# CS553: Cloud Computing

# Programming Assignment 2: Sorting Application

# Assignment Report

**cwid: A20327606**

**INDEX**

# Shared Memory

## 1.1 Introduction

**Problem Statement:**

The goal of this experiment is to sort a large dataset which contains one record per line of 100 bytes and we need to sort them on the basis of ASCII values of first 10 bytes. The major challenge here is the memory size. Since we have a very less memory which is much smaller value than the total file size.

**Methodology:**

The logic here is to divide the total number of records into small chunks which can fit and be sorted within memory space. Thus I have implemented external merge sort which first divides the size into smaller chunks, then individually sorts it and finally merges all the chunks. For sorting the individual chunks, I have used quick sort.

**Environment details:**

| | |
|---|---|
| Type of EC2 instance used: | c3.large |
| Region: | N. Virginia |
| OS: | Ubuntu Server 14.04 LTS (HVM), SSD Volume Type – ami-fce3c696 |
| Implementation Language: | JAVA |
| JAVA version | java version "1.7.0_95"<br>OpenJDK Runtime Environment (IcedTea 2.6.4) (7u95-2.6.4-0ubuntu0.14.04.1)<br>OpenJDK 64-Bit Server VM (build 24.95-b01, mixed mode) |

Installation and experiment running steps:

1. Connect to AWS instance through ssh terminal.
2. check the names of disks using “lslbk” command (xvdb,xvdc and xvdd)
3. mount the disks on a folder using the script mount.sh
4. transfer the files (java files, gensort and valsort) to the mounted location
5. run the gensort to generate input file of required size using following command:

    ***./gensort -a num_of_records input.txt***

   where, num_of_records = 10000000 for 1GB
   = 100000000 for 10 GB
   = 10000000000 for 1TB
6. now run the script SharedMemory.sh which would install java, compile the code, run it and then save the output to log.txt

7. (optional) The code can also be run without using script by directly running the following commands:

***javac FileSorter.java***
***java -Xmx3G FileSorter***

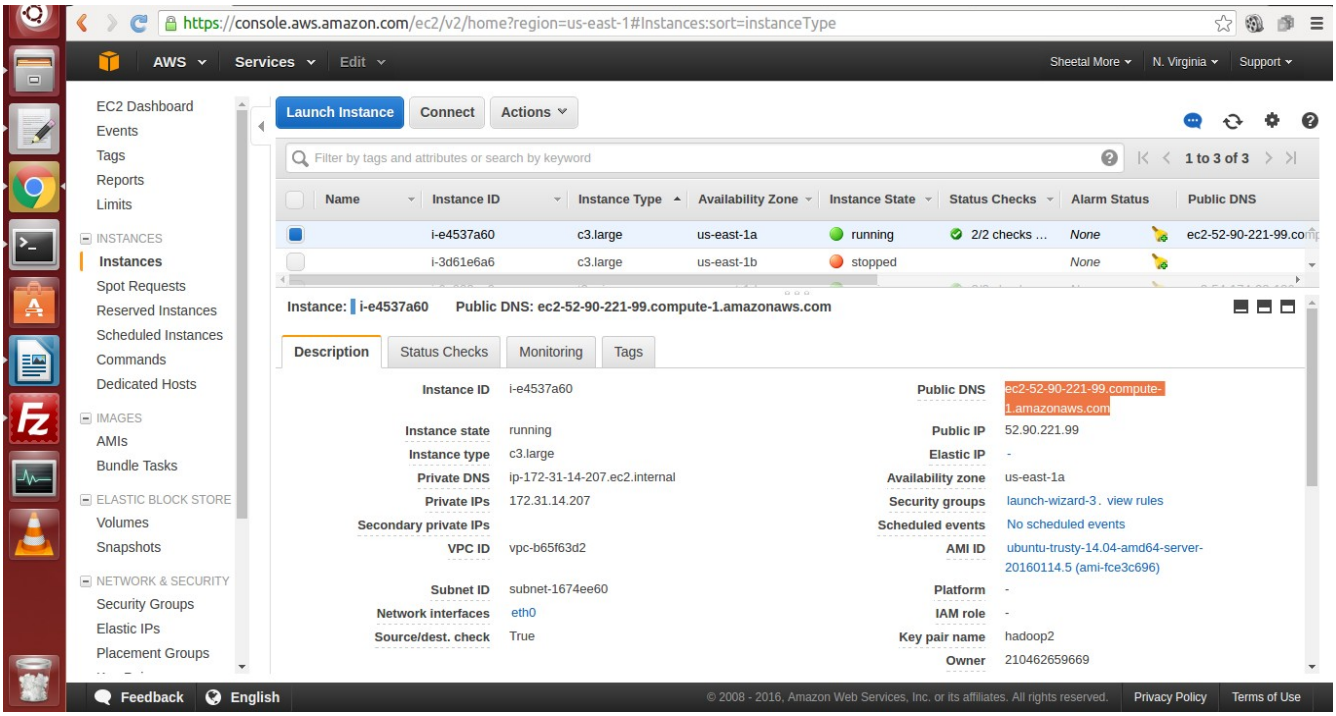
8. Now the output file is generated in the same folder which contains sorted records
9. in order to verify the code, run the valsort using command:

./valsort output.txt

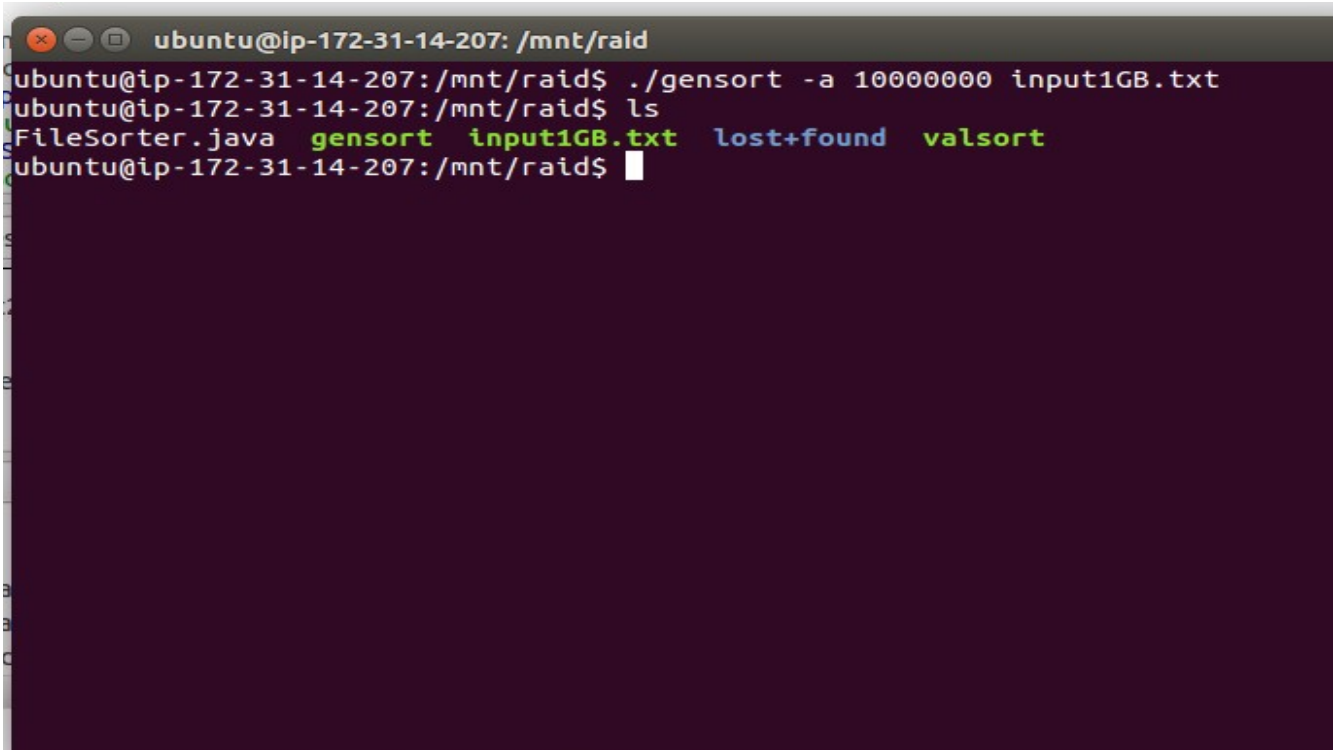
10. The head and tail of the output file can be found using head and tail command as shown in screenshots below:

## 1.2 Experiment 1 with 1GB Dataset

1. Aws with ipaddress:

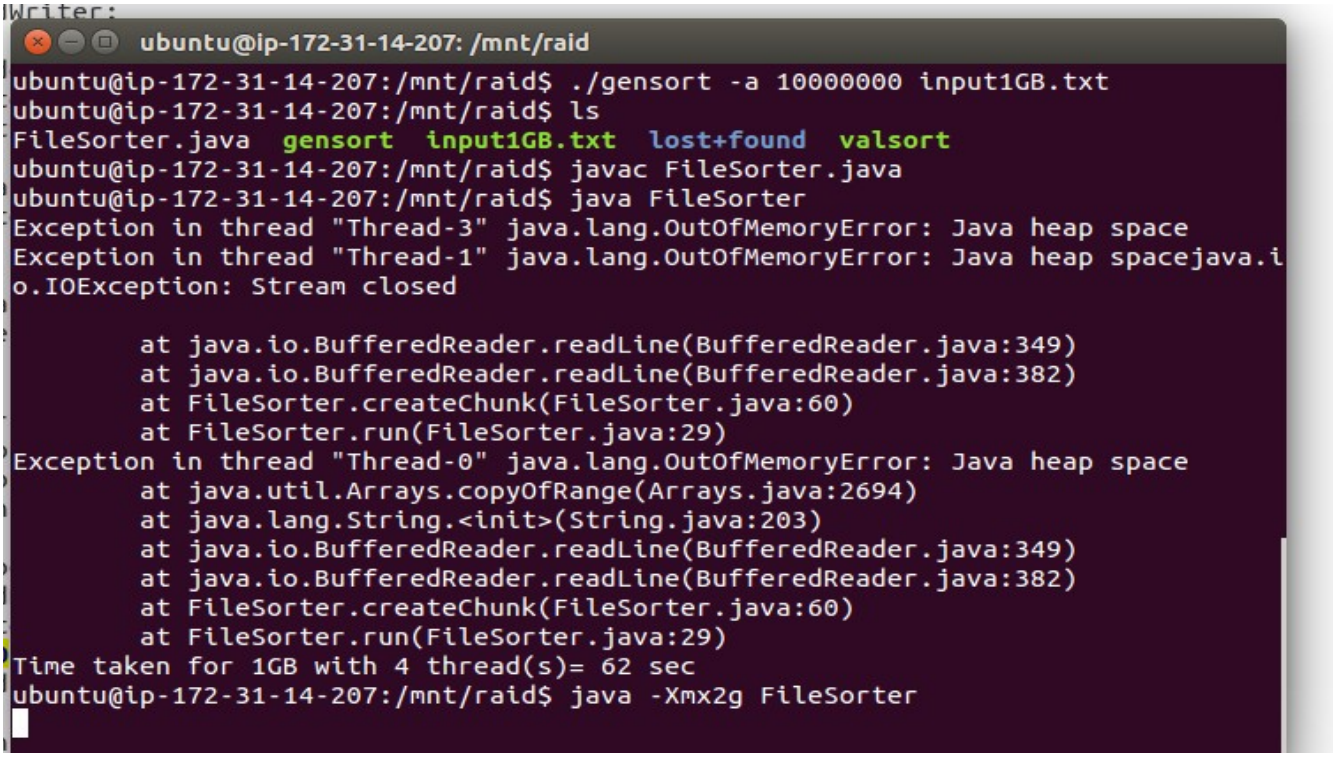



2. gensort




3. running java program:

a) 4 threads: error screenshot for out of memort heap space

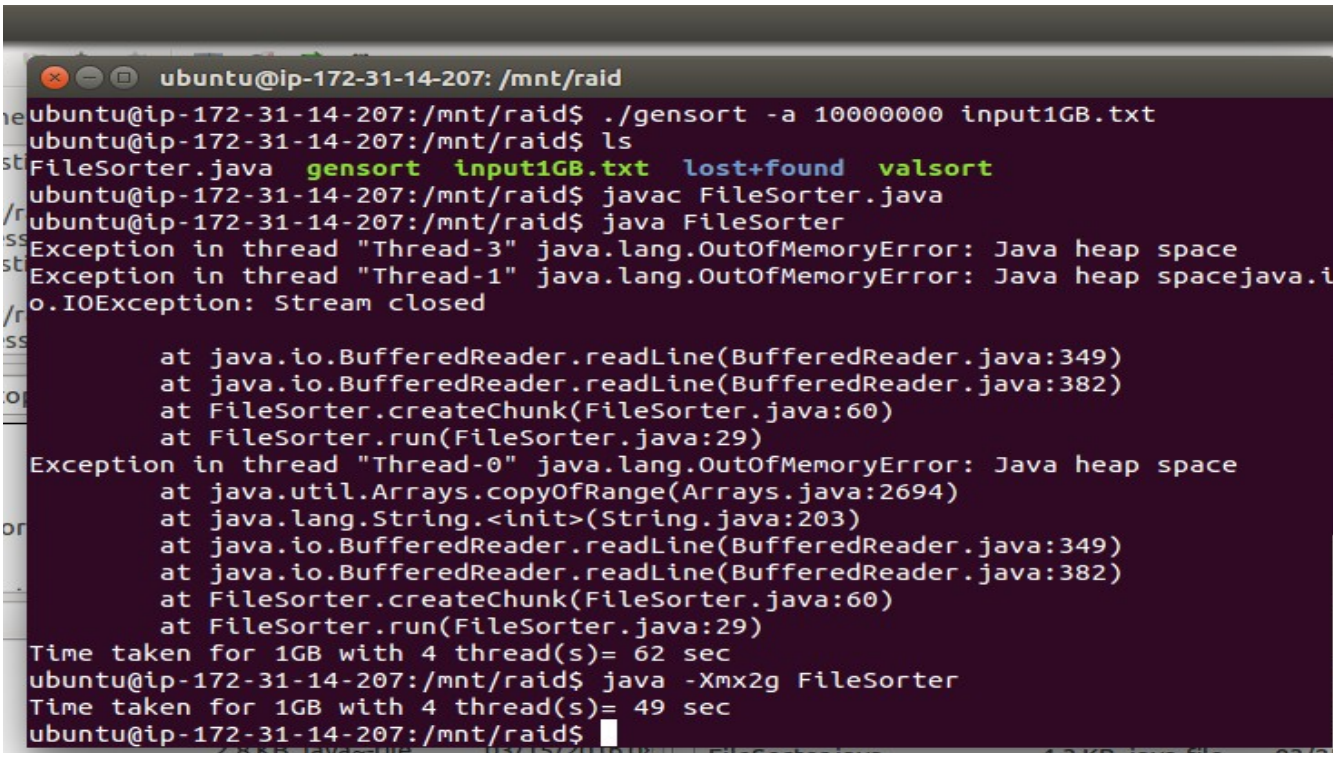


As seen in above document, I got an error due to heap space which I then solved by increasing the heap size. The command used is as follows:

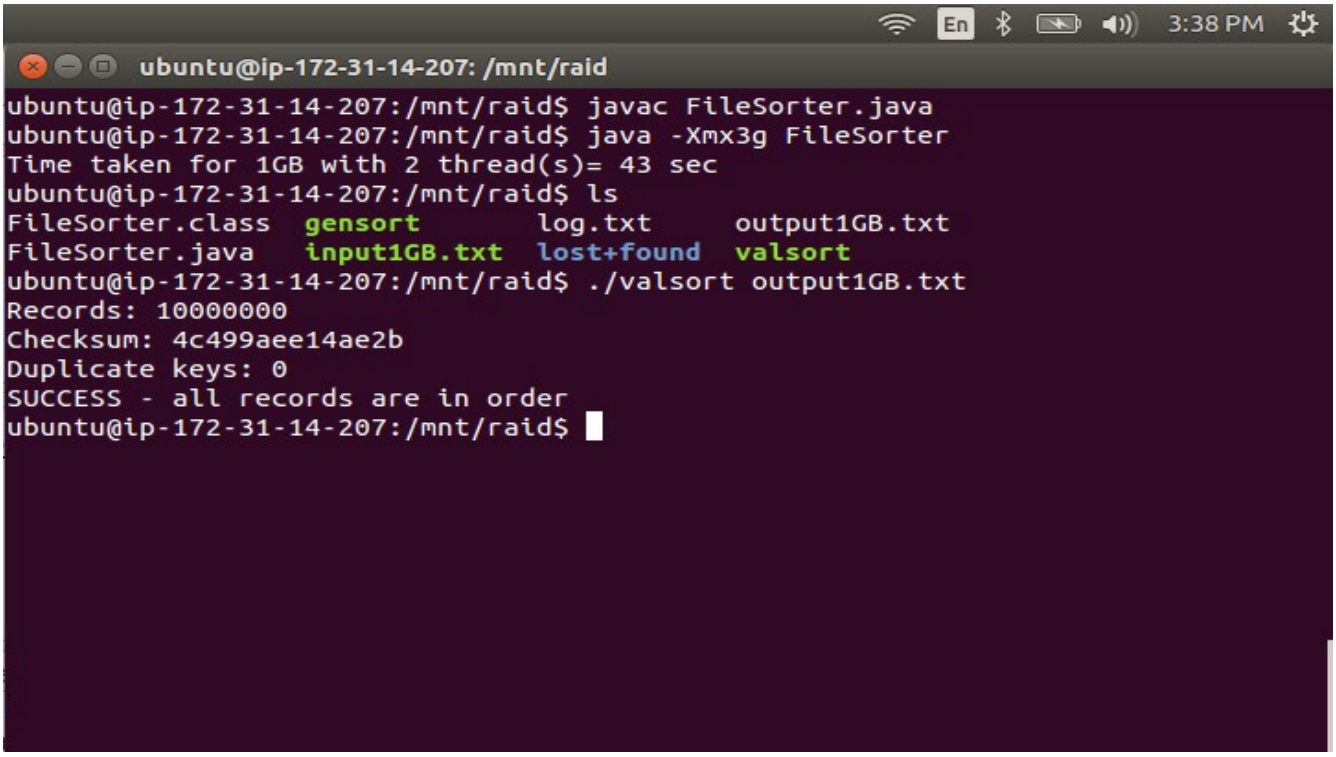
java -Xmx2g FileSorter

Thus allocating 2GB RAM which solved the problem and I got the output as follows:

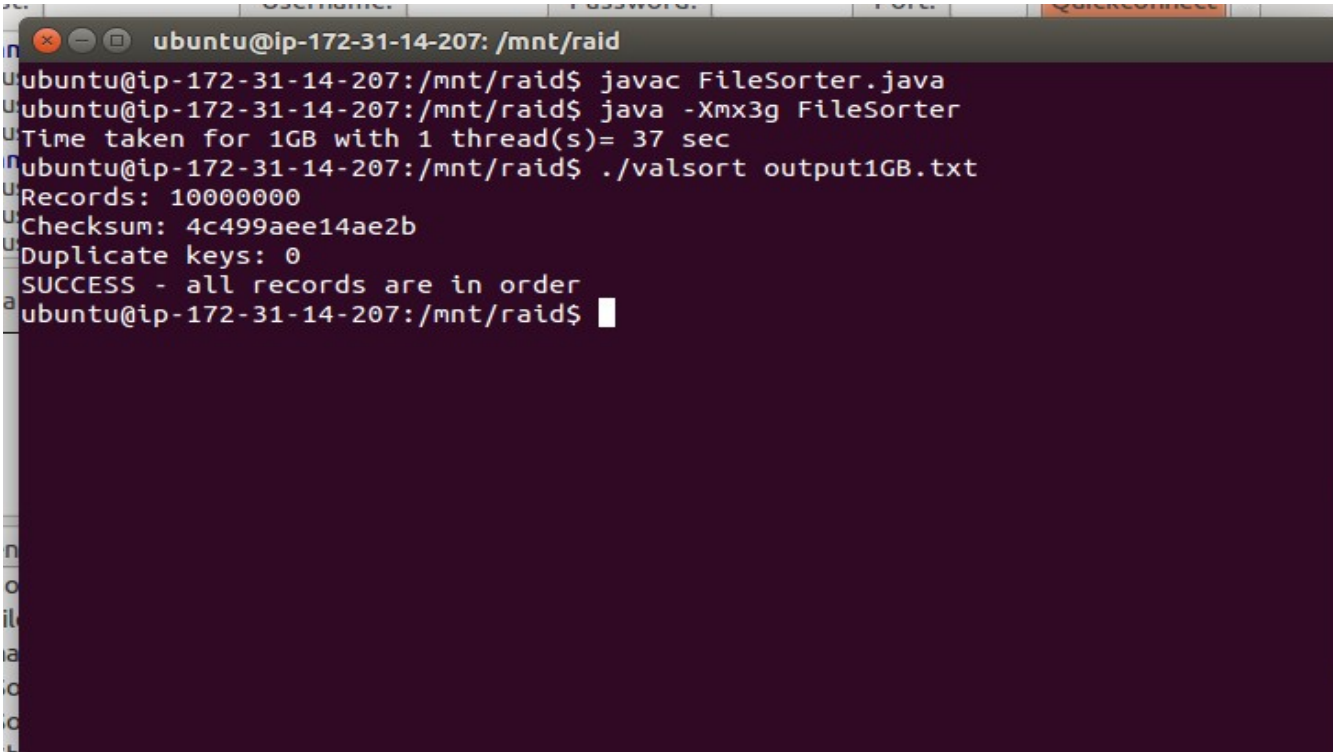
Actual time: 49 secs



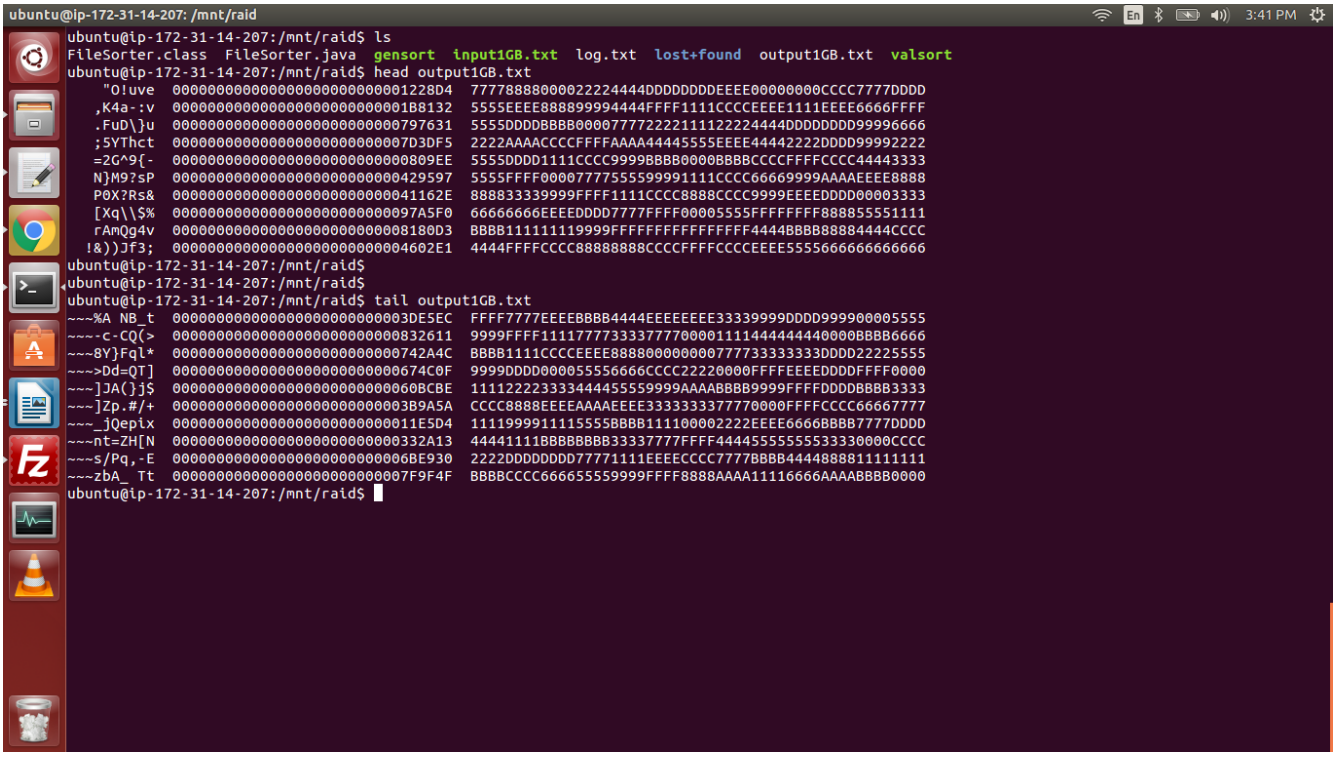

b) 2 threads:




3. 1 thread:

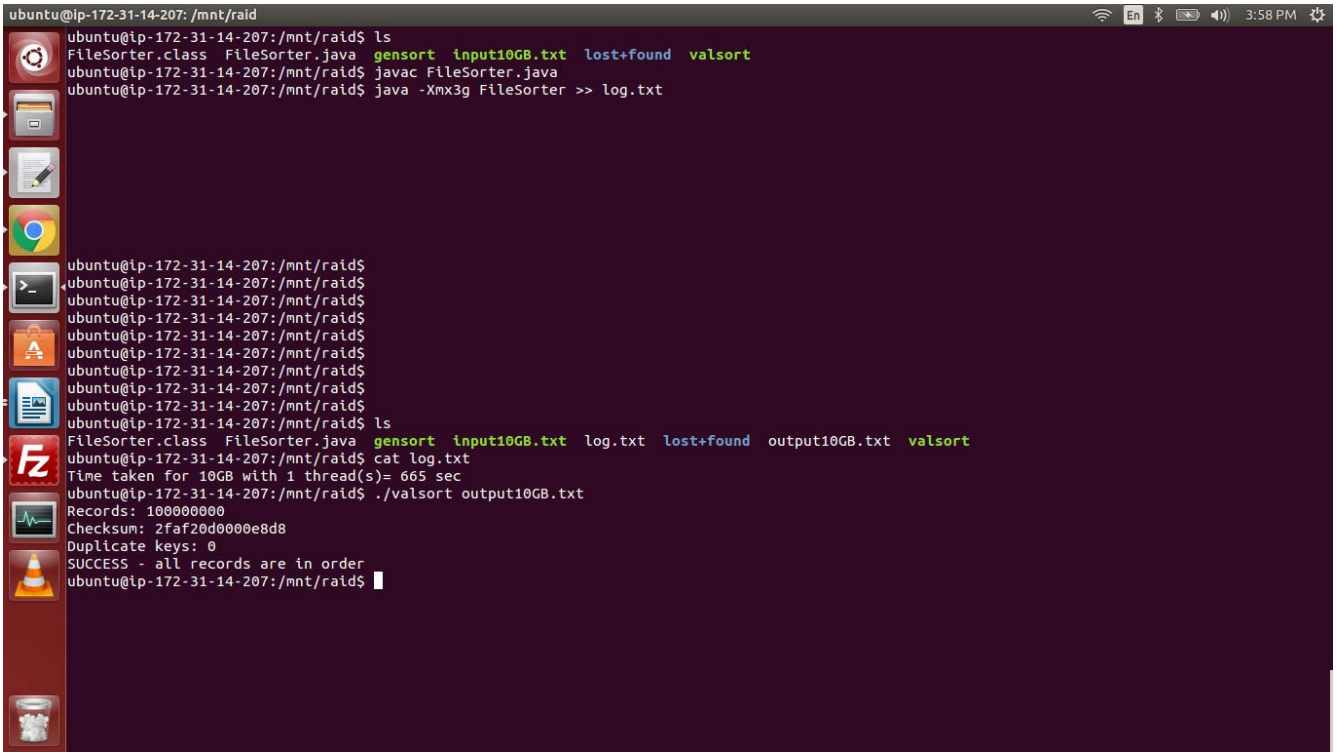



head and tail

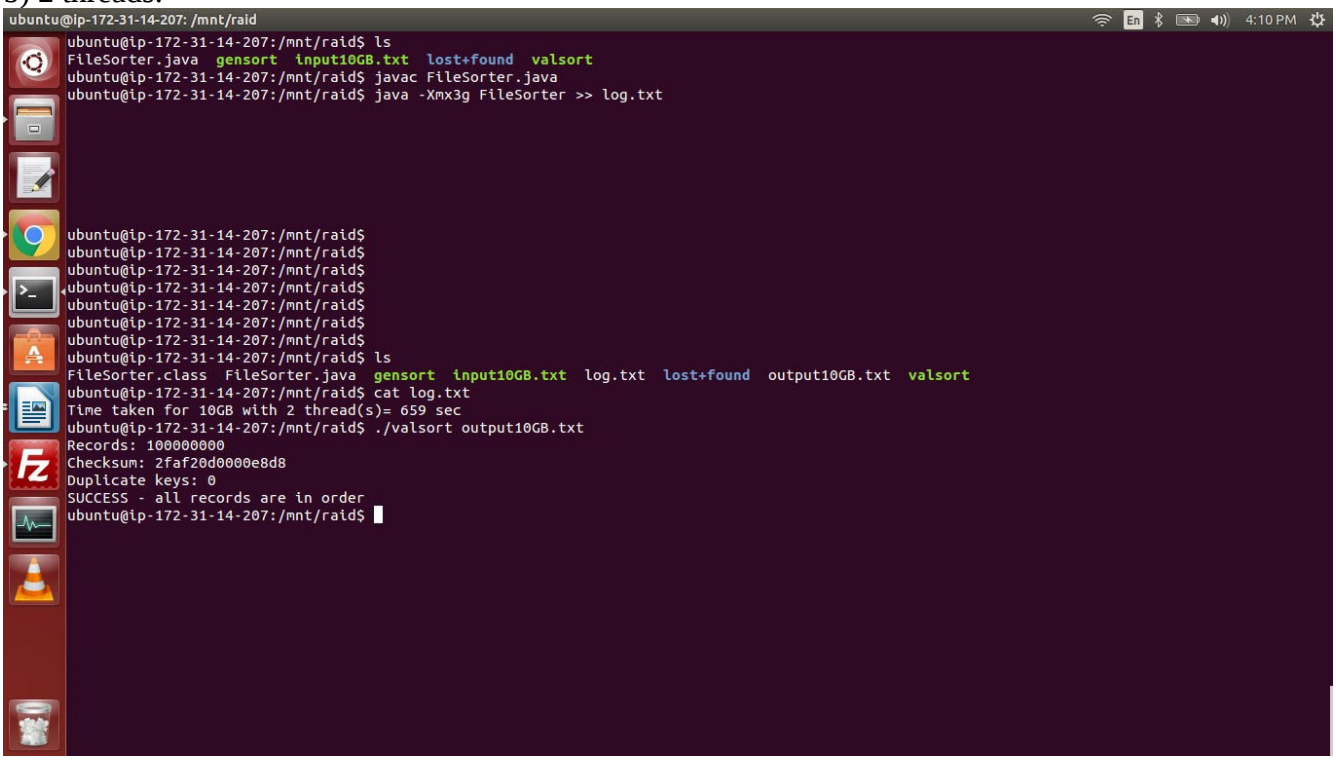


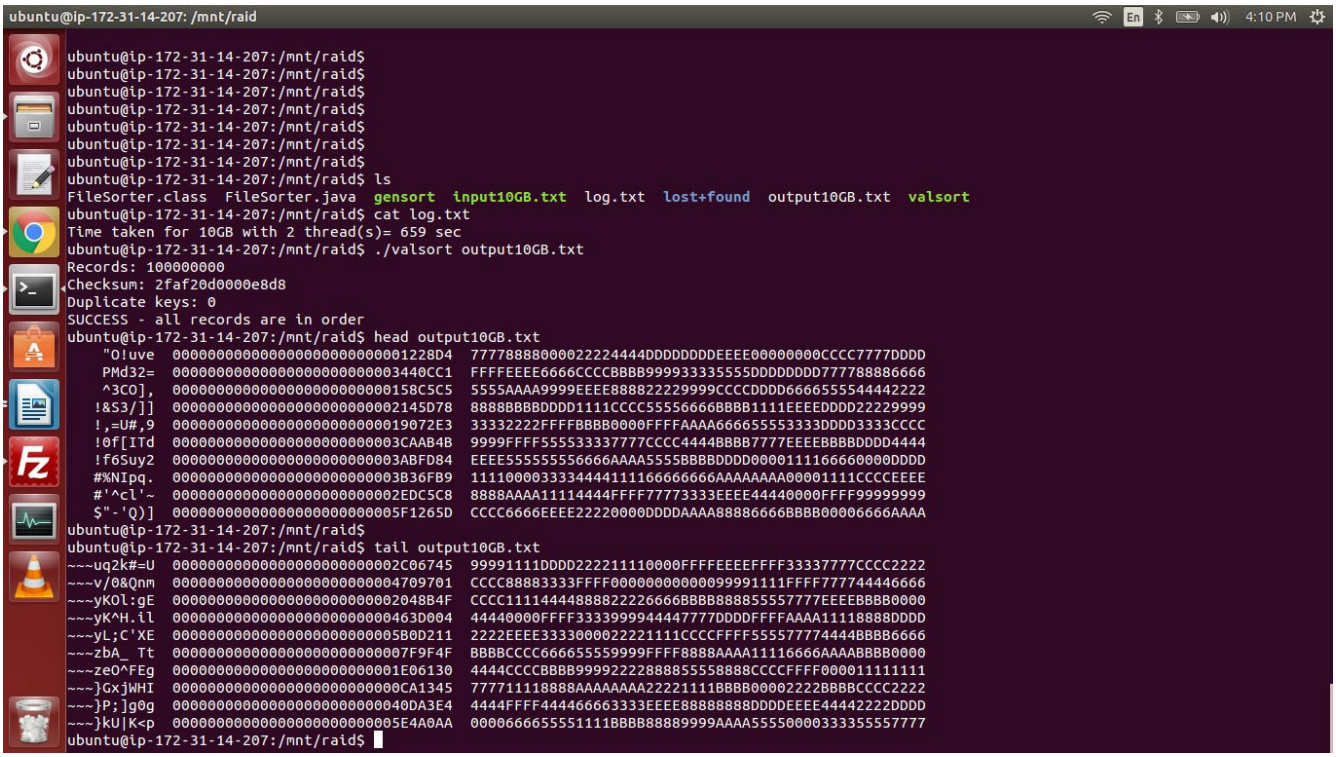

## 1.3 Experiment 2: SharedMemory 10GB on c3.large instance

a) 1 thread:

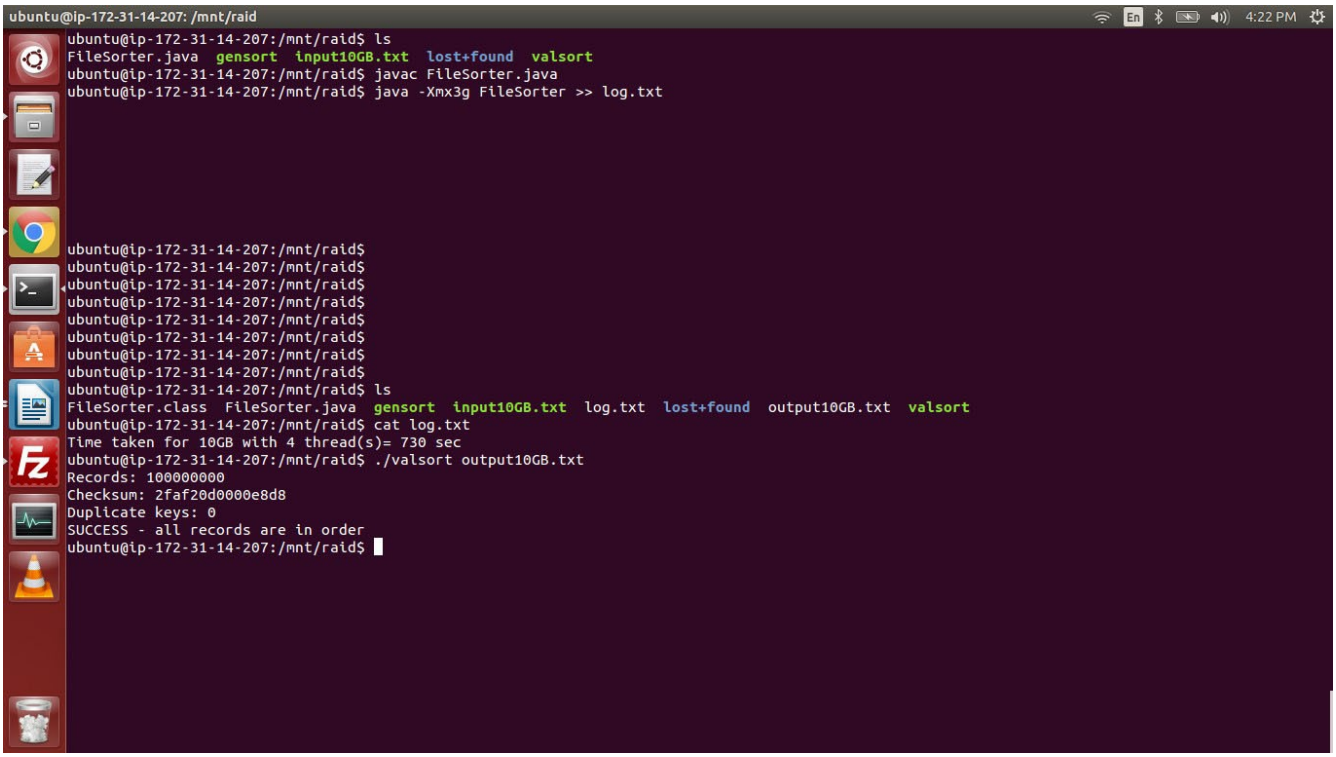


b) 2 threads:

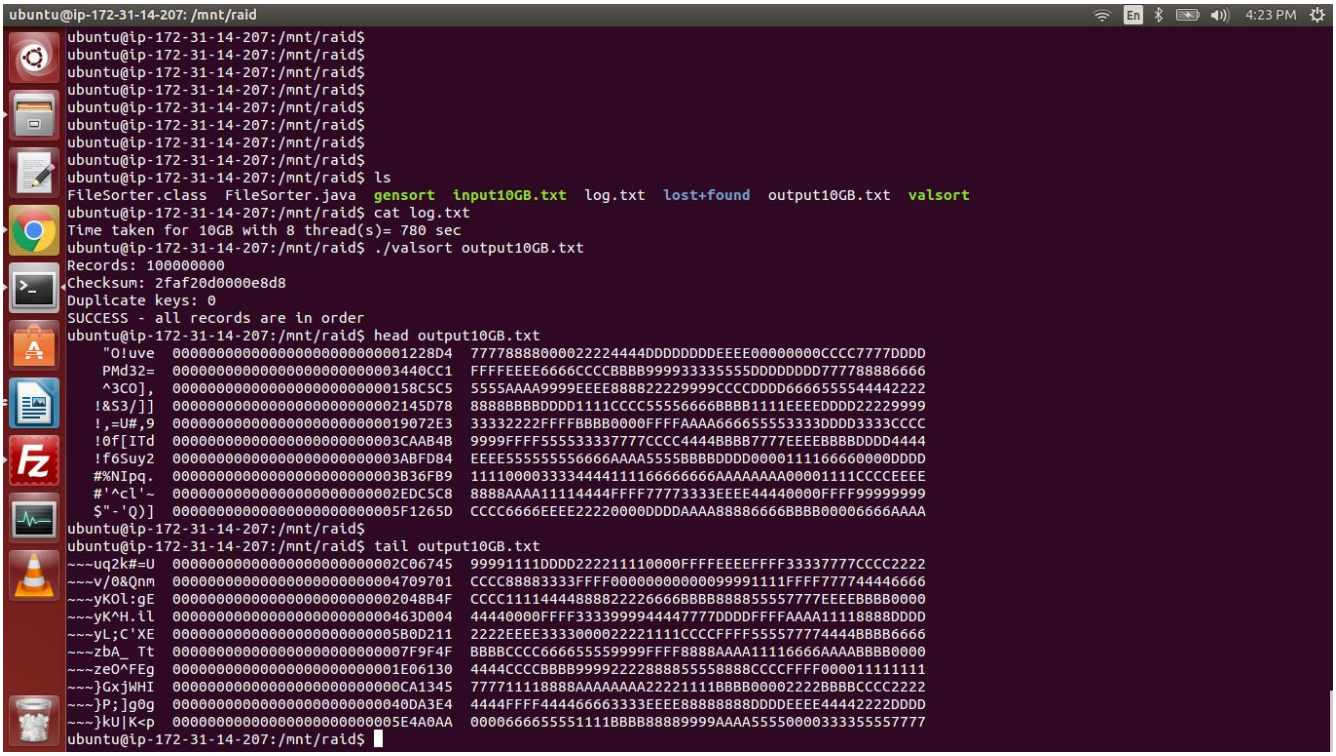

ubuntu@ip-172-31-14-207: /mnt/raid
4:10 PM
ubuntu@ip-172-31-14-207:/mnt/raid$ ls
FileSorter.java gensort input10GB.txt lost+found valsort
ubuntu@ip-172-31-14-207:/mnt/raid$ javac FileSorter.java
ubuntu@ip-172-31-14-207:/mnt/raid$ java -Xmx3g FileSorter >> log.txt
ubuntu@ip-172-31-14-207:/mnt/raid$ ls
FileSorter.class FileSorter.java gensort input10GB.txt log.txt lost+found output10GB.txt valsort
ubuntu@ip-172-31-14-207:/mnt/raid$ cat log.txt
Time taken for 10GB with 2 thread(s)= 659 sec
ubuntu@ip-172-31-14-207:/mnt/raid$ ./valsort output10GB.txt
Records: 100000000
Checksum: 2faf20d0000e8d8
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-14-207:/mnt/raid$


ubuntu@ip-172-31-14-207: /mnt/raid
4:10 PM
ubuntu@ip-172-31-14-207:/mnt/raid$ ls
FileSorter.class FileSorter.java gensort input10GB.txt log.txt lost+found output10GB.txt valsort
ubuntu@ip-172-31-14-207:/mnt/raid$ cat log.txt
Time taken for 10GB with 2 thread(s)= 659 sec
ubuntu@ip-172-31-14-207:/mnt/raid$ ./valsort output10GB.txt
Records: 100000000
Checksum: 2faf20d0000e8d8
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-14-207:/mnt/raid$ head output10GB.txt
"O!uve 0000000000000000000000000001228D4 77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
PMd32= 00000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB99993333555SDDDDDDDD777788886666
^3CO], 0000000000000000000000000158C5C5 5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
!&S3/]] 00000000000000000000000002145D78 8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
!,=U#,9 00000000000000000000000019072E3 33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
!0f[ITd 0000000000000000000000003CAAB4B 9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
!f6Suy2 00000000000000000000000003ABFD84 EEEE555555556666AAAA5555BBBBDDDD00001111666600000DDDD
#%NIpq. 00000000000000000000000003B36FB9 11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
#'^cl'~ 0000000000000000000000002EDC5C8 8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
$"-'Q)] 00000000000000000000000005F1265D CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
ubuntu@ip-172-31-14-207:/mnt/raid$
ubuntu@ip-172-31-14-207:/mnt/raid$ tail output10GB.txt
~~~uq2k#=U 00000000000000000000000002C06745 99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm 0000000000000000000000004709701 CCCC88883333FFFF0000000000009999111FFFF777744446666
~~~yKOl:gE 00000000000000000000000002048B4F CCCC1111444488882222666BBBB888855557777EEEEBBBB0000
~~~yK^H.il 00000000000000000000000463D004 44440000FFFF3333999944447777DDDDFFFFAAAA1111888DDDD
~~~yL;C'XE 0000000000000000000000005B0D211 2222EEEE333300002222111CCCCFFFF5555777744448BBBB6666
~~~zbA_ Tt 00000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg 00000000000000000000001E06130 4444CCCCBBBB99992222888855558888CCCCFFFF000011111111
~~~}GxjWHI 0000000000000000000000000CA1345 777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g 00000000000000000000000040DA3E4 4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p 00000000000000000000000005E4A0AA 0000666655551111BBBB88889999AAAA55550000333355557777
ubuntu@ip-172-31-14-207:/mnt/raid$

c) 4 threads:

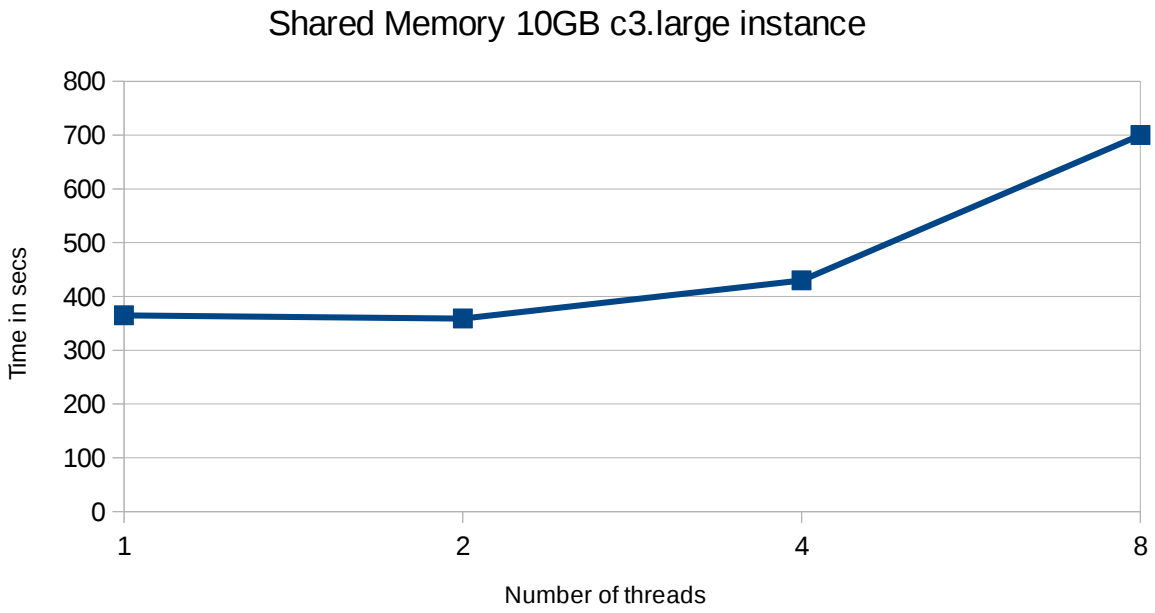



d) 8 threads



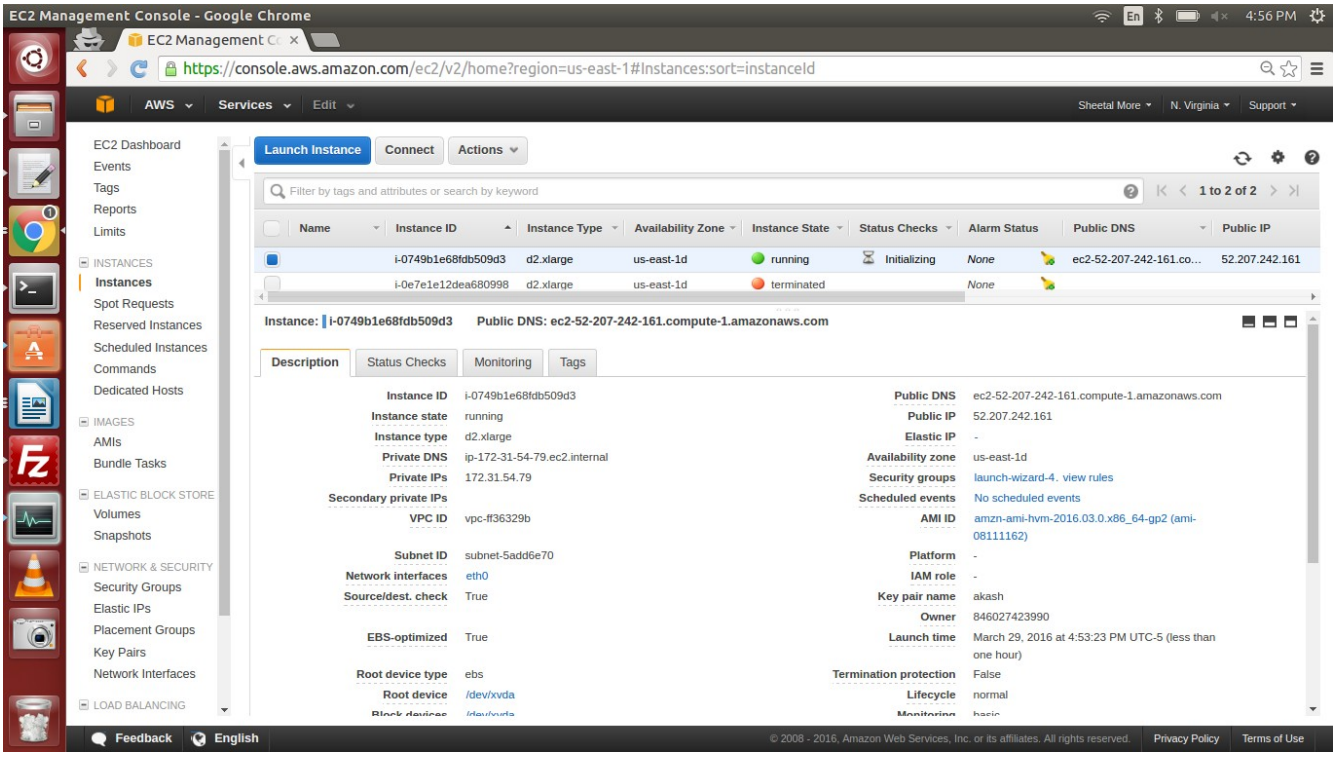

## Observations:

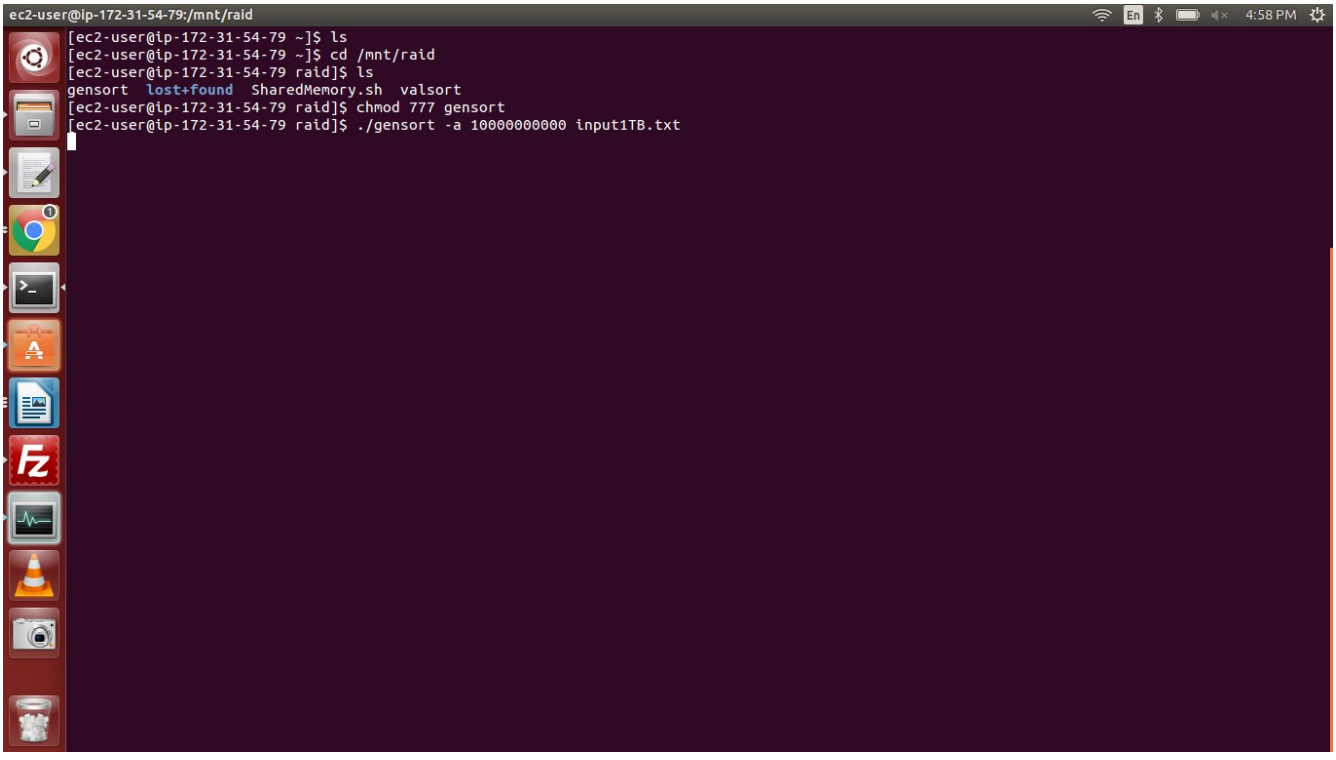
| Number of threads | Time taken (secs) |
|---|---|
| 1 | 365 |
| 2 | 359 |
| 4 | 430 |
| 8 | 700 |

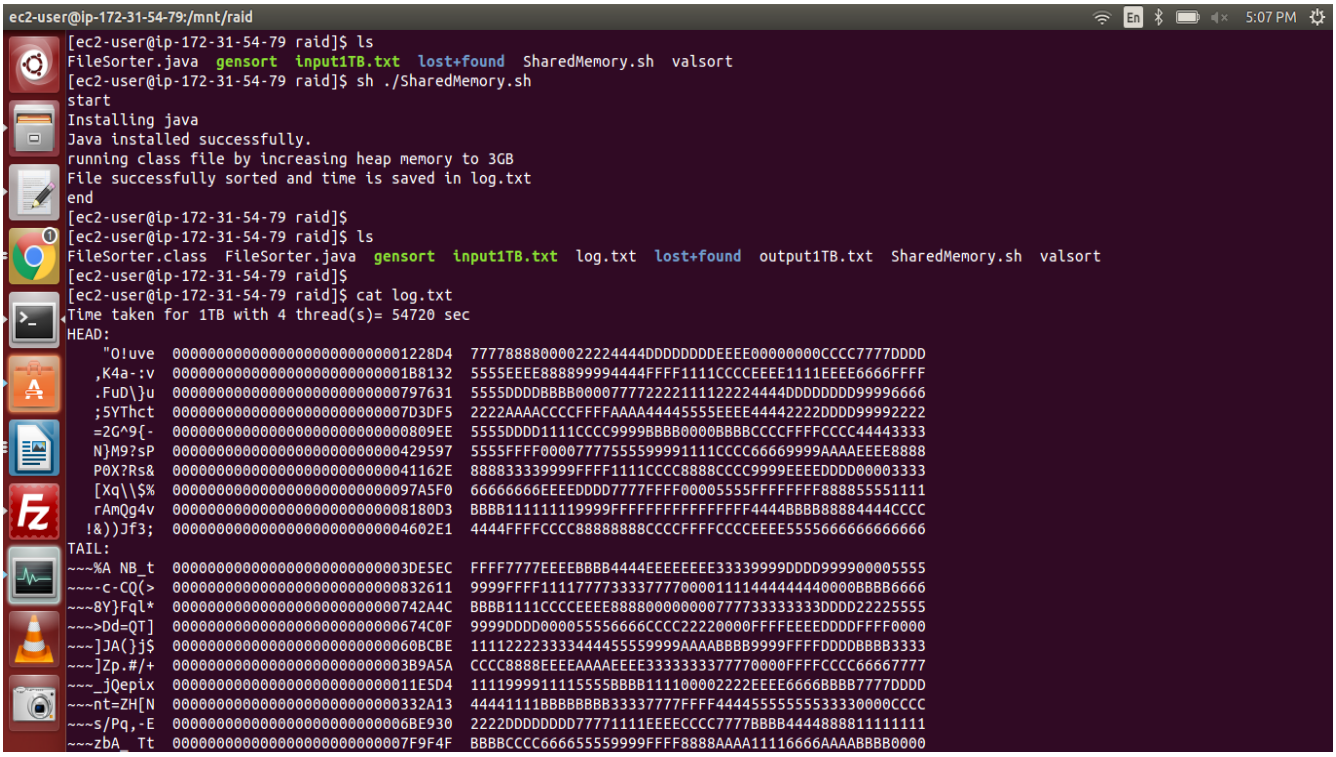


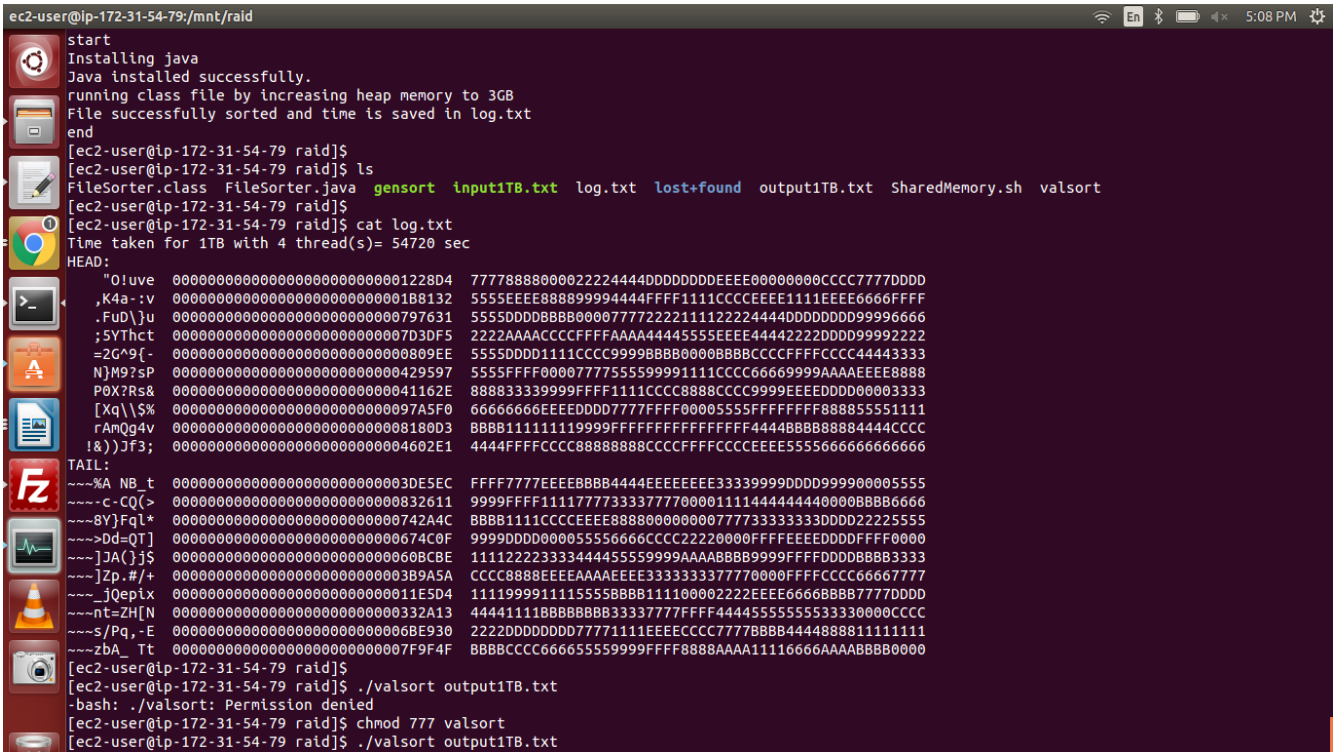

As seen above the least time required for sorting is 359 secs for 2 number of threads.

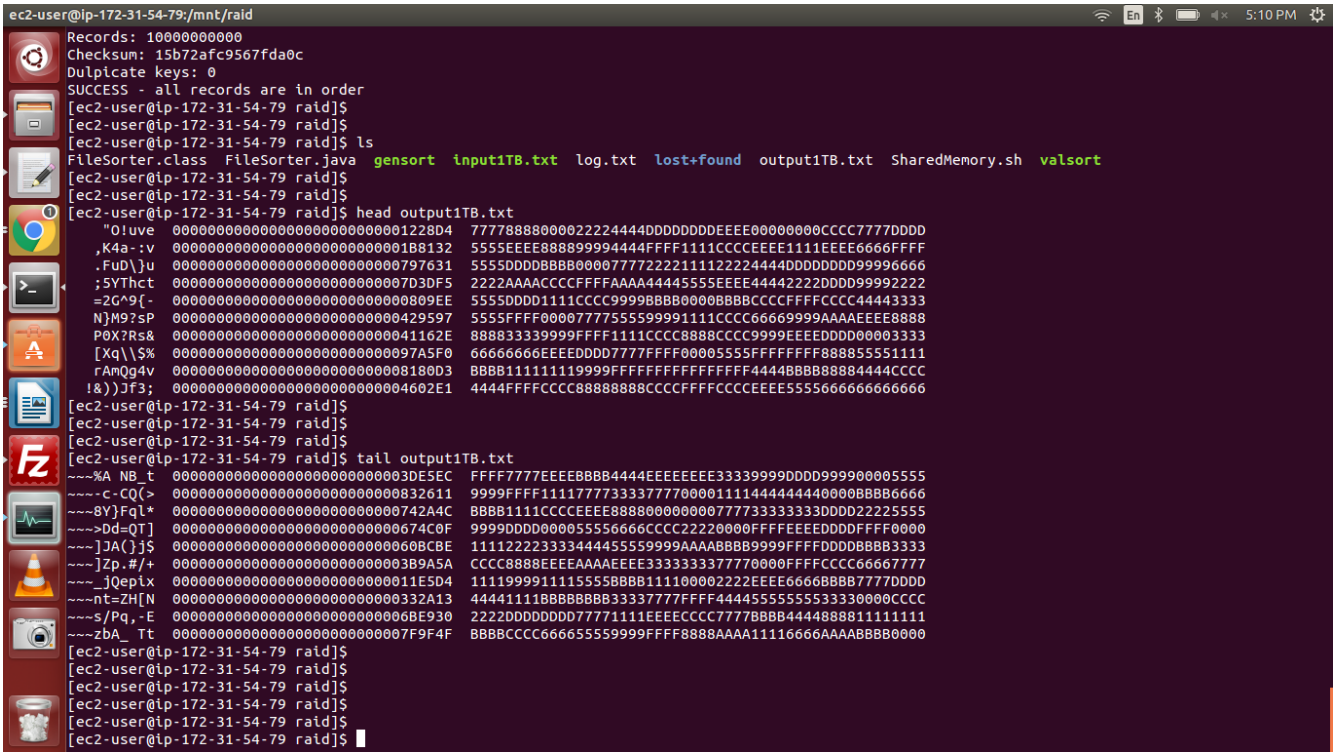
# 1.4: Experiment Shared Memory 1TB




ec2-user@ip-172-31-54-79:/mnt/raid
5:07 PM
[ec2-user@ip-172-31-54-79 raid]$ ls
FileSorter.java gensort input1TB.txt lost+found SharedMemory.sh valsort
[ec2-user@ip-172-31-54-79 raid]$ sh ./SharedMemory.sh
start
Installing java
Java installed successfully.
running class file by increasing heap memory to 3GB
File successfully sorted and time is saved in log.txt
end
[ec2-user@ip-172-31-54-79 raid]$
[ec2-user@ip-172-31-54-79 raid]$ ls
FileSorter.class FileSorter.java gensort input1TB.txt log.txt lost+found output1TB.txt SharedMemory.sh valsort
[ec2-user@ip-172-31-54-79 raid]$
[ec2-user@ip-172-31-54-79 raid]$ cat log.txt
Time taken for 1TB with 4 thread(s)= 54720 sec
HEAD:
"O!uve 00000000000000000000000001228D4 77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
,K4a-:v 00000000000000000000000001B8132 5555EEEE888899994444FFFF1111CCCCEEEE1111EEEE6666FFFF
.FuD\}u 00000000000000000000000000797631 5555DDDDBBBB000077772222111122224444DDDDDDDD99996666
;5YThct 00000000000000000000000007D3DF5 2222AAAACCCCFFFFAAAA44445555EEEE44442222DDDD99992222
=2G^9{- 00000000000000000000000000809EE 5555DDDD1111CCCC9999BBBB0000BBBBCCCCFFFFCCCC44443333
N}M9?sP 00000000000000000000000000429597 5555FFFF000077775555999991111CCCC66669999AAAAEEEE8888
P0X?Rs& 00000000000000000000000041162E 8888333399999FFFF1111CCCC8888CCCC9999EEEEDDDD00003333
[Xq\\$% 00000000000000000000000097A5F0 66666666EEEEDDDD7777FFFF00005555FFFFFFFF888855551111
rAmQg4v 00000000000000000000000008180D3 BBBB111111119999FFFFFFFFFFFFFFFF4444BBBB88884444CCCC
!&))Jf3; 00000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCEEEE5555666666666666
TAIL:
~~~%A NB_t 00000000000000000000000003DE5EC FFFF7777EEEEBBBB4444EEEEEEEE33339999DDDD999900005555
~~~-c-CQ(> 00000000000000000000000000832611 9999FFFF1111777733337777000011114444444440000BBBB6666
~~~8Y}Fql* 00000000000000000000000000742A4C BBBB1111CCCCEEEE888800000000077773333333333DDDD22225555
~~~>Dd=QT] 00000000000000000000000000674C0F 9999DDDD000055556666CCCC22220000FFFFEEEEDDDDFFFF0000
~~~]JA(}j$ 00000000000000000000000000060BCBE 11112222333344445555999AAAABBBB9999FFFFDDDDBBBB3333
~~~]Zp.#/+ 00000000000000000000000003B9A5A CCCC8888EEEEAAAAEEEE33333333777700000FFFFCCCC66667777
~~~_jQepix 00000000000000000000000011E5D4 1111999911115555BBBB111100002222EEEE6666BBBB7777DDDD
~~~nt=ZH[N 00000000000000000000000000332A13 44441111BBBBBBBB33337777FFFF444455555555333330000CCCC
~~~s/Pq,-E 00000000000000000000000006BE930 2222DDDDDDDD77771111EEEECCCC7777BBBB4444888811111111
~~~zbA_ Tt 00000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000


ec2-user@ip-172-31-54-79:/mnt/raid
5:08 PM
start
Installing java
Java installed successfully.
running class file by increasing heap memory to 3GB
File successfully sorted and time is saved in log.txt
end
[ec2-user@ip-172-31-54-79 raid]$
[ec2-user@ip-172-31-54-79 raid]$ ls
FileSorter.class FileSorter.java gensort input1TB.txt log.txt lost+found output1TB.txt SharedMemory.sh valsort
[ec2-user@ip-172-31-54-79 raid]$
[ec2-user@ip-172-31-54-79 raid]$ cat log.txt
Time taken for 1TB with 4 thread(s)= 54720 sec
HEAD:
"O!uve 00000000000000000000000001228D4 77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
,K4a-:v 00000000000000000000000001B8132 5555EEEE888899994444FFFF1111CCCCEEEE1111EEEE6666FFFF
.FuD\}u 00000000000000000000000000797631 5555DDDDBBBB000077772222111122224444DDDDDDDD99996666
;5YThct 00000000000000000000000007D3DF5 2222AAAACCCCFFFFAAAA44445555EEEE44442222DDDD99992222
=2G^9{- 00000000000000000000000000809EE 5555DDDD1111CCCC9999BBBB0000BBBBCCCCFFFFCCCC44443333
N}M9?sP 00000000000000000000000000429597 5555FFFF000077775555999991111CCCC66669999AAAAEEEE8888
P0X?Rs& 00000000000000000000000041162E 8888333399999FFFF1111CCCC8888CCCC9999EEEEDDDD00003333
[Xq\\$% 00000000000000000000000097A5F0 66666666EEEEDDDD7777FFFF00005555FFFFFFFF888855551111
rAmQg4v 00000000000000000000000008180D3 BBBB111111119999FFFFFFFFFFFFFFFF4444BBBB88884444CCCC
!&))Jf3; 00000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCEEEE5555666666666666
TAIL:
~~~%A NB_t 00000000000000000000000003DE5EC FFFF7777EEEEBBBB4444EEEEEEEE33339999DDDD999900005555
~~~-c-CQ(> 00000000000000000000000000832611 9999FFFF1111777733337777000011114444444440000BBBB6666
~~~8Y}Fql* 00000000000000000000000000742A4C BBBB1111CCCCEEEE888800000000077773333333333DDDD22225555
~~~>Dd=QT] 00000000000000000000000000674C0F 9999DDDD000055556666CCCC22220000FFFFEEEEDDDDFFFF0000
~~~]JA(}j$ 00000000000000000000000000060BCBE 11112222333344445555999AAAABBBB9999FFFFDDDDBBBB3333
~~~]Zp.#/+ 00000000000000000000000003B9A5A CCCC8888EEEEAAAAEEEE33333333777700000FFFFCCCC66667777
~~~_jQepix 00000000000000000000000011E5D4 1111999911115555BBBB111100002222EEEE6666BBBB7777DDDD
~~~nt=ZH[N 00000000000000000000000000332A13 44441111BBBBBBBB33337777FFFF444455555555333330000CCCC
~~~s/Pq,-E 00000000000000000000000006BE930 2222DDDDDDDD77771111EEEECCCC7777BBBB4444888811111111
~~~zbA_ Tt 00000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
[ec2-user@ip-172-31-54-79 raid]$
[ec2-user@ip-172-31-54-79 raid]$ ./valsort output1TB.txt
-bash: ./valsort: Permission denied
[ec2-user@ip-172-31-54-79 raid]$ chmod 777 valsort
[ec2-user@ip-172-31-54-79 raid]$ ./valsort output1TB.txt




***Total time taken = 54720 secs***

# HADOOP SORT APPLCATION

## 2.1. Introduction:

### Problem Statement:

The goal of this experiment is to sort a large dataset which contains one record per line of 100 bytes and we need to sort them on the basis of ASCII values of first 10 bytes. If we use the traditional methodology of sorting then it would not only require higher space, but will also take higher time.

***What is master node? What is slave node?***

Hadoop consists of master and slaves in which master is responsible for storing large dataset and doing parallel computations across nodes (i.e. slaves). It had the Namenode, JobTracker, TaskTracker, whereas the slave node acts as the Datanode.
Slave nodes are the ones that actually perform the computations in parallel and are responsible for executing the actual logic of the problem statement.

***Configuration files:***

1. conf/masters: In this configuration file, we mention the ip address of the master node. This also optionally consists of the hostnames of the secondary node servers in case the master (namenode) fails.

2. conf/slaves: In this configuration file of masters, we mention the ip addresses of all the slave nodes present in the cluster. And in case of slaves, it would consist of localhost.

3. conf/core-site.xml: This file consists of the metadata which specifies the hadoop where the namenode (master) is located, thus we mention the ip address of the master node. We also mention the port number for the master and mostly used is 8020.

4. conf/hdfs-site.xml: This file specfies the configuration for hadoop which consists of the master, secondary name node and data nodes. We also mention the number of replications to keep for the data to be retrieved in case of failure.

5. cong/map-red.xml: This file mentions the location of job tracker. The property mapred.job.tracker takes the value of the master ip address and port number for the communication purpose.

***Setting unique ports:***

The reason we set unique ports is that we do not want it to match with the well known ports, else there would be problem when using the common processes like ssh etc. Also, if each user has same port number then they won't be able to use the hadoop deamons since it would then clash with the 2 users.

***Changing mapper and reducers:***

In order to change the number of mapper and reducers, we need to edit the configuration file mapred-site.xml in which there is a parameter *mapreduce.job.maps* and *mapreduce.job.maps*

**Methodology:**

- One of the ways of solving above problem efficiently is to use Hadoop framework which is used for distributed storage and distributed processing.
- Hadoop used the MapReduce Programming model.
- The storage system of Hadoop is called the HDFS (Hadoop Distributed File System) and the actual processing is the Map-reduce which has 2 phases Map and reduce.
- Hadoop works by distributing a large dataset into number of chunks and then distributing those on various nodes in the cluster which then runs in parallel.

**Runtime environment settings:**

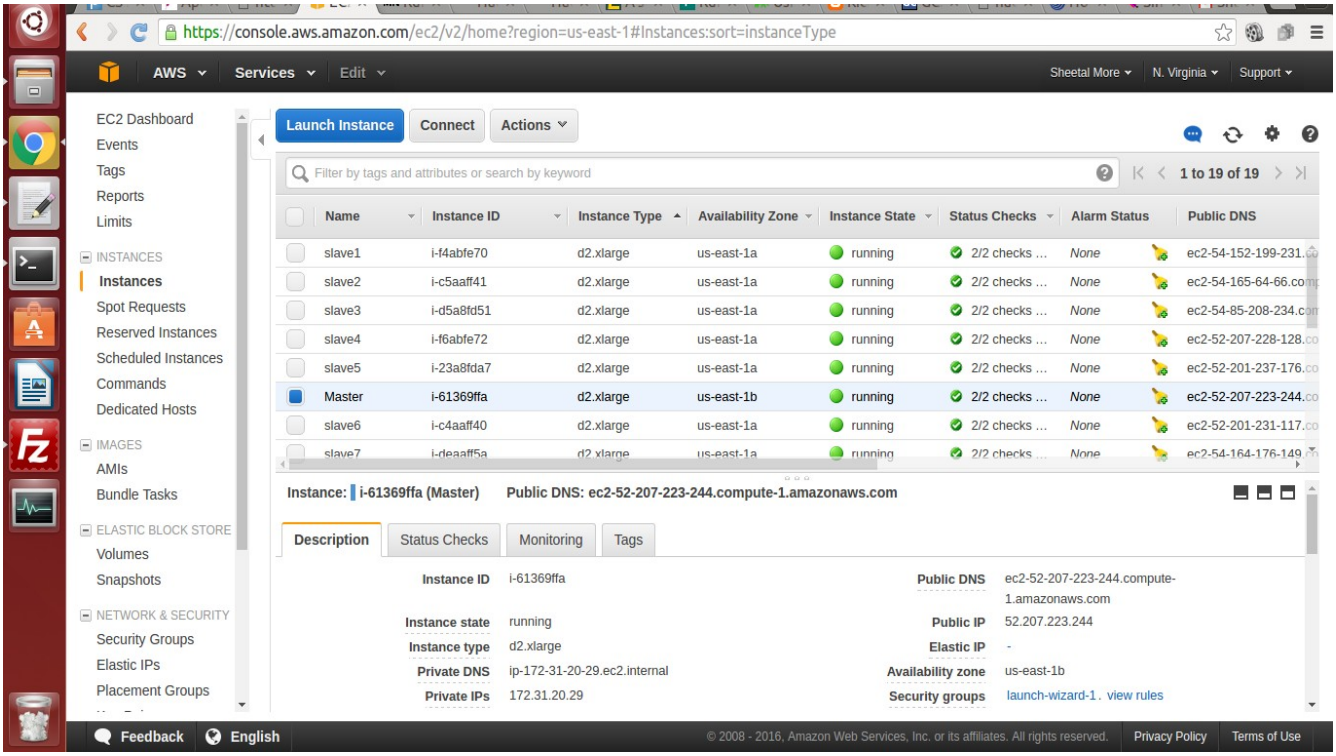
| | |
|---|---|
| Type of EC2 instance used | c3.large |
| Region | N. Virginia |
| OS and kernel | Linux version 4.4.5-15.26.amzn1.x86_64 (mockbuild@gobi-build-60007) (gcc version 4.8.3 20140911 (Red Hat 4.8.3-9) (GCC) ) |
| Implementation Language | JAVA |
| JAVA version | java version "1.8.0_73"<br>Java(TM) SE Runtime Environment (build 1.8.0_73-b02)<br>Java HotSpot(TM) 64-Bit Server VM (build 25.73-b02, mixed mode) |
| Hadoop Version | Hadoop 2.7.2 |

**Installation steps to set up virtual cluster:**

1. Request a spot instance of c3.large on aws. Transfer the script file to the newly created instance.
2. Run the script file which would then install java, hadoop on the master and update the configuration files as well.
3. Create an image of this instance and then add EBS volume of 400 GB.
4. Run the EBS script to mount the 400 GB volume on a folder /mnt/raid.
5. Using the AMI created in step 3, launch 16 instances (spot requests) along with 4000GB EBS volume and note down their ip addresses.
6. Edit the slaves files in configuration files of hadoop and add the ip addresses of slaves.
7. The EBS volumes of slaves are mounted using the pssh (parallel ssh command) which would then mount on /mnt/raid folder of all slaves.
8. We create a tmp directories on all the instances which would be used to store the intermediate data generated during running the hadoop framework.
9. The hdfs namenode has to be formatted before we begin the hdfs since it would then remove all the intermediate data.
10. Next is to run the hdfs using the script file “start-dfs.sh”, for the newer versions we also need to run the start-yarn.sh file
11. Verify if the cluster is created using the *jps* command which shows the services running on master and slave.

**Experiment 1: Hadoop d2.xlarge single node 1GB**

In this experiment I used d2.xlarge instance to sort 1GB file on single node (one master, one slave). The results are as shown in screenshots:



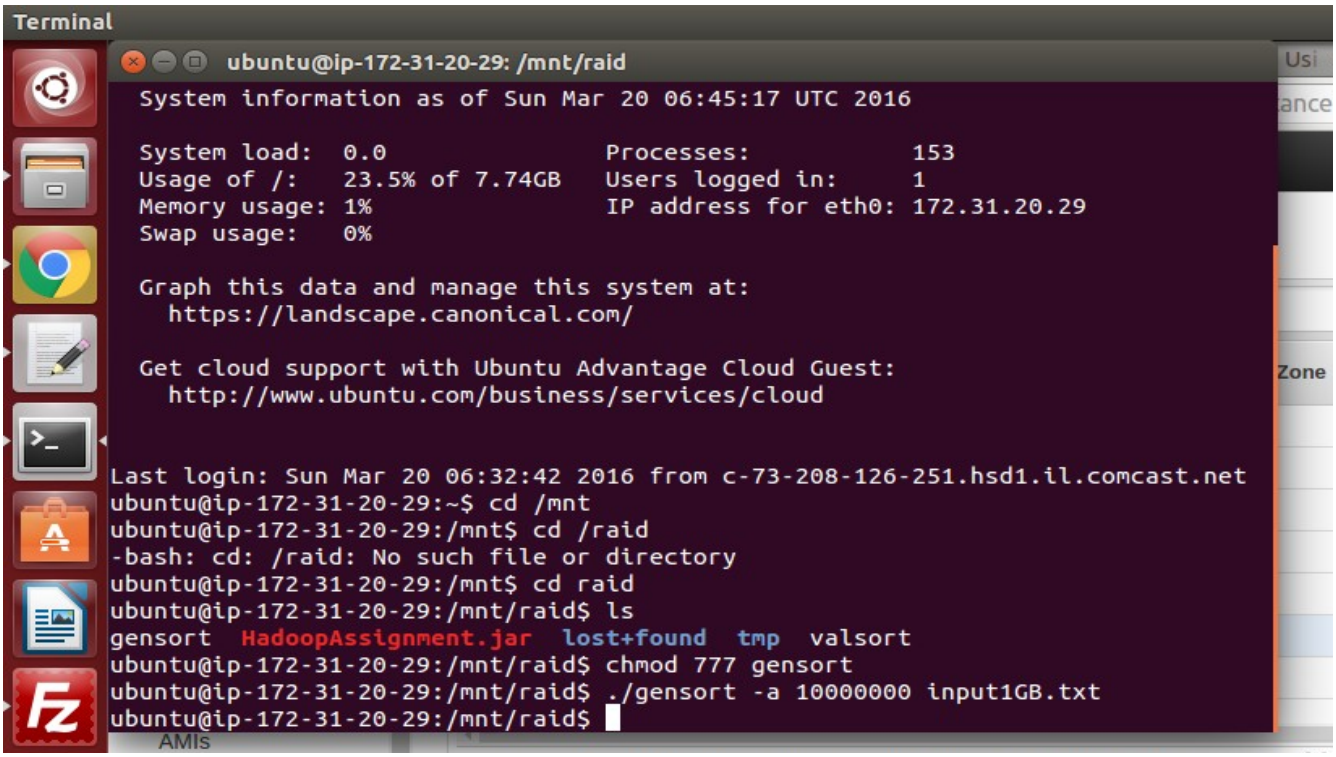



*Illustration A: gensort*

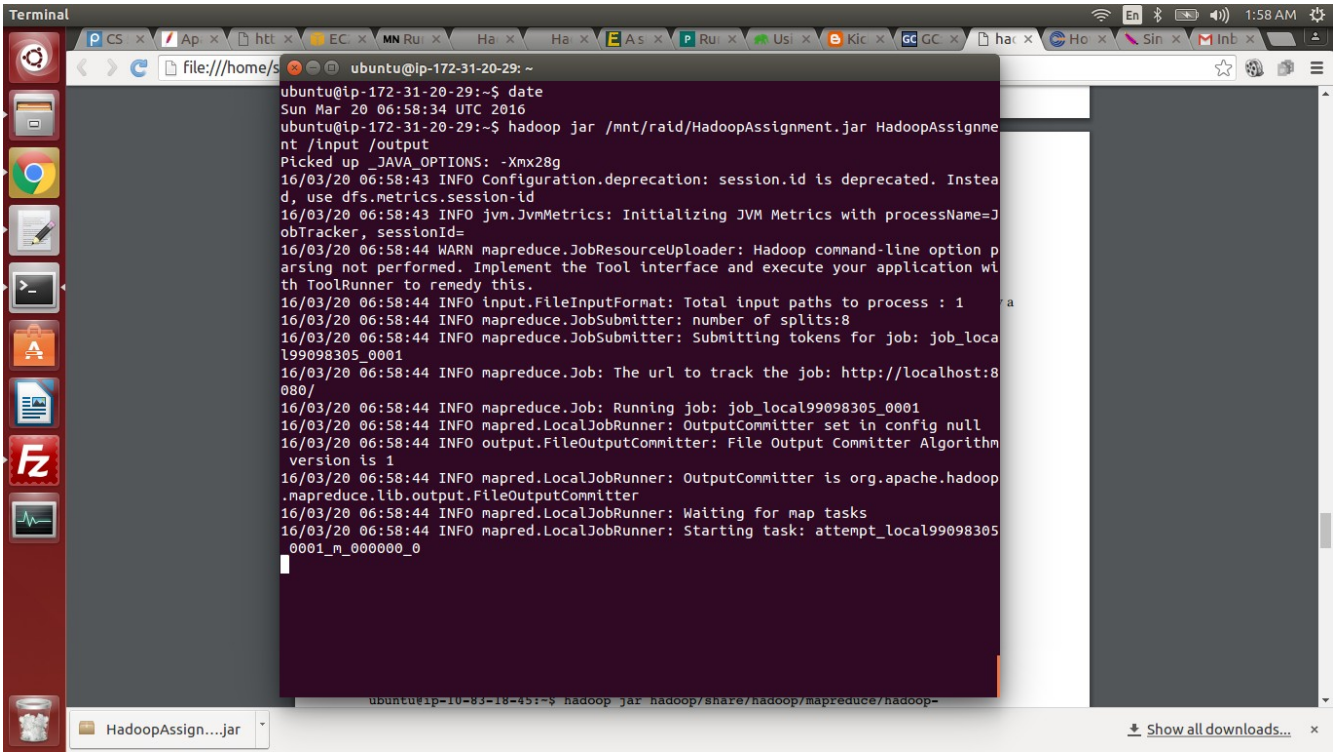



*Illustration B: running jar using hadoop command*

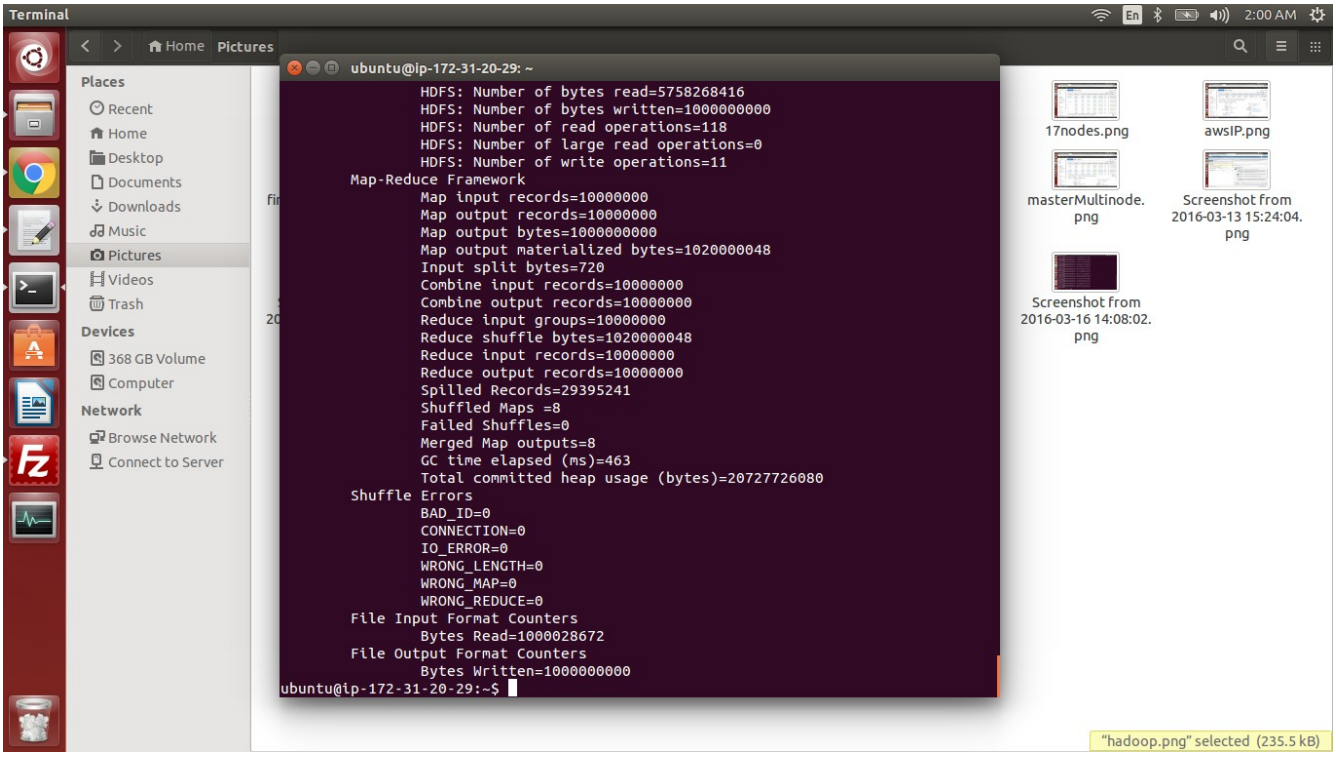



*Illustration C: hadoop 1GB output*

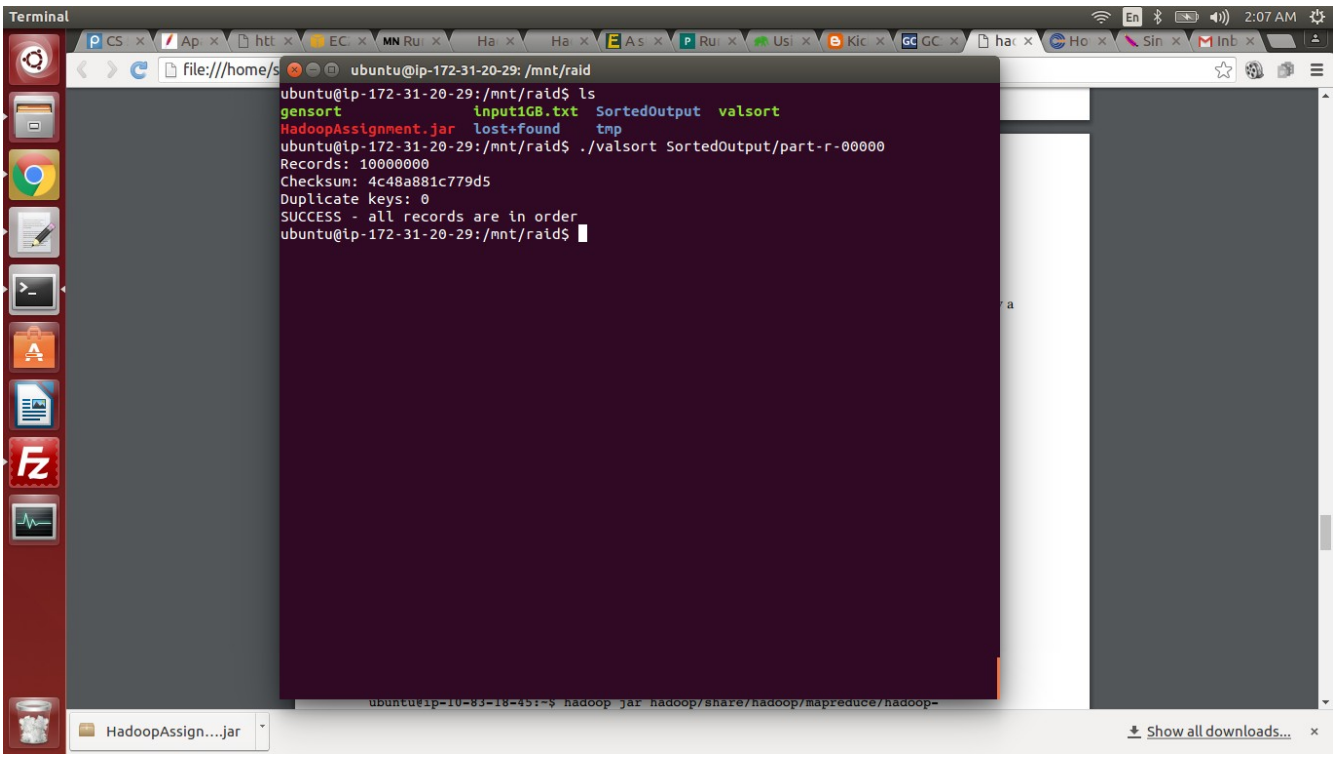


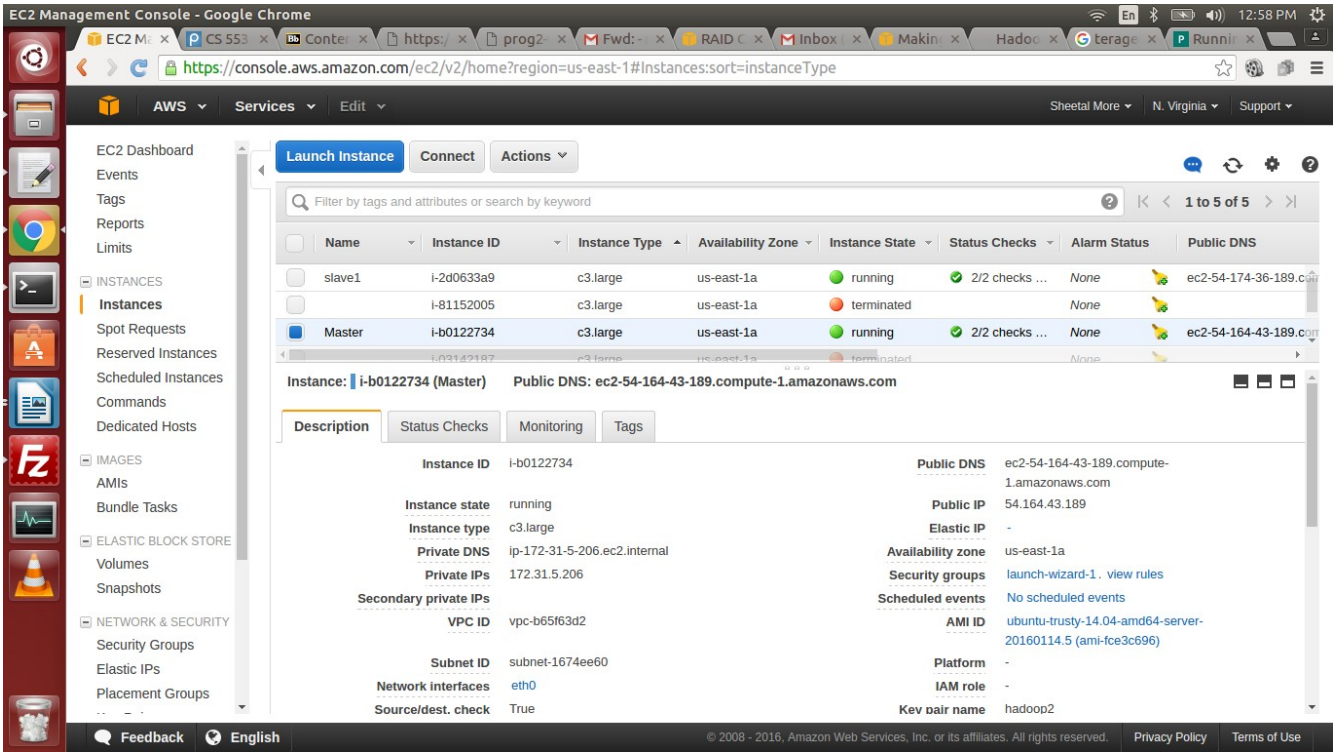

*Illustration D: hadoop 1GB valsort*

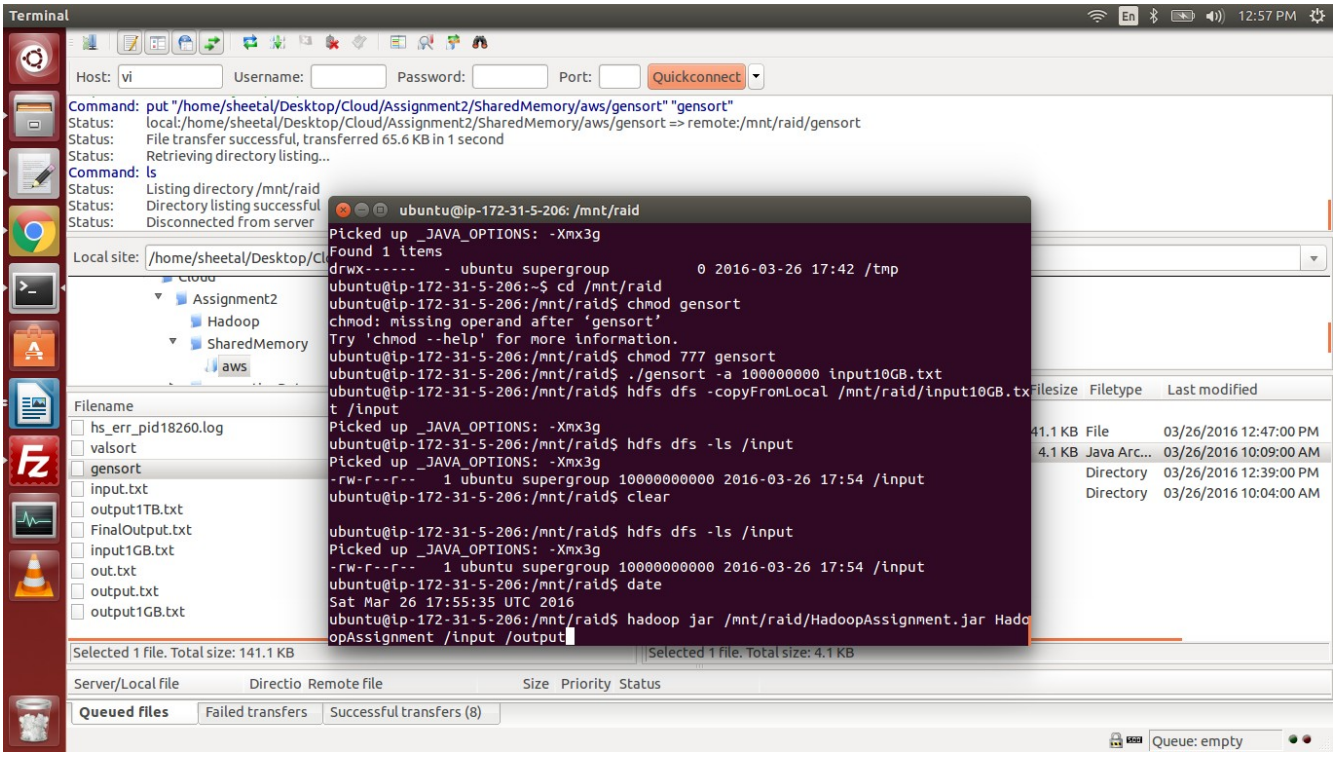
**Observations:**

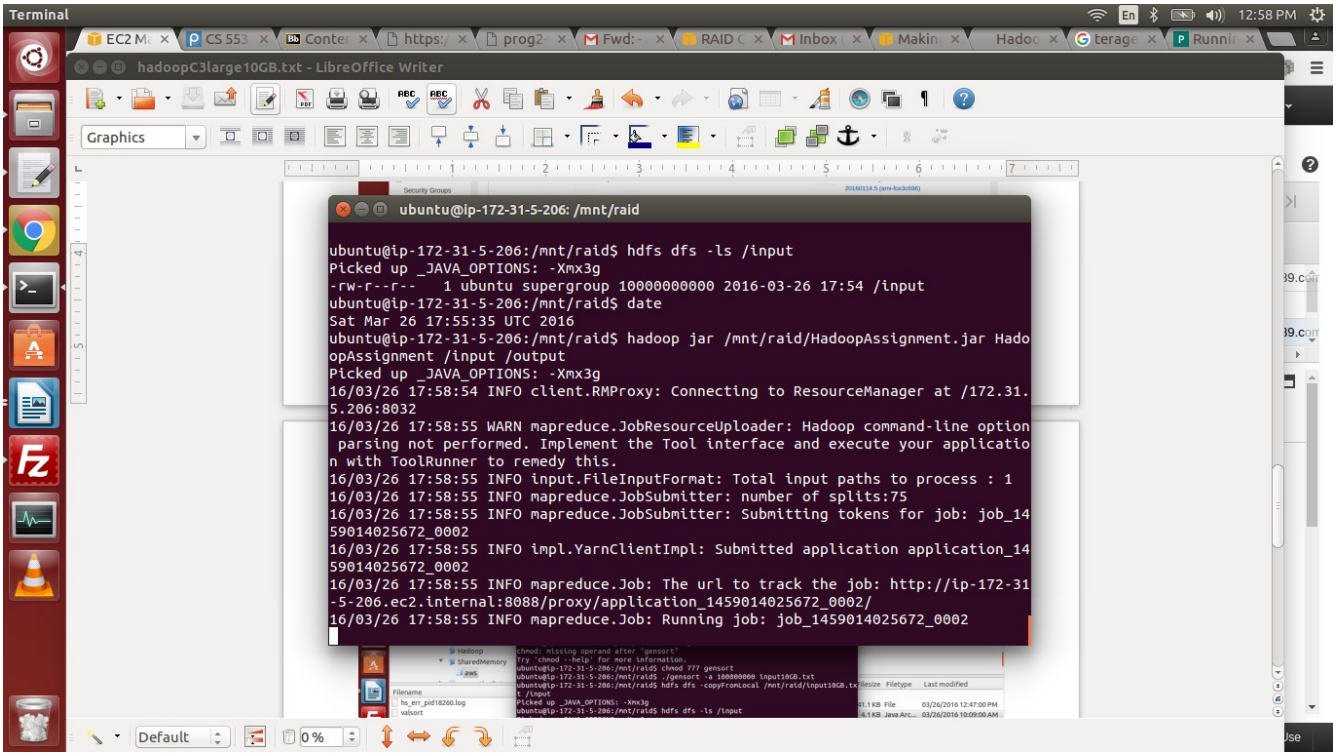
***Total time taken = 128 secs***

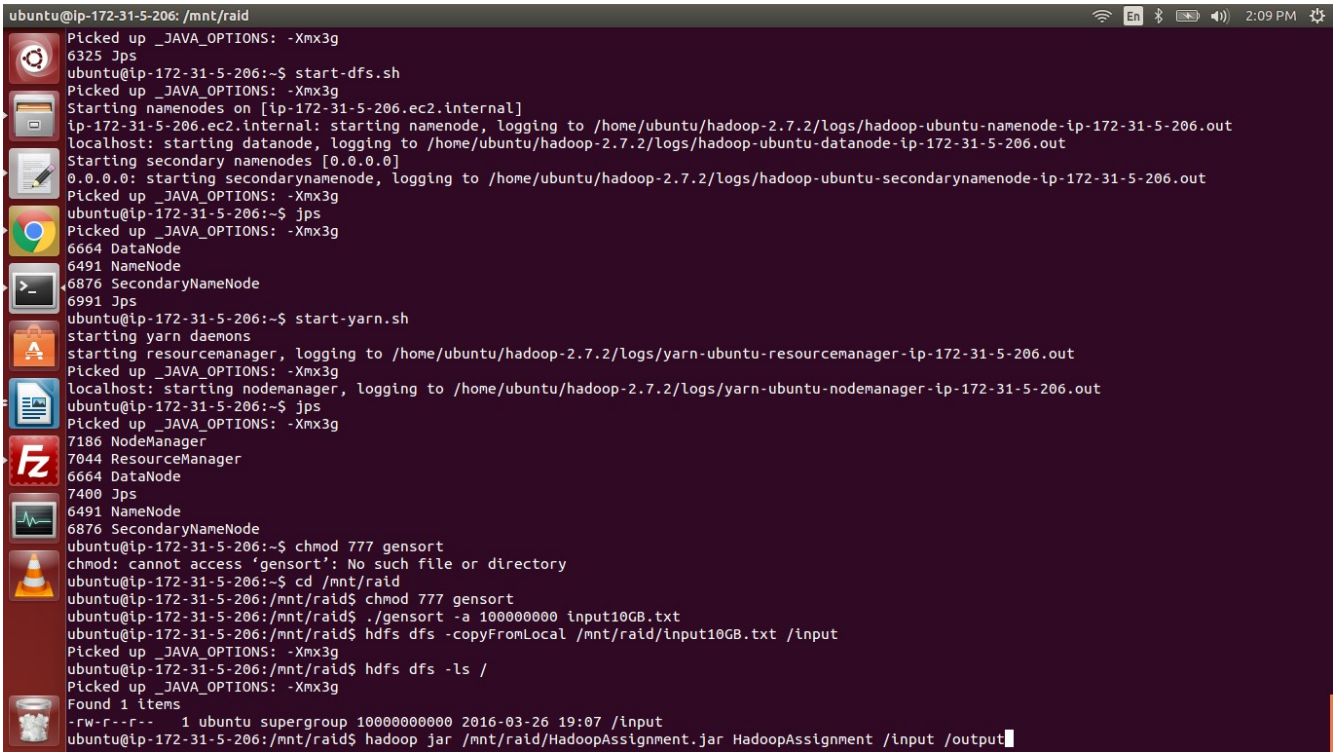
## Experiment 2: Hadoop c3.large single node 10GB

C3 large single node

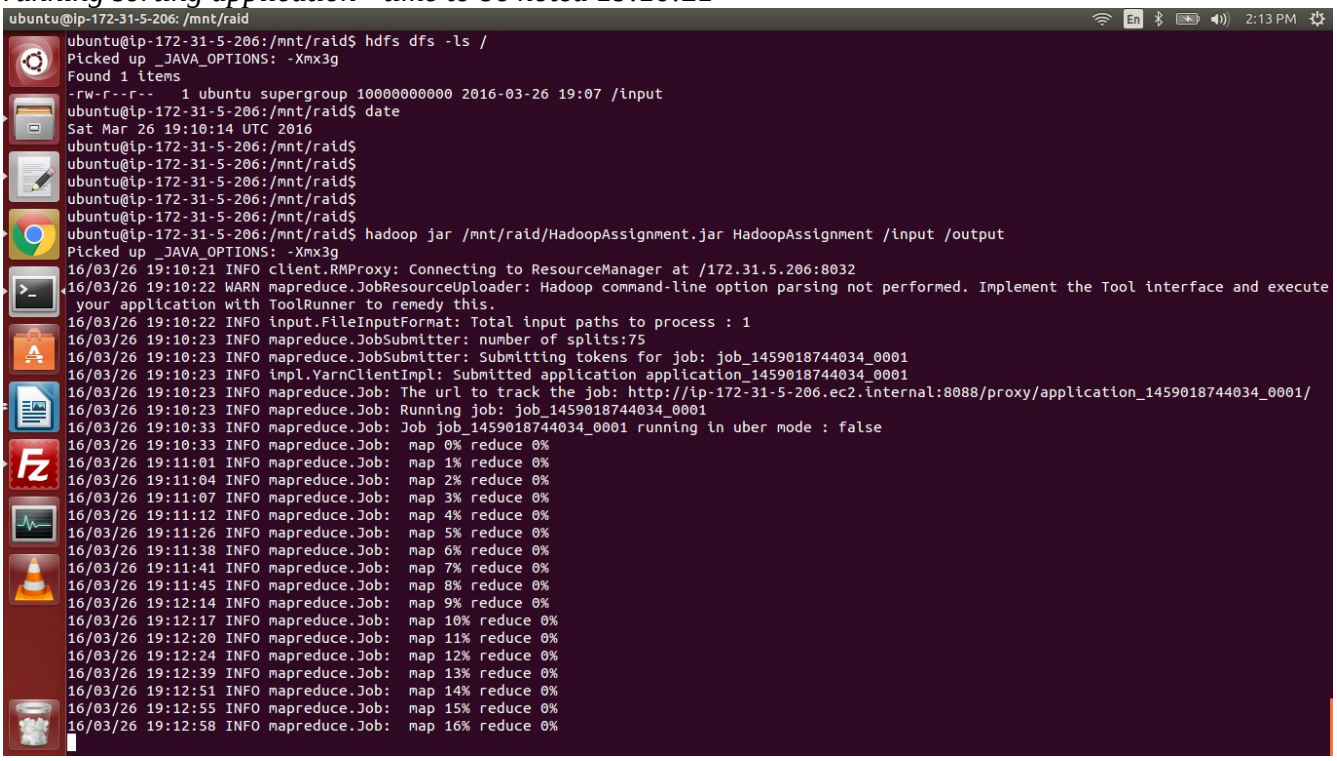


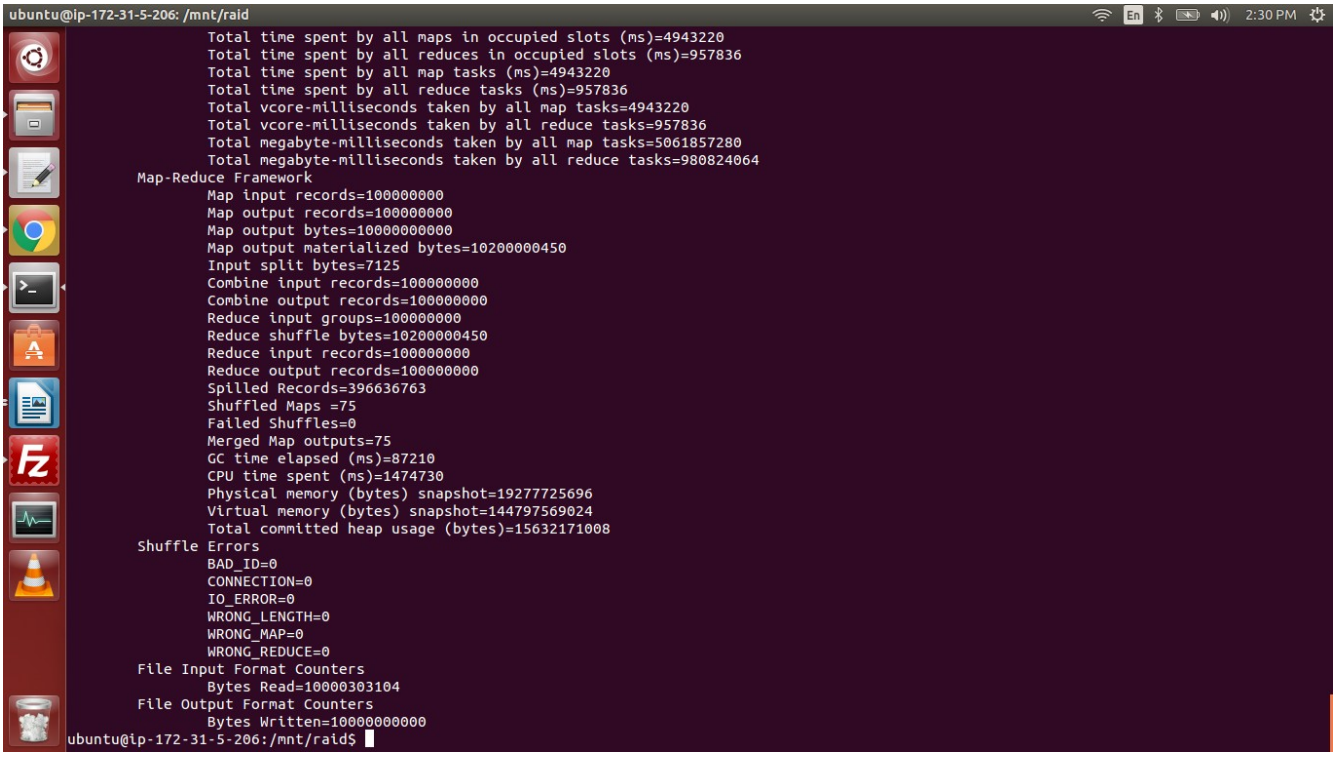



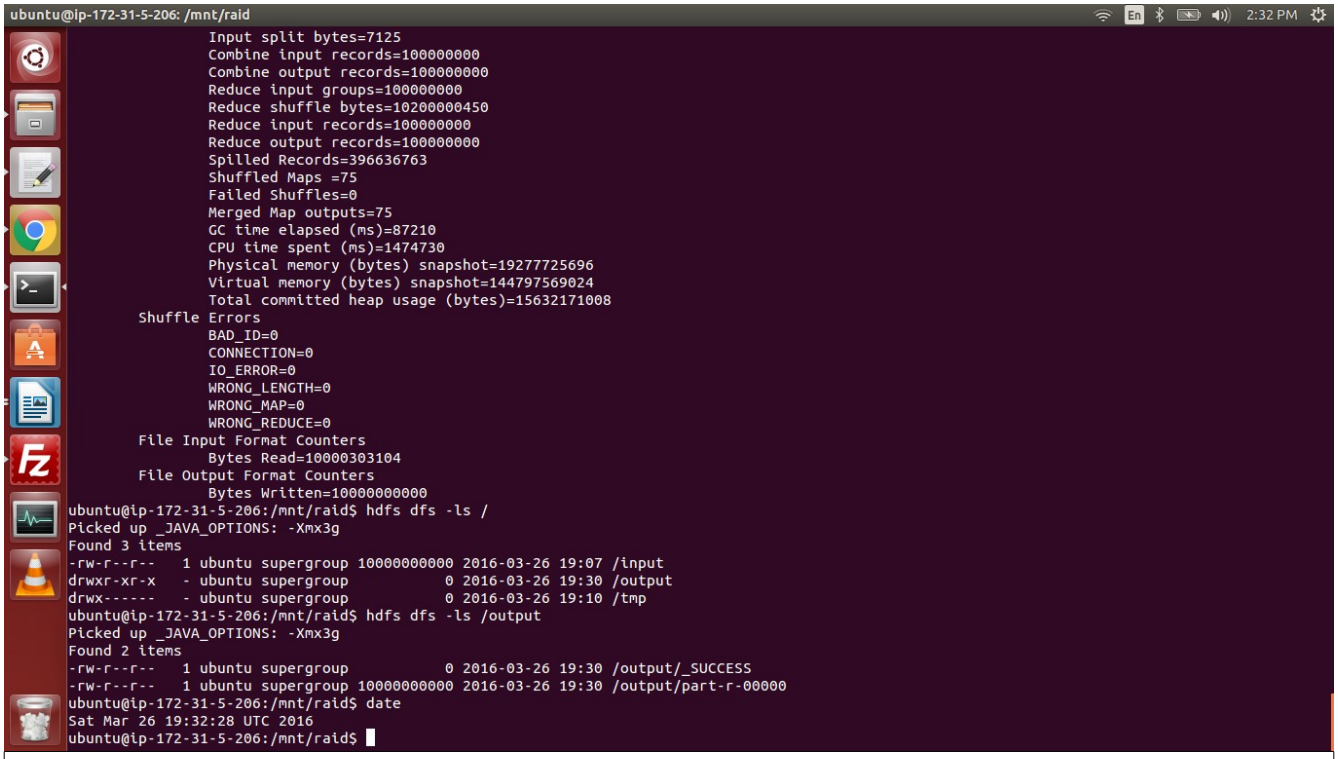

Terminal
12:58 PM
hadoopC3large10GB.txt - LibreOffice Writer
Graphics
ubuntu@ip-172-31-5-206: /mnt/raid
ubuntu@ip-172-31-5-206:/mnt/raid$ hdfs dfs -ls /input
Picked up _JAVA_OPTIONS: -Xmx3g
-rw-r--r--   1 ubuntu supergroup 10000000000 2016-03-26 17:54 /input
ubuntu@ip-172-31-5-206:/mnt/raid$ date
Sat Mar 26 17:55:35 UTC 2016
ubuntu@ip-172-31-5-206:/mnt/raid$ hadoop jar /mnt/raid/HadoopAssignment.jar HadoopAssignment /input /output
Picked up _JAVA_OPTIONS: -Xmx3g
16/03/26 17:58:54 INFO client.RMProxy: Connecting to ResourceManager at /172.31.5.206:8032
16/03/26 17:58:55 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
16/03/26 17:58:55 INFO input.FileInputFormat: Total input paths to process : 1
16/03/26 17:58:55 INFO mapreduce.JobSubmitter: number of splits:75
16/03/26 17:58:55 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1459014025672_0002
16/03/26 17:58:55 INFO impl.YarnClientImpl: Submitted application application_1459014025672_0002
16/03/26 17:58:55 INFO mapreduce.Job: The url to track the job: http://ip-172-31-5-206.ec2.internal:8088/proxy/application_1459014025672_0002/
16/03/26 17:58:55 INFO mapreduce.Job: Running job: job_1459014025672_0002
Default
0 %


ubuntu@ip-172-31-5-206: /mnt/raid
2:09 PM
Picked up _JAVA_OPTIONS: -Xmx3g
6325 Jps
ubuntu@ip-172-31-5-206:~$ start-dfs.sh
Picked up _JAVA_OPTIONS: -Xmx3g
Starting namenodes on [ip-172-31-5-206.ec2.internal]
ip-172-31-5-206.ec2.internal: starting namenode, logging to /home/ubuntu/hadoop-2.7.2/logs/hadoop-ubuntu-namenode-ip-172-31-5-206.out
localhost: starting datanode, logging to /home/ubuntu/hadoop-2.7.2/logs/hadoop-ubuntu-datanode-ip-172-31-5-206.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/ubuntu/hadoop-2.7.2/logs/hadoop-ubuntu-secondarynamenode-ip-172-31-5-206.out
Picked up _JAVA_OPTIONS: -Xmx3g
ubuntu@ip-172-31-5-206:~$ jps
Picked up _JAVA_OPTIONS: -Xmx3g
6664 DataNode
6491 NameNode
6876 SecondaryNameNode
6991 Jps
ubuntu@ip-172-31-5-206:~$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-resourcemanager-ip-172-31-5-206.out
Picked up _JAVA_OPTIONS: -Xmx3g
localhost: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-5-206.out
ubuntu@ip-172-31-5-206:~$ jps
Picked up _JAVA_OPTIONS: -Xmx3g
7186 NodeManager
7044 ResourceManager
6664 DataNode
7400 Jps
6491 NameNode
6876 SecondaryNameNode
ubuntu@ip-172-31-5-206:~$ chmod 777 gensort
chmod: cannot access ‘gensort’: No such file or directory
ubuntu@ip-172-31-5-206:~$ cd /mnt/raid
ubuntu@ip-172-31-5-206:/mnt/raid$ chmod 777 gensort
ubuntu@ip-172-31-5-206:/mnt/raid$ ./gensort -a 100000000 input10GB.txt
ubuntu@ip-172-31-5-206:/mnt/raid$ hdfs dfs -copyFromLocal /mnt/raid/input10GB.txt /input
Picked up _JAVA_OPTIONS: -Xmx3g
ubuntu@ip-172-31-5-206:/mnt/raid$ hdfs dfs -ls /
Picked up _JAVA_OPTIONS: -Xmx3g
Found 1 items
-rw-r--r--   1 ubuntu supergroup 10000000000 2016-03-26 19:07 /input
ubuntu@ip-172-31-5-206:/mnt/raid$ hadoop jar /mnt/raid/HadoopAssignment.jar HadoopAssignment /input /output

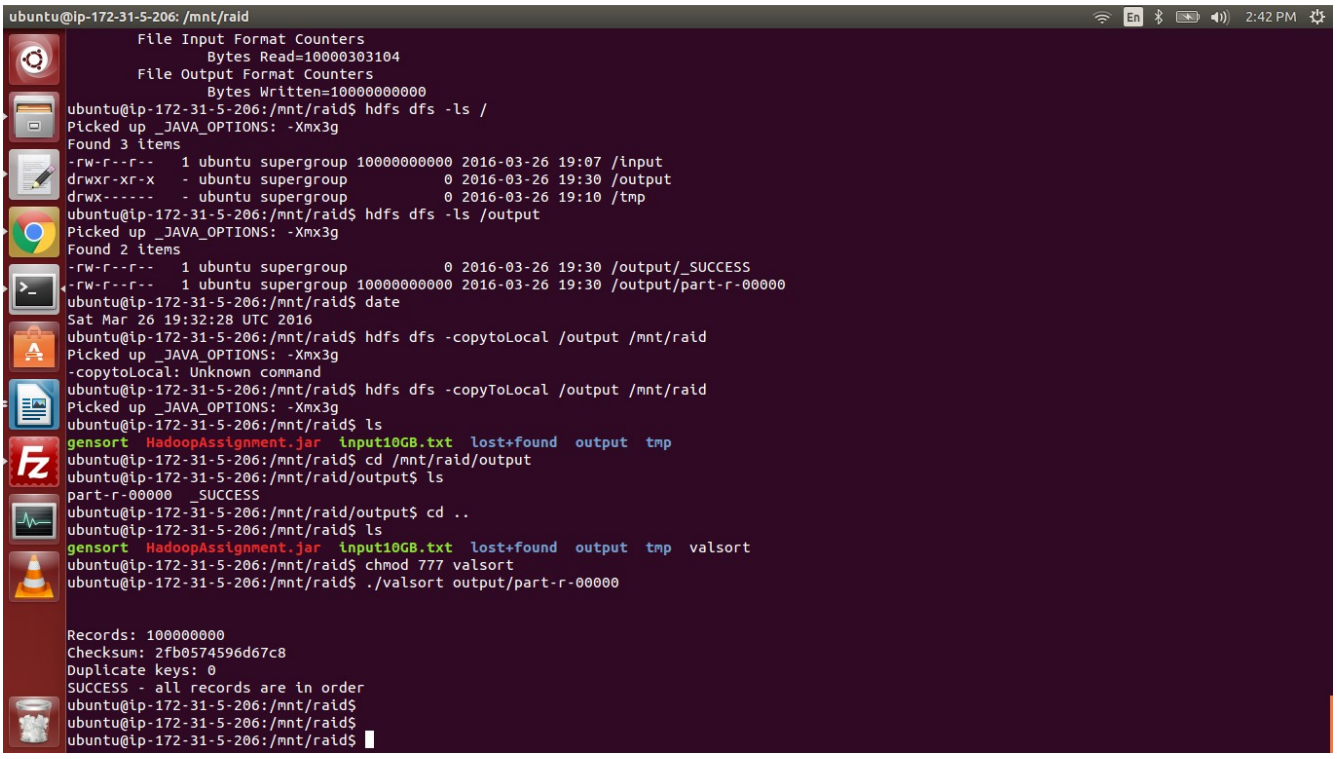
*running sorting application - time to be noted 19:10:21*

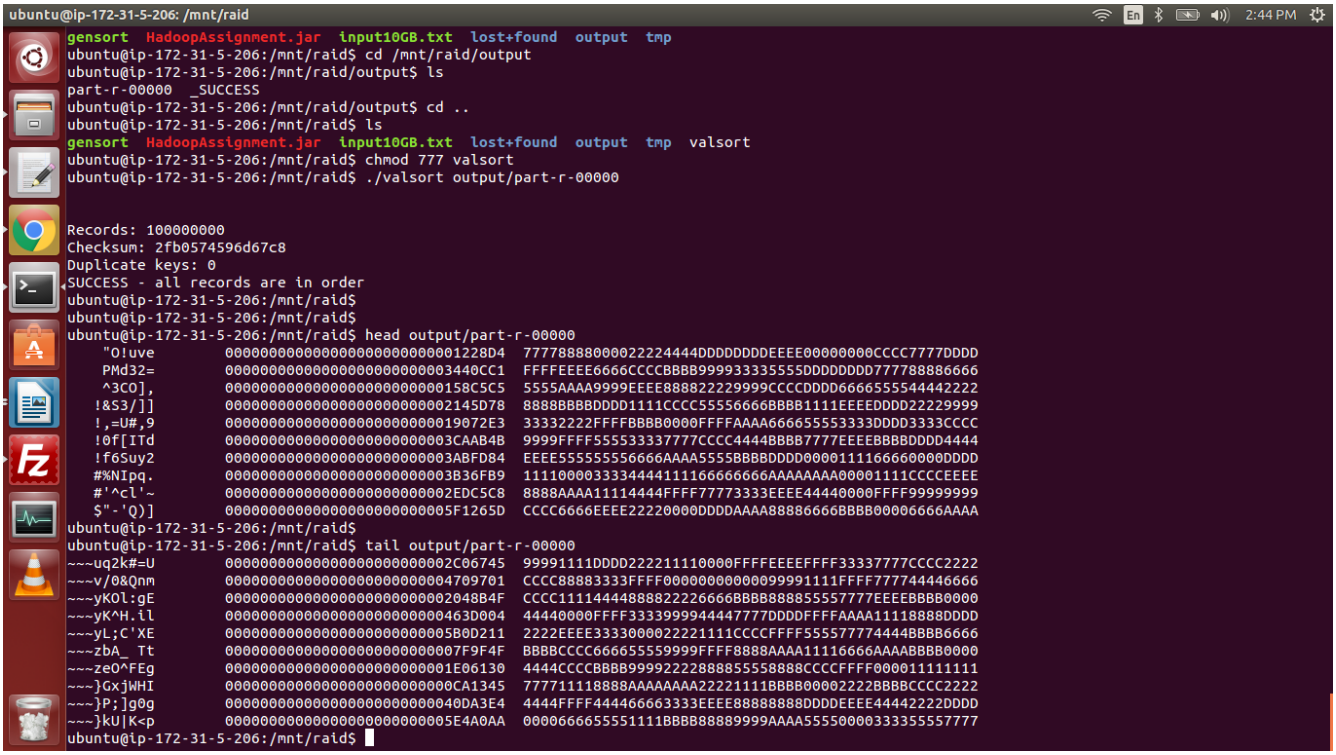





time to be noted above in creation of output file – 19:30

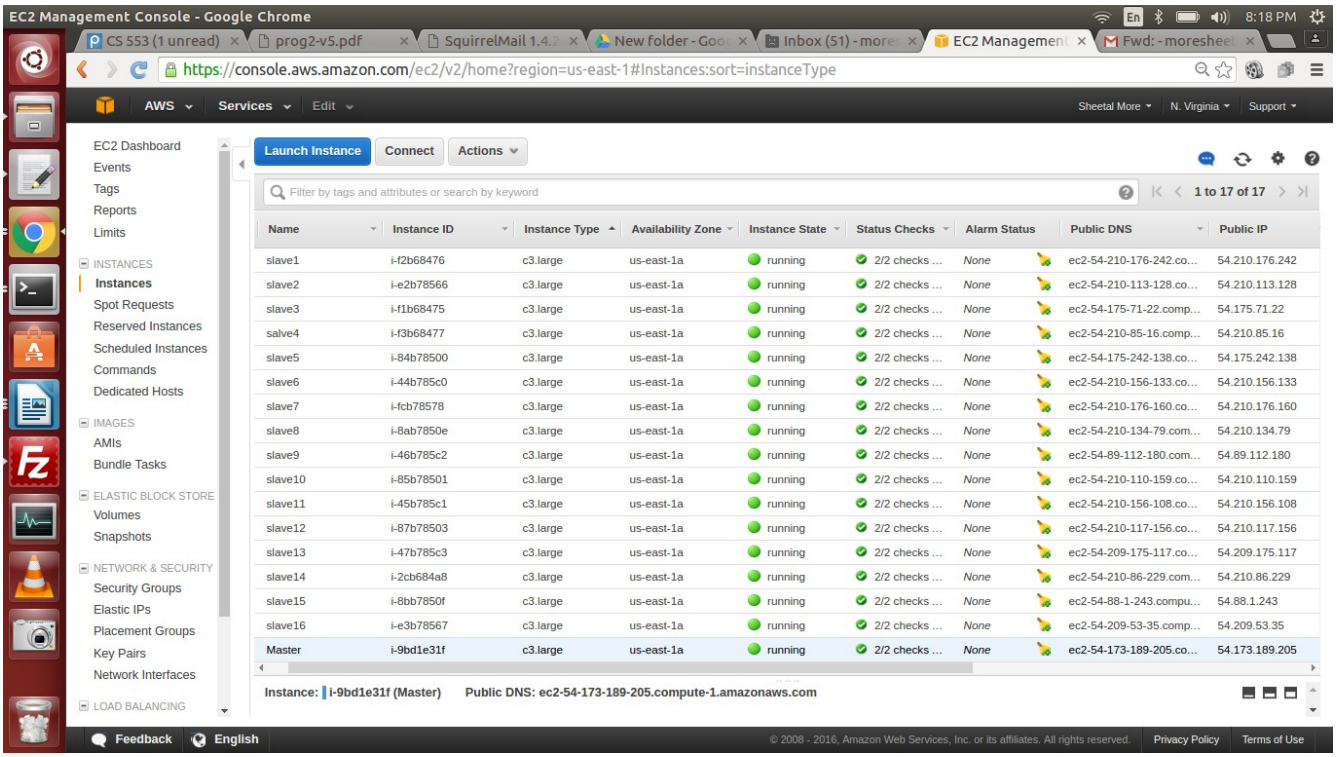


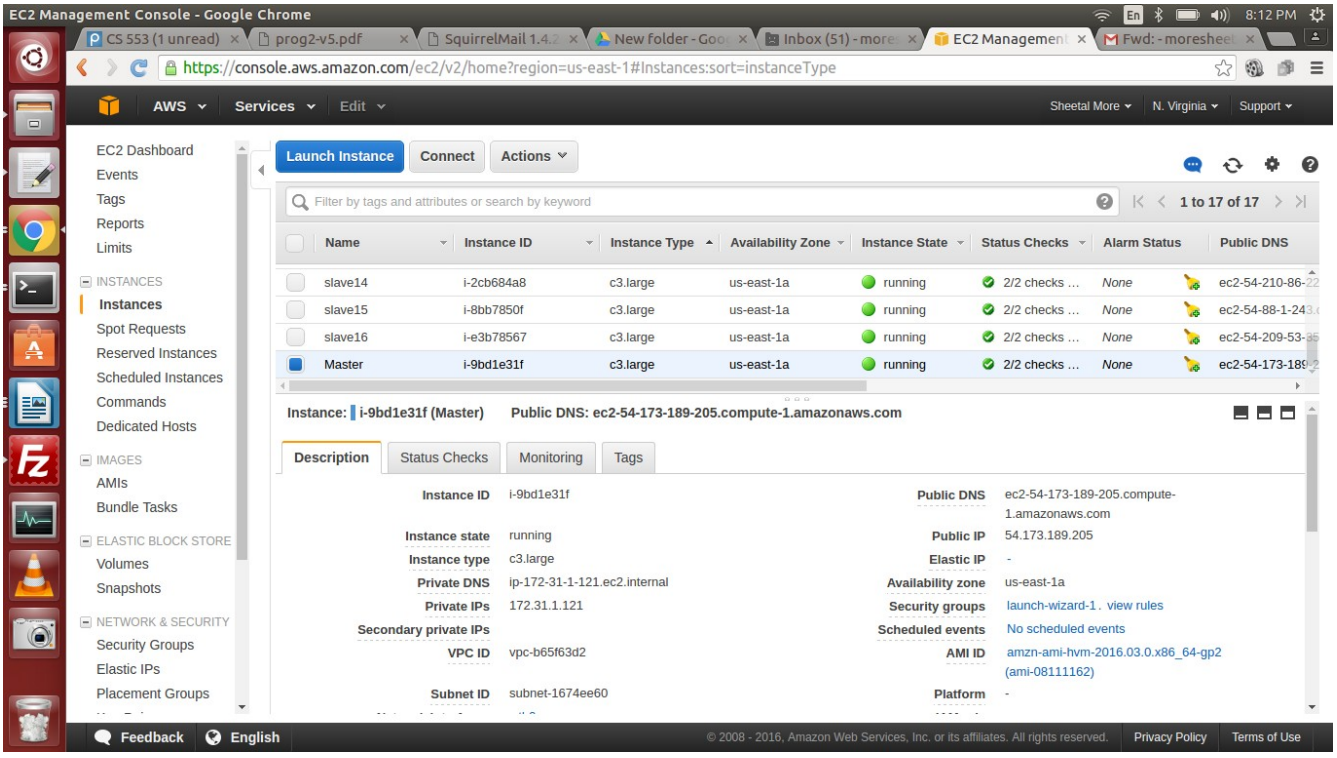


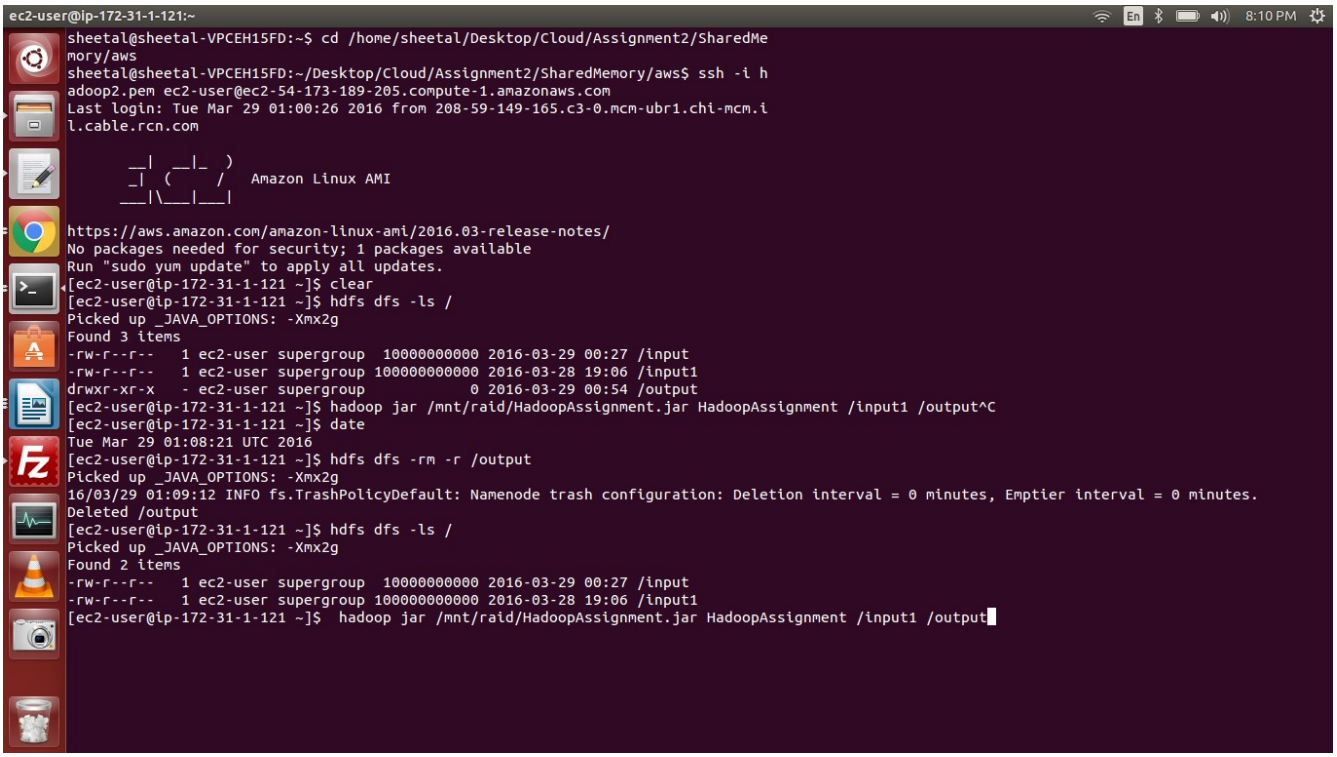

| *Total time taken = 1179 secs* |
|---|

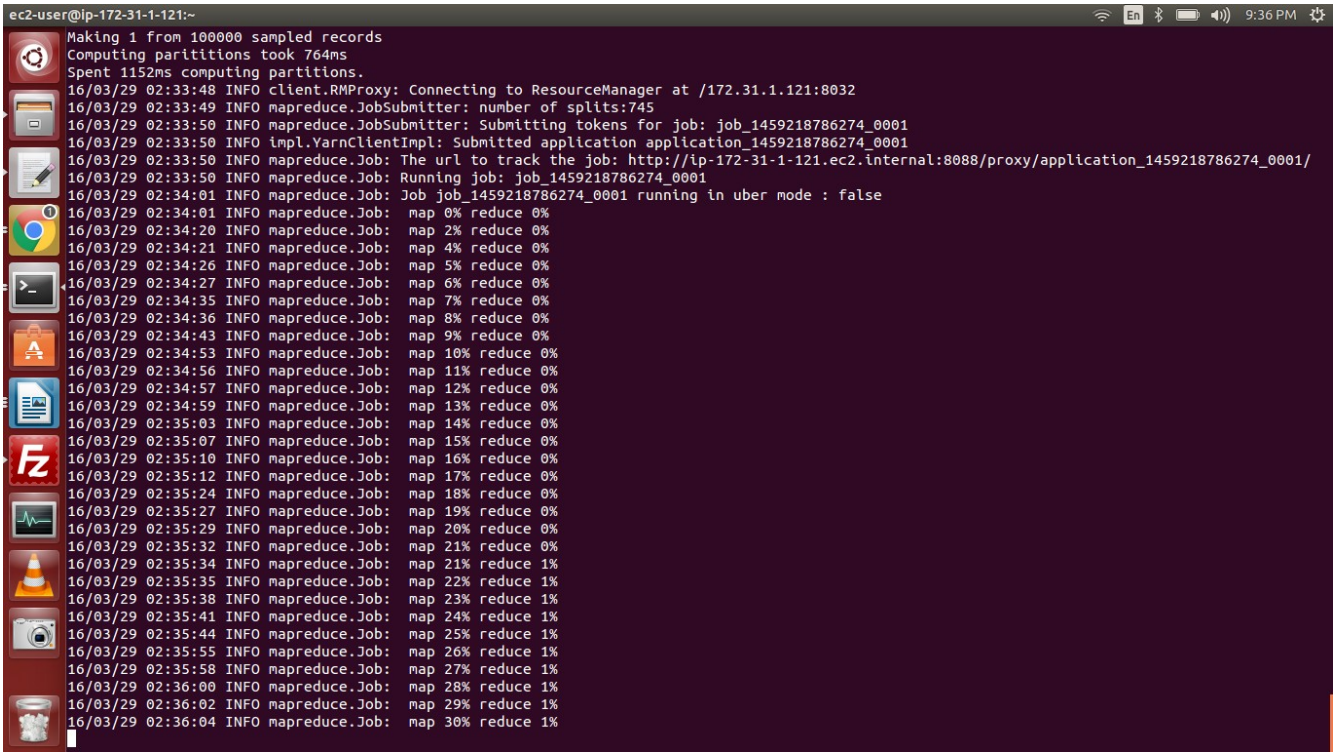
# Hadoop Multinode

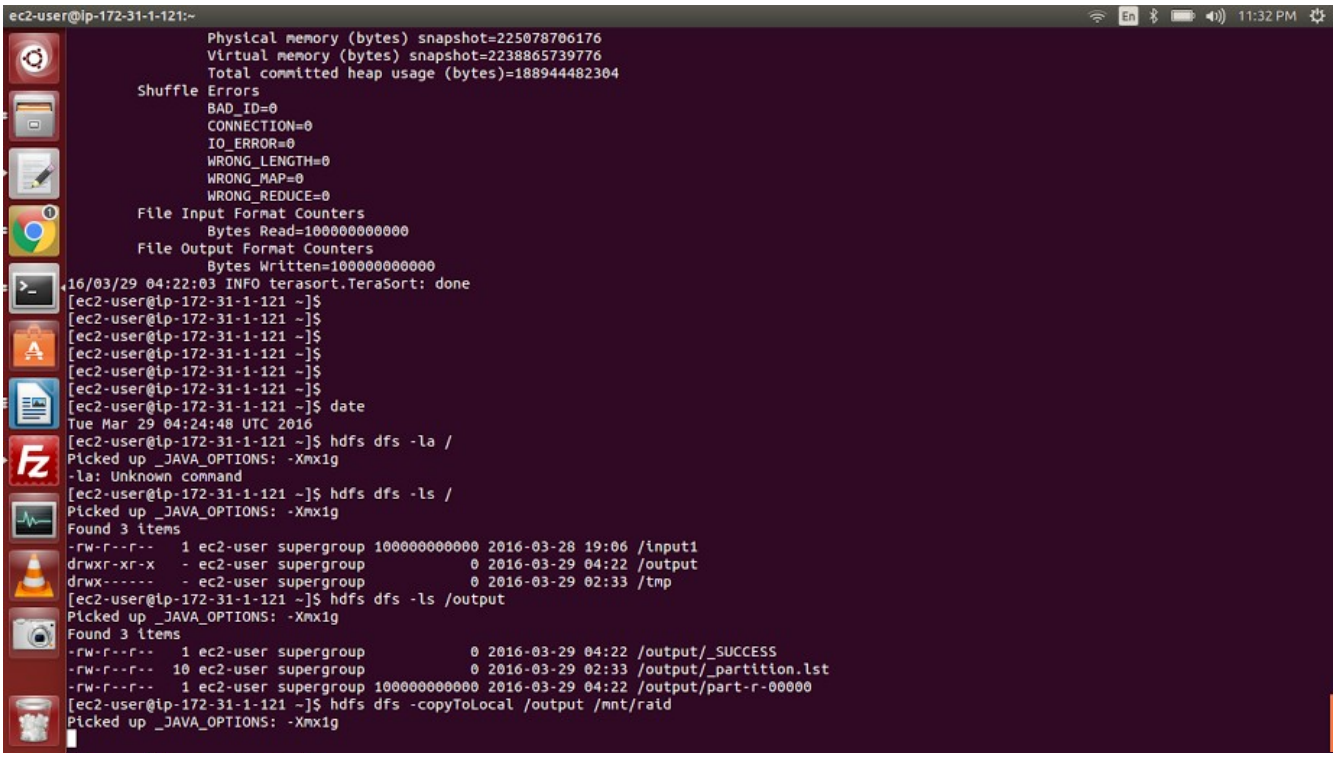
## Experiment: 100 GB dataset on cluster of 16 nodes of c3.large instances

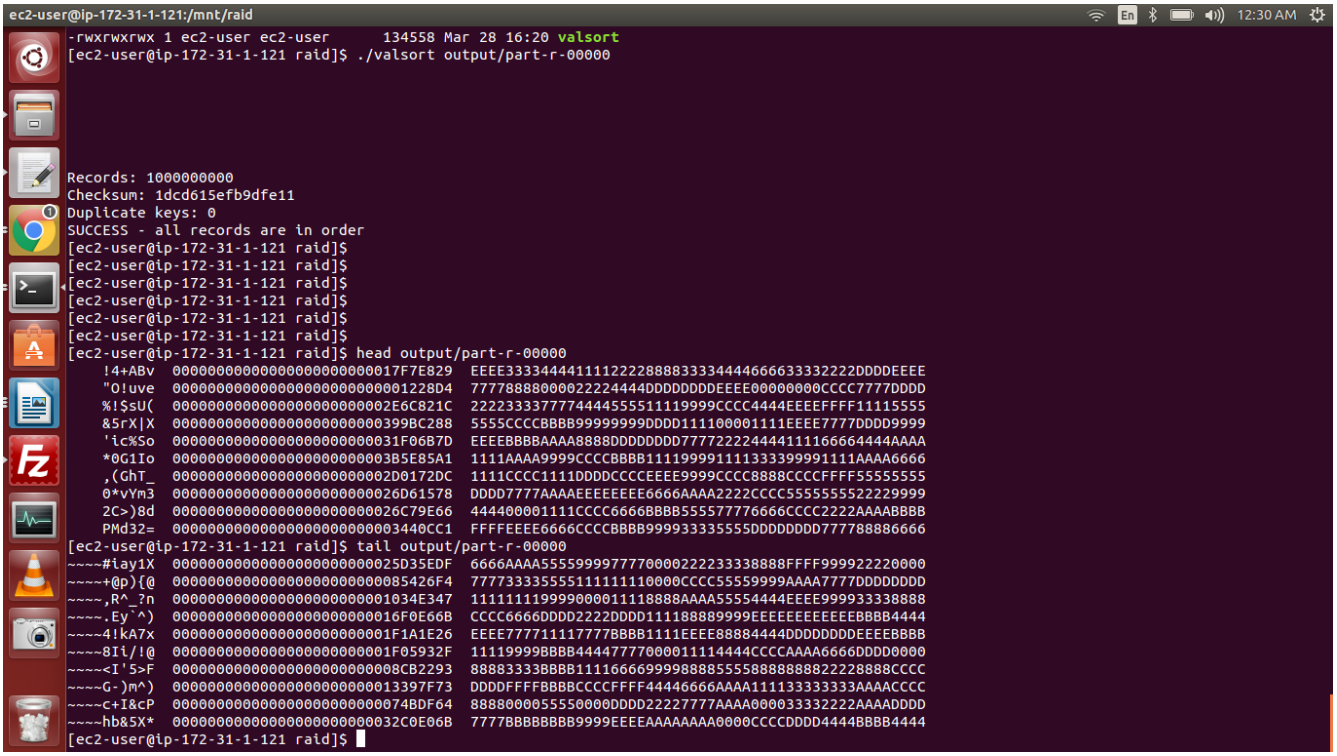



ec2-user@ip-172-31-1-121:~
8:10 PM
sheetal@sheetal-VPCEH15FD:~$ cd /home/sheetal/Desktop/Cloud/Assignment2/SharedMe
mory/aws
sheetal@sheetal-VPCEH15FD:~/Desktop/Cloud/Assignment2/SharedMemory/aws$ ssh -i h
adoop2.pem ec2-user@ec2-54-173-189-205.compute-1.amazonaws.com
Last login: Tue Mar 29 01:00:26 2016 from 208-59-149-165.c3-0.mcm-ubr1.chi-mcm.i
l.cable.rcn.com
__| __|_ )
_| ( / Amazon Linux AMI
___|___|___|
https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/
No packages needed for security; 1 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-1-121 ~]$ clear
[ec2-user@ip-172-31-1-121 ~]$ hdfs dfs -ls /
Picked up _JAVA_OPTIONS: -Xmx2g
Found 3 items
-rw-r--r-- 1 ec2-user supergroup 10000000000 2016-03-29 00:27 /input
-rw-r--r-- 1 ec2-user supergroup 100000000000 2016-03-28 19:06 /input1
drwxr-xr-x - ec2-user supergroup 0 2016-03-29 00:54 /output
[ec2-user@ip-172-31-1-121 ~]$ hadoop jar /mnt/raid/HadoopAssignment.jar HadoopAssignment /input1 /output^C
[ec2-user@ip-172-31-1-121 ~]$ date
Tue Mar 29 01:08:21 UTC 2016
[ec2-user@ip-172-31-1-121 ~]$ hdfs dfs -rm -r /output
Picked up _JAVA_OPTIONS: -Xmx2g
16/03/29 01:09:12 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /output
[ec2-user@ip-172-31-1-121 ~]$ hdfs dfs -ls /
Picked up _JAVA_OPTIONS: -Xmx2g
Found 2 items
-rw-r--r-- 1 ec2-user supergroup 10000000000 2016-03-29 00:27 /input
-rw-r--r-- 1 ec2-user supergroup 100000000000 2016-03-28 19:06 /input1
[ec2-user@ip-172-31-1-121 ~]$ hadoop jar /mnt/raid/HadoopAssignment.jar HadoopAssignment /input1 /output


ec2-user@ip-172-31-1-121:~
9:36 PM
Making 1 from 100000 sampled records
Computing parititions took 764ms
Spent 1152ms computing partitions.
16/03/29 02:33:48 INFO client.RMProxy: Connecting to ResourceManager at /172.31.1.121:8032
16/03/29 02:33:49 INFO mapreduce.JobSubmitter: number of splits:745
16/03/29 02:33:50 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1459218786274_0001
16/03/29 02:33:50 INFO impl.YarnClientImpl: Submitted application application_1459218786274_0001
16/03/29 02:33:50 INFO mapreduce.Job: The url to track the job: http://ip-172-31-1-121.ec2.internal:8088/proxy/application_1459218786274_0001/
16/03/29 02:33:50 INFO mapreduce.Job: Running job: job_1459218786274_0001
16/03/29 02:34:01 INFO mapreduce.Job: Job job_1459218786274_0001 running in uber mode : false
16/03/29 02:34:01 INFO mapreduce.Job: map 0% reduce 0%
16/03/29 02:34:20 INFO mapreduce.Job: map 2% reduce 0%
16/03/29 02:34:21 INFO mapreduce.Job: map 4% reduce 0%
16/03/29 02:34:26 INFO mapreduce.Job: map 5% reduce 0%
16/03/29 02:34:27 INFO mapreduce.Job: map 6% reduce 0%
16/03/29 02:34:35 INFO mapreduce.Job: map 7% reduce 0%
16/03/29 02:34:36 INFO mapreduce.Job: map 8% reduce 0%
16/03/29 02:34:43 INFO mapreduce.Job: map 9% reduce 0%
16/03/29 02:34:53 INFO mapreduce.Job: map 10% reduce 0%
16/03/29 02:34:56 INFO mapreduce.Job: map 11% reduce 0%
16/03/29 02:34:57 INFO mapreduce.Job: map 12% reduce 0%
16/03/29 02:34:59 INFO mapreduce.Job: map 13% reduce 0%
16/03/29 02:35:03 INFO mapreduce.Job: map 14% reduce 0%
16/03/29 02:35:07 INFO mapreduce.Job: map 15% reduce 0%
16/03/29 02:35:10 INFO mapreduce.Job: map 16% reduce 0%
16/03/29 02:35:12 INFO mapreduce.Job: map 17% reduce 0%
16/03/29 02:35:24 INFO mapreduce.Job: map 18% reduce 0%
16/03/29 02:35:27 INFO mapreduce.Job: map 19% reduce 0%
16/03/29 02:35:29 INFO mapreduce.Job: map 20% reduce 0%
16/03/29 02:35:32 INFO mapreduce.Job: map 21% reduce 0%
16/03/29 02:35:34 INFO mapreduce.Job: map 21% reduce 1%
16/03/29 02:35:35 INFO mapreduce.Job: map 22% reduce 1%
16/03/29 02:35:38 INFO mapreduce.Job: map 23% reduce 1%
16/03/29 02:35:41 INFO mapreduce.Job: map 24% reduce 1%
16/03/29 02:35:44 INFO mapreduce.Job: map 25% reduce 1%
16/03/29 02:35:55 INFO mapreduce.Job: map 26% reduce 1%
16/03/29 02:35:58 INFO mapreduce.Job: map 27% reduce 1%
16/03/29 02:36:00 INFO mapreduce.Job: map 28% reduce 1%
16/03/29 02:36:02 INFO mapreduce.Job: map 29% reduce 1%
16/03/29 02:36:04 INFO mapreduce.Job: map 30% reduce 1%




As observed in above experiment screenshot number 4, the application started running at time 2:33:48 and as seen screenshot 6, the output file was created at 4:22:00, thus we can say the the time required for running th experiment was 5952 secs (1 hour 48 mins 12 secs).

| ***Total time taken = 6492 secs*** |
|---|

Spark

### 3.1. Introduction

**Problem Statement:**

The goal of this experiment is to sort a large dataset which contains one record per line of 100 bytes and we need to sort them on the basis of ASCII values of first 10 bytes. If we use the traditional methodology of sorting then it would not only require higher space, but will also take higher time.

**Methodology:**

The spark framework runs on top of hadoop framework and that it runs much faster

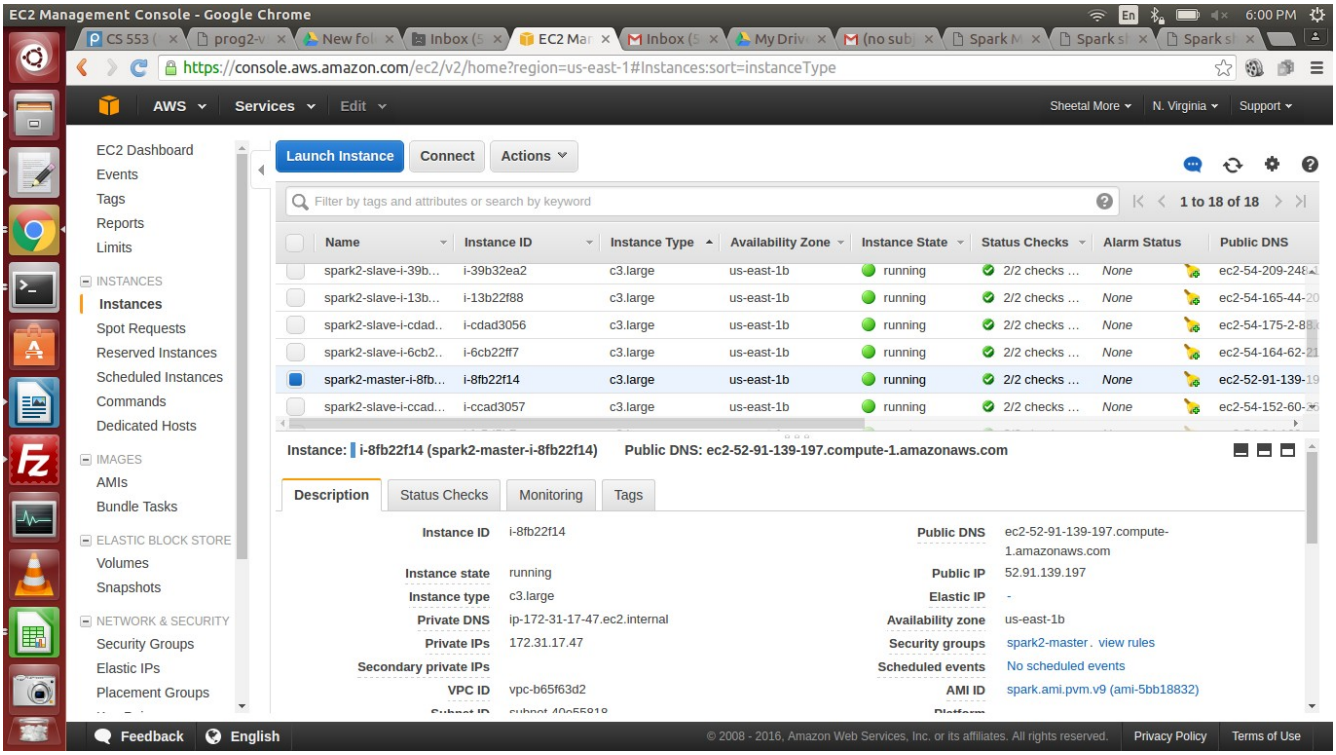
**Environment Settings:**

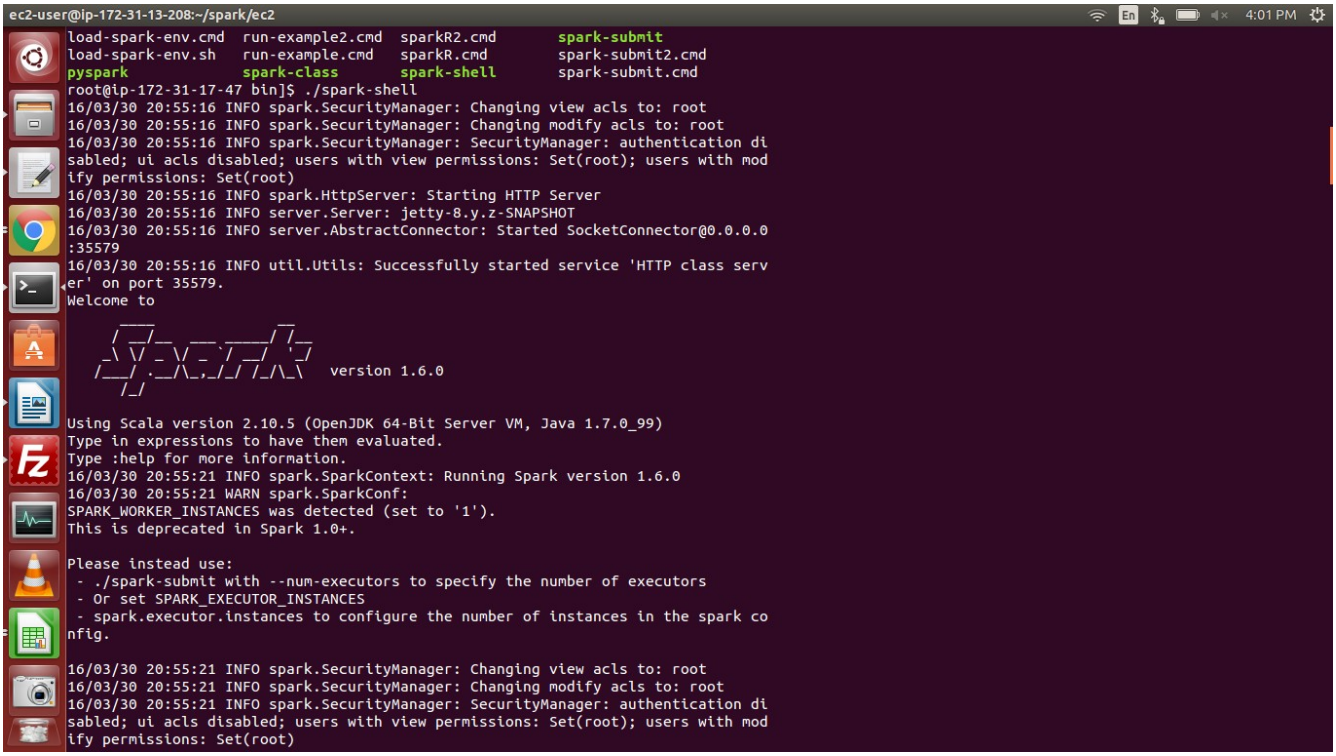
| | |
|---|---|
| Type of EC2 instance used | c3.large |
| Region | N. Virginia |
| OS and kernel | Red Hat kernel-3.4.37-40.44.amzn1.x86_64 |
| Implementation Language | Scala |
| JAVA version | Open JDK 1.7.0_99 |
| Hadoop Version | Hadoop 1.0. |
| Spark Version | Spark 1.6.0 |
| Scala version | Scala 2.10.3 |

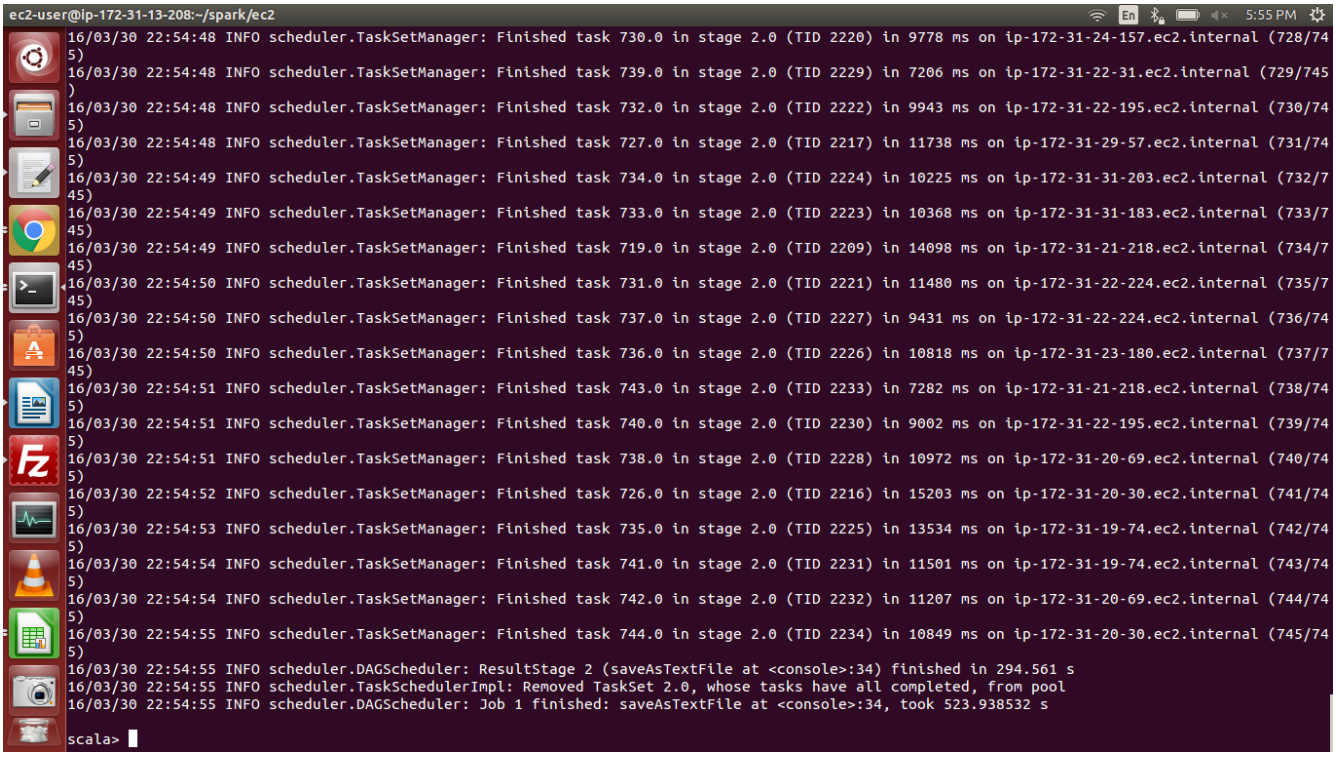
**Installation steps:**

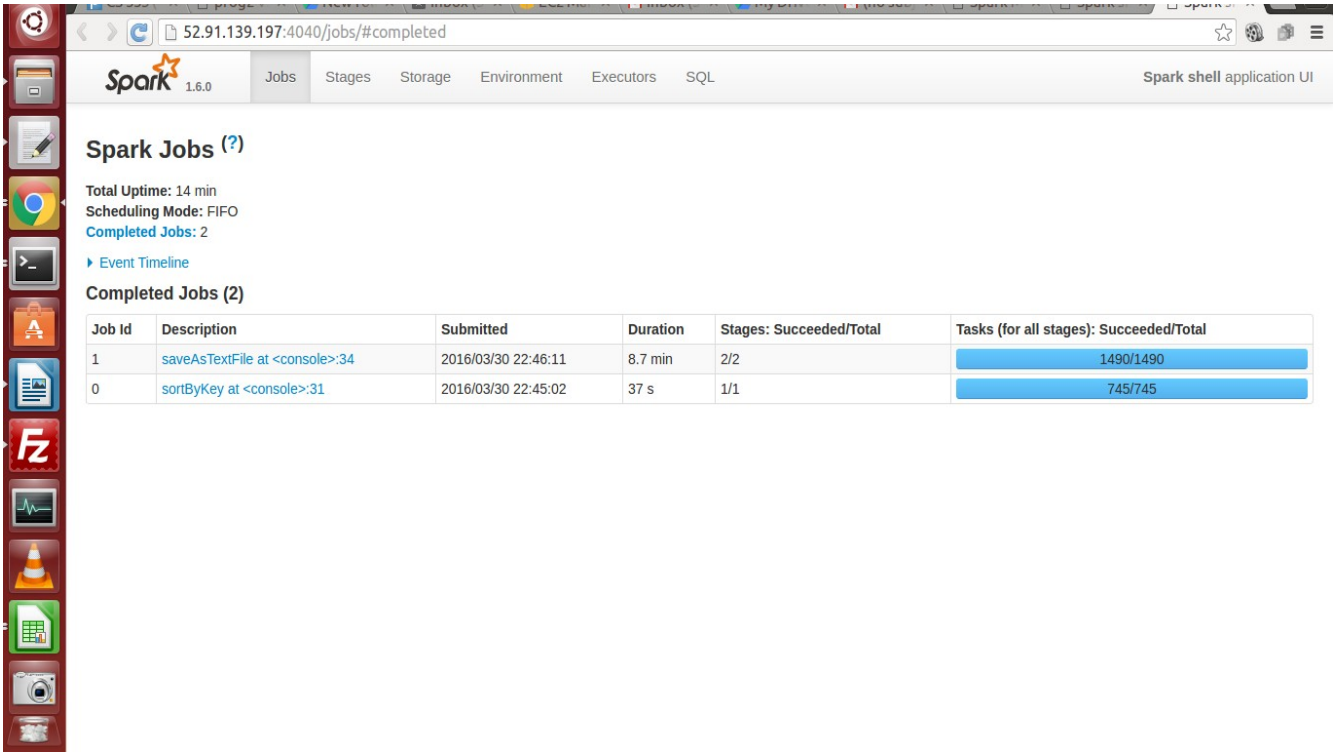
1. Create any instance on aws.
2. Download Java, Spark, Scala, Ganglia on that instance
3. Upload the pem file to instance and five permissions "chmod 400 hadoop2.pem"
4. Run export commands:
    export AWS_ACCESS_KEY_ID=<your key ID>
    export AWS_SECRET_ACCESS_KEY=<your key>
5. Edit security group to add rule for All TCP and ALL traffic
6. run the command to launch spark:
./spark-ec2 -k hadoop2 -i hadoop2.pem --region=us-east-1 -s 16 --instance-type=c3.large --ebs-vol-size=100 -m c3.large --ebs-vol-size=400 --spot-price=0.023 launch spark2
7. once cluster is created, login to spark master using command:
./spark-ec2 -k hadoop2 -i hadoop2.pem --region=us-east-1 login spark2
8.generate the input file using gensort and transfer it to hdfs
9. run the spark-shell command and then spark shell is opened
10. Here we need to run the scala code by typing commands one after other as given in source code
11. once the output file is generated, it can be transferred to local and verify the result using valsort

## 3.2 Experiment: 100 GB

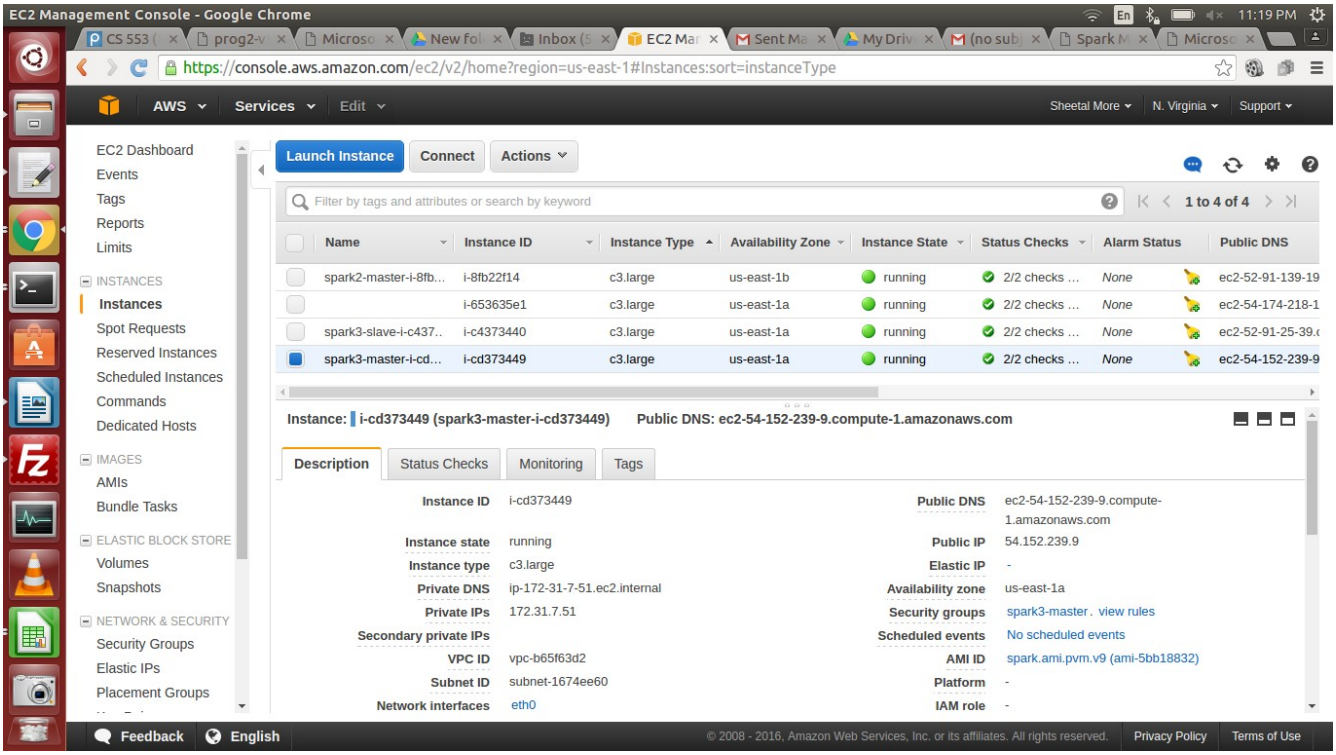


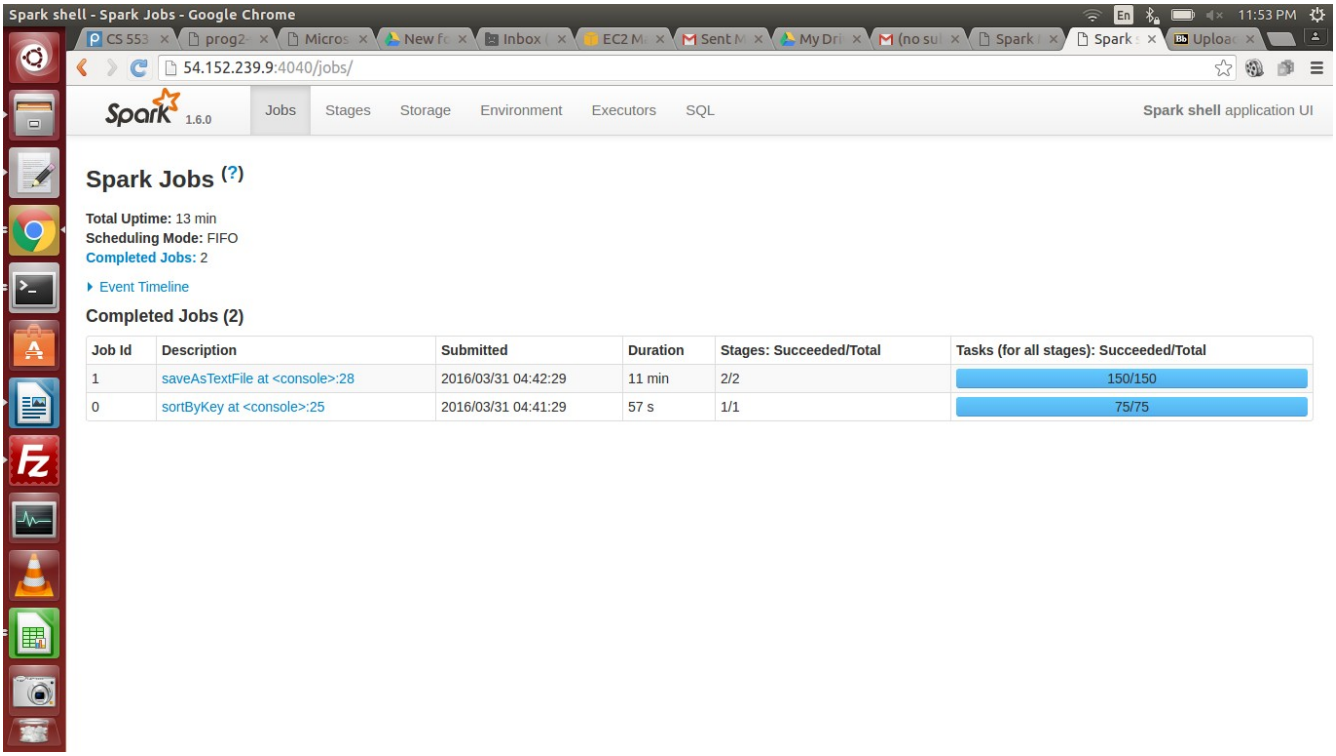



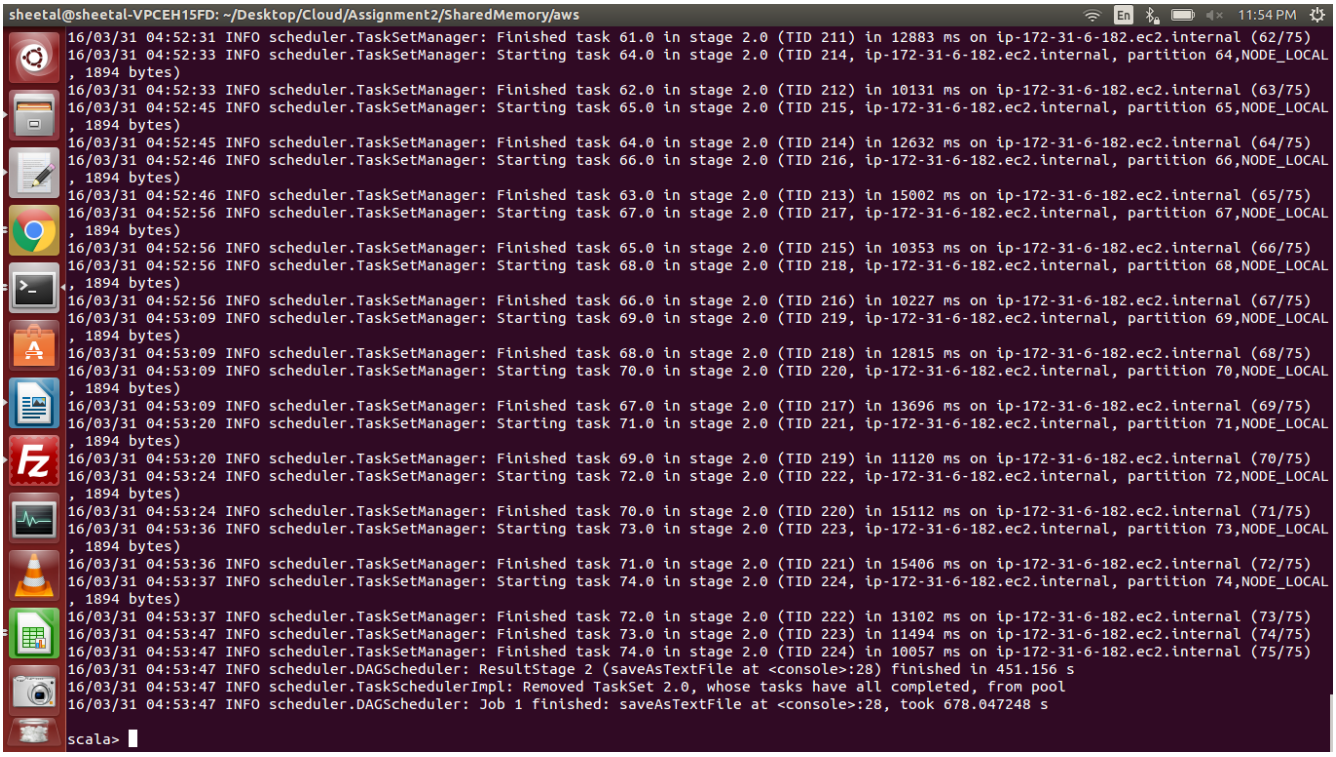


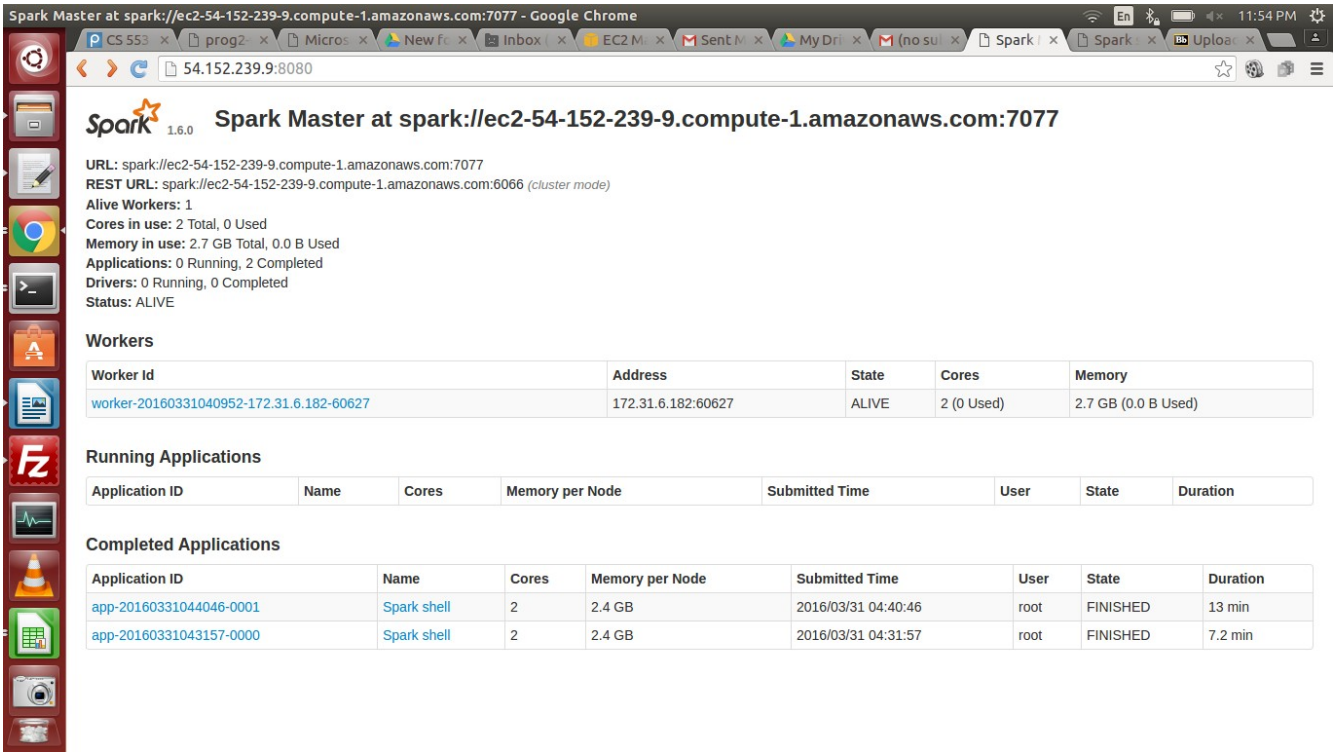




## 3.3 Experiment: 10GB

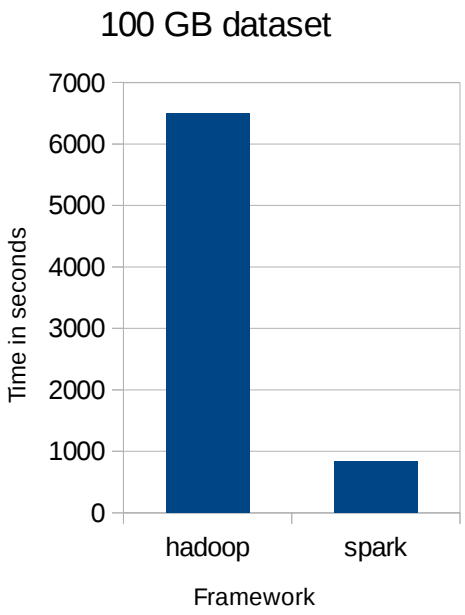

EC2 Management Console - Google Chrome
https://console.aws.amazon.com/ec2/v2/home?region=us-east-1#Instances:sort=instanceType
AWS
Services
Edit
Launch Instance
Connect
Actions
EC2 Dashboard
Events
Tags
Reports
Limits
INSTANCES
Instances
Spot Requests
Reserved Instances
Scheduled Instances
Commands
Dedicated Hosts
IMAGES
AMIs
Bundle Tasks
ELASTIC BLOCK STORE
Volumes
Snapshots
NETWORK & SECURITY
Security Groups
Elastic IPs
Placement Groups
Filter by tags and attributes or search by keyword
1 to 4 of 4
Name
Instance ID
Instance Type
Availability Zone
Instance State
Status Checks
Alarm Status
Public DNS
spark2-master-i-8fb...
i-8fb22f14
c3.large
us-east-1b
running
2/2 checks ...
None
ec2-52-91-139-19
i-653635e1
c3.large
us-east-1a
running
2/2 checks ...
None
ec2-54-174-218-1
spark3-slave-i-c437..
i-c4373440
c3.large
us-east-1a
running
2/2 checks ...
None
ec2-52-91-25-39.
spark3-master-i-cd...
i-cd373449
c3.large
us-east-1a
running
2/2 checks ...
None
ec2-54-152-239-9
Instance: i-cd373449 (spark3-master-i-cd373449)
Public DNS: ec2-54-152-239-9.compute-1.amazonaws.com
Description
Status Checks
Monitoring
Tags
Instance ID
i-cd373449
Instance state
running
Instance type
c3.large
Private DNS
ip-172-31-7-51.ec2.internal
Private IPs
172.31.7.51
Secondary private IPs
VPC ID
vpc-b65f63d2
Subnet ID
subnet-1674ee60
Network interfaces
eth0
Public DNS
ec2-54-152-239-9.compute-1.amazonaws.com
Public IP
54.152.239.9
Elastic IP
-
Availability zone
us-east-1a
Security groups
spark3-master. view rules
Scheduled events
No scheduled events
AMI ID
spark.ami.pvm.v9 (ami-5bb18832)
Platform
-
IAM role
-
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

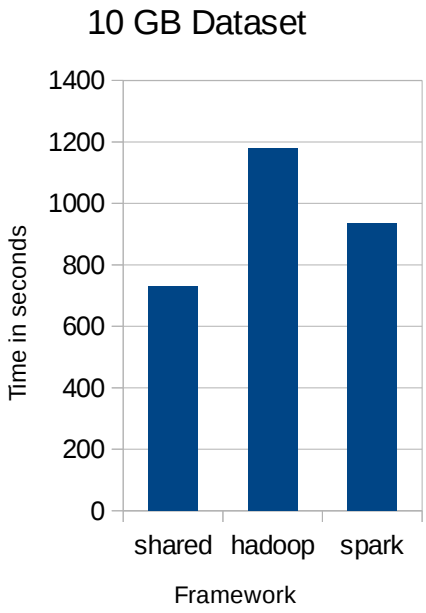

Spark shell - Spark Jobs - Google Chrome
54.152.239.9:4040/jobs/
Spark 1.6.0
Jobs
Stages
Storage
Environment
Executors
SQL
Spark shell application UI
Spark Jobs (?)
Total Uptime: 13 min
Scheduling Mode: FIFO
Completed Jobs: 2
Event Timeline
Completed Jobs (2)
Job Id
Description
Submitted
Duration
Stages: Succeeded/Total
Tasks (for all stages): Succeeded/Total
1
saveAsTextFile at <console>:28
2016/03/31 04:42:29
11 min
2/2
150/150
0
sortByKey at <console>:25
2016/03/31 04:41:29
57 s
1/1
75/75

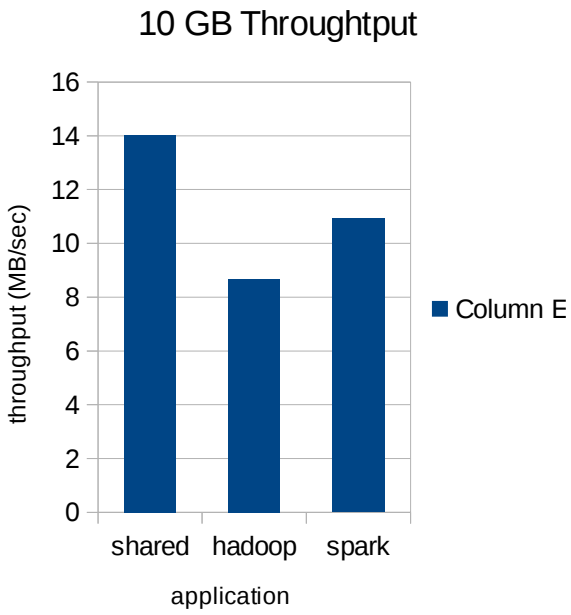

sheetal@sheetal-VPCEH15FD: ~/Desktop/Cloud/Assignment2/SharedMemory/aws
11:54 PM
16/03/31 04:52:31 INFO scheduler.TaskSetManager: Finished task 61.0 in stage 2.0 (TID 211) in 12883 ms on ip-172-31-6-182.ec2.internal (62/75)
16/03/31 04:52:33 INFO scheduler.TaskSetManager: Starting task 64.0 in stage 2.0 (TID 214, ip-172-31-6-182.ec2.internal, partition 64,NODE_LOCAL, 1894 bytes)
16/03/31 04:52:33 INFO scheduler.TaskSetManager: Finished task 62.0 in stage 2.0 (TID 212) in 10131 ms on ip-172-31-6-182.ec2.internal (63/75)
16/03/31 04:52:45 INFO scheduler.TaskSetManager: Starting task 65.0 in stage 2.0 (TID 215, ip-172-31-6-182.ec2.internal, partition 65,NODE_LOCAL, 1894 bytes)
16/03/31 04:52:45 INFO scheduler.TaskSetManager: Finished task 64.0 in stage 2.0 (TID 214) in 12632 ms on ip-172-31-6-182.ec2.internal (64/75)
16/03/31 04:52:46 INFO scheduler.TaskSetManager: Starting task 66.0 in stage 2.0 (TID 216, ip-172-31-6-182.ec2.internal, partition 66,NODE_LOCAL, 1894 bytes)
16/03/31 04:52:46 INFO scheduler.TaskSetManager: Finished task 63.0 in stage 2.0 (TID 213) in 15002 ms on ip-172-31-6-182.ec2.internal (65/75)
16/03/31 04:52:56 INFO scheduler.TaskSetManager: Starting task 67.0 in stage 2.0 (TID 217, ip-172-31-6-182.ec2.internal, partition 67,NODE_LOCAL, 1894 bytes)
16/03/31 04:52:56 INFO scheduler.TaskSetManager: Finished task 65.0 in stage 2.0 (TID 215) in 10353 ms on ip-172-31-6-182.ec2.internal (66/75)
16/03/31 04:52:56 INFO scheduler.TaskSetManager: Starting task 68.0 in stage 2.0 (TID 218, ip-172-31-6-182.ec2.internal, partition 68,NODE_LOCAL, 1894 bytes)
16/03/31 04:52:56 INFO scheduler.TaskSetManager: Finished task 66.0 in stage 2.0 (TID 216) in 10227 ms on ip-172-31-6-182.ec2.internal (67/75)
16/03/31 04:53:09 INFO scheduler.TaskSetManager: Starting task 69.0 in stage 2.0 (TID 219, ip-172-31-6-182.ec2.internal, partition 69,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:09 INFO scheduler.TaskSetManager: Finished task 68.0 in stage 2.0 (TID 218) in 12815 ms on ip-172-31-6-182.ec2.internal (68/75)
16/03/31 04:53:09 INFO scheduler.TaskSetManager: Starting task 70.0 in stage 2.0 (TID 220, ip-172-31-6-182.ec2.internal, partition 70,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:09 INFO scheduler.TaskSetManager: Finished task 67.0 in stage 2.0 (TID 217) in 13696 ms on ip-172-31-6-182.ec2.internal (69/75)
16/03/31 04:53:20 INFO scheduler.TaskSetManager: Starting task 71.0 in stage 2.0 (TID 221, ip-172-31-6-182.ec2.internal, partition 71,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:20 INFO scheduler.TaskSetManager: Finished task 69.0 in stage 2.0 (TID 219) in 11120 ms on ip-172-31-6-182.ec2.internal (70/75)
16/03/31 04:53:24 INFO scheduler.TaskSetManager: Starting task 72.0 in stage 2.0 (TID 222, ip-172-31-6-182.ec2.internal, partition 72,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:24 INFO scheduler.TaskSetManager: Finished task 70.0 in stage 2.0 (TID 220) in 15112 ms on ip-172-31-6-182.ec2.internal (71/75)
16/03/31 04:53:36 INFO scheduler.TaskSetManager: Starting task 73.0 in stage 2.0 (TID 223, ip-172-31-6-182.ec2.internal, partition 73,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:36 INFO scheduler.TaskSetManager: Finished task 71.0 in stage 2.0 (TID 221) in 15406 ms on ip-172-31-6-182.ec2.internal (72/75)
16/03/31 04:53:37 INFO scheduler.TaskSetManager: Starting task 74.0 in stage 2.0 (TID 224, ip-172-31-6-182.ec2.internal, partition 74,NODE_LOCAL, 1894 bytes)
16/03/31 04:53:37 INFO scheduler.TaskSetManager: Finished task 72.0 in stage 2.0 (TID 222) in 13102 ms on ip-172-31-6-182.ec2.internal (73/75)
16/03/31 04:53:47 INFO scheduler.TaskSetManager: Finished task 73.0 in stage 2.0 (TID 223) in 11494 ms on ip-172-31-6-182.ec2.internal (74/75)
16/03/31 04:53:47 INFO scheduler.TaskSetManager: Finished task 74.0 in stage 2.0 (TID 224) in 10057 ms on ip-172-31-6-182.ec2.internal (75/75)
16/03/31 04:53:47 INFO scheduler.DAGScheduler: ResultStage 2 (saveAsTextFile at <console>:28) finished in 451.156 s
16/03/31 04:53:47 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/03/31 04:53:47 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at <console>:28, took 678.047248 s
scala>

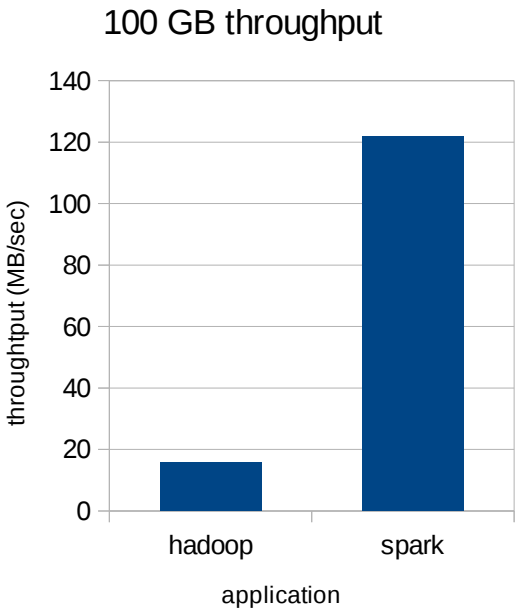

Spark Master at spark://ec2-54-152-239-9.compute-1.amazonaws.com:7077 - Google Chrome
11:54 PM
54.152.239.9:8080
Spark 1.6.0
Spark Master at spark://ec2-54-152-239-9.compute-1.amazonaws.com:7077
URL: spark://ec2-54-152-239-9.compute-1.amazonaws.com:7077
REST URL: spark://ec2-54-152-239-9.compute-1.amazonaws.com:6066 (cluster mode)
Alive Workers: 1
Cores in use: 2 Total, 0 Used
Memory in use: 2.7 GB Total, 0.0 B Used
Applications: 0 Running, 2 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers
Worker Id | Address | State | Cores | Memory
worker-20160331040952-172.31.6.182-60627 | 172.31.6.182:60627 | ALIVE | 2 (0 Used) | 2.7 GB (0.0 B Used)
Running Applications
Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration
Completed Applications
Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration
app-20160331044046-0001 | Spark shell | 2 | 2.4 GB | 2016/03/31 04:40:46 | root | FINISHED | 13 min
app-20160331043157-0000 | Spark shell | 2 | 2.4 GB | 2016/03/31 04:31:57 | root | FINISHED | 7.2 min

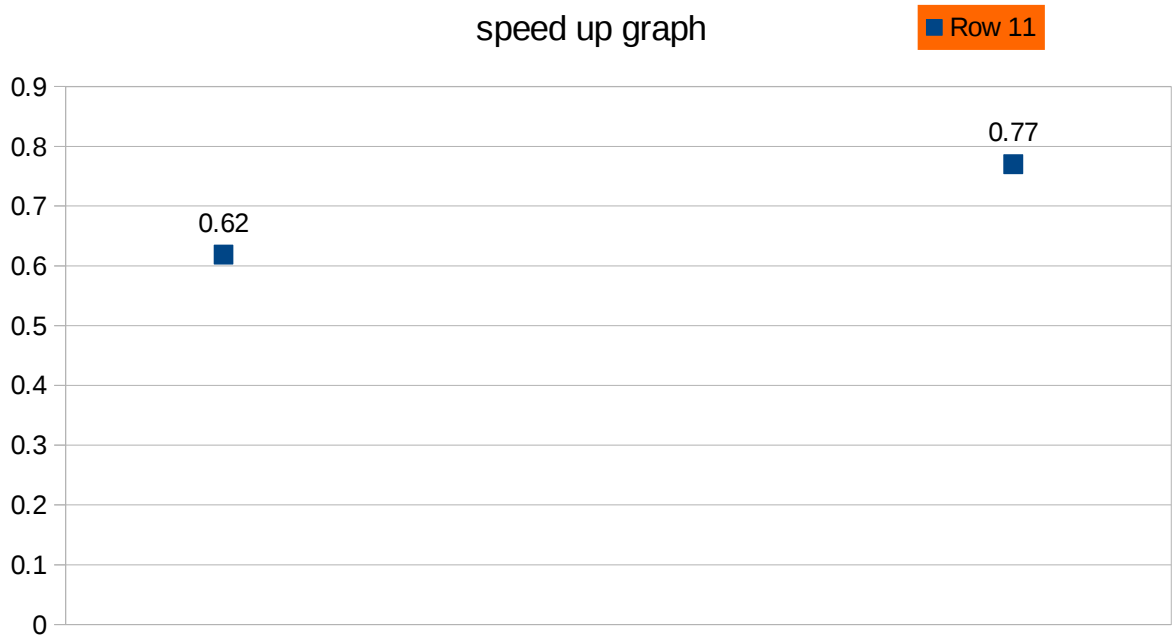
**Performance Evaluation**






**0.62 – hadoop, 0.77 -spark with respect to shared memory**

**Shared Memory, Spark and Hadoop on 10GB dataset:**

As seen from above graphs, we can say that for a 10GB dataset, shared memory has taken least time to sort. The reason for this is that the frameworks like hadoop and spark had higher overhead in distributing the data over nodes and that they are made for running on higher datasets. It is one of the limitations of hadoop that it works inefficiently on smaller datasets since the hdfs cannot read the small files in an efficient way.

**Spark and Hadoop comparison on 100GB dataset:**

- As observed in above graphs, the throughput for spark is much higher. The reason for this is that spark writes data directly to memory. This can be done using the RDDs and it prevents from slowing down of any process. A limited amount of time is spend in moving data in and out of disks.
- It can seen from graph that the throughput is almost 10 times more than that of hadoop.
- Also for recovering from failed nodes, spark does a much better job than hadoop just by using the RDDs.

**Comparison with benchmark:**

The benchmark has used a much better performing instance and the number of nodes were set to a higher value in thousands. As a result, they have achieved an outstanding performance. We on the other hand have used a c3.large instance having low memory and have run experiments on a smaller dataset

using only 16 nodes. Hadoop framework works better on a larger dataset with more number of datasets. Thus we can say that the comparison is not the same scale.

We can learn from the CloudSort benchmark that the reason for using external sort is that it is representative of many input output intensive operations. It is of advantage for using public cloud.