

# 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 1<sup>st</sup> 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 3<sup>rd</sup> 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:

```
ubuntu@ip-172-31-14-204: ~
ubuntu@ip-172-31-14-204:~$ ant run
Buildfile: /home/ubuntu/build.xml

compile:
    [mkdir] Created dir: /home/ubuntu/build/classes
    [javac] Compiling 9 source files to /home/ubuntu/build/classes

jar:
    [mkdir] Created dir: /home/ubuntu/build/jar
    [jar] Building jar: /home/ubuntu/build/jar/SharedMemory.jar

run:
    [java] Total Time Taken to Sort in Miliseconds: 36400.48499

BUILD SUCCESSFUL
Total time: 37 seconds
ubuntu@ip-172-31-14-204:~$
```

## Valsort Output:

```
ubuntu@ip-172-31-14-204:~/64
ubuntu@ip-172-31-14-204:~/64$ ./valsort outputfile.txt
Records: 10000000
Checksum: 4c48a881c779d5
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-14-204:~/64$
```

### Screenshot of First 10 lines of sorted output:

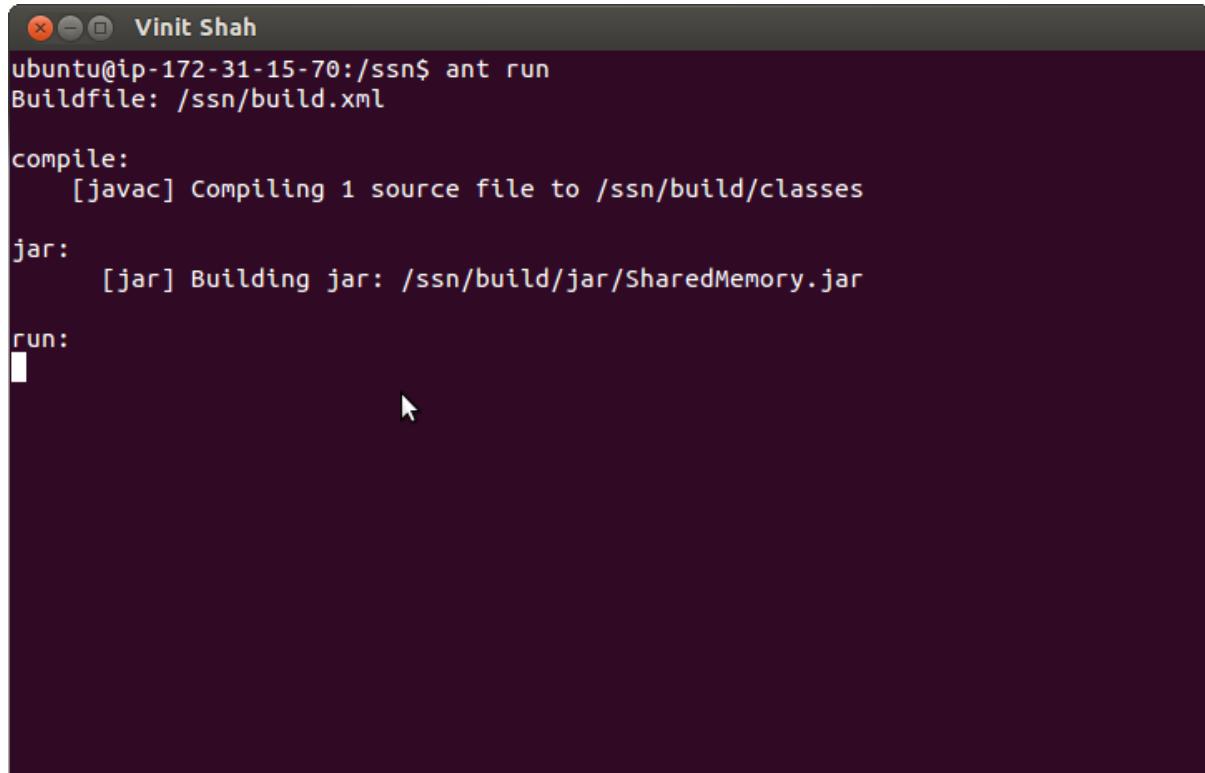
```
ubuntu@ip-172-31-14-204: ~/64
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-14-204:~/64$ head -10 outputfile.txt
    "O!uve 000000000000000000000000001228D4 77778888000022224444DDDDDDDEEEE00
000000CCCC7777DDDD
    ,K4a-:v 000000000000000000000000000000001B8132 5555EEEE888899994444FFFF1111CCCCEE
EE1111EEEE6666FFFF
    .FuD\}u 00000000000000000000000000000000797631 5555DDDBBBB00007777222111122244
44DDDDDDDD99996666
    ;SYThct 000000000000000000000000000000007D3DF5 2222AAAACCCCFFFFAAAA44445555EEEE44
442222DDDD99992222
    =2G^9{- 00000000000000000000000000000000809EE 5555DDDD1111CCCC9999BBBB0000BBBBCC
CCFFFC44443333
    N}M9?SP 00000000000000000000000000000000429597 5555FFFF00007777555599991111CCCC66
669999AAAEEE8888
    P0X?Rs& 0000000000000000000000000000000041162E 888833399999FFFF1111CCCC8888CCCC99
99EEEEEDDD00003333
    [Xq]\%$ 0000000000000000000000000000000097A5F0 66666666EEEEDDD7777FFFF00005555FF
FFFFF88885551111
    rAmQg4v 000000000000000000000000000000008180D3 BBBB111111119999FFFFFFF99999999
44BBBB88884444CCCC
    !))Jf3; 000000000000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCEE
EE5555666666666666
ubuntu@ip-172-31-14-204:~/64$
```

### Screenshot of last 10 lines of sorted output:

```
ubuntu@ip-172-31-14-204: ~/64
!&){JF3; 000000000000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCCE
EE5555666666666666
ubuntu@ip-172-31-14-204:~/64$ tail -10 outputfile.txt
~~~%A NB_t 000000000000000000000000000000003DE5EC FFFF7777EEEEBBBB4444EEEEEEE333399
99DDDD999900005555
~~~c-CQ(> 00000000000000000000000000000000832611 9999FFFF11117777333377770000111144
4444440000BBB6666
~~~8Y)FqI* 00000000000000000000000000000000742A4C BBBB1111CCCCEEE888800000000777733
333333DDD22225555
~~~>Dd=QT] 00000000000000000000000000000000674C0F 9999DDDD000055556666CCCC2222000FF
FFEEEEEDDDDDFFF0000
~~~]JA(){j$ 0000000000000000000000000000000060BCBE 111122223333444455559999AAAABBBB99
99FFFFDDDDBBBBB3333
~~~]Zp.#/+ 000000000000000000000000000000003B9A5A CCCC8888EEEEAAAAEEE3333333777700
00FFFFCCCC66667777
~~~]Qepix 0000000000000000000000000000000011E5D4 1111999911115555BBBB111100002222EE
EE6666BBBBB7777DDD
~~~nt=ZH[N 000000000000000000000000000000000332A13 44441111BBBBBBBBB33337777FFFF444455
555555333000CCCC
~~~s/Pq,-E 000000000000000000000000000000006BE930 2222DDDDDDDD77771111EEEECCCC7777BB
BB4444888811111111
~~~zBA_Tt 000000000000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAA11
116666AAABBBB0000
ubuntu@ip-172-31-14-204:~/64$
```

## D2.xlarge 1 TB:

1 TB started:



```
Vinit Shah
ubuntu@ip-172-31-15-70:/ssn$ ant run
Buildfile: /ssn/build.xml

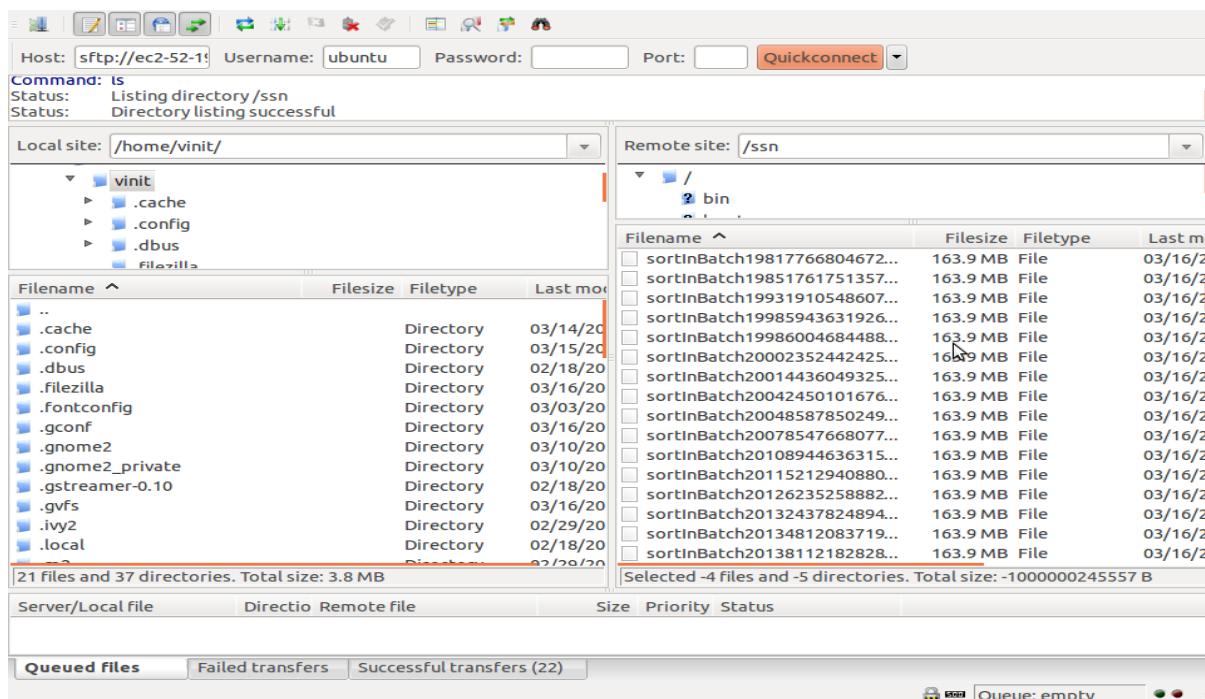
compile:
[javac] Compiling 1 source file to /ssn/build/classes

jar:
[jar] Building jar: /ssn/build/jar/SharedMemory.jar

run:

```

1 TB temporary file generations:



**1 TB time to sort:**

```
BUILD SUCCESSFUL
Total time: 239 minutes 34 seconds
ubuntu@ip-172-31-15-70:/ssn$ cd 64
ubuntu@ip-172-31-15-70:/ssn/64$ clear

ubuntu@ip-172-31-15-70:/ssn/64$ ./valsort output.txt
```

### **C3.large 1 GB**

**1 GB Time:**

```
ubuntu@ip-172-31-10-219:~$ ant run
Buildfile: /home/ubuntu/build.xml

compile:

jar:

run:

BUILD SUCCESSFUL
Total time: 32 seconds
ubuntu@ip-172-31-10-219:~$
```

## Valsort output:

```
Vinit Shah
ubuntu@ip-172-31-10-219:~$ ant run
Buildfile: /home/ubuntu/build.xml

compile:

jar:

run:

BUILD SUCCESSFUL
Total time: 32 seconds
ubuntu@ip-172-31-10-219:~$ cd 64
ubuntu@ip-172-31-10-219:~/64$ unix2dos outputfile.txt
unix2dos: converting file outputfile.txt to DOS format ...
ubuntu@ip-172-31-10-219:~/64$ ./valsrt outputfile.txt
Records: 10000000
Checksum: 4c48a881c779d5
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-10-219:~/64$
```

## First 10 lines of output:

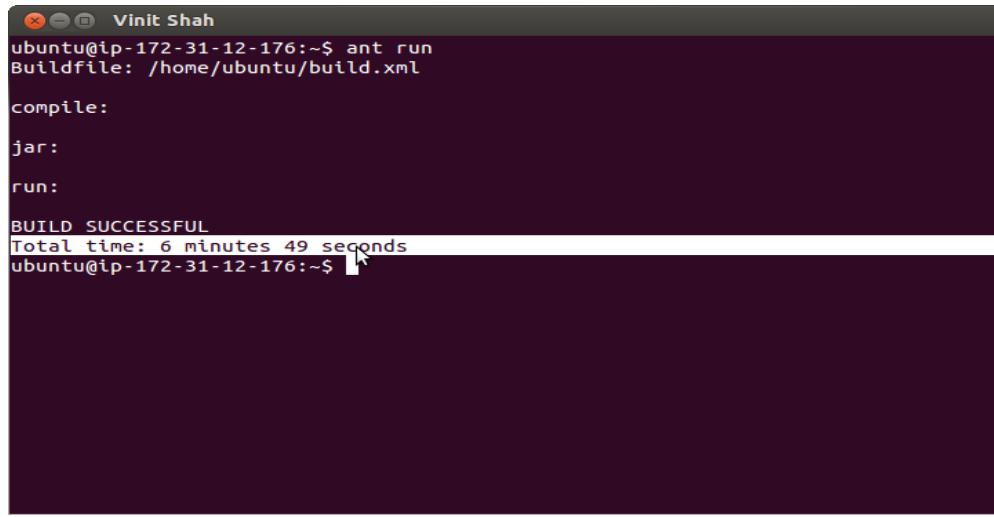
```
Vinit Shah
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-10-219:~/64$ head -10 outputfile.txt
    "O!uve 000000000000000000000000000000001228D4 7777888800022224444DDDDDDDEEEE00
000000CCCC7777DDDD
    ,K4a:v 00000000000000000000000000000001B8132 5555EEEE888899994444FFFF1111CCCC
EE1111EEEE6666FFFF
    .FuDvju 0000000000000000000000000000000797631 5555DDDBBBB00007772221111222244
44DDDDDDDD99996666
    ;5Ytht 00000000000000000000000000000007D3DF5 2222AAAACCCCFFFFAAAA44445555EEEE44
442222DDDD99992222
    =2G^9[- 0000000000000000000000000000000809EE 5555DDDD1111CCCC9999BBBB0000BBBBCC
CCFFFFCCCC44443333
    N}M9?sp 0000000000000000000000000000000429597 5555FFFF000077755599991111CCCC66
669999AAAAEEE8888
    POX?Rs& 000000000000000000000000000000041162E 888833339999FFFF1111CCCC8888CCCC99
99EEEEEDDD00003333
    [Xq\\$% 000000000000000000000000000000097A5F0 66666666EEEEDDDD7777FFFF00005555FF
FFFFFF888855551111
    rAmQg4v 00000000000000000000000000000008180D3 BBBB111111119999FFFFFFBBBBBBBB44
44BBBBB88884444CCCC
    !&))Jf3; 000000000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCEE
EE5555666666666666
ubuntu@ip-172-31-10-219:~/64$
```

## Last 10 Lines of output:

```
!&))Jf3; 000000000000000000000000000000004602E1 4444FFFFCCCC88888888CCCCFFFFCCCCEE  
EE555566666666666666  
ubuntu@ip-172-31-10-219:~/64$ tail -10 outputfile.txt  
~~~%A NB_t 00000000000000000000000000000003DE5EC FFFF7777EEEBBBBB4444EEEEEEE333399  
99DDDD999900005555  
~~~c-CQ(> 0000000000000000000000000000000832611 9999FFFF1111777333377770000111144  
4444440000BBB6666  
~~~8YFqL* 0000000000000000000000000000000742A4C BBBB1111CCCCEE888800000000777733  
333333DDDD22225555  
~~~Dd=QT] 0000000000000000000000000000000674C0F 99999DDDD000055556666CCCC22220000FF  
FFEEEEEDDDDDFFF0000  
~~~)JA(){j$ 000000000000000000000000000000060BCBE 11112223333444455559999AAABBBB99  
99FFFFDDDBBBB3333  
~~~)Zp.#/+ 00000000000000000000000000000003B9A5A CCCC8888EEEEAAAEEE3333333777700  
00FFFFCCCC66667777  
~~~_jQepix 000000000000000000000000000000011E5D4 111199991111555BBBBB111100002222EE  
EE6666BBB7777DDDD  
~~~nt=ZH[N 0000000000000000000000000000000332A13 44441111BBBBBBBBB33337777FFFF444455  
5555553330000CCCC  
~~~s/Pq,-E 00000000000000000000000000000006BE930 2222DDDDDDDD77771111EEEECCCC7777BB  
BB4444888111111111  
~~~zBa_Tt 00000000000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAA11  
116666AAABBBB0000  
ubuntu@ip-172-31-10-219:~/64$
```

## C3.large 10 GB

**10 GB Sort Time:**



```
Vinit Shah
ubuntu@ip-172-31-12-176:~$ ant run
Buildfile: /home/ubuntu/build.xml

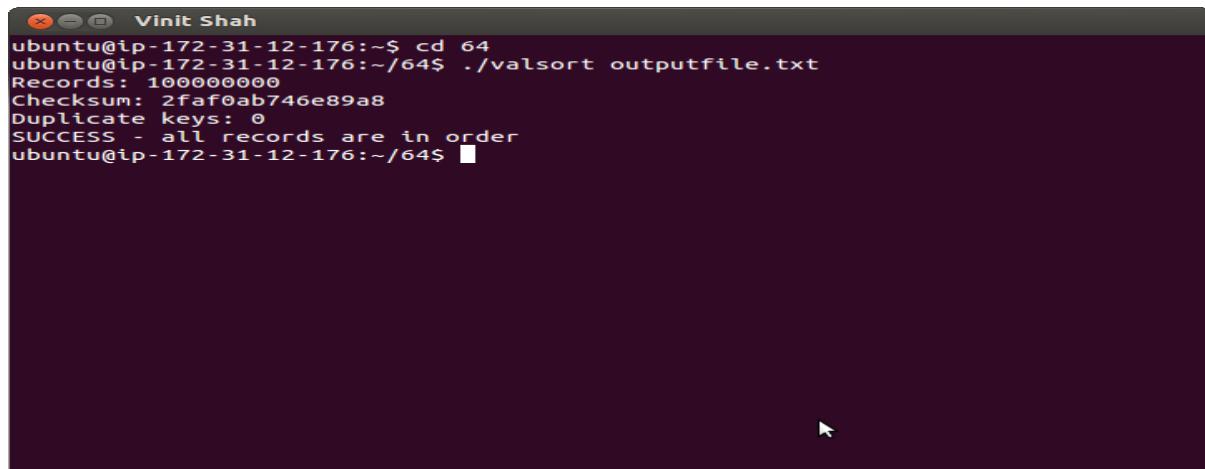
compile:

jar:

run:

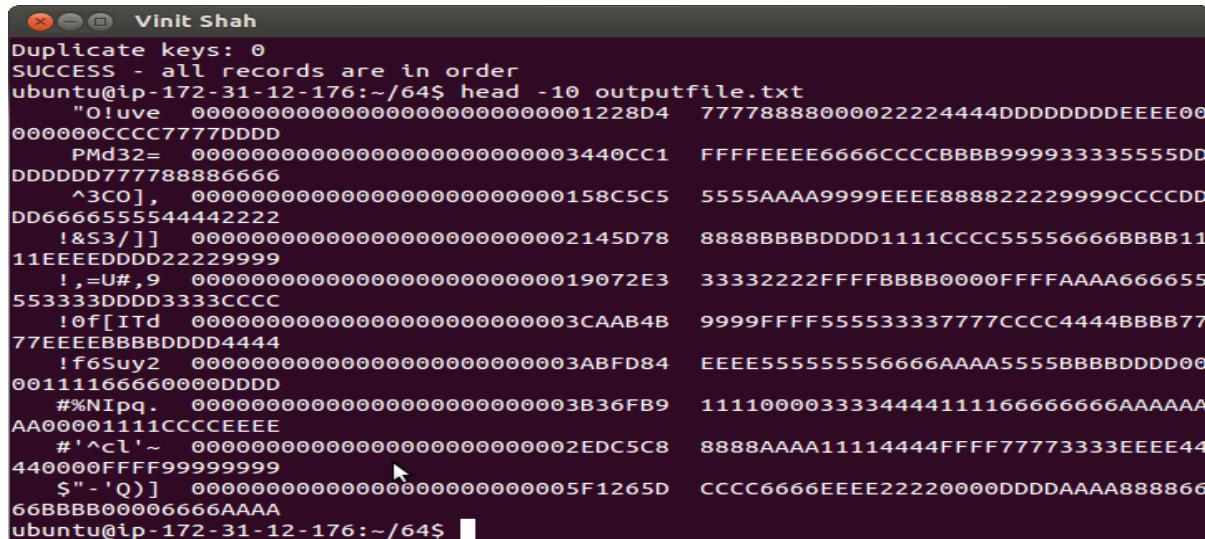
BUILD SUCCESSFUL
Total time: 6 minutes 49 seconds
ubuntu@ip-172-31-12-176:~$
```

**Valsort:**



```
Vinit Shah
ubuntu@ip-172-31-12-176:~$ cd 64
ubuntu@ip-172-31-12-176:~/64$ ./valsrt outfile.txt
Records: 100000000
Checksum: 2faf0ab746e89a8
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-12-176:~/64$
```

**First 10 Lines of output:**

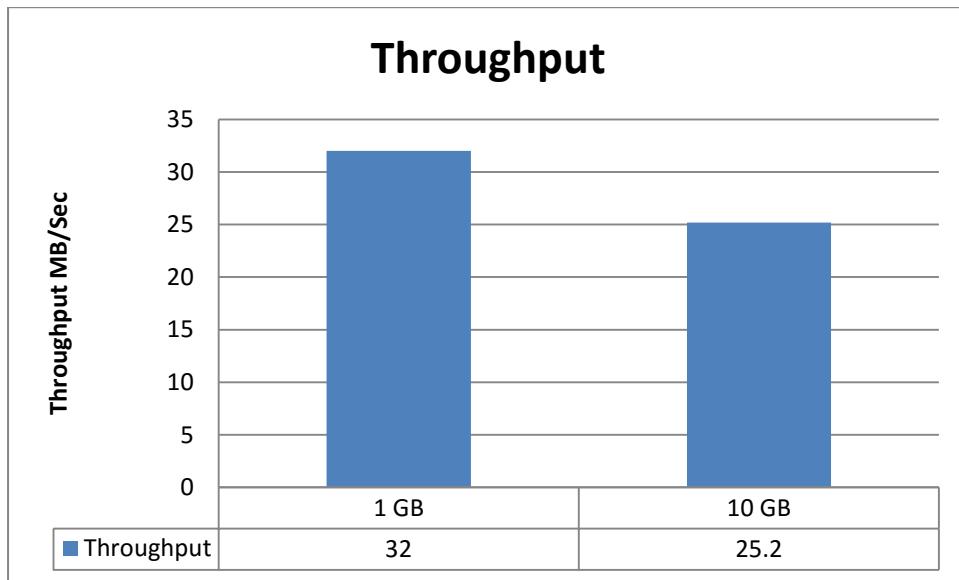


```
Vinit Shah
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-12-176:~/64$ head -10 outfile.txt
"!uv'e 000000000000000000000000000000001228D4 77778888000022224444DDDDDDDEEEE00
000000CCCC7777DDDD
    PMd32= 000000000000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB999933335555DD
DDDDDD777788886666
    ^3C0], 00000000000000000000000000000000158C5C5 5555AAA9999EEEE888822229999CCCCDD
DD666655554442222
    !&S3/] ] 000000000000000000000000000000002145D78 8888BBBBBDDDD1111CCCC55556666BBBB11
11EEEEEDDD22229999
    ! ,=U#,9 0000000000000000000000000000000019072E3 33332222FFFFBBBB0000FFFFAAAA666655
553333DDDD3333CCCC
    !Of[ITd 000000000000000000000000000000003CAAB4B 9999FFFF555533337777CCCC4444BBBB77
77EEEEBBBBBDDDD4444
    !f6Suy2 000000000000000000000000000000003ABFD84 EEEE55555556666AAA5555BBBBDDDD00
00111166660000DDDD
    #%"NIpq. 000000000000000000000000000000003B36FB9 111100003333444411116666666AAAAAA
AA00001111CCCCEEE
    #'^c\l'~ 000000000000000000000000000000002EDC5C8 8888AAAA11114444FFFF77773333EEEE44
440000FFF9999999
    $"-'Q)] 00000000000000000000000000000005F1265D CCCC6666EEEE22220000DDDDAAAA888866
66BBBB00006666AAAA
ubuntu@ip-172-31-12-176:~/64$
```

Last 10 Lines of output:

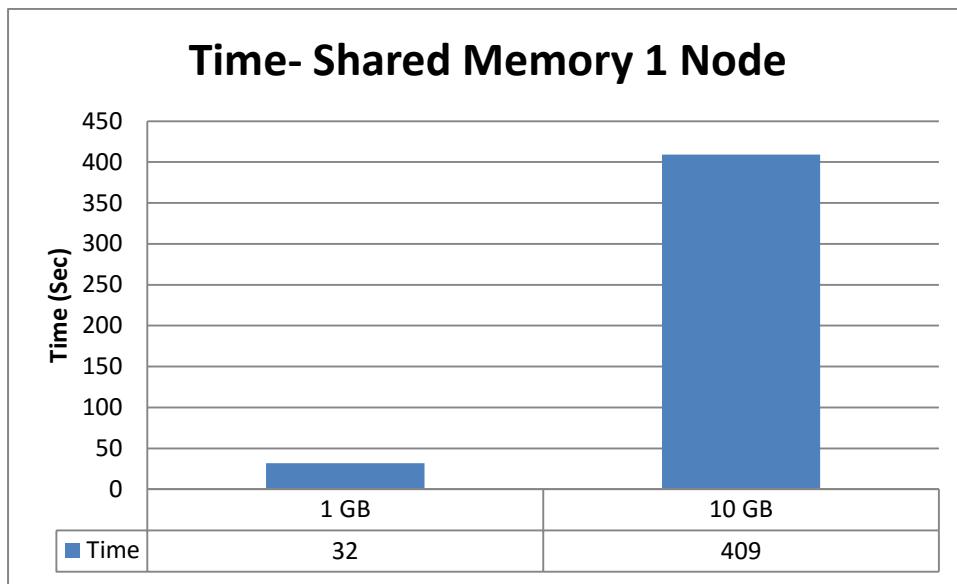
```
Vinit Shah
$"- 'Q)] 00000000000000000000000000000005F1265D  CCCC6666EEEE22220000DDDDAAAA888866
66BBBB00006666AAAA
ubuntu@ip-172-31-12-176:~/64$ tail -10 outputfile.txt
~~~uq2k#=U 00000000000000000000000000000002C06745  99991111DDDD22221110000FFFFEEEFF
FF33337777CCCC2222
~~~v/0&Qnm 00000000000000000000000000000004709701  CCCC88883333FFFF000000000000999911
11FFFFF777744446666
~~~yK0l:ge 00000000000000000000000000000002048B4F  CCCC11114444888822226666BBBB888855
557777EEEBBBBB0000
~~~yK^H.il 0000000000000000000000000000000463D004  44440000FFFF3333999944447777DDDDFF
FFAAAAA11118888DDDD
~~~yL;C'XE 00000000000000000000000000000005B0D211  2222EEEE3333000022221111CCCCFFFF55
5577774444BBBB6666
~~~zba_Tt 00000000000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAAA11
116666AAAA BBBB0000
~~~ze0^FEg 00000000000000000000000000000001E06130  4444CCCCBBBB99992222888855558888CC
CCFFFFF000011111111
~~~]GxjWHI 0000000000000000000000000000000CA1345  777711118888AAAAAAA22221111BBBB00
002222BBBBCCCC2222
~~~]P;]g0g 000000000000000000000000000000040DA3E4  4444FFFF444466663333EEEE88888888DD
DDEEEE44442222DDDD
~~~]KU|K<p 00000000000000000000000000000005E4A0AA  0000666655551111BBBB88889999AAAA55
550000333355557777
ubuntu@ip-172-31-12-176:~/64$
```

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



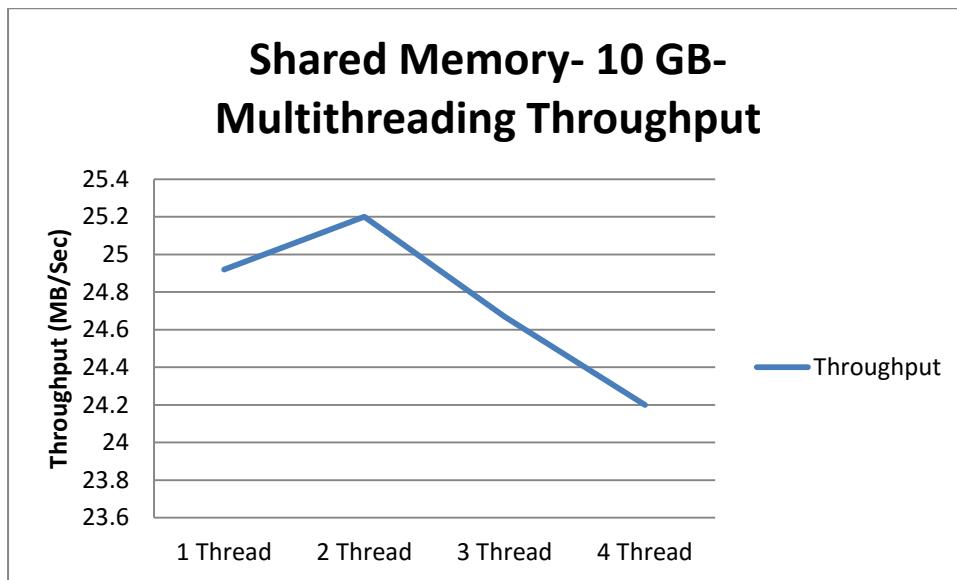
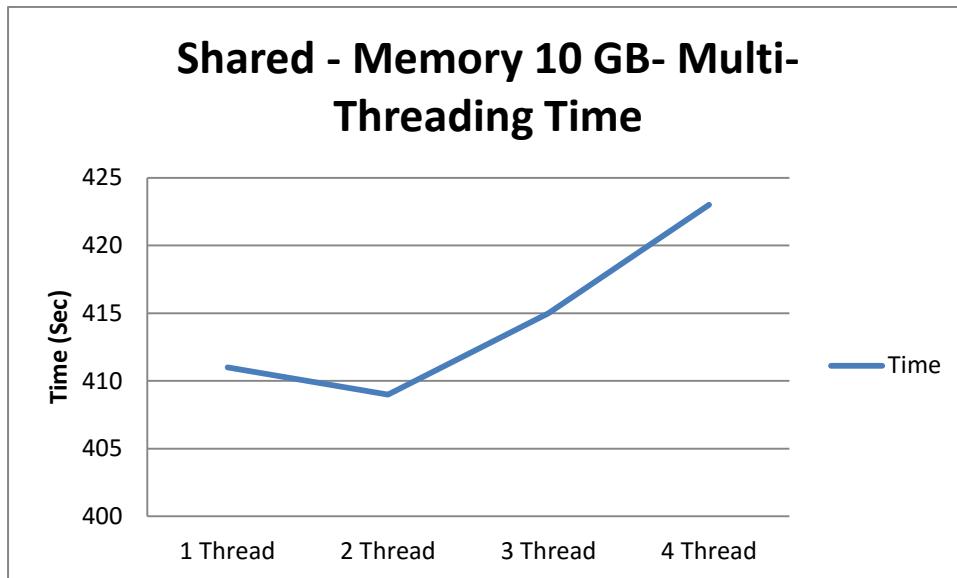
Through the observation of above graph, we can notice that the throughput to actually sort 1 GB and 10 GB using shared memory approach is almost at the constant level.

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?**

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:



```
ubuntu@ip-172-31-12-107: ~
ubuntu@ip-172-31-12-107:~$ ls
cloud.pem  hadoop-2.7.2  hadoop-2.7.2.tar.gz  Install_Hadoop_Ec2.sh
ubuntu@ip-172-31-12-107:~$
```

## D2.Xlarge 1 GB:

Hadoop 1 GB 1 Node Time required to sort:

The screenshot shows the EC2 Management Console's "All Applications" page. At the top, there is a summary table for memory usage:

Allocated	Memory Used	Memory Total	Memory Reserved	VCPUs Used	VCPUs Total	VCPUs Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
0 B	8 GB	0 B	0	8	0	1	0	0	0	0	0

Below this, there are sections for "Scheduling Resource Type" showing allocation ranges (<memory:1024, vCores:1> to <memory:8192, vCores:8>) and a search bar. A table lists completed jobs:

ID	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
1	MAPREDUCE	default	Thu Mar 10 20:37:57 -0600 2016	Thu Mar 10 20:39:23 -0600 2016	FINISHED	SUCCEEDED	N/A	History	N/A

At the bottom, there are navigation links for "First", "Previous", "1", "Next", and "Last".

## Hadoop 16 Nodes 1 GB:

```
ubuntu@ip-172-31-15-244: ~/hadoop-2.7.2/sbin
ec2-52-87-174-236.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-3-101.out
ec2-52-91-99-158.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-7-80.out
ec2-54-165-91-219.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-13-204.out
ec2-54-85-4-152.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-9-244.out
ec2-52-90-95-47.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-1-49.out
ec2-52-87-172-17.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-12-233.out
ec2-54-85-192-253.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-15-244.out
ec2-54-174-18-101.compute-1.amazonaws.com: starting nodemanager, logging to /home/ubuntu/hadoop-2.7.2/logs/yarn-ubuntu-nodemanager-ip-172-31-4-147.out
ubuntu@ip-172-31-15-244:~/hadoop-2.7.2/sbin$ jps
12730 SecondaryNameNode
12324 NameNode
12887 ResourceManager
13358 Jps
12523 DataNode
13064 NodeManager
ubuntu@ip-172-31-15-244:~/hadoop-2.7.2/sbin$
```

## 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

\* Find: 52.91.174   ◀ Previous   ▶ Next   👉 Highlight all    Match case

## Hadoop 16 Nodes Time required sorting 1 GB:

All Applications											
Running		Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
		0 B	128 GB	0 B	0	128	0	16	0	0	0
Using Resource Type											
<memory:1024, vCores:1>											
Search:											
Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Block Node			
MAPREDUCE	default	Fri Mar 11 21:26:10 -0600 2016	Fri Mar 11 21:26:49 -0600 2016	FINISHED	SUCCEEDED	N/A	History	N/A			

First Previous 1 Next

\* Find: 52.91.174   ◀ Previous   ▶ Next   👉 Highlight all    Match case

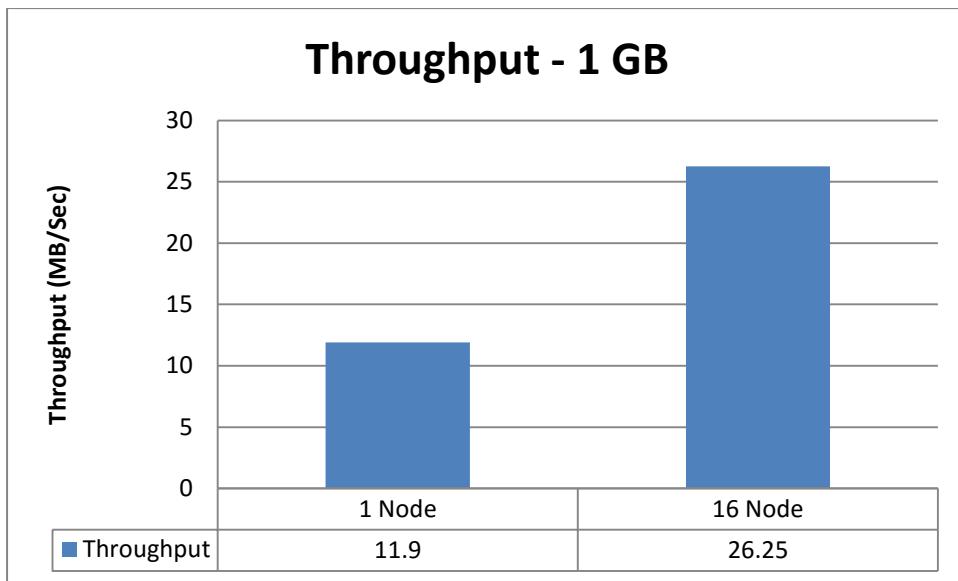
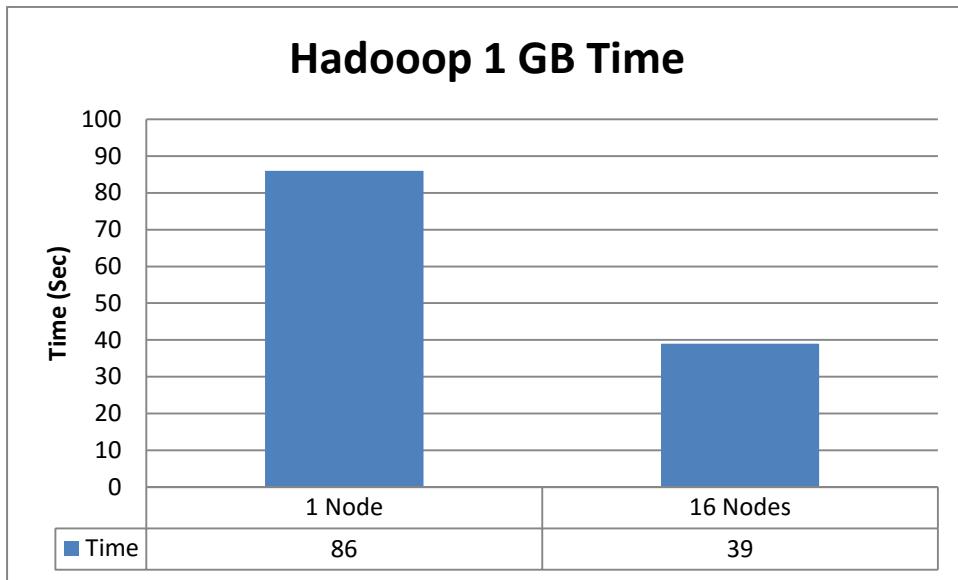
### First 10 Lines of sorted Output:

```
ubuntu@ip-172-31-15-244: /vinit/Output1Gb
Gb
ubuntu@ip-172-31-15-244:~/hadoop-2.7.2$ cd /vinit/Output1Gb/
ubuntu@ip-172-31-15-244:/vinit/Output1Gb$ head -10 part-r-00000
    "O!uve  000000000000000000000000000000001228D4  77778888000022224444DDDDDDDEEEE00
000000CCCC7777DDDD
    ,K4a-:v  000000000000000000000000000000001B8132  5555EEEE888899994444FFFF1111CCCCEE
EE1111EEEE6666FFFF
    .FuD\}u  00000000000000000000000000000000797631  5555DDDDBBBBB0000777722221111222244
44DDDDDDDD99996666
    ;5YThct  000000000000000000000000000000007D3DF5  2222AAAACCCCFFFFAAAA44445555EEEE44
442222DDDD99992222
    =2G^9{-  00000000000000000000000000000000809EE  5555DDDD1111CCCC9999BBBBB0000BBBBCC
CCFFFFCCCC44443333
    N}M9?sP  00000000000000000000000000000000429597  5555FFFF00007777555599991111CCCC66
669999AAAAEEE8888
    P0X?Rs&  0000000000000000000000000000000041162E  888833339999FFFF1111CCCC8888CCCC99
99EEEEDDDD00003333
    [Xq\\$%  0000000000000000000000000000000097A5F0  66666666EEEEDDDD7777FFFF00005555FF
FFFFFFFFFF888855551111
    rAmQg4v  000000000000000000000000000000008180D3  BBBB11111119999FFFFFFFFFFFF44
44BBBBB88884444CCCC
    !&))Jf3;  000000000000000000000000000000004602E1  4444FFFFCCCC88888888CCCCFFFFCCCCEE
EE5555666666666666
ubuntu@ip-172-31-15-244:/vinit/Output1Gb$
```

### Last 10 Lines of sorted output:

```
ubuntu@ip-172-31-15-244: /vinit/Output1Gb$ ^C
ubuntu@ip-172-31-15-244: /vinit/Output1Gb$ ^C
ubuntu@ip-172-31-15-244:/vinit/Output1Gb$ tail -10 part-r-00000
~~~%A NB_t  000000000000000000000000000000003DE5EC  FFFF7777EEEEBBBB4444EEEEEEE333399
99DDDD999900005555
~~~-c-CQ(>  00000000000000000000000000000000832611  9999FFFF11117773337770000111144
4444440000BBBB6666
~~~8Y}Fql*  00000000000000000000000000000000742A4C  BBBB1111CCCCEEE888800000000777733
333333DDDD22225555
~~->Dd=QT]  00000000000000000000000000000000674C0F  9999DDDD000055556666CCCC22220000FF
FFEEEEDDDDFFFF0000
~~~]JA({}j$  0000000000000000000000000000000060BCBE  1111222233344445559999AAAABBBB99
99FFFFDDDBBBB3333
~~~]Zp.#/+  000000000000000000000000000000003B9A5A  CCCC8888EEEEAAAEEE3333333777700
00FFFFCCCC66667777
~~~_jQepix  0000000000000000000000000000000011E5D4  111199991115555BBBB11100002222EE
EE6666BBBB7777DDDD
~~~nt=ZH[N  00000000000000000000000000000000332A13  44441111BBBBBBBBB3337777FFFF444455
5555553330000CCCC
~~~s/Pq,-E  000000000000000000000000000000006BE930  2222DDDDDDDD77771111EEEECCCC7777BB
BB4444888811111111
~~~zbA_Tt  000000000000000000000000000000007F9F4F  BBBBCCCC666655559999FFFF8888AAA11
116666AAAABBBB0000
ubuntu@ip-172-31-15-244:/vinit/Output1Gb$
```

**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:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1458860176316_0001	ubuntu	Hadoop_Sort	MAPREDUCE	default	Thu Mar 24 18:07:02 -0500 2016	Thu Mar 24 18:31:07 -0500 2016	FINISHED	SUCCEEDED		History

Valsort:

```
Vinit Shah
ubuntu@ip-172-31-7-63:~/64$ ./valsrt part-r-00000
Records: 100000000
Checksum: 2faf0ab746e89a8
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-7-63:~/64$
```

### First 10 Lines of output:

```
Vinit Shah
Bytes Written=100000000000
16/03/24 23:31:08 INFO terasort.TeraSort: done
ubuntu@ip-172-31-7-63:~/hadoop-2.7.2$ bin/hadoop fs -get output /home/ubuntu/output
ubuntu@ip-172-31-7-63:~/hadoop-2.7.2$ cd ..
ubuntu@ip-172-31-7-63:~$ cd output/
ubuntu@ip-172-31-7-63:~/output$ ls
_partition.lst part-r-00000 _SUCCESS
ubuntu@ip-172-31-7-63:~/output$ head -10 part-r-00000
    "O!uve 00000000000000000000000000001228D4 7777888000022224444DDDDDDDD
EEEE0000000CCCC7777DDDD
    PMd32= 00000000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB99993333
5555DDDDDDDD777788886666
    ^3CO], 0000000000000000000000000000158C5C5 5555AAAA9999EEEE888822229999
CCCCDDDD6666555544442222
    !&S3/] 00000000000000000000000000002145D78 8888BBBBDDDD1111CCCC55556666
BBBBB1111EEEEDDD22229999
    !,=U#,9 000000000000000000000000000019072E3 33332222FFFFBBBB0000FFFFAAAA
666655553333DDDD3333CCCC
    !0f[ ITd 00000000000000000000000000003CAAB4B 9999FFFF555533337777CCCC4444
BBBBB7777EEEEBBBBDDDD4444
    !f6Suy2 00000000000000000000000000003ABFD84 EEEE55555556666AAAA5555BBBB
DDDD0000111166660000DDDD
    #%NIPq. 00000000000000000000000000003B36FB9 1111000033334444111166666666
AAAAAAA00001111CCCCEE
    #'^cl'~ 00000000000000000000000000002EDC5C8 8888AAAA11114444FFFF77773333
EEEE44440000FFFF9999999
    $"-'Q)] 000000000000000000000000000005F1265D CCCC6666EEEE22220000DDDDAAAA
88886666BBBB00006666AAAA
ubuntu@ip-172-31-7-63:~/output$
```

### Last 10 Lines of output:

```
Vinit Shah
BBBBB7777EEEEBBBBDDDD4444
    !f6Suy2 00000000000000000000000000003ABFD84 EEEE55555556666AAAA5555BBBB
DDDD0000111166660000DDDD
    #%NIPq. 00000000000000000000000000003B36FB9 1111000033334444111166666666
AAAAAAA00001111CCCCEE
    #'^cl'~ 00000000000000000000000000002EDC5C8 8888AAAA11114444FFFF77773333
EEEE44440000FFFF9999999
    $"-'Q)] 000000000000000000000000000005F1265D CCCC6666EEEE22220000DDDDAAAA
88886666BBBB00006666AAAA
ubuntu@ip-172-31-7-63:~/output$ tail -10 part-r-00000
~~~uq2k#=U 00000000000000000000000000002C06745 99991111DDDD222211110000FFFF
EEEEFFFF33337777CCCC2222
~~~v/0&Qnm 00000000000000000000000000004709701 CCCC8888333FFFF000000000000
99991111FFFF777744446666
~~~yK0l:gE 00000000000000000000000000002048B4F CCCC11114444888822226666BBBB
888855557777EEEEBBBB0000
~~~yK^H.il 0000000000000000000000000000463D004 44440000FFFF3333999944447777
DDDDFFFFFAAAA11118888DDDD
~~~yL;C'XE 000000000000000000000000000005B0D211 2222EEEE3333000022221111CCCC
FFFF555577774444BBBB6666
~~~zbA_Tt 000000000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888
AAAA11116666AAAABB0000
~~~ze0^FEg 00000000000000000000000000000001E06130 4444CCCCBBBB9999222288885555
8888CCCCFFFF000011111111
~~~}GxjWHI 00000000000000000000000000000000CA1345 777711118888AAAAAAA22221111
BBBBB00002222BBBBCCCC2222
~~~}P;]g0g 000000000000000000000000000000040DA3E4 4444FFFF444466663333EEEE8888
8888DDDEEEE44442222DDDD
~~~}kU|K<p 00000000000000000000000000000005E4A0AA 0000666655551111BBBB88889999
AAAA55550000333355557777
ubuntu@ip-172-31-7-63:~/output$
```

### Map-Reduce Job success:

```
Vinit Shah
Map output materialized bytes=10400000450
Input split bytes=10875
Combine input records=0
Combine output records=0
Reduce input groups=1000000000
Reduce shuffle bytes=10400000450
Reduce input records=1000000000
Reduce output records=1000000000
Spilled Records=396636763
Shuffled Maps =75
Failed Shuffles=0
Merged Map outputs=75
GC time elapsed (ms)=16861
CPU time spent (ms)=973730
Physical memory (bytes) snapshot=21526769664
Virtual memory (bytes) snapshot=63132782592
Total committed heap usage (bytes)=15048114176
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=100000000000
File Output Format Counters
Bytes Written=100000000000
16/03/24 23:31:08 INFO terasort.TeraSort: done
ubuntu@ip-172-31-7-63:~/hadoop-2.7.2$
```

### C3.large 100GB:

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

The screenshot shows the Hadoop cluster metrics and scheduler metrics in the YARN web interface.

**Cluster Metrics:**

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Vcores Used	Vcores Total	Vcores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	1	0	0 B	51 GB	0 B	0	136	0	17	0	0	0	0

**Scheduler Metrics:**

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:3072, vCores:8>

**All Applications:**

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1459107905454_0001	ubuntu	Hadoop_Sort	MAPREDUCE	default	Sun Mar 27 14:59:02 -0500 2016	Sun Mar 27 15:21:59 -0500 2016	FINISHED	SUCCEEDED	N/A	History	N/A

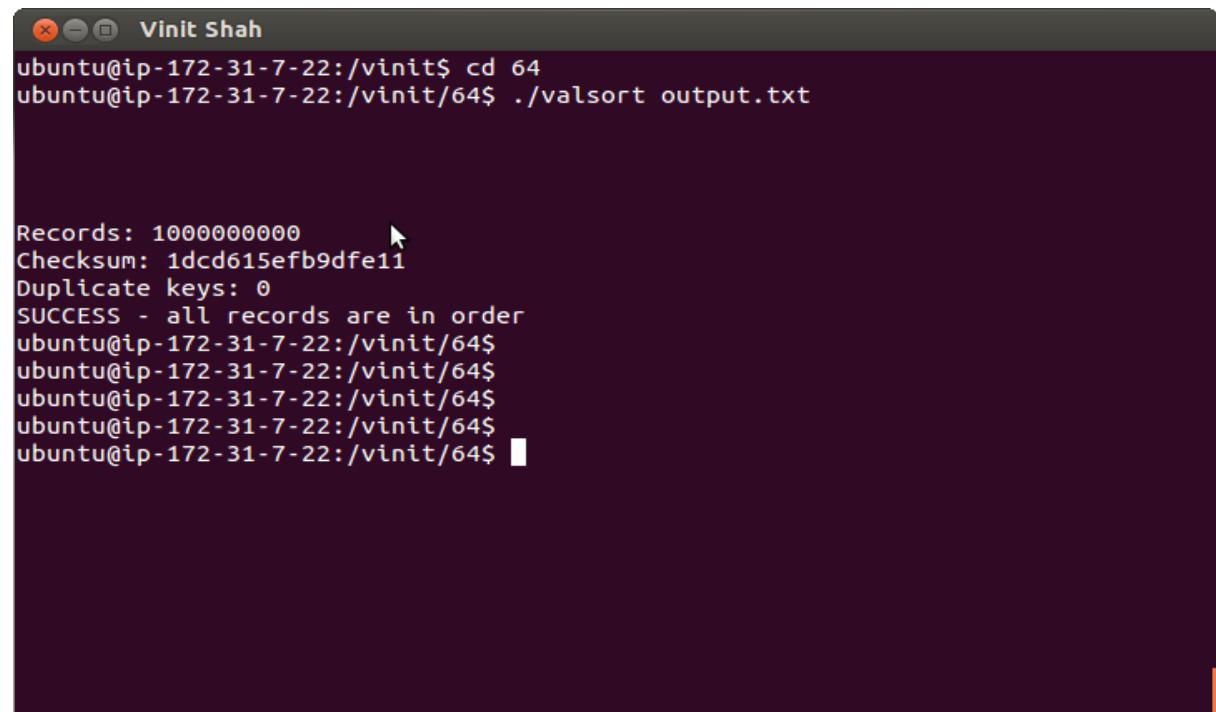
## 16 Nodes Cluster Setup:

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
ec2-52-90-0-197.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 (172.31.31.14.175:50010)	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

## **100 GB JOB sorted:**

The screenshot shows the Hadoop Application Overview page for application ID `application_1459107905454_0001`. The top navigation bar includes tabs for EC2 Management, yarn and mapred, Namenode information, MapReduce Applications, and Application application. The URL is `ec2-54-165-72-19.compute-1.amazonaws.com:9064/clusters/cluster1/applications/application_1459107905454_0001`. A sidebar on the left lists cluster management options like About, Nodes, Node Labels, Applications (with categories NEW, SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), and Scheduler. Below this is a Kill Application button. The main content area displays application details: User: ubuntu, Name: Hadoop.Sort, Application Type: MAPREDUCE, Application Tags: YarnApplicationState: FINISHED, FinalStatus reported by AM: SUCCEEDED, Started: Sun Mar 27 19:59:02 +0000 2016, Elapsed: 22mins, 56sec, Tracking URL: History, and Diagnostics. Below this is an Application Metrics section with various resource preemption statistics. At the bottom is a table listing application attempts, with one entry shown: Attempt ID `appattempt_1459107905454_0001_000001`, Started `Sun Mar 27 14:59:02 -0500 2016`, Node `http://ec2-54-86-252-221.compute-1.amazonaws.com:8042`, Logs `N/A`, and Blacklisted Nodes `N/A`. The footer shows pagination from 1 to 1.

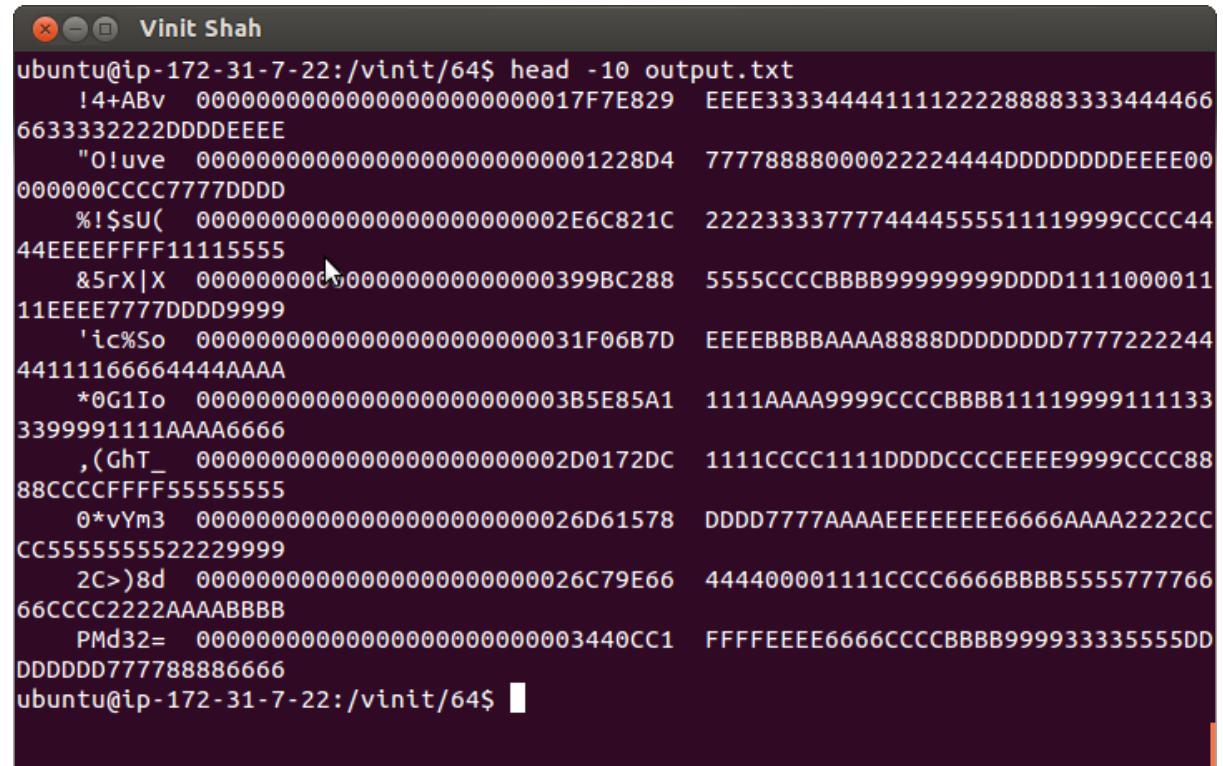
**Valsort:**



```
ubuntu@ip-172-31-7-22:/vinit$ cd 64
ubuntu@ip-172-31-7-22:/vinit/64$ ./valsrt output.txt

Records: 1000000000
Checksum: 1dc615efb9dfe11
Duplicate keys: 0
SUCCESS - all records are in order
ubuntu@ip-172-31-7-22:/vinit/64$
```

**First 10 Lines of output:**



```
ubuntu@ip-172-31-7-22:/vinit/64$ head -10 output.txt
!4+ABv 0000000000000000000000000000000017F7E829 EEEE333344441111222288883333444466
6633332222DDDEEE
"O!uve 000000000000000000000000000000001228D4 77778888000022224444DDDDDDDEEE00
000000CCCC7777DDDD
%!$sU( 000000000000000000000000000000002E6C821C 2222333377774444555511119999CCCC44
44EEEEFFFF11115555
&5rX|X 0000000000000000000000000000399BC288 5555CCCCBBBB99999999DDDD1111000011
11EEEE7777DDDD9999
'ic%So 0000000000000000000000000000000031F06B7D EEEEBBBBAAAA8888DDDDDD7777222244
44111166664444AAAA
*0G1Io 000000000000000000000000000000003B5E85A1 1111AAAA9999CCCCBBBB11119999111133
3399991111AAAA6666
,(GhT_ 000000000000000000000000000000002D0172DC 1111CCCC1111DDDCCCCEEEE9999CCCC88
88CCCCFFFF55555555
0*vYm3 0000000000000000000000000000000026D61578 DDDD7777AAAAEEEEEE6666AAAA2222CC
CC5555555522229999
2C>)8d 0000000000000000000000000000000026C79E66 444400001111CCCC6666BBBB5555777766
66CCCC2222AAAABB
PMd32= 000000000000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB999933335555DD
DDDDDD777788886666
ubuntu@ip-172-31-7-22:/vinit/64$
```

### Last 10 Lines of Output:

```
Vinit Shah
PMd32= 000000000000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB99993335555DD
DDDDDD777788886666
ubuntu@ip-172-31-7-22:/vinit/64$ tail -10 output.txt
~~~~#iay1X 00000000000000000000000000000025D35EDF 6666AAAA55559999777700002222333388
88FFFF999922220000
~~~~+@p){@ 00000000000000000000000000000085426F4 777733355511111110000CCCC555599
99AAAA7777DDDDDD
~~~~,R^_?n 00000000000000000000000000001034E347 111111119999000011118888AAAA555544
44EEEE99993333888
~~~~.Ey`^) 000000000000000000000000000016F0E66B CCCC6666DDDD2222DDDD111188889999EE
EEEEEEEEEEBBBB4444
~~~~4!kA7x 00000000000000000000000000001F1A1E26 EEEE777711117777BBBB1111EEEE888844
44DDDDDDDEEEEEEBBBB
~~~~8Ii/*!@ 00000000000000000000000000001F05932F 11119999BBBB44447777000011114444CC
CCAAAA6666DDDD0000
~~~~<I'5>F 0000000000000000000000000000008CB2293 88883333BBBB1111666699998888555588
88888822228888CCCC
~~~~G-)m^) 000000000000000000000000000013397F73 DDDDFFFFBBBBCCCCFFFF44446666AAAA11
1133333333AAAACCCC
~~~~c+I&CP 00000000000000000000000000000074BDF64 8888000055550000DDDD22227777AAAA00
0033332222AAAADDDD
~~~~hb&5X* 000000000000000000000000000032C0E06B 7777BBBBBBBB9999EEEEAAAAAAA0000CC
CCDDDD4444BBBB4444
ubuntu@ip-172-31-7-22:/vinit/64$
```

### Name Node Health Page:

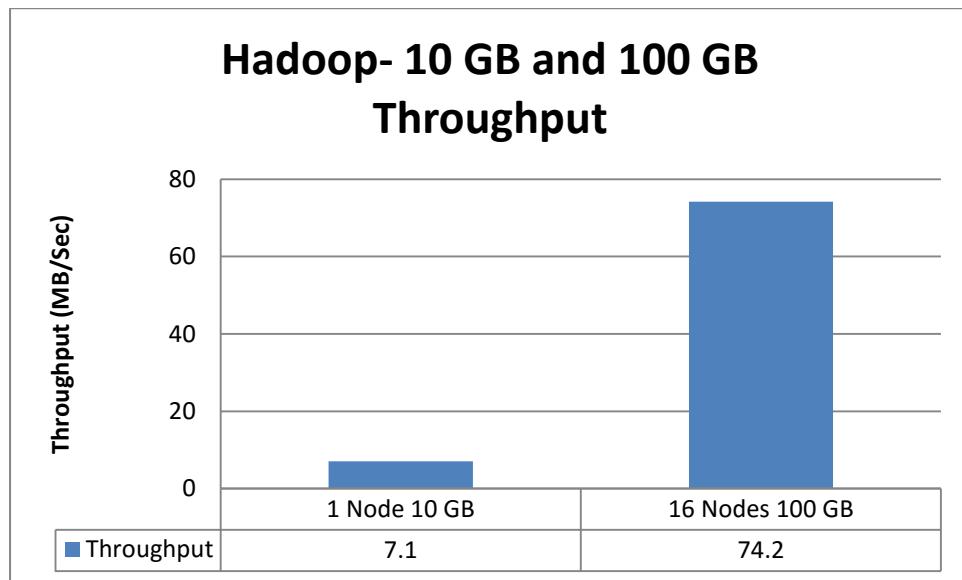
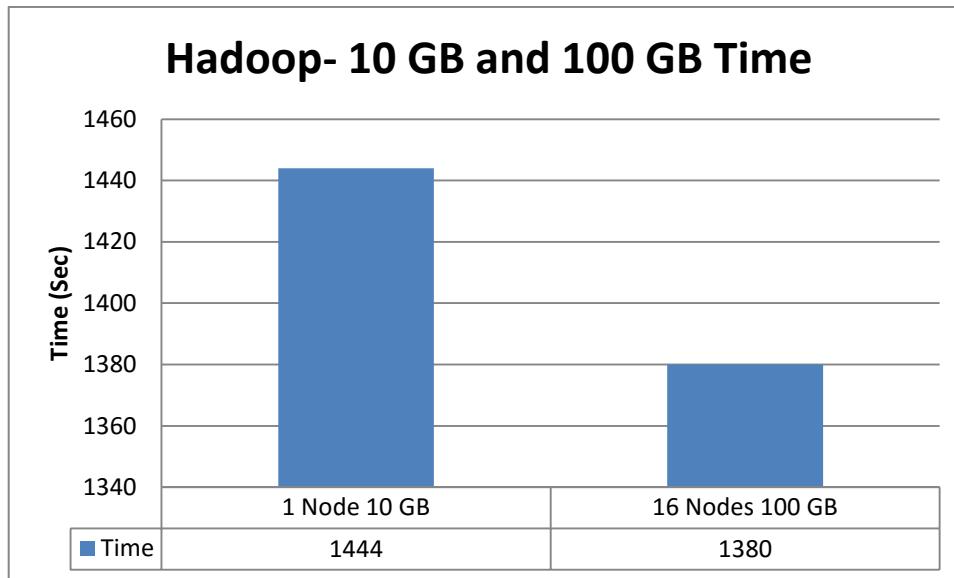
The screenshot shows a web-based monitoring interface for a NameNode. At the top, there are three tabs: 'EC2 Management C...', 'yarn and mapred - vi...', and 'Namenode information'. The current view is under the 'Namenode information' tab. Below the tabs, a message states: 'Non Heap Memory used 33.23 MB of 33.94 MB Committed Non Heap Memory. Max Non Heap Memory is 214 MB.' A large table follows, displaying various metrics:

Configured Capacity:	6.95 TB
DFS Used:	35.15 GB (0.49%)
Non DFS Used:	465.8 GB
DFS Remaining:	6.46 TB (92.96%)
Block Pool Used:	35.15 GB (0.49%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.63% / 0.15%
Live Nodes	17 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion	0
Block Deletion Start Time	3/27/2016, 2:44:49 PM

### NameNode Journal Status

Current transaction ID: 053

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:**

```
vinit@vinit-VirtualBox: ~/Spark/spark-1.6.0-bin-hadoop2.6/ec2
(TID 16) in 9285 ms on ip-172-31-49-155.ec2.internal (3/8)
16/03/13 06:48:37 INFO scheduler.TaskSetManager: Starting task 7.0 in stage 2.0
(TID 23, ip-172-31-49-155.ec2.internal, partition 7, NODE_LOCAL, 1894 bytes)
16/03/13 06:48:37 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 2.0
(TID 18) in 10913 ms on ip-172-31-49-155.ec2.internal (4/8)
16/03/13 06:48:44 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 2.0
(TID 20) in 10614 ms on ip-172-31-49-155.ec2.internal (5/8)
16/03/13 06:48:45 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 2.0
(TID 22) in 9992 ms on ip-172-31-49-155.ec2.internal (6/8)
16/03/13 06:48:45 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 2.0
(TID 21) in 11046 ms on ip-172-31-49-155.ec2.internal (7/8)
16/03/13 06:48:45 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 2.0
(TID 23) in 8909 ms on ip-172-31-49-155.ec2.internal (8/8)
16/03/13 06:48:45 INFO scheduler.DAGScheduler: ResultStage 2 (saveAsTextFile at
<console>:30) finished in 19.822 s
16/03/13 06:48:45 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose t
asks have all completed, from pool
16/03/13 06:48:45 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at
<console>:30, took 30.140582 s
t1: Long = 2311361236276
Elapsed time: 37664472416ns
```

scala> Time:

## Spark 1 GB 1 Node Running:

The screenshot shows the EC2 Management Console interface with a tab for the Spark Master at spark://ec2-54-208-7-187.compute-1.amazonaws.com:8080. The main content area displays the Spark 1.6.0 master interface. It shows the URL as spark://ec2-54-208-7-187.compute-1.amazonaws.com:7077 and the REST URL as spark://ec2-54-208-7-187.compute-1.amazonaws.com:6066 (cluster mode). The status indicates 1 Alive Worker with 4 cores and 28.7 GB memory used. One application, "Spark shell", is listed as running. The "Completed Applications" section is empty.

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20160313062333-172.31.49.155-36531	172.31.49.155:36531	ALIVE	4 (4 Used)	28.7 GB (26.7 GB Used)

**Running Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160313064757-0000	(kill) Spark shell	4	26.7 GB	2016/03/13 06:47:57	root	RUNNING	2.1 min

**Completed Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

## Spark 1 GB 1 Node sort Complete:

The screenshot shows the EC2 Management Console interface with a tab for the Spark Master at spark://ec2-54-208-7-187.compute-1.amazonaws.com:8080. The main content area displays the Spark 1.6.0 master interface. It shows the URL as spark://ec2-54-208-7-187.compute-1.amazonaws.com:7077 and the REST URL as spark://ec2-54-208-7-187.compute-1.amazonaws.com:6066 (cluster mode). The status indicates 1 Alive Worker with 4 cores and 28.7 GB memory used. One application, "Spark shell", is listed as completed. The "Completed Applications" section shows the completed application.

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20160313062333-172.31.49.155-36531	172.31.49.155:36531	ALIVE	4 (0 Used)	28.7 GB (0.0 B Used)

**Running Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration

**Completed Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20160313064757-0000	Spark shell	4	26.7 GB	2016/03/13 06:47:57	root	FINISHED	2.7 min

## Spark 1 GB 16 Nodes Time for Sort:

```
vinit@vinit-VirtualBox: ~/Spark/spark-1.6.0-bin-hadoop2.6/ec2
(TID 17) in 6068 ms on ip-172-31-36-43.ec2.internal (2/8)
16/03/13 21:45:50 INFO scheduler.TaskSetManager: Finished task 6.0 in stage 2.0
(TID 22) in 7029 ms on ip-172-31-36-78.ec2.internal (3/8)
16/03/13 21:45:50 INFO scheduler.TaskSetManager: Finished task 7.0 in stage 2.0
(TID 23) in 7030 ms on ip-172-31-42-56.ec2.internal (4/8)
16/03/13 21:45:51 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2.0
(TID 16) in 8057 ms on ip-172-31-46-81.ec2.internal (5/8)
16/03/13 21:45:51 INFO scheduler.TaskSetManager: Finished task 2.0 in stage 2.0
(TID 18) in 8103 ms on ip-172-31-40-150.ec2.internal (6/8)
16/03/13 21:45:51 INFO scheduler.TaskSetManager: Finished task 5.0 in stage 2.0
(TID 21) in 8318 ms on ip-172-31-37-60.ec2.internal (7/8)
16/03/13 21:45:51 INFO scheduler.TaskSetManager: Finished task 4.0 in stage 2.0
(TID 20) in 8360 ms on ip-172-31-47-224.ec2.internal (8/8)
16/03/13 21:45:51 INFO scheduler.DAGScheduler: ResultStage 2 (saveAsTextFile at <console>:30) finished in 8.363 s
16/03/13 21:45:51 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/03/13 21:45:51 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at <console>:30, took 12.929590 s
t1: Long = 1906283286854
Elapsed time: 18251569545Ns

scala>
```

## Spark 1 GB 16 Nodes Configured Name Node:

The screenshot shows the EC2 Management Console with the Application Detail UI for a Spark shell application. The application ID is app-20160313214523-0000. The application name is "Spark shell". The configuration includes:

- ID: app-20160313214523-0000
- Name: Spark shell
- User: root
- CoreMemory: Unlimited (64 granted)
- Executor Memory: 25.7 GB
- Submit Date: Sun Mar 13 21:45:23 UTC 2016
- Status: RUNNING

Below the configuration, the "Executor Summary" table lists 16 workers, each with 4 cores and a memory of 27341 MB, all in a RUNNING state with logs directed to stdout/stderr.

ExecutorID	Worker	Cores	Memory	State	Logs
3	worker-20160313213127-172.31.42.56-40725	4	27341	RUNNING	stdout stderr
11	worker-20160313213128-172.31.36.78-56328	4	27341	RUNNING	stdout stderr
6	worker-20160313213128-172.31.47.224-42766	4	27341	RUNNING	stdout stderr
14	worker-20160313213127-172.31.34.182-43230	4	27341	RUNNING	stdout stderr
10	worker-20160313213127-172.31.35.46-45090	4	27341	RUNNING	stdout stderr
8	worker-20160313213127-172.31.34.221-59841	4	27341	RUNNING	stdout stderr
2	worker-20160313213127-172.31.43.10-33056	4	27341	RUNNING	stdout stderr
15	worker-20160313213127-172.31.36.43-41639	4	27341	RUNNING	stdout stderr
1	worker-20160313213127-172.31.37.60-43122	4	27341	RUNNING	stdout stderr
9	worker-20160313213127-172.31.33.156-39794	4	27341	RUNNING	stdout stderr
5	worker-20160313213127-172.31.35.12-45632	4	27341	RUNNING	stdout stderr
0	worker-20160313213127-172.31.40.31-44995	4	27341	RUNNING	stdout stderr
12	worker-20160313213127-172.31.46.81-39464	4	27341	RUNNING	stdout stderr
7	worker-20160313213127-172.31.42.71-44467	4	27341	RUNNING	stdout stderr
13	worker-20160313213127-172.31.43.194-46674	4	27341	RUNNING	stdout stderr
4	worker-20160313213127-172.31.40.190-56456	4	27341	RUNNING	stdout stderr

### Spark 1 GB First 10 Lines of Sorted Output:

```
vinit@vinit-VirtualBox: ~/Spark/spark-1.6.0-bin-hadoop2.6/ec2
root@ip-172-31-38-156 out1]$ head -10 part-00000
    "0!uve 000000000000000000000000000000001228D4 7777888800002224444DDDDDDDEEE0
0000000CCCC7777DDDD
    ,K4a-:v 000000000000000000000000000000001B8132 5555EEEE888899994444FFFF1111CCCCE
EEE1111EEEE6666FFFF
    .FuD\}u 00000000000000000000000000000000797631 5555DDDBBBB000077722211122224
444DDDDDDDD99996666
    ;5YThct 000000000000000000000000000000007D3DF5 2222AAAACCCFFFFAAAA44445555EEEE4
44422222DDDD99992222
    =2G^9{- 00000000000000000000000000000000809EE 5555DDDD1111CCCC9999BBBB0000BBBBCC
CCCCFFFCCCC44443333
    N}M9?sP 00000000000000000000000000000000429597 5555FFFF0000777555599991111CCCC6
6669999AAAEEEE8888
    P0X?Rs& 0000000000000000000000000000000041162E 888833339999FFFF1111CCCC8888CCCC9
999EEEEDDDD00003333
    [Xq\\$% 0000000000000000000000000000000097A5F0 66666666EEEEDDD7777FFFF00005555F
FFFFFFFFFF888855551111
    rAmQg4v 000000000000000000000000000000008180D3 BBBB11111119999FFFFFFFFFF4
444BBBBB88884444CCCC
    !&))Jf3; 000000000000000000000000000000004602E1 4444FFFFCCCC8888888CCCCFFFFCCCC
EEE5555666666666666
root@ip-172-31-38-156 out1]$
```

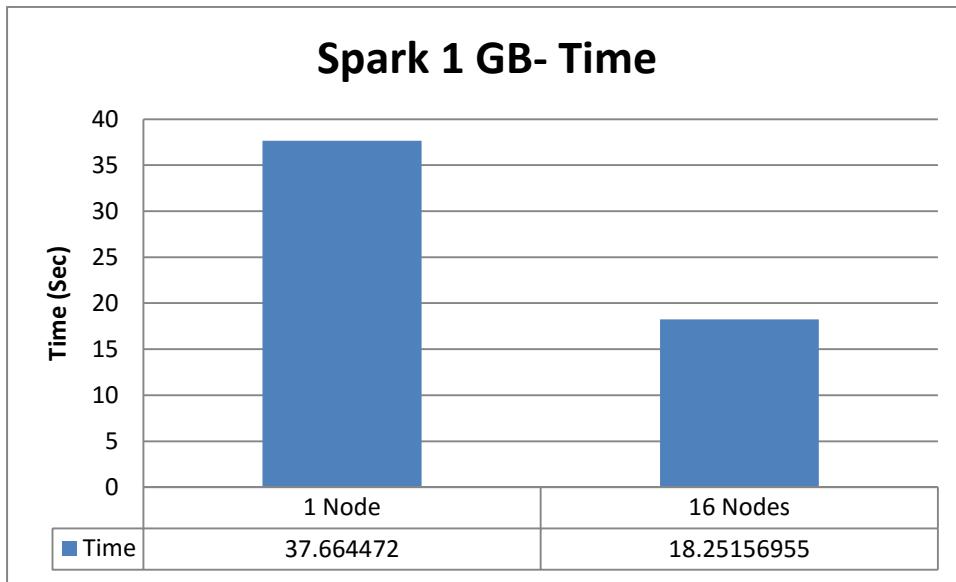
### Spark 1 GB Last 10 Lines of sorted output:

```
root@ip-172-31-38-156:/vinit/output/out1
part-00000  part-00002  part-00004  part-00006 _SUCCESS
part-00001  part-00003  part-00005  part-00007
root@ip-172-31-38-156 out1]$ tail -10 part-00007
~~~%A NB_t 000000000000000000000000000003DE5EC FFFF7777EEEEBBBB4444EEEEEEE33339
999DDDD999900005555
~~~c-CQ(> 00000000000000000000000000000832611 9999FFFF11117773337777000011114
44444440000BBBB6666
~~~8Y}Fql* 00000000000000000000000000000742A4C BBBB1111CCCCEEE88880000000077773
3333333DDDD22225555
~~~>Dd=QT] 00000000000000000000000000000674C0F 9999DDDD000055556666CCCC22220000F
FFFEFEEDDDDFFFF0000
~~~]JA(}j$ 0000000000000000000000000000000060BCBE 111122223333444455559999AAAABBBB9
999FFFFDDDBBBB3333
~~~]Zp.#/+ 000000000000000000000000000000003B9A5A CCCC8888EEEEAAAAEEE333333377770
000FFFFCCCC66667777
~~~_jQepix 0000000000000000000000000000000011E5D4 1111999911115555BBBB111100002222E
EEE6666BBBB7777DDDD
~~~nt=ZH[N 00000000000000000000000000000000332A13 4444111BBBBBBBB33337777FFFF44445
5555553330000CCCC
~~~s/Pq,-E 000000000000000000000000000000006BE930 2222DDDDDD77771111EEECCCC7777B
BBBB444488881111111
~~~zbA_Tt 000000000000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA1
1116666AAAABBBB0000
root@ip-172-31-38-156 out1]$
```

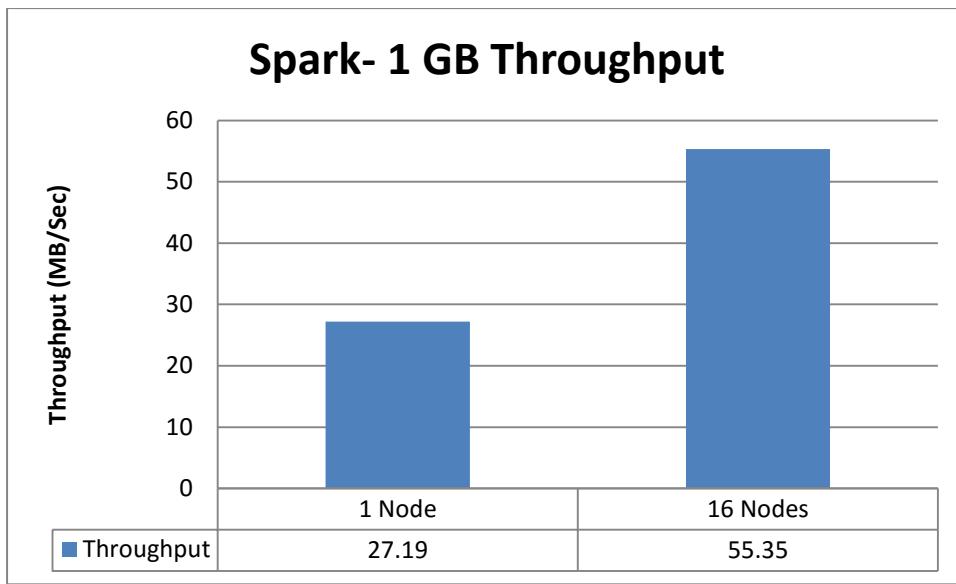
**Valsort of Sorted output:**

```
root@ip-172-31-38-156:/vinit/64
root@ip-172-31-38-156 64]$ ./valsrt output.txt
Records: 10000000
Checksum: 4c4b60f300315d
Duplicate keys: 0
SUCCESS - all records are in order
root@ip-172-31-38-156 64]$
```

**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 ip-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, from pool
16/03/25 02:20:13 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at <console>:30, took 662.523950 s
t1: Long = 3812300433297
Elapsed time: 829036970881ns
```

scala> █

Valsort:

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

## Spark Job:

The screenshot shows the Spark Master UI at [spark://ec2-54-174-223-38.compute-1.amazonaws.com:7077](http://spark://ec2-54-174-223-38.compute-1.amazonaws.com:7077). The UI includes:

- Cluster Summary:** URL: [spark://ec2-54-174-223-38.compute-1.amazonaws.com:7077](http://spark://ec2-54-174-223-38.compute-1.amazonaws.com:7077), REST URL: [spark://ec2-54-174-223-38.compute-1.amazonaws.com:6066](http://spark://ec2-54-174-223-38.compute-1.amazonaws.com:6066) (cluster mode), Alive Workers: 1, Cores in use: 2 Total, 2 Used, Memory in use: 2.7 GB Total, 2.4 GB Used, Applications: 1 Running, 0 Completed, Drivers: 0 Running, 0 Completed, Status: ALIVE.
- Workers:** A table showing one worker: worker-20160325012907-172.31.8.254-37643, Address: 172.31.8.254:37643, State: ALIVE, Cores: 2 (2 Used), Memory: 2.7 GB (2.4 GB Used).
- Running Applications:** A table showing one application: app-20160325020611-0000, Name: Spark shell, Cores: 2, Memory per Node: 2.4 GB, Submitted Time: 2016/03/25 02:06:11, User: root, State: RUNNING, Duration: 15 min.
- Completed Applications:** An empty table.

## 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
    "Oluve 000000000000000000000000000000001228D4 7777888000022224444DDDDDDDEEE00000000CCCC7777DDDD
PMD32= 0000000000000000000000000000003440CC1 FFFFEEEE6666CCCCBBBB99993335555DDDDDD77778886666
^3C0], 000000000000000000000000000000158C5C5 5555AAA9999EEE88882229999CCCCDDDD6666555544442222
!&S3/] 0000000000000000000000000000002145D78 8888BBBBDDDD1111CCCC55556666BBBB1111EEEEEDDD22229999
!,=U#,9 00000000000000000000000000000019072E3 3333222FFFFBBBBB0000FFFAAA6666555333DDDD3333CCCC
!0f[Itd 0000000000000000000000000000003CAAB4B 9999FFFF555533337777CCCC4444BBBB7777EEEEBBBBDDDD4444
!f6Suy2 0000000000000000000000000000003ABFD84 EEEE55555556666AAA5555BBBBDDDD00001111666600000DDDD
#%NIpq. 0000000000000000000000000000003B36FB9 1111000033334444111166666666AAAAAAA00001111CCCCEEEE
#'^c[l'~ 0000000000000000000000000000002EDC5C8 8888AAA11114444FFFF77773333EEEE44440000FFFF99999999
$"-Q)] 0000000000000000000000000000005F1265D CCCC6666EEE22220000DDDAAAA88886666BBBB00006666AAA
root@ip-172-31-0-134 output]$
```

## 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 part-00066 _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 00000000000000000000000000000002C06745 99991111DDDD222211110000FFFFEEEEFFFF33337777CCCC2222
~~~v/0&Qnm 0000000000000000000000000000004709701 CCCC88883333FFF00000000000099991111FFF777744446666
~~~yK0l:gE 0000000000000000000000000000002048B4F CCCC11114444888822226666BBBB888855557777EEEEBBBB0000
~~~yK^H.il 000000000000000000000000000000463D004 444400000FFF3333999944447777DDDFFFAAAAA11118888DDDD
~~~yL;C'XE 0000000000000000000000000000005B0D211 2222EEEE3333000022221111CCCCFFFF555577774444BBBB6666
~~~zbA_Tt 0000000000000000000000000000007F9F4F BBBBCCCC666655559999FFFF8888AAAA11116666AAAABBBB0000
~~~ze0^FEg 0000000000000000000000000000001e06130 4444CCCCBBBB999922288885558888CCCCFFF000011111111
~~~}GxjWHI 0000000000000000000000000000000CA1345 777711118888AAAAAAA22221111BBBB00002222BBBBCCC2222
~~~}P;]g0g 00000000000000000000000000000040DA3E4 4444FFF444466663333EEEE88888888DDDEEEE44442222DDDD
~~~}kU|K<p 00000000000000000000000000000005E4A0AA 0000666655551111BBBB88889999AAAA55550000333355557777
root@ip-172-31-0-134 output]$
```

## C3.large 100 GB:

### 16 Nodes Cluster of Spark:

The screenshot shows the EC2 Management Console with the following details:

- URL:** spark://ec2-52-207-217-241.compute-1.amazonaws.com:7077
- REST URL:** spark://ec2-52-207-217-241.compute-1.amazonaws.com:6066 (cluster mode)
- Alive Workers:** 16
- Cores in use:** 32 Total, 0 Used
- Memory in use:** 42.7 GB Total, 0.0 B Used
- Applications:** 0 Running, 0 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20160326143649-172.31.17.233-50657	172.31.17.233:50657	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.18.1-33315	172.31.18.1:33315	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.19.72-53054	172.31.19.72:53054	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.21.244-37891	172.31.21.244:37891	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.21.32-60482	172.31.21.32:60482	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.22.53-58009	172.31.22.53:58009	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.23.131-37020	172.31.23.131:37020	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.25.147-44011	172.31.25.147:44011	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.25.82-53672	172.31.25.82:53672	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.27.183-48543	172.31.27.183:48543	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.27.222-34339	172.31.27.222:34339	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.27.236-37498	172.31.27.236:37498	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.29.212-34244	172.31.29.212:34244	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.30.255-40790	172.31.30.255:40790	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.31.139-52372	172.31.31.139:52372	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)
worker-20160326143649-172.31.31.233-59278	172.31.31.233:59278	ALIVE	2 (0 Used)	2.7 GB (0.0 B Used)

**Running Applications**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

## Time required

```
Vinit Shah
0 (TID 2232) in 148169 ms on ip-172-31-29-212.ec2.internal (739/745)
16/03/26 16:49:32 INFO scheduler.TaskSetManager: Finished task 737.0 in stage 2.
0 (TID 2227) in 156396 ms on ip-172-31-21-244.ec2.internal (740/745)
16/03/26 16:49:44 INFO scheduler.TaskSetManager: Finished task 739.0 in stage 2.
0 (TID 2229) in 167586 ms on ip-172-31-25-82.ec2.internal (741/745)
16/03/26 16:49:46 INFO scheduler.TaskSetManager: Finished task 744.0 in stage 2.
0 (TID 2234) in 167043 ms on ip-172-31-31-233.ec2.internal (742/745)
16/03/26 16:49:47 INFO scheduler.TaskSetManager: Finished task 731.0 in stage 2.
0 (TID 2221) in 173729 ms on ip-172-31-27-222.ec2.internal (743/745)
16/03/26 16:49:49 INFO scheduler.TaskSetManager: Finished task 734.0 in stage 2.
0 (TID 2224) in 173716 ms on ip-172-31-31-139.ec2.internal (744/745)
16/03/26 16:49:49 INFO scheduler.TaskSetManager: Finished task 740.0 in stage 2.
0 (TID 2230) in 171194 ms on ip-172-31-31-233.ec2.internal (745/745)
16/03/26 16:49:49 INFO scheduler.DAGScheduler: ResultStage 2 (saveAsTextFile at
<console>:30) finished in 860.989 s
16/03/26 16:49:49 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose t
asks have all completed, from pool
16/03/26 16:49:49 INFO scheduler.DAGScheduler: Job 1 finished: saveAsTextFile at
<console>:30, took 1154.712702 s
t1: Long = 9051668807757
Elapsed time: 1209866800094ns
```

## Valsort:

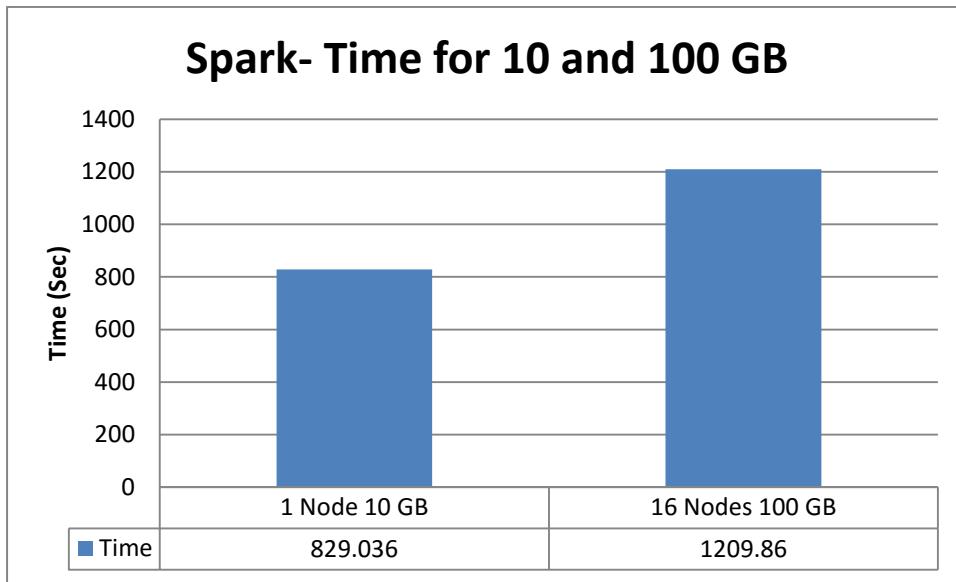
### First 10 Lines of Output:

```
Vinit Shah
part-00123 part-00248 part-00373 part-00498 part-00623
part-00124 part-00249 part-00374 part-00499 part-00624
root@ip-172-31-19-40 output]$ head -10 part-00000
    !4+ABv  0000000000000000000000000000000017F7E829  EEEE3333444411122228888333344446
66633332222DDDEEE
    "0!uve  000000000000000000000000000000001228D4  77778888000022224444DDDDDDDEEE0
0000000CCCC7777DDDD
    %!$sU(  000000000000000000000000000000002E6C821C  2222333377774444555511119999CCCC4
444EEEEFFFF11115555
    &5rX|X  00000000000000000000000000000003ABC288  5555CCCCBBBB99999999DDDD111100001
111EEEE7777DDDD9999
    'ic%So  0000000000000000000000000000000031F06B7D  EEEEBBBBAAA8888DDDDDD77772224
444111166664444AAAA
    *0G1Io  000000000000000000000000000000003B5E85A1  1111AAAA9999CCCCBBBB111199991113
33399991111AAAA6666
    ,(GhT_  000000000000000000000000000000002D0172DC  1111CCCC1111DDDDCCCCEE9999CCCC8
888CCCCFFFF55555555
    0*vYm3  0000000000000000000000000000000026D61578  DDDD7777AAAAEEEEEE6666AAAA2222C
CCC5555555522229999
    2C>)8d  0000000000000000000000000000000026C79E66  444400001111CCCC6666BBBB555577776
666CCCC2222AAABB
    PMd32=  000000000000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555D
DDDDDD777788886666
root@ip-172-31-19-40 output]$
```

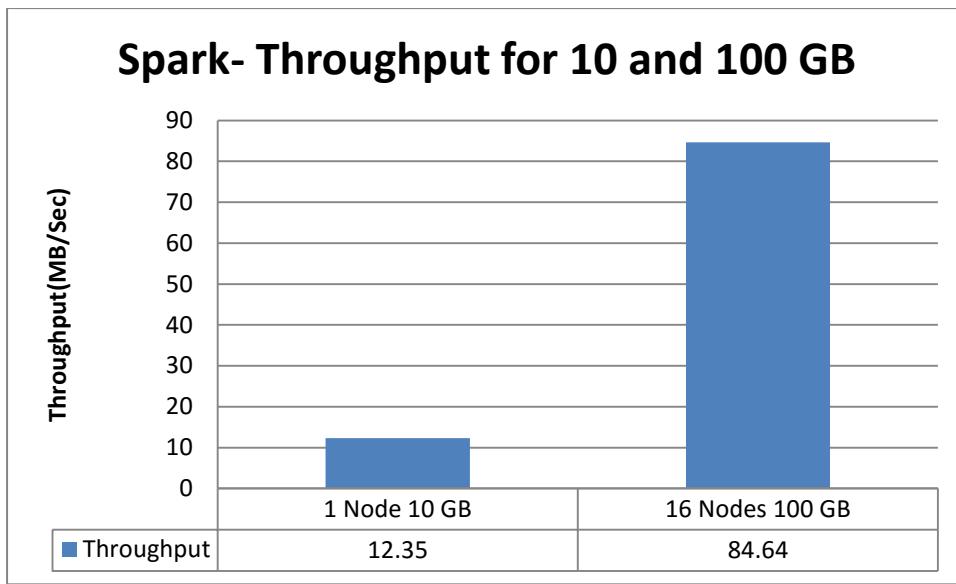
### Last 10 Lines of output:

```
Vinit Shah
PMd32=  000000000000000000000000000000003440CC1  FFFFEEEE6666CCCCBBBB999933335555D
DDDDDD777788886666
root@ip-172-31-19-40 output]$ tail -10 part-00744
~~~~#iay1X  0000000000000000000000000000000025D35EDF  6666AAAA555599997777000022223338
888FFFF999922220000
~~~~@p){@  0000000000000000000000000000000085426F4  7777333355551111110000CCCC55559
999AAAA7777DDDDDD
~~~~,R^_?n  000000000000000000000000000000001034E347  11111119999000011118888AAAA55554
444EEEE999933338888
~~~~.Ey`^)  0000000000000000000000000000000016F0E66B  CCCC6666DDDD2222DDDD111188889999E
EEEEEEEEEEBBBB4444
~~~~!kA7x  000000000000000000000000000000001F1A1E26  EEEE777711117777BBBB1111EEEE88884
444DDDDDDDDDEEEEBBBB
~~~~8Ii!/@  000000000000000000000000000000001F05932F  11119999BBBB44447777000011114444C
CCCAAAA6666DDDD0000
~~~~<I'5>F  000000000000000000000000000000008CB2293  88883333BBBB111166669999888855558
88888822228888CCCC
~~~~G-)m^)  0000000000000000000000000000000013397F73  DDDDFFFFB BBBBCCCCFFFF44446666AAAA1
1113333333AAAACCC
~~~~c+I&CP  0000000000000000000000000000000074BDF64  8888000055550000DDDD22227777AAAA0
00033332222AAAADD
~~~~hb&5X*  0000000000000000000000000000000032C0E06B  7777BBBBBBBB9999EEEEAAAAAAA0000C
CCCCDDDD4444BBBB4444
root@ip-172-31-19-40 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

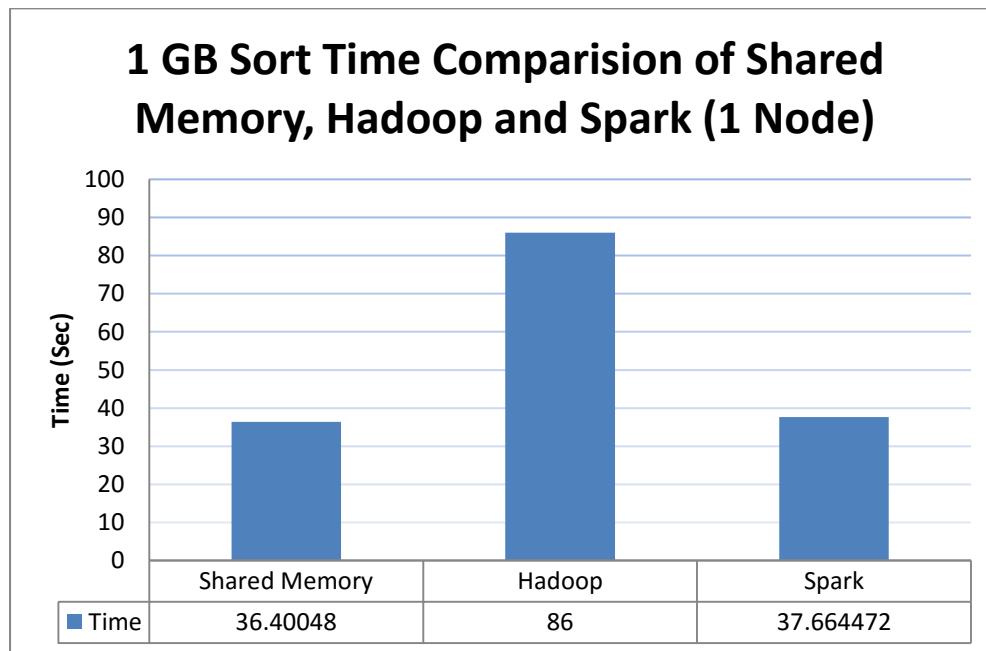
### **Trade-off of Shared Memory, Hadoop and Spark for 1 GB Sorting D2.xlarge:**

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.

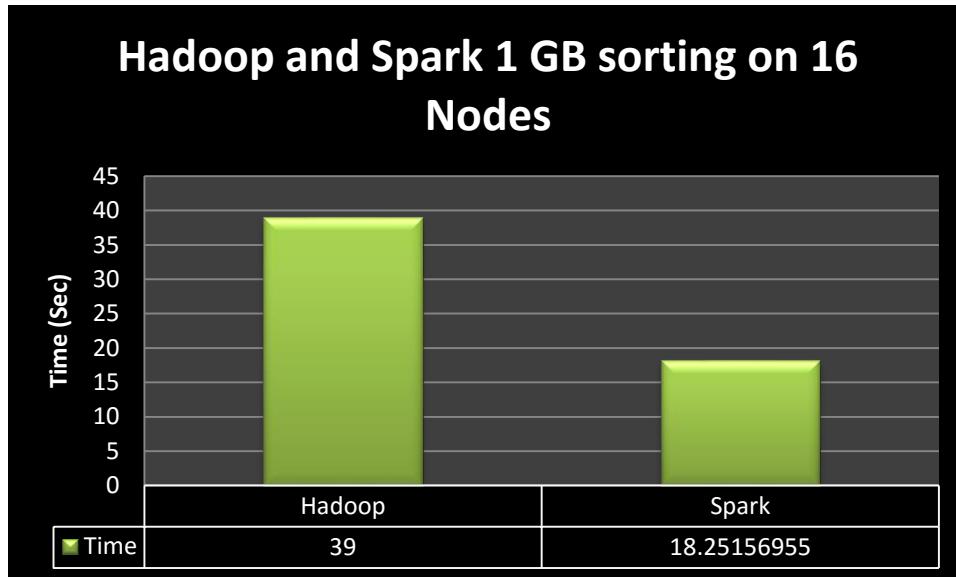
**For Hadoop**- The Hadoop uses the concept of map- reduce which is first loads the data in HDFS and then tries to map the data and shuffles individual chunks and write it to the HDFS during MAP phase, Whereas during the Reduce phase it reads those mapped shuffled data and tries to combine them based 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.

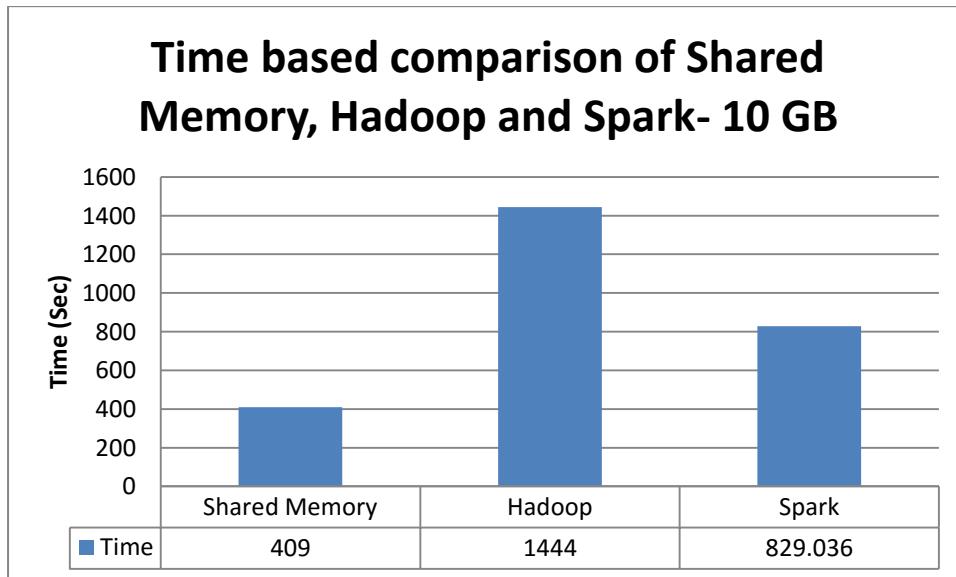


#### **Trade-off of Hadoop and Spark for 1 GB Sorting with 16 Nodes D2.xlarge:**

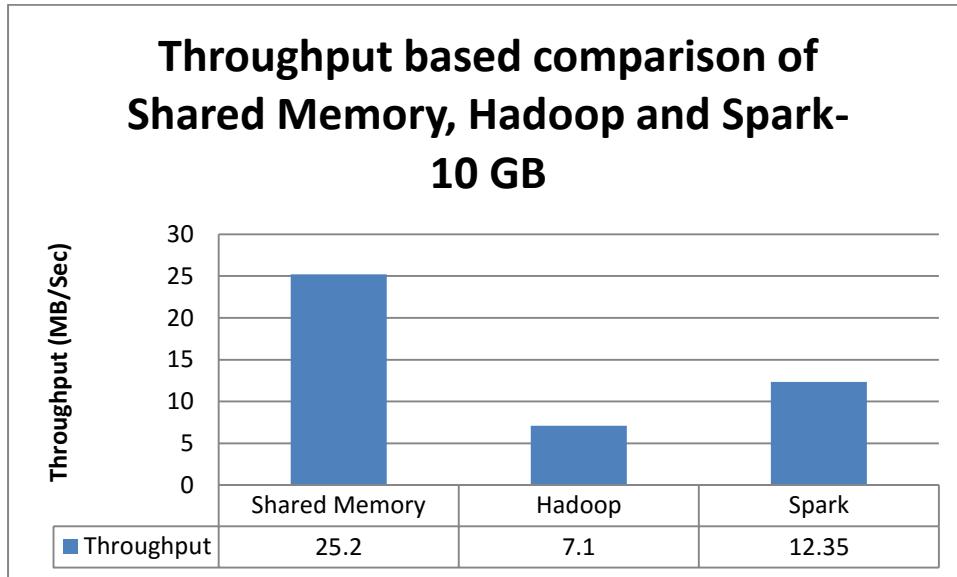
Scaling up from 1 Node to 16 nodes cluster for Hadoop and Spark, we can make out from the graph that, time required to sort 1 GB of data using Hadoop takes much time as compared to the spark. It's because the map phase of Hadoop creates individual chunks of mapper, which will take time to read and write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



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

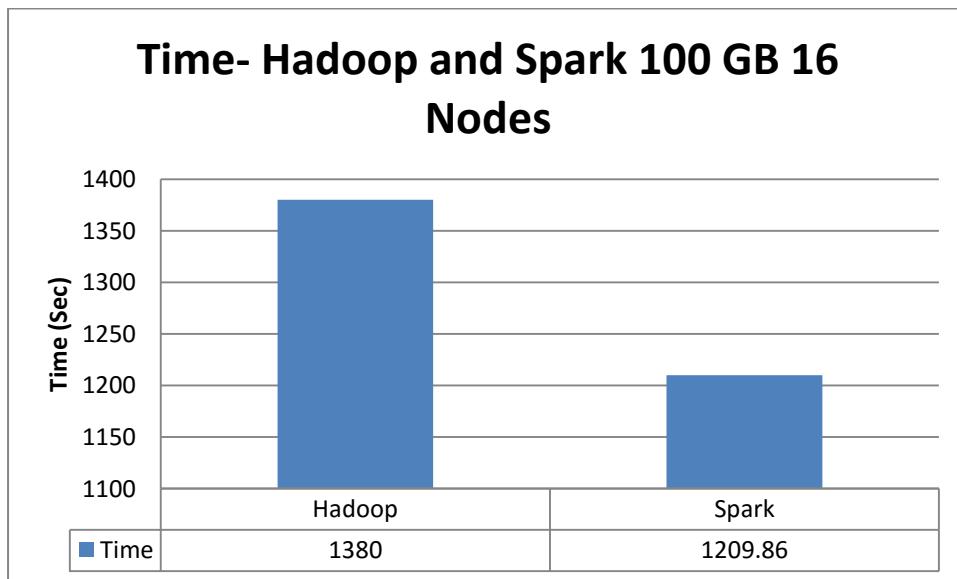


**Speed-up Trade-off of Shared Memory, Hadoop and Spark for 10 GB on C3.large:**

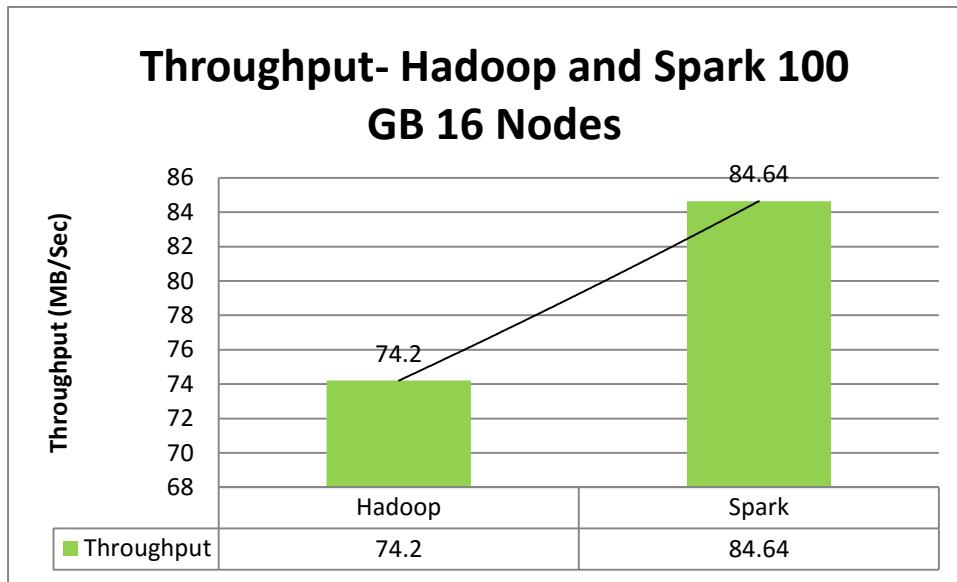


Based on the throughput and time based comparison, we can conclude that the time required to sort Shared memory will be less, compared to the time required to sort the Hadoop and Spark, as shared memory uses the memory to sort data in place based on available free memory. Wherein, case of Hadoop and 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 performance will drastically increase 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:

```
Vinit Shah
Reading http://pypi.python.org/simple/ecdsa/
Best match: ecdsa 0.13
Downloading https://pypi.python.org/packages/source/e/ecdsa/ecdsa-0.13.t
r.gz#md5=1f60eda9cb5c46722856db41a3ae6670
Processing ecdsa-0.13.tar.gz
Running ecdsa-0.13/setup.py -q bdist_egg --dist-dir /tmp/easy_install-wUc
0ug/ecdsa-0.13/egg-dist-tmp-zwqXIW
zip_safe flag not set; analyzing archive contents...
Adding ecdsa 0.13 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/ecdsa-0.13-py2.7.egg
Finished processing dependencies for StarCluster
vinit@vinit-VirtualBox:~$ starcluster help
vinit@vinit-VirtualBox:~$ starcluster createkey mykey -o ~/.ssh/mykey.rsa
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

>>> Successfully created keypair: mykey
>>> fingerprint: 25:6a:f6:42:a5:aa:4a:2e:b3:ce:e7:45:5f:0f:90:38:48:e1:1e:bb
>>> keypair written to /home/vinit/.ssh/mykey.rsa
vinit@vinit-VirtualBox:~$ sudo nano ~/.starcluster/config
[sudo] password for vinit:
Sorry, try again.
[sudo] password for vinit:
vinit@vinit-VirtualBox:~$
```

Star Cluster Installed:

```
Vinit Shah
six: module references __path__
Adding six 1.10.0 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/six-1.10.0-py2.7.egg
Searching for ecdsa>=0.11
Reading http://pypi.python.org/simple/ecdsa/
Best match: ecdsa 0.13
Downloading https://pypi.python.org/packages/source/e/ecdsa/ecdsa-0.13.t
r.gz#md5=1f60eda9cb5c46722856db41a3ae6670
Processing ecdsa-0.13.tar.gz
Running ecdsa-0.13/setup.py -q bdist_egg --dist-dir /tmp/easy_install-wUc
0ug/ecdsa-0.13/egg-dist-tmp-zwqXIW
zip_safe flag not set; analyzing archive contents...
Adding ecdsa 0.13 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/ecdsa-0.13-py2.7.egg
Finished processing dependencies for StarCluster
vinit@vinit-VirtualBox:~$
```

## Starting the star Cluster:

## **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/mailnglist.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:~# █