Report

1. Problem Description

Performance of sorting a large data set measured using Shared Memory Sorting, Hadoop and Spark. The performance of shared memory sorting, Hadoop sort and spark sort are measured on data of 10 GB with 1 node. The performance of Hadoop sort and spark sort is measured on dataset 100 GB on 17 nodes (1 master 16 slave).

2. Shared Memory Sorting

a. Description:

Shared memory sorting is an implementation of the external merge sort written in java which uses a combination of in memory sort and merging the in memory sorted data.

b. Environment settings:

The environment in this case was a Java Virtual Machine which was used to run the java code on the instance. The Heap Size had to be increased in order to match the high need of storing the in memory data.
OS distribution: Linux (preconfigured Hadoop AMI)
Java version: java-1.6.0-openjdk-devel (for jps and other developer tools)

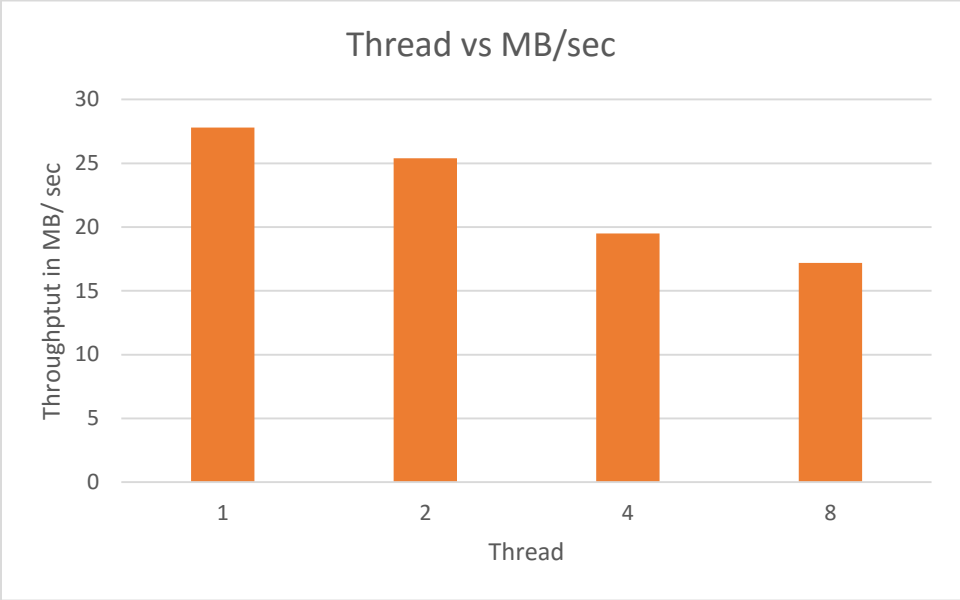
Problem Encountered

A) Java out of Memory

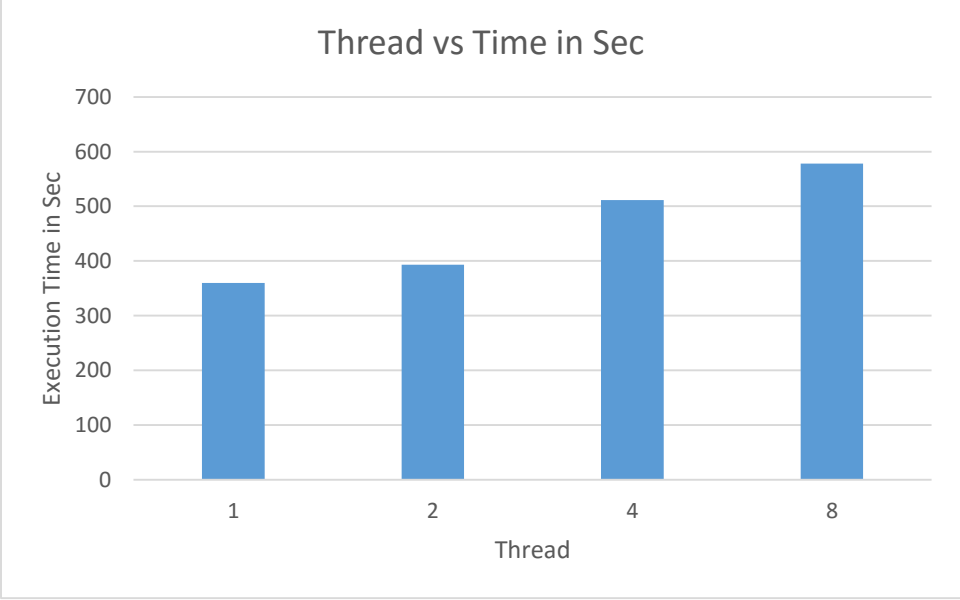
Solution: Increase Heap memory was increased –Xmx 3G
The program was run on a 10GB dataset with 1, 2, 4 and 8 threads.
Results were:

Threads	Time(sec)	MB/sec
1	359.6	27.8
2	393.15	25.4
4	511.6	19.5
8	578.31	17.2

Graphs:



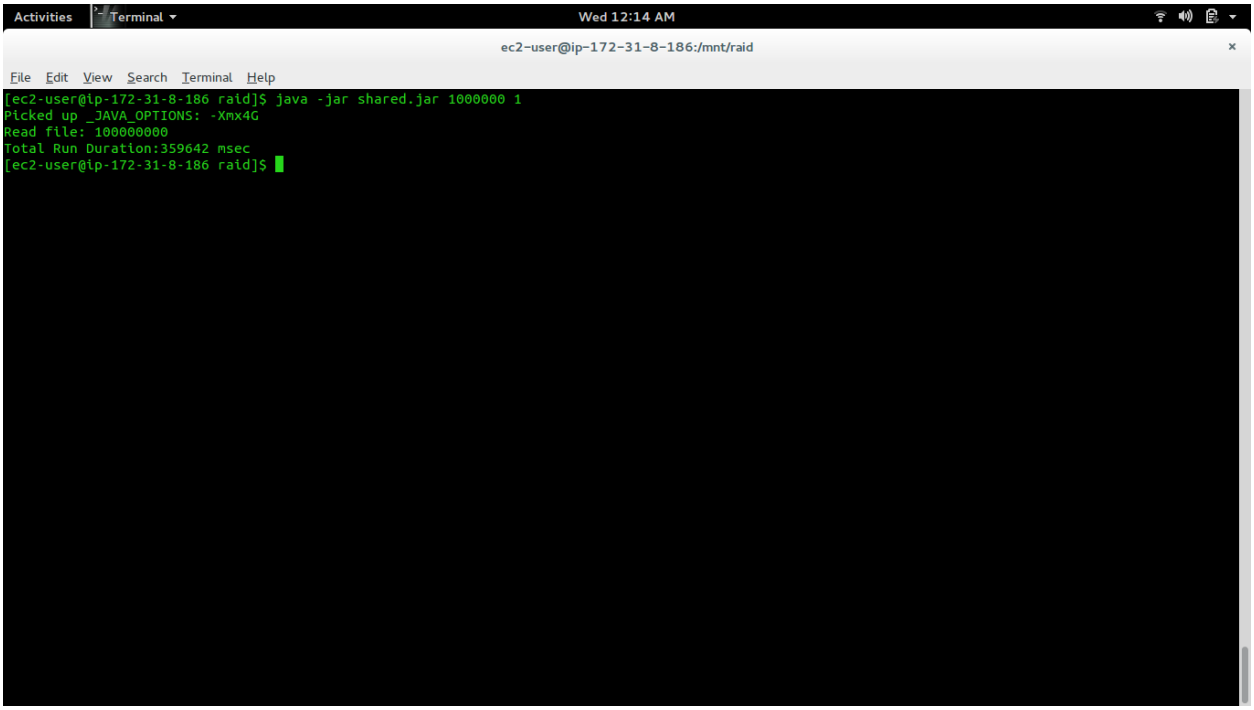
Graph: Threads vs MB/sec



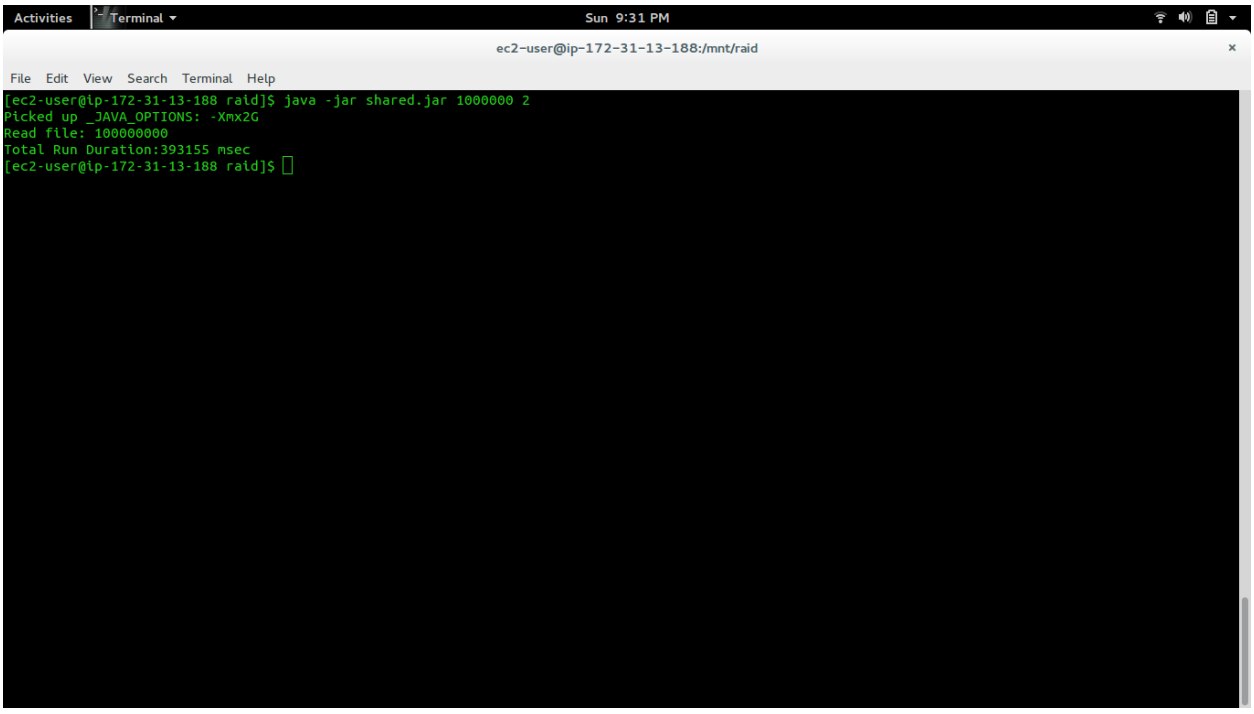
Graph Execution Time vs Threads

Screenshots:

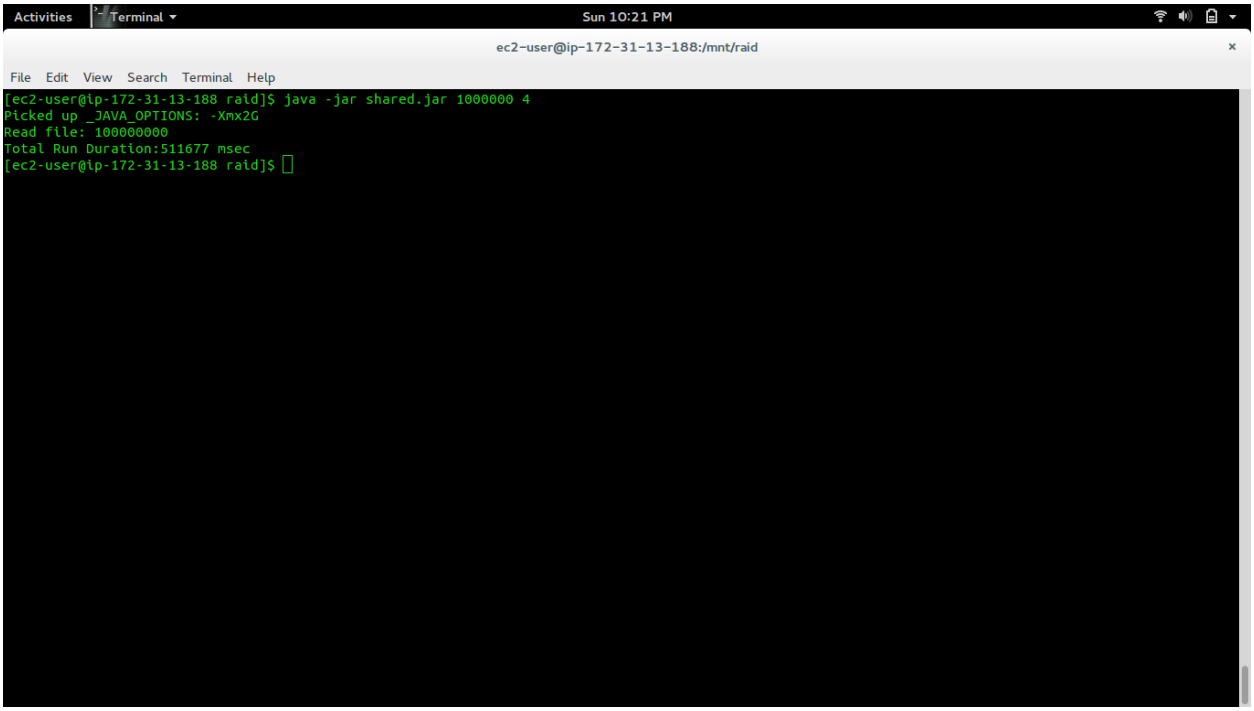
1 Thread



2. Threads



4. Threads



Explaining Results:

The Best Output is found for 1 and 2 threads as maximum parallelism is achieved because the CPU has 2 vCPU.

3. Hadoop

a. Description:

Shared memory sorting is an implementation of the external merge sort written in java which uses a combination of in memory sort and merging the in memory sorted data.

b. Environment settings:

The Hadoop environment had to be configured and file had to be modified as described in the readme.txt. An AMI was built with Initial master preconfigured with IP addresses. Then the slaves were built from the AMI and raid storage was added to the instance. Pssh was used to configure the raid on the slaves.

Hadoop Version: 2.7.2

OS distribution: Linux

Java version: java-1.6.0-openjdk-devel (for jps and other developer tools)

Problem Encountered

B) Datanode not starting even though namenode and secondary namenode are running.
Solution: Delete Hadoop temp directory and format namenode and restart.

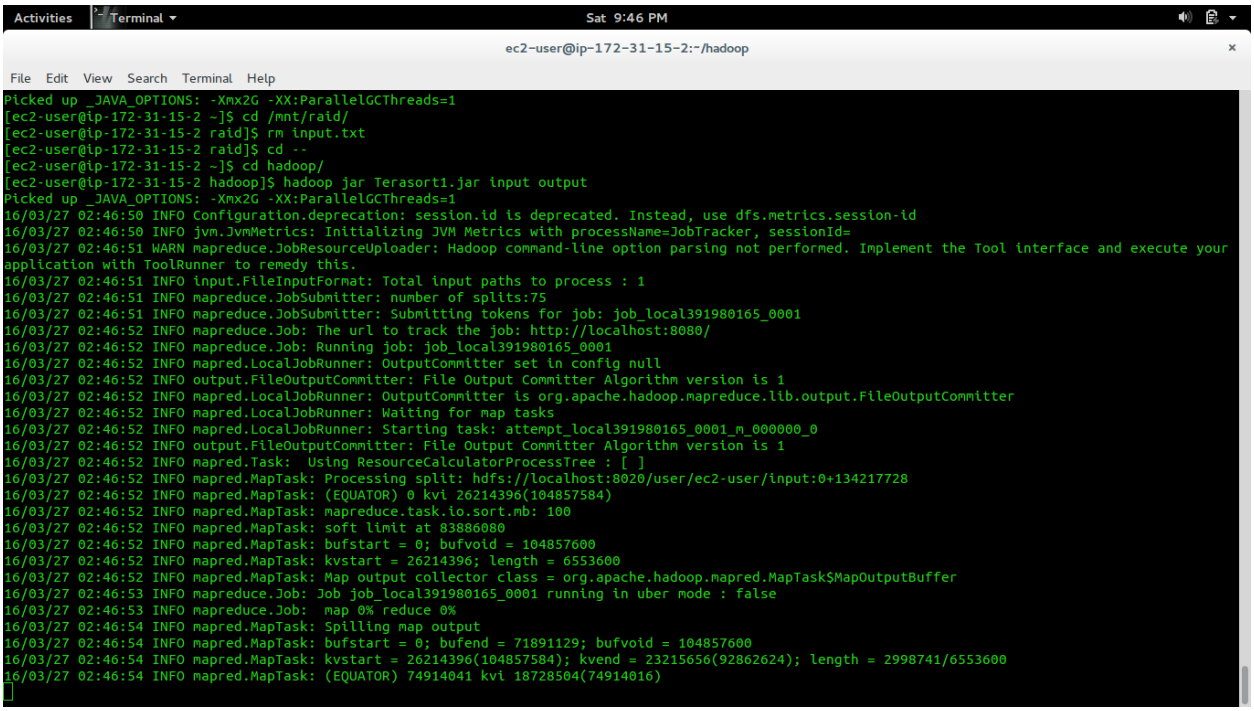
C) Slave not detected
Solution: Change Slave file from localhost to IP address and add Master file at each slave.

c. Results and Graphs

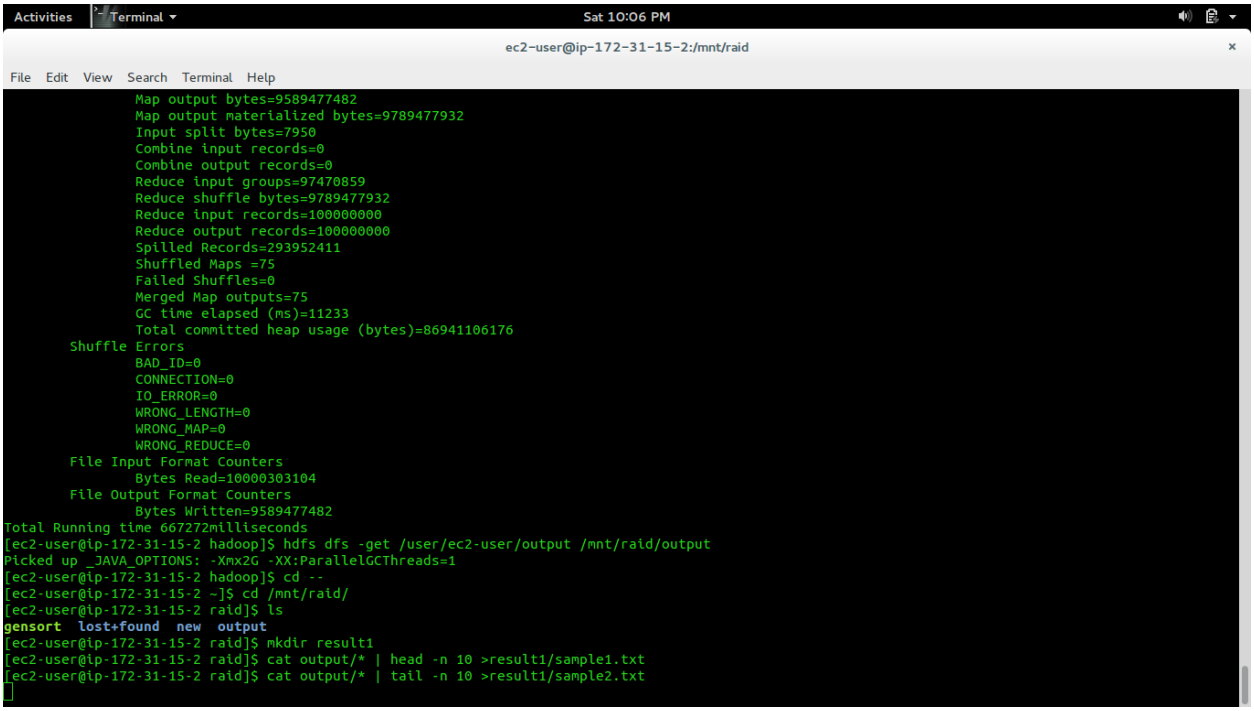
Nodes	Time	MB/sec
1	667.2	14.9
16	10220.2	9.78

Screenshots

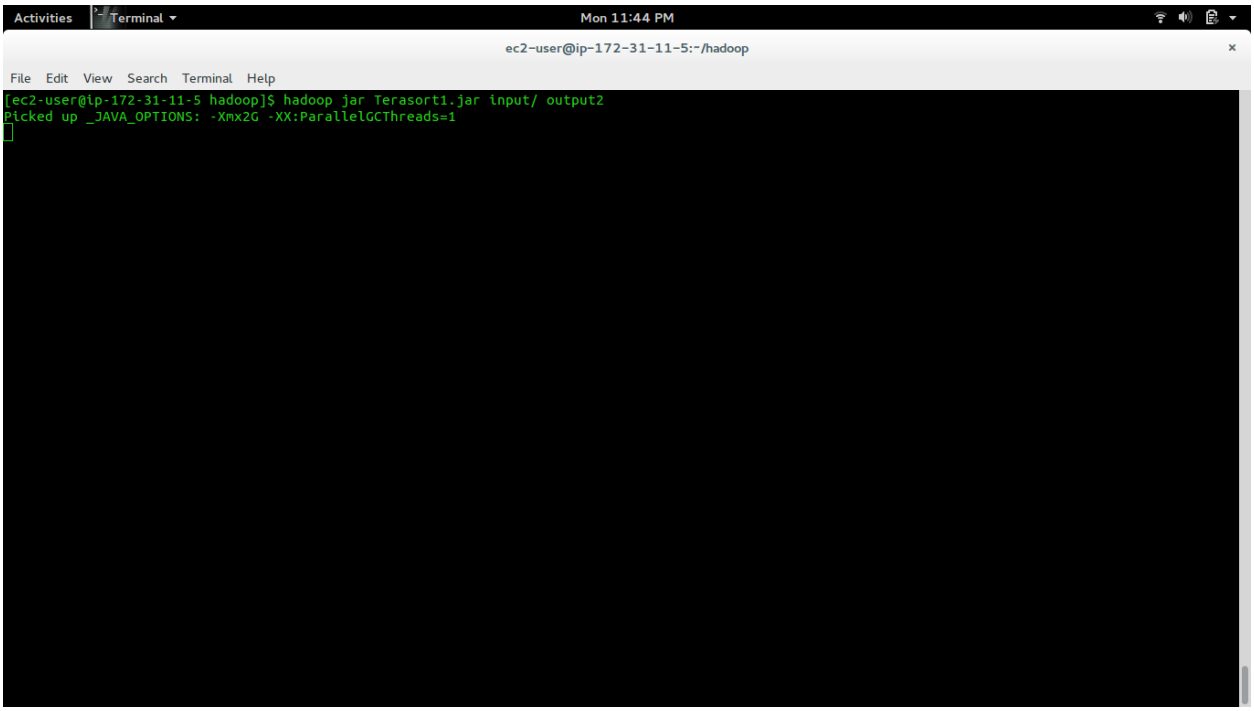
I. 10 GB 1 Node Start



Ii. 10 GB 1 Node Finish



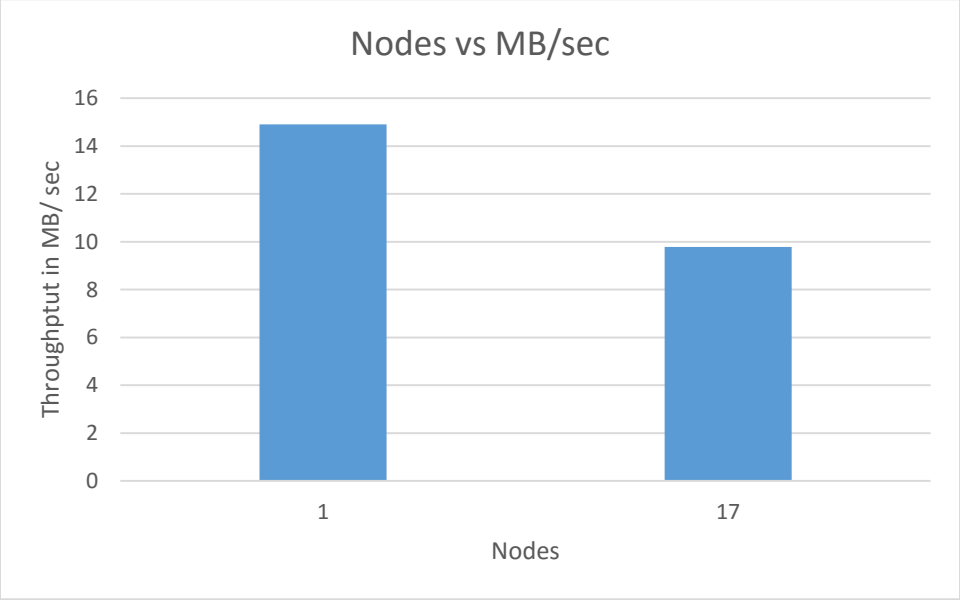
Iii. 16 node 100 GB Start



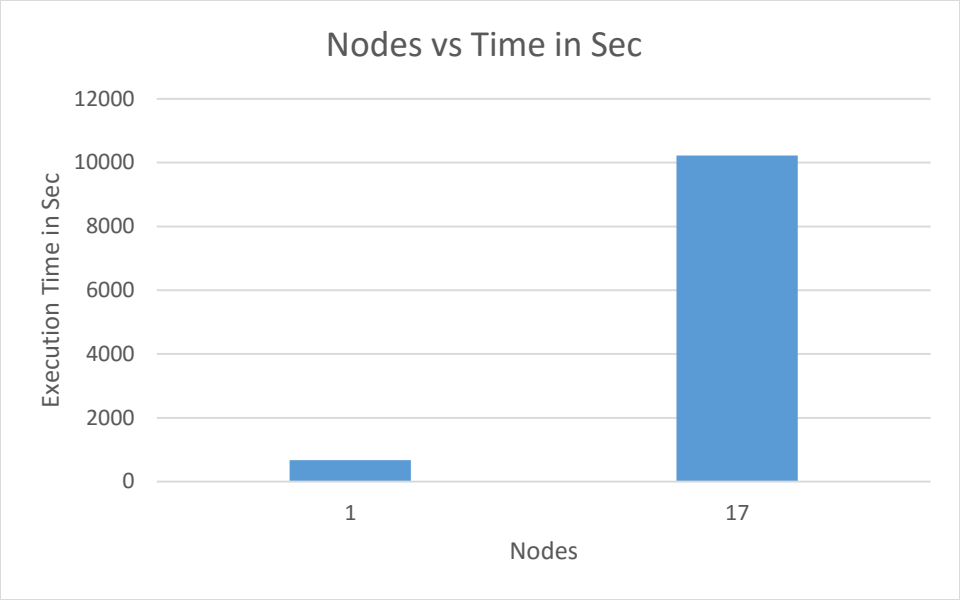
Iv 16 node end

```
Activities  Terminal  Tue 2:47 AM
ec2-user@ip-172-31-11-5:~/hadoop
File  Edit  View  Search  Terminal  Help
                FILE: Number of write operations=0
                HDFS: Number of bytes read=37404037074944
                HDFS: Number of bytes written=95894739072
                HDFS: Number of read operations=559501
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=748
        Map-Reduce Framework
                Map input records=1000000000
                Map output records=1000000000
                Map output bytes=95894739072
                Map output materialized bytes=97894743542
                Input split bytes=110260
                Combine input records=0
                Combine output records=0
                Reduce input groups=968948158
                Reduce shuffle bytes=97894743542
                Reduce input records=1000000000
                Reduce output records=1000000000
                Spilled Records=3998657822
                Shuffled Maps =745
                Failed Shuffles=0
                Merged Map outputs=745
                GC time elapsed (ms)=162112
                Total committed heap usage (bytes)=1504549273600
        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=100003047424
        File Output Format Counters
                Bytes Written=95894739072
Total Running time 10220344milliseconds
[ec2-user@ip-172-31-11-5 hadoop]$
```

Graphs:



Graph: Hadoop 1 node vs 17 nodes



Nodes vs Time in Seconds

D. questions

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

Ans. Master is the node that control the flow of data and execution over the cluster and manages all the slaves. The Namenode that manages storage and job tracker used to keep track of the running jobs.

Slaves usually run jobs assigned to them by the master and slaves can also be used as data store to maintain duplicates.

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

Ans. The unique ports help us identify, communicate and check if the node is active. Node can be uniquely identified in the network. If the same port is used for each user it will be difficult to communicate and identify.

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

Ans. The number of mappers and reducers can be changed by using -D mapred.map.tasks and -D mapred.reducer.tasks specified at runtime or the mapred-site.xml could be edited and mapreduce.job.running.map.limit could be set.

4. Spark

b. Description:

Shared memory sorting is an implementation of the external merge sort written in java which uses a combination of in memory sort and merging the in memory sorted data.

b. Environment settings:

The Spark Environment is built over the Hadoop version 2.6 and later. Hadoop is firstly configured and then spark is started over the Hadoop cluster. The Hadoop environment is adjusted according to the multimode cluster as setup above and then Spark for Hadoop 2.6 and later is configured to use the Hadoop HDFS and job tracker. The Spark environment is adjusted to use the raid for storage. Detailed description for files modified and changed is mentioned in read

Spark Version: 1.6.0
Hadoop Version: 2.7.2
OS distribution: Linux (preconfigured Hadoop AMI)
Java version: java-1.6.0-openjdk-devel (for jps and other developer tools)

Problem Encountered

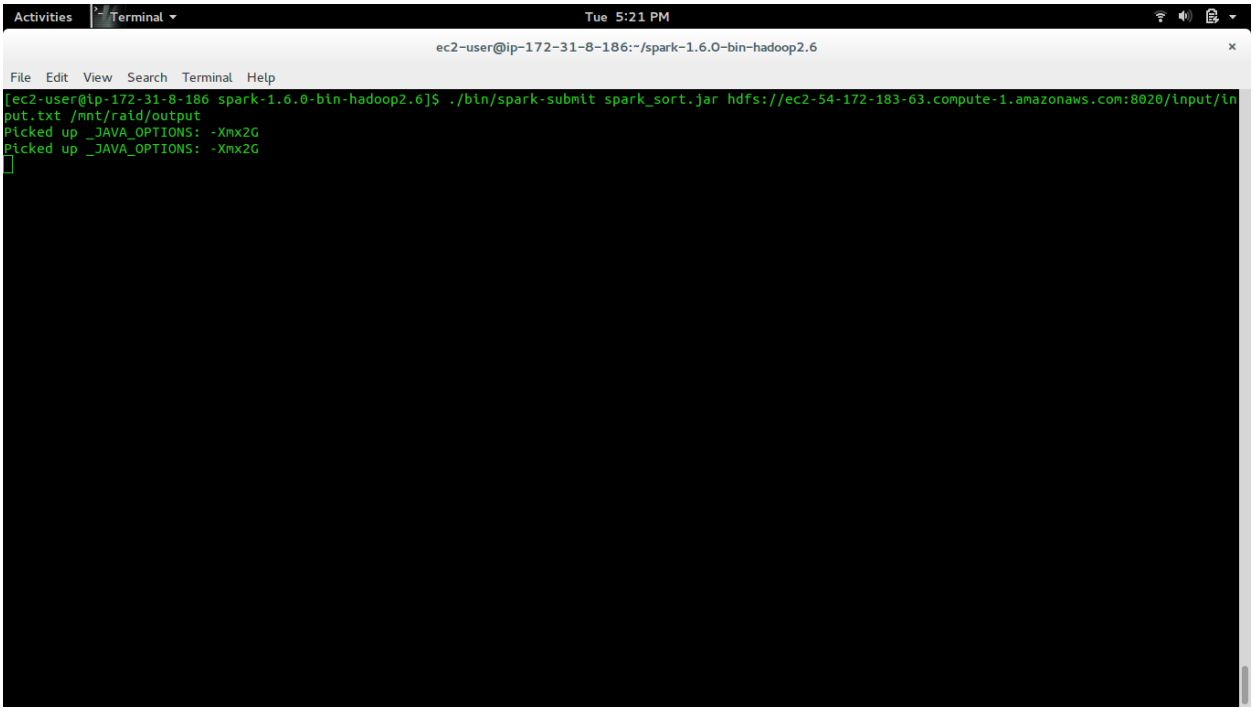
A) Memory Full on device
   Solution: Changed Temp directory for Spark to Raid 0

c. Results and Graphs

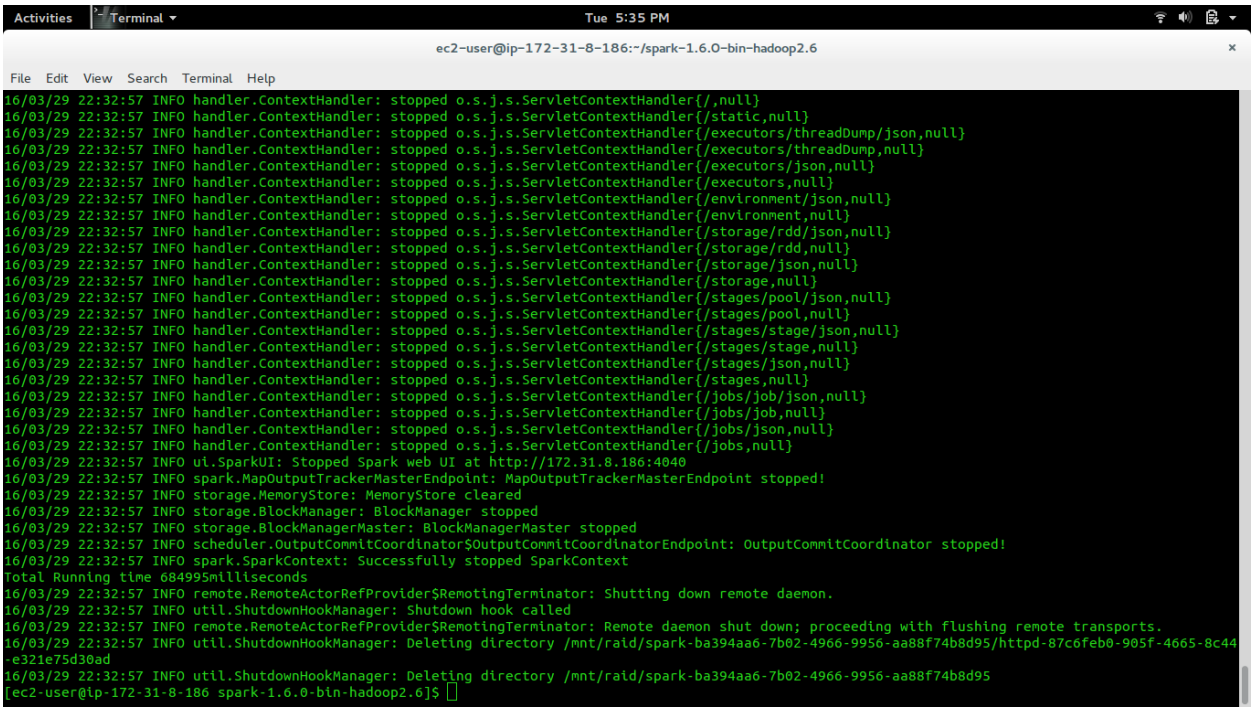
Nodes	Time	MB/sec
1	684.9	14.6
16	10024.9	9.97

Screenshot

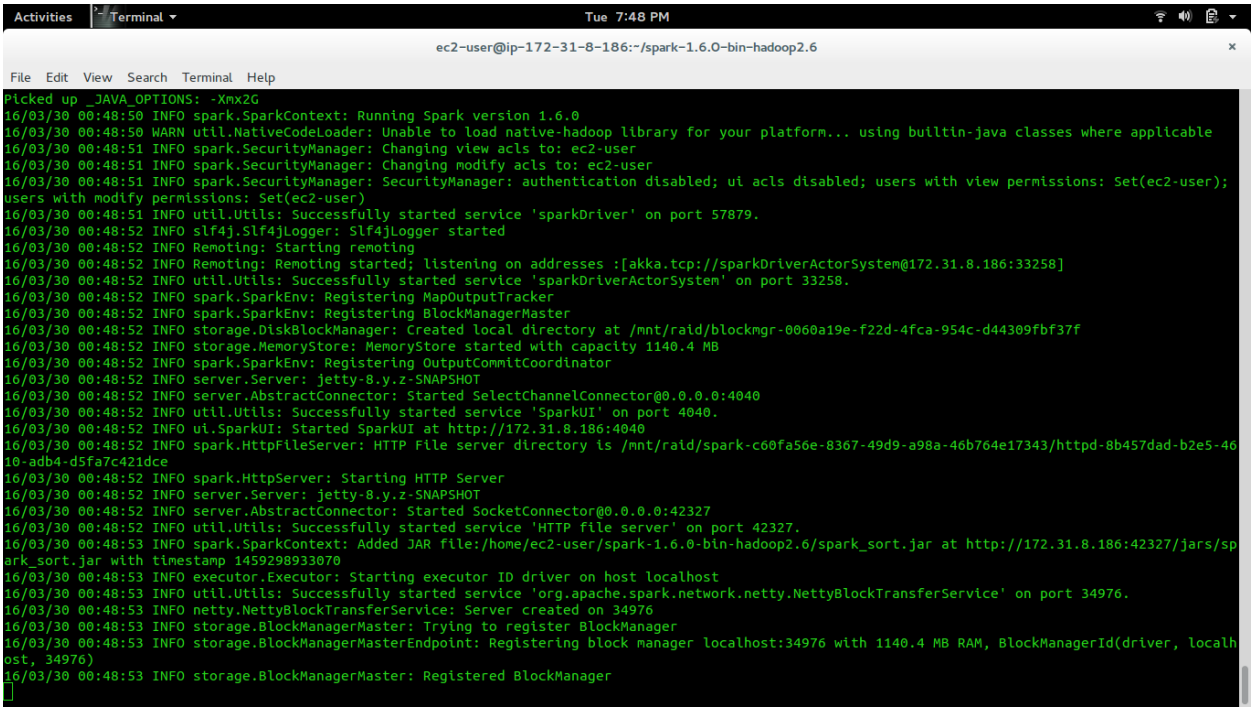
I. 1 node start



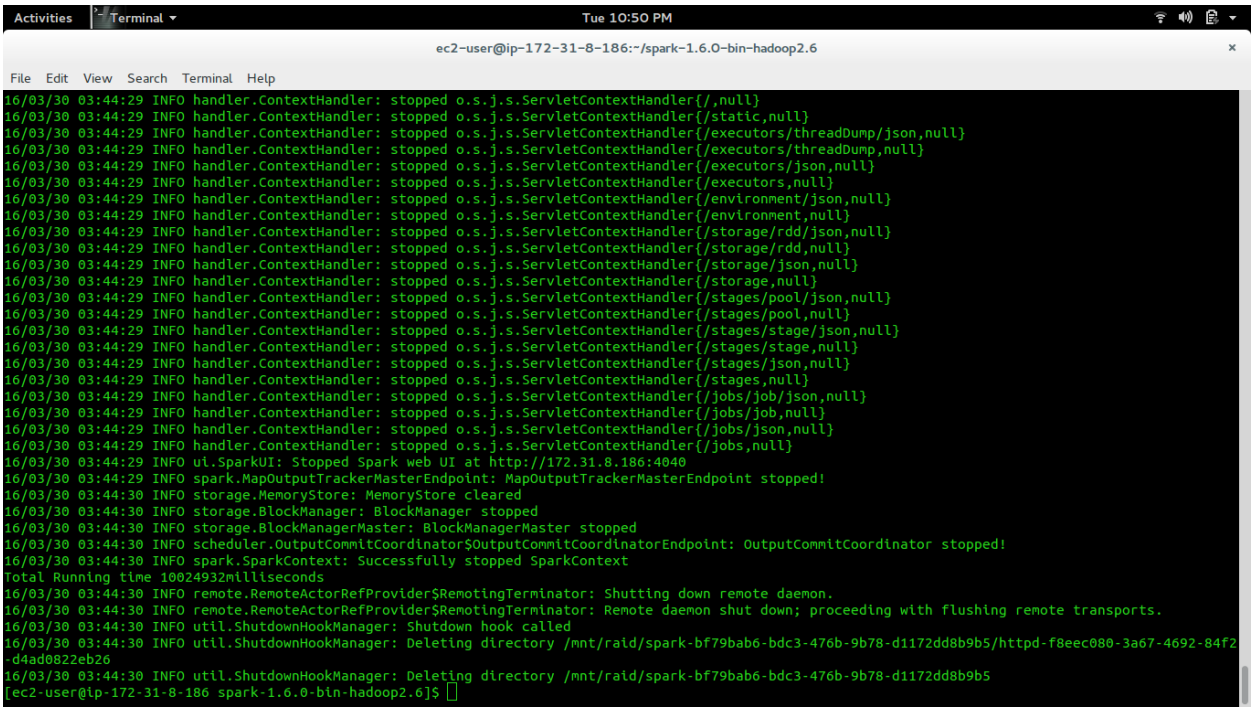
Ii 1 node end



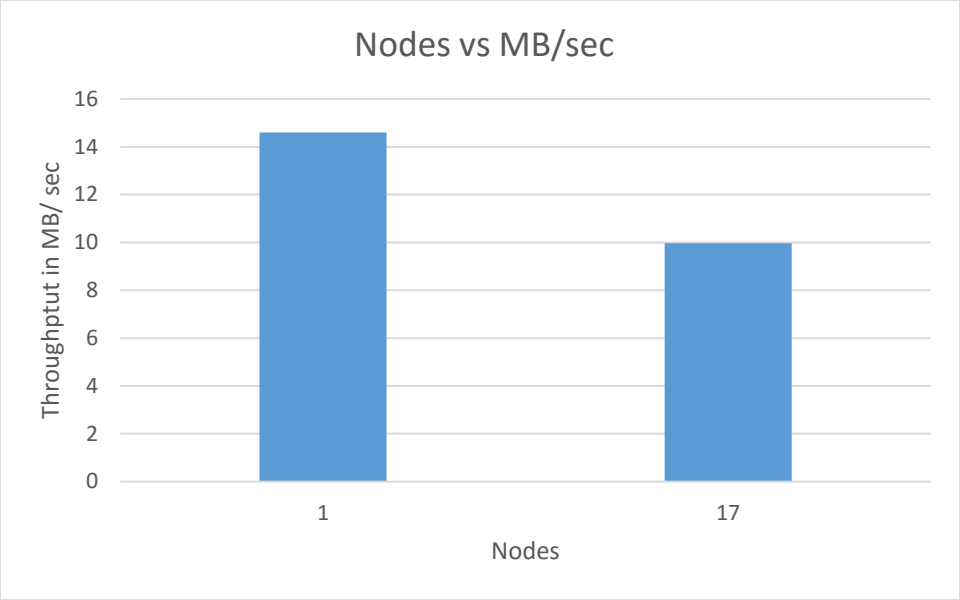
Iii. 16 node start



Iv 16 node end



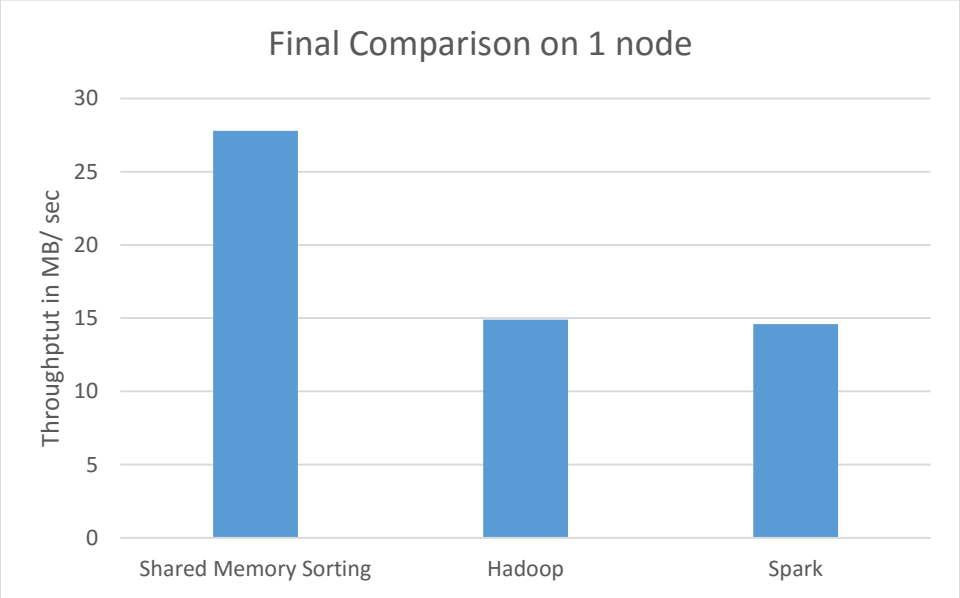
Graphs:



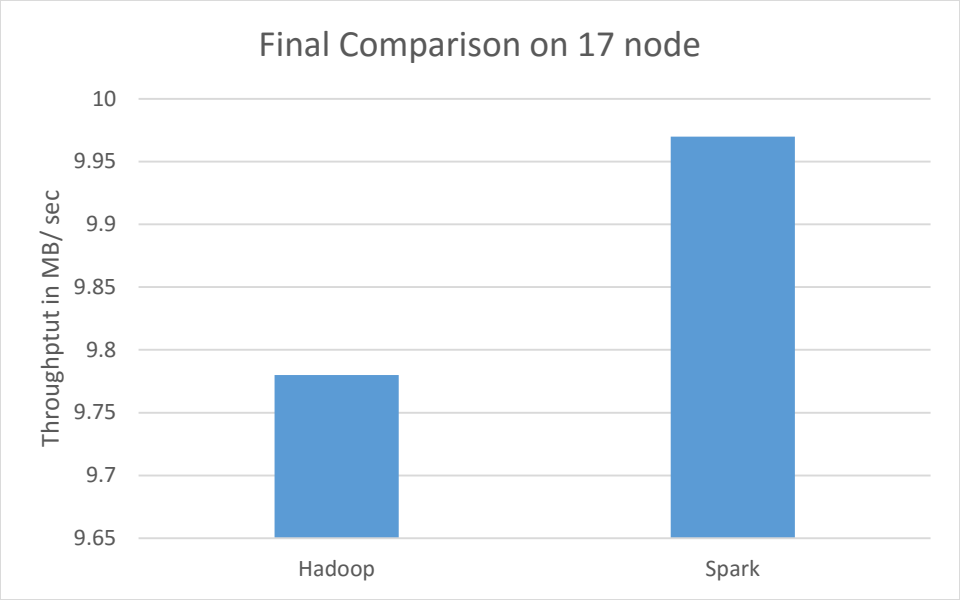
Graph: Hadoop 1 node vs 17 nodes

5. Performance

a.Graphs



Graph: Comparing performance on 1 node for each framework.



Graph: Comparing performance on 17 node for Hadoop and Spark.

b) Results and Conclusion

1) What conclusions can you draw?
   Ans. For 1 node the best performance is given by Shared Memory sort as it doesn't involve overheads involved in Hadoop and Spark.
   For 17 nodes the performance of Spark is slightly better than that of Hadoop but the factor is very less.

2) Can you predict which would be best at 100 node scale?
   Ans. As seen from the comparison Spark performance much better in 17 nodes and close to Hadoop in case of 1 node. Hence Scaling up the node will boost the performance for Spark as compared to Hadoop.

3) Can you predict which would be best at 1000 node scale?
   Ans. The above answer indicates the same as we scale the nodes the performance gets better for Spark, and hence I feel that Spark will perform well in case of 1000 nodes

4) Compare your results with those from the Sort Benchmark [9], specifically the winners in 2013 and 2014 who used Hadoop and Spark.
   Ans. The winner of sort benchmark in 2014 with Apache Spark gave a throughput 4.27 TB/min 207 Instances and sorting a total of 100 TB. The winners in the 2013 experiment with Hadoop did sorting 104 TB with the throughput of 1.42 TB/sec implemented on 2100 nodes. The Throughput is very high as compared to our experiments which is approximately which is approximately 594 MB/min for Spark and 582 MB/min.

5) What can you learn from the CloudSort benchmark?
   Ans. The CloudSort total cost ownership Benchmark that uses public cloud and majorly aims for IO intensive applications. It is developed for the teams that have limited budgets and require high end hardware to run the external sort algorithm. It prefers external sort as it is IO intensive and has high workloads.