Report - Cloud Computing

Programming Assignment 2

vinit shah (A20350453)

Problem: - The Problem of Programming assign 2 is to sort a Small (1 GB), 10GB and 100GB and Large (1 TB) data sets using 3 different ways of sorting, like Shared Memory, Hadoop and Apache Spark. To compare and contrast the Terasort application of each of the methodologies so as to come up with the outcome as which is the better way to sort the large dataset in a minimum Time and Maximum speed, so as to increase the performance of sorting of large dataset.

Methodologies used for Terasort Application:-

1. Shared Memory
2. Hadoop
3. Apache Spark

Server Used:

AWS EC2 Instance, d2.xlarge, C3.large, with Ubuntu Server 14.04 LTS

d2.xlarge configuration

VCores- 4

Memory GB- 30.5

Storage- 3 * 2000 GB

Processor- Intel Xeon E5-2676 v3

C3.large Configuration

Vcores- 2

Memory- 3.75

Processor-

Intel Xeon E5-2680 v2

Software Versions:-

Apache Ant – 1.9.6

Java- OpenJdk7

Hadoop – Hadoop 2.7.2

Spark- spark-1.6.0-bin-hadoop2.6

Setup a Cluster:-

How to setup a cluster is addressed in each of the respective section of Hadoop and Spark.

Shared Memory

Shared memory Tera-sort is an in-memory sort of a large dataset, the Idea of implementation is as follows:

Sort Phase:

The Large dataset might not fit inside the memory, so the best way to sort the data in memory is to divide the large data set into chunks of size say Block Size, and read the block size of data in memory and sort them in-place, now write that temporary sorted data into the temporary file which will then be added to the list of temp files.

Merge Phase:

During the merge phase, it will open all temp files in the list and read the first line of data from each of the file and sort it based on Priority queue logic. Now the 1st line of data from each temp file is sorted so, it will then be added to new temp. File and likewise the steps followed. Now once in the end we will have two temps. Files to merge at that time, the data is read and compared against each other and written it back to output file in the sorted form.

Here, the dataset is such that out of 100 bytes of each line. First 10 characters are reserved for the key rest characters are the value. So the sorting data is done based on the key, and merge is also done on key, giving sorted file as an output file.

Observations of Shared Memory:

Now the experiment, in the Assignment is to run the shared memory, for 1 to 8 threads and report the best time for sorting the dataset.

As the Program runs for 1 thread to 8 threads, from the time and reading it's been observed that as you increase the number of threads, the time required to sort a dataset is nearly constant, as the number of cores available for C3.large instance is 2, so when the shared memory job runs of 1 thread, it will utilize the 2 cores, so that the time required will be less. Now the job with thread 2 will distribute their task among two cores, resulting in the highly fast running of sort. Now after 3rd thread because of limitation of disk head and number of cores the time required to sort the job will take almost the same.

The best time of sorting the large file is with the 2 threads as, noticed by running the program for several time for each number of threads, but by looking at the results of output of each thread, the time required to sort the dataset file with multiple threads is almost constant, so that it's advisable to sort the file using a single thread.

Also, the screenshot for each of the dataset sorting is given in their separate folder:

Execute the following script to install required software and dependencies:

Basic_Install.sh

To generate dataset execute the following script:

install-gensort.sh

Shared Memory Readings:

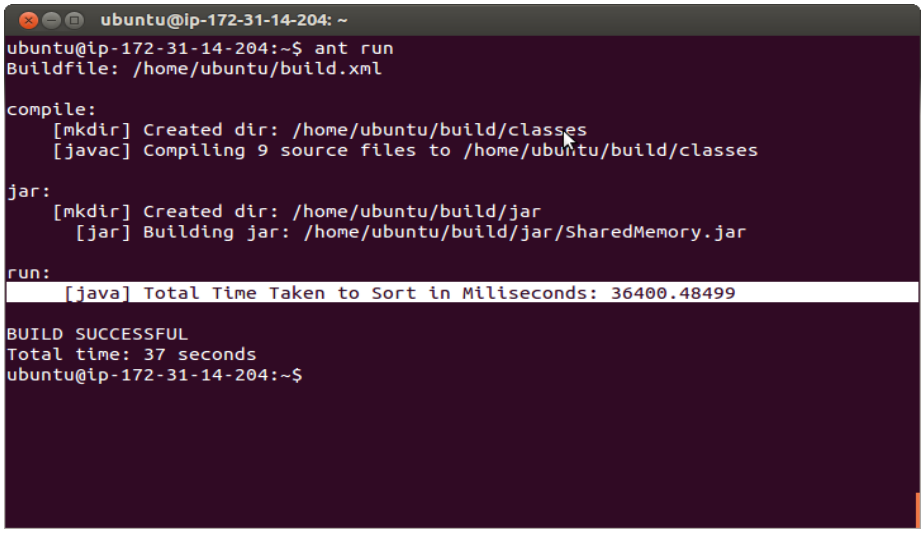
Dataset Size	Number of Nodes	Time (Sec)	Time (Minute)	Throughput (MB/Sec)
1 GB (D2.xlarge)	1	36.40048	0.6	28.44
1 GB(C3.large)	1	32 (Best for 2 threads)	0.53	32
10 GB(C3.large)	1	409 (Best for 2 threads)	6 Minute 49 Sec	25.20
1 TB (D2.xlarge)	1	14374	239 Minutes 34 Sec	71.23

Shared Memory 10 GB sort in Multi-threading:

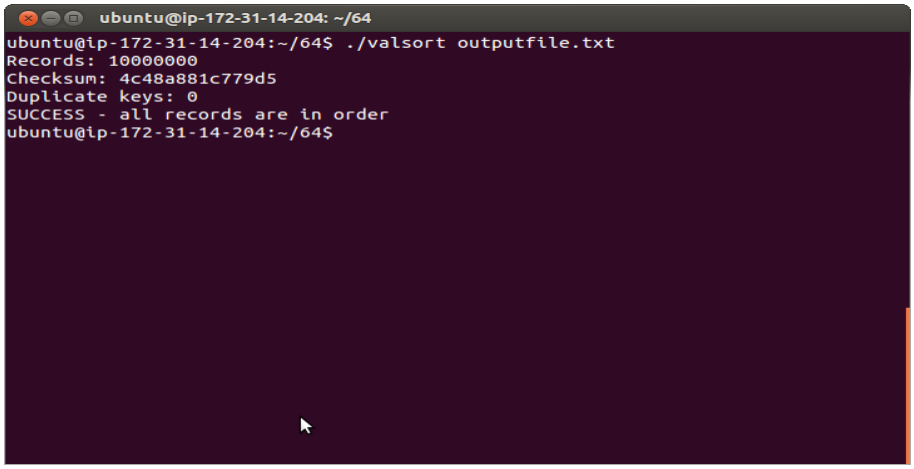
Number of Threads	Time (Sec)	Time (Minute)	Throughput (MB/Sec)
1	411	6 Minutes 51 Sec	24.92
2	409	6 Minutes 49 Sec	25.20
4	415	6 Minutes 55 Sec	24.67
8	423	7 Minutes 3 Sec	24.20

D2.xlarge-1 GB

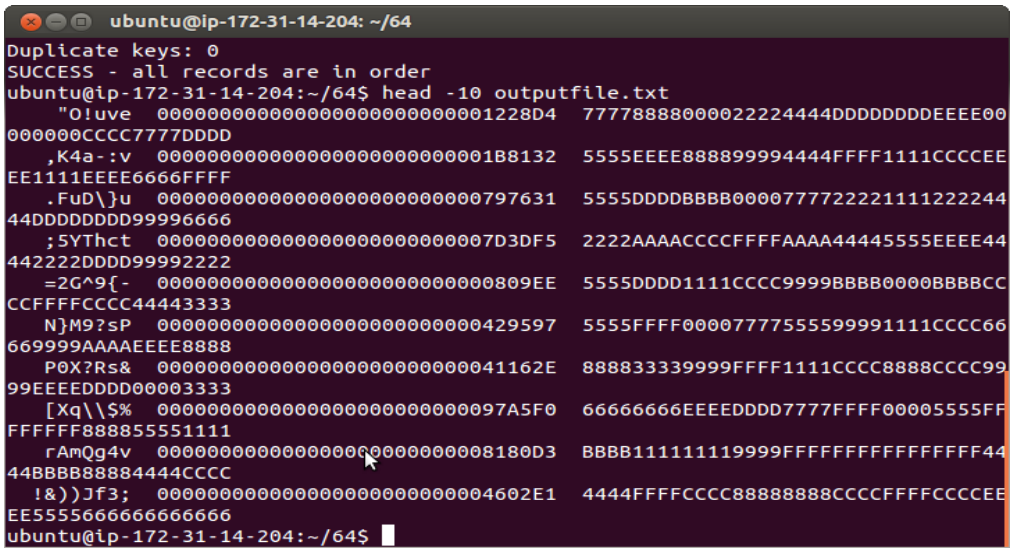
1 GB Time:



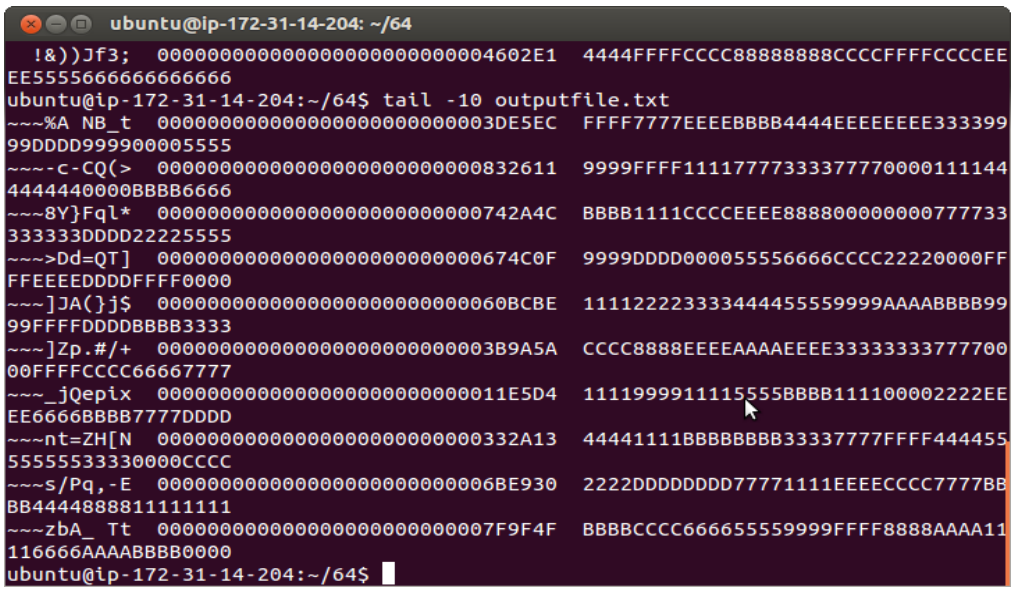
Valsort Output:



Screenshot of First 10 lines of sorted output:

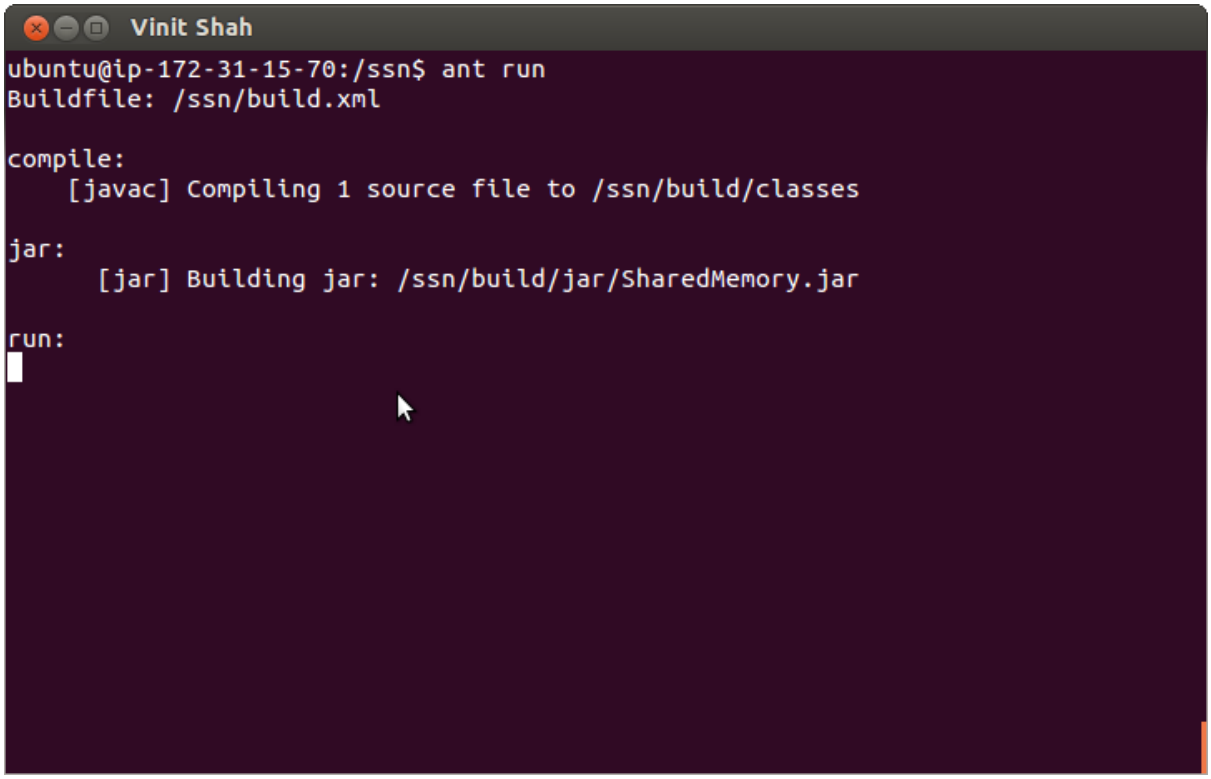


Screenshot of last 10 lines of sorted output:

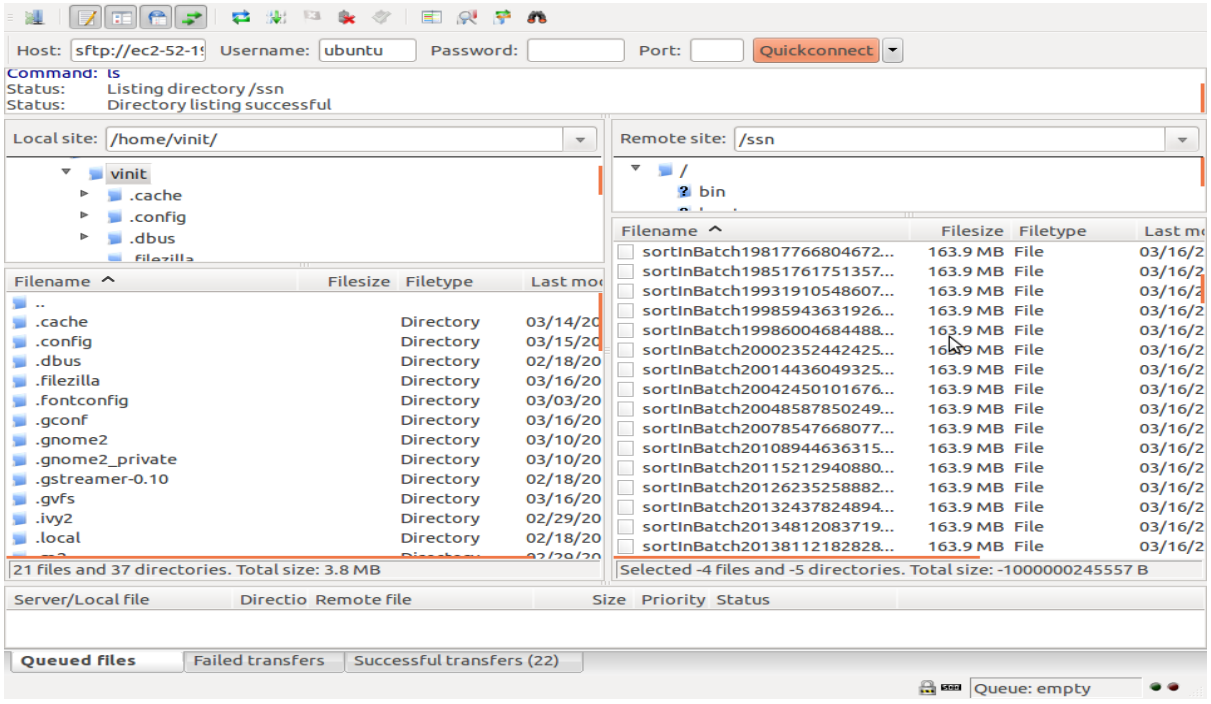


D2.xlarge 1 TB:

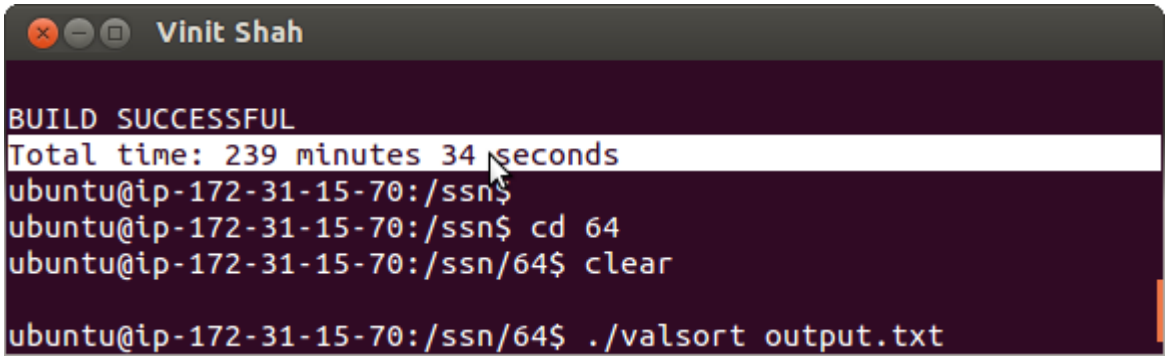
1 TB started:



1 TB temporary file generations:

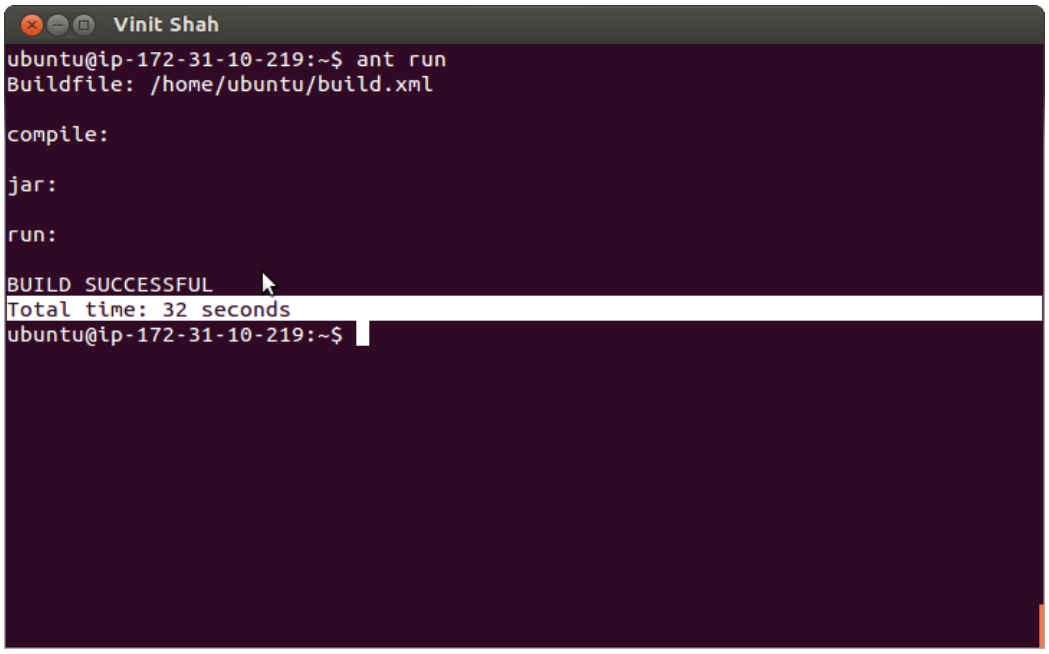


1 TB time to sort:

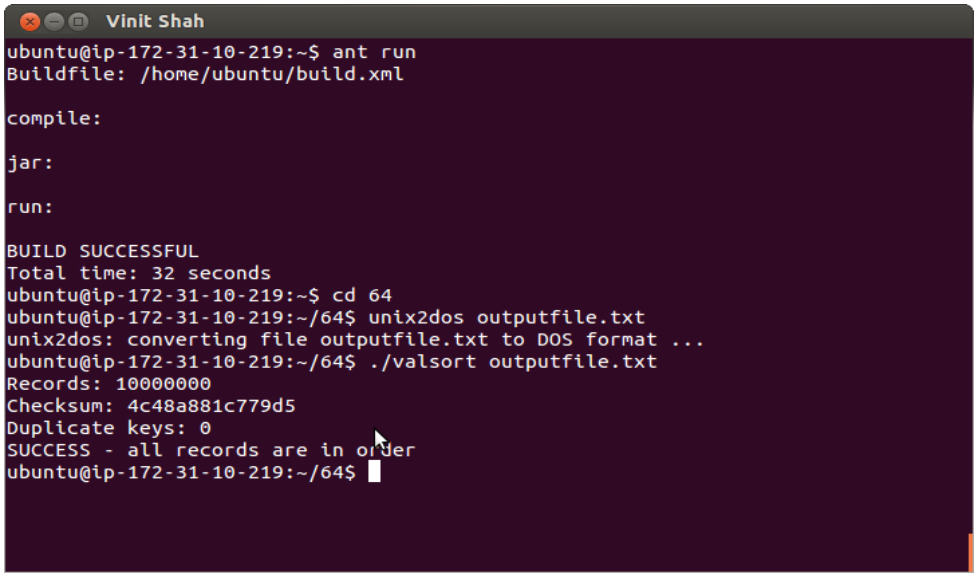


C3.large 1 GB

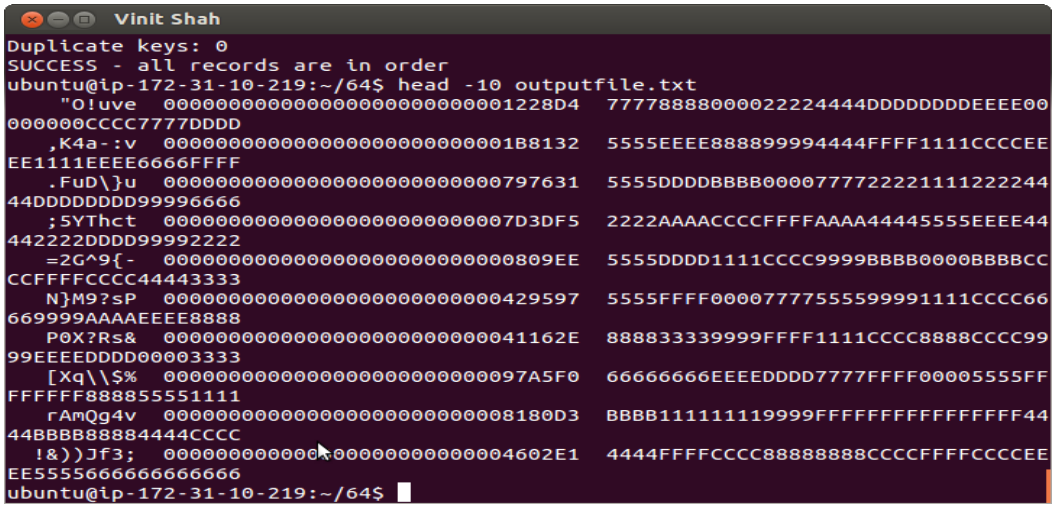
1 GB Time:



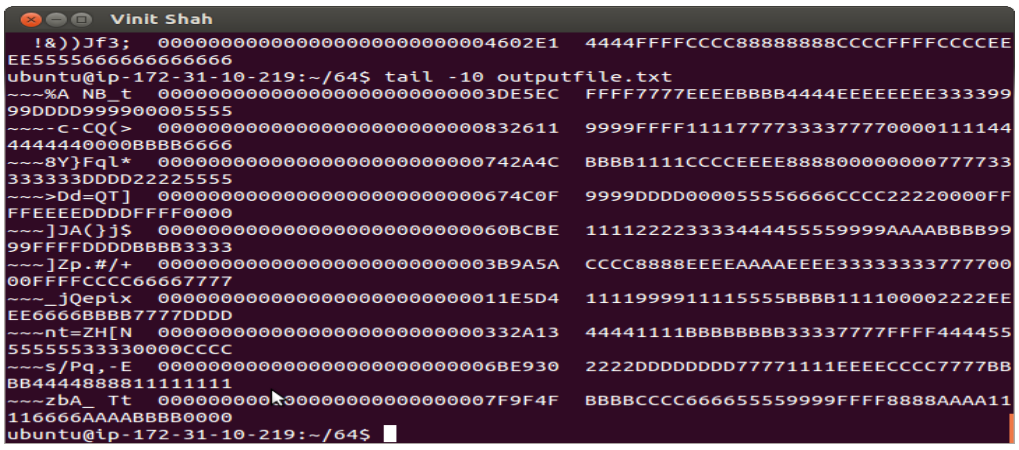
Valsort output:



First 10 lines of output:

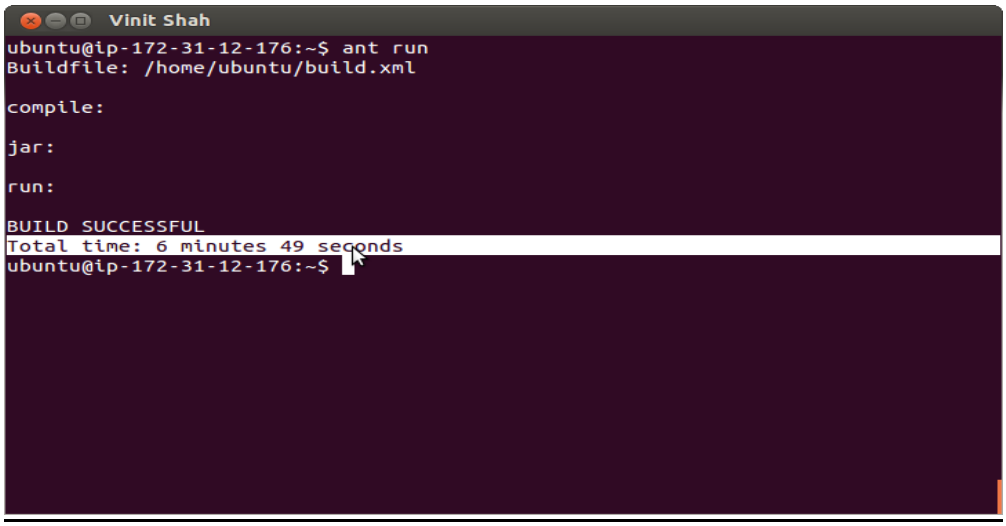


Last 10 Lines of output:



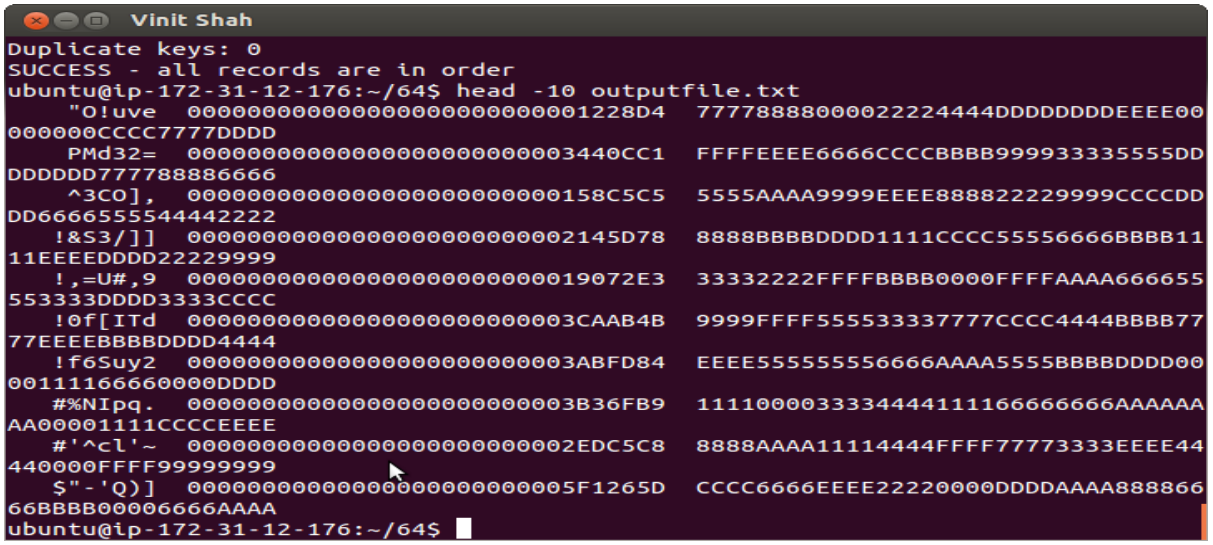
C3.large 10 GB

10 GB Sort Time:

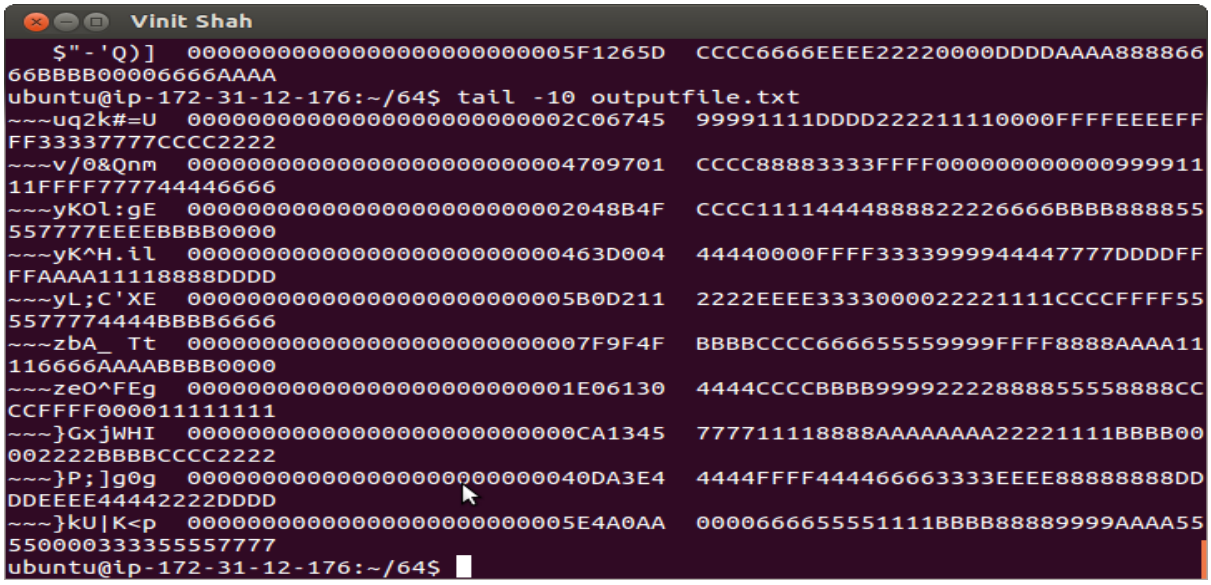


Valsort:

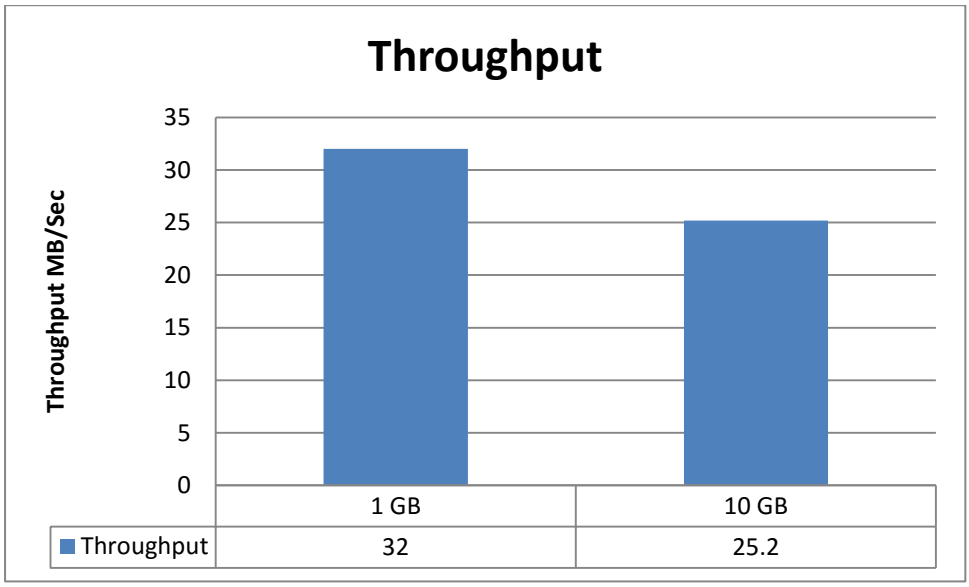
First 10 Lines of output:



Last 10 Lines of output:

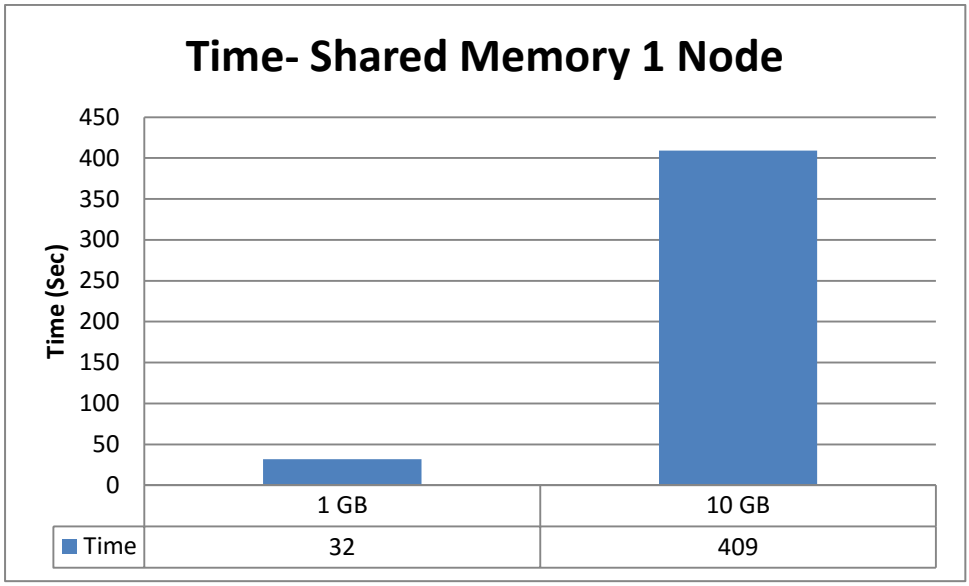


Throughput in MB/sec comparison while scaling data size from 1 GB to 10 GB on C3.large.



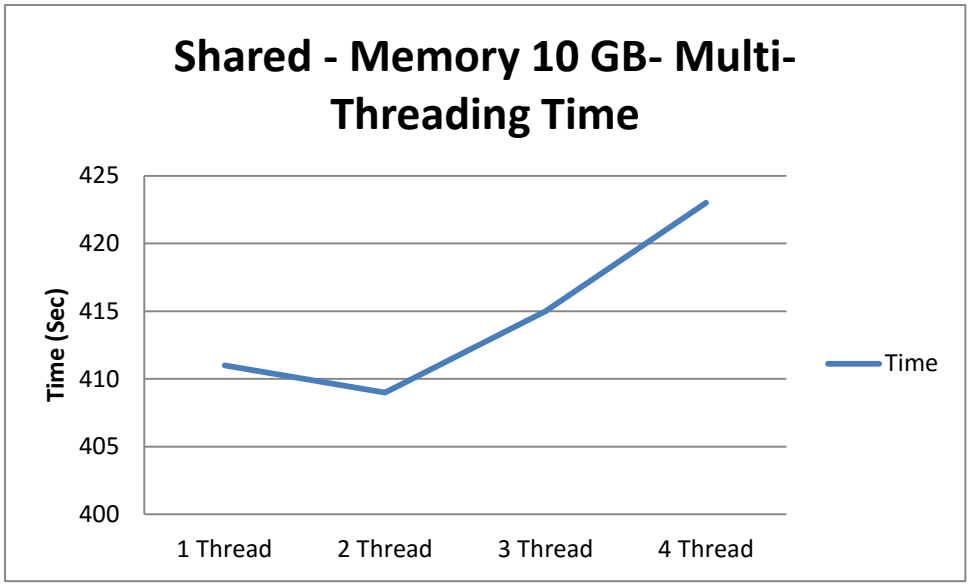
Through the observation of above graph, we can notice that the throughput to actually sort 1 GB and 10 GB using shared memory approach is almost at the constant level.

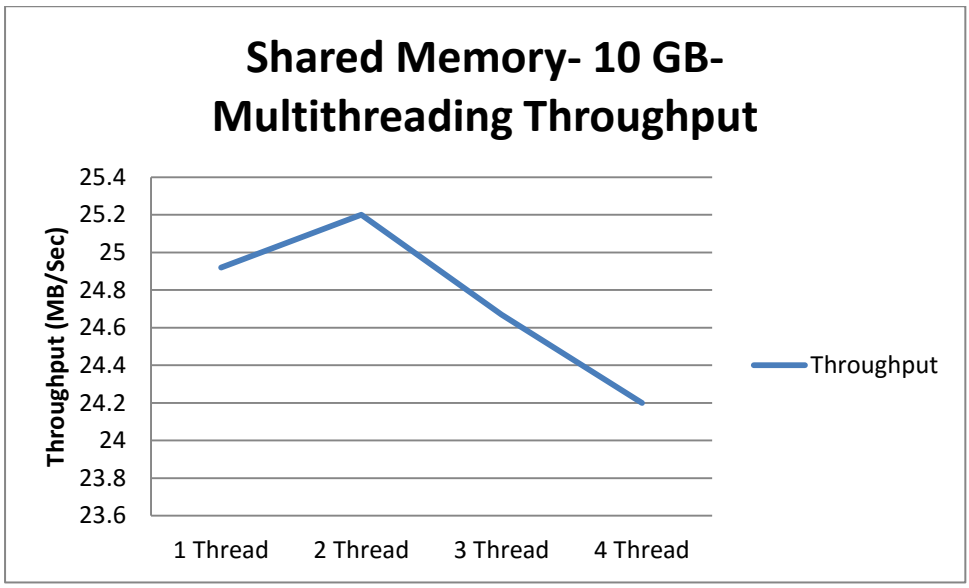
Time level comparison while scaling from small to large data set on C3.large.



From above graphs, we can conclude that the time required sorting 1 GB to 10 GB is increasing but the throughput of the system is nearly constant.

Shared Memory Multi-Threading 10 GB Time and Throughput Based Comparison:





Through the observation of graph, we can conclude that the Time required sorting a file of 10 GB with 2 threads is less than the time required to sort file with number of threads greater than 2, so the throughput achieved through 2 threads is high. It's because C3.large comes with 2 virtual cores, so that the performance of shared memory will be at its saturation after 2 threads, resulting in a decline of throughput and increase in time.

Apache Hadoop

Apache Hadoop is the distributed software processing framework for a large dataset. Hadoop map reduce essentially contains two main process which is Map and Reduce.

Map Phase:

This Phase of Hadoop takes the dataset as an input stored in the Hadoop File system (HDFS) and converts it into another dataset where, each data set is broken down into set to tuples represented as Key, Value Pair. Map phase main responsibility is to create the map of key- value pair.

Reduce Phase:

This phase takes an output from Map phase as an input and combines those set up tuples into a smaller dataset such that value with similar dataset will be in one set during reduce phase. The generated output is stored back to the HDFS.

The biggest advantage of Hadoop Map Reduce is that it's easy to scale the processing of data from one node to multiple nodes.

Hadoop cluster with many nodes works in master-slave fashion, where one master (Controller) who is responsible to distribute the jobs across Slaves (Workers).

What is Master?

Hadoop's master node (Name Node) is responsible to manage the operations of file system namespace like opening, closing, renaming files and determining the mapping of blocks to Data Nodes.

What is slave?

Hadoop's Slaves (Data Nodes) are responsible for serving read and write requests from the file system's clients along with perform block creation, deletion, and replication upon instruction from the Master.

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?

Hadoop master (Name node) gives services on different ports to each Hadoop daemon, so when a slaves which are nothing but data nodes tries to get the services of master, then data nodes will not be able to get the service as it tries to access the services on a port which is being shared by all the services of master and master is buys serving other slaves, so that its always advisable to configure the services on different ports so as to avoid the conflicts.

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

Number of mappers set to run is completely dependent on 1) File Size and 2) Block Size

We can change the split size of input file to decide on number of mappers, so mapred.map.tasks is just a hint to the InputFormat for the number of maps.

Configure Following Property in mapred-site.xml

<property>

<name>mapreduce.job.maps</name>

<value>600</value>

</property>

To change the number of Reducer configure the following property in mapred-site.xml

<property>

<name>mapreduce.job.reduces</name>

<value>40</value>

</property>

Setup a Hadoop Cluster:

Execute the following script to install Hadoop and its required dependencies also mount disk:


Install_Hadoop_Ec2.sh


RaidScript.sh

Following File changes are required, attached are the individual file, which is required to be changed. While setting-up the cluster, change the following files to as follows.


hdfs-site.xml


hadoop-env.sh


core-site.xml


yarn-site.xml


slaves


mapred-site.xml

To setup a Hadoop Cluster, follow steps mentioned in following file:


Hadoop Cluster Configuration Steps

Hadoop Readings:

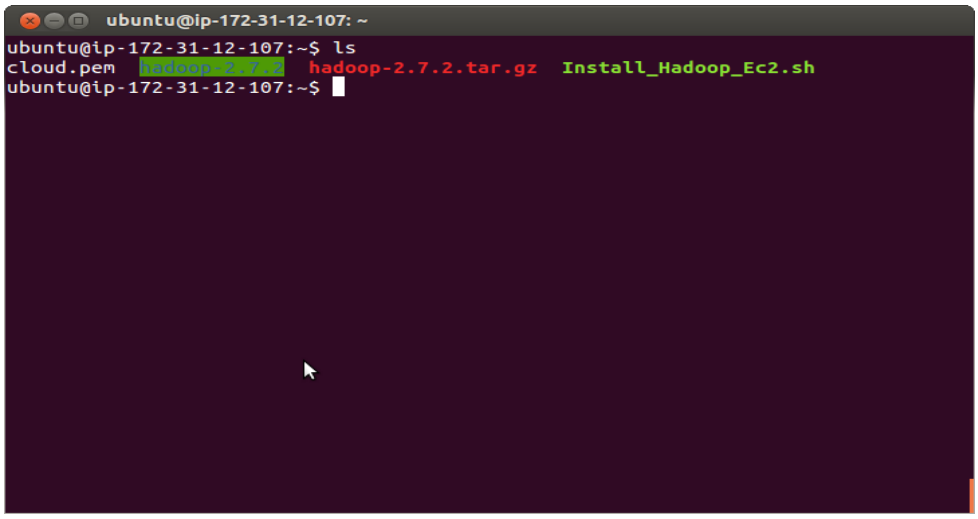
Dataset Size	Number Of Nodes	Time (Sec)	Throughput (MB/Sec)
1 GB (D2.xlarge)	1	86	11.90
1 GB(D2.xlarge)	16	39	26.25
10 GB (C3.large)	1	1444	7.10
100 GB (C3.large)	16 (40 Reducer)	1380	74.20

Observation:

Hadoop Sorting with D2.xlarge for 1 GB on 1 Node and 16 Nodes, we can conclude from throughput that as we scale the nodes from 1 to 16, the amount of parallelism in the application will increase result in the scale up in the performance while considering the throughput.

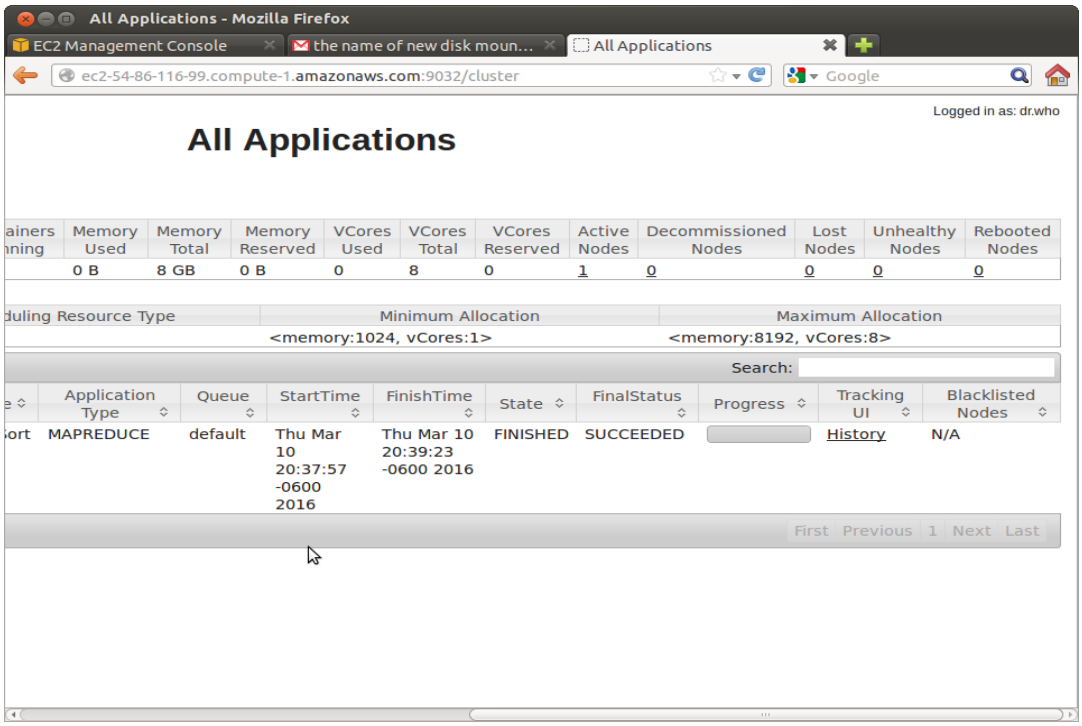
Hadoop sorting with C3.large on 1 Node for 10 GB and 16 Nodes for 100 GB, we can observe from readings that when we scale from 1 node to 16 nodes with linear increase in data set size, the time required to sort will nearly constant but throughput will increase considerable amount of times.

Hadoop after Installation:

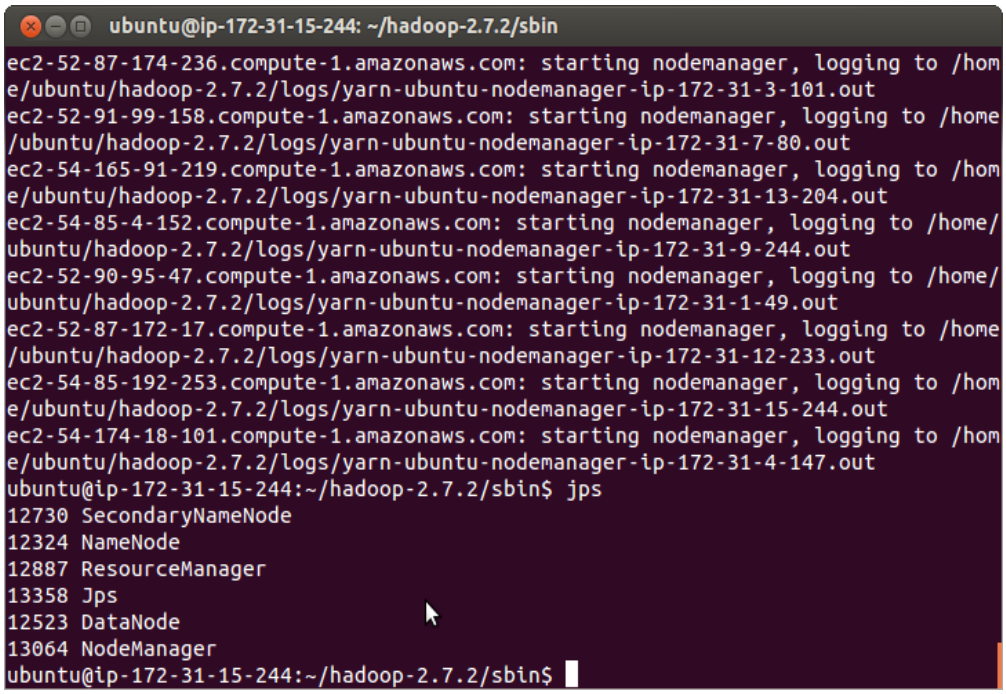


D2.Xlarge 1 GB:

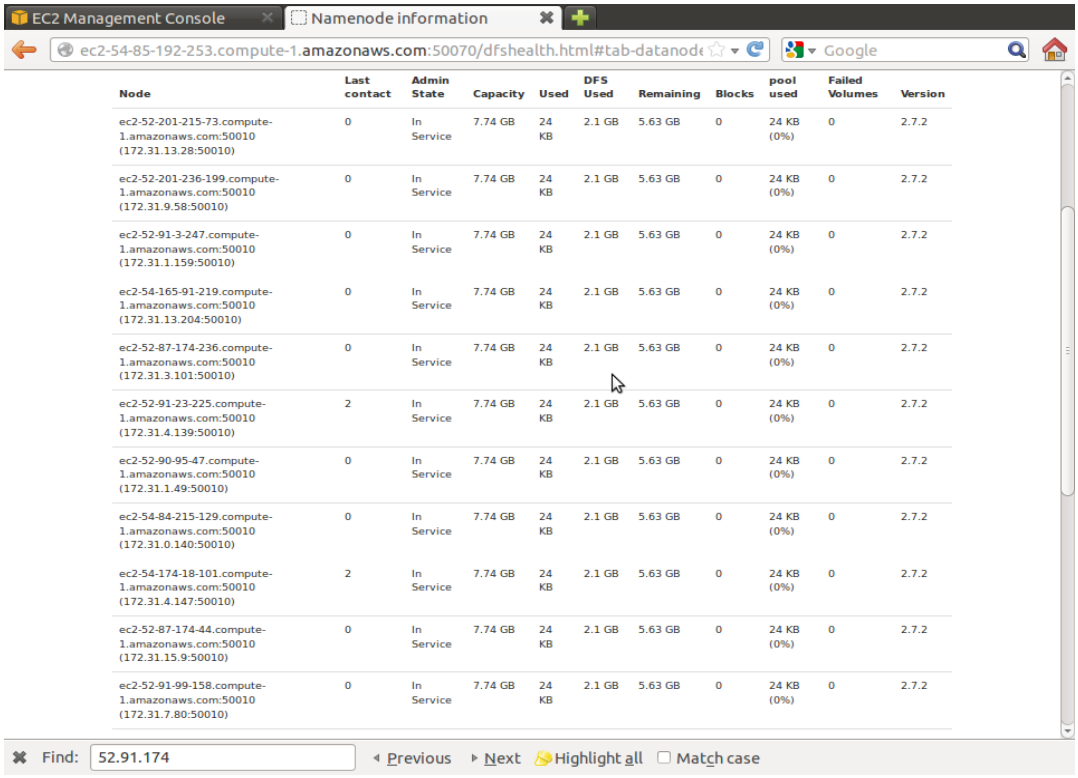
Hadoop 1 GB 1 Node Time required to sort:



Hadoop 16 Nodes 1 GB:

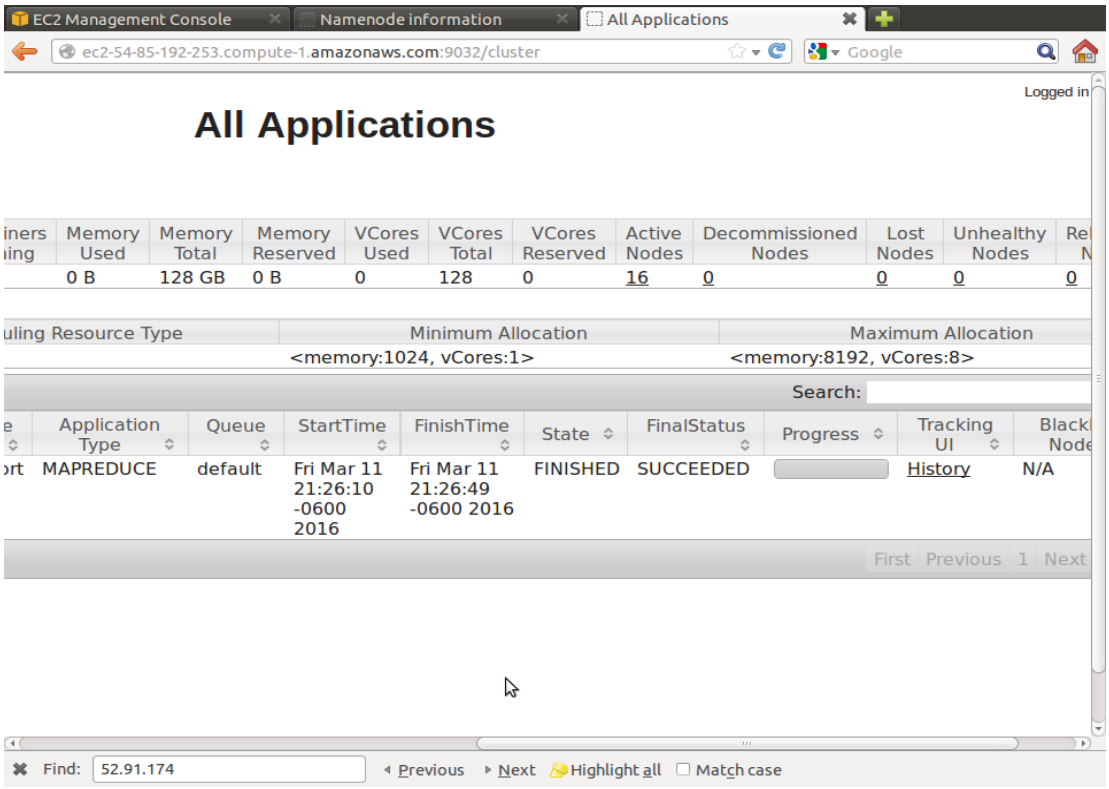


Hadoop Name Node Report of 16 Nodes cluster:

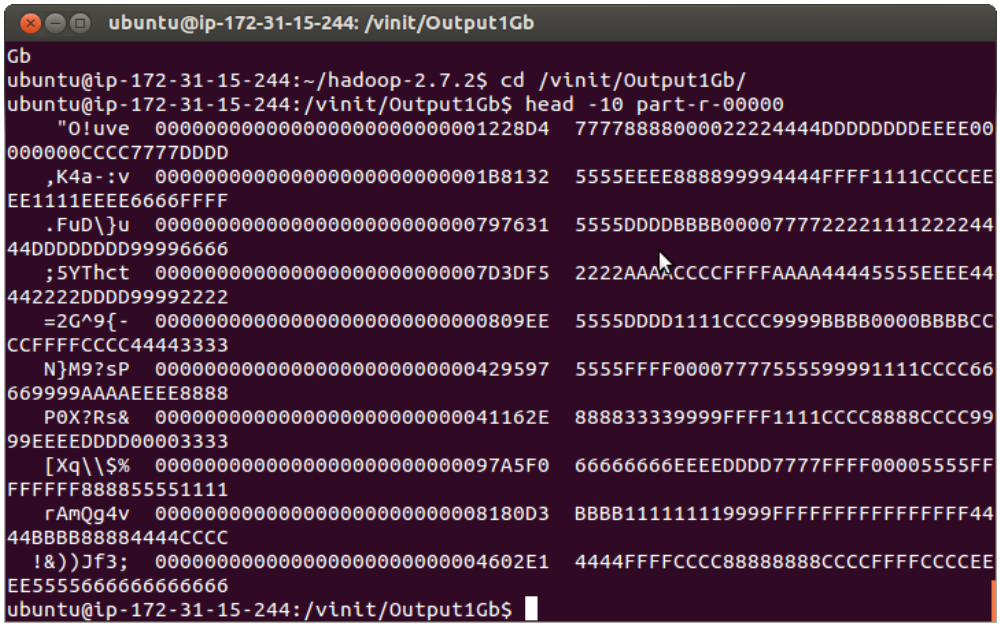


Node	Last contact	Admin State	Capacity	Used	DFS Used	Remaining	Blocks	pool used	Failed Volumes	Version
ec2-52-201-215-73.compute-1.amazonaws.com:50010 (172.31.13.28:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-201-236-199.compute-1.amazonaws.com:50010 (172.31.9.58:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-91-3-247.compute-1.amazonaws.com:50010 (172.31.1.159:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-54-165-91-219.compute-1.amazonaws.com:50010 (172.31.13.204:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-87-174-236.compute-1.amazonaws.com:50010 (172.31.3.101:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-91-23-225.compute-1.amazonaws.com:50010 (172.31.4.139:50010)	2	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-90-95-47.compute-1.amazonaws.com:50010 (172.31.1.49:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-54-84-215-129.compute-1.amazonaws.com:50010 (172.31.0.140:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-54-174-18-101.compute-1.amazonaws.com:50010 (172.31.4.147:50010)	2	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-87-174-44.compute-1.amazonaws.com:50010 (172.31.15.9:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2
ec2-52-91-99-158.compute-1.amazonaws.com:50010 (172.31.7.80:50010)	0	In Service	7.74 GB	24 KB	2.1 GB	5.63 GB	0	24 KB (0%)	0	2.7.2

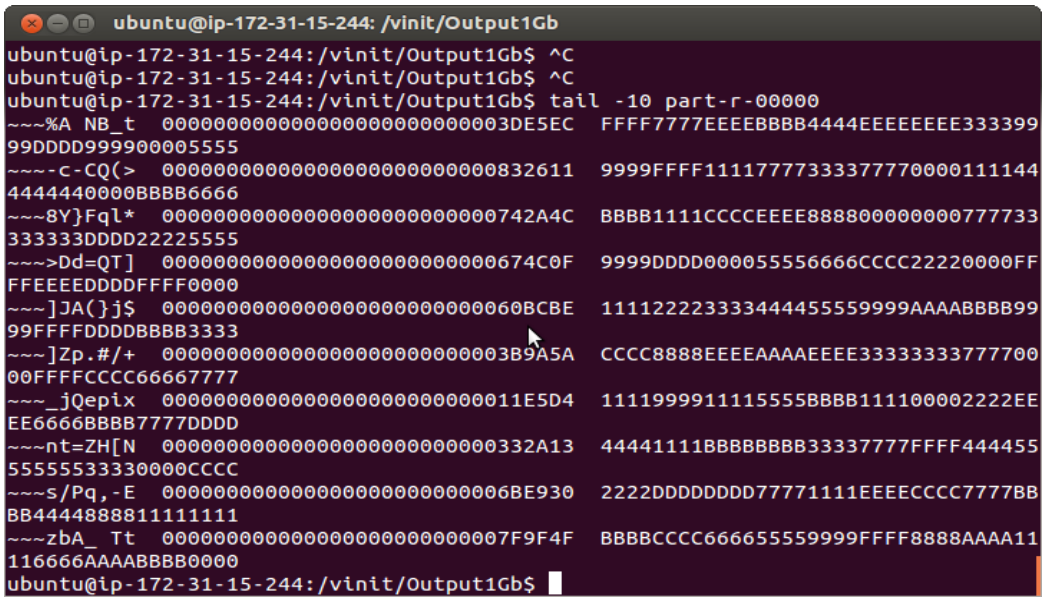
Hadoop 16 Nodes Time required sorting 1 GB:



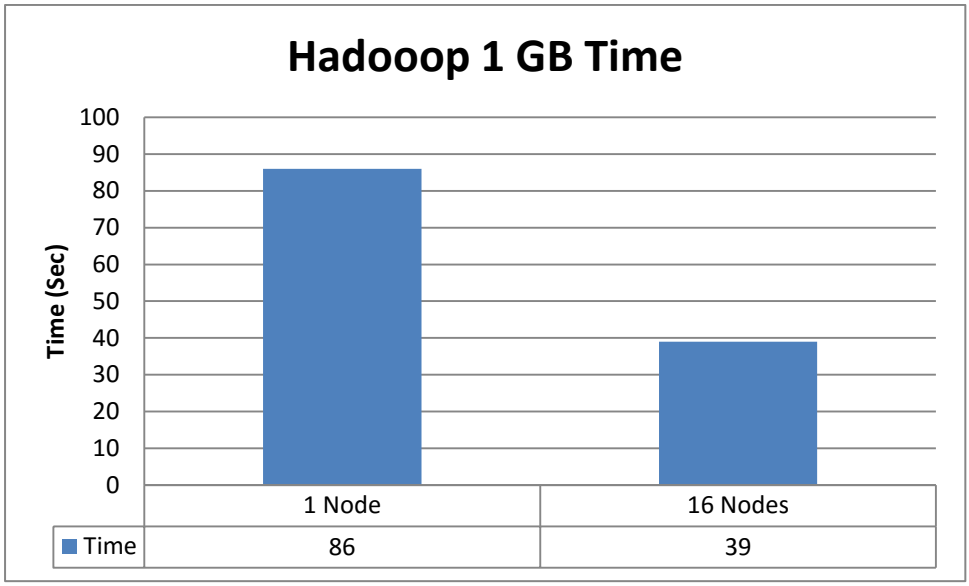
First 10 Lines of sorted Output:

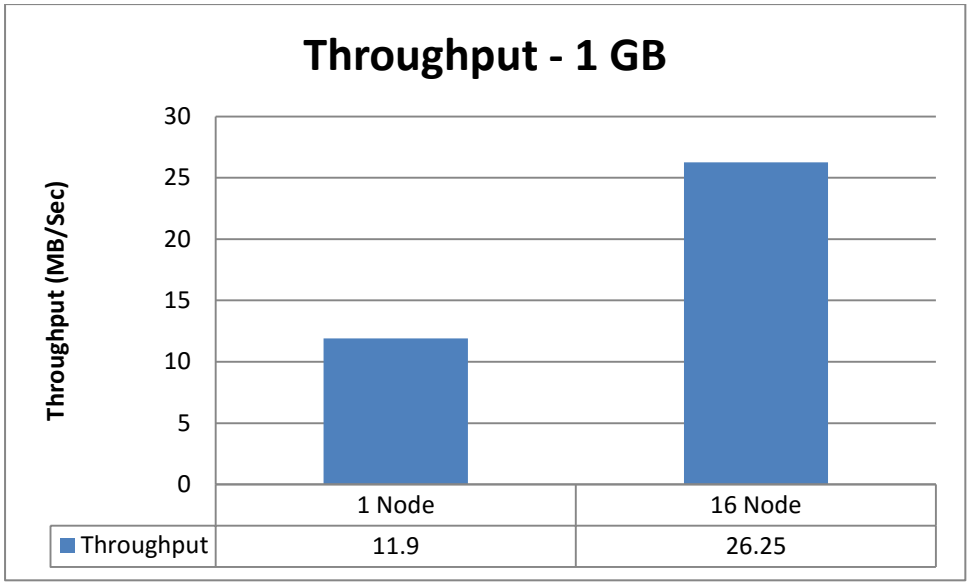


Last 10 Lines of sorted output:



Trade-off of Hadoop 1 GB sorting with 1 Node and 16 Nodes on D2.xlarge:

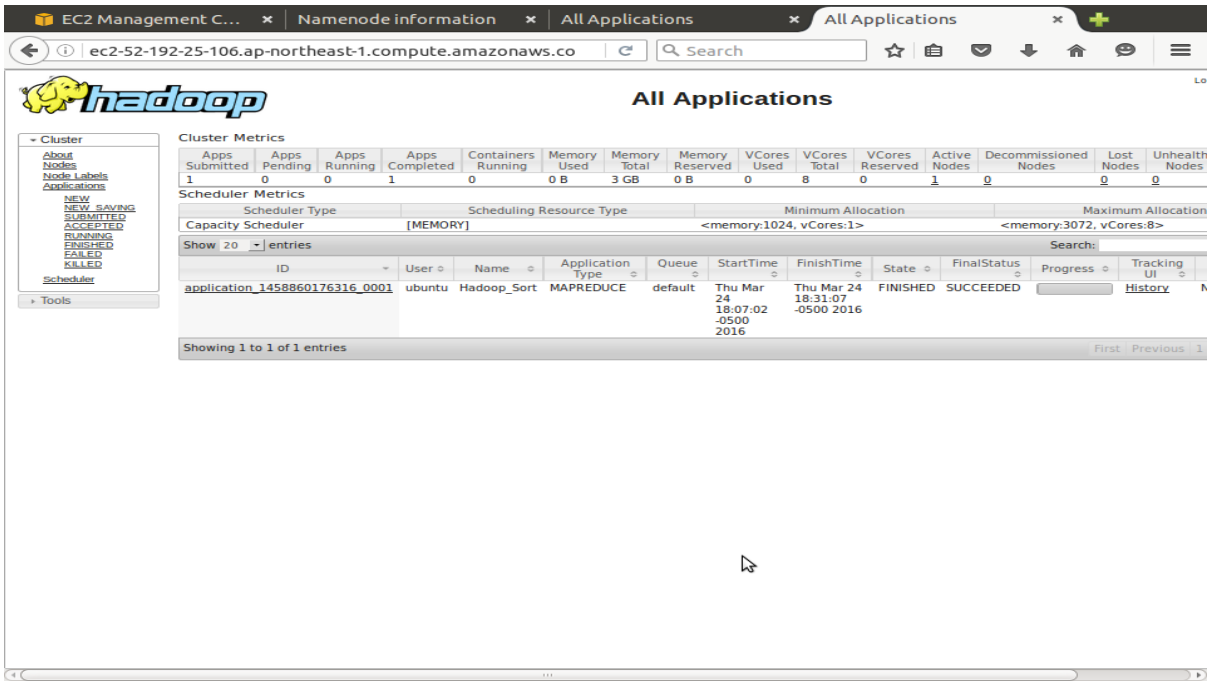




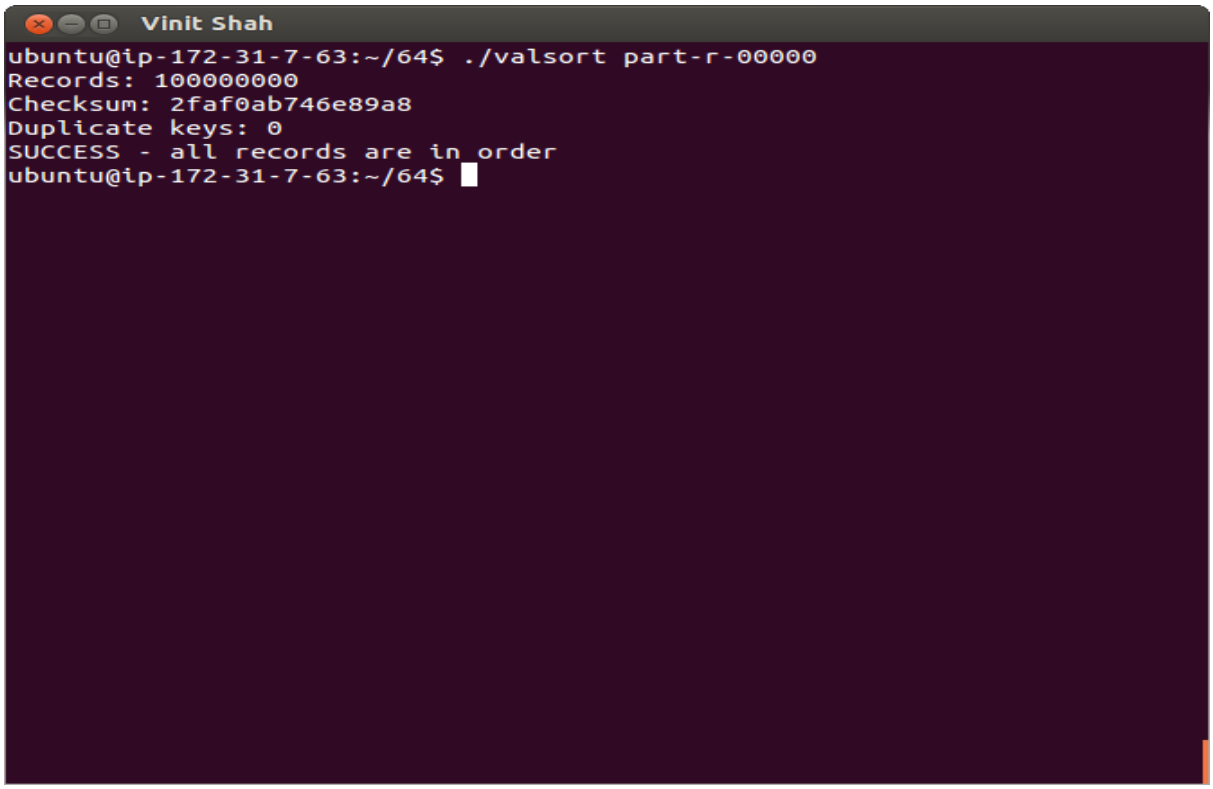
Hadoop Sorting with D2.xlarge for 1 GB on 1 Node and 16 Nodes, we can conclude from throughput that as we scale the nodes from 1 to 16, the amount of parallelism in the application will increase result in the scale up in the performance while considering the throughput.

C3.large 10 GB:

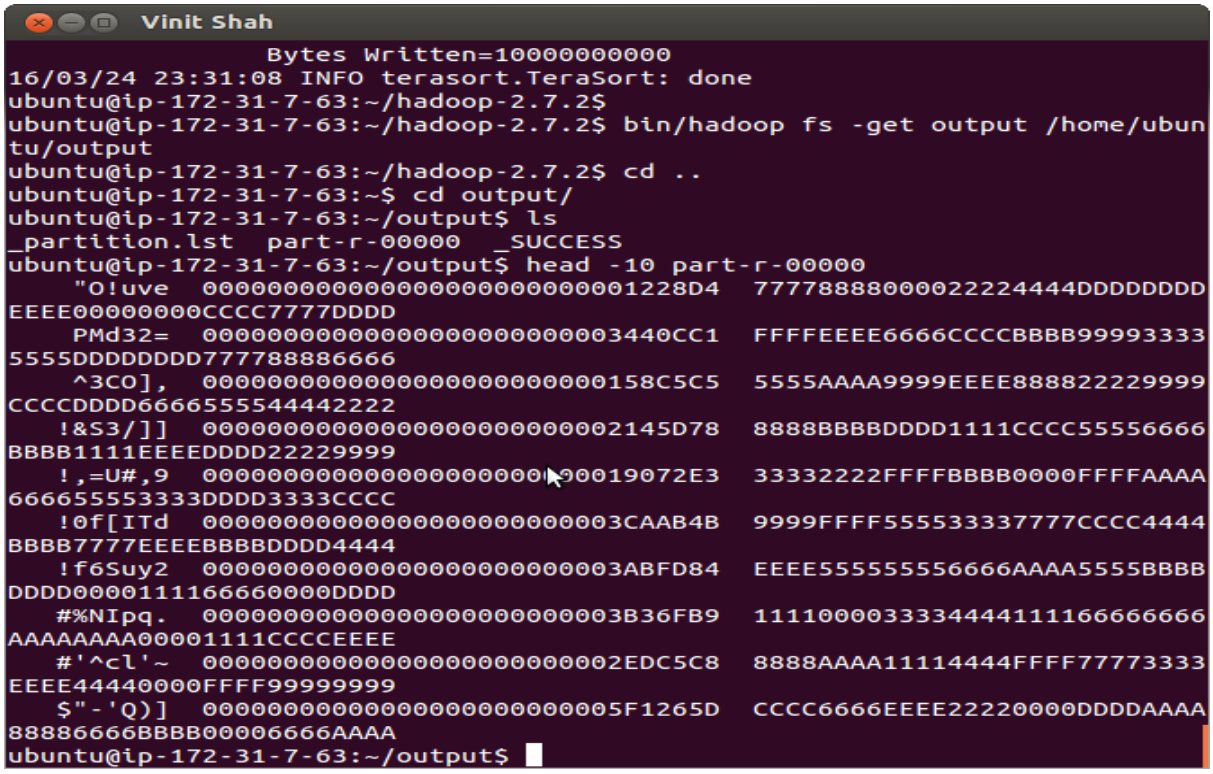
Time required to sort:



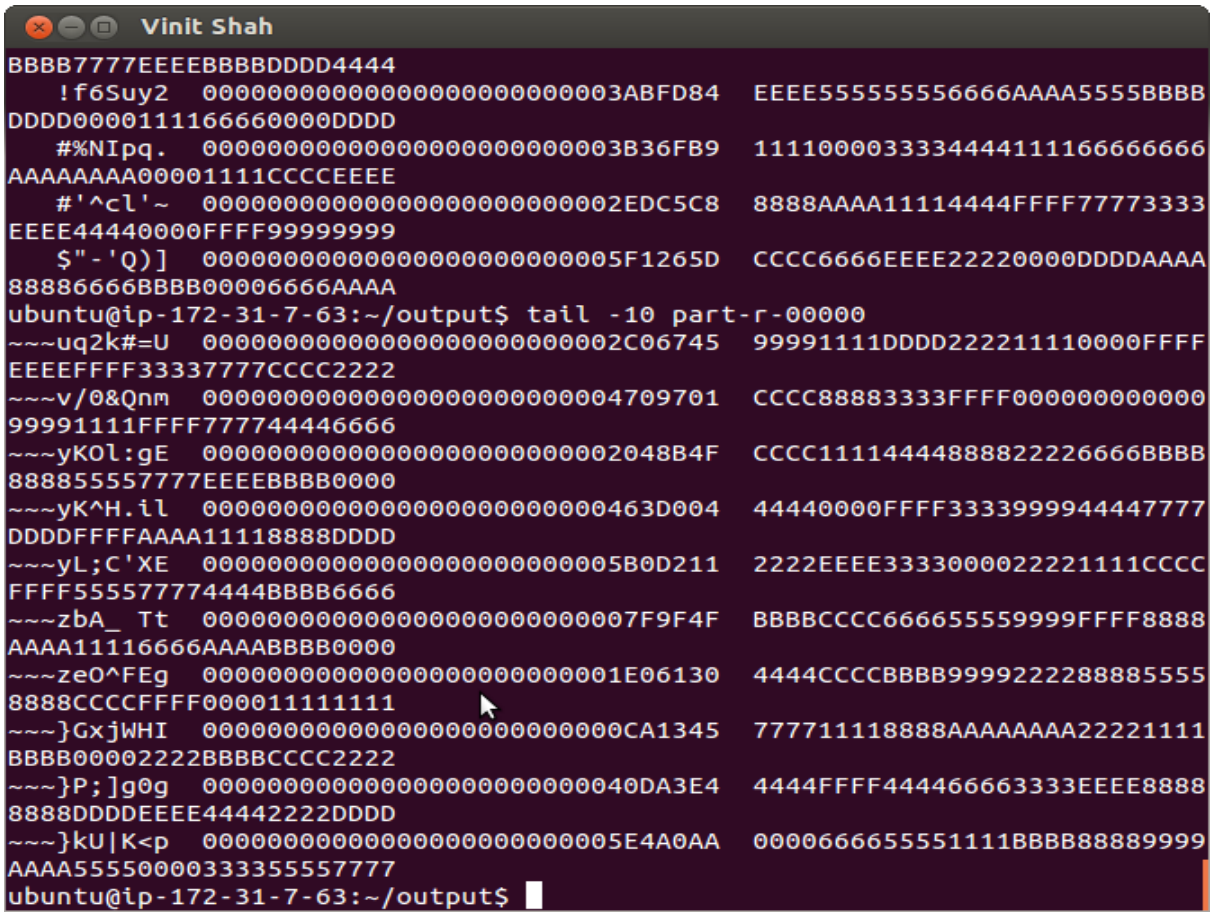
Valsort:



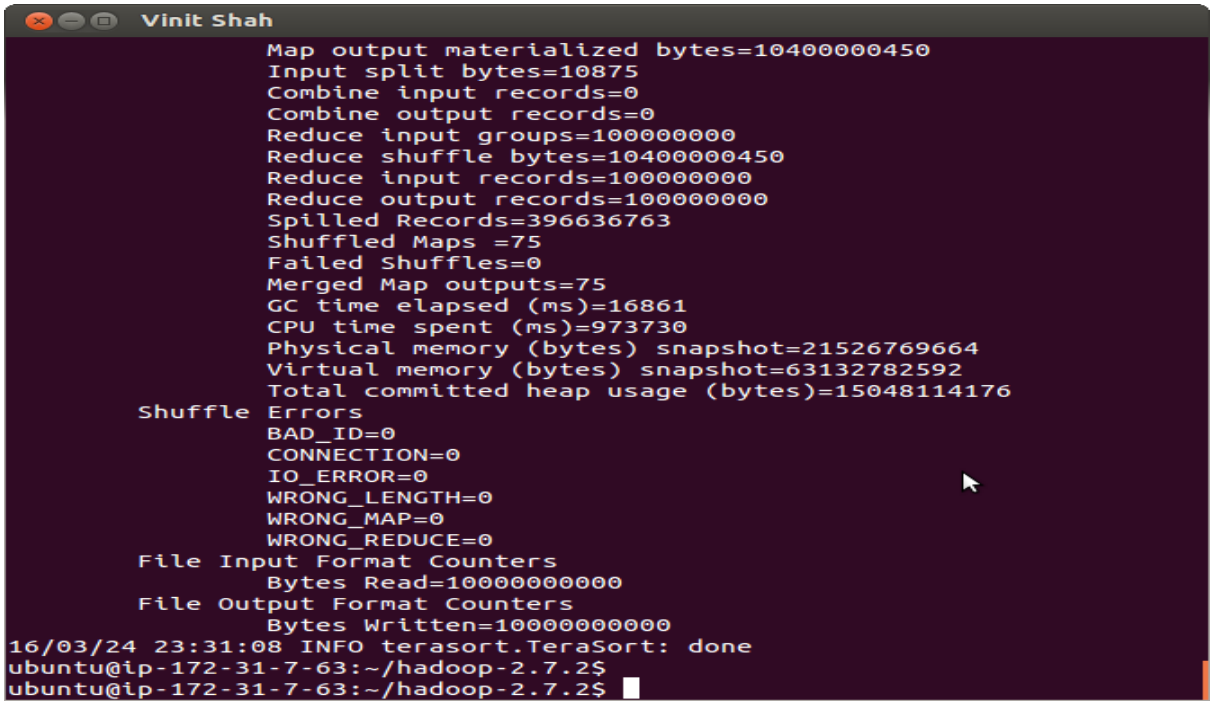
First 10 Lines of output:



Last 10 Lines of output:

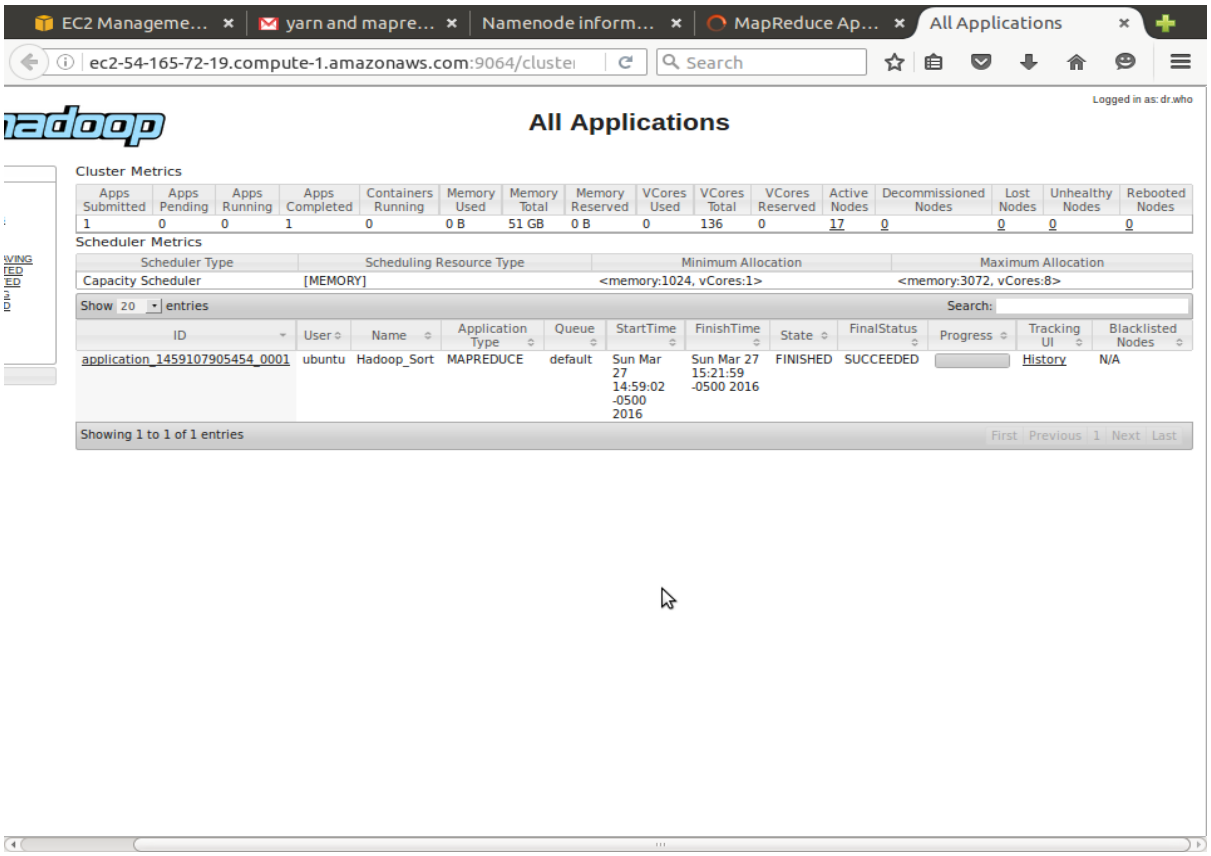


Map-Reduce Job success:



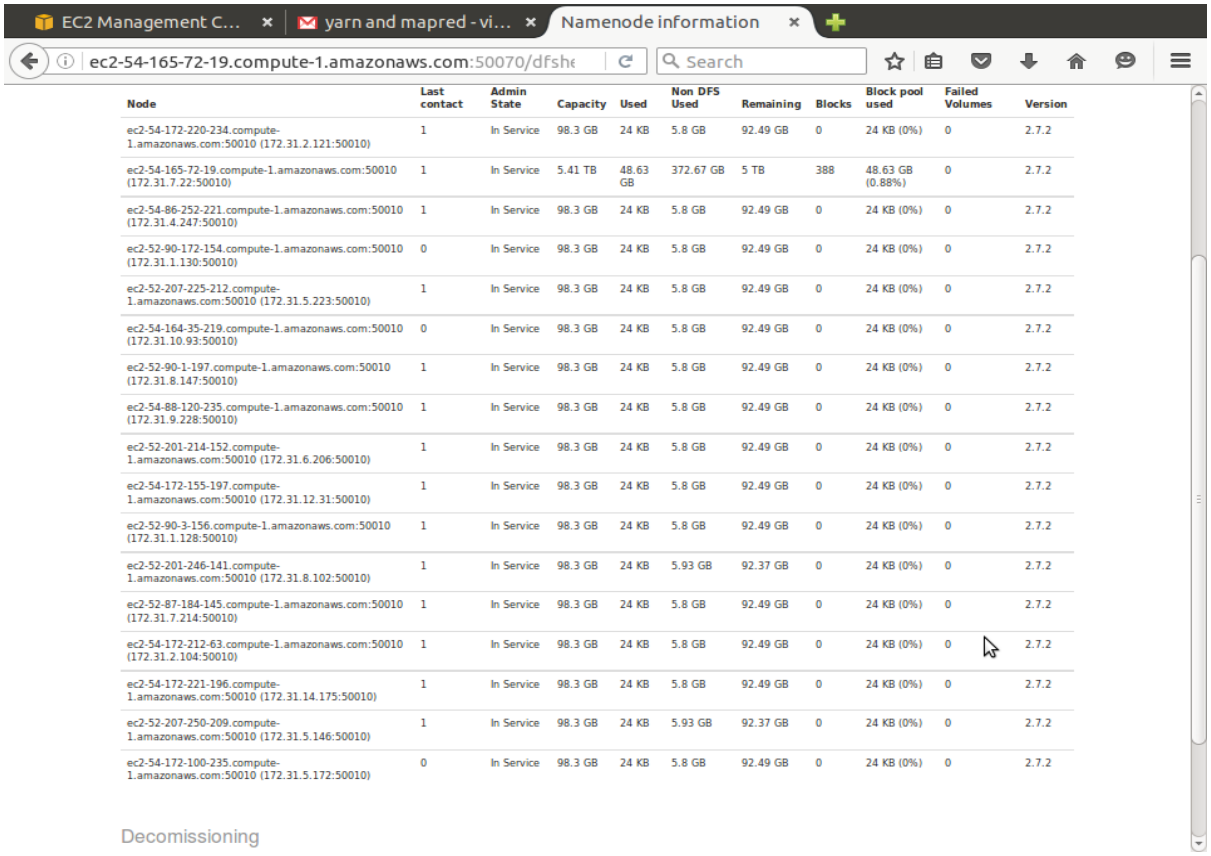
C3.large 100GB:

Time to required 100GB on 16 Nodes and 40 reducer:



:

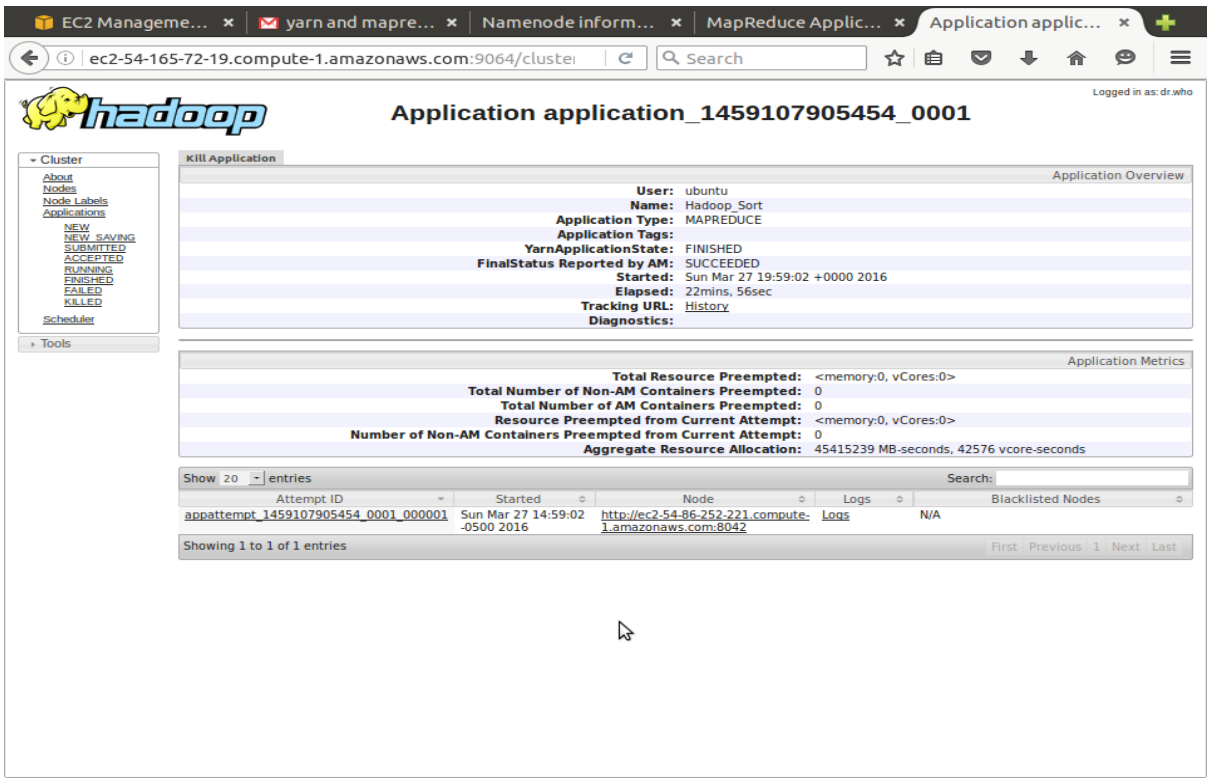
16 Nodes Cluster Setup:



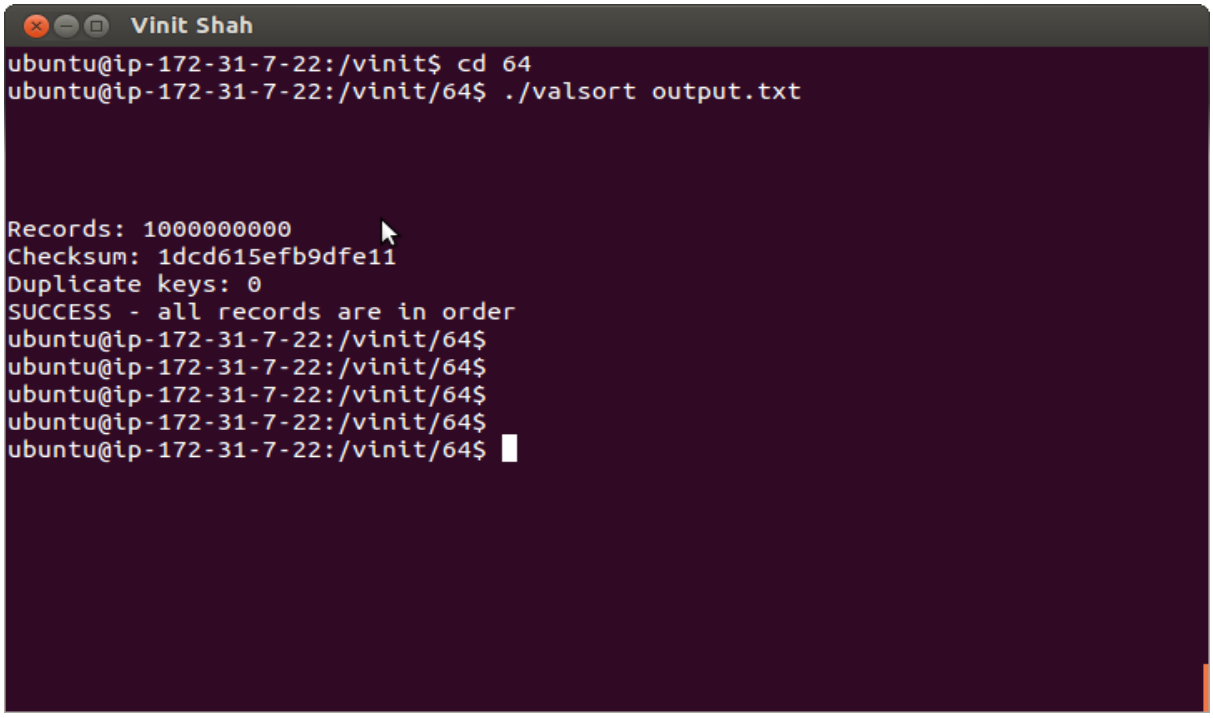
Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ec2-54-172-220-234.compute-1.amazonaws.com:50010 (172.31.2.121:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-165-72-19.compute-1.amazonaws.com:50010 (172.31.7.22:50010)	1	In Service	5.41 TB	48.63 GB	372.67 GB	5 TB	388	48.63 GB (0.88%)	0	2.7.2
ec2-54-86-252-221.compute-1.amazonaws.com:50010 (172.31.4.247:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-90-172-154.compute-1.amazonaws.com:50010 (172.31.1.130:50010)	0	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-207-225-212.compute-1.amazonaws.com:50010 (172.31.5.223:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-164-35-219.compute-1.amazonaws.com:50010 (172.31.10.93:50010)	0	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-90-1-197.compute-1.amazonaws.com:50010 (172.31.8.147:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-88-120-235.compute-1.amazonaws.com:50010 (172.31.9.228:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-201-214-152.compute-1.amazonaws.com:50010 (172.31.6.206:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-172-155-197.compute-1.amazonaws.com:50010 (172.31.12.31:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-90-3-156.compute-1.amazonaws.com:50010 (172.31.1.128:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-201-246-141.compute-1.amazonaws.com:50010 (172.31.8.102:50010)	1	In Service	98.3 GB	24 KB	5.93 GB	92.37 GB	0	24 KB (0%)	0	2.7.2
ec2-52-87-184-145.compute-1.amazonaws.com:50010 (172.31.7.214:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-172-212-63.compute-1.amazonaws.com:50010 (172.31.2.104:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-54-172-221-196.compute-1.amazonaws.com:50010 (172.31.14.175:50010)	1	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2
ec2-52-207-250-209.compute-1.amazonaws.com:50010 (172.31.5.146:50010)	1	In Service	98.3 GB	24 KB	5.93 GB	92.37 GB	0	24 KB (0%)	0	2.7.2
ec2-54-172-100-235.compute-1.amazonaws.com:50010 (172.31.5.172:50010)	0	In Service	98.3 GB	24 KB	5.8 GB	92.49 GB	0	24 KB (0%)	0	2.7.2

Decomissioning

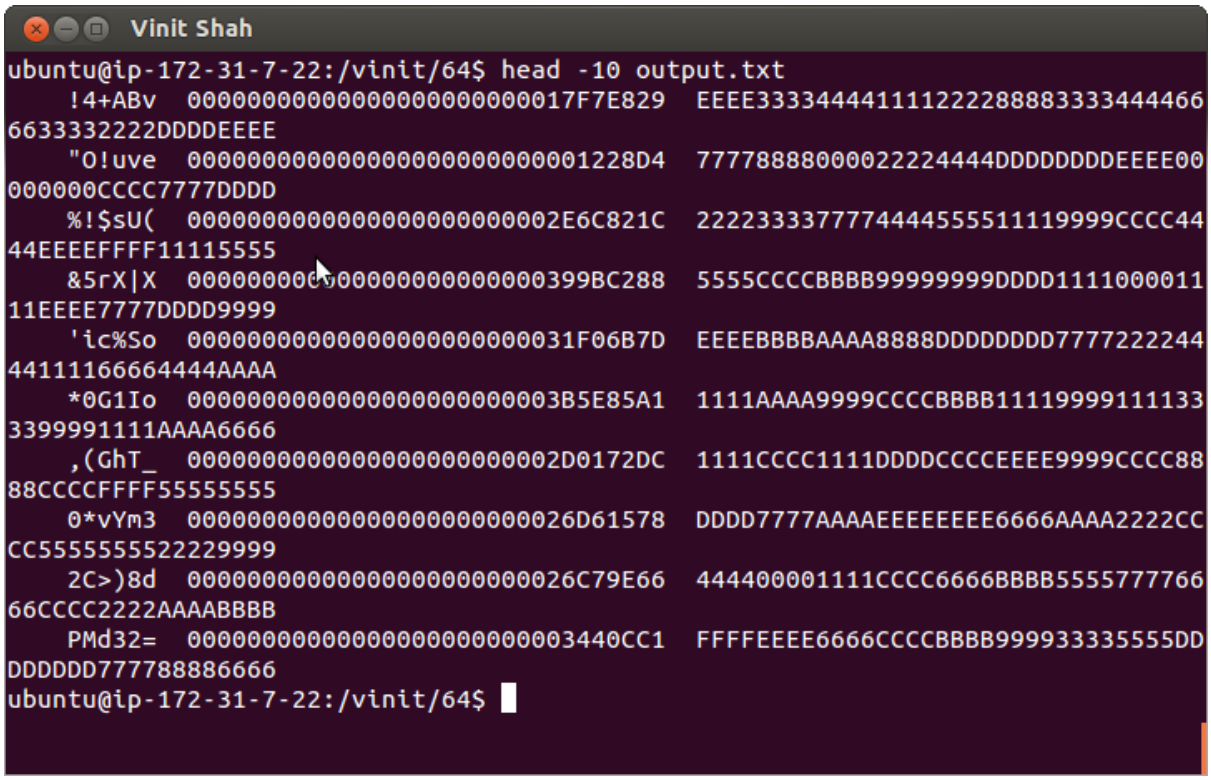
100 GB JOB sorted:



Valsort:



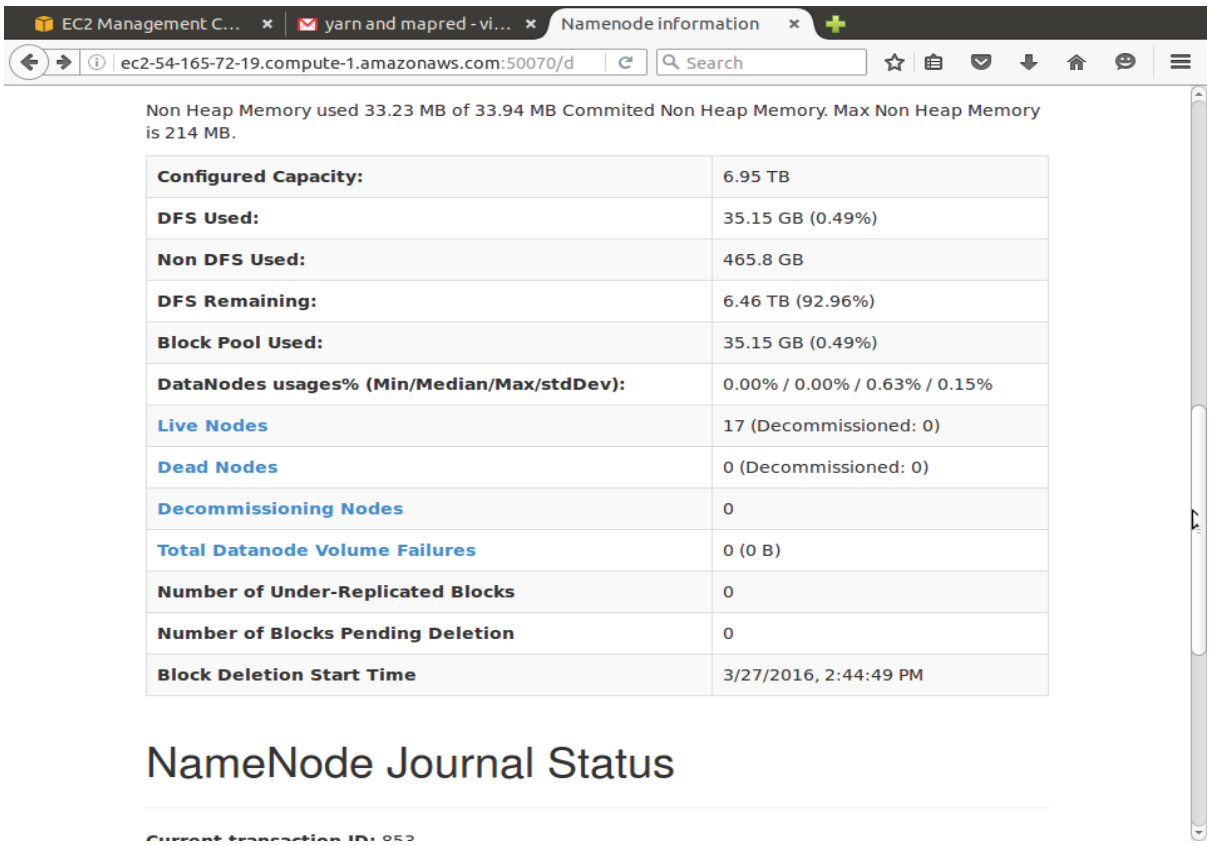
First 10 Lines of output:



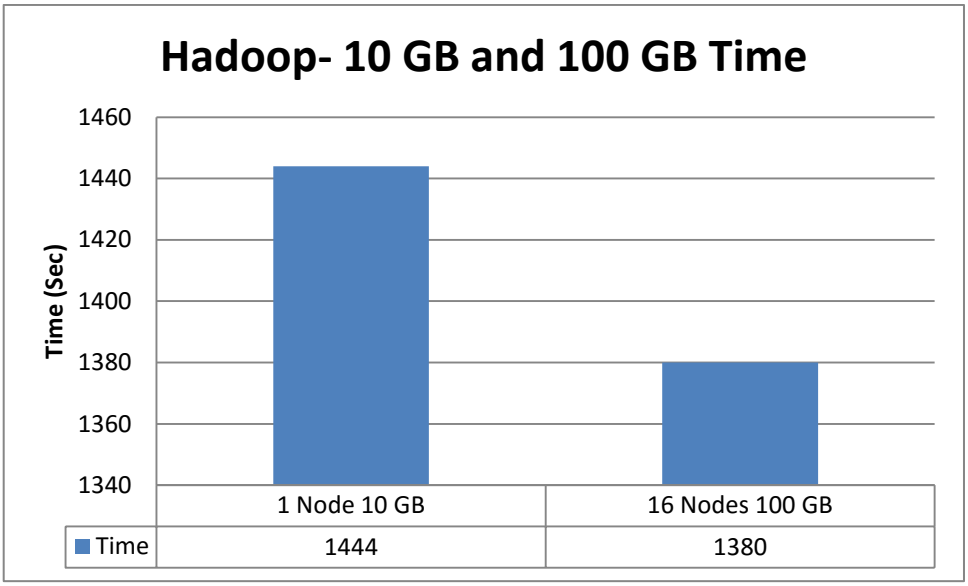
Last 10 Lines of Output:

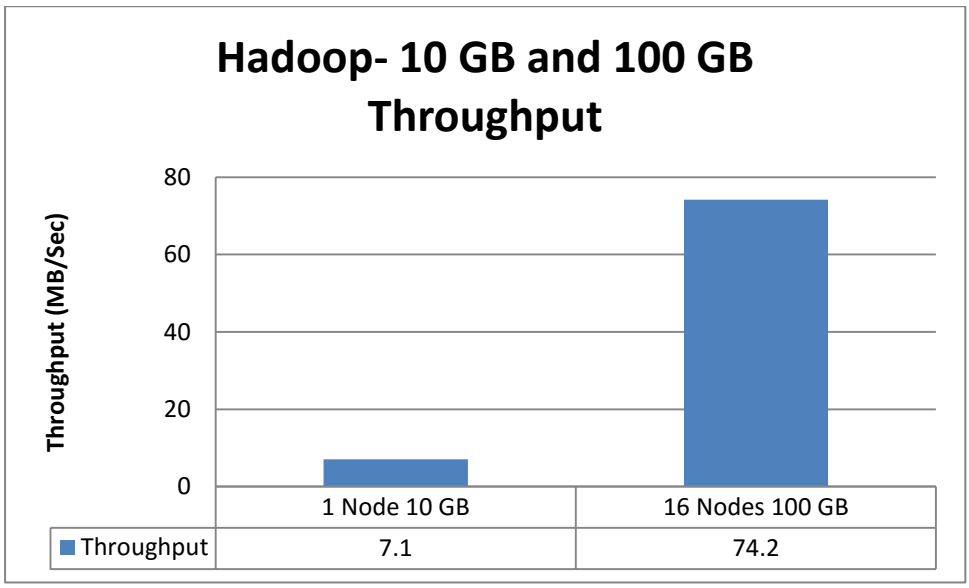
```
Vinit Shah
    PMd32=  00000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555DD
DDDDDD777788886666
ubuntu@ip-172-31-7-22:/vinit/64$ tail -10 output.txt
~~~~#iay1X  00000000000000000000000025D35EDF  6666AAAA555599997777000022223333888
88FFFF999922220000
~~~~+@p){@  00000000000000000000000085426F4  777733335555111111110000CCCC555599
99AAAA7777DDDDDDDD
~~~~,R^_?n  00000000000000000000000001034E347  111111119999000011118888AAAA555544
44EEEE999933338888
~~~~.Ey`^)  00000000000000000000000016F0E66B  CCCC6666DDDD2222DDDD111188889999EE
EEEEEEEEEBBBB4444
~~~~4!kA7x  00000000000000000000000001F1A1E26  EEEE777711117777BBBB1111EEEE888844
44DDDDDDDDEEEEBBBB
~~~~8Ii/!@  00000000000000000000000001F05932F  11119999BBBB444477770000111144444CC
CCAAAA6666DDDD0000
~~~~<I'5>F  00000000000000000000000008CB2293  88883333BBB11116666999998888555588
88888822228888CCCC
~~~~G-)m^)  00000000000000000000000013397F73  DDDDFFFFBBBBCCCCFFFF44446666AAAA11
1133333333AAAACCCC
~~~~c+I&cP  00000000000000000000000074BDF64  8888000055550000DDDD22227777AAAA00
0033332222AAAADDDD
~~~~hb&5X*  00000000000000000000000032C0E06B  7777BBBBBBBB9999EEEEAAAAAAAA0000CC
CCDDDD4444BBBB4444
ubuntu@ip-172-31-7-22:/vinit/64$
```

Name Node Health Page:



Time and Throughput based comparison of sorting on 1 Node 10 GB to 16 Node 100 GB:





Hadoop sorting with C3.large on 1 Node for 10 GB and 16 Nodes for 100 GB, we can observe from readings that when we scale from 1 node to 16 nodes with linear increase in data set size, the time required to sort will nearly constant but throughput will increase considerable amount of times.

Apache Spark

Apache spark which is built on Hadoop is the lighting fast cluster computing designed for fast and efficient computing. It uses and extends Hadoop's MapReduce to introduce more types of which includes Interactive Queries and Stream Processing. It's an in-memory data processing engine with elegant and expressive development APIs in Scala, Java, Python, and R that allow data workers to efficiently execute machine learning algorithms that require fast iterative access to datasets.

Resilient Distributed Dataset:

At the core of Spark is the notion of a **Resilient Distributed Dataset** (RDD), which is an immutable collection of objects that is partitioned and distributed across multiple physical nodes of a YARN cluster and that can be operated in parallel.

Spark uses Hadoop in two ways – one is **storage** and second is **processing**. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only.

To setup a Spark Cluster:

To configure the spark follow commands in following file and execute the script.

To execute the Spark for sorting, execute the following script.

Spark Readings:

Dataset Size	Number Of Nodes	Time (Sec)	Throughput (MB/Sec)
1 GB(D2.xlarge)	1	37.664472	27.19
1 GB(D2.xlarge)	16	18.251569545	55.35
10 GB (C3.large)	1	829.036	12.35
100 GB (C3.large)	16	1209.86	84.64

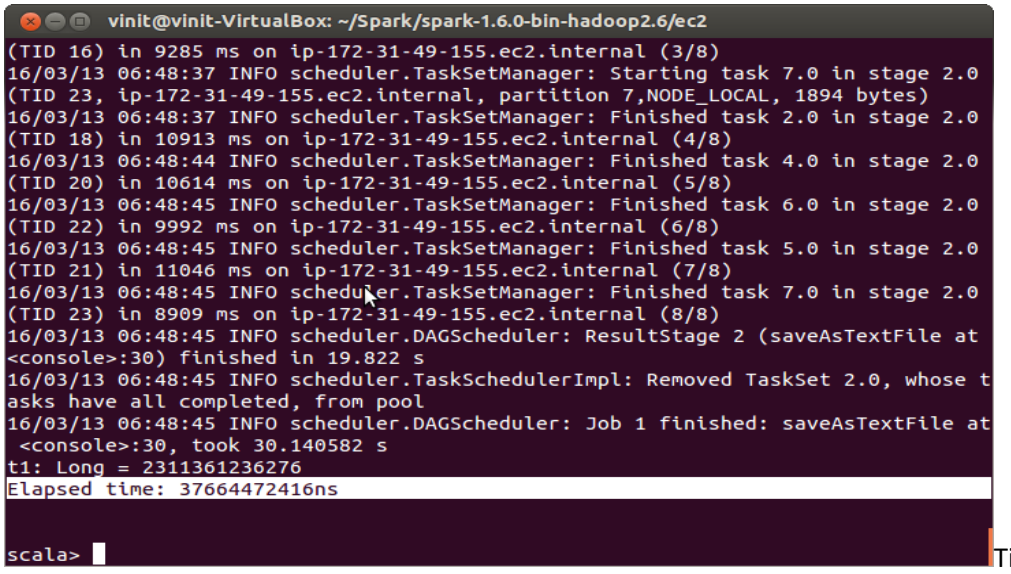
Observations:

Spark on D2.xlarge, Observing the reading of Time and Throughput, we can conclude that the spark's sorting of 1 GB with 1 Node compared with 16 Node will take half of the time, but the throughput is almost double.

Spark on C3.large, observing the reading of time and throughput, we can conclude that the time to sort 10 GB and 100 GB is increased to around 400 sec, but the throughput is almost 7 times scaled-up. This gives the faster performance in the case of large dataset.

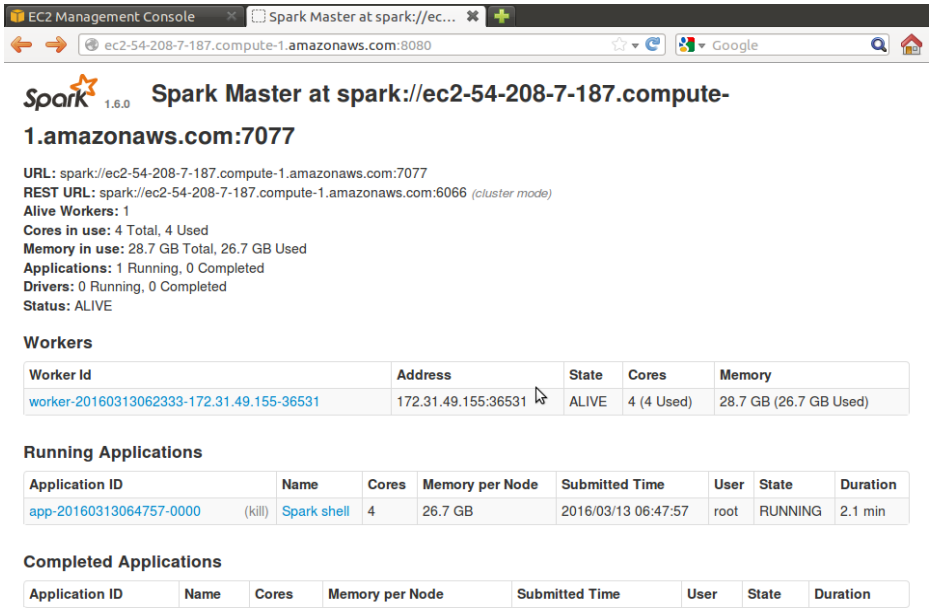
D2.xlarge 1 GB:

Spark Sort 1 GB 1 Node:

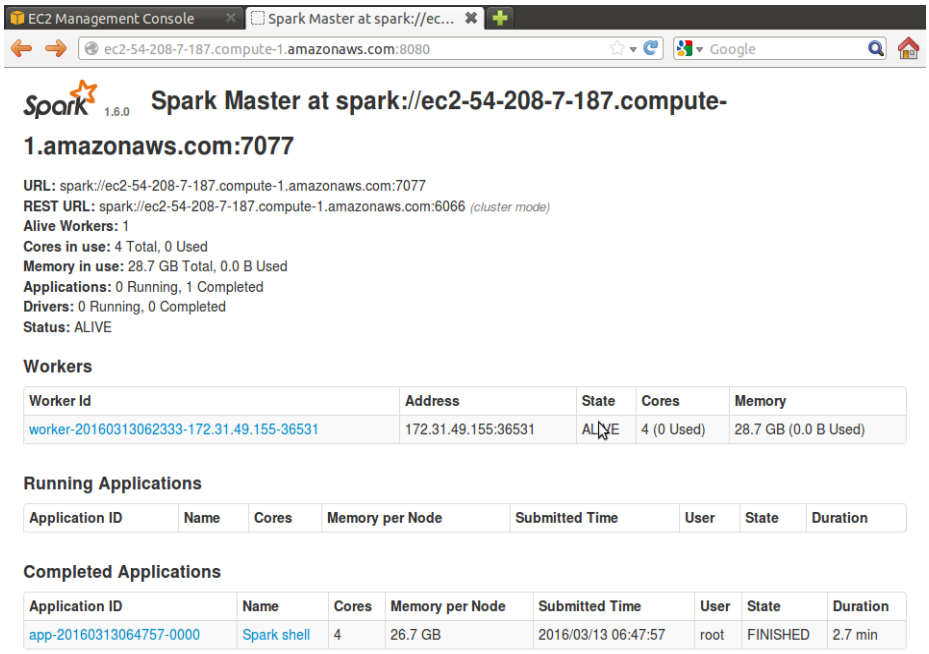


Time:

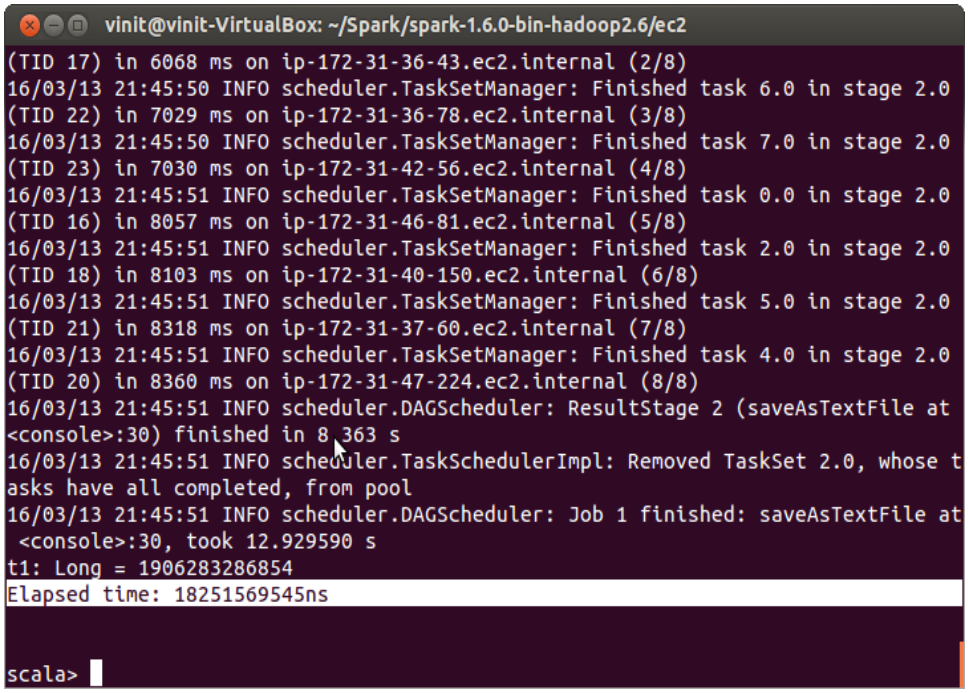
Spark 1 GB 1 Node Running:



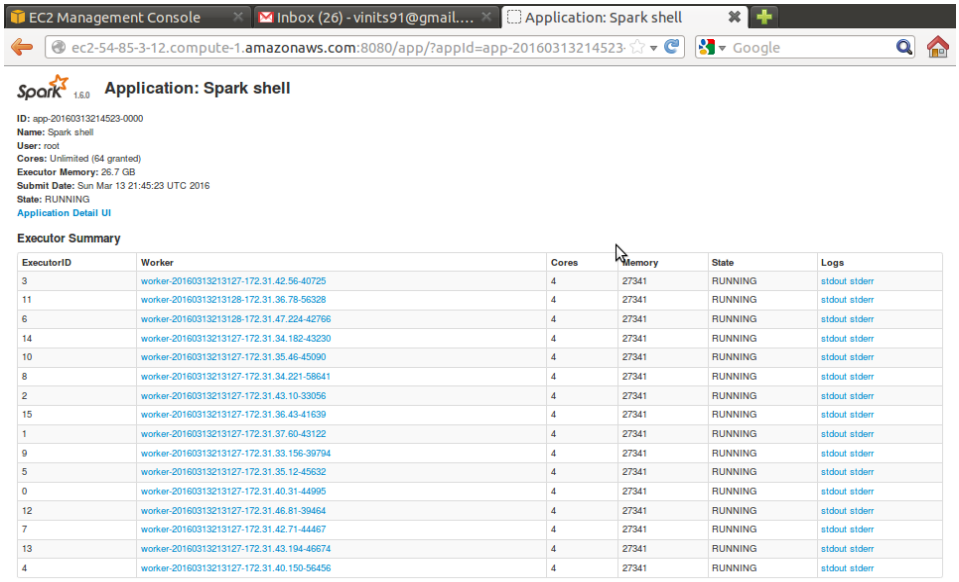
Spark 1 GB 1 Node sort Complete:



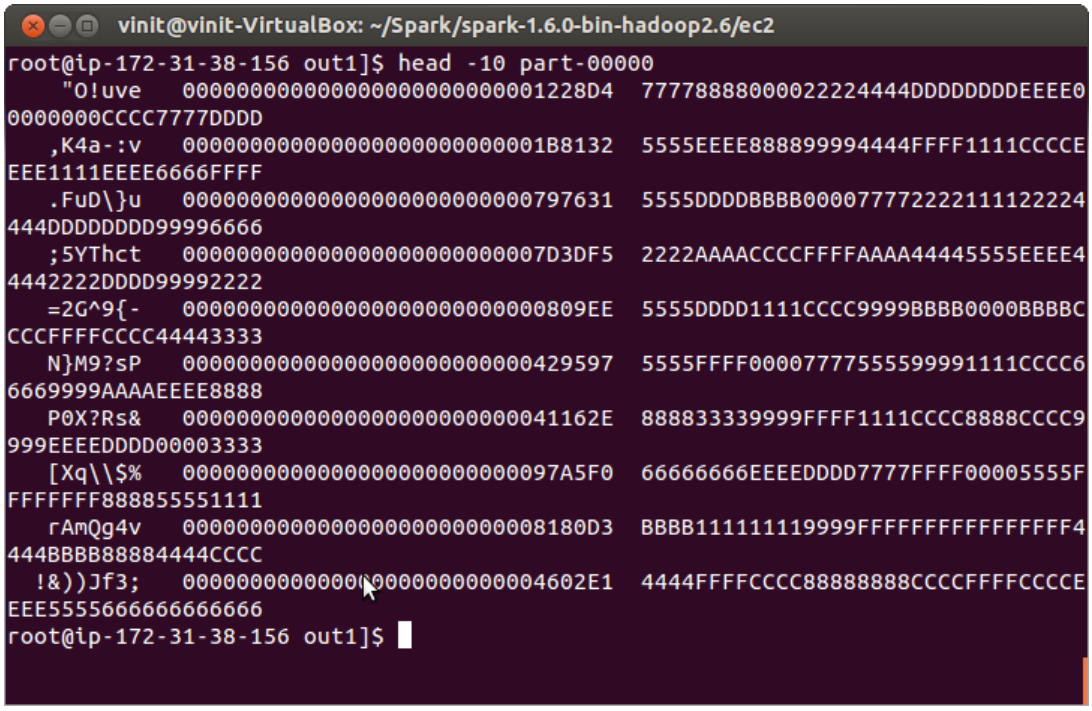
Spark 1 GB 16 Nodes Time for Sort:



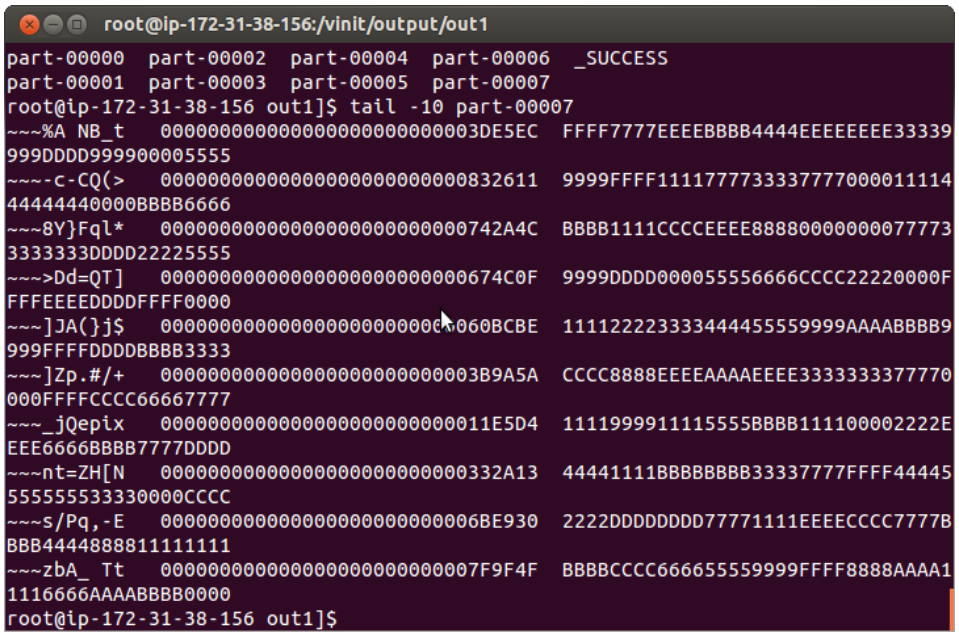
Spark 1 GB 16 Nodes Configured Name Node:



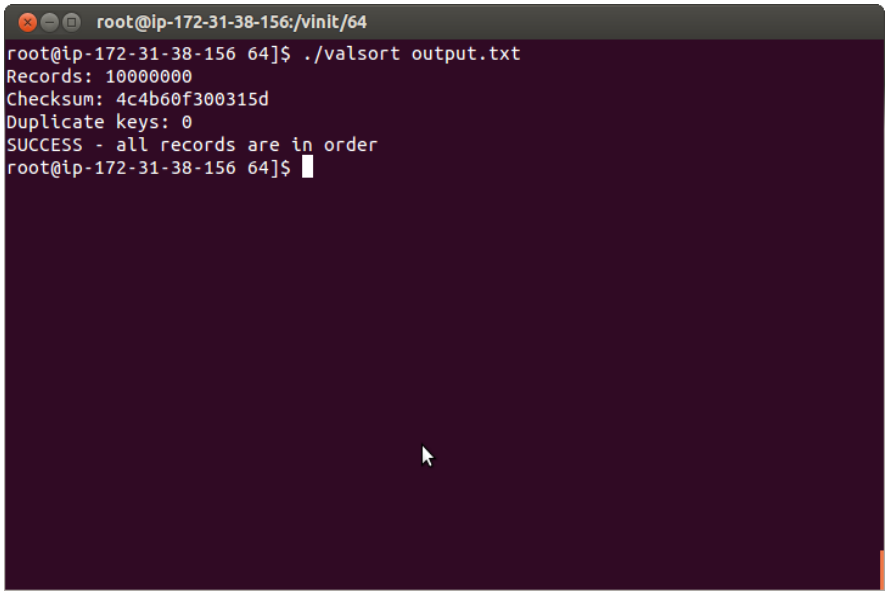
Spark 1 GB First 10 Lines of Sorted Output:



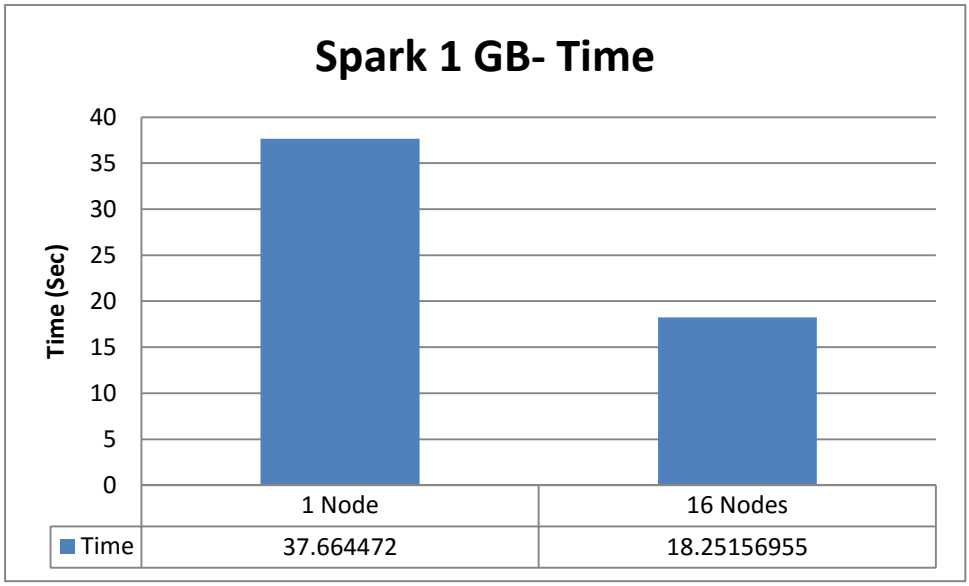
Spark 1 GB Last 10 Lines of sorted output:



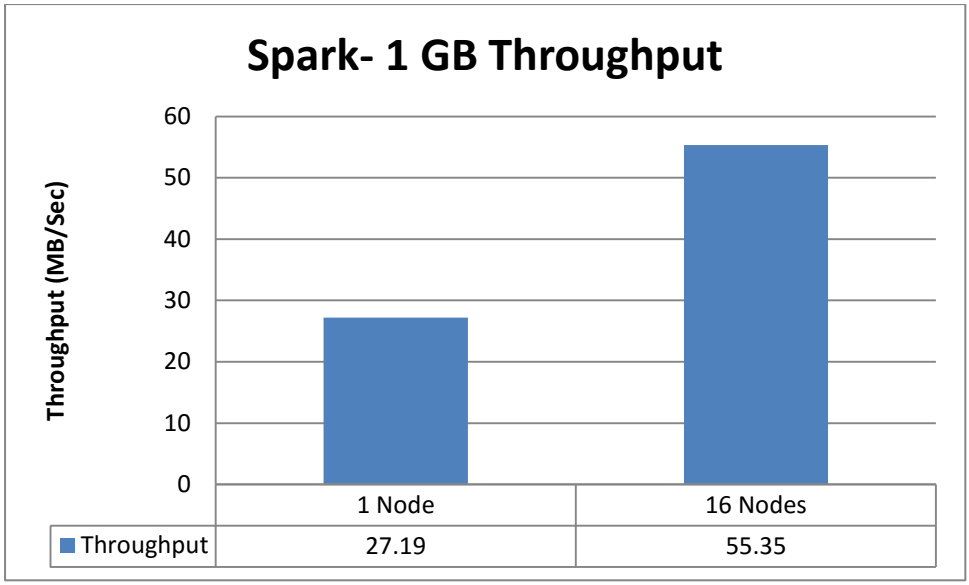
Valsort of Sorted output:



Trade-off of 1 GB Sorting Spark on 1 Node and 16 Nodes D2.xlarge:-



Throughput of 1 GB sorting Spark on 1 Node and 16 Nodes D2.xlarge:



Observing the graph of Time and Throughput, we can conclude that the spark's sorting of 1 GB with 1 Node compared with 16 Node will take half of the time, but the throughput is almost double. Resulting in faster performance.

C3.large 10 GB:

Time taken to sort 10 GB on 1 Node:

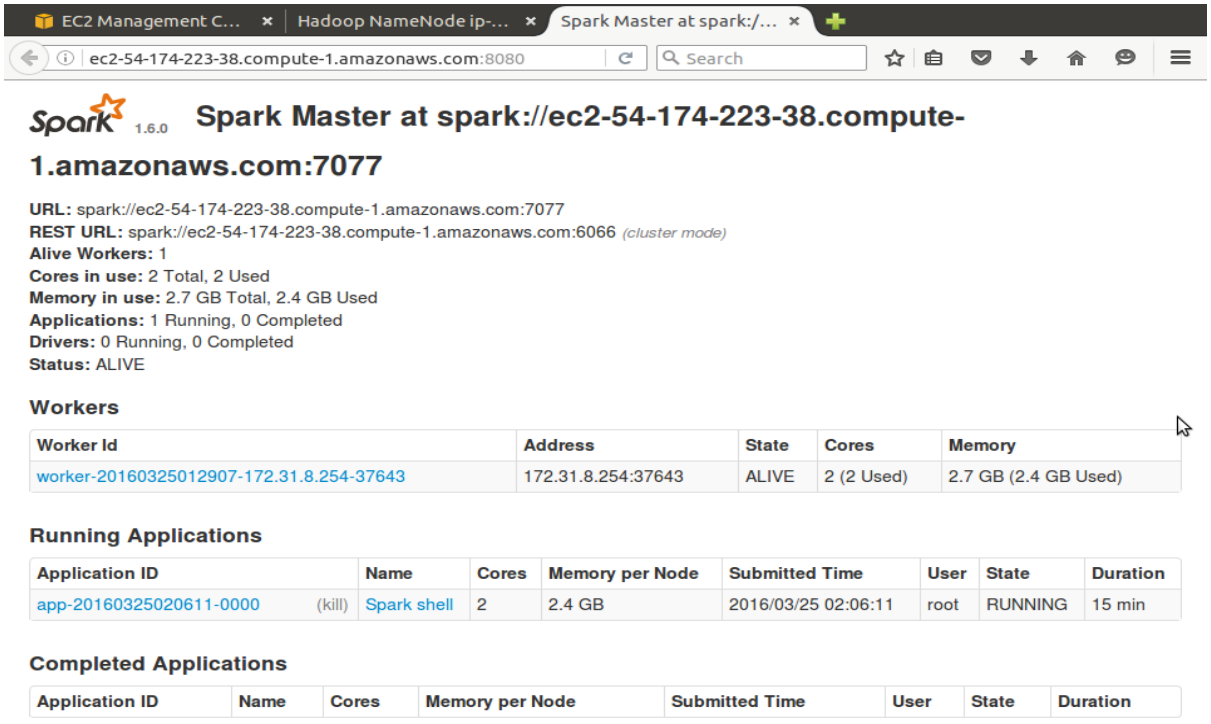
```
16/03/25 02:19:24 INFO scheduler.TaskSetManager: Finished task 66.0 in stage 2.0 (TID 216) in 10976 ms on
ip-172-31-8-254.ec2.internal (67/75)
16/03/25 02:19:35 INFO scheduler.TaskSetManager: Starting task 69.0 in stage 2.0 (TID 219, ip-172-31-8-254
.ec2.internal, partition 69,NODE_LOCAL, 1894 bytes)
16/03/25 02:19:35 INFO scheduler.TaskSetManager: Finished task 67.0 in stage 2.0 (TID 217) in 12172 ms on
ip-172-31-8-254.ec2.internal (68/75)
16/03/25 02:19:37 INFO scheduler.TaskSetManager: Starting task 70.0 in stage 2.0 (TID 220, ip-172-31-8-254
.ec2.internal, partition 70,NODE_LOCAL, 1894 bytes)
16/03/25 02:19:37 INFO scheduler.TaskSetManager: Finished task 68.0 in stage 2.0 (TID 218) in 12031 ms on
ip-172-31-8-254.ec2.internal (69/75)
16/03/25 02:19:47 INFO scheduler.TaskSetManager: Starting task 71.0 in stage 2.0 (TID 221, ip-172-31-8-254
.ec2.internal, partition 71,NODE_LOCAL, 1894 bytes)
16/03/25 02:19:47 INFO scheduler.TaskSetManager: Finished task 69.0 in stage 2.0 (TID 219) in 12066 ms on
ip-172-31-8-254.ec2.internal (70/75)
16/03/25 02:19:52 INFO scheduler.TaskSetManager: Starting task 72.0 in stage 2.0 (TID 222, ip-172-31-8-254
.ec2.internal, partition 72,NODE_LOCAL, 1894 bytes)
16/03/25 02:19:52 INFO scheduler.TaskSetManager: Finished task 70.0 in stage 2.0 (TID 220) in 15440 ms on
ip-172-31-8-254.ec2.internal (71/75)
16/03/25 02:20:00 INFO scheduler.TaskSetManager: Starting task 73.0 in stage 2.0 (TID 223, ip-172-31-8-254
.ec2.internal, partition 73,NODE_LOCAL, 1894 bytes)
16/03/25 02:20:00 INFO scheduler.TaskSetManager: Finished task 71.0 in stage 2.0 (TID 221) in 13330 ms on
ip-172-31-8-254.ec2.internal (72/75)
16/03/25 02:20:05 INFO scheduler.TaskSetManager: Starting task 74.0 in stage 2.0 (TID 224, ip-172-31-8-254
.ec2.internal, partition 74,NODE_LOCAL, 1894 bytes)
16/03/25 02:20:05 INFO scheduler.TaskSetManager: Finished task 72.0 in stage 2.0 (TID 222) in 12831 ms on
ip-172-31-8-254.ec2.internal (73/75)
16/03/25 02:20:11 INFO scheduler.TaskSetManager: Finished task 73.0 in stage 2.0 (TID 223) in 10798 ms on
ip-172-31-8-254.ec2.internal (74/75)
16/03/25 02:20:13 INFO scheduler.TaskSetManager: Finished task 74.0 in stage 2.0 (TID 224) in 8497 ms on i
p-172-31-8-254.ec2.internal (75/75)
16/03/25 02:20:13 INFO scheduler.DAGScheduler: ResultStage 2 (saveAsTextFile at <console>:30) finished in
430.176 s
16/03/25 02:20:13 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, f
rom pool
16/03/25 02:20:13 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at <console>:30, took 662.52
3950 s
t1: Long = 3812300433297
Elapsed time: 829036970881ns


scala>
```

Valsort:

```
root@ip-172-31-0-134 64]$ ./valsort output.txt
Records: 100000000
Checksum: 2fae59101920038
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-0-134 64]$
```

Spark Job:



First 10 Lines of output:

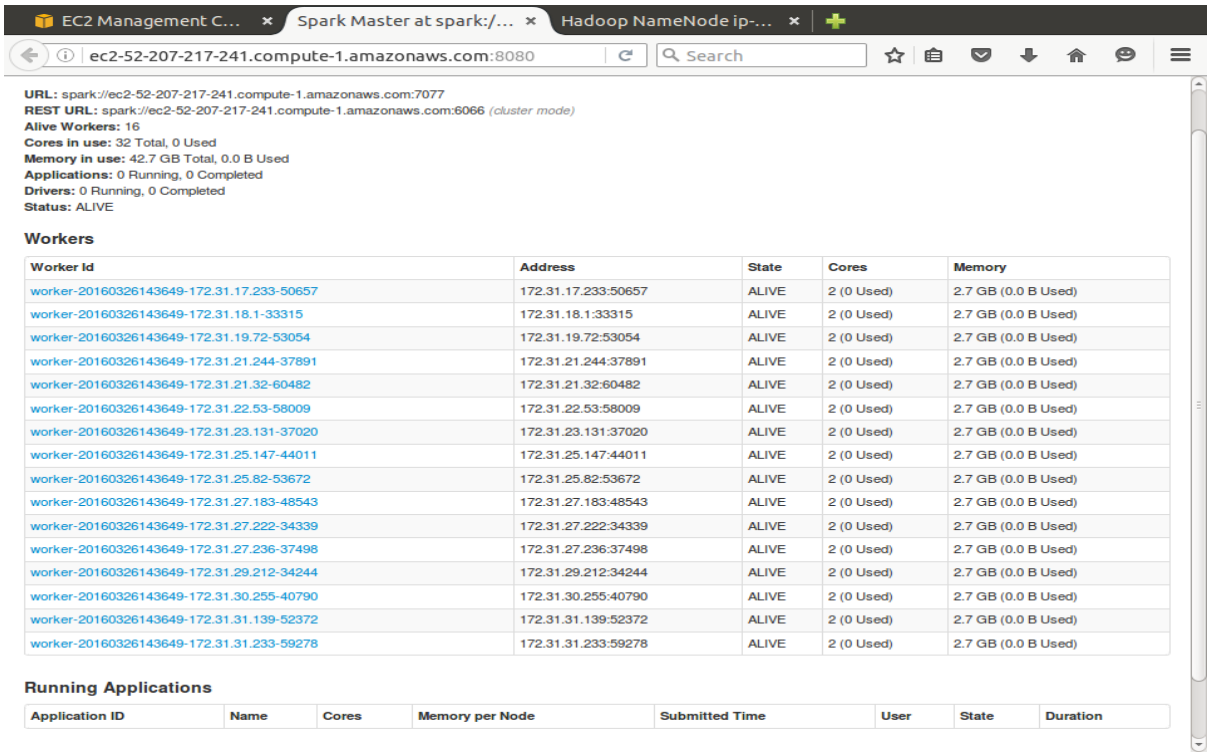
```
root@ip-172-31-0-134 bin]$ cd /vol0/output/
root@ip-172-31-0-134 output]$ ls
part-00000  part-00010  part-00020  part-00030  part-00040  part-00050  part-00060  part-00070
part-00001  part-00011  part-00021  part-00031  part-00041  part-00051  part-00061  part-00071
part-00002  part-00012  part-00022  part-00032  part-00042  part-00052  part-00062  part-00072
part-00003  part-00013  part-00023  part-00033  part-00043  part-00053  part-00063  part-00073
part-00004  part-00014  part-00024  part-00034  part-00044  part-00054  part-00064  part-00074
part-00005  part-00015  part-00025  part-00035  part-00045  part-00055  part-00065  _SUCCESS
part-00006  part-00016  part-00026  part-00036  part-00046  part-00056  part-00066
part-00007  part-00017  part-00027  part-00037  part-00047  part-00057  part-00067
part-00008  part-00018  part-00028  part-00038  part-00048  part-00058  part-00068
part-00009  part-00019  part-00029  part-00039  part-00049  part-00059  part-00069
root@ip-172-31-0-134 output]$ head -10 part-00000
    "O!uve   00000000000000000000000001228D4  77778888000022224444DDDDDDDDEEEE00000000CCCC7777DDDD
    PMd32=   00000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB9999333335555DDDDDDDD777788886666
    ^3CO],   00000000000000000000000000158C5C5  5555AAAA9999EEEE888822229999CCCCDDDD6666555544442222
   !&S3/]]   00000000000000000000000002145D78  8888BBBBDDDD1111CCCC55556666BBBB1111EEEEDDDD22229999
   !,=U#,9   00000000000000000000000019072E3  33332222FFFFBBBB0000FFFFAAAA666655553333DDDD3333CCCC
   !0f[ITd   00000000000000000000000003CAAB4B  9999FFFF5555333377777CCCC4444BBBB7777EEEEBBBBDDDD4444
   !f6Suy2   00000000000000000000000003ABFD84  EEEE555555556666AAAA5555BBBBDDDD00001111666600000DDDD
   #%NIpq.   00000000000000000000000003B36FB9  11110000333344441111666666666AAAAAAAA00001111CCCCEEEE
   #'^cl'~   00000000000000000000000002EDC5C8  8888AAAA11114444FFFF77773333EEEE44440000FFFF99999999
   $"-'Q)]   00000000000000000000000005F1265D  CCCC6666EEEE22220000DDDDAAAA88886666BBBB00006666AAAA
root@ip-172-31-0-134 output]$
```

Last 10 Lines of output:

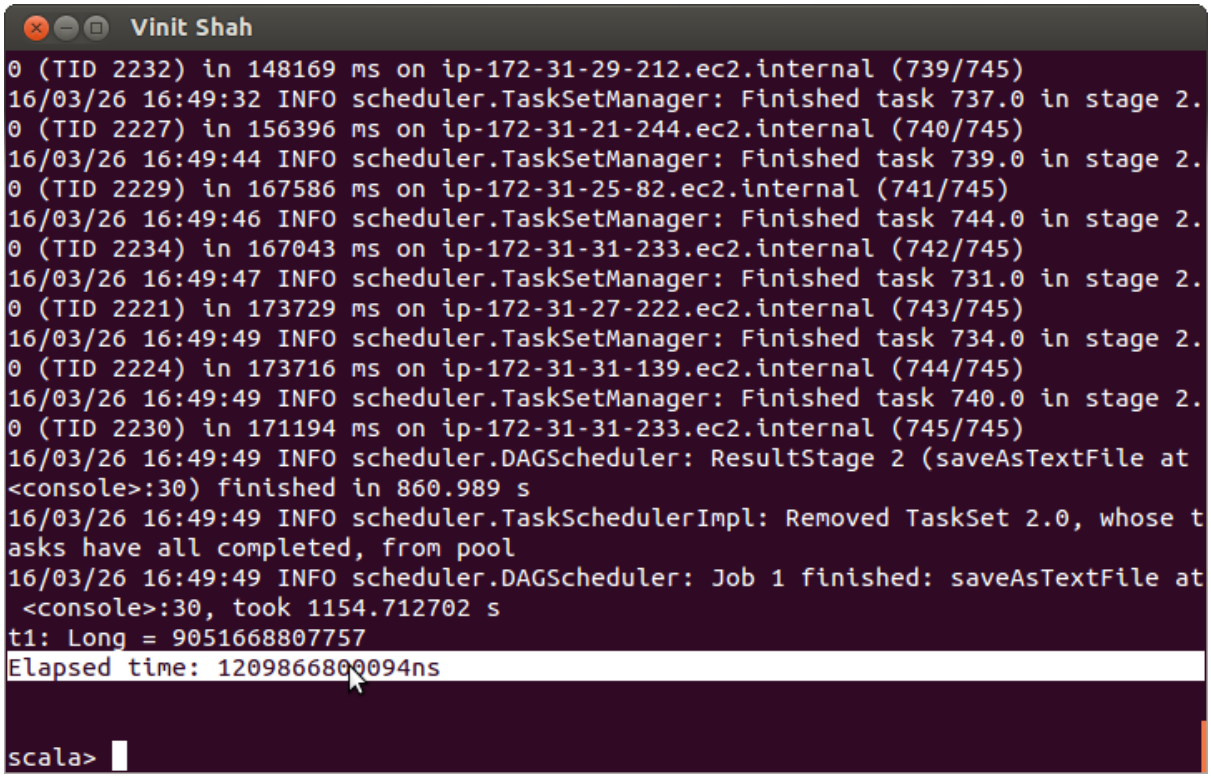
```
root@ip-172-31-0-134 output]$ ls
part-00000  part-00010  part-00020  part-00030  part-00040  part-00050  part-00060  part-00070
part-00001  part-00011  part-00021  part-00031  part-00041  part-00051  part-00061  part-00071
part-00002  part-00012  part-00022  part-00032  part-00042  part-00052  part-00062  part-00072
part-00003  part-00013  part-00023  part-00033  part-00043  part-00053  part-00063  part-00073
part-00004  part-00014  part-00024  part-00034  part-00044  part-00054  part-00064  part-00074
part-00005  part-00015  part-00025  part-00035  part-00045  part-00055  part-00065  _SUCCESS
part-00006  part-00016  part-00026  part-00036  part-00046  part-00056  part-00066
part-00007  part-00017  part-00027  part-00037  part-00047  part-00057  part-00067
part-00008  part-00018  part-00028  part-00038  part-00048  part-00058  part-00068
part-00009  part-00019  part-00029  part-00039  part-00049  part-00059  part-00069
root@ip-172-31-0-134 output]$ tail -10 part-00074
~~~uq2k#=U    00000000000000000000000002C06745   99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm    00000000000000000000000004709701   CCCC88883333FFFF000000000000099991111FFFF777744446666
~~~yKOl:gE    00000000000000000000000002048B4F   CCCC11114444888822226666BBBB888855557777EEEEBBBB0000
~~~yK^H.il    00000000000000000000000000463D004   44440000FFFF33339999444477777DDDDFFFFAAAA11118888DDDD
~~~yL;C'XE    00000000000000000000000005B0D211   2222EEEE33330000222221111CCCCFFFF5555777744444BBBB6666
~~~zbA_ Tt    00000000000000000000000007F9F4F   BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~zeO^FEg    00000000000000000000000001E06130   4444CCCCBBBB99992222888855558888CCCCFFFF000011111111
~~~}GxjWHI    00000000000000000000000000CA1345   777711118888AAAAAAAA22221111BBBB00002222BBBBCCCC2222
~~~}P;]g0g    00000000000000000000000040DA3E4   4444FFFF444466663333EEEE88888888DDDDEEEE44442222DDDD
~~~}kU|K<p    00000000000000000000000005E4A0AA   0000666655551111BBBB88889999AAAA55550000333355557777
root@ip-172-31-0-134 output]$
```

C3.large 100 GB:

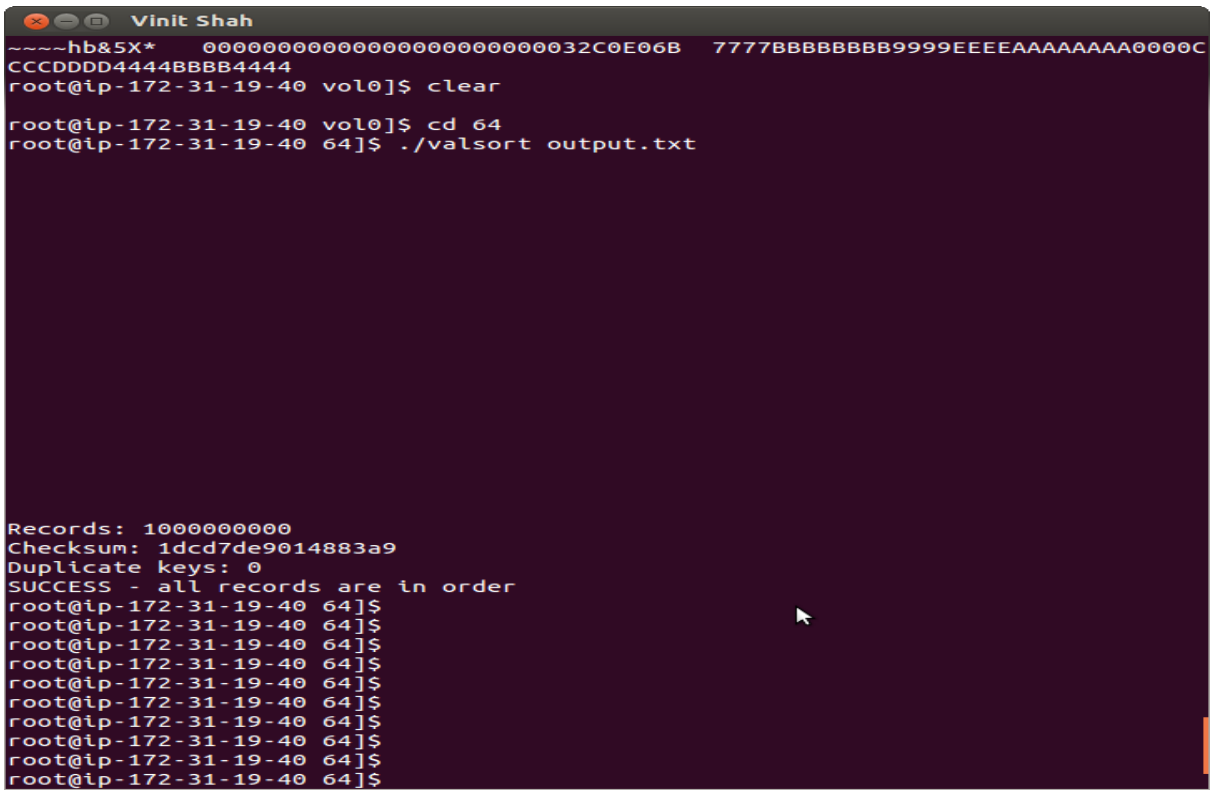
16 Nodes Cluster of Spark:



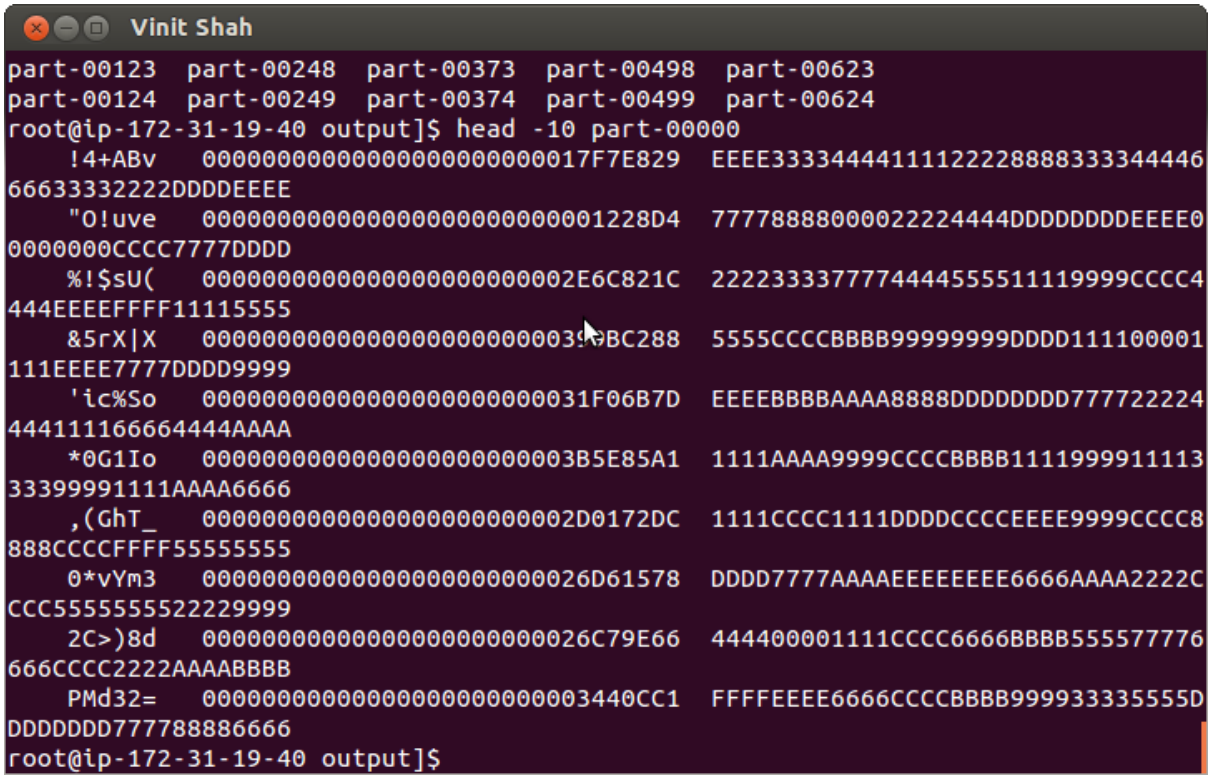
Time required



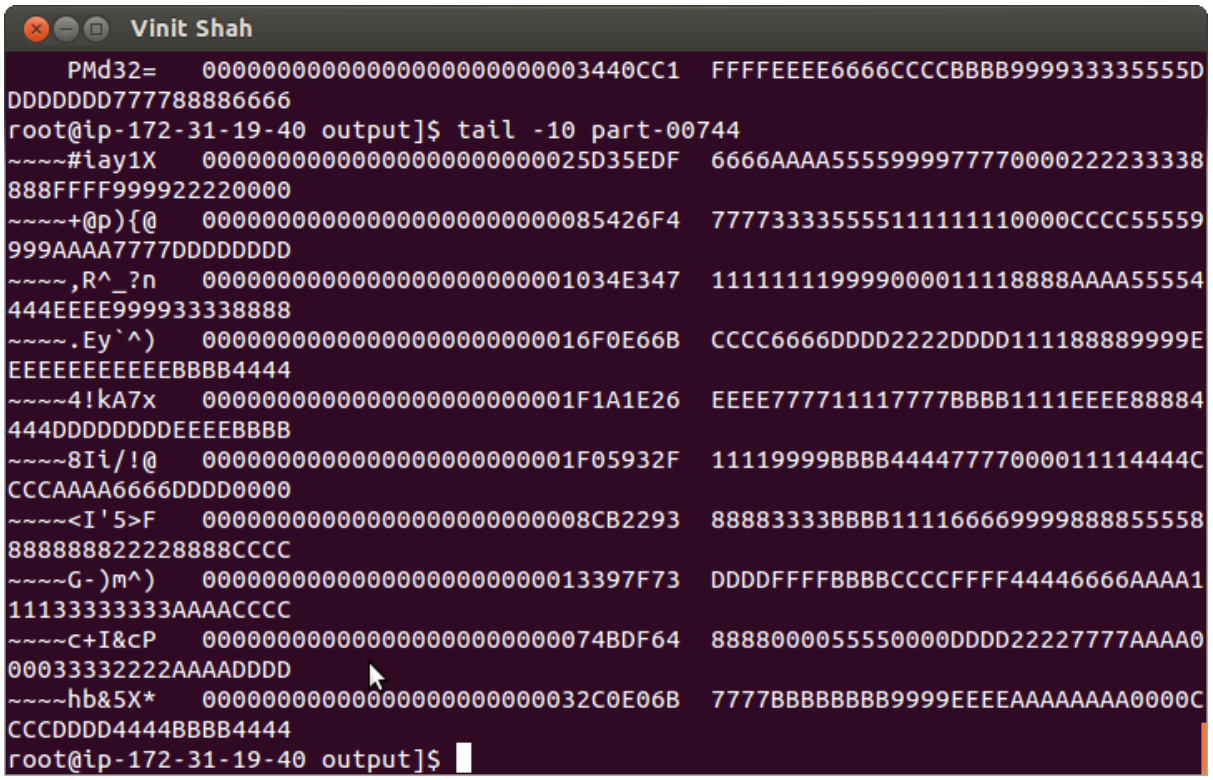
Valsort:



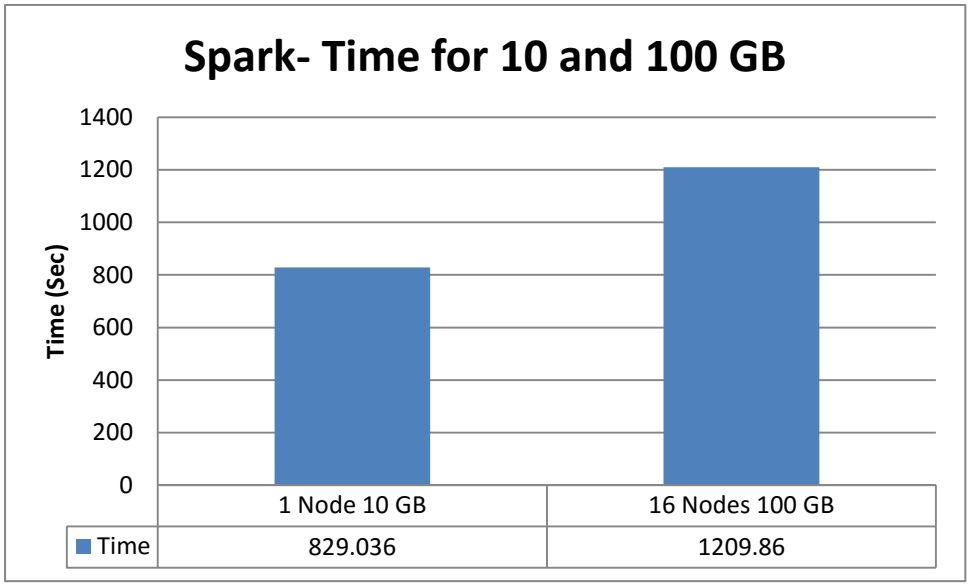
First 10 Lines of Output:



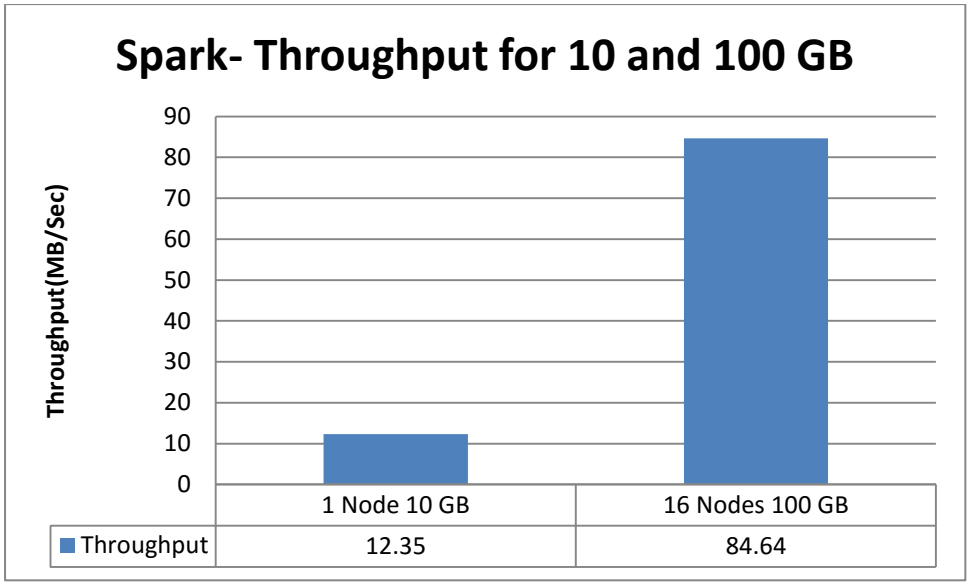
Last 10 Lines of output:



Spark Time trade-off of 1 Node 1 GB to 16 Nodes 100 GB:



Spark Throughput for 1 Node 1 GB and 16 Nodes 100 GB:



Spark on C3.large, observing the reading of time and throughput, we can conclude that the time to sort 10 GB and 100 GB is increased to around 400 sec, but the throughput is almost 7 times scaled-up. This gives the faster performance in the case of large dataset.

Trade-offs for Performance and Speed-up

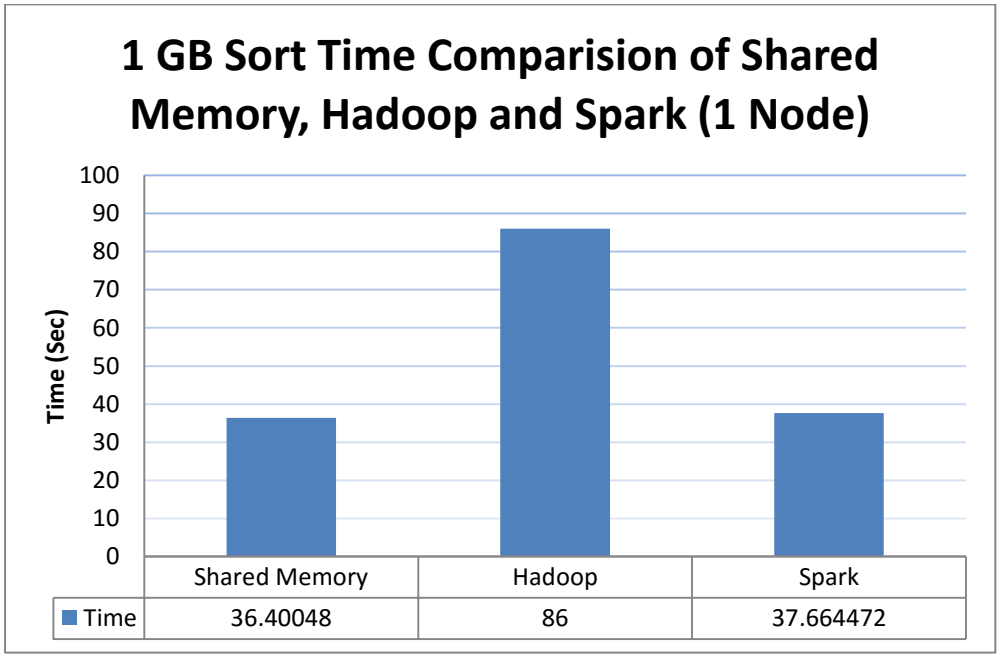
Trade-off of Shared Memory, Hadoop and Spark for 1 GB Sorting D2.xlarge:

Observing the time taken by each of the methods:

For Shared Memory – As we can see that, the amount of data to be sorted is big enough to fit in the memory available for d2.xlarge, so the time required to sort the 1 GB of data within memory is comparatively less, as compared to the other two counter-parts.

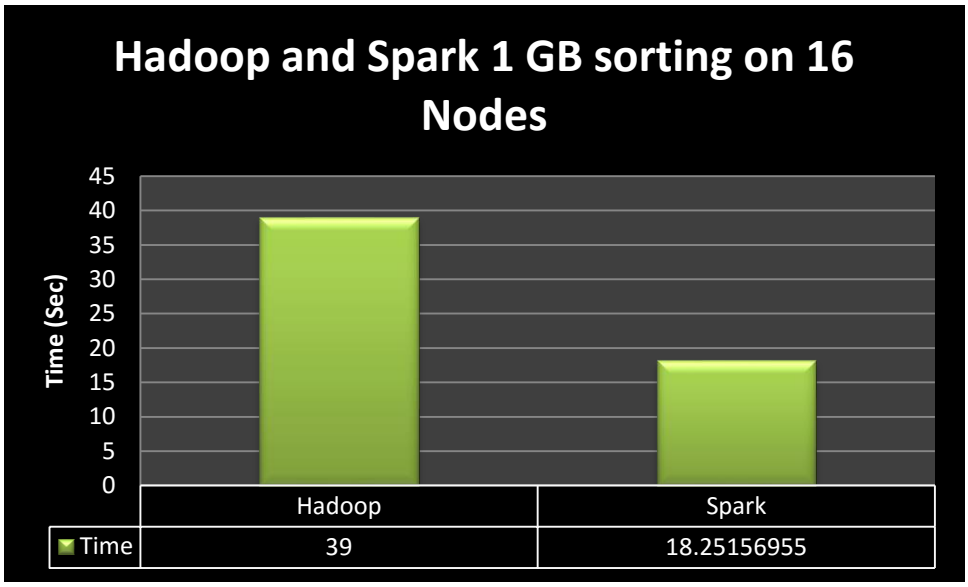
For Hadoop- The Hadoop uses the concept of map- reduce which is first loads the data in HDFS and then tries to map the data and shuffles individual chunks and write it to the HDFS during MAP phase, Whereas during the Reduce phase it reads those mapped shuffled data and tries to combine them based on some function, such as sort, so the time it takes is significantly large.

For Spark- It uses the Hadoop storage system for storage purpose where as it uses its own mechanism to map and reduce data, so it processes the data from HDFS, and applies the function on the data, such as Sorting, so it's much faster as compared to Hadoop.

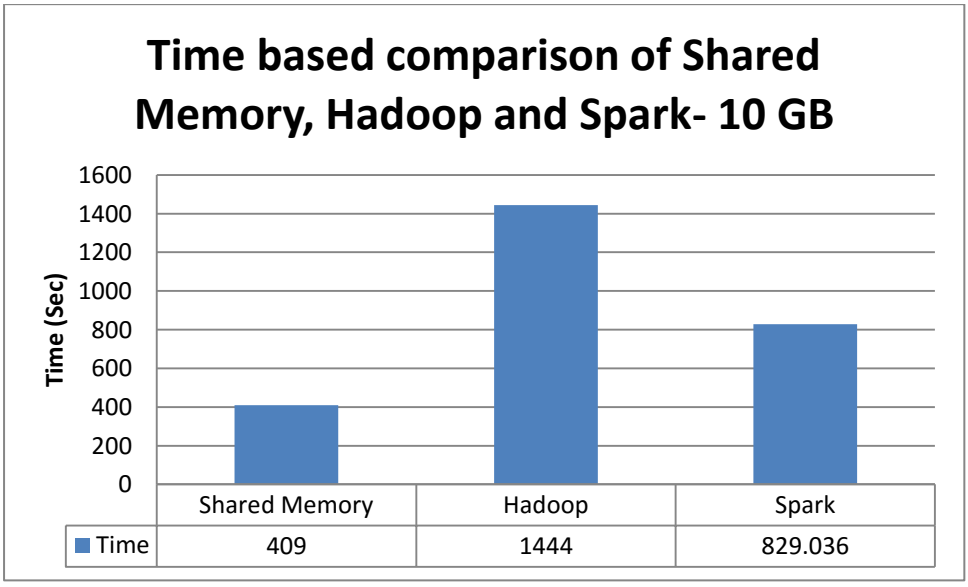


Trade-off of Hadoop and Spark for 1 GB Sorting with 16 Nodes D2.xlarge:

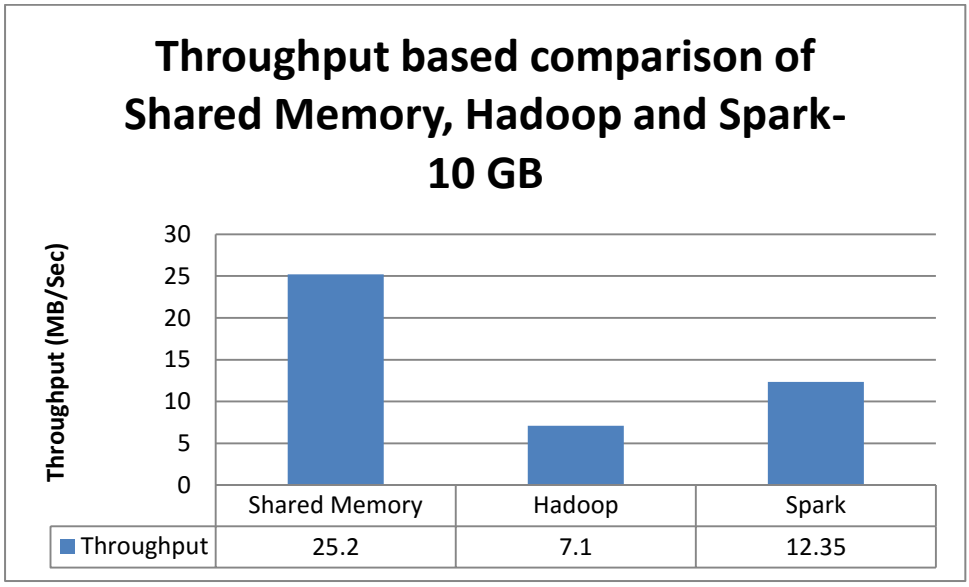
Scaling up from 1 Node to 16 nodes cluster for Hadoop and Spark, we can make out from the graph that, time required to sort 1 GB of data using Hadoop takes much time as compared to the spark. It's because the map phase of Hadoop creates individual chunks of mapper, which will take time to read and write the data to and from HDFS.



Time Trade-off of Shared Memory, Hadoop and Spark for 10 GB on C3.large:

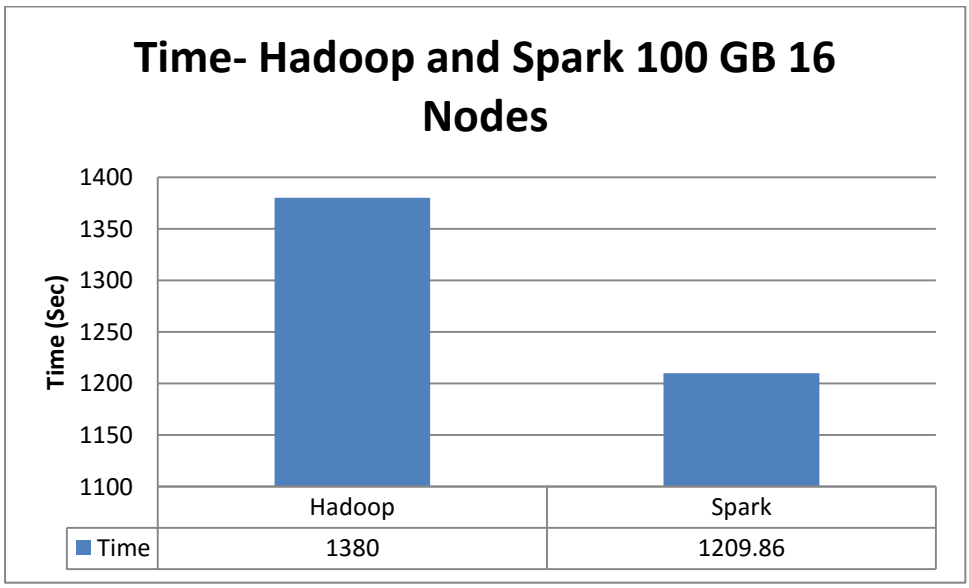


Speed-up Trade-off of Shared Memory, Hadoop and Spark for 10 GB on C3.large:

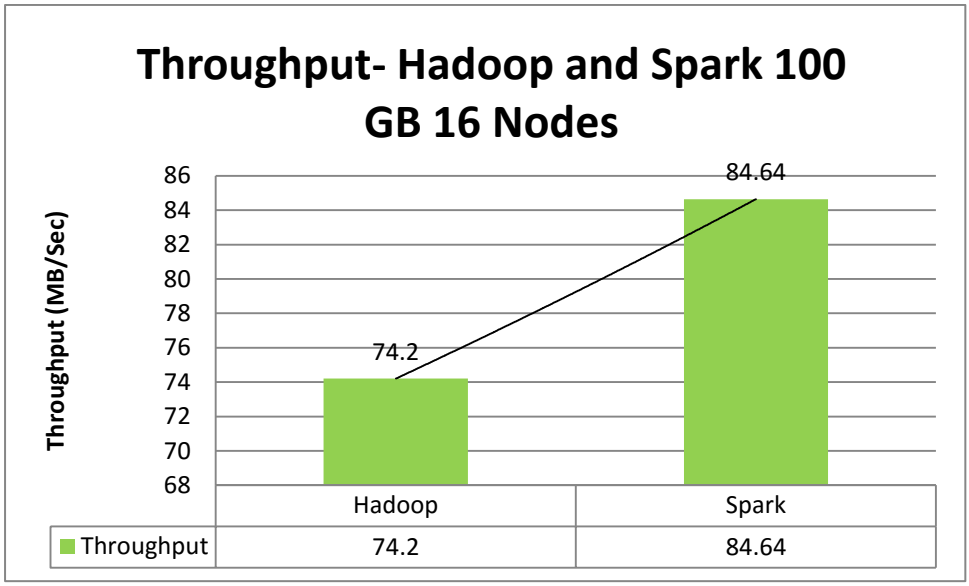


Based on the throughput and time based comparison, we can conclude that the time, required to sort Shared memory will be less, compared to the time required to sort the Hadoop and Spark, as shared memory uses the memory to sort data in place based on available free memory. Wherein, case of Hadoop and spark the time to sort data will be more, resulting in a less throughput as it sorts data based on fixed block size, which will get loaded in memory ad sorted and written back to disk.

Time Trade-off of Hadoop and Spark for 16 Nodes 100 GB on C3.large:



Speed-up graph of Hadoop and Spark for 16 Nodes 100 GB on C3.large:



As, we scale-up from 1 Node to 16 Nodes in Spark and Hadoop, the performace will drastically incresae by a factor. Where in case of increase in the time is certainly linear resulting in a steep slope of throughput.

Can you predict which would be best at 100 node scale?

Sorting with 1000 nodes will speed up the performance of sorting will improve, 463.75 MB/sec for Hadoop and for spark it would be 529.10 MB/Sec.

Can you predict which would be best at 1000 node scale?

Sorting with 1000 nodes will speed up the performance of sorting will improve, 1536.67 MB/sec for Hadoop and for spark it would be 1874.34 MB/Sec.

Message Passing Interface (MPI)

Message Passing Interface (MPI) is a library specification to create parallel programs, for parallel computers, clusters, and heterogeneous networks.

MPI consists of:

1. a header file mpi.h
2. a library of routines and functions
3. a runtime system

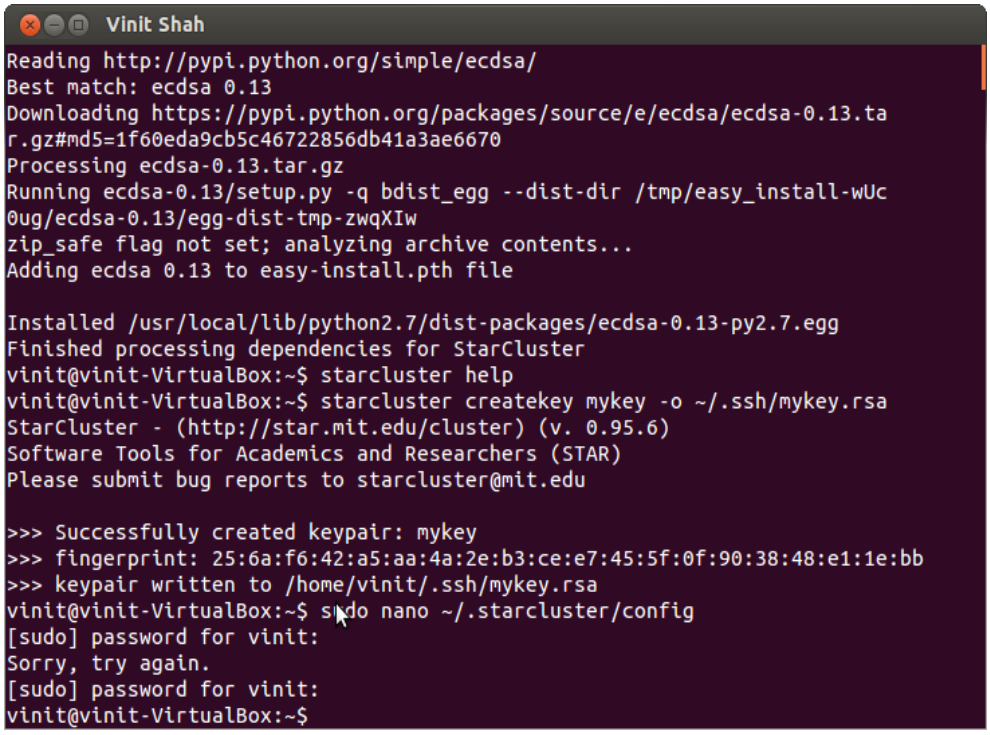
It's designed to provide access to advanced parallel hardware for

1. End users
2. Library Writers
3. Tool Developers

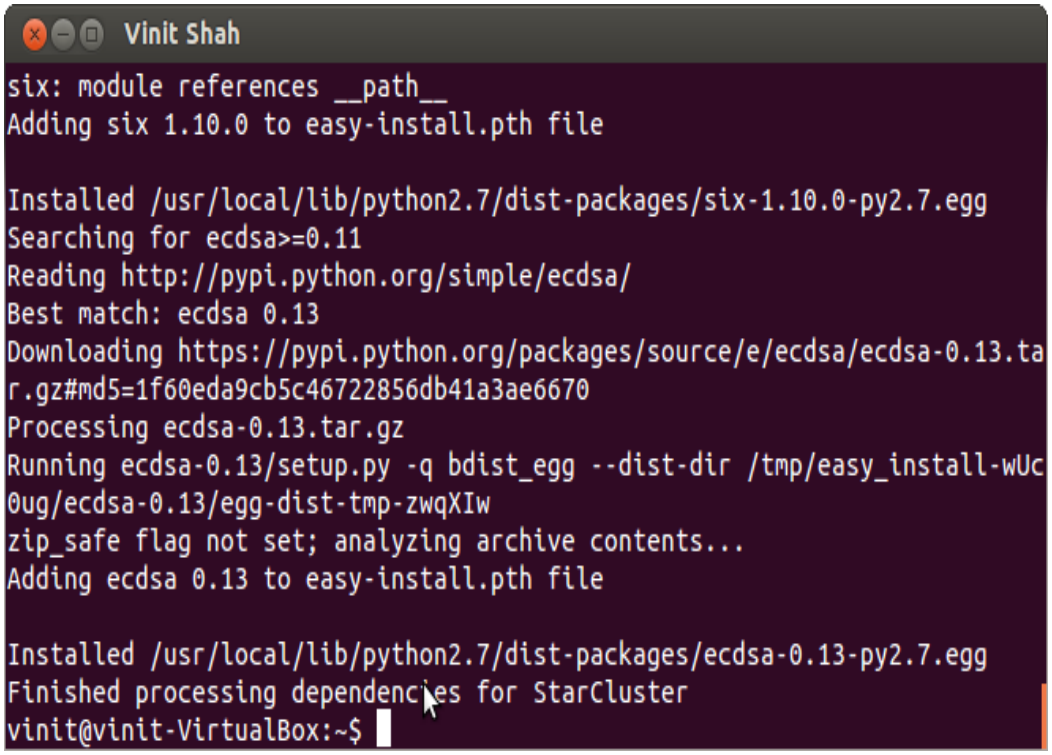
Follow the following steps to setup and start the star Cluster:

Configuration Steps

Star Cluster Setup for AWS:



Star Cluster Installed:



Starting the star Cluster:

```
StarCluster - (http://star.mit.edu/cluster) (v. 0.95.6)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Using default cluster template: largecluster
>>> Validating cluster template settings...
>>> Cluster template settings are valid
>>> Starting cluster...
>>> Launching a 2-node cluster...
>>> Creating security group @sc-mpicluster...
Reservation:r-74c9a1a6
>>> Waiting for instances to propagate...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for cluster to come up... (updating every 30s)
>>> Waiting for all nodes to be in a 'running' state...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for SSH to come up on all nodes...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Waiting for cluster to come up took 1.916 mins
>>> The master node is ec2-52-90-19-117.compute-1.amazonaws.com
>>> Configuring cluster...
>>> Running plugin starcluster.clustersetup.DefaultClusterSetup
>>> Configuring hostnames...
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Creating cluster user: sgeadmin (uid: 1001, gid: 1001)
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring scratch space for user(s): sgeadmin
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Configuring /etc/hosts on each node
2/2 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Starting NFS server on master
>>> Configuring NFS exports path(s):
/home
>>> Mounting all NFS export path(s) on 1 worker node(s)
1/1 |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| 100%
>>> Setting up NFS took 0.235 mins
>>> Configuring passwordless ssh for root
>>> Configuring passwordless ssh for sgeadmin
>>> Running plugin starcluster.plugins.sge.SGEPlugin
>>> Configuring SGE...
```

Login into Star Cluster Master:

```
__/\_____| |_ __ _ _ __ ___| |_   _ ___| |_ ___ _ __
\    / __| __/ _` | '__/ __| | | | / __| __/ _ \ '__|
/_  _\__ \ || (_| | | | (__| | |_| \__ \ ||  __/ |
  \/ |___/\__\__,_|_|  \___|_|\__,_|___/\__\___|_|

StarCluster Ubuntu 13.04 AMI
Software Tools for Academics and Researchers (STAR)
Homepage: http://star.mit.edu/cluster
Documentation: http://star.mit.edu/cluster/docs/latest
Code: https://github.com/jtriley/StarCluster
Mailing list: http://star.mit.edu/cluster/mailinglist.html

This AMI Contains:

  * Open Grid Scheduler (OGS - formerly SGE) queuing system
  * Condor workload management system
  * OpenMPI compiled with Open Grid Scheduler support
  * OpenBLAS - Highly optimized Basic Linear Algebra Routines
  * NumPy/SciPy linked against OpenBlas
  * Pandas - Data Analysis Library
  * IPython 1.1.0 with parallel and notebook support
  * Julia 0.3pre
  * and more! (use 'dpkg -l' to show all installed packages)

Open Grid Scheduler/Condor cheat sheet:

  * qstat/condor_q - show status of batch jobs
  * qhost/condor_status- show status of hosts, queues, and jobs
  * qsub/condor_submit - submit batch jobs (e.g. qsub -cwd ./job.sh)
  * qdel/condor_rm - delete batch jobs (e.g. qdel 7)
  * qconf - configure Open Grid Scheduler system

Current System Stats:

  System load:  0.13              Processes:           135
  Usage of /:   34.6% of 7.84GB   Users logged in:     0
  Memory usage: 0%                IP address for eth0: 172.31.10.42
  Swap usage:   0%

    https://landscape.canonical.com/
root@master:~#
```