not we run valsort program on output file.
    - Run this command to check it:
      ./valsort Outputfile.txt
    - The same steps are followed by changing the number of threads in the code and readings are taken.
5. **Difficulties:**
    - The only difficulty faced here was setting up enough memory so that program can run successfully for 10 GB of data.
    - Initially as enough memory wasn't configured on instance the program stopped in between.
6. **Versions:**
    - JAVA version: 7
    - ANT version: 1.8.2

Hadoop:

1. **Problem Description:**
   - Implement Hadoop Sort application, and evaluate its performance on 1 node and 16 nodes.
   - Sort is kind of very basic and perfect example to understand the Map/Reduce programming model of Hadoop and its underlying system.
   - Hadoop 1 node and 17 node cluster has to set up and then evaluation is to be carried out for 1 GB on 1 node and 10 GB on 17 nodes cluster.
2. **Methodology:**
   - First part is to write Sorting Application for Hadoop.
   - Map Reduce programming style of Hadoop consists of 3 main parts:
     - Mapper class: This is an identity class which simply parses each line of input using the TextInputFormat and emits first 10 bytes of each line as key and rest as value.
     - Reducer class: This is also an identity reducer class which simply emits every value for every key.
     - Driver class: This is the main driver program which creates a Hadoop job and configures it.
     - Comparator class: This class sorts the input in ascending order of key.
   - 

3. **Runtime environment setup:**
   - First step would be download and setup java on instance.
   - To install java use following commands:
     - Sudo apt-get update
     - Sudo apt-get install openjdk-7-jdk -y
     - Sudo apt-get install ssh –y
   - Now install Hadoop using the install_hadoop_script.sh
   - Allocate enough memory on AmazonEC2 instance around 500 GB.
   - Also remember to turn off replication of Master node as replication in Hadoop is done around 3x which needs 3 times more memory.
4. **Installation steps:**
   - Start Amazon AWS.
   - Launch Amazon EC2 c3.large instance on spot request and add storage for 1 GB and 100 GB.

FOR 1 NODE VIRTUAL CLUSTER:

- Connect your instance to the local host by following commands on your terminal:
  - Chmod 400 pem filename (E.g. Hadoop.pem)
  - Ssh –i filename.pem ec2_public_dns
- Connect your local host to server host using FileZilla by giving master public DNS for transferring files to the Server.
- Put the script for installing Hadoop and Java on Server Host thru FileZilla.
- Run this script to install Hadoop and Java.
- On installing Hadoop, go to etc folder inside Hadoop folder.
- Change following files: Replace with your public dns in following files
  - Core-site.xml:
  - Mapred-site.xml:
  - Slaves:
  - Yarn-site.xml
- Now generate 10 GB of data using gensort. For this, put gensort folder on server thru FileZilla. Now run the script for downloading and generate 10 gb data file using following command:
  - ./gensort –a 100000000 Input.txt.
- Now change entry in host file of your master. Replace the private ip and public dns of local host with that of your master node. For this perform following steps:
  - Cd /etc
  - Sudo nano hosts
  - Now edit this file as mentioned above.
- The cluster setup is done. You can now start the nodes on Hadoop and put file in the HDFS with following commands:
  - Cd Hadoop-2.7/bin
  - ./Hadoop namenode –format
  - Cd Hadoop-2.7/sbin
  - ./start-all.sh
  - Cd Hadoop-2.7/bin
  - ./Hadoop fs –put /Input.txt /src
  - ./Hadoop dfsadmin –safemode leave
  - Cd Hadoop-2.7
  - Bin/Hadoop jar ~/Hadoop_Sort /src Output_Files will start the sorting application.
- To get output files on local disk perform following commands:
  - Bin/Hadoop fs –get Output_Files /output

- Now go to output:
  - Cd /output
  - Head -10 part-r-00000
  - Tail -10 part-r-00000
- Here for 10 GB we get only 1 output file name part-r-00000 as we have used one reducer.
- To check if file was sorted give following command from gensort folder:
  - ./valsort ~/output/part-r-00000

FOR 17 NODE VIRTUAL CLUSTER:

- Connect your instance to the local host by following commands on your terminal:
  - Chmod 400 pem filename (E.g. Hadoop.pem)
  - Ssh –i filename.pem ec2_public_dns
- Connect your local host to server host using FileZilla by giving master public DNS for transferring files to the Server.
- Put the script for installing Hadoop and Java on Server Host thru FileZilla.
- Run this script to install Hadoop and Java.
- On installing Hadoop, go to etc folder inside Hadoop folder.
- Change following files: Replace with your public dns in following files
  - Core-site.xml:
  - Mapred-site.xml:
  - Slaves:
  - Yarn-site.xml
- Create an AMI image of this instance and launch 16 instances for this image same way as we created an instance in the beginning.
- Now add the public DNS of all the slaves in the Hadoop/etc/slaves.xml file.
- Now generate 100 GB of data using gensort. For this, put gensort folder on server thru FileZilla. Now run the script for downloading and generate 100 gb data file using following command:
  - ./gensort –a 1000000000 Input.txt.
- Now change entry in host file of your master. Replace the private ip and public dns of local host with that of your master node as well as of all 16 salves. And for each of 16 slaves host file add private n public ip of that node and master. For this perform following steps:
  - Cd /etc
  - Sudo nano hosts
  - Now edit this file as mentioned above.
- A software named terminator can be used to open multiple terminals of 17 instances and broadcast instructions for all the instances.
- To bind master with slaves an RSA is generated and all 17 instances are binded with that common key.

- To create key give following command:
  - Ssh-keygen –t rsa
- To bind them together give following commands:
  - Ssh-copy-id –i ~/.ssh/id_rsa.pub ec2_master@ip
  - Ssh-copy-id –i ~/.ssh/id_rsa.pub ec2_slave1@ip
  - Ssh-copy-id –i ~/.ssh/id_rsa.pub ec2_slave16@ip
- The cluster setup is done. You can now start the nodes on Hadoop and put file in the HDFS with following commands:
  - Cd Hadoop-2.7/bin
  - ./Hadoop namenode –format
  - Cd Hadoop-2.7/sbin
  - ./start-all.sh
  - Cd Hadoop-2.7/bin
  - ./Hadoop fs –put /Input.txt /src
  - ./Hadoop dfsadmin –safemode leave
  - Cd Hadoop-2.7
  - Bin/Hadoop jar ~/Hadoop_Sort /src Output_Files will start the sorting application.
- To get output files on local disk perform following commands:
  - Bin/Hadoop fs –get Output_Files /output
- Now go to output:
  - Cd /output
  - Head -10 part-r-00000
  - Tail -10 part-r-00039
- Here for 100 GB we get 40 output files named from part-r-00000 to part-r-00039.
- So we have to concatenate all these parts to get original sorted file:
  - Cat * > /output/*
- To check if file was sorted give following command from gensort folder:
  - ./valsort ~/output/Output.txt

5. **Difficulties:**
   - The most challenging part here was changing all the entries in host files of 17 instances and configuring the 4 files of Hadoop/etc.
   - The cluster didn't start once as I had made a mistake in the host files.
   - Also, if enough memory isn't provided the sorting couldn't be performed.
   - Initially I had given only 100 GB on instance but got error in between that not enough memory as DFS needs more space while sorting.
6. **Versions:**
   - **Hadoop**: 2.7.2
   - **JAVA**: 7
   - **FOLLOWING MODIFICATIONS WERE MADE TO FILES FOR MULTIPLE NODES:**
     i. **Slaves.xml:**

Added 17 entries for all 17 instances- 1 master n 17 slaves public DNS.

ii. **Core-site.xml:**
Add following property in this file.

```
<property>
   <name>hadoop.tmp.dir</name>
   <value>/achaud/hadoop</value>
   <description>base location for other hdfs directories.</description>
 </property>
```

iii. **Hdfs-site.xml:**
No change as such

iv. **Mapred-site.xml: Add following property**

```
<property>
  <name>mapreduce.job.reduces</name>
  <value>40</value>
</property>
```

v. **Yarn-site.xml: Add following property**

```
<property>
    <name>yarn.nodemanager.disk-health-checker.max-disk-utilization-per-
    disk-percentage</name>
    <value>98.5</value>
     <description>checks all the datanodes until 98.5 percent of the disk
    utilization has reached</description>
</property>
```

- **What is master node ? What is slave node ?**
  - Master nodes oversee the following key operations that comprise Hadoop: storing data in the Hadoop Distributed File System (HDFS) and running parallel computations on that data using MapReduce. The NameNode coordinates the data storage function (with the HDFS), while the JobTracker oversees and coordinates the parallel processing of data using MapReduce.
  - Worker nodes make up the majority of virtual machines and perform the job of storing the data and running computations. Each worker node runs both a DataNode and TaskTracker service that communicates with, and receives instructions from their master nodes. The TaskTracker service is subordinate to the JobTracker, and the DataNode service is subordinate to the NameNode.
- **Why do we need to set unique port numbers to those configuration files on a shared environment? What errors or side-effects will show up if we use same ports for each user?**
  - The configuration files contain ip address with different port numbers because they are assigned for different services such as resource manager as resource tracker, webapp, admin, scheduler.

- Suppose we assign same port number for all the services then each of these services won't be able to work properly as the port number will be already occupied and so that service won't be able to start on that port.

- **How can we change number of mappers and reducers from the configuration file?**
  - We can change number of mappers and reducers from mapred-site file by adding following property.

```
<property>
  <name>mapreduce.job.reduces</name>
  <value>40</value>
</property>
```

SPARK:

1. **Problem Description:**
   - Implement Spark Sort Application. Setup cluster of 1 node and 17 nodes for 10 GB and 100 GB respectively Perform the sort on 10 GB and 100 GB for 1 node and 17 nodes on Spark.
   - Apache Spark is a lightning-fast cluster computing designed for fast computation. It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations.
2. **Methodology:**
   - At the core of Spark is the notion of a **Resilient Distributed Dataset** (RDD), which is an immutable collection of objects that is partitioned and distributed across multiple physical nodes of a YARN cluster and that can be operated in parallel.
   - Typically, RDDs are instantiated by loading data from a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat on a YARN cluster.
   - Sort Application for spark is developed in Scala which comes from Scalable Language.
   - As sort uses HDFS for storing files, first the file is taken from HDFS and loaded into memory of spark.
   - Then mapping is done in terms of key and values paires.
   - Then Sorting is performed on keys.
   - After sorting, the lines are written to output file.
   - Timing is noted for the sorting done on 10 GB and 100 GB data.
3. **Runtime Environment Setup:**
   - To setup environment for spark, first we have to download spark with its latest version.
   - Compared to Hadoop, configuration as well as setup for spark is very easy and less complicated.
   - Once spark is downloaded follow the steps mentioned in Installation steps and you're done with setup.

4. **Installation Steps:**

<u>1 NODE VIRTUAL CLUSTER</u>

- First login to Amazon AWS account.
- Get your AWS Access key and Secret Access Key id in a safe place.
- Also, store your pem file in a safe place.
- To create an instance from terminal, follow following commands:
  - export AWS_ACCESS_KEY_ID=…
  - export AWS_SECRET_ACCESS_KEY=…
  - chmod 400 Spark.pem
- Go to the folder where you downloaded the spark
  - cd/spark.../ec2
- Launch spark using following command.
  - ./spark-ec2 -k Spark -i /home/anuradha/Desktop/Spark/Spark.pem -s 1 -t c3.large --spot-price=0.15 --ebs-vol-size=50 launch spark
- Login to spark using following command.
  - ./spark-ec2 -k Spark -i /home/anuradha/Desktop/Spark/Spark.pem login spark
- Generate 10 GB file using following command.
  - ./gensort -a 10000000 /achaud/Input
- Put data into HDFS by this command.
  - ./hadoop fs -Ddfs.replication=1 -put /vol0/Input /root/ephemeral-hdfs/bin/Input
- Sort the data using following command.
  - ./spark-shell -i /root/sort.scala
- Now to get the output file which is currently in HDFS use following commands.
  - cd ephemeral-hdfs/bin
  - ./hadoop fs -copyToLocal /root /root/output
- For 10 GB 7 parts will be created.
  - head -10 part-r-00000
  - tail -10 part-r-00007

<u>17 NODE VIRTUAL CLUSTER:</u>

- For 17 node cluster same steps are to be followed except you have to now generate 100 GB of data using gensort and while launching give following command:
  - ./spark-ec2 -k Spark -i /home/anuradha/Desktop/Spark/Spark.pem -s 16 -t c3.large --spot-price=0.15 --ebs-vol-size=500 launch spark

5. **Difficulties:**

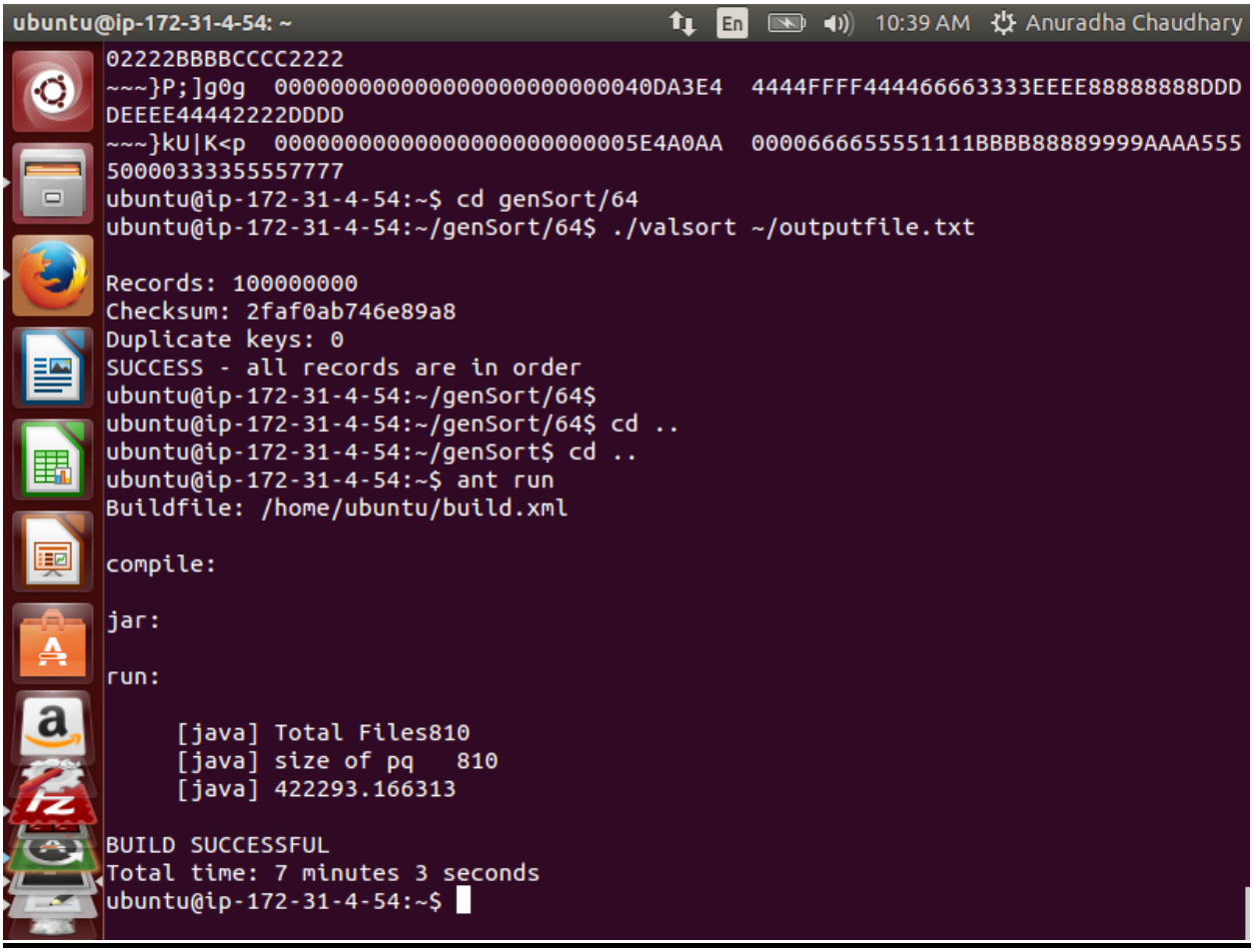
- Due to insufficient memory, program stopped. Then mounted ebs volume to provide enough memory.

6. **Versions:**
   - **JAVA: 7**
   - **SPARK: 1.6.1**

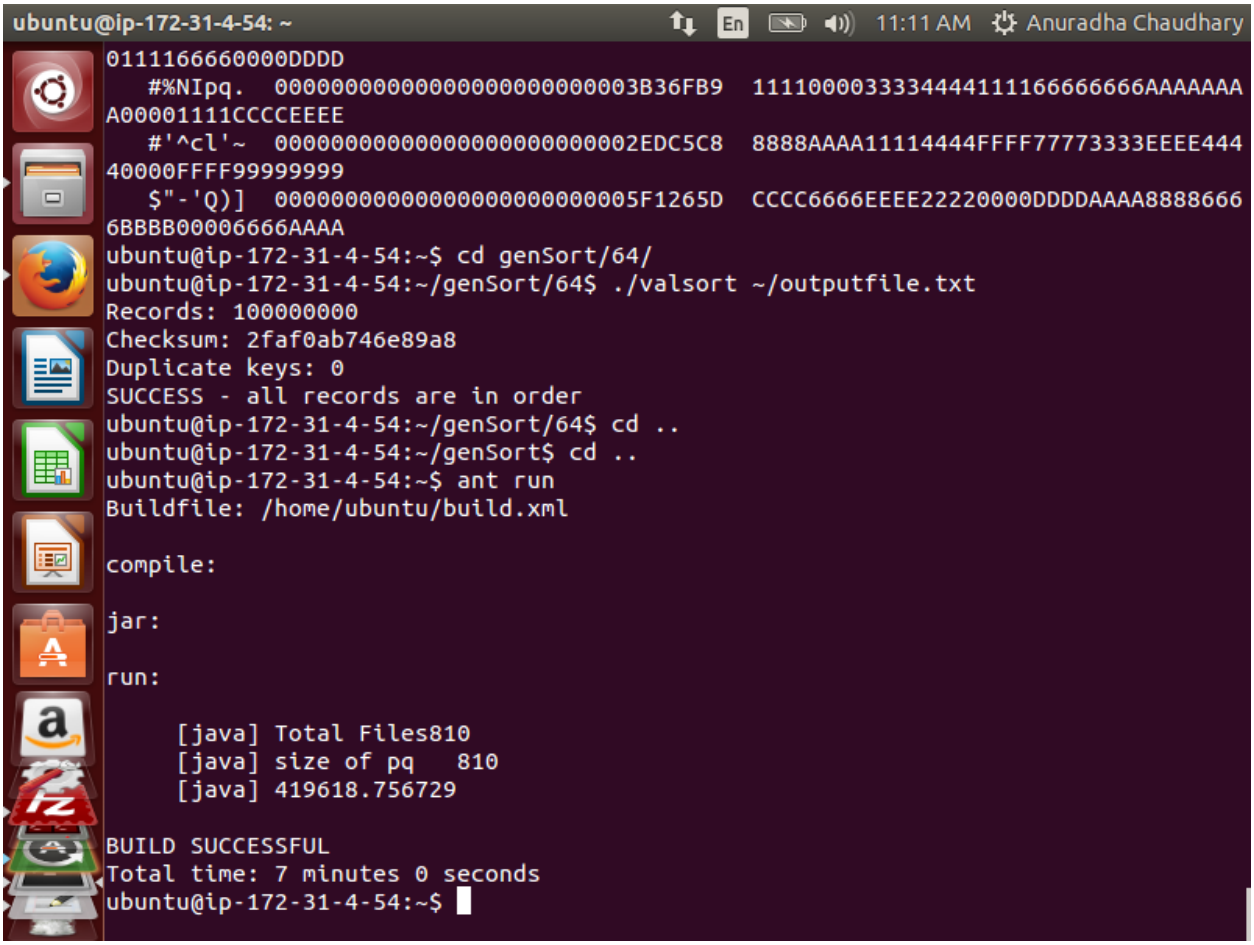
PERFORMANCE:

1. SHARED MEMORY:

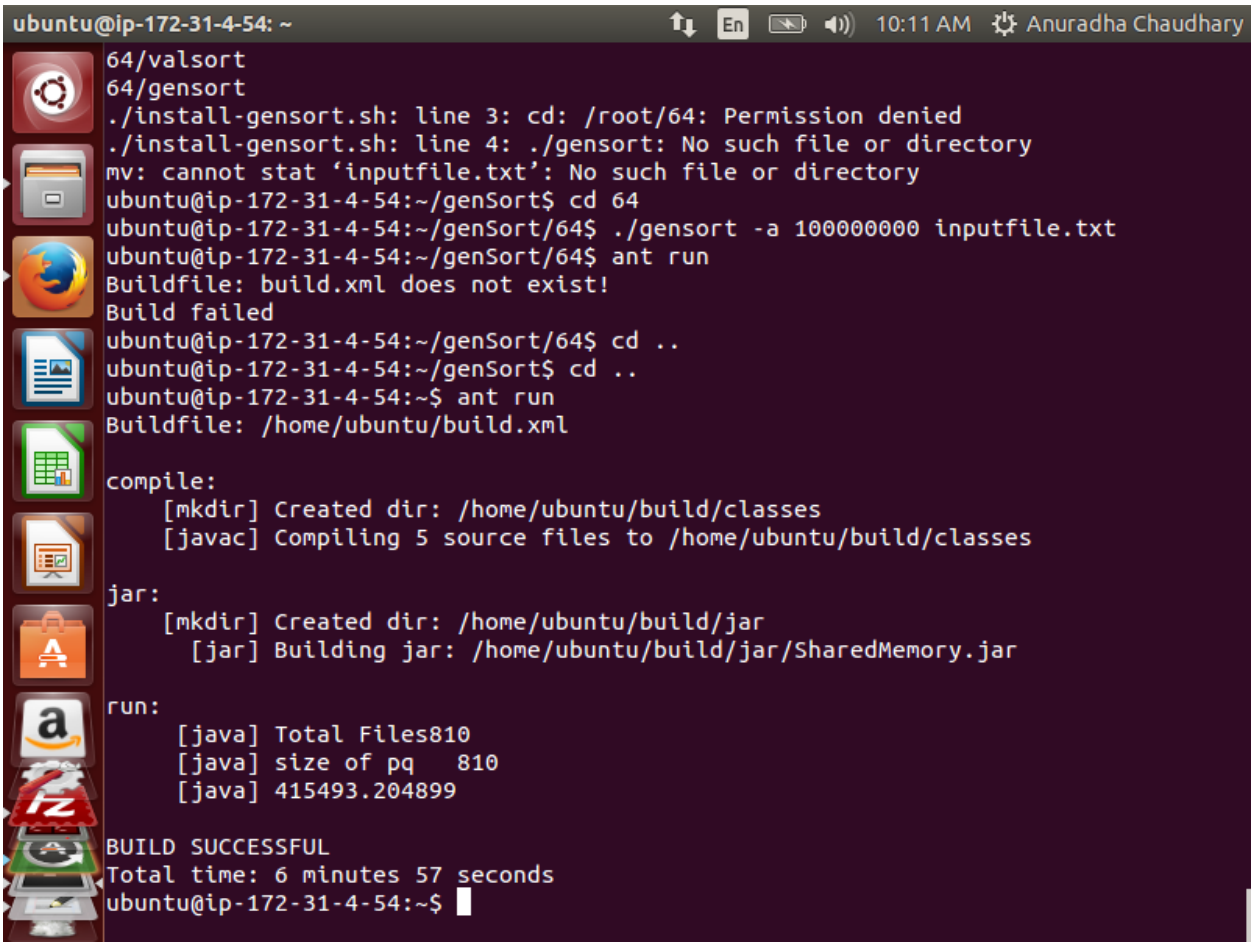
A. Time taken for sorting with 1 thread



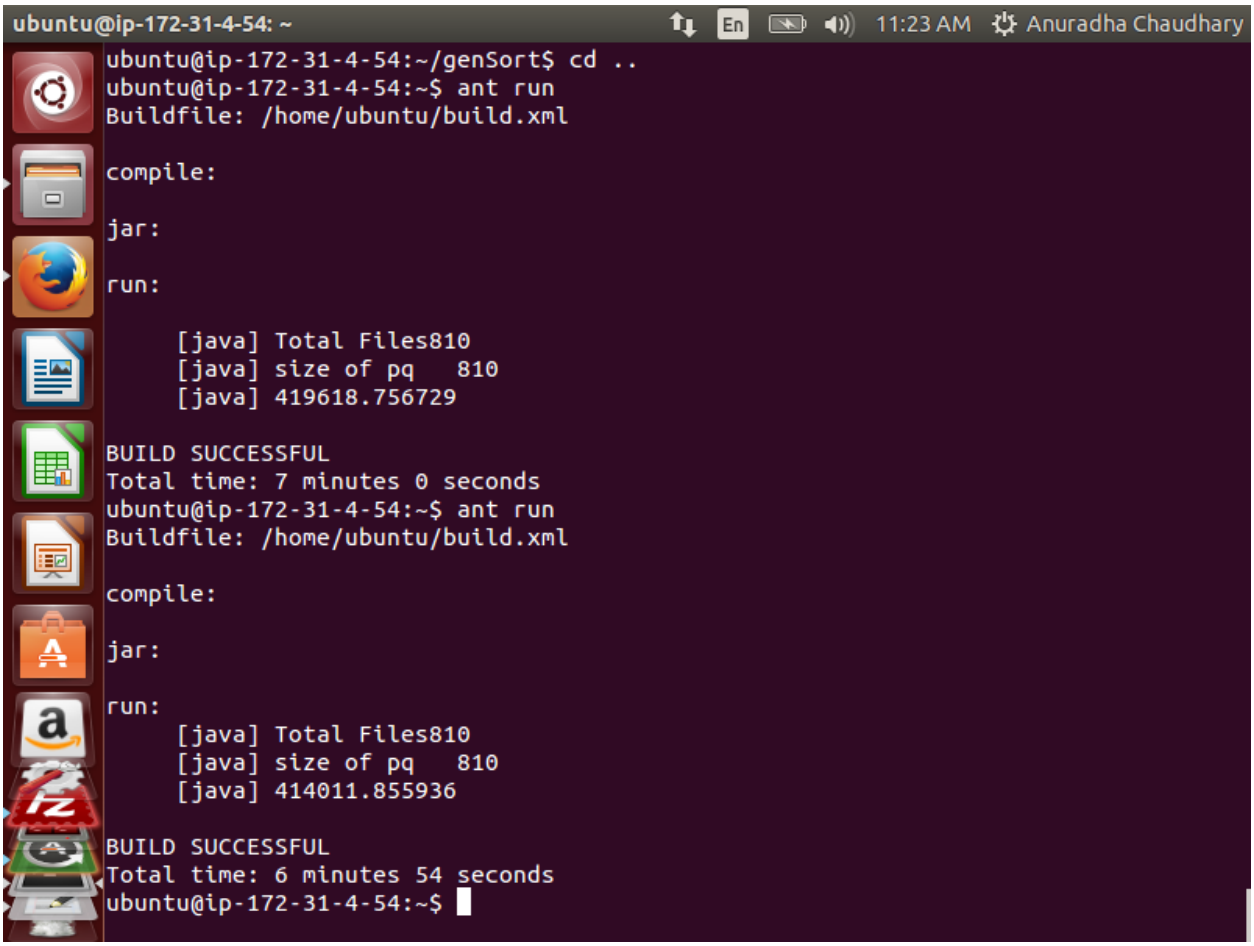
B. Time taken for sorting with 2 threads:



C. Time taken for sorting 4 threads:

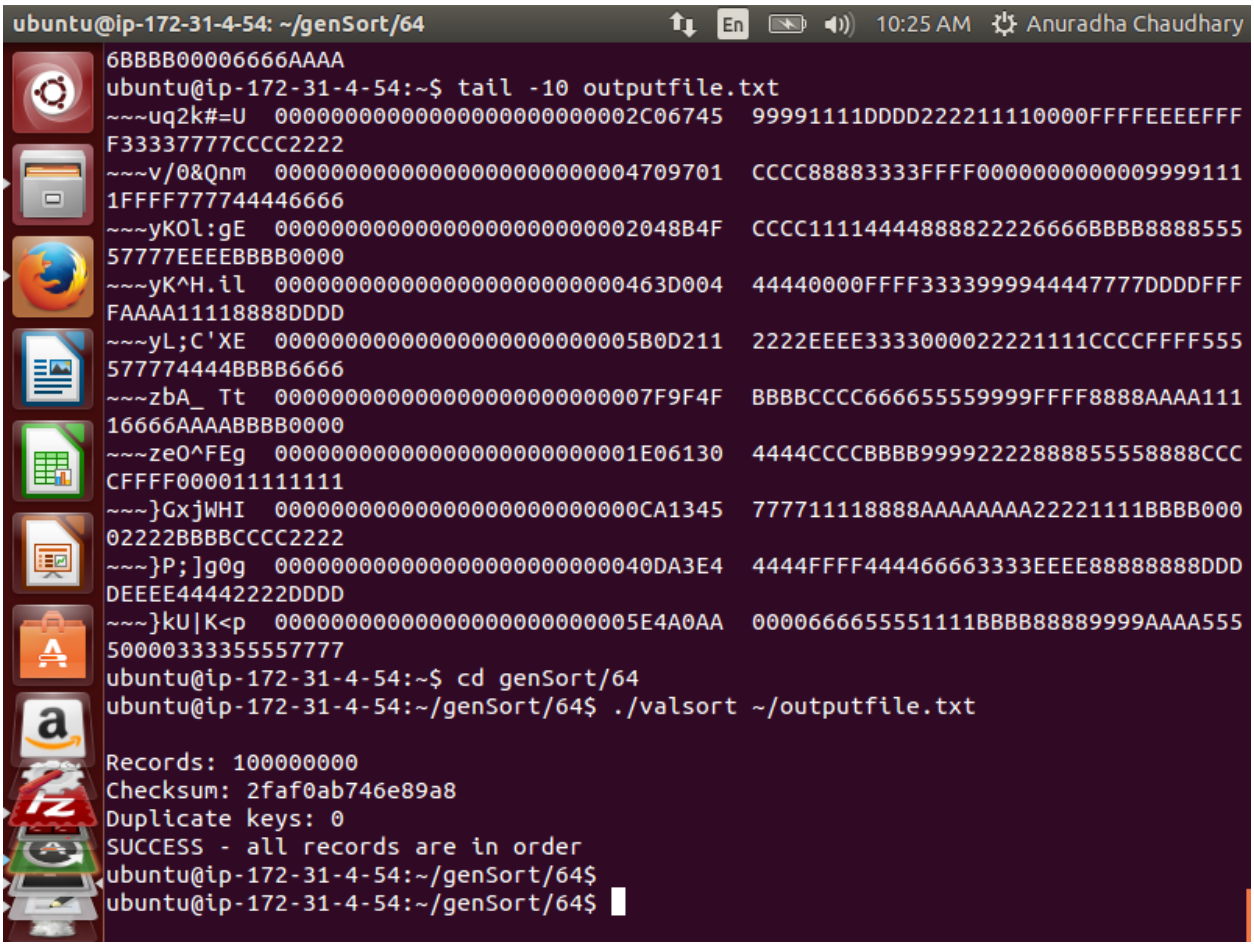


D. Time taken for sorting with 8 threads:

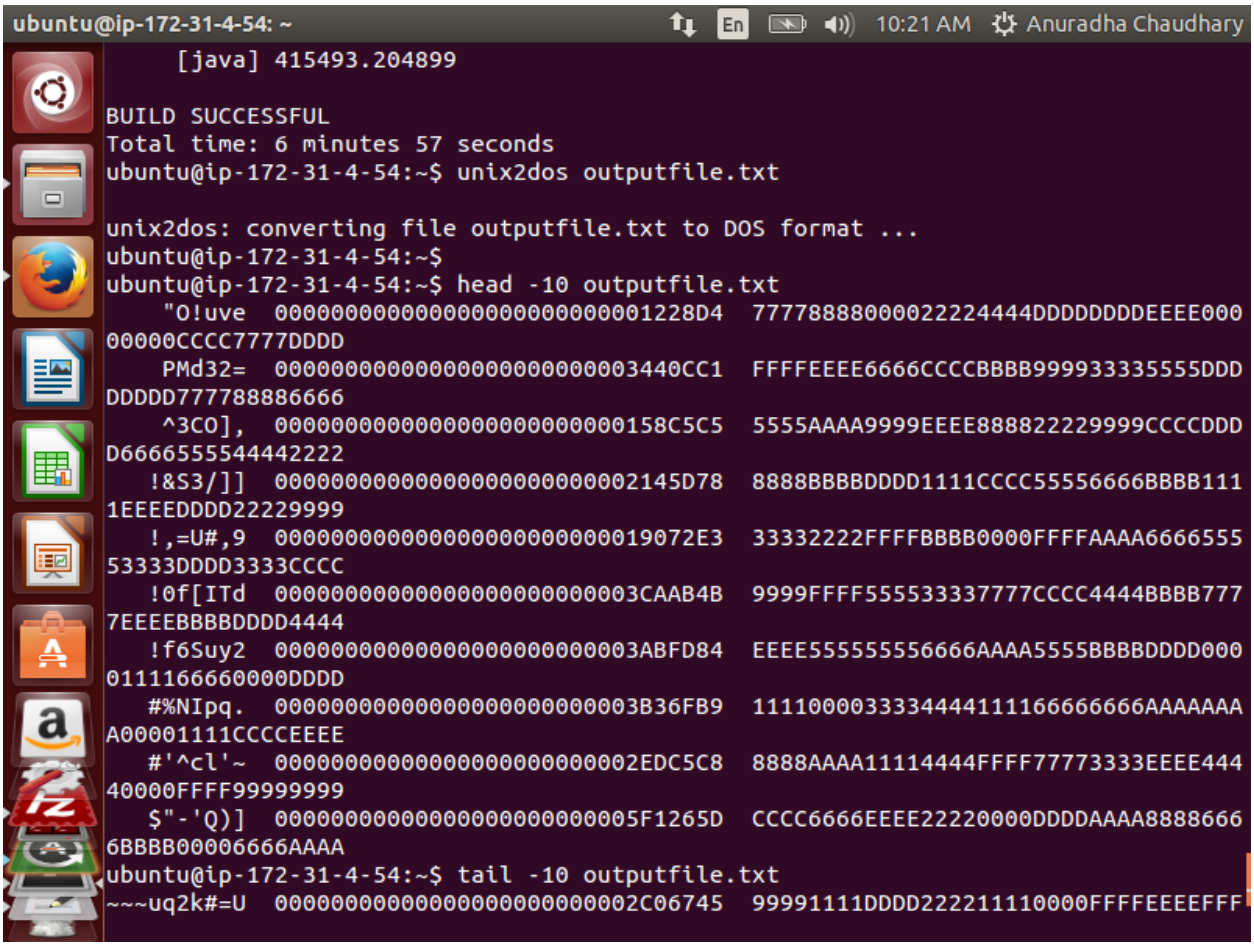


As it can be seen from above screenshots, we get least time for sorting 10 GB of data when 8 threads are running in parallel.

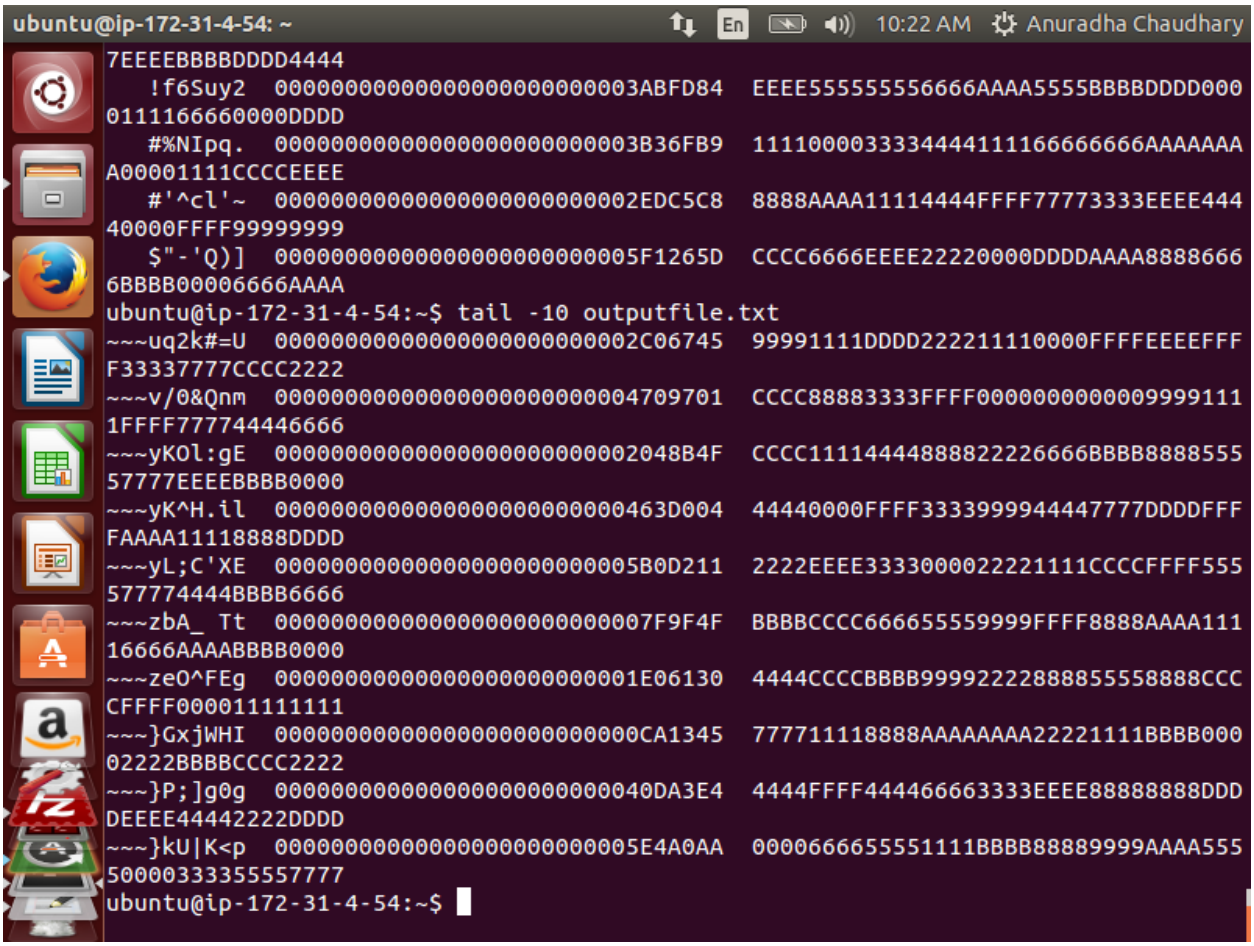
E. Output of valsort:



F. First 10 lines of sort file:

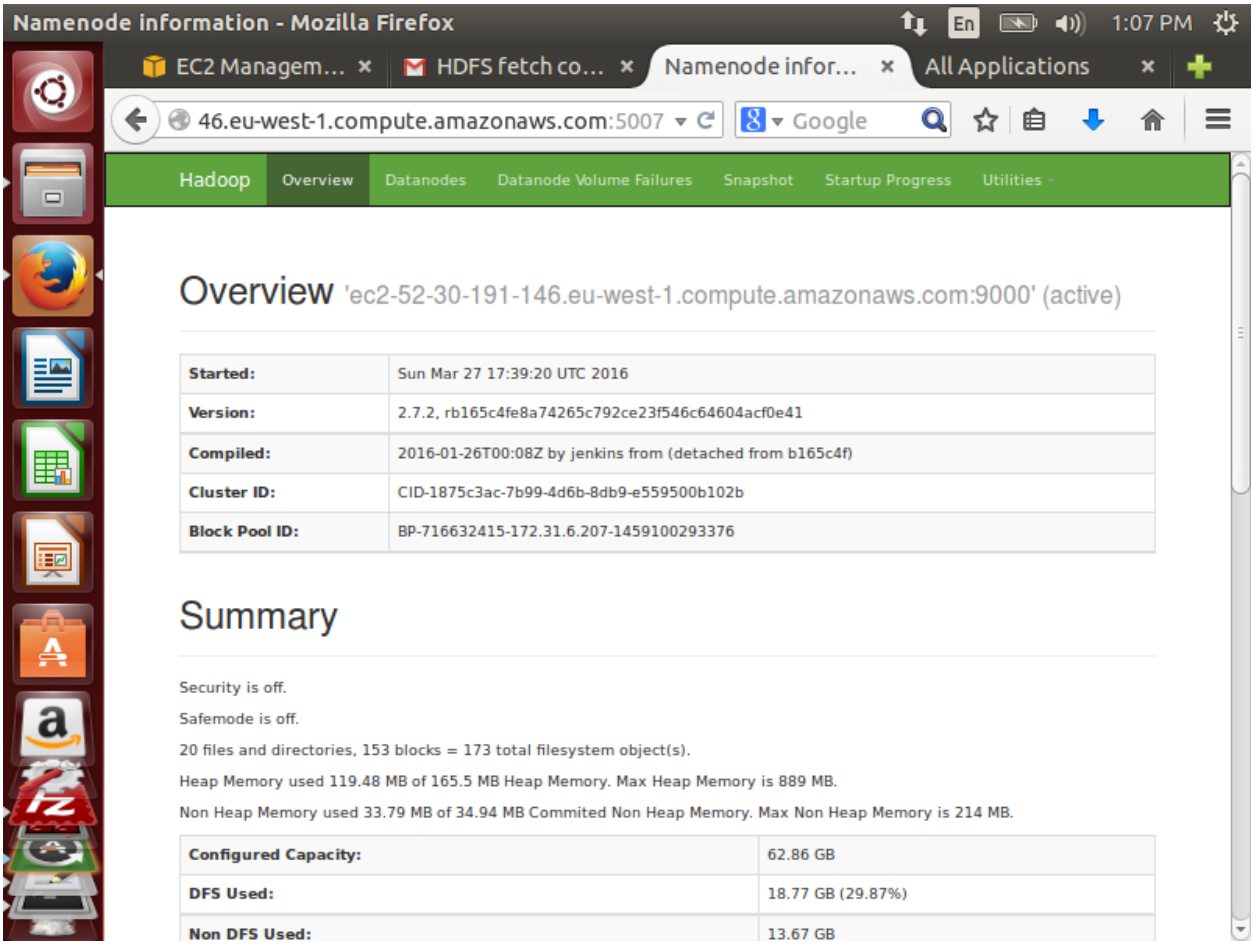


G. Last 10 lines of a sorted file:

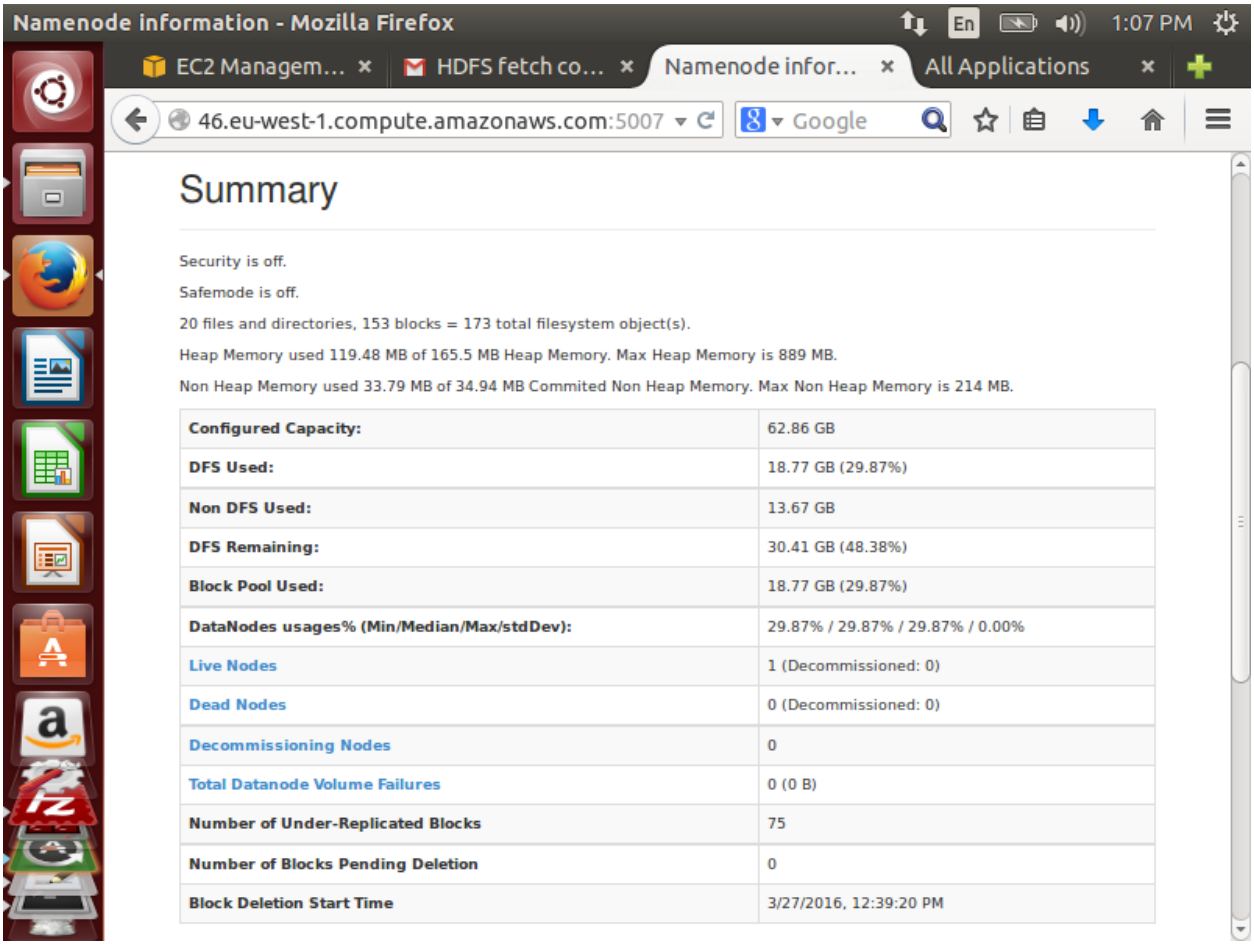


2. HADOOP:

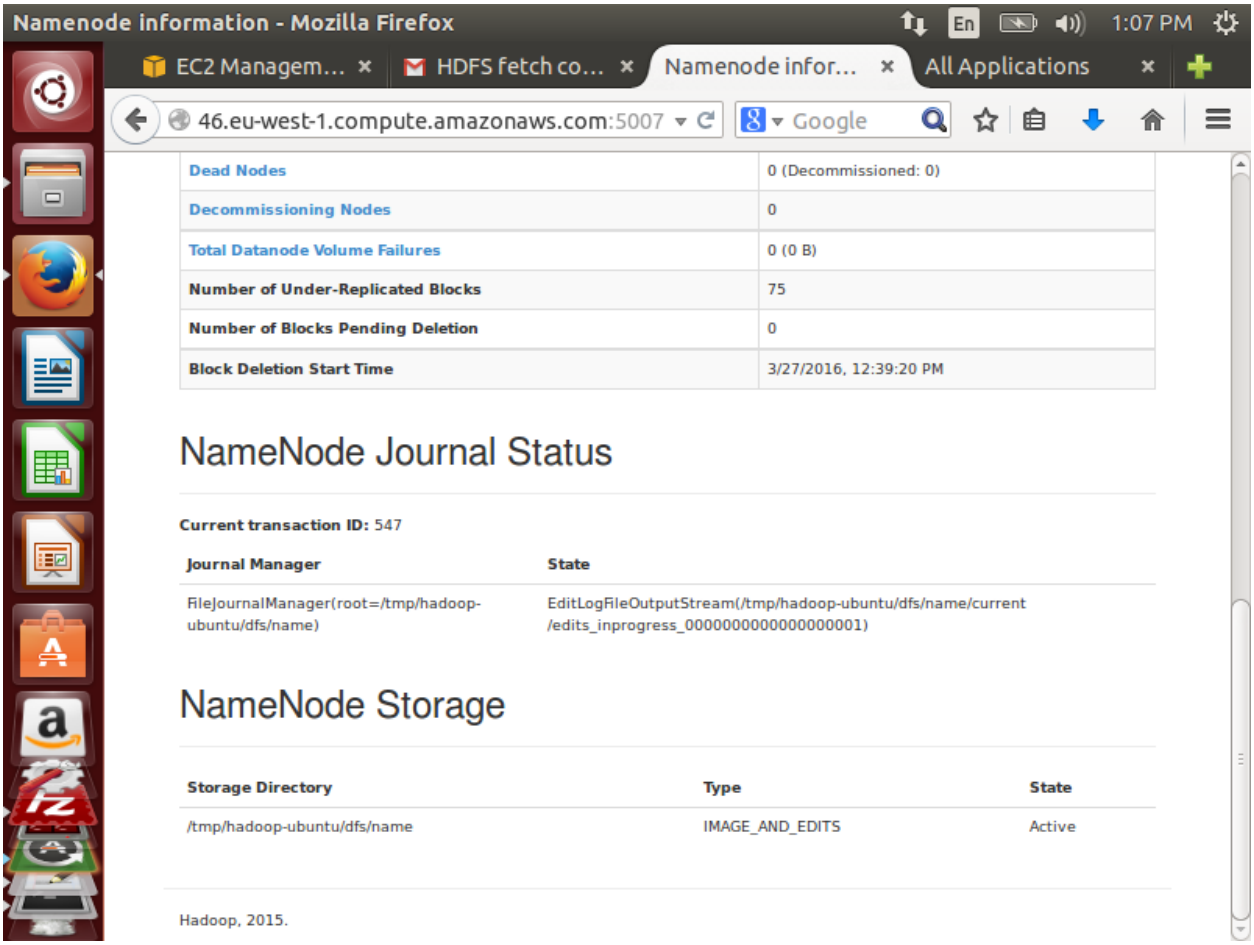
A. Namenode started for sorting 10 GB of data on 1 Node Virtual cluster.



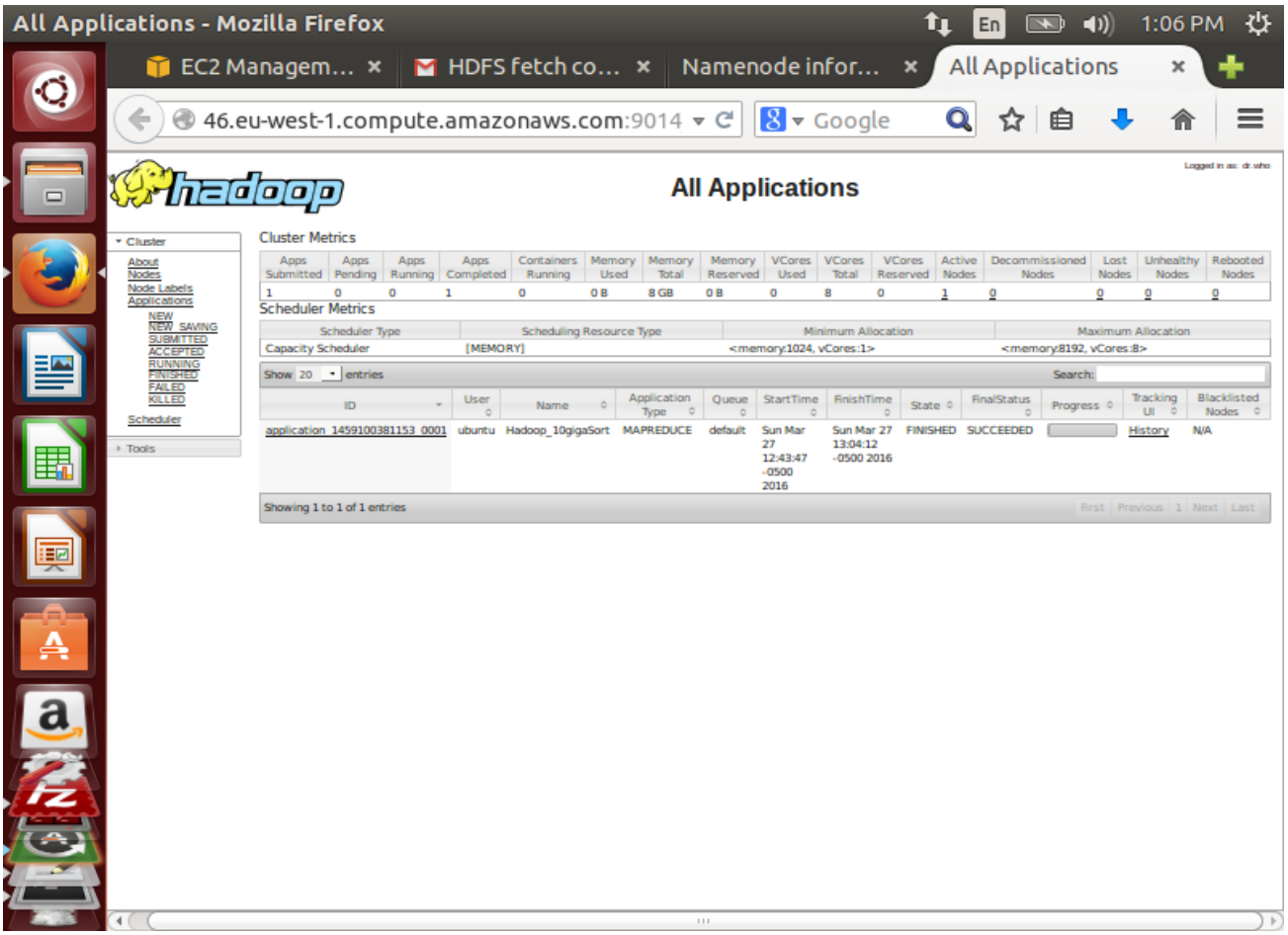
B. Namenode started for sorting 10 GB of data on 1 Node Virtual cluster.



C. Namenode started for sorting 10 GB of data on 1 Node Virtual cluster.

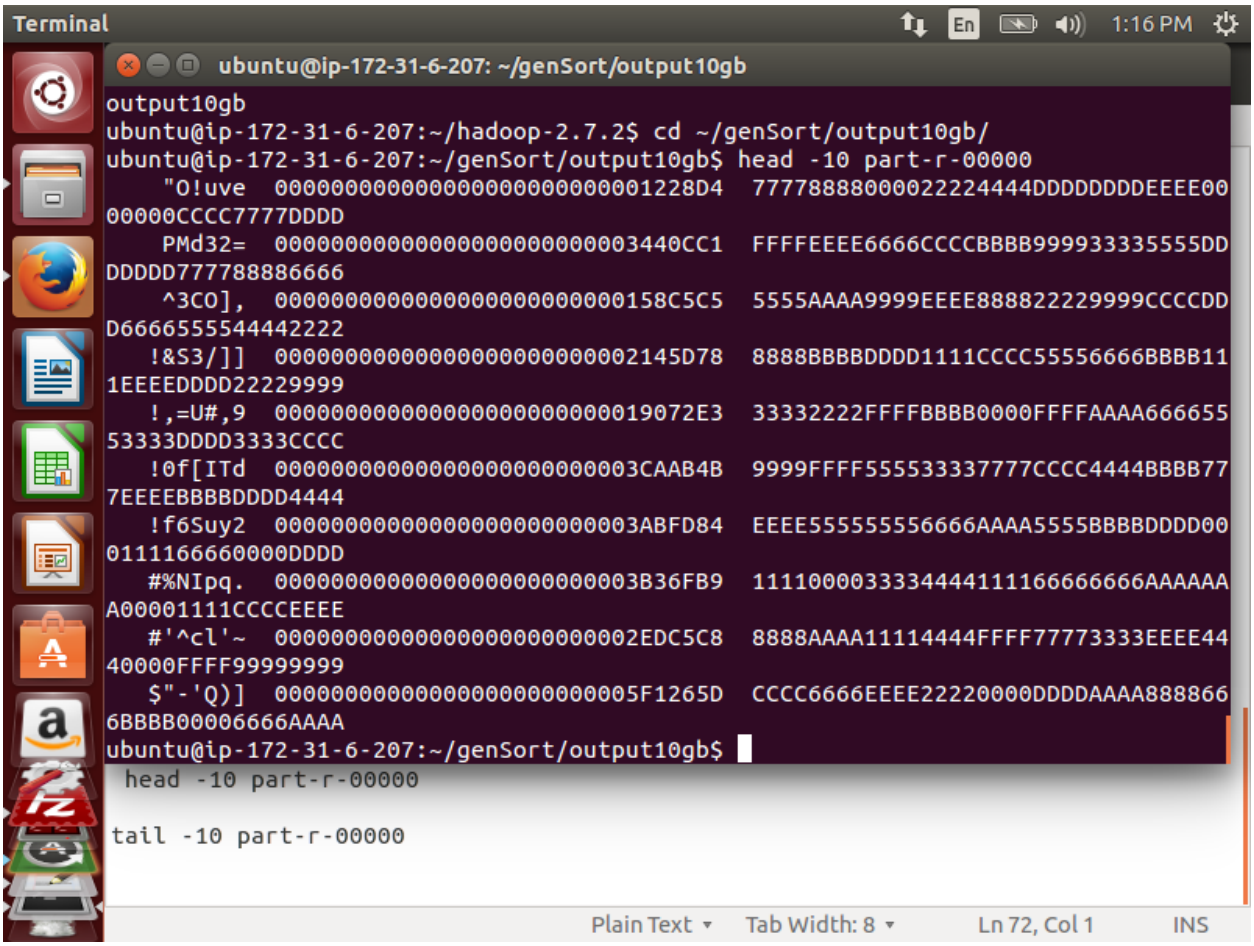


D. Application status on job completion

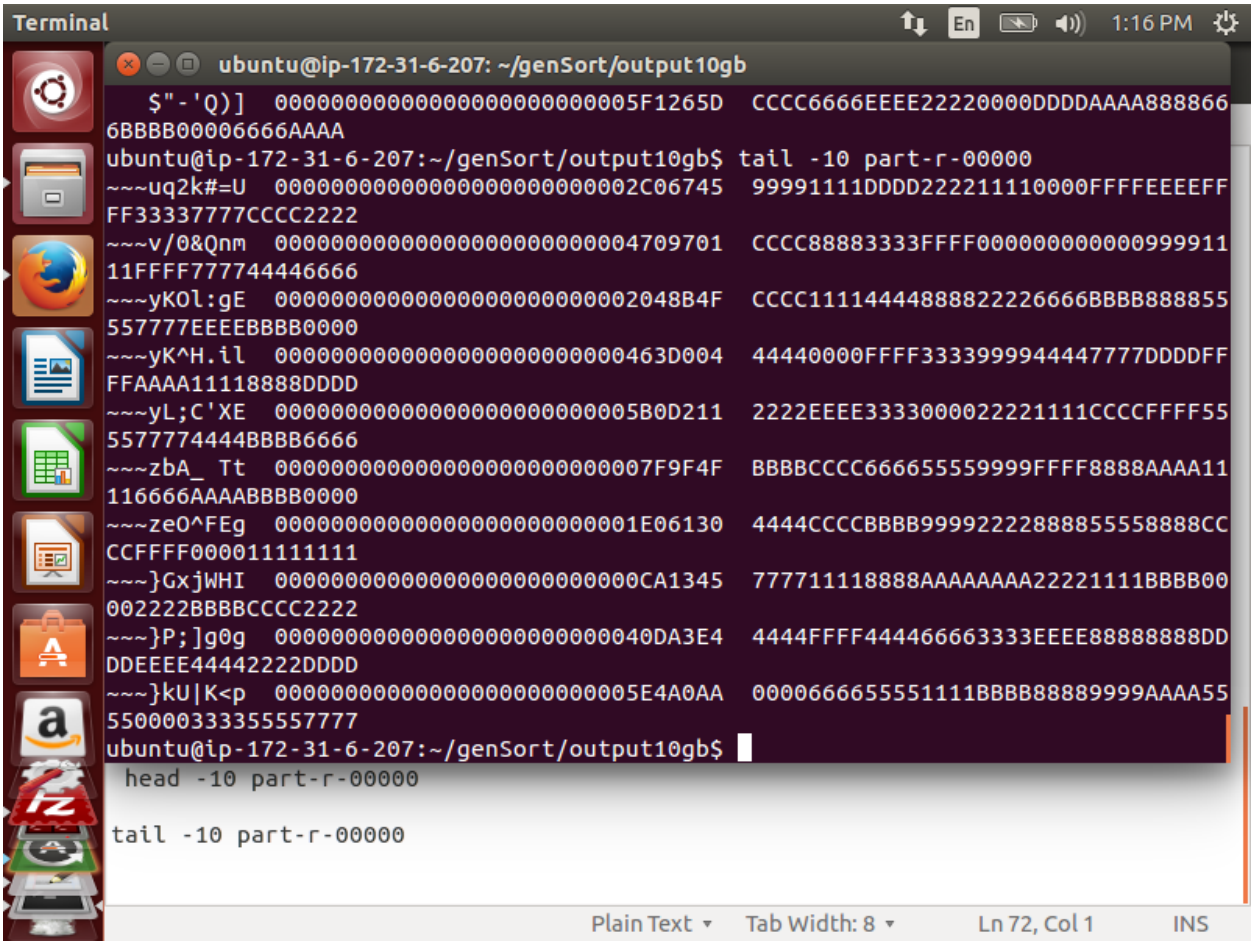


Here, time taken by Hadoop to sort 10 gb file is about 20 mins.

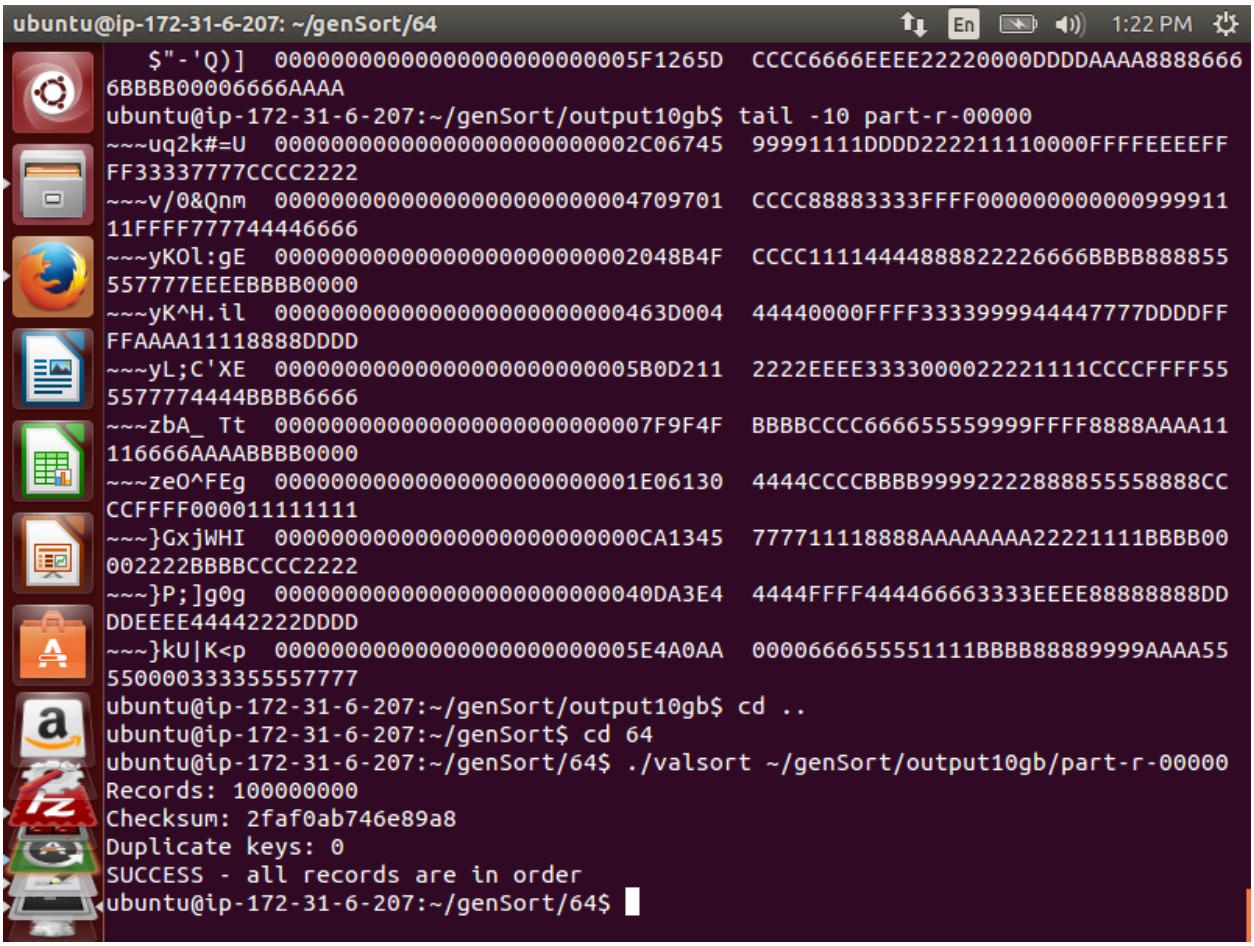
E. First 10 lines of sorted data:



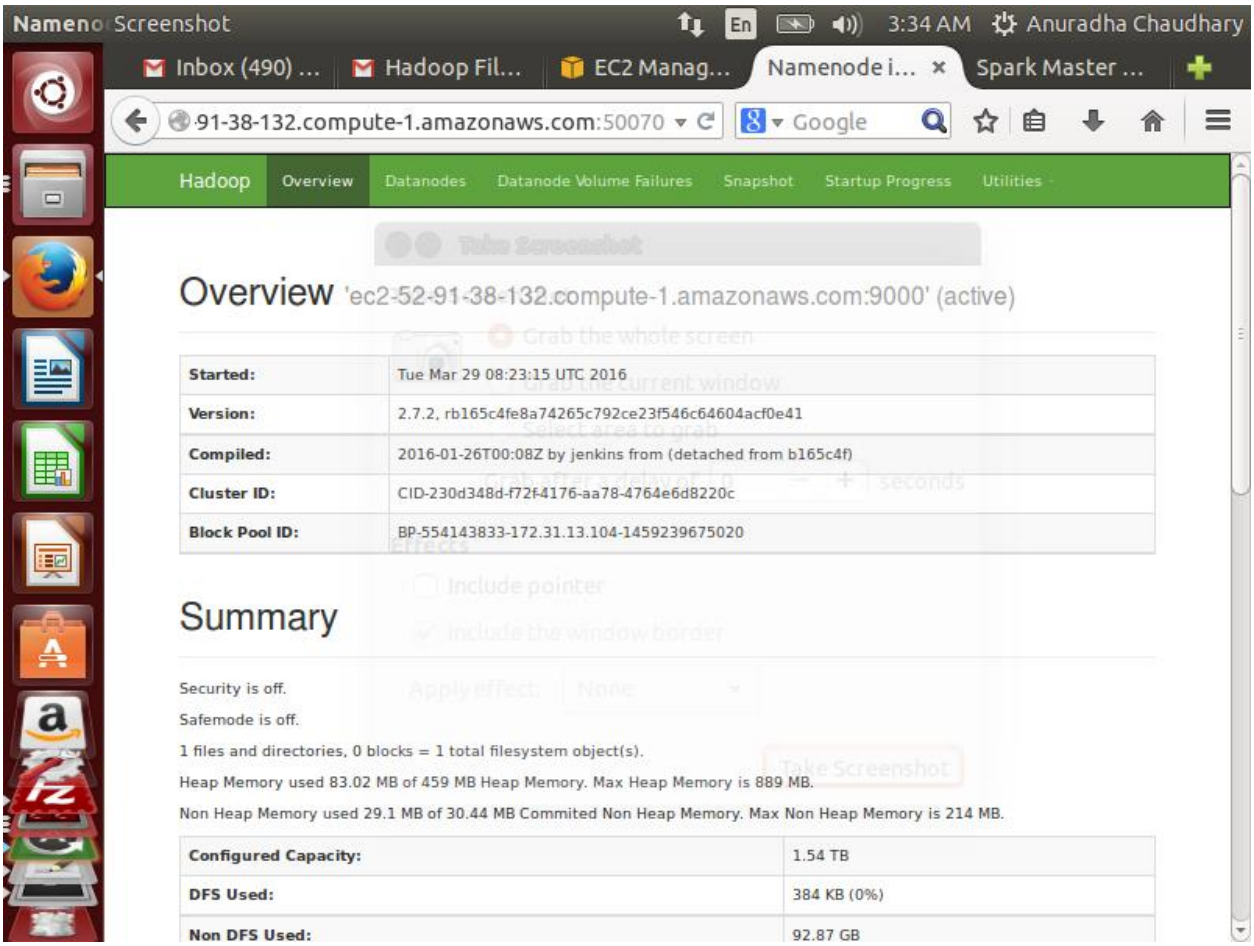
F. Last 10 lines of sorted data file



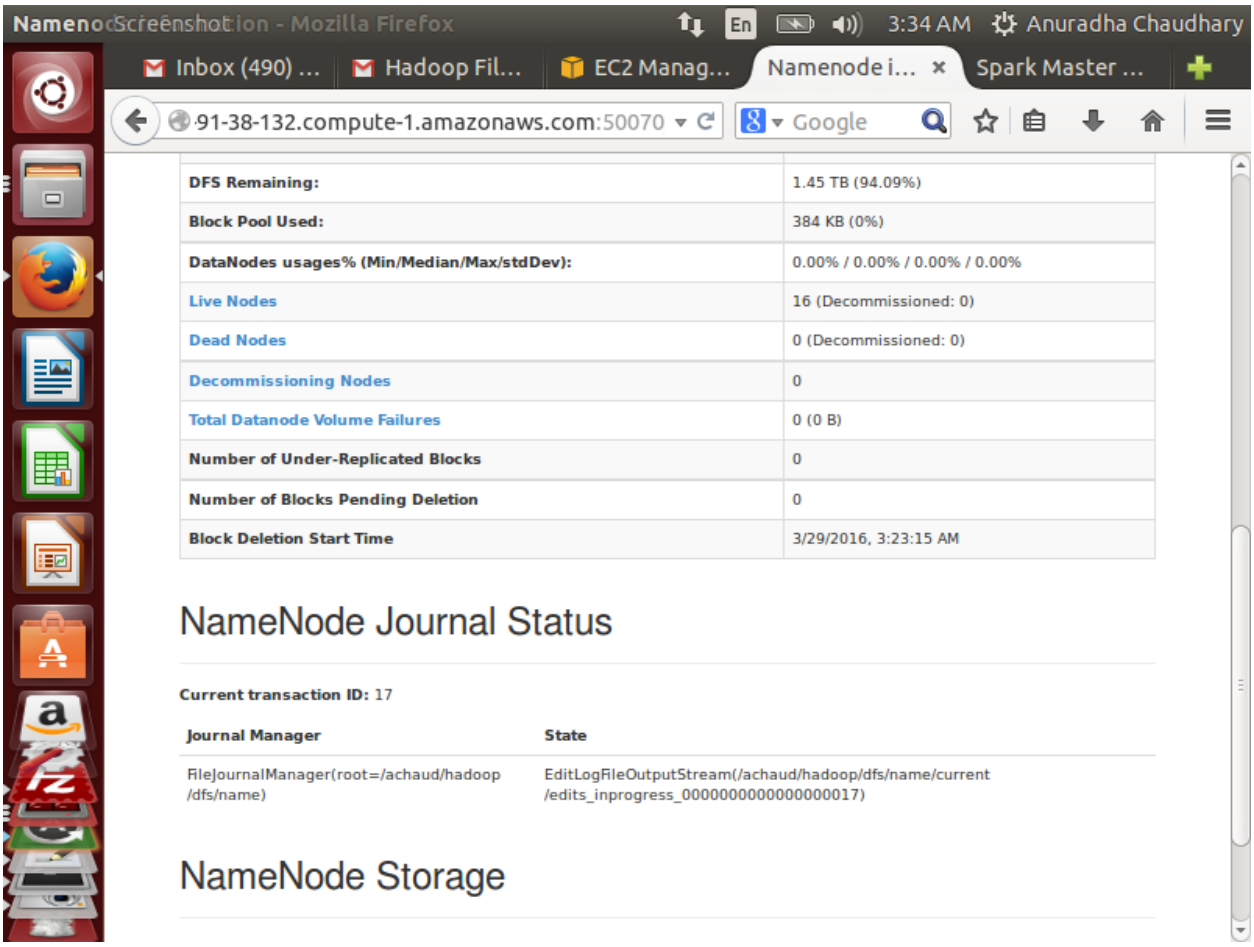
G. Result of valsort on 10gb data file



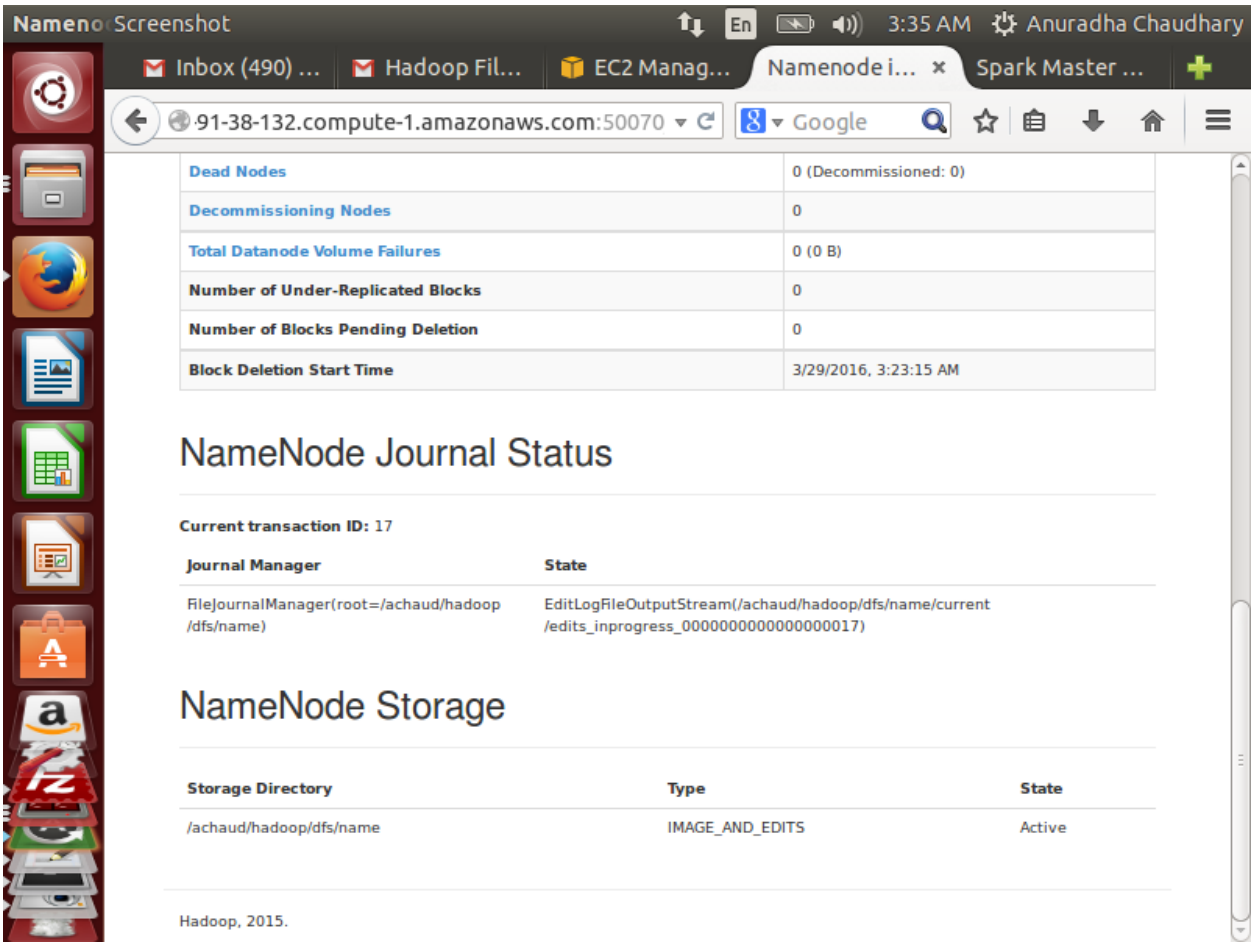
H. Namenode started for 100 GB of data file with 16 nodes.



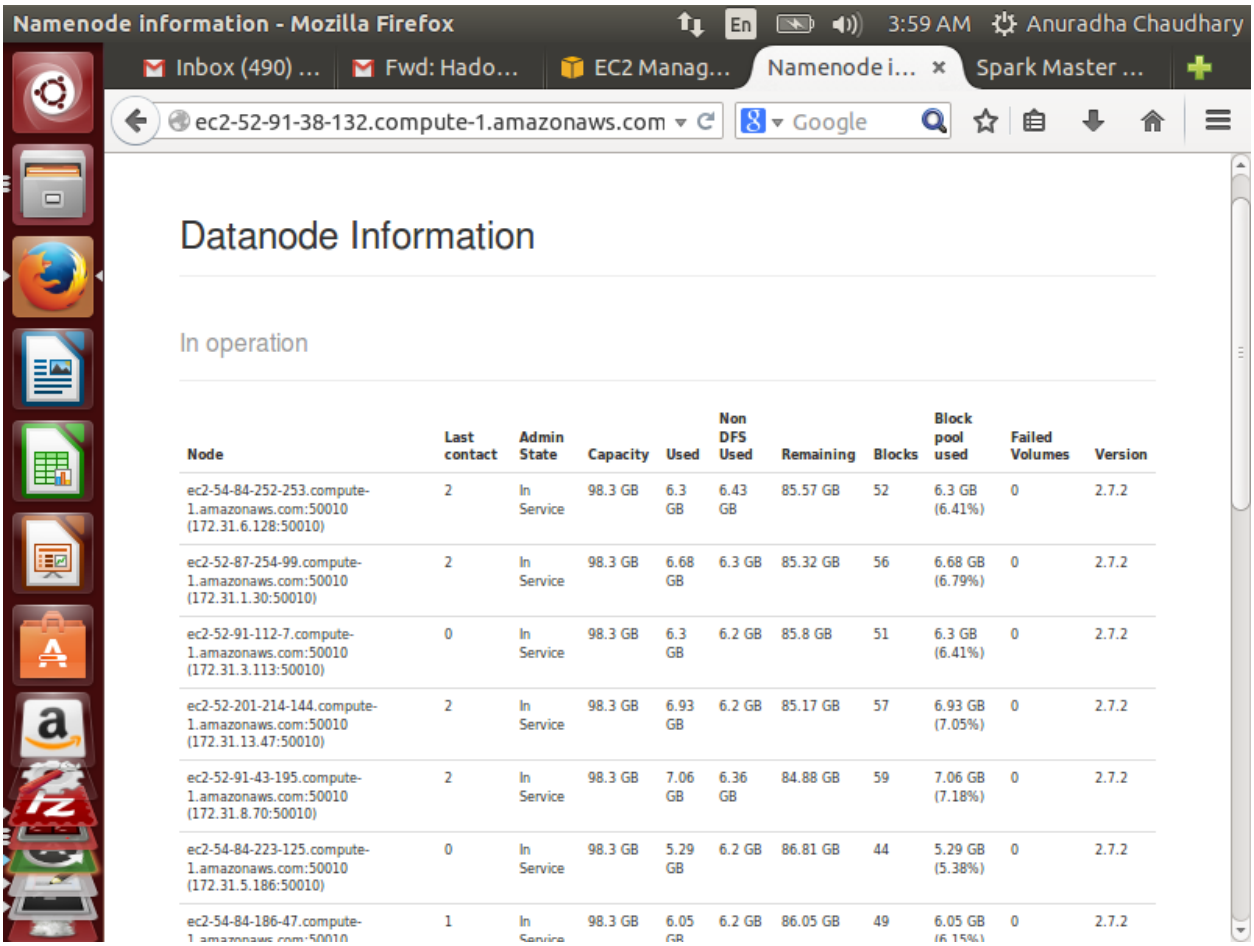
I. Namenode started for 100 GB of data file with 16 nodes.



J. Namenode started for 100 GB of data file with 16 nodes.



K. Datanode started for 100 GB of data file with 16 nodes.

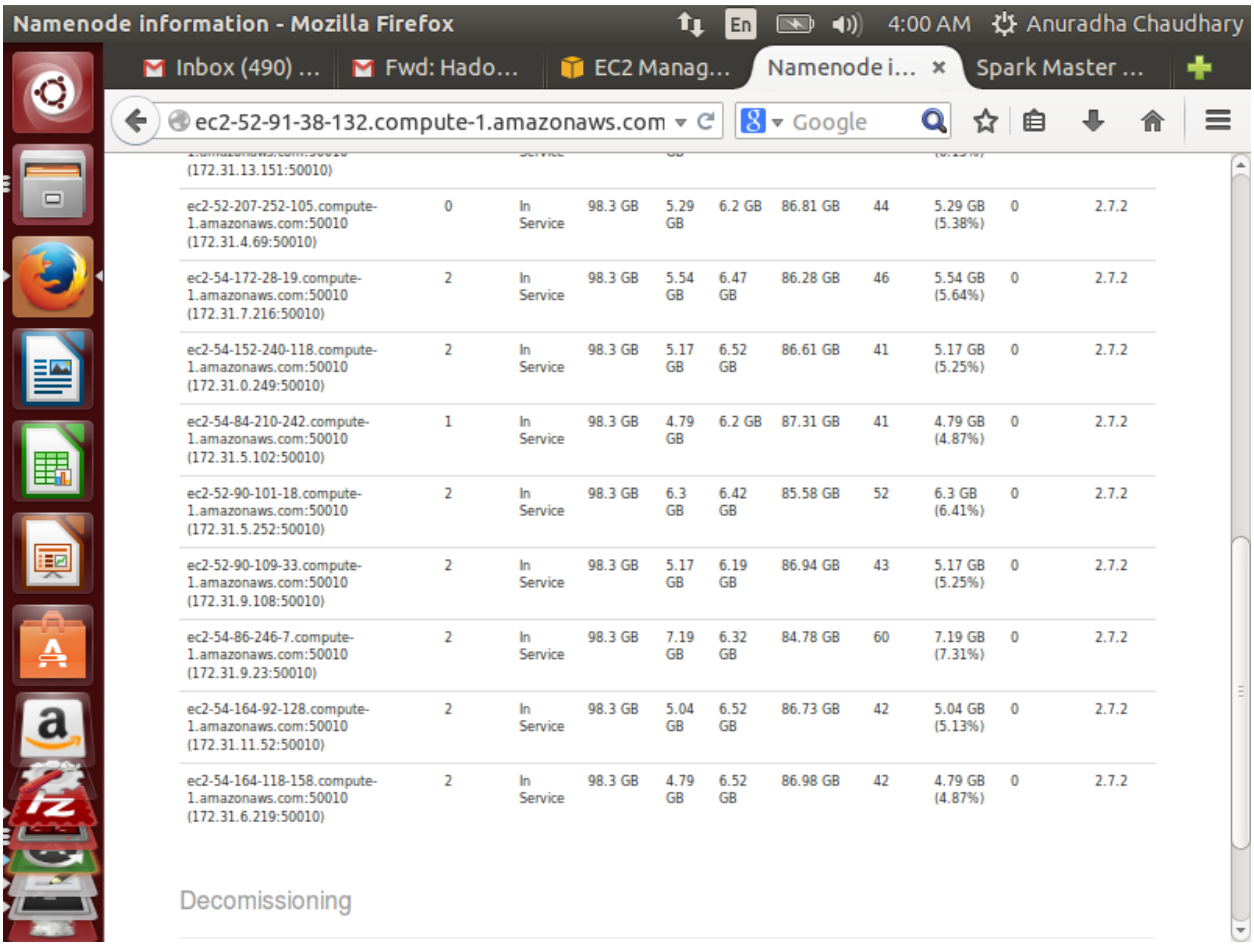


Datanode Information

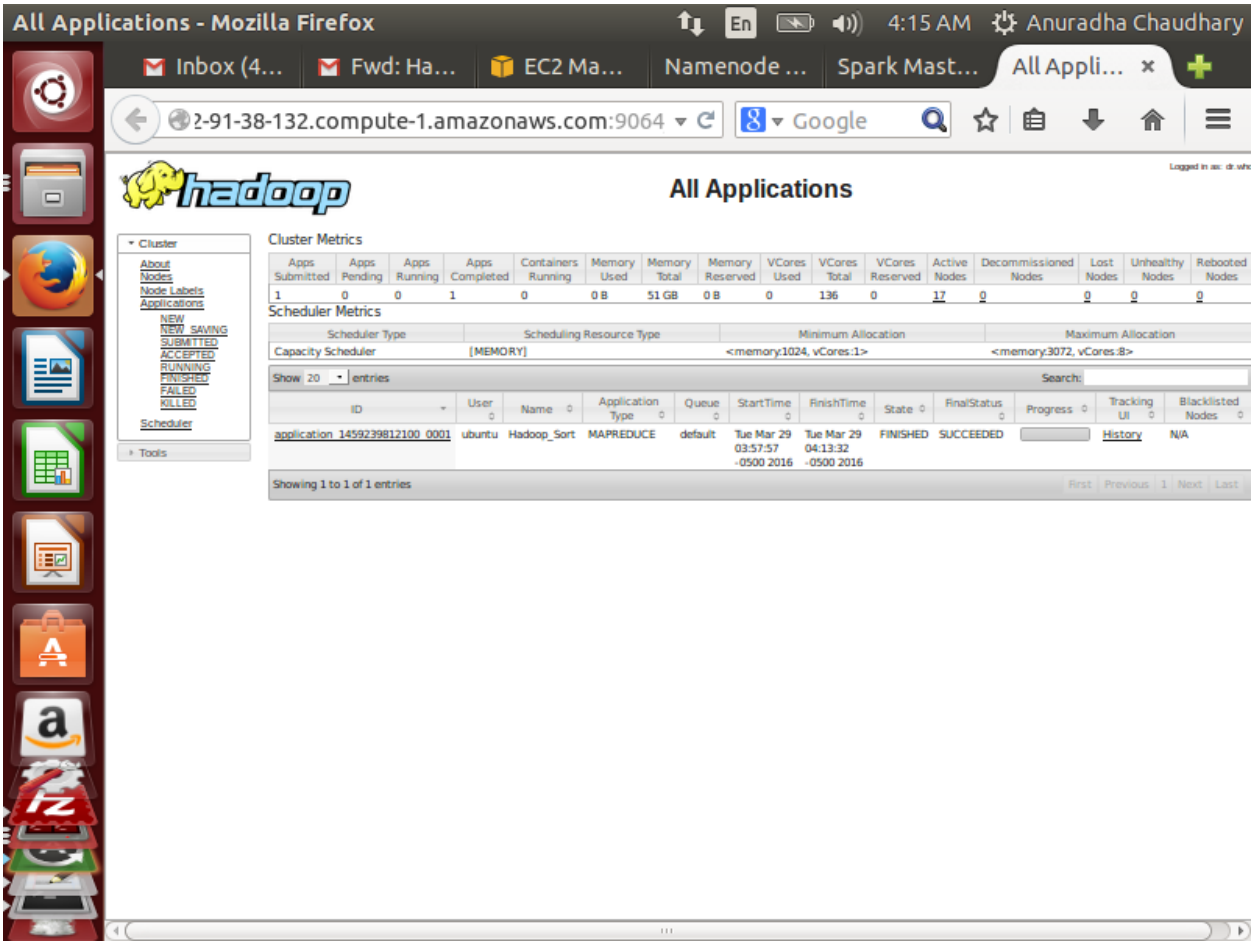
In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ec2-54-84-252-253.compute-1.amazonaws.com:50010 (172.31.6.128:50010)	2	In Service	98.3 GB	6.3 GB	6.43 GB	85.57 GB	52	6.3 GB (6.41%)	0	2.7.2
ec2-52-87-254-99.compute-1.amazonaws.com:50010 (172.31.1.30:50010)	2	In Service	98.3 GB	6.68 GB	6.3 GB	85.32 GB	56	6.68 GB (6.79%)	0	2.7.2
ec2-52-91-112-7.compute-1.amazonaws.com:50010 (172.31.3.113:50010)	0	In Service	98.3 GB	6.3 GB	6.2 GB	85.8 GB	51	6.3 GB (6.41%)	0	2.7.2
ec2-52-201-214-144.compute-1.amazonaws.com:50010 (172.31.13.47:50010)	2	In Service	98.3 GB	6.93 GB	6.2 GB	85.17 GB	57	6.93 GB (7.05%)	0	2.7.2
ec2-52-91-43-195.compute-1.amazonaws.com:50010 (172.31.8.70:50010)	2	In Service	98.3 GB	7.06 GB	6.36 GB	84.88 GB	59	7.06 GB (7.18%)	0	2.7.2
ec2-54-84-223-125.compute-1.amazonaws.com:50010 (172.31.5.186:50010)	0	In Service	98.3 GB	5.29 GB	6.2 GB	86.81 GB	44	5.29 GB (5.38%)	0	2.7.2
ec2-54-84-186-47.compute-1.amazonaws.com:50010	1	In Service	98.3 GB	6.05 GB	6.2 GB	86.05 GB	49	6.05 GB (6.15%)	0	2.7.2

L. Datanode started for 100 GB of data file with 16 nodes.

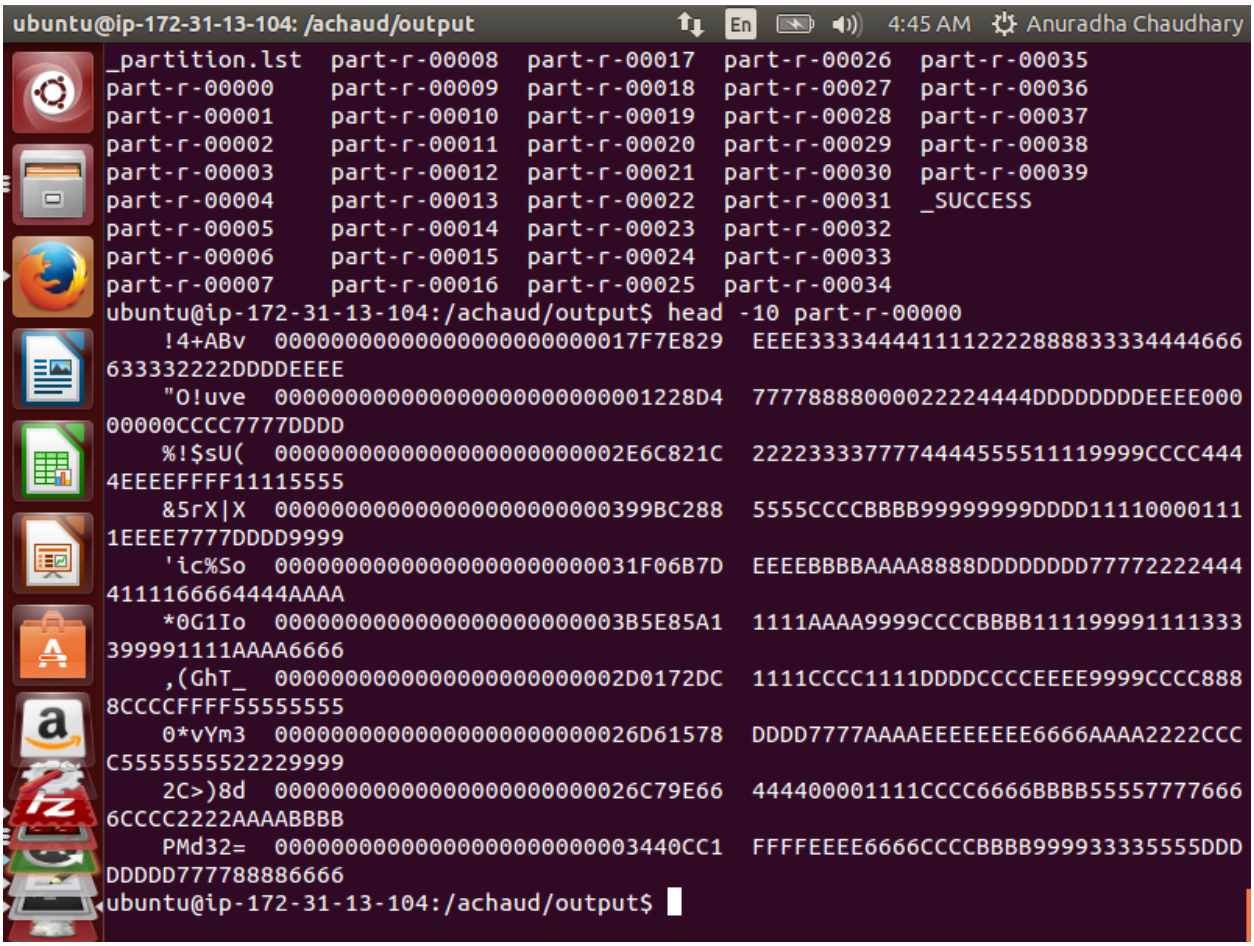


M. Application status on completion of task for 100 GB of data file with 16 nodes.

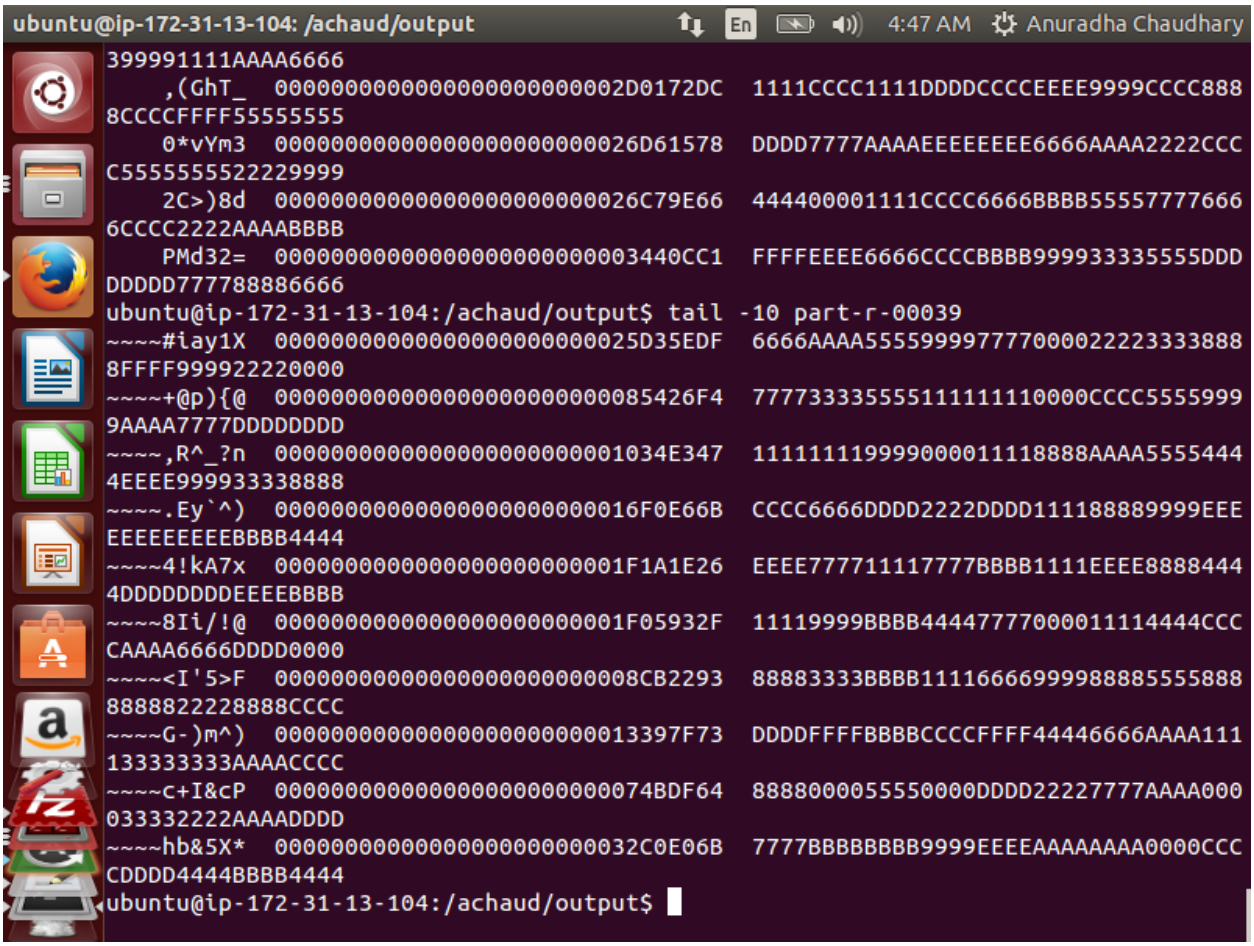


- AS it can be seen from the screen shot above, time taken for sorting 100 GB of data on 16 nodes in around 15 mins which lesser than the time taken for sorting 10 GB of data on 1 node.
- This shows that with more numer of slaves we get better and faster results on Hadoop.

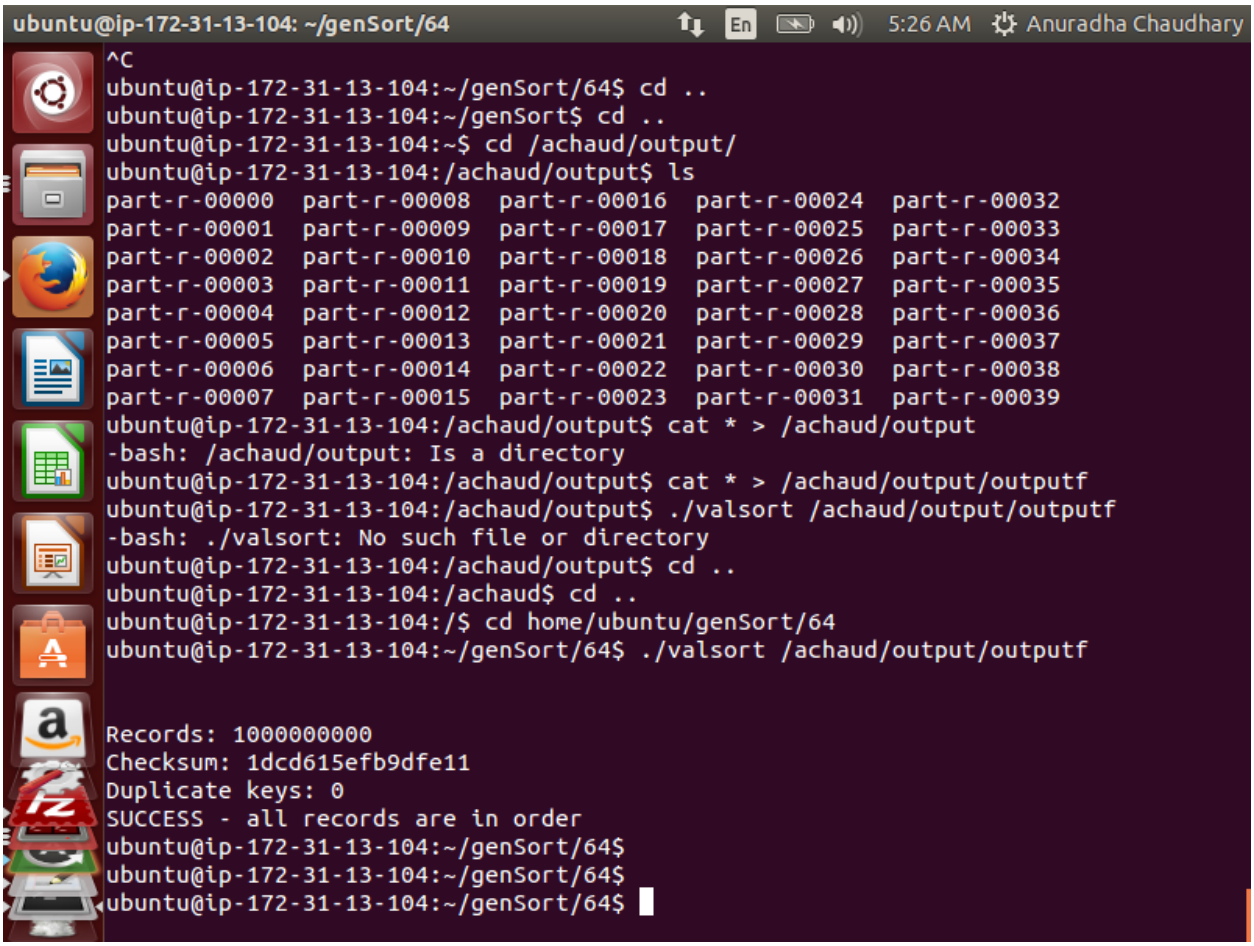
N. First 10 lines of sorted file



O. Last 10 lines of sorted file

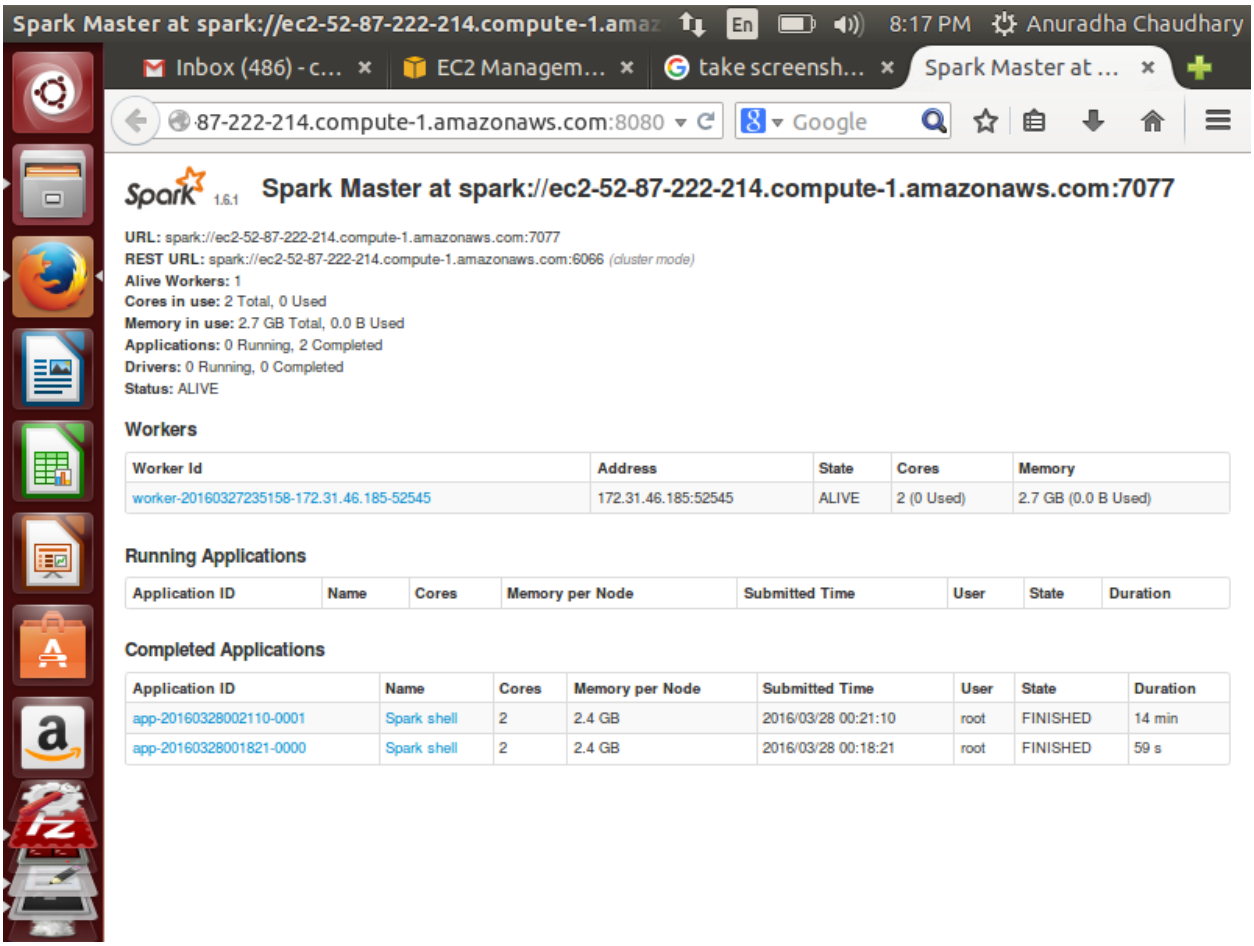


P. Result of valsort for 100Gb sorted file

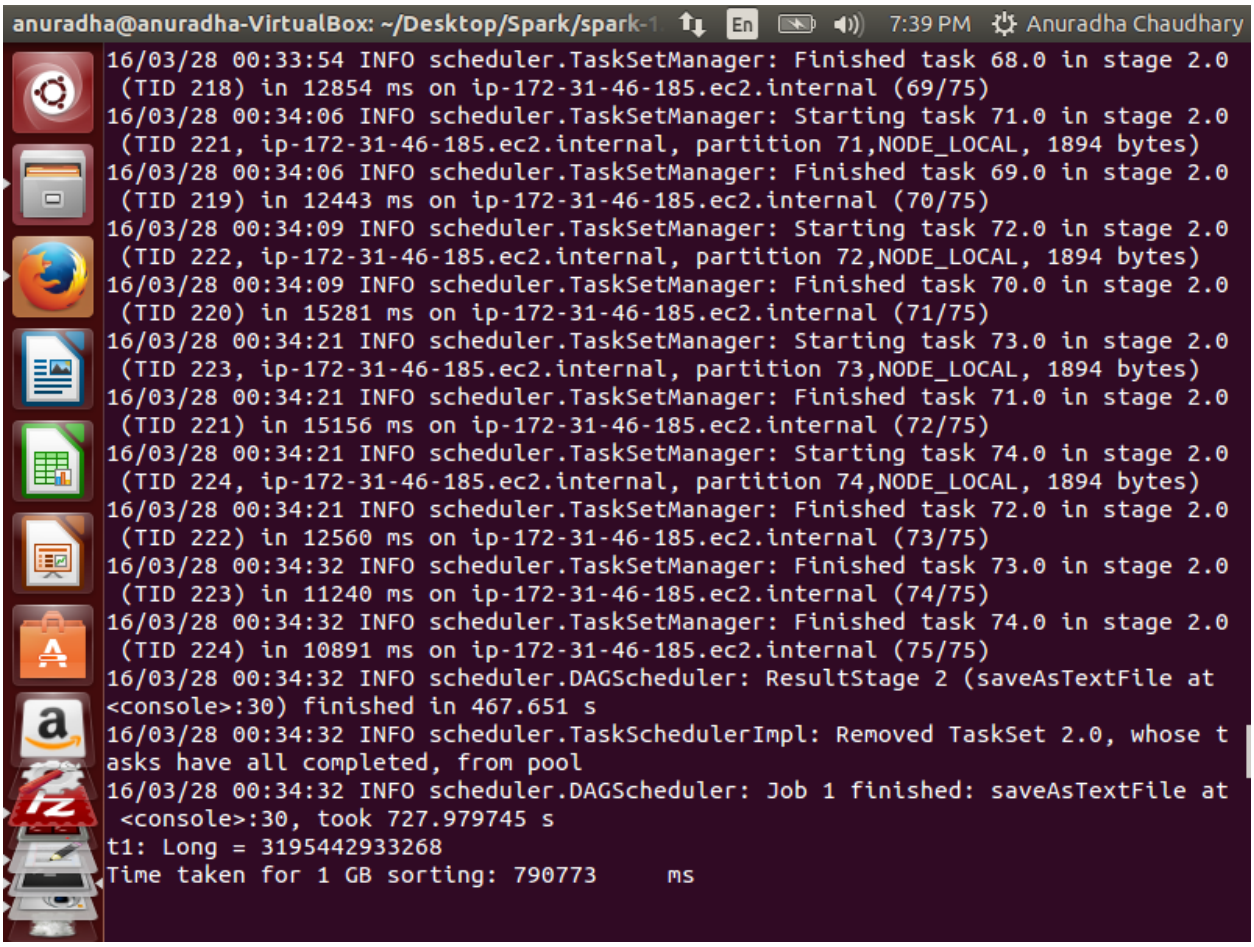


3. SPARK:

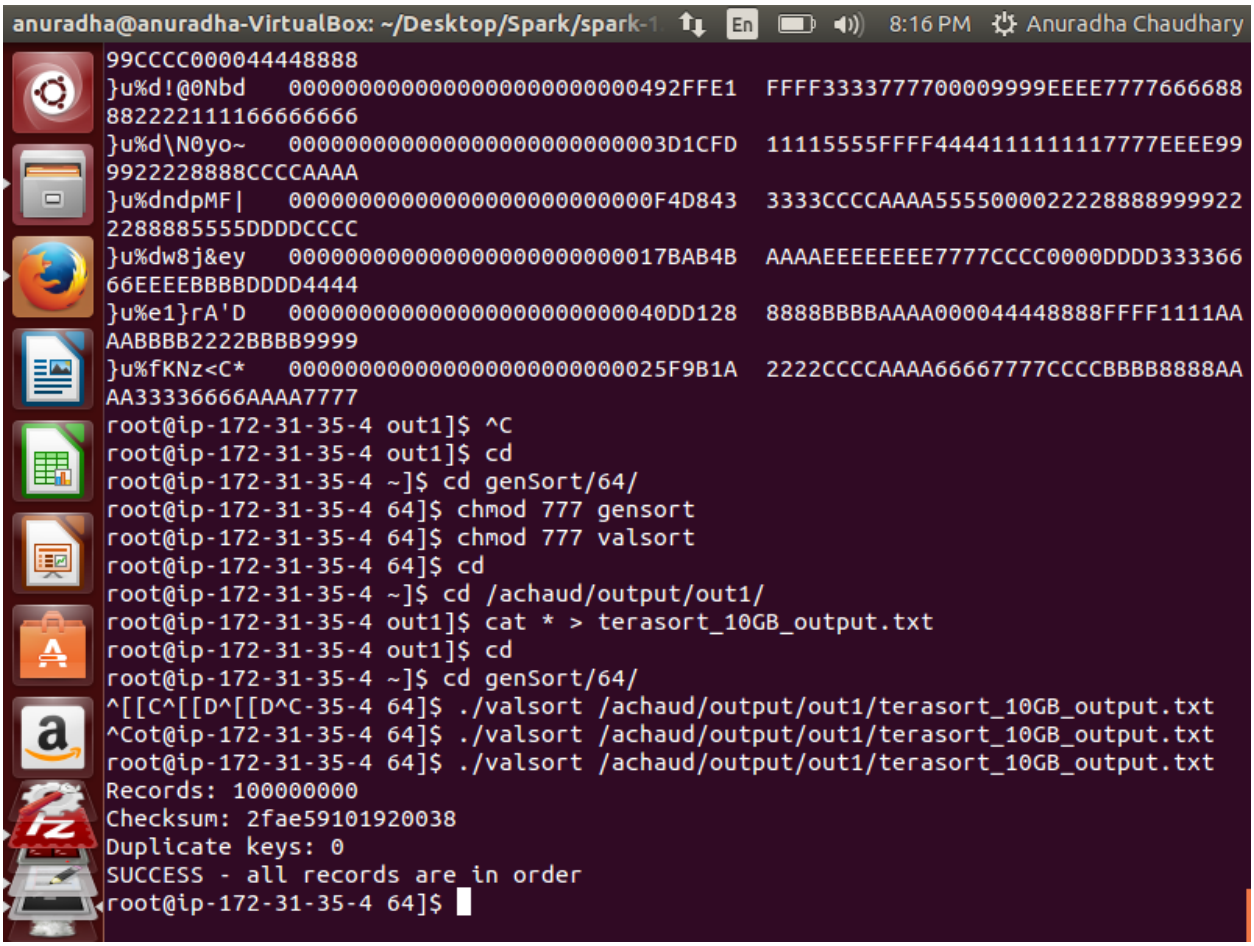
A. Job completion status for sorting 10 GB data file. As it can be seen time taken for sorting 10 GB data by Spark is 14 mins.



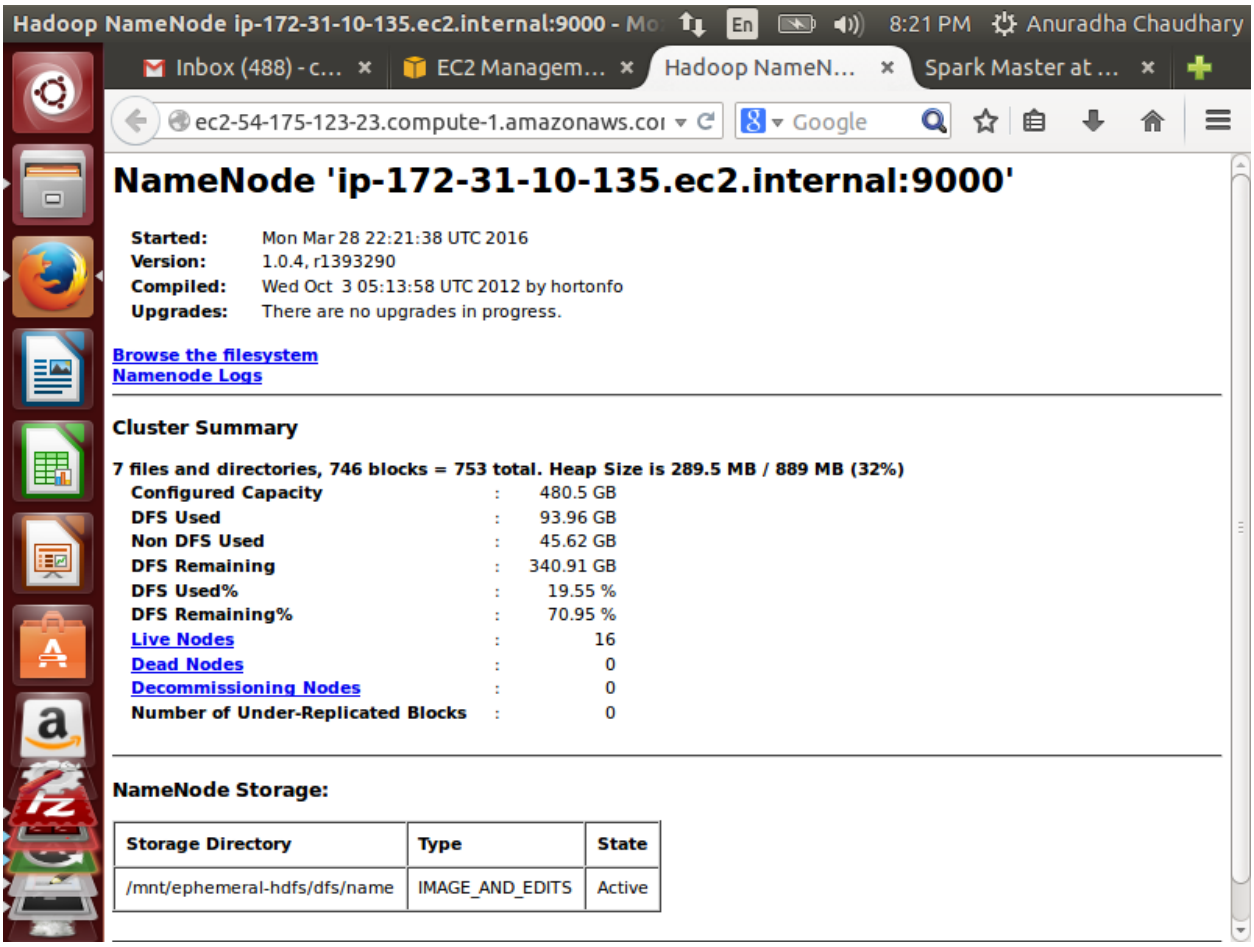
B. Terminal status on completion of sorting 10 GB data file. As it can be seen here as it shows time taken is 14 mins



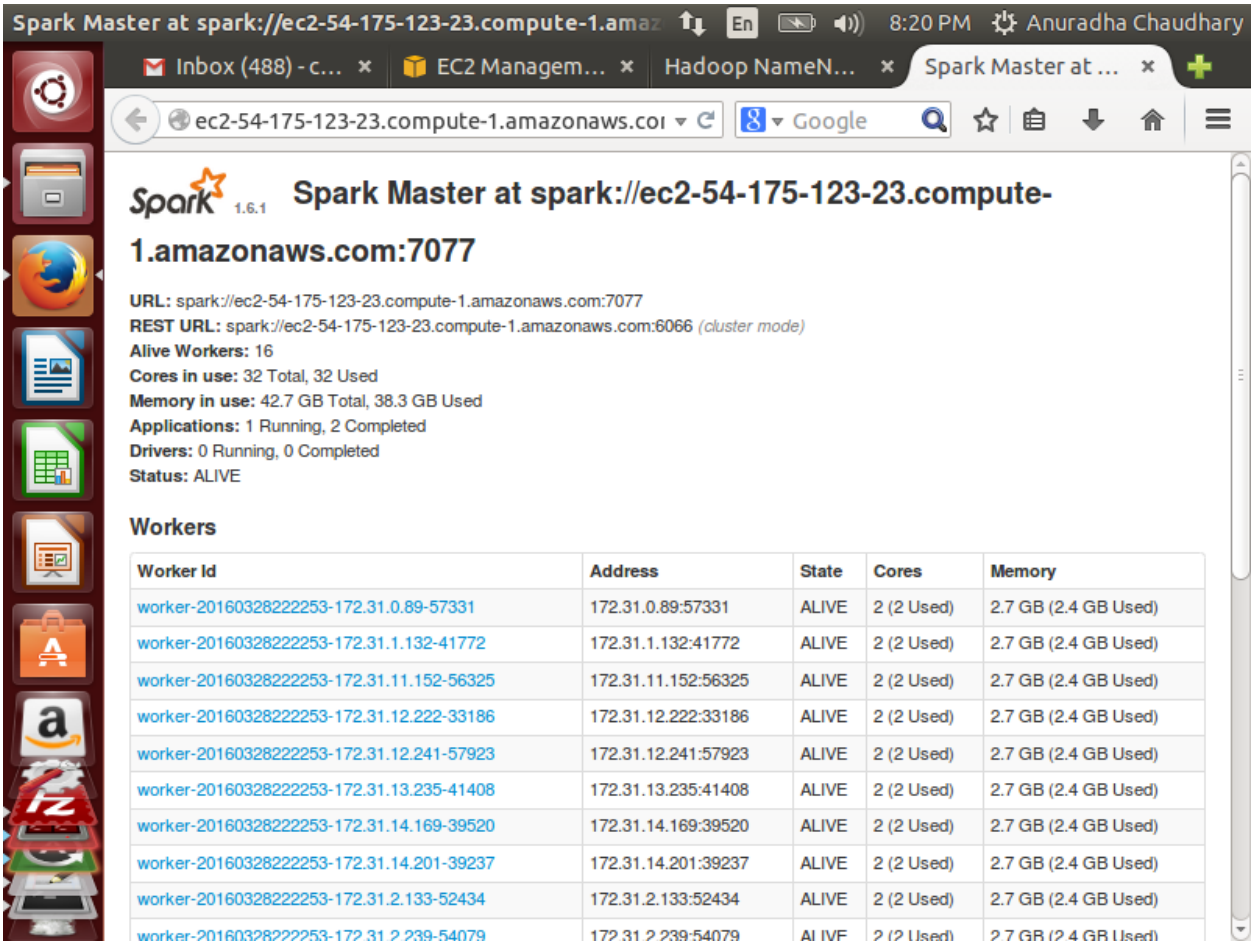
C. Result of valsort for 10Gb sorted file



D. Namenode status on starting sort of 100 GB data on 16 nodes. As it can be seen that 16 nodes are live here.

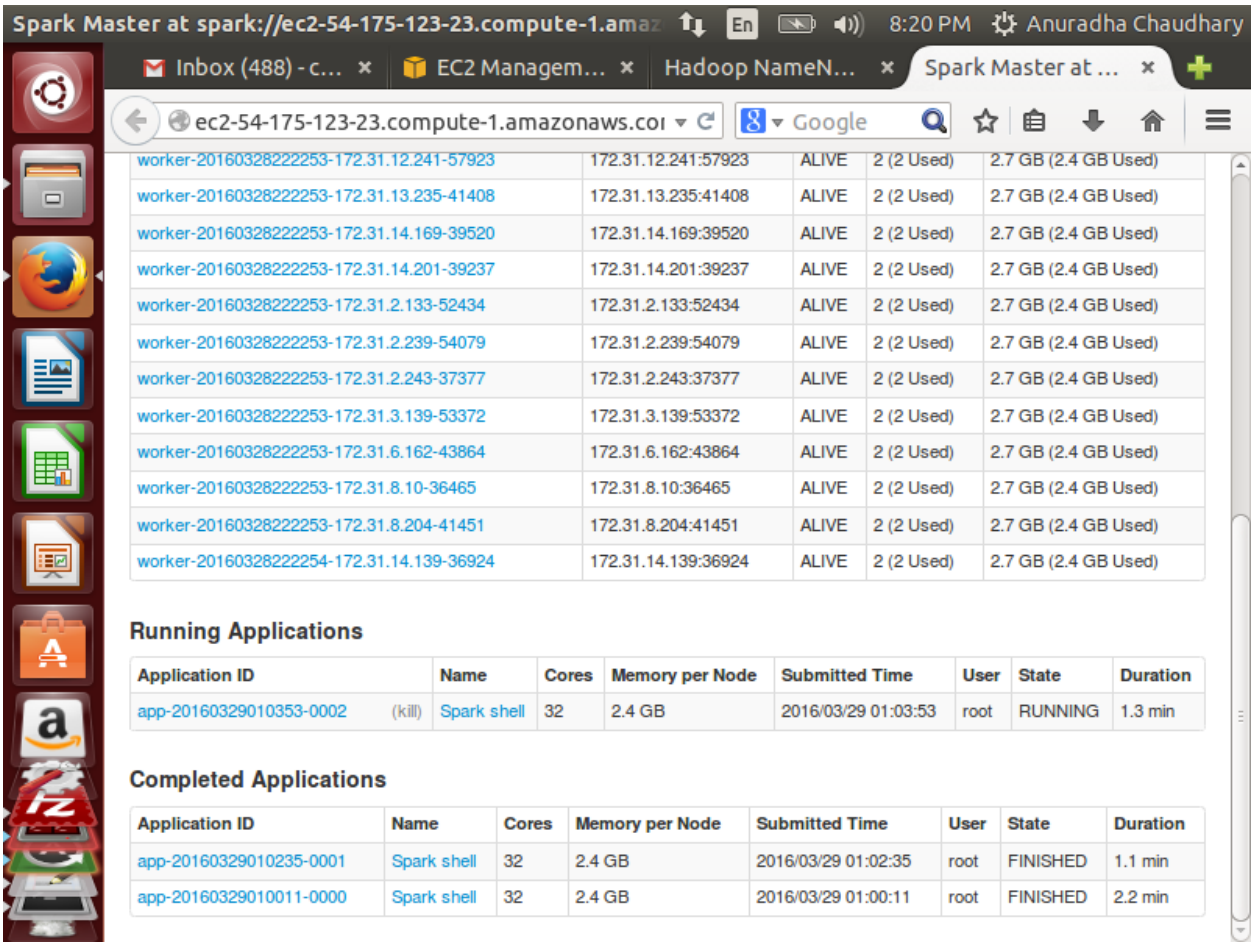


E. On Job completion status of Spark for 16 nodes.

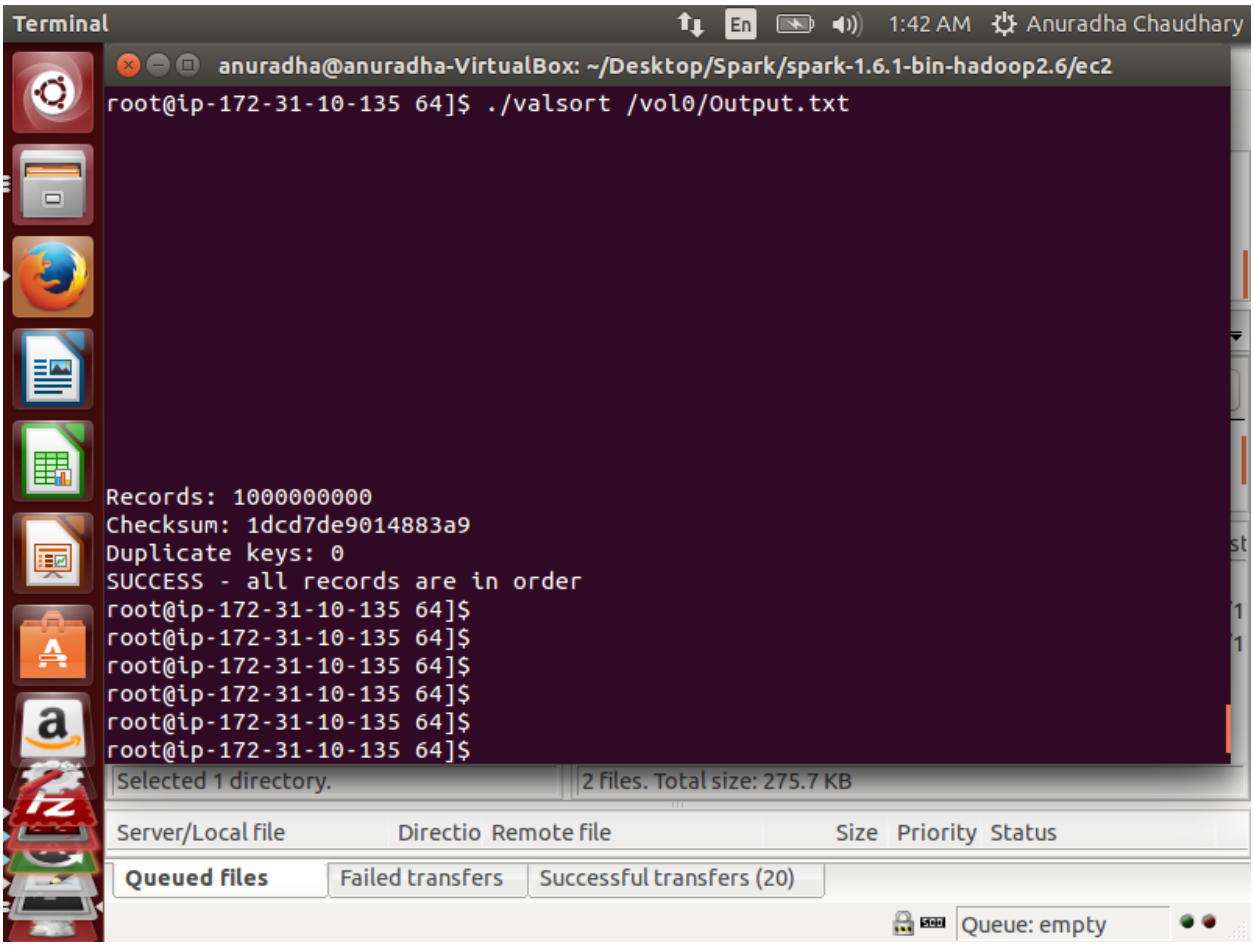


Worker Id	Address	State	Cores	Memory
worker-20160328222253-172.31.0.89-57331	172.31.0.89:57331	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.1.132-41772	172.31.1.132:41772	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.11.152-56325	172.31.11.152:56325	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.12.222-33186	172.31.12.222:33186	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.12.241-57923	172.31.12.241:57923	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.13.235-41408	172.31.13.235:41408	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.14.169-39520	172.31.14.169:39520	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.14.201-39237	172.31.14.201:39237	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.2.133-52434	172.31.2.133:52434	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)
worker-20160328222253-172.31.2.239-54079	172.31.2.239:54079	ALIVE	2 (2 Used)	2.7 GB (2.4 GB Used)

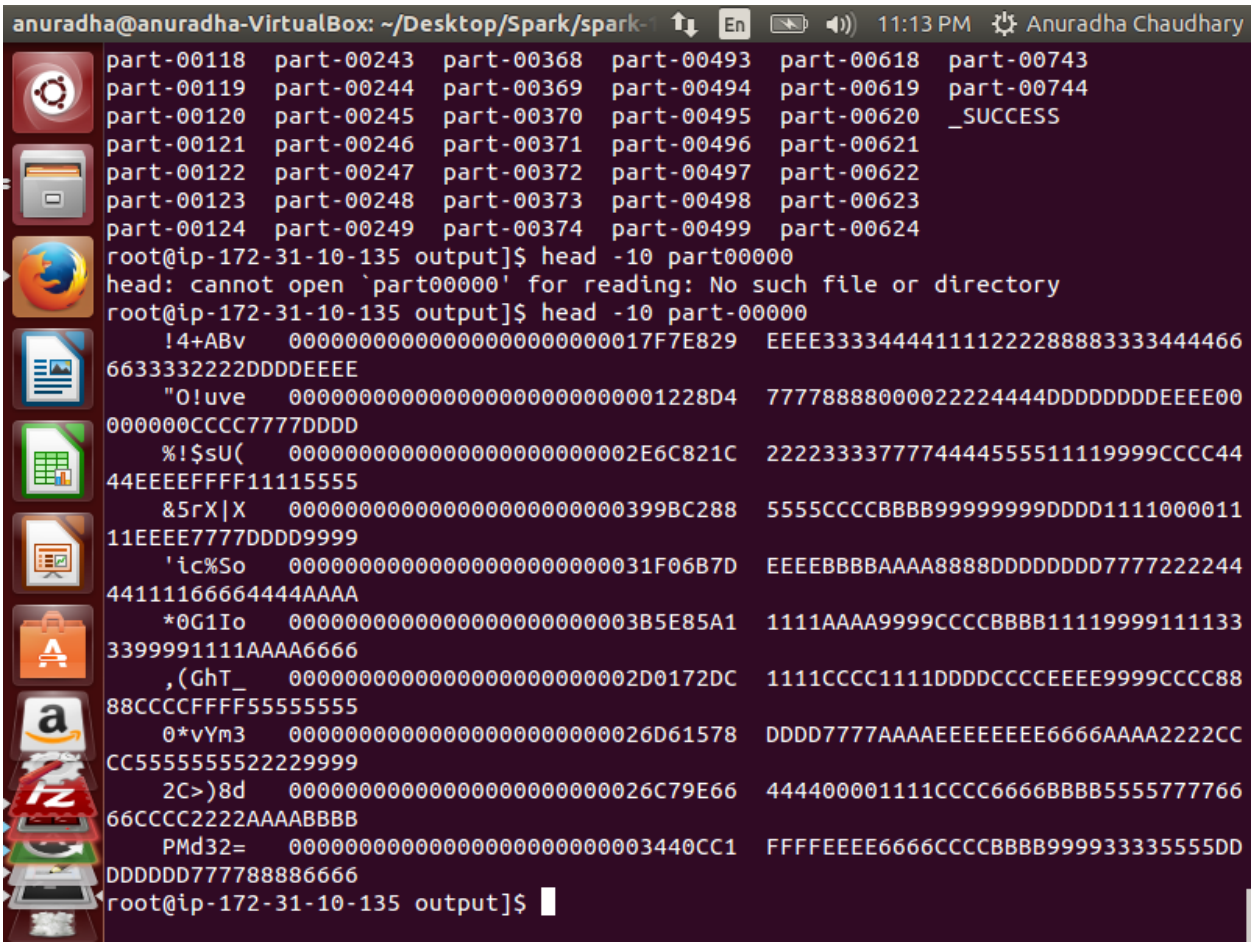
F. On Job completion status of Spark for 16 nodes.



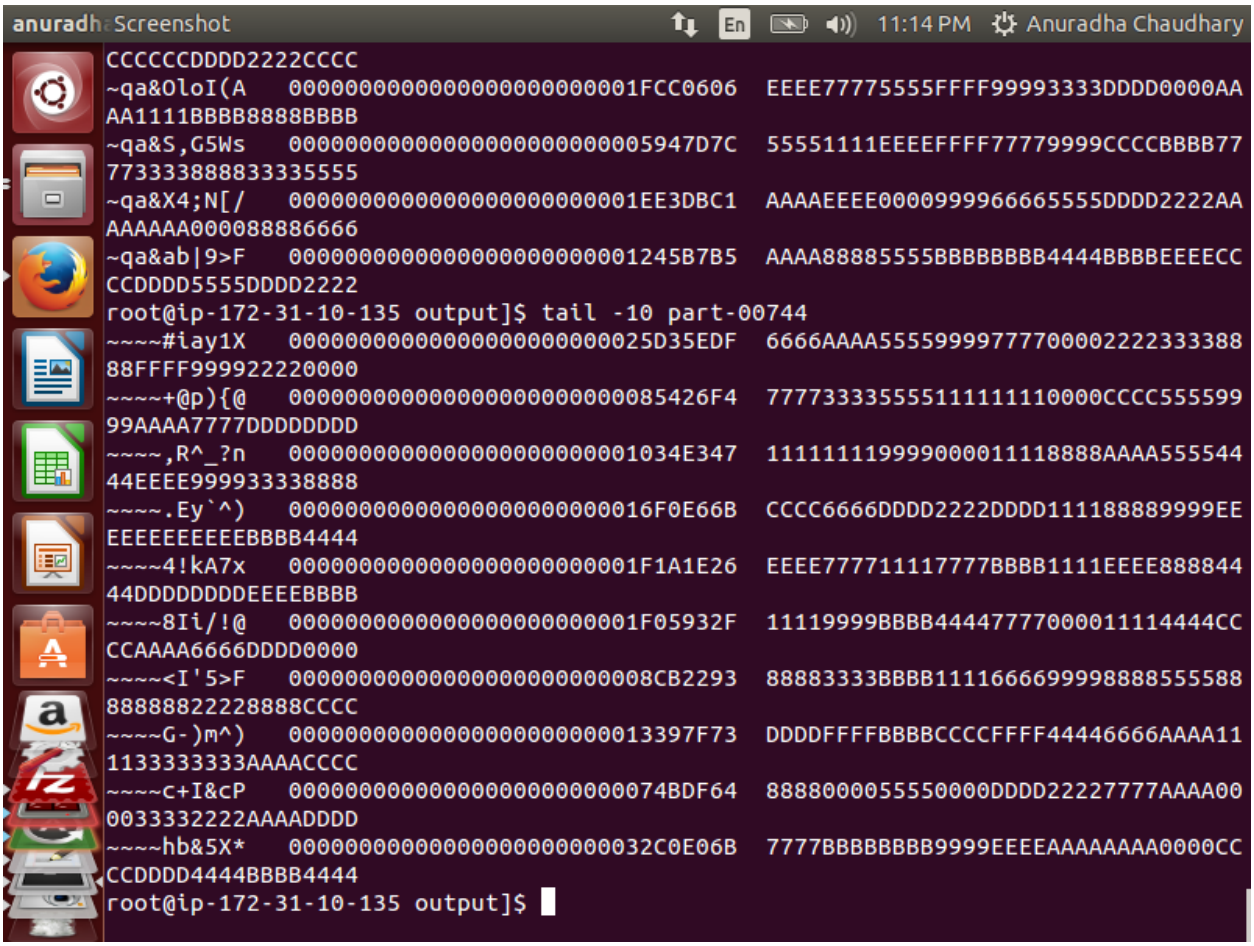
G. Result of valsort for 100Gb sorted file



H. First 10 lines of sorted file

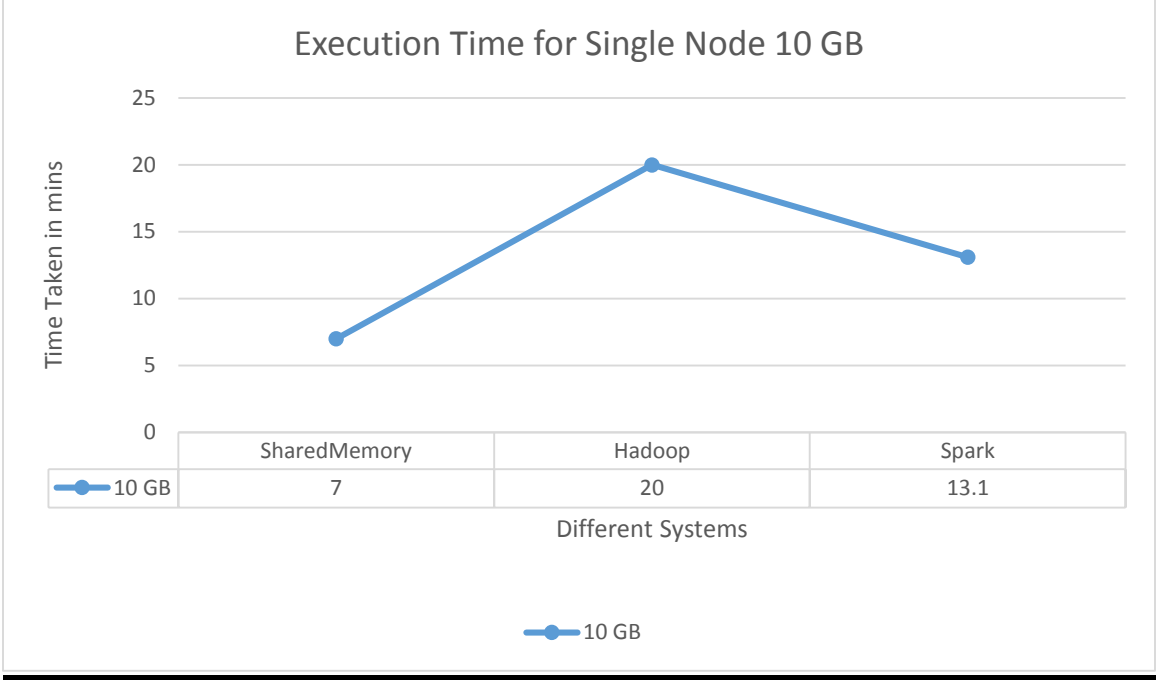


I. Last 10 lines of sorted file



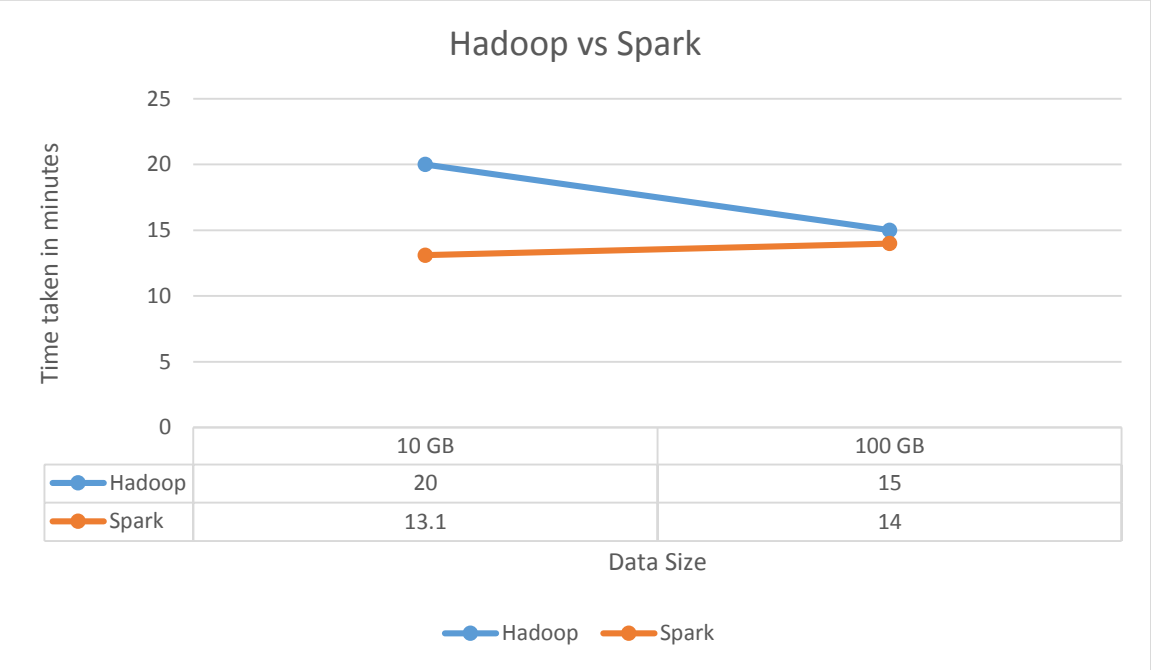
EXECUTION TIME DATA:

- **10 GB, Single Node:**



- Y-axis represents here time taken in mins for sorting.
- X-axis represents here different systems.
- As it can be seen from the graph that least time is taken by shared memory program i.e. 7 mins while Hadoop takes 20 mins and Spark takes 13.1 mins.
- This is because we know, shared memory executes simply on machine using multiple threads and does not require any configuration while Hadoop and Spark create over head for setting up the 1 Node virtual cluster and hence the delay.
- Thus, results make sense as shared memory takes the least time.

- **<u>100 GB, 16 Nodes and 10 GB, 1 Node:</u>**

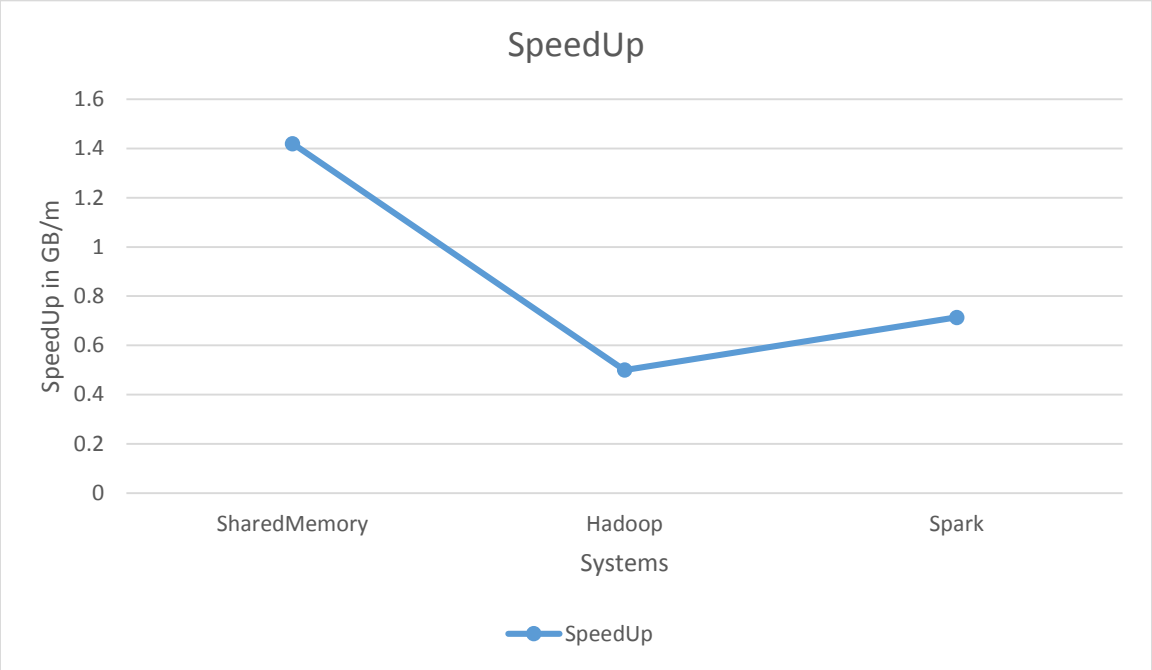


- Graph shows comparison of Hadoop and Spark for 10 GB sorting on 1 Node and 100 GB sorting on 17 Nodes.
- X-axis represents here datasize
- Y-axis represents here time taken in mins for sorting.
- As it can be seen from the graph, time taken by Hadoop for sorting 10 GB data on single is 20 mins while sorting 100 GB data on 17 nodes is 15 mins. We see a decrease in time taken. This is because Hadoop is easily scalable and as it has more number of workers on 17 nodes cluster, work is distributed and hence done efficiently resulting in decrease in time taken.
- Also it can be seen from graph, time taken by Spark for sorting 10GB data on single node is 13.1 mins while sorting 100 GB data on 17 nodes is 14 mins. Here we see time taken for 100 GB on 17 nodes and 10 GB on single node remains almost the same. This is because Spark reuses intermediate generated data and so even after X number of iterations the time taken will remain the same.
- On comparing Hadoop and Spark we know Spark is super lightweight analytic tool. Spark is faster than Hadoop as Spark was designed in order to overcome the drawbacks of Hadoop.This is because Spark uses RDDs underneath and keeps data in memory which fastens the sorts. There is no such notion of keeping data in memory in Hadoop.

- From graph above we can verify that Spark is faster than Hadoop.

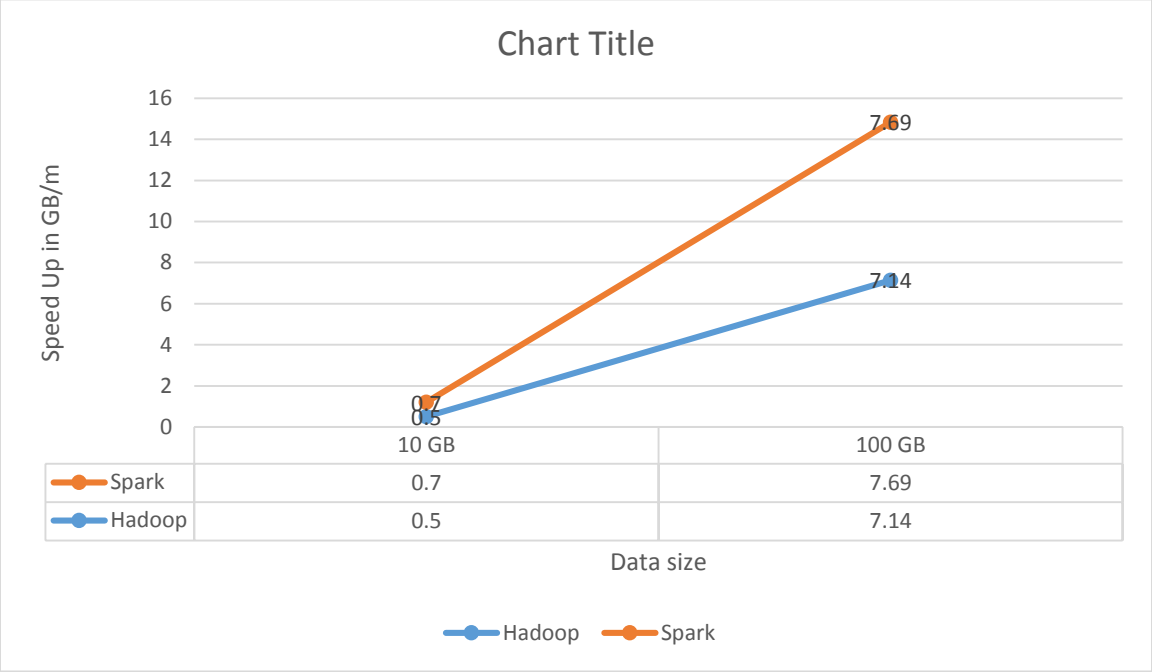
SPEEDUP:

- **10 GB 1 Node:**



- We know speed up is Data_Size/Time_Taken.
- Y-axis in graph shows Speed Up in GB/mins.
- X-axis in graph shows different systems used.
- As time taken by shared memory was least for 100 GB sorting amongst Hadoop, Spark and Shared Memory, Speed Up of Shared Memory is most i.e 1.42.
- Second comes spark which is faster than Hadoop but slower than shared memory for 1 Node i.e. 0.7.
- Last comes Hadoop with speedup of 0.5.
- Hence, the results shown here in graph makes sense.

- **<u>10 GB 1 Node vs 100 GB 16 Nodes:</u>**



- X-axis here represents data size
- Y-axis here represents speed up in GB/mins.
- We know, speed up is data size/time taken.
- As from graph it can be seen that speed Up of spark increases linearly and very fast compared to Hadoop.
- This is because the time taken by Hadoop increases with number of iterations while it remains nearly same for spark.
- Speed Up of Spark is more compared to that of Hadoop.

From these experiments following conclusions can be drawn:

- At 1 node scale, shared memory seems best as it the fastest amongst 3 for single node.
- At16 nodes, either Hadoop or Spark can be used as time will be nearly same for both.
- At 100 nodes, as Spark is faster than Hadoop as it uses RDDs which keeps data in memory, spark should be used.
- At 1000 nodes scale too Spark should be used as it is faster than Hadoop and is very light weight.