Assignment 2

Terasort on Amazon AWS

Team Members

Niket Patel (A20384264) npatel102@hawk.iit.edu

Purvank Patel (A20380792) ppatel104@hawk.iit.edu

Parth Hindia (A20384443) phindia@hawk.iit.edu

Introduction

Our aim for this assignment is to check performance of Terasort program using various approach such as Shared Memory External sort, Hadoop Terasort and Spark Terasort. We have generated 128GB and 1TB random datasets using gensort program and perform all the those three approach on amazon AWS i3.large, i3.4xlarge and 8 node i3.large cluster.

To perform the experiment we have used following environment setup on amazon AWS.

Operation System	Ubuntu 16.04
Java Version	OpenJDK-1.8.0
Hadoop Version	Hadoop-2.7.1
Spark Version	Spark-2.2.0

Configuration: 1

Virtual Cluster (1-Node, i3.large)

AWS I3.large configuration:

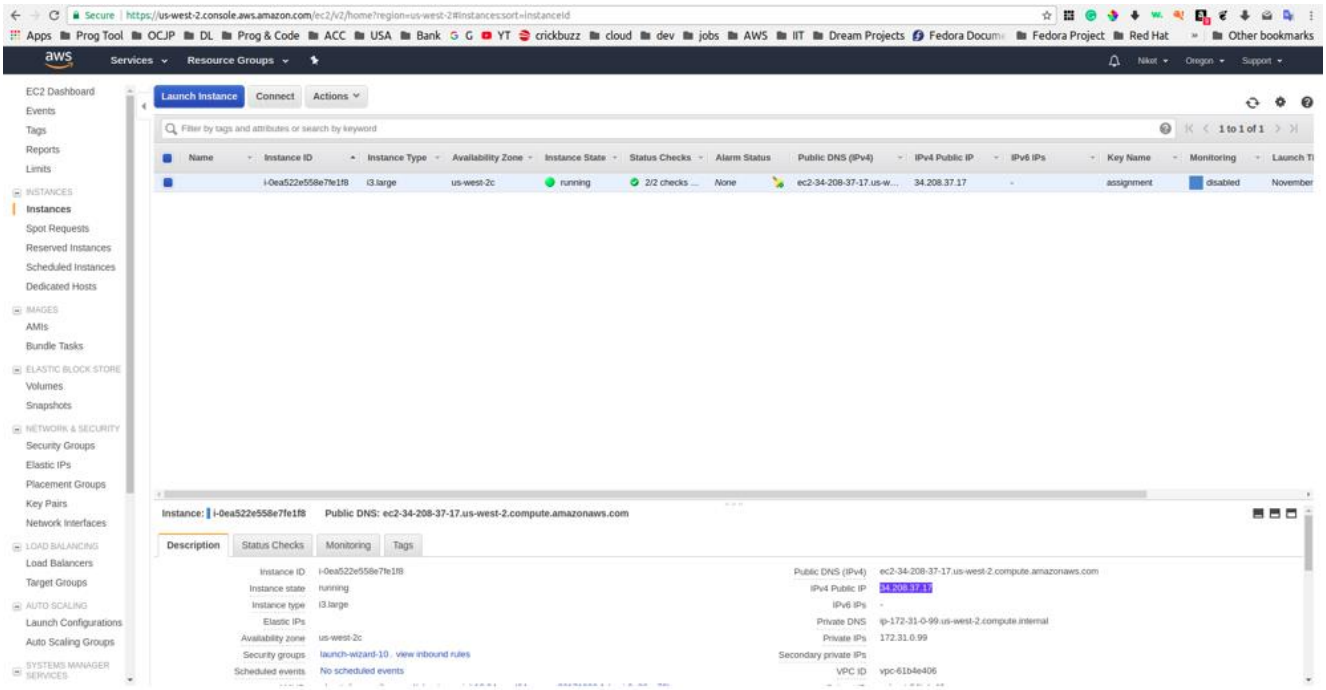
Instance Name	i3.large
vCPU count	2
Memory	15.25 GiB
Instance Storage (NVMe SSD)	0.475 TB
Price/Hour	$0.15
Operating System	Ubuntu 16.04.3 LTS

To set up the environment and run the program follow the following steps respectively. We also can use automation script to simplify the test.

a. Shared Memory Terasort:

To perform shared Memory Terasort for 128 GB on i3.large instance, we have used Merge Sort as Main Sorting algorithm.
To perform External Sort follow following steps.

1. Create i3.large instance on Amazon AWS.



2. Upload ExternalSort.java file to the instance using following command.

```
$ scp -i assignment.pem ExternalSort.java ubuntu@ec2-52-40-13-239.us-west-2.compute.amazonaws.com
```

3. Generate 128GB file using following command.

```
$ cd /data
```

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

```
$ tar xvzf gensort-linux-1.5.tar.gz
```

```
$ cd 64/
```

```
$ ./gensort -a -t8 1374389534 sample128gb.txt
```

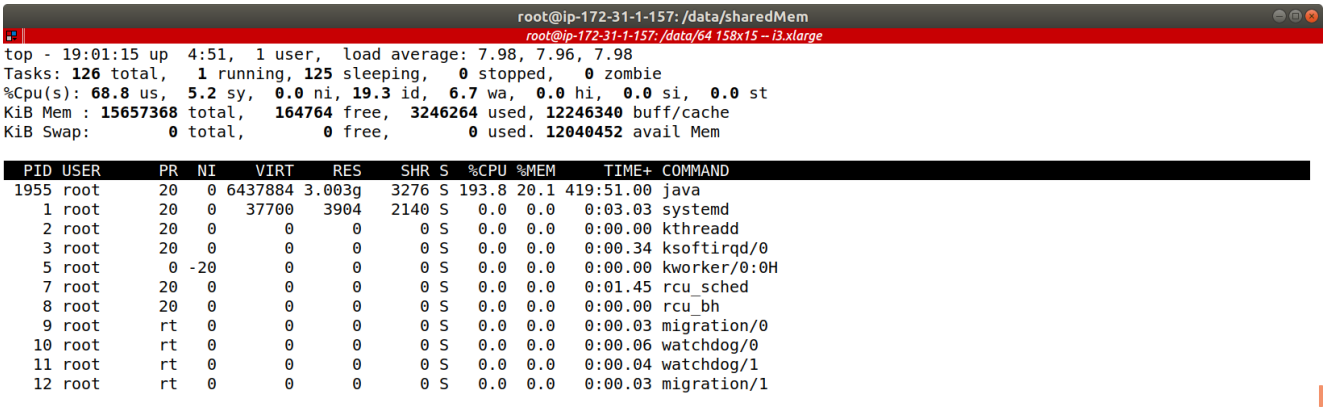
4. Now also create two directory in the same directory. Stmp and tmp directory using following command.

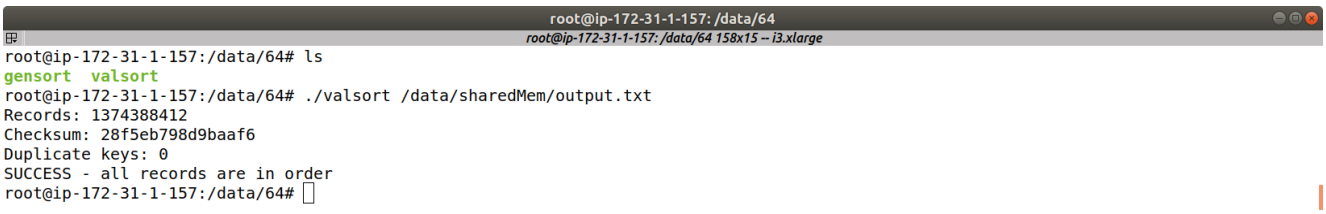
```
$ mkdir tmp
```

```
$ mkdir stmp
```

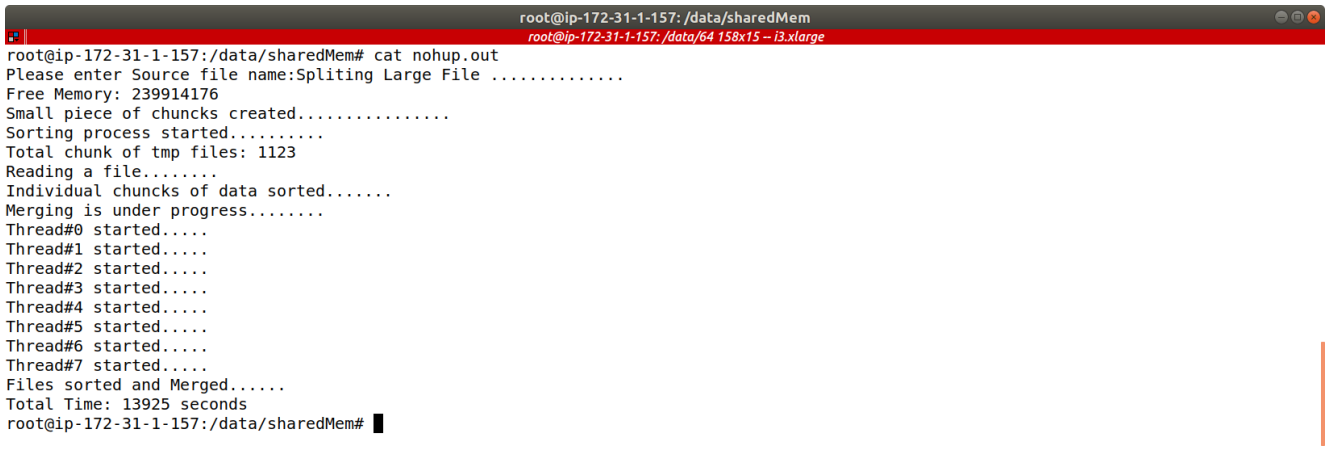
5. Now run the program using following command as a background service so that we can avoid any undesirable condition such as connection lost or connection broken error. It will also save result to nohup.out file.

```
$ nohup java ExternalSort sample128gb.txt &
```





Output generated by background service. nohup.out



Result to sort 128GB data using External Sort.

Data size	128 GB = 137438953472 Bytes
No. of Bytes Read (FILE)	692563476480
No. of Bytes Write (FILE)	277025390592
Time Taken by whole process	13925 seconds
Throughput	9.947 MB/s

Explanation:

Here the bottleneck is the merging the file as it need to to load data from several sorted temporary file. To solve this problem and reduce the time to open and seek file pointer, we have used Priority Queue which will refer to the File Pointer and read the data from the file line by line and compare with other file.

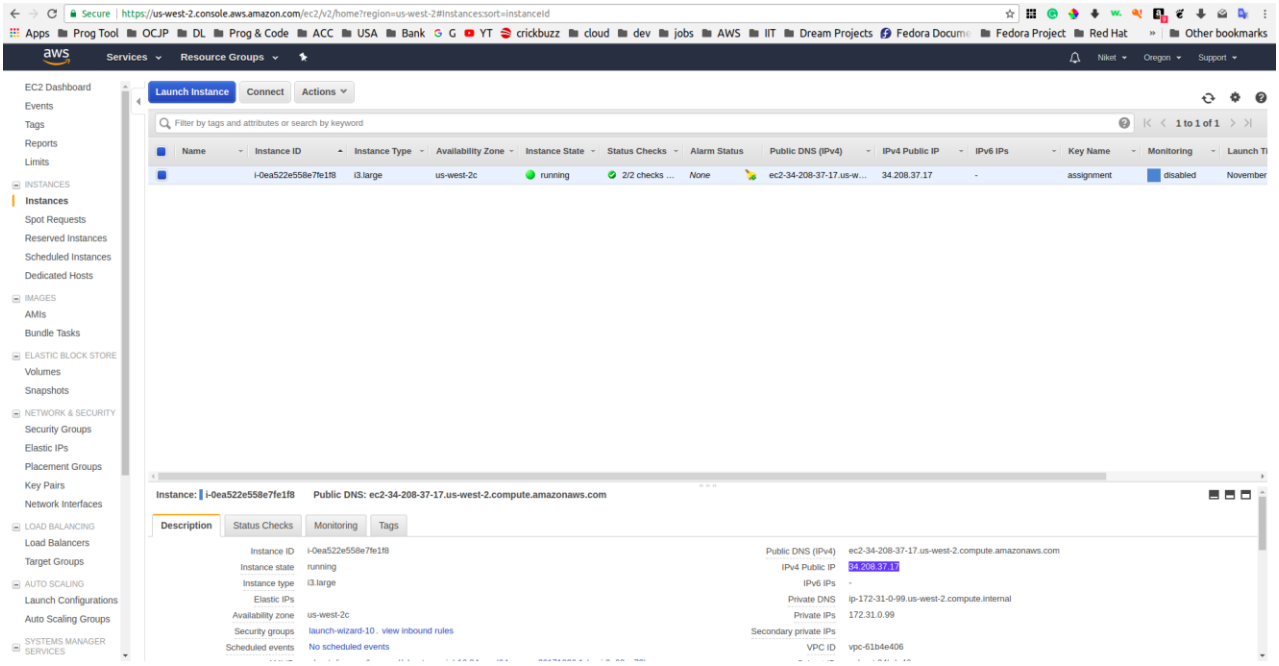
Difficulties:

As the connection with aws instance through terminal, several times my connection broken and my all process lost in between. To resolve this issue, I tried to run it as background service and used nohub command. This command will generate one nohup.out file which is the output of program which we used as logs.

b. Hadoop Terasort:

To perform the terasort hadoop program for 128GB on i3.large instance, follow the following steps.

1. Create i3.large instance on Amazon AWS.



2. Mount the 0.475 SSD with our instance using following command.

```
$ sudo su
$ sudo mkfs -t ext4 /dev/nvme0n1
$ mkdir /data
$ mount /dev/nvme0n1 /data
$ chmod 777 /data
```

3. Install Hadoop from apache hadoop official website using following commands in /data directory where we have mounted our SSD.

We have used hadoop-2.7.4 version for our assignment.

```
$ cd /data
$ wget https://archive.apache.org/dist/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz
```

```
$ tar xvzf hadoop-2.7.1.tar.gz
$ rm -rf hadoop-2.7.1.tar.gz
$ mv hadoop-2.7.1/ hadoop
```

4. After downloading Hadoop, install OpenJdk-8 and SSH using following command.

```
$ sudo apt-get update
$ sudo apt-get install openjdk-8*
$ sudo apt-get install ssh
```

5. Now go to /root directory and set the JAVA_HOME and HADOO_HOME environment variable to .bashrc file.

```
$ sudo su
$ cd /root
$ vi .bashrc
```

And add the following lines at the end of the file.

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
export PATH="$PATH:$JAVA_HOME/bin"
export HADOOP_HOME="/data/hadoop"
export PATH="$PATH:$HADOOP_HOME/bin"
export PATH="$PATH:$HADOOP_HOME/sbin"
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_MEPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export YARN_HOME=$HADOOP_HOME
```

```
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_OPTS="-Djava.library.path=$HADOOP_INSTALL/lib/native"
export HADOOP_CLASSPATH=/usr/lib/jvm/java-8-openjdk-amd64/lib/tools.jar
```

After updating .bashrc file, update the changes to the system using following command.

```
$ source .bashrc
```

6. Now create ssh free login to localhost using following command in the /root directory.

```
$ ssh-keygen -t rsa -P ""
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

7. Now download TeraGen to /data directory and generate 128 GB file using following command.

```
$ cd /data
$ wget http://www.ordinal.com/try.cgi/gensort-linux-1.5.tar.gz
$ tar xvzf gensort-linux-1.5.tar.gz
$ cd 64/
$ ./gensort -a -t8 1374389534 sample128gb.txt
```

8. After completing above steps, logout from root and create hdfs directory and hdfs/namenode, hdfs/datanode directory in /data folder.

```
$ cd /data
$ mkdir hdfs/
$ mkdir hdfs/namenode
$ mkdir hdfs/datanode
```

9. Now, configure hadoop environment by changing default property to following files.

a. hadoop-env.sh
Set JAVA_HOME variable in hadoop-env.sh

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
```

b. Core-site.xml
Add following lines at the end of the file.

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/data/tmp</value>
  </property>
</configuration>
```

c. Hdfs-site.xml
Add following lines at the end of the file.

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

```
    <value>file:/data/hdfs/namenode</value>
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/data/hdfs/datanode</value>
  </property>
  <property>
    <name>dfs.blocksize</name>
    <value>1073741824</value>
  </property>
</configuration>
```

d. Mapred-site.xml

In order to edit this file, use following command first.

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

Add following lines at the end of the file.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.job.reduces</name>
    <value>2</value>
  </property>
</configuration>
```

e. Yarn-site.xml

Add following lines at the end of the file.

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

10. Now format the namenode using following command.

```
$ hdfs namenode -format
```

11. Now run the dfs and yarn services using following command.

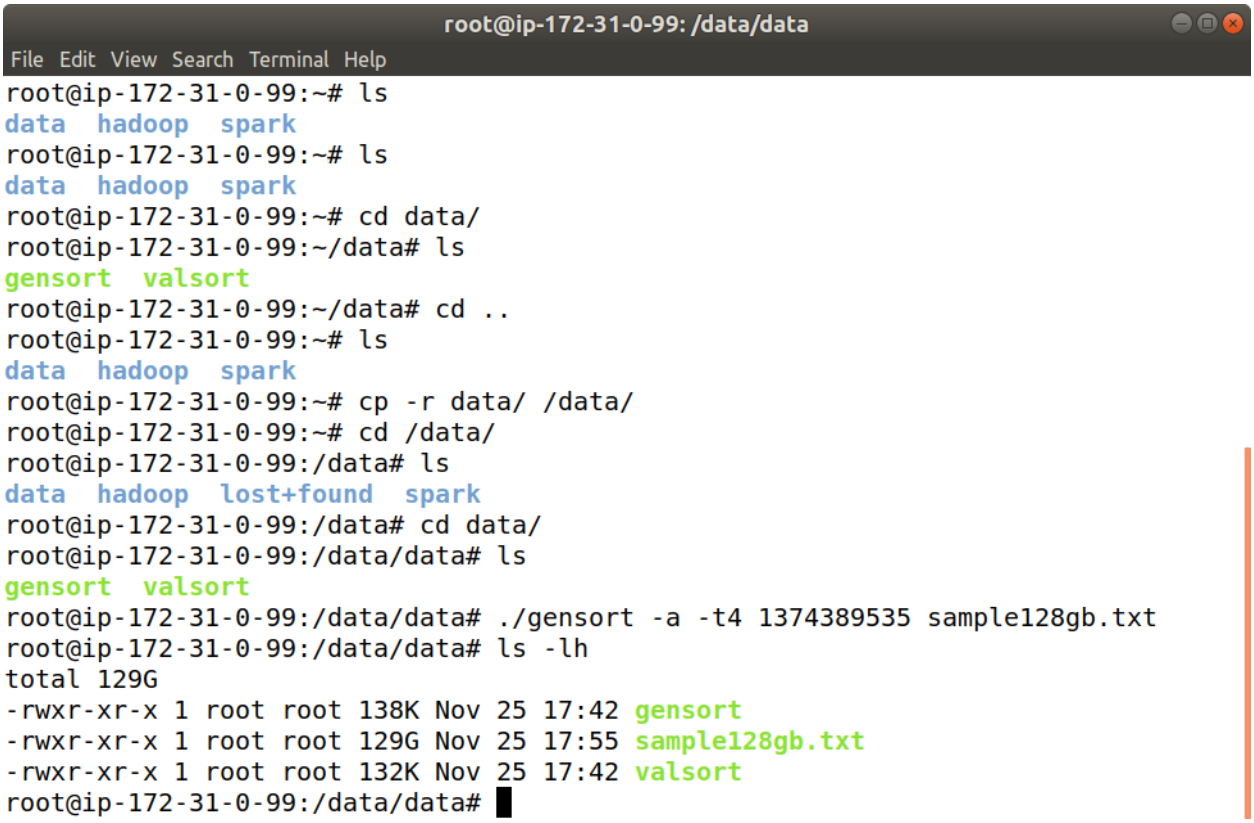
```
$ start-dfs.sh
$ start-yarn.sh
```

12. To check service running use following command.

```
$ jps
```

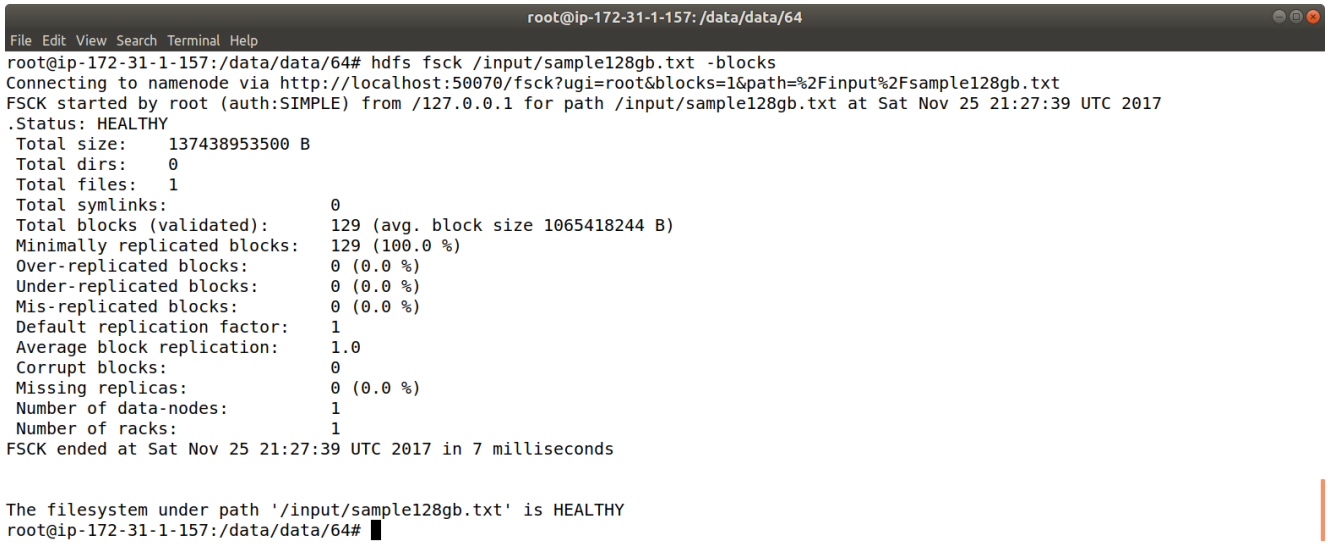
13. After completing and running the hadoop, upload the 'sample128gb.txt' file to the hadoop by setting 1024MB block size.By setting block size 1024MB will reduce I/O latency. Use following command to upload the file.

```
$ ssh localhost
$ cd /data
$ cd 64/
$ hdfs dfs -mkdir /input
$ hadoop fs -D dfs.block.size=536870912 -put sample128gb.txt /input
```



14. To check number of blocks in the file, use the following command.

```
$ hdfs fsck /input/sample128gb.txt -blocks
```



We can also check the result in web browser using following link.

```
http://<public_id>:50070/fsck?ugi=root&blocks=1&path=%2Finput%2Fsample128gb.txt
```

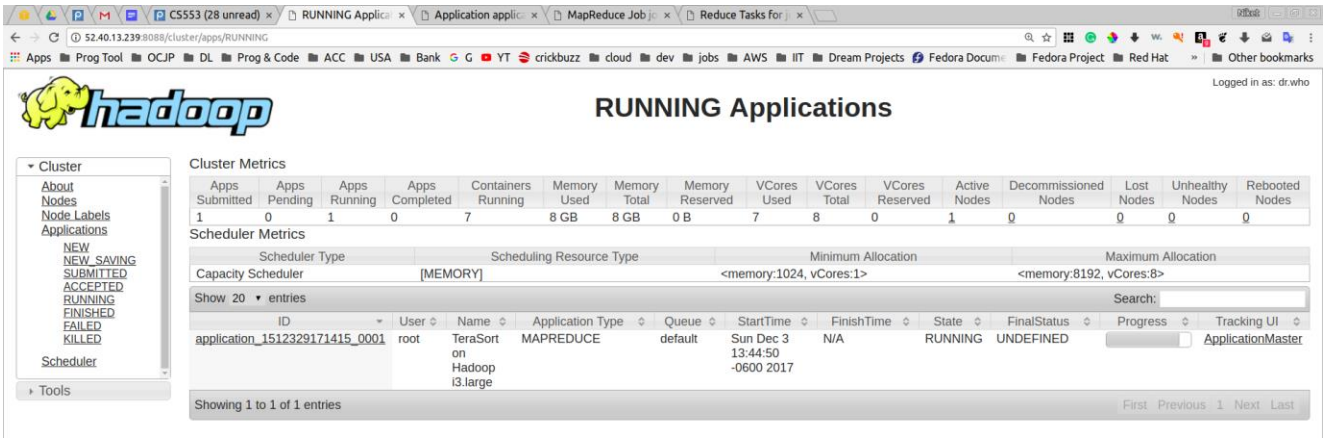
```
.Status: HEALTHY
 Total size:    137438953500 B
 Total dirs:    0
 Total files:   1
 Total symlinks:                0
 Total blocks (validated):      129 (avg. block size 1065418244 B)
 Minimally replicated blocks:   129 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1
```

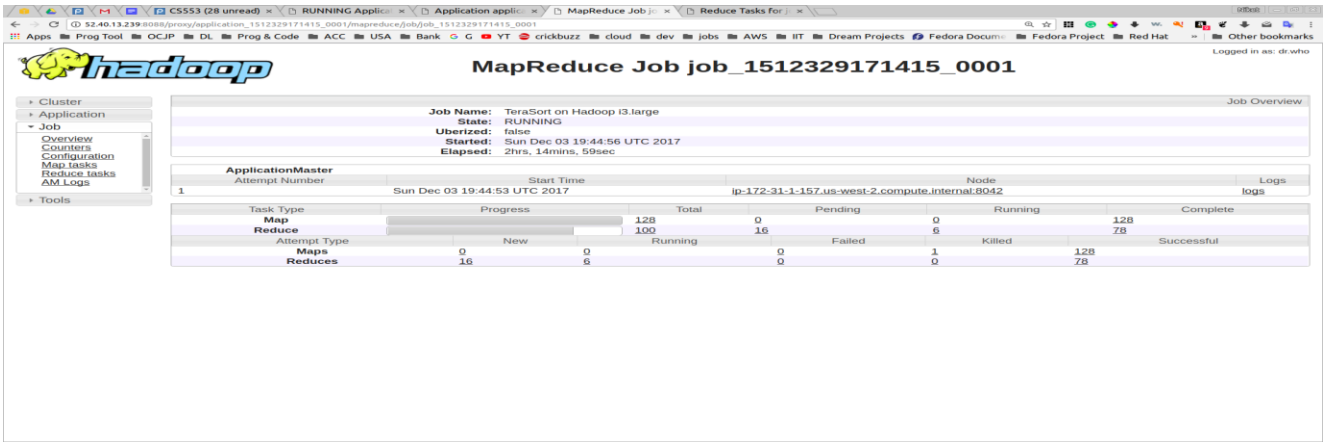
15. Now run the Terasort Hadoop jar file to sort the 128GB data.

```
$ hadoop jar hadoop_demo-0.0.1-SNAPSHOT.jar  assigment.hadoop_demo.Terasort /input /output
```

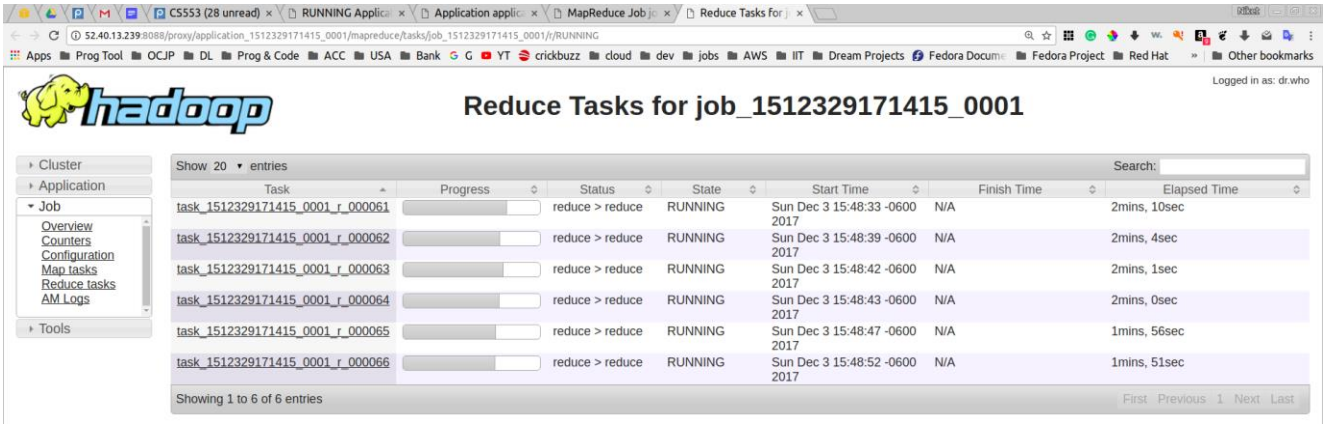
We can also trace the process using public ip and port number as follow.

```
$ http://52.40.13.239:8088/cluster
```

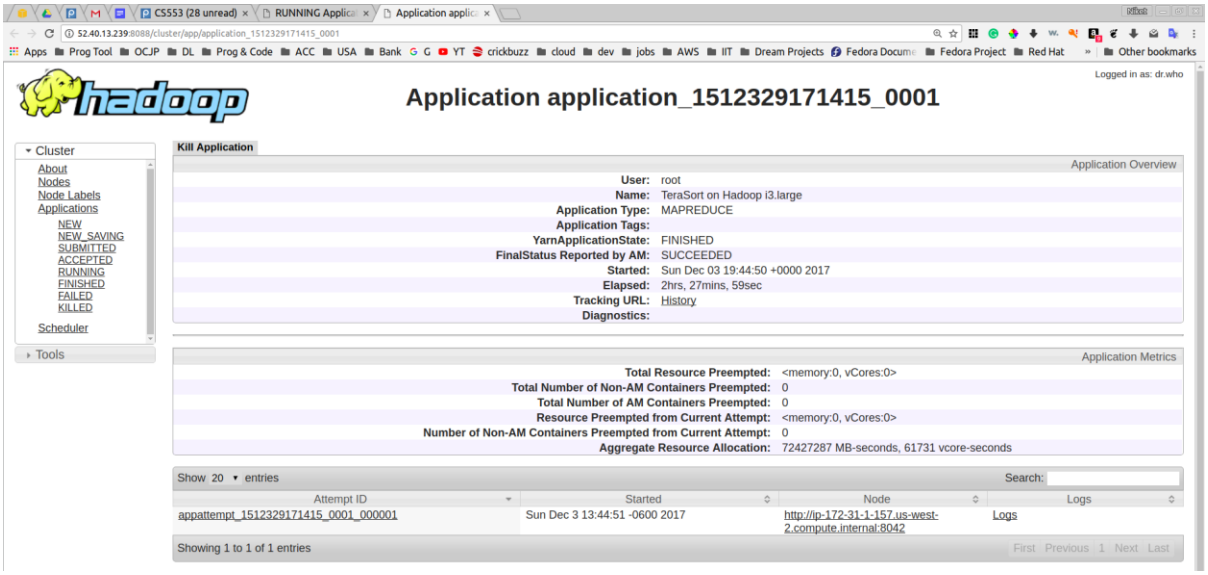




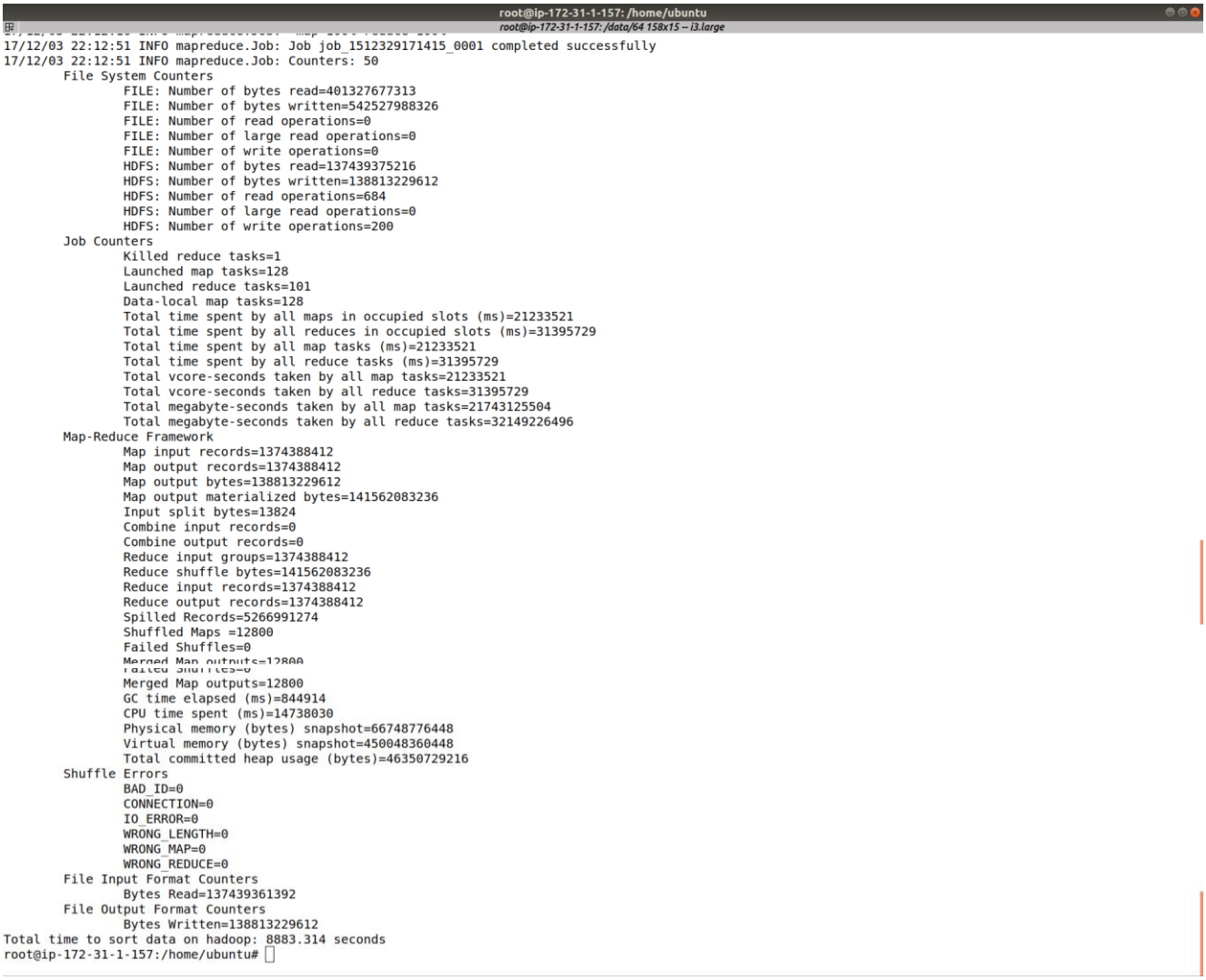
Reducer Phase.



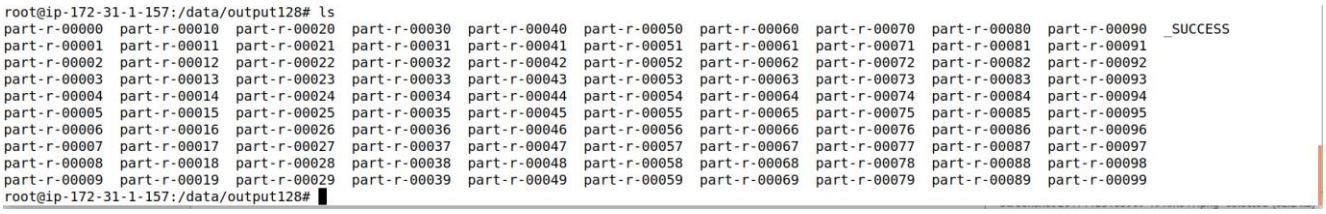
When mapreduce completes its job, we can see following screen on hadoop webpage.



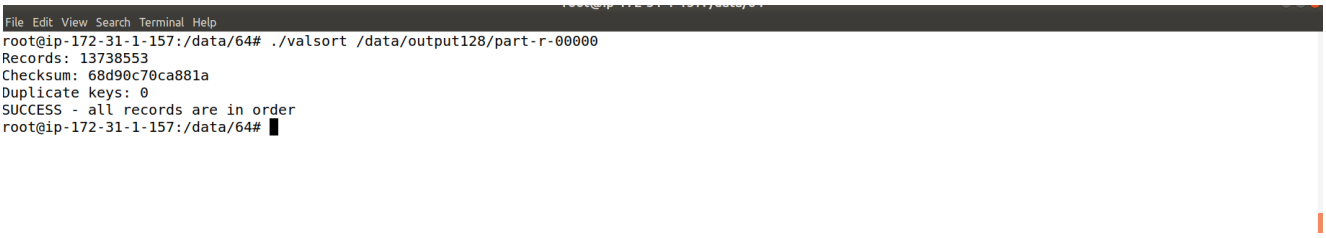
We can also see the details on the terminal as follow.



Output files will be divided into 100 small size files. We can verify sorting results with valsort.



Validating sorting results.



Results for 128GB *Hadoop Terasort.*

Data size	128 GB = 137438953472 Bytes
No of Mappers	128
No of Reducer	2
Total Bytes Read (HDFS)	137439375216
Total Bytes Write (HDFS)	138813229612
No of Read ops (HDFS)	684
No Of Write ops (HDFS)	200
Time Taken by whole process	8883.314 seconds
Throughput	15.471 MB/s

Explanation:

As here we have limited storage available and hadoop mapreduce also generates extra temporary files along with input files, consumes storages space and with increasing size of data, hadoop process slows down because it needs to clean up the junk files in order to save more data, which takes time and halts the mapreduce process for some amount of time.

Difficulties:

I have setup hadoop environment with 3 trial. During 1st trial, I have tried to run Terasort program with default configuration for 50GB dataset and note the time.

After this, I have update configuration like namenode and datanode path, tmp file path and jobtracker URI. I have noted that with this configuration I got improvisation than default configuration.

At the last I tried to change hadoop dfs.block.size property which highly improved my hadoop system performance as the file block

size changes from 128MB to 1024MB. It reduces total number of Mapper and reducer and eventually helped me to increase performance.

c. Spark Terasort:

To perform Spark Terasort, I assume we have already set up our hadoop environment and We have uploaded 128GB unsorted file in our hdfs system.

Follow the following steps to set up spark environment.
We have used spark apache spark-2.2.0 for our assignment.

1. Download spark-2.2.0 from apache spark website and move it to /data directory.

```
$ cd /data
$ wget http://apache.claz.org/spark/spark-2.2.0/spark-2.2.0-bin-hadoop2.7.tgz
$ tar xvzf spark-2.2.0-bin-hadoop2.7.tgz
$ rm spark-2.2.0-bin-hadoop2.7.tgz
$ mv spark-2.2.0-bin-hadoop2.7 spark
```

2. Now initiate spark-shell with following command.

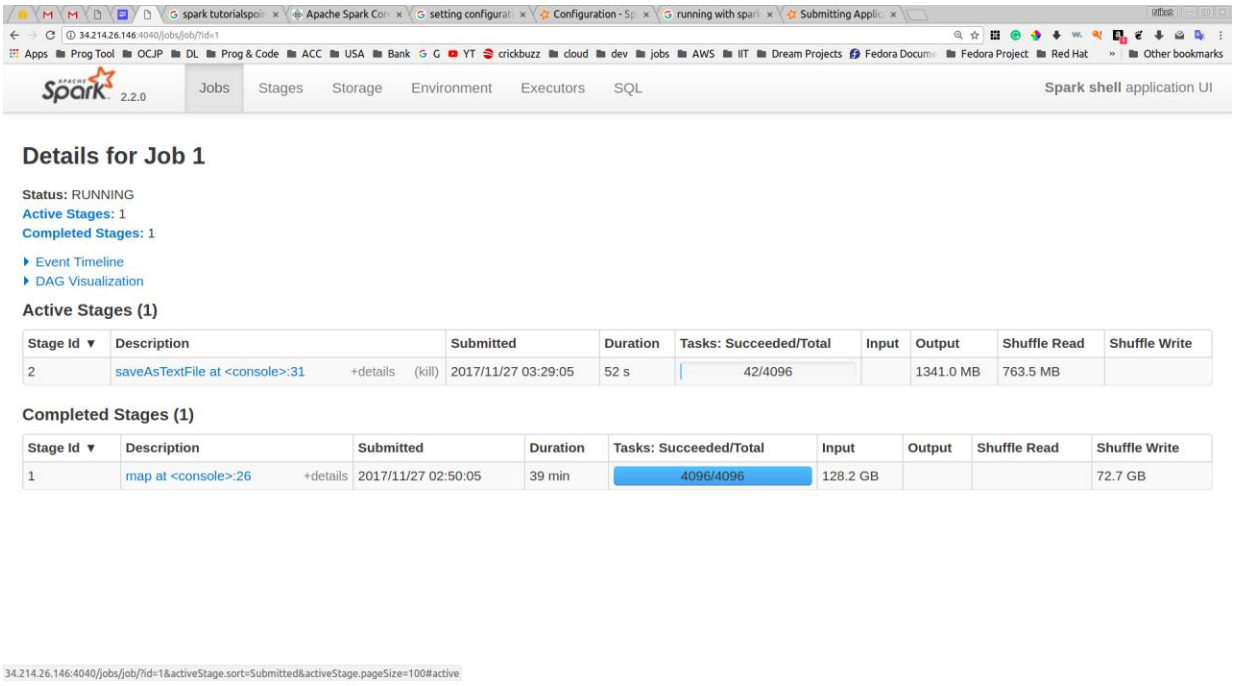
```
$ spark-shell --conf spark.local.dir=/data/sparktmp --executor-memory 6g --num-executors 2 --driver-memory 4g
```

3. Write following scala Terasort program in spark shell.

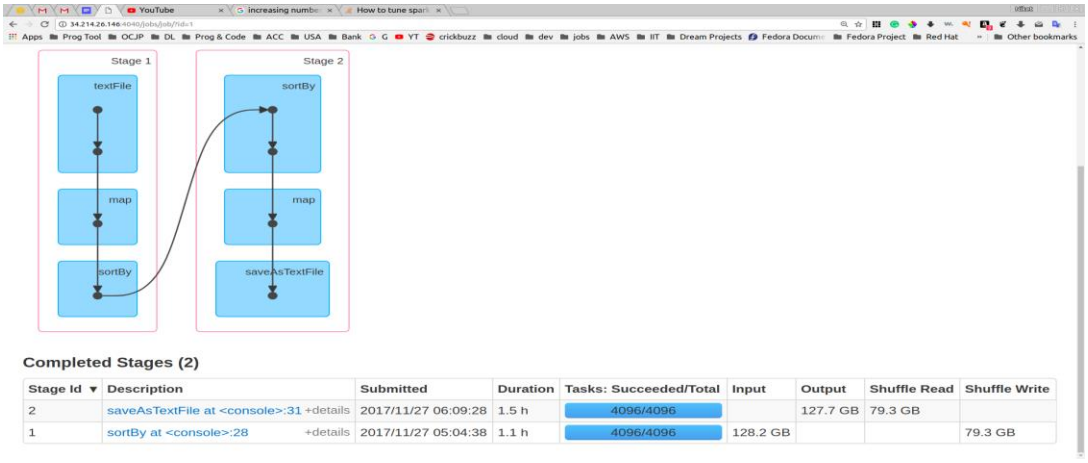
```
Scala> val lines = sc.textFile("hdfs://localhost:9000/input/sample128gb.txt")
scala> val rdd1 = lines.map(x=>(x.slice(0,10), x.slice(10,x.length)))
scala> val rdd2 = rdd1.sortBy(_._1)
scala> rdd2.map(k => k._1+k._2).saveAsTextFile("hdfs://localhost:9000/output")
```

4. After submitting Spark job, we can trace it through our local webpage:

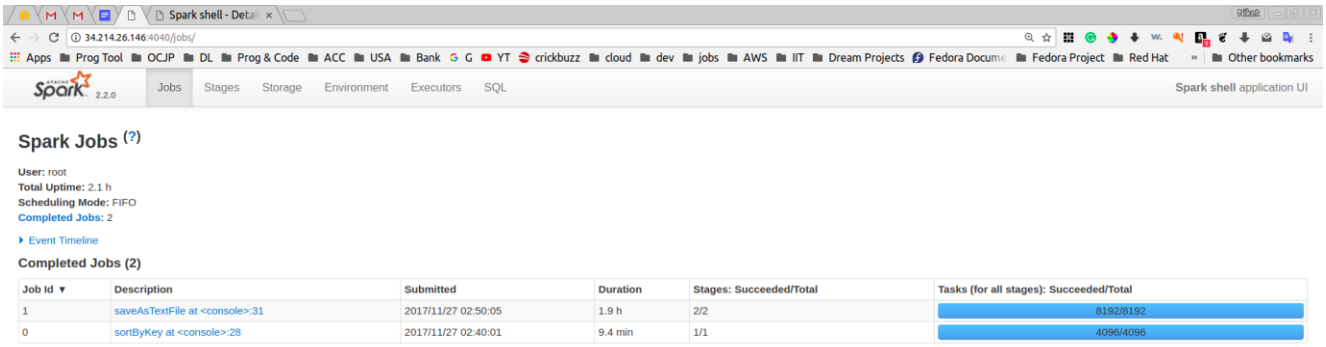
```
http://<public_ip>:4040
```



DAG Visualization of our program.



When Spark completes its execution.



Results for 128GB *[actually 129GB generated (overhead bytes per line)]*

Data size	128 GB = 137438953472 Bytes
Total Bytes Read (HDFS)	137439487516
Total Bytes Write (HDFS)	137438953500
Time Taken by whole process	10080 seconds
Time Taken to sort the data	9.4 min = 564 seconds
Throughput	13.63 MB/s

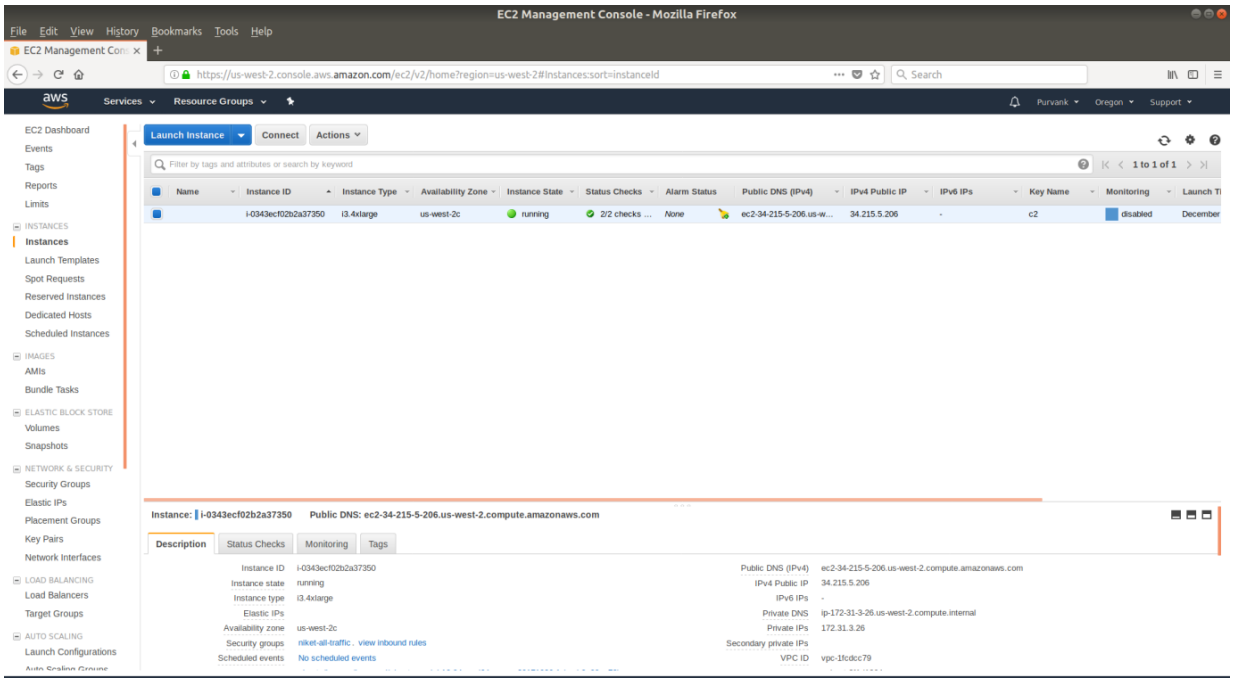
Configuration: 2

Virtual Cluster (1-Node, i3.4xlarge)

AWS I3.4xlarge configuration:

Instance Name	i3.4xlarge
vCPU count	16
Memory	122 GiB
Instance Storage (NVMe SSD)	3.8 TB
Price/Hour	$1.248
Operating System	Ubuntu 16.04.3 LTS

Launch the instance i3.4xlarge.



a. Shared Memory Sort

TO perform shared Memory Terasort on i3.4xlarge follow the same steps as we performed for i3.large Terasort and generate 1TB data using gensort.

```
$ ./gensort -a -t12 10995116277 sample1tb.txt
```

Use the following command to run it as background service.

```
$ nohup java ExernalSort sample1tb.txt &
```

```
ubuntu@ip-172-31-1-208: ~
File  Edit  View  Search  Terminal  Help
top - 05:25:18 up  1:34,  3 users,  load average: 3.30, 1.91, 1.41
Tasks: 235 total,   1 running, 234 sleeping,   0 stopped,   0 zombie
%Cpu(s):  5.8 us,  0.5 sy,  0.0 ni, 93.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 12582787+total, 38612836 free, 20979120 used, 66235920 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 10401057+avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
17640 root      20   0 34.471g 0.019t  16656 S 101.0 16.4   9:13.11 java
    1 root      20   0   37752   5784   3960 S   0.0  0.0   0:02.81 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.00 ksoftirqd/0
    5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    7 root      20   0       0      0      0 S   0.0  0.0   0:01.45 rcu_sched
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
    9 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.02 watchdog/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 watchdog/1
   12 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/1
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 ksoftirqd/1
   15 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   16 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 watchdog/2
   17 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/2
   18 root      20   0       0      0      0 S   0.0  0.0   0:00.00 ksoftirqd/2
```

Output of Program.

```
root@ip-172-31-1-208: /data/sharedMem
File  Edit  View  Search  Terminal  Help
4.0K    nohup.out
1.0T    output.txt
32K     stmp
root@ip-172-31-1-208:/data/sharedMem# cat nohup.out
Please enter Source file name:Spliting Large File ..............
Free Memory: 1919313496
Small piece of chuncks created.................
Sorting process started..........
Total chunk of tmp files: 1123
Reading a file........
Individual chuncks of data sorted.......
Merging is under progress........
Thread#0 started.....
Thread#1 started.....
Thread#2 started.....
Thread#3 started.....
Thread#4 started.....
Thread#5 started.....
Thread#6 started.....
Thread#7 started.....
Thread#8 started.....
Thread#9 started.....
Thread#10 started.....
Thread#11 started.....
Thread#12 started.....
Thread#13 started.....
Files sorted and Merged......
Total Time: 45485 seconds
root@ip-172-31-1-208:/data/sharedMem#
```

- Result using valsort.

```
root@ip-172-31-1-208: /data/64
File  Edit  View  Search  Terminal  Help
root@ip-172-31-1-208:/data/64# cat nohup.out
Records: 10995115155
Checksum: 147ada84970251286
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-1-208:/data/64#
```

Result to sort 1 TB data using External Sort.

Data size	1 TB = 1099511627776 Bytes
No. of Bytes Read (FILE)	5.4975581e+12
No. of Bytes Write (FILE)	2199023255552
Time Taken by whole process	45485 seconds
Throughput	24.17 MB/s

b. Hadoop Terasort

We will have 2 1.7 TB SSD attached with our i3.4xlarge instance. To utilize these disk, we will create RAID 0 disk array.

To create RAID 0 array, follow the steps.

```
$ Sudo su
$ sudo apt-get install mdadm
$ mdadm --create /dev/md0 --run --level=0 --name=RAID --force --raid-device=2 /dev/nvme0n1 /dev/nvme1n1
$ mkfs.ext4 -L RAID /dev/md0
$ mkdir /data
$ mount LABEL=RAID /data
$ chmod 777 /data
```

Follow configuration-1 hadoop installation steps to install and run Hadoop program.

To create 1TB data use following command.

```
$ ./gensort -a -t12 10995116277 sample1tb.txt
```

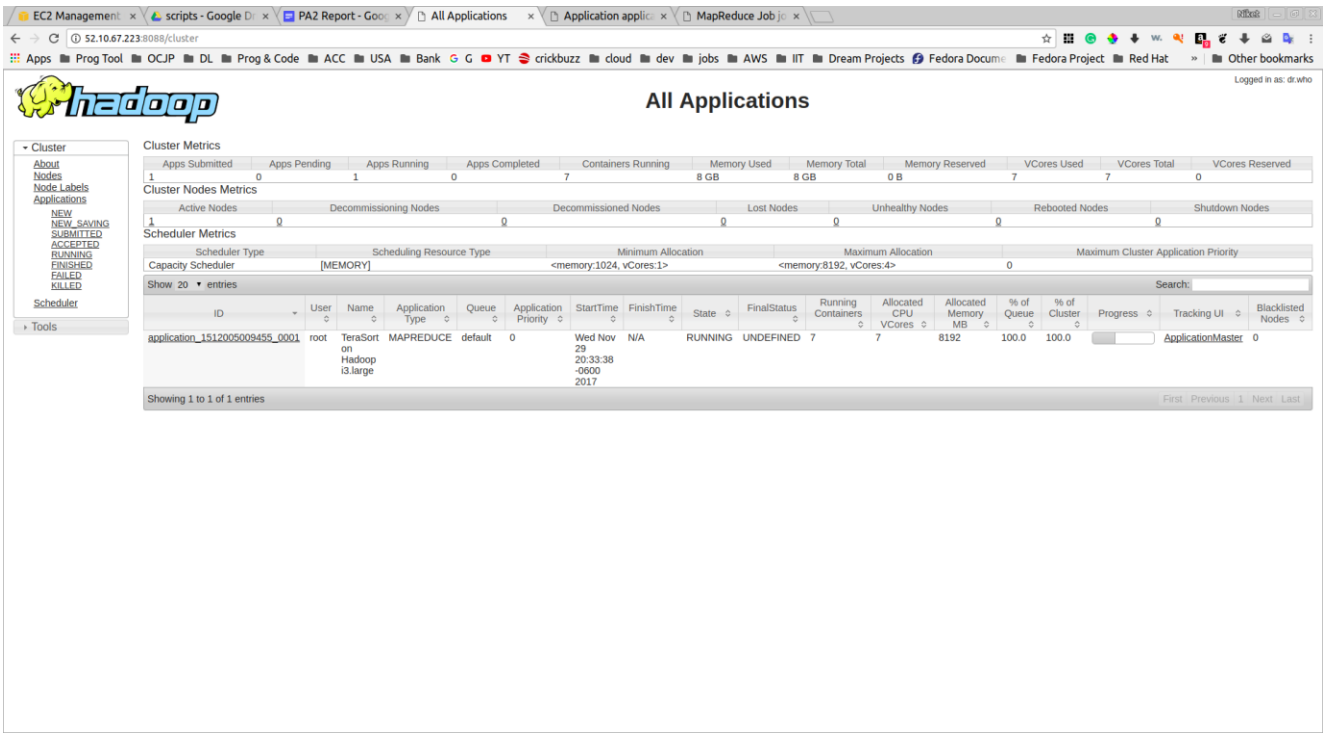
Load the data to hdfs using following command.

```
$ hdfs dfs -copyFromLocal sample1tb.txt /input
```

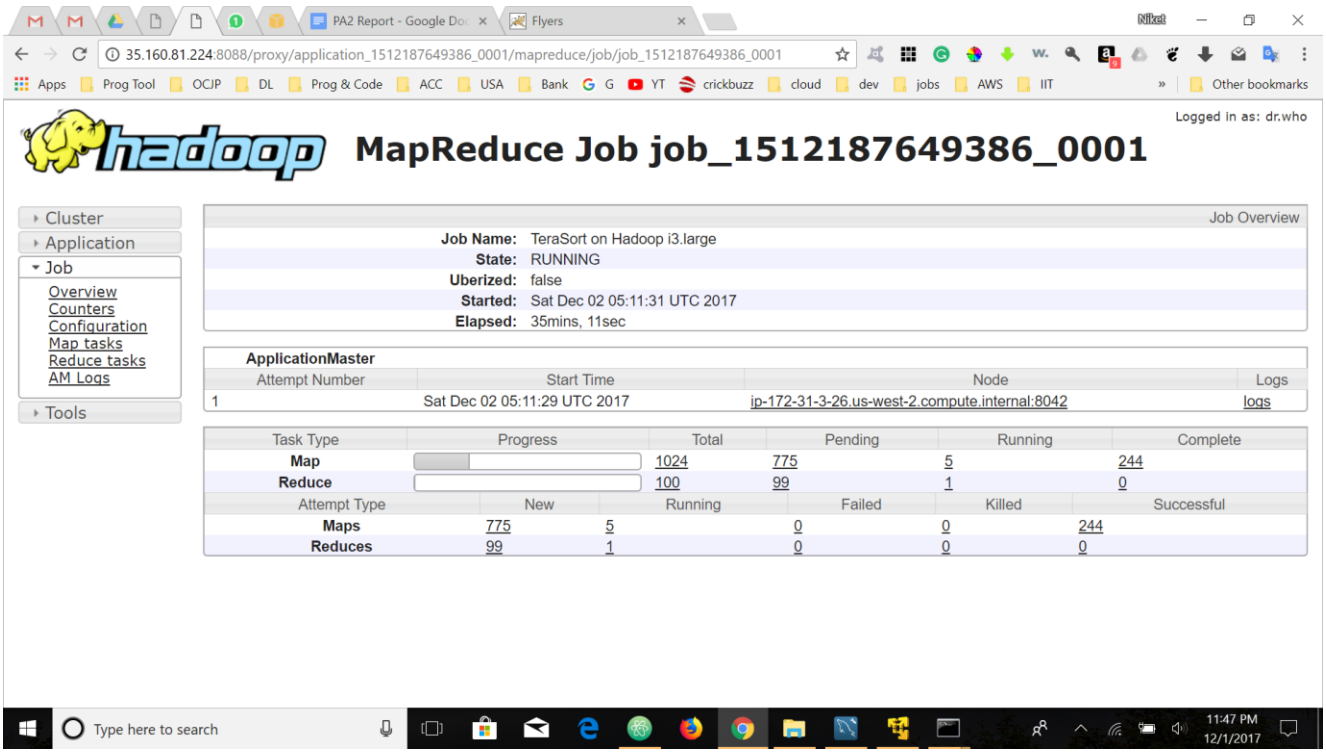
Execute the Terasort Jar file using following command.

```
$ hadoop jar hadoop_demo-0.0.1-SNAPSHOT.jar  assigment.hadoop_demo.Terasort /input /output
```

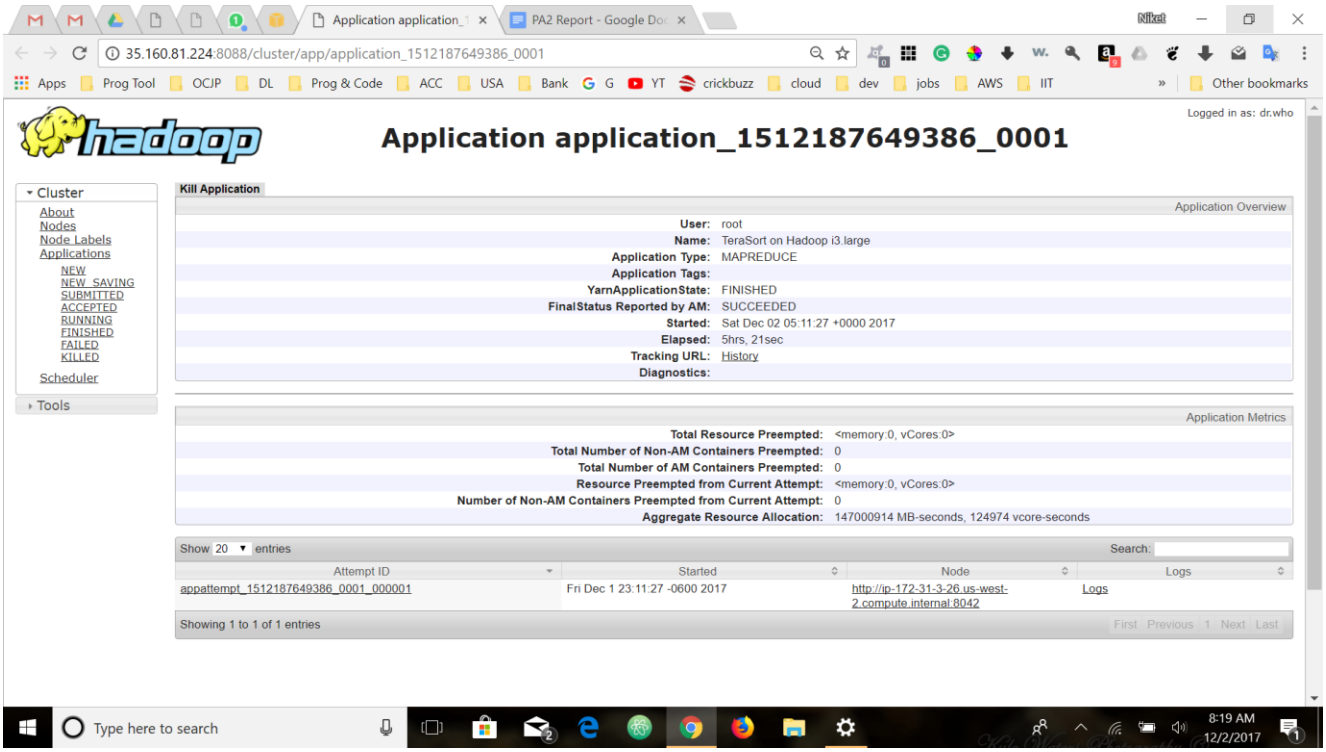
The process will be traced using hadoop UI portal.

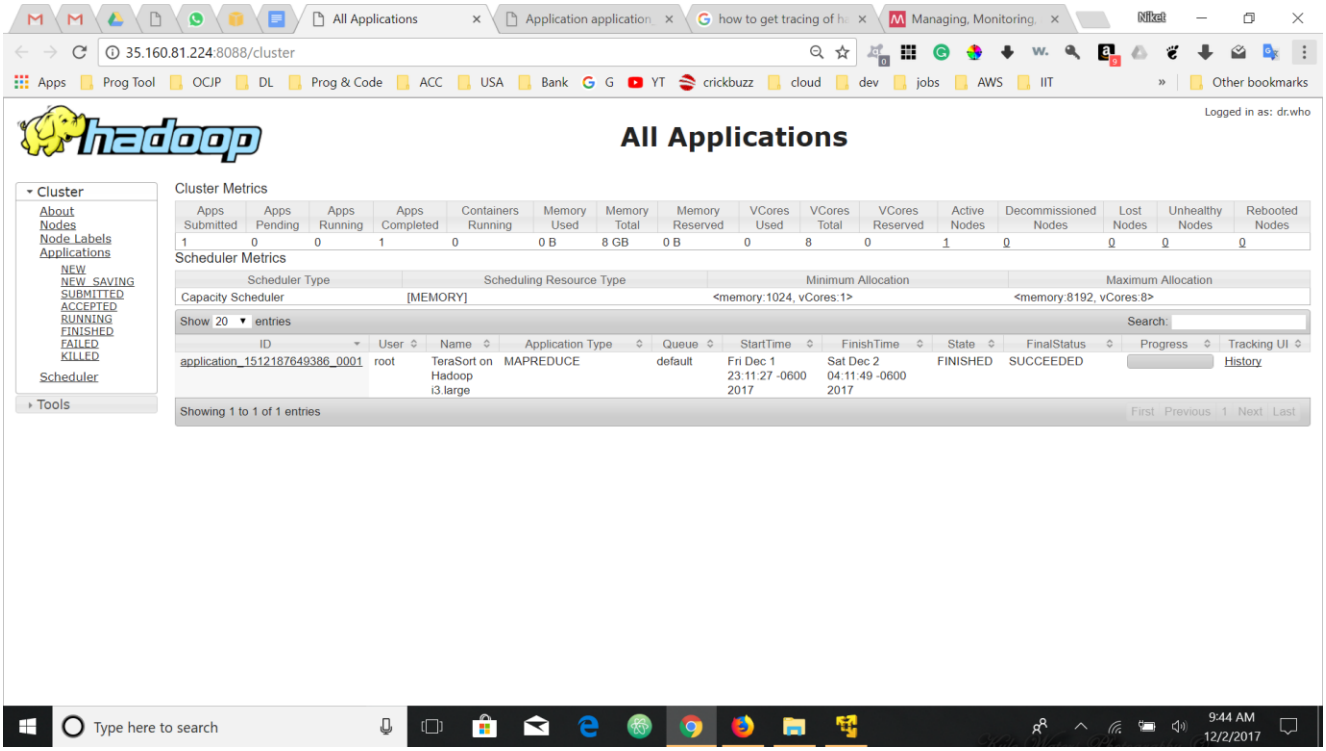


The running process.



Note: The name displaying here in the screenshot is referring to the name provided in Terasort.java file.





After Completing process.

```
root@ip-172-31-3-26: /home/ubuntu
root@ip-172-31-3-26: /home/ubuntu 158x41
Counters: 49
        File System Counters
                FILE: Number of bytes read=4096648974240
                FILE: Number of bytes written=5216339944692
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1099515926452
                HDFS: Number of bytes written=1098375069600
                HDFS: Number of read operations=3369
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=199
        Job Counters
                Launched map tasks=1024
                Launched reduce tasks=100
                Data-local map tasks=1024
                Total time spent by all maps in occupied slots (ms)=44646946
                Total time spent by all reduces in occupied slots (ms)=61196180
                Total time spent by all map tasks (ms)=44646946
                Total time spent by all reduce tasks (ms)=61196180
                Total vcore-seconds taken by all map tasks=44646946
                Total vcore-seconds taken by all reduce tasks=61196180
                Total megabyte-seconds taken by all map tasks=45718472704
                Total megabyte-seconds taken by all reduce tasks=62664888320
        Map-Reduce Framework
                Map input records=10995116277
                Map output records=10995116277
                Map output bytes=1099511627700
                Map output materialized bytes=1121502474654
                Input split bytes=108544
                Combine input records=0
                Combine output records=0
                Reduce input groups=10983760690
                Reduce shuffle bytes=1121502474654
                Reduce input records=10983760690
                Reduce output records=10983760690
                Spilled Records=51029224524
                Shuffled Maps =102400
                Failed Shuffles=0
                Merged Map outputs=102400
                GC time elapsed (ms)=4595515
                CPU time spent (ms)=110342470
```

```
        CPU time spent (ms)=110342470
        Physical memory (bytes) snapshot=341006716928
        Virtual memory (bytes) snapshot=2271661056000
        Total committed heap usage (bytes)=231906738176
Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
        Bytes Read=1099515817908
File Output Format Counters
        Bytes Written=1098376070100
```

```
root@ip-172-31-3-26: /data
File  Edit  View  Search  Terminal  Help
0 updates are security updates.


Last login: Sat Dec  2 08:34:58 2017 from 69.47.179.48
ubuntu@ip-172-31-3-26:~$ clear
ubuntu@ip-172-31-3-26:~$ ls
hadoop_demo-0.0.1-SNAPSHOT.jar
ubuntu@ip-172-31-3-26:~$ sudo su
root@ip-172-31-3-26:/home/ubuntu# ls
hadoop_demo-0.0.1-SNAPSHOT.jar
root@ip-172-31-3-26:/home/ubuntu# cd /data/
root@ip-172-31-3-26:/data# ls
datagen  hadoop  hadoop-2.7.1.tar.gz  hdfs  lost+found  tmp
root@ip-172-31-3-26:/data# hadoop job -list
DEPRECATED: Use of this script to execute mapred command is deprecated.
Instead use the mapred command for it.

17/12/02 14:23:28 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
Total jobs:0
                  JobId      State            StartTime      UserName           Queue      Priority       UsedContainers  RsvdContainers  UsedMem
        RsvdMem NeededMem         AM info
root@ip-172-31-3-26:/data# hadoop job -list all
DEPRECATED: Use of this script to execute mapred command is deprecated.
Instead use the mapred command for it.

17/12/02 14:24:46 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
Total jobs:1
                  JobId      State            StartTime      UserName           Queue      Priority       UsedContainers  RsvdContainers  UsedMem
        RsvdMem NeededMem         AM info
 job_1512187649386_0001  SUCCEEDED       1512191487864          root         default        NORMAL                N/A             N/A      N/A
          N/A       N/A      http://ip-172-31-3-26:8088/proxy/application_1512187649386_0001/
root@ip-172-31-3-26:/data#
```

Validating data using ./valsort

```
root@ip-172-31-3-26: /data/datagen/64
File  Edit  View  Search  Terminal  Help
root@ip-172-31-3-26:/data/datagen/64# ./valsort /data/output/part-r-00000
Records: 109955710
Checksum: 34703aaabe67458
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-3-26:/data/datagen/64#
```

Results for 1 TB Data.

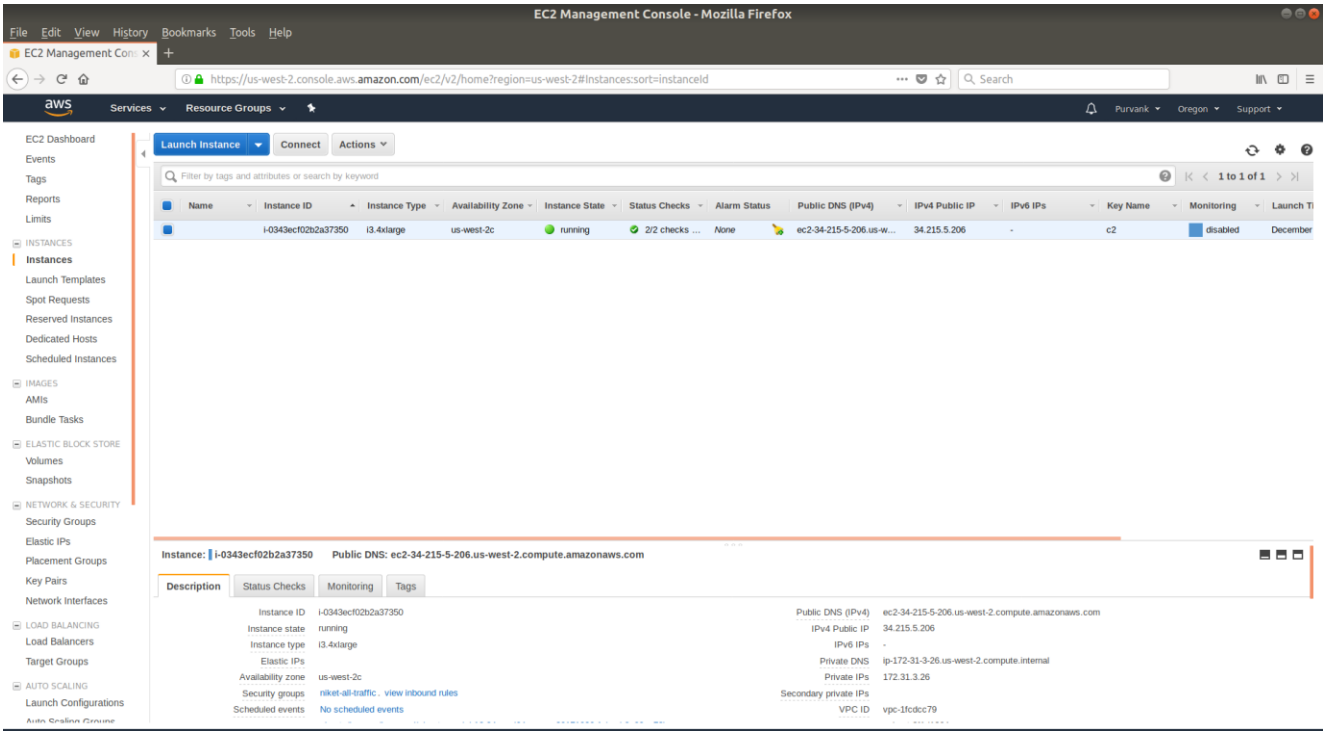
Data size	1 TB = 1,099,511,627,776 Bytes
No of Mappers	1024
No of Reducer	16
Total Bytes Read (HDFS)	1099515926452

Total Bytes Write (HDFS)	1098375069600
No of Read ops (HDFS)	3369
No Of Write ops (HDFS)	200
Time Taken by whole process	18021 seconds
Throughput	61.01 MB/s

c. Spark Terasort

To run spark on i3.4xlarge instance, follow the above mentioned steps to download the spark and use the following code to run spark.

1. Launch i3.4xlarge instance on AWS.



2. Now initiate spark-shell with following command.

```
$ spark-shell --conf spark.local.dir=/data/sparktmp --executor-memory 12g --num-executors 2 --driver-memory 4g
```

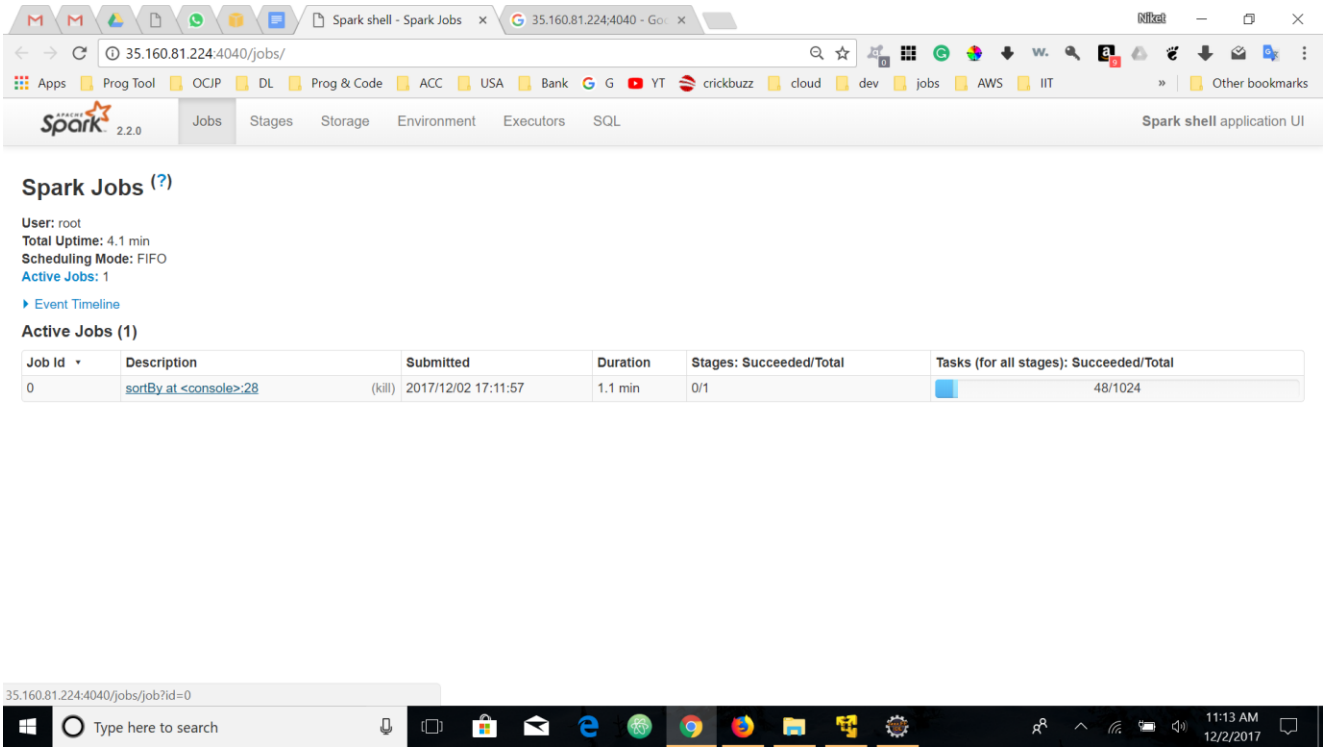
2. Write following scala Terasort program in spark shell.

```
Scala> val lines = sc.textFile("hdfs://localhost:9000/input/part50.txt")
scala> val rdd1 = lines.map(x=>(x.slice(0,10), x.slice(10,x.length)))
scala> val rdd2 = rdd1.sortBy(_._1)
scala> rdd2.map(k => k._1+k._2).saveAsTextFile("hdfs://localhost:9000/output50")
```

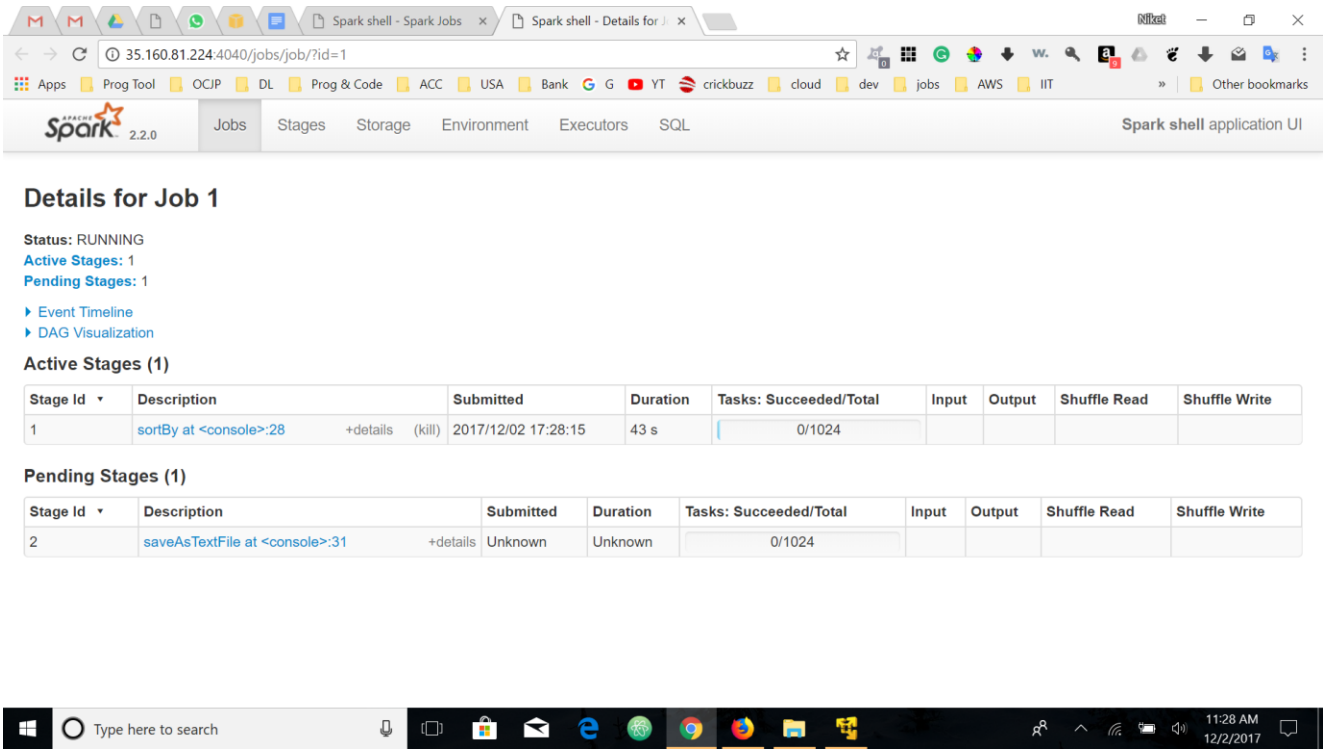
3. After submitting Spark job, we can trace it through our local webpage:

```
http://<public_ip>:4040
```

Stage:1



Stage:2



Results for 1 TB data.

Data size	1 TB = 1,099,511,627,776 Bytes
Total Bytes Read (HDFS)	1099515926452
Total Bytes Write (HDFS)	1098375069600
Time Taken by whole process	25920 seconds
Time Taken to sort the data	15 min = 900 seconds
Throughput	42.45 MB/s

Configuration: 3

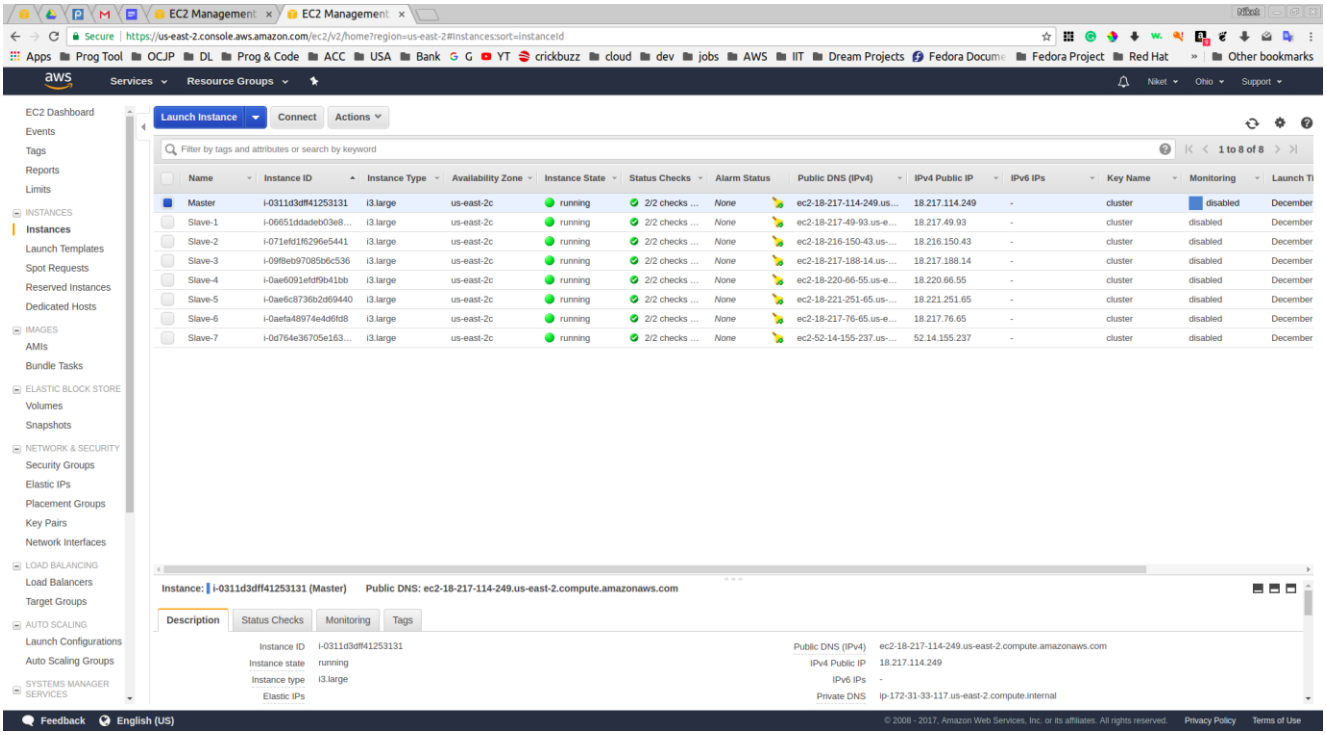
Virtual Cluster (8-Node, i3.large)

AWS I3.large configuration:

Instance Name	i3.large
vCPU count	2
Memory	15.25 GiB
Instance Storage (NVMe SSD)	0.475 TB
Price/Hour	$0.15
Operating System	Ubuntu 16.04.3 LTS

a. Hadoop 8 Node Terasort.

To perform Terasort on hadoop 8 node cluster, we need to create 8 VM machine of i3.large configuration.



Follow the following steps to create cluster using this 8 node.

1. Upload all autonomous script from Autonomous Script folder using following command to each node.

```
$ scp -i cluster.pem autonomus.xz ubuntu@ec2-18-217-114-249.us-east-2.compute.amazonaws.com
```

2. Run Mount.sh script to mount 475 GB hard disk with each running node.

```
$ ./Mount.sh
```

3. Run install-required-package.sh script from each node to install java and ssh.

```
$ ./install-required-packcage.sh
```

4. Add private ip node_name pair to /etc/hosts file and save it.

```
$ vi /etc/hosts
```

```
172.31.33.117 master
172.31.46.18 slave1
172.31.42.104 slave2
172.31.45.111 slave3
172.31.46.173 slave4
172.31.41.130 slave5
172.31.33.251 slave6
172.31.44.75 slave7
```

5. Now create public private key pair using generate-ssh.sh script from master node and copy public key to each slave node authorized_keys file residing in .ssh folder

```
$ ssh-keygen -t rsa -P ""
```

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
$ chmod 0600 ~/.ssh/authorized_keys
```

6. Now, download hadoop-2.7.1 using Hadoop-Download.sh script from each node.

```
$ ./Hadoop-Download.sh
```

Now update following files to the /data/hadoop/etc/hadoop/ directory.

a. Hadoop-env.sh

```
export JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-amd64"
```

b. Core-site.xml

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master:9000</value>
  </property>
  <property>
```

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

c. Hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.blocksize</name>
    <value>1073741824</value>
  </property>
</configuration>
```

d. Mapred-site.xml

In order to edit this file, use following command first.

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

Add following lines at the end of the file.

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.job.reduces</name>
    <value>16</value>
  </property>
</configuration>
```

e. Yarn-site.xml

Add following lines at the end of the file.

```
<configuration>
```

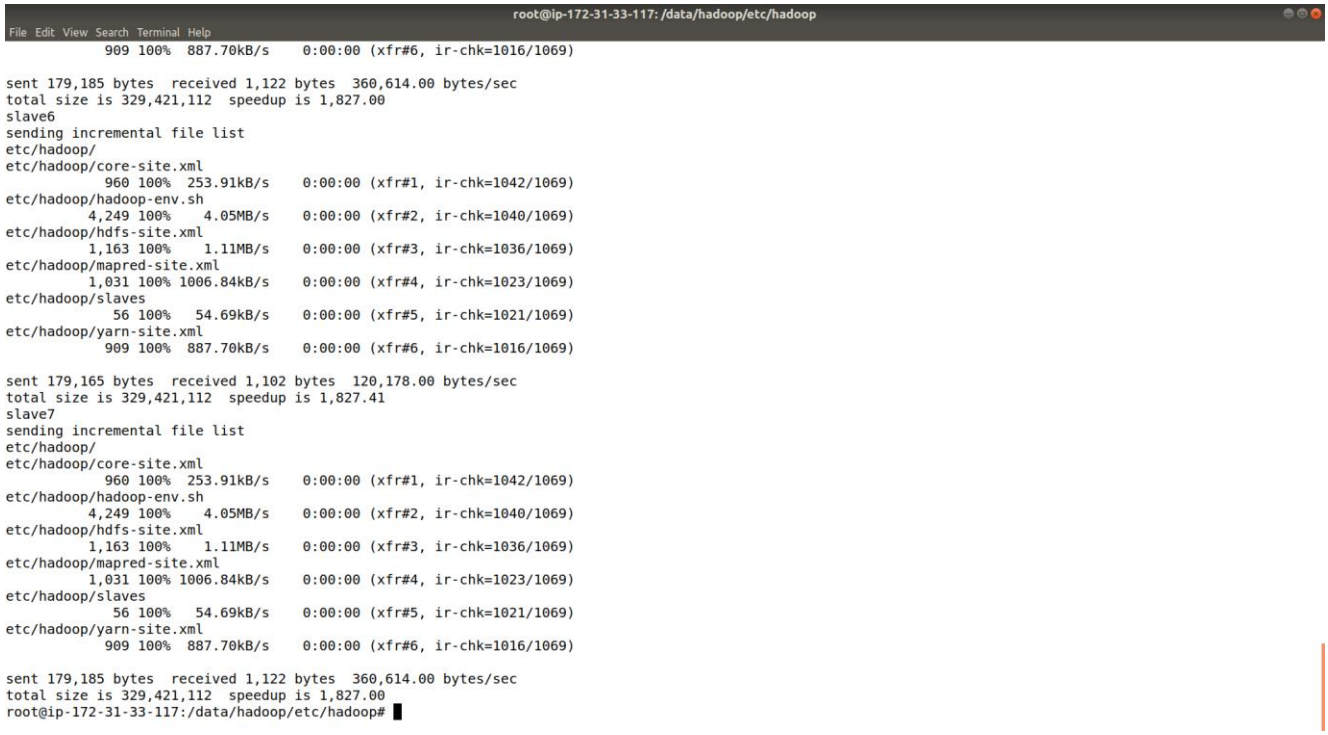
```
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>master:54311</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

f. slaves

```
master
slave1
slave2
slave3
slave4
slave5
slave6
slave7
```

7. Copy this configuration to all the other node using following script.

```
$ for i in `cat /data/hadoop/etc/hadoop/slaves`; do \
> echo $i; rsync -avxP --exclude=logs /data/hadoop/
$i:/data/hadoop/; \
> done
```

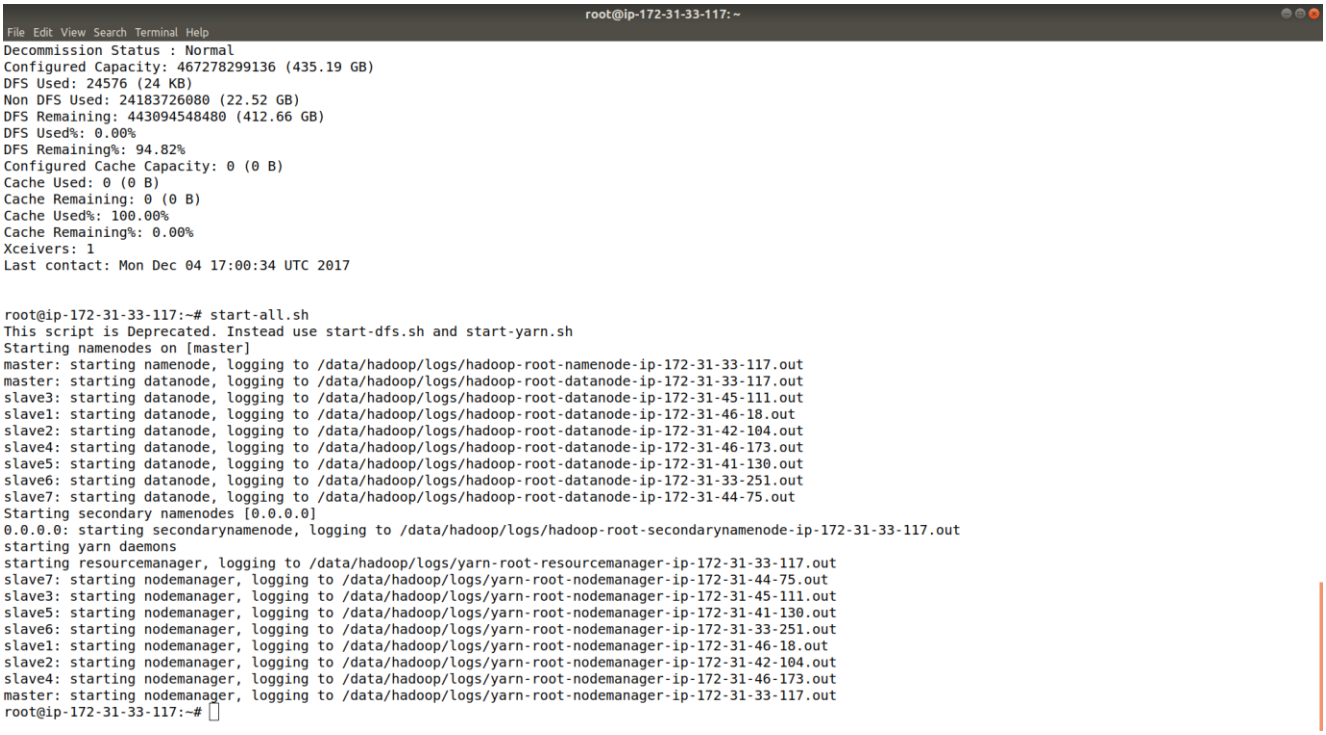


8. Export JAVA_HOME and HADOOP_HOME path using bashrc.sh script.

```
$ ./Mount.sh
```

9. Now use following command to format the namenode from master and run the daemons.

```
$ hadoop namenode -format
$ hdfs start-dfs.sh
$ hdfs start-yarn.sh
```



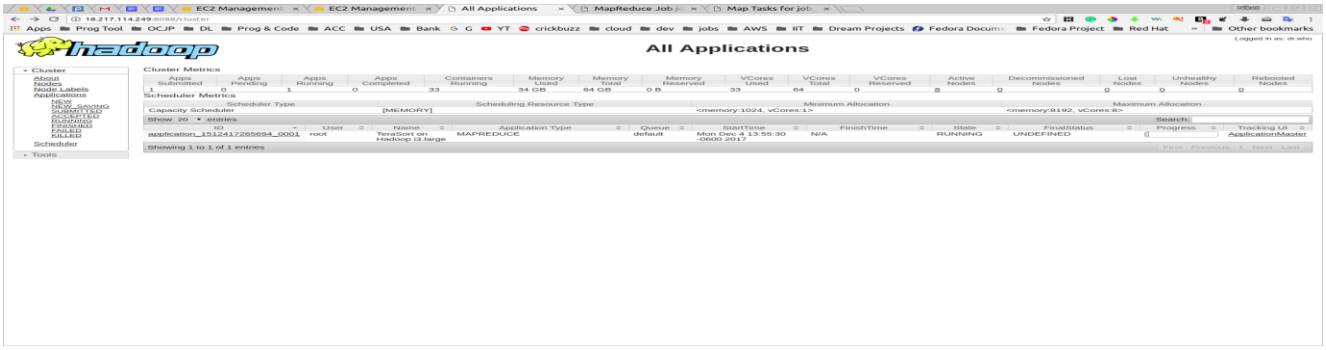
10. Now generate 128gb file using gensort using Gensort-128gb.sh script and load the data to the hdfs. This process will add 1 TB data to hdfs.

```
$ ./Gensort-128gb.sh
$ hdfs dfs -mkdir /input
$ hdfs dfs -put part1.txt /input
```

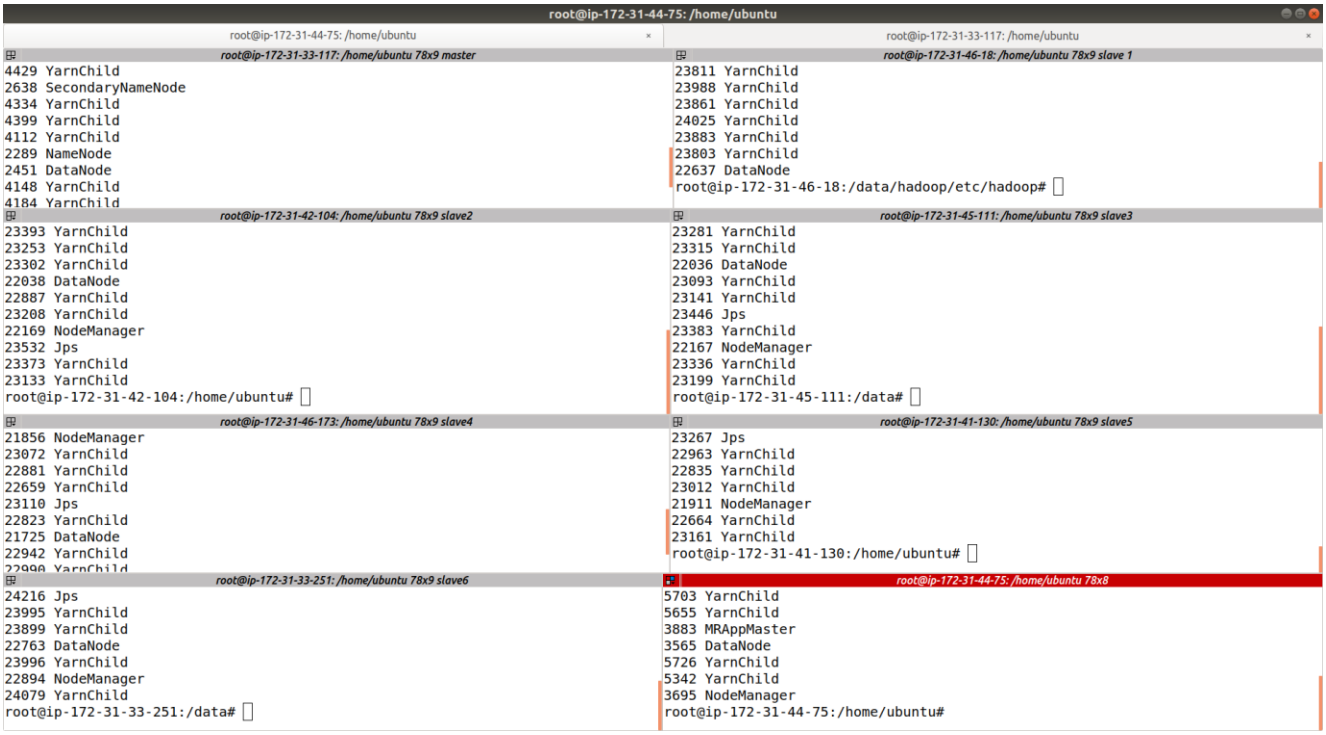
11. Now run the Hadoop program using following command.

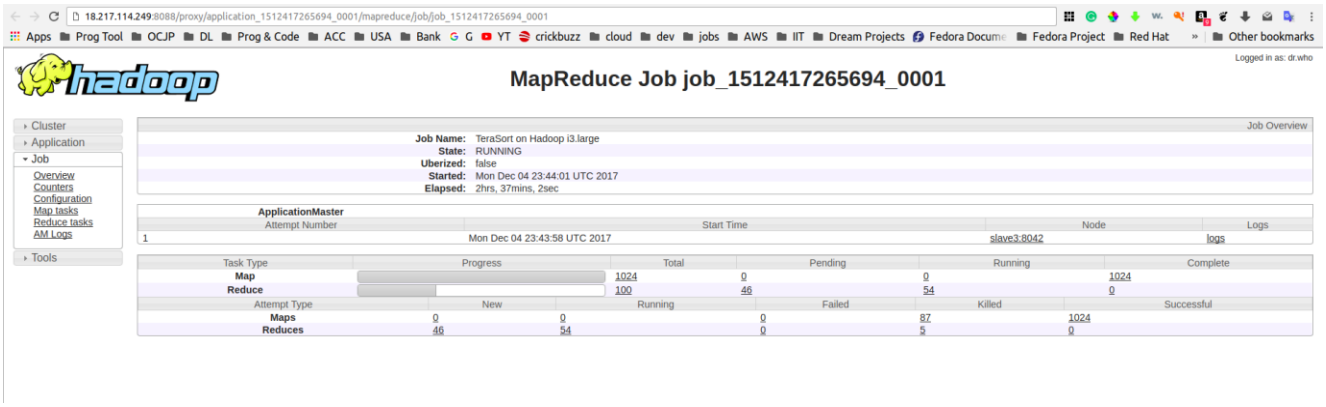
```
$ hadoop jar hadoop_demo-0.0.1-SNAPSHOT.jar  assigment.hadoop_demo.Terasort /input /output
```

We can also trace the process using public ip of master node and port number 8088.

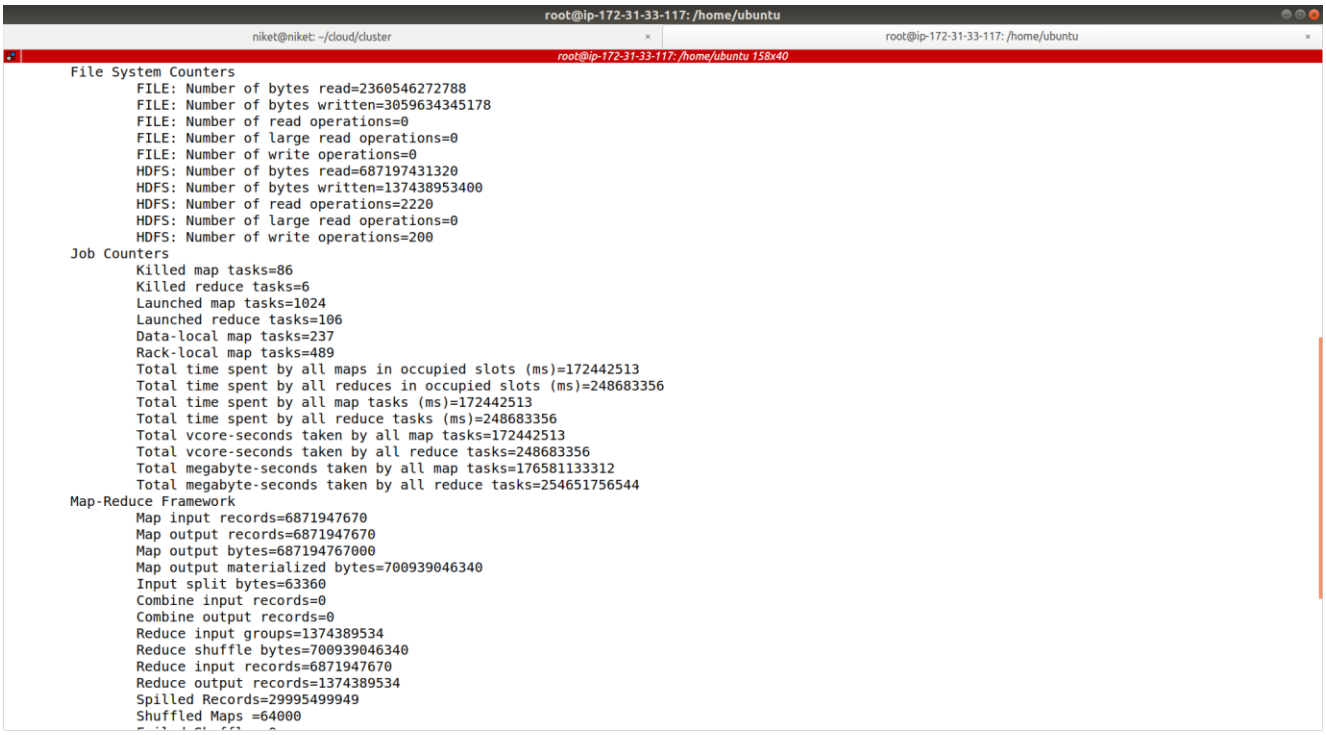


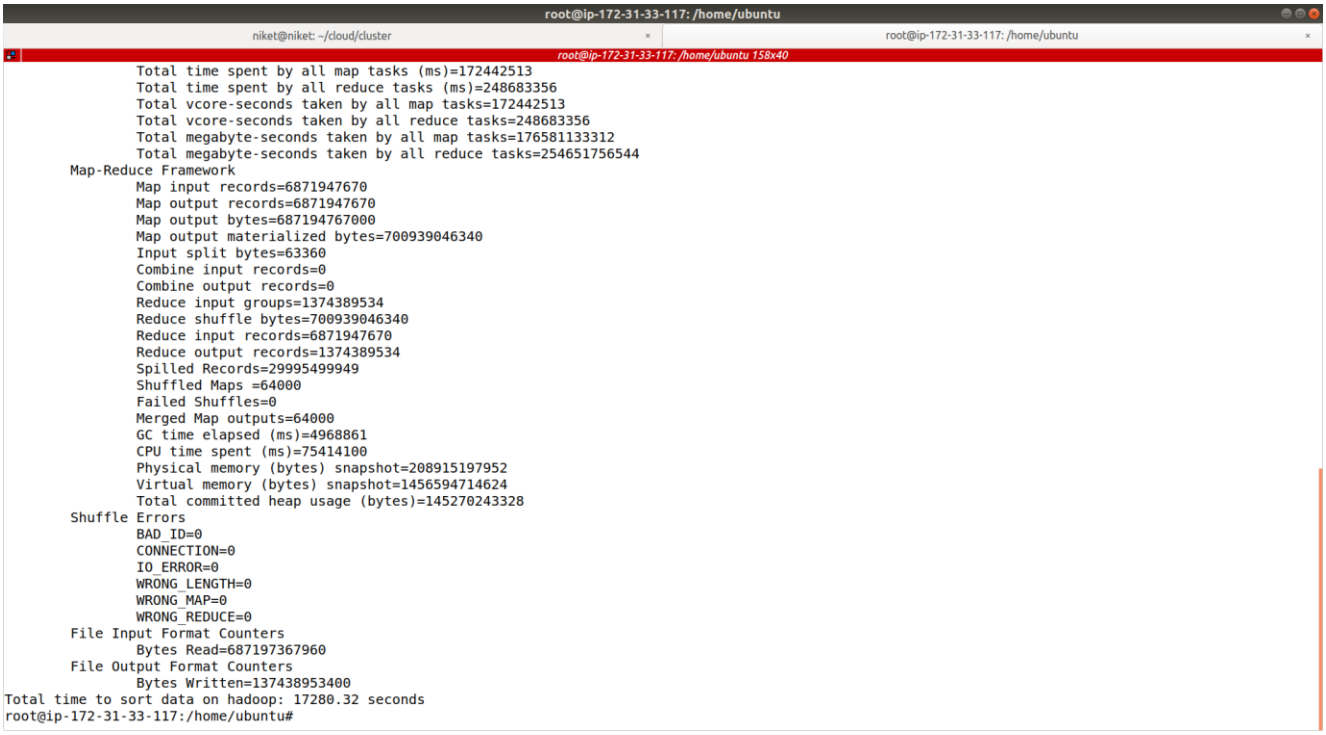
- 8 node JPS screenshot [ip of each node is visible on each terminal header.]





When Job completes.





- Validating data using ./valsort

Results for 1 TB Data.

Data size	1 TB = 1,099,511,627,776 Bytes
No of Mappers	1024
No of Reducer	16
Total Bytes Read (HDFS)	1099515926452
Total Bytes Write (HDFS)	1098375069600
No of Read ops (HDFS)	3369
No Of Write ops (HDFS)	200

Time Taken by whole process	17280 seconds
Throughput	63.62 MB/s

Difficulties:

To run the hadoop program over 8 node cluster, I haven't used any extra storage. So when the any slave node is running out of memory, nodemanager needs to clean the disk space by cleaning temporary file. Which was increasing time taken by mapper task. But to solve this, I used extra space using EBS and perform the Terasort. Which eventually gave me result almost similar to i3.4xlarge instance.

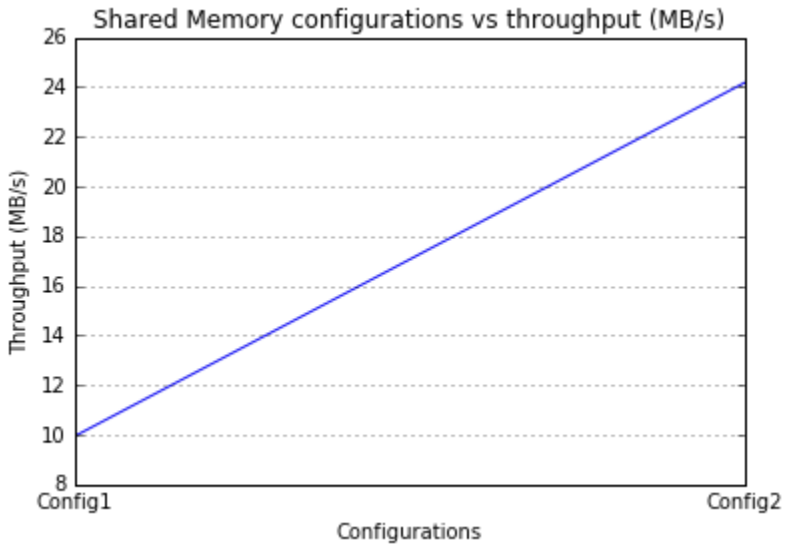
- **Conclusion**

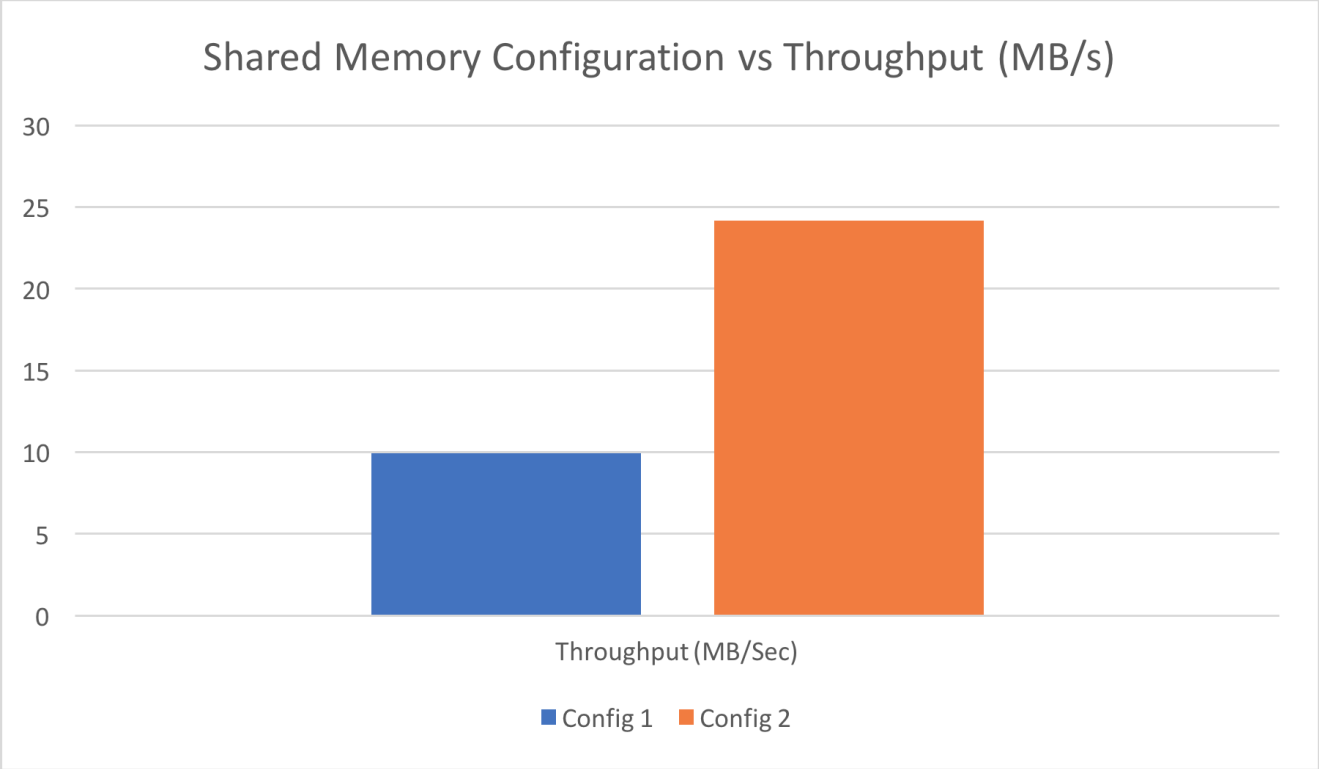
Experiment	Shared Memory Terasort	Hadoop Terasort	Spark Terasort
Compute Time (sec) [1xi3.large 128GB]	13925 seconds	8883.314 seconds	10080 seconds > 9.4 min [To sort data]
Data Read (GB) [1xi3.large 128GB]	645 GB	256 GB	256 GB
Data Write (GB) [1xi3.large 128GB]	384 GB	256 GB	256 GB
I/O Throughput (MB/sec) [1xi3.large 128GB]	9.947 MB/s	15.471 MB/s	13.63 MB/s
Compute Time (sec) [1xi3.4xlarge 1TB]	45485 seconds	18021 seconds	25920 seconds > 15 MIN [To sort data]
Data Read (GB) [1xi3.4xlarge 1TB]	5120 GB	2048 GB	2048 GB
Data Write (GB) [1xi3.4xlarge 1TB]	3072 GB	2048 GB	2048 GB
I/O Throughput (MB/s) [1xi3.4xlarge 1TB]	24.17 MB/s	61.01 MB/s	42.45 MB/s

Compute Time (sec) [8xi3.large 1TB]	N/A	17280 seconds	---
Data Read (GB) [8xi3.large 1TB]	N/A	2048 GB	---
Data Write (GB) [8xi3.large 1TB]	N/A	2048 GB	---
I/O Throughput (MB/sec) [8xi3.large 1TB]	N/A	63.62 MB/s	---
Speedup (weak scale)	N/A	4.11	3.11
Efficiency (weak scale)	N/A	0.51375	0.38875

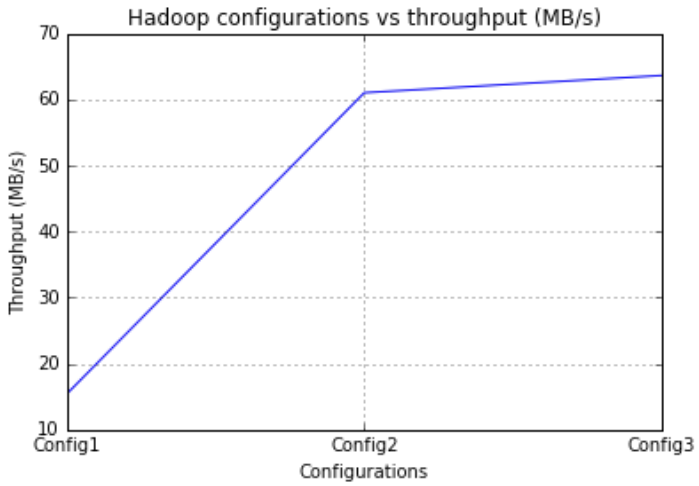
Here we have performed Terasort configuration for all three different configuration and with different environment. Following graph shows the performance for each configuration using their respective I/O throughput.

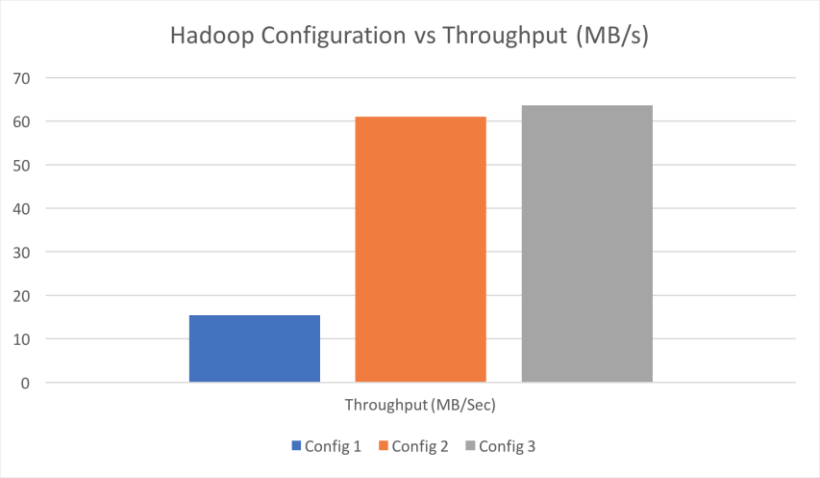
- Shared Memory Terasort



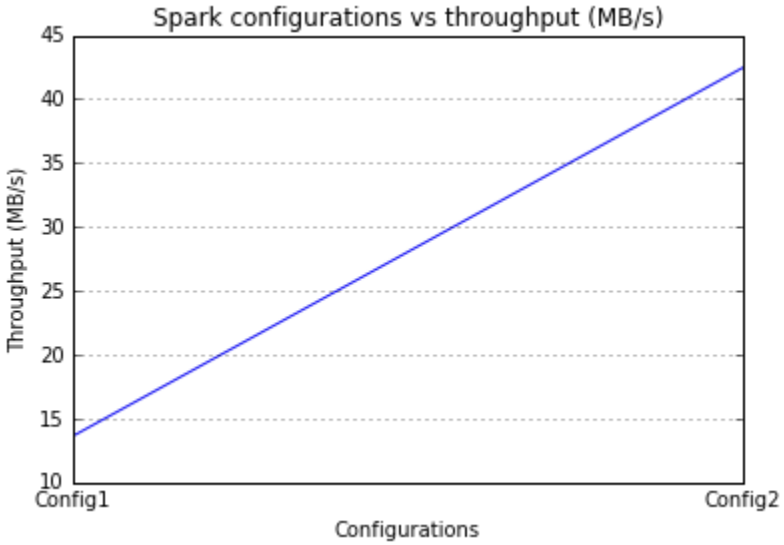


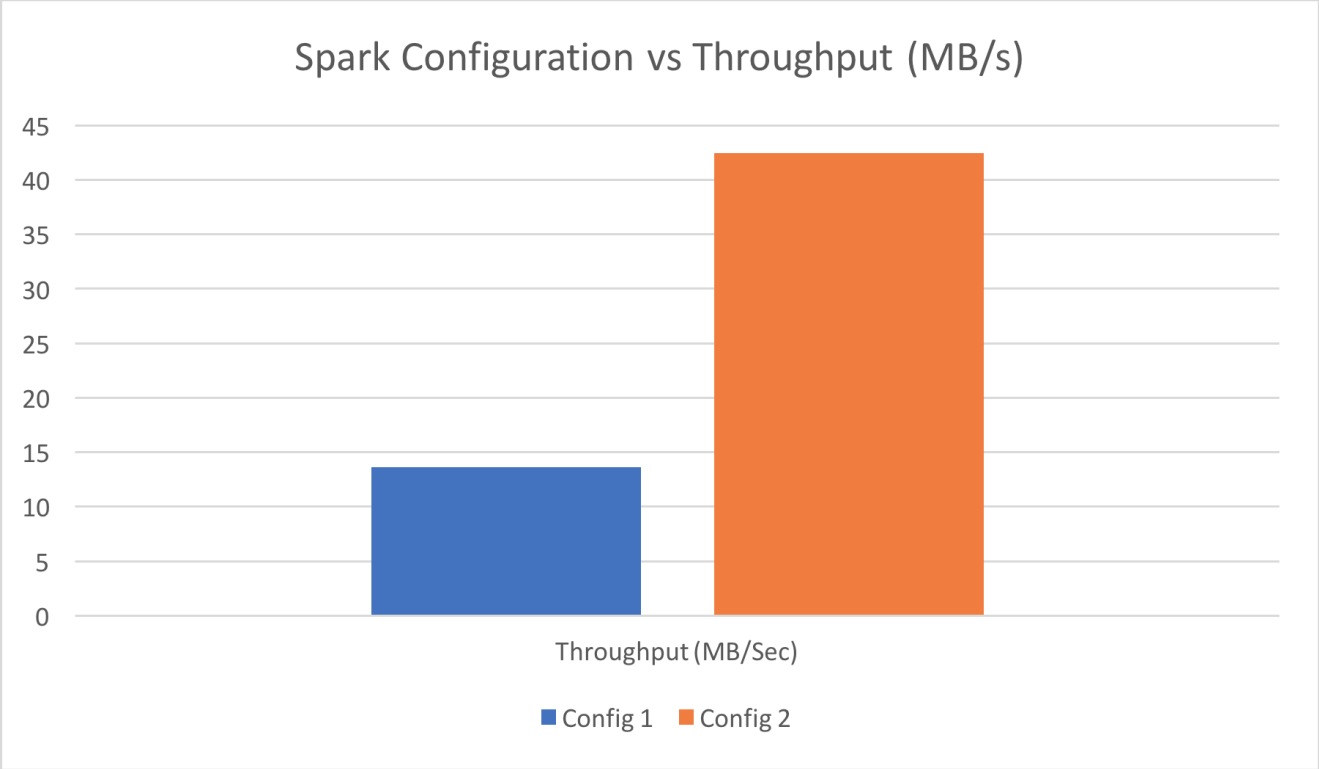
- Hadoop Terasort

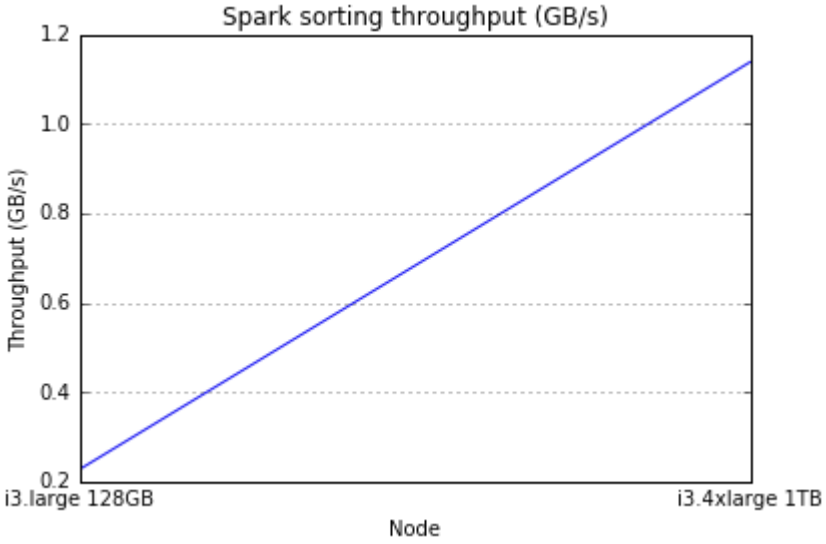




- Spark Terasort







Best Performance:

To sort the data, spark is best at all number of node as it is performing in memory sorting, spark always wins. The major bottleneck for spark is disk i/o.

Comparing with 2013 & 2014 Sort Benchmark Winner using data available from http://sortbenchmark.org/ .

Year 2013 Hadoop

Benchmark configuration: (2013)

System: 2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)
Throughput: 4328 seconds to sort 102.5 TB data. I.e. 23.68 GB/Sec

If we consider it is linear scaling and find throughput for 8 node cluster it will give us throughput around 90.20 MB/sec.

By comparing our 8 node cluster result with this, we can say that our Cluster performance is almost 70% of the standard benchmark.

Year 2014 Spark

Benchmark configuration:

System: 207 nodes **x**(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD
Throughput: 1406 seconds to sort 100 TB data. I.e. 71.12 GB/Sec

If we consider it is linear scaling and find throughput for 1 node cluster it will give us throughput around 343.59 MB/sec.

CloudSort Benchmark

The paper presents a benchmark called cloudsort for IO-intensive workloads, that calculates a minimum cost of sorting a fixed number of records for any public cloud. It is a total-cost-of-ownership benchmark based on an external sort. It is better than other benchmarks in terms of Accessibility, Affordability, and Auditability.