CS-553 Prog Assignment #2 Ms. Sampada L. Pingale (A20361180)

Report

CONTENTS

---

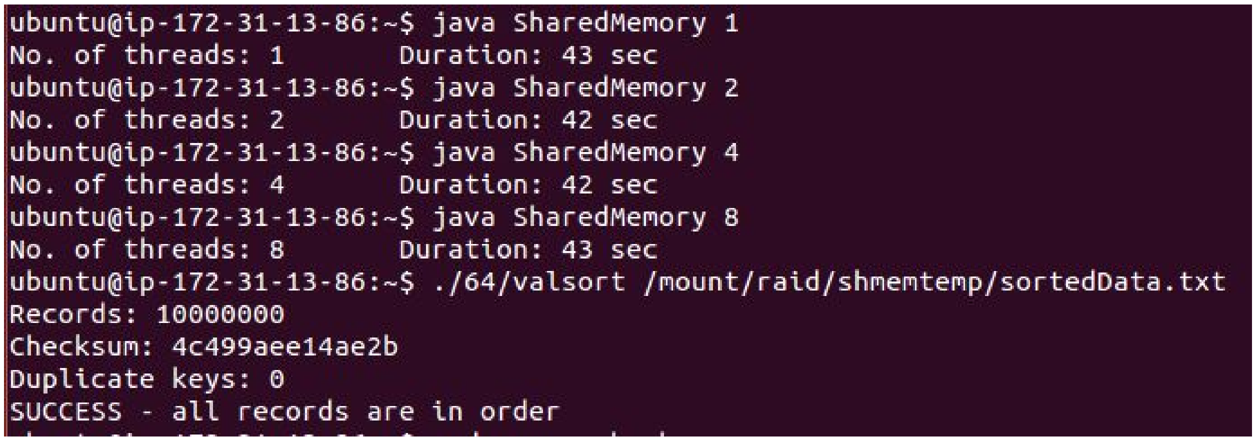
1. Overview:

TeraSort: It is a popular benchmark for measuring the time taken to sort one TB(terabyte) of randomly distributed data.

In this assignment, a sort application is developed in 3 ways i.e. Java, Hadoop and Spark which is capable of sorting datasets that are larger than memory. The goal is to observe and analyze performance of sorting application in these 3 ways. To keep this unaffected from other factors like memory specifications, other background process etc, we will use AWS EC2 C3.large instances for conducting these experiments. Input to all these will be generated by Gensort application. This will ensure each application gets same input. Each part is discussed in detail in following sections.

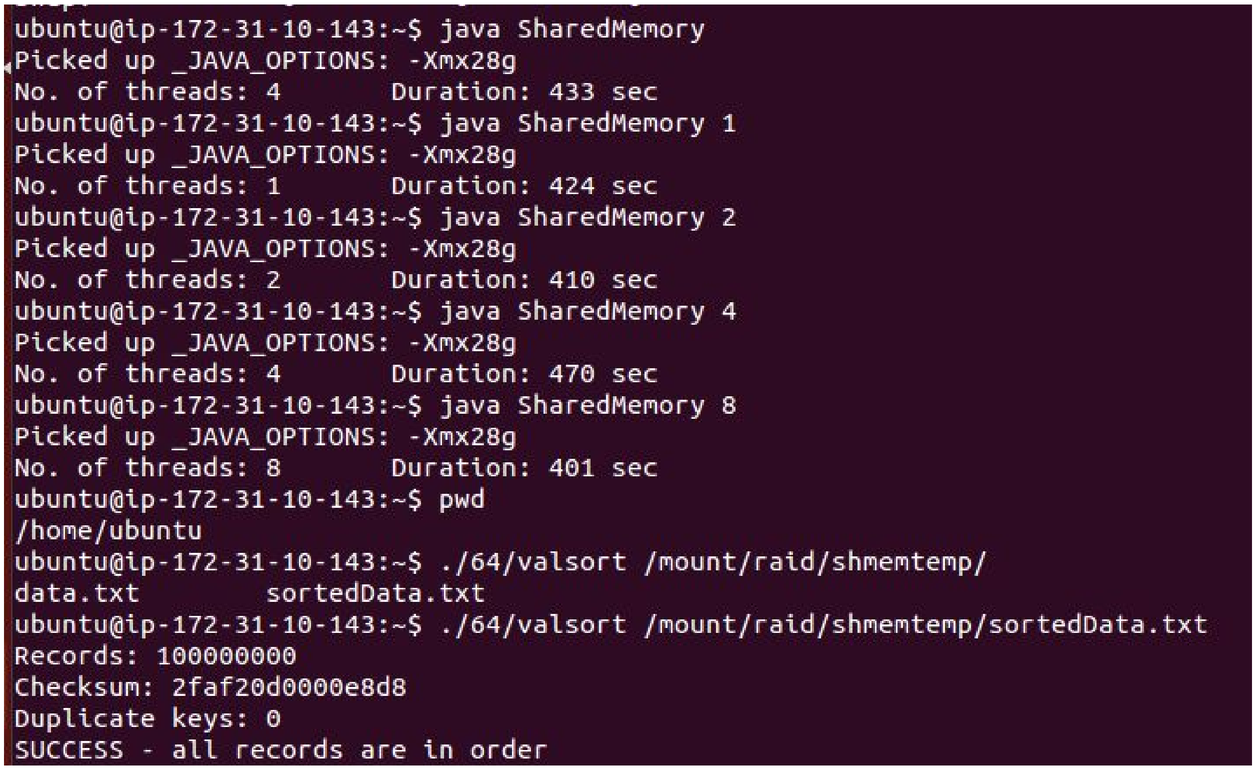
2. SHARED-MEMORY Java:

We performed 4*2 = 8 experiments of Shared-memory. Following are the observations obtained after performing 1, 2, 4 and 8 threaded sort on 1 GB data :



```
ubuntu@ip-172-31-13-86:~$ java SharedMemory 1
No. of threads: 1       Duration: 43 sec
ubuntu@ip-172-31-13-86:~$ java SharedMemory 2
No. of threads: 2       Duration: 42 sec
ubuntu@ip-172-31-13-86:~$ java SharedMemory 4
No. of threads: 4       Duration: 42 sec
ubuntu@ip-172-31-13-86:~$ java SharedMemory 8
No. of threads: 8       Duration: 43 sec
ubuntu@ip-172-31-13-86:~$ ./64/valsort /mount/raid/shmemtemp/sortedData.txt
Records: 10000000
Checksum: 4c499aee14ae2b
Duplicate keys: 0
SUCCESS - all records are in order
```

In the 2nd experiment, increased the data size to 10GB and performed the same sort application. Following are the observations obtained after performing 1, 2, 4 and 8 threaded sort on 10 GB data :



```
ubuntu@ip-172-31-10-143:~$ java SharedMemory
Picked up _JAVA_OPTIONS: -Xmx28g
No. of threads: 4       Duration: 433 sec
ubuntu@ip-172-31-10-143:~$ java SharedMemory 1
Picked up _JAVA_OPTIONS: -Xmx28g
No. of threads: 1       Duration: 424 sec
ubuntu@ip-172-31-10-143:~$ java SharedMemory 2
Picked up _JAVA_OPTIONS: -Xmx28g
No. of threads: 2       Duration: 410 sec
ubuntu@ip-172-31-10-143:~$ java SharedMemory 4
Picked up _JAVA_OPTIONS: -Xmx28g
No. of threads: 4       Duration: 470 sec
ubuntu@ip-172-31-10-143:~$ java SharedMemory 8
Picked up _JAVA_OPTIONS: -Xmx28g
No. of threads: 8       Duration: 401 sec
ubuntu@ip-172-31-10-143:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-10-143:~$ ./64/valsort /mount/raid/shmemtemp/
data.txt        sortedData.txt
ubuntu@ip-172-31-10-143:~$ ./64/valsort /mount/raid/shmemtemp/sortedData.txt
Records: 100000000
Checksum: 2faf20d0000e8d8
Duplicate keys: 0
SUCCESS - all records are in order
```

Observations:

Table 1

Threads	Time(sec)	MB per second
1	424 seconds	2.358491
2	410 seconds	2.439024
4	470 seconds	2.12766
8	401 seconds	2.493766

Analysis:

To analyse these observations, plotting above time against no. of threads in following chart:

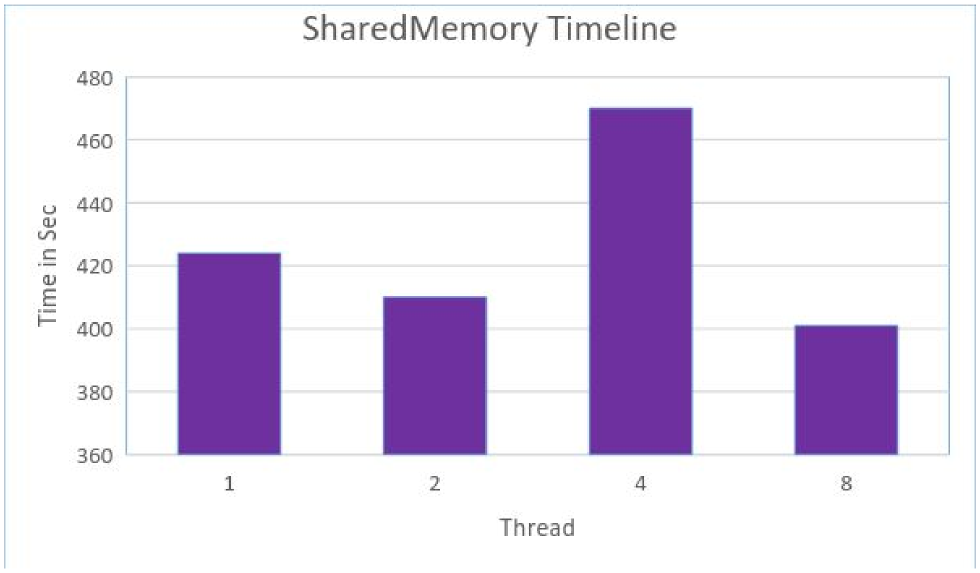
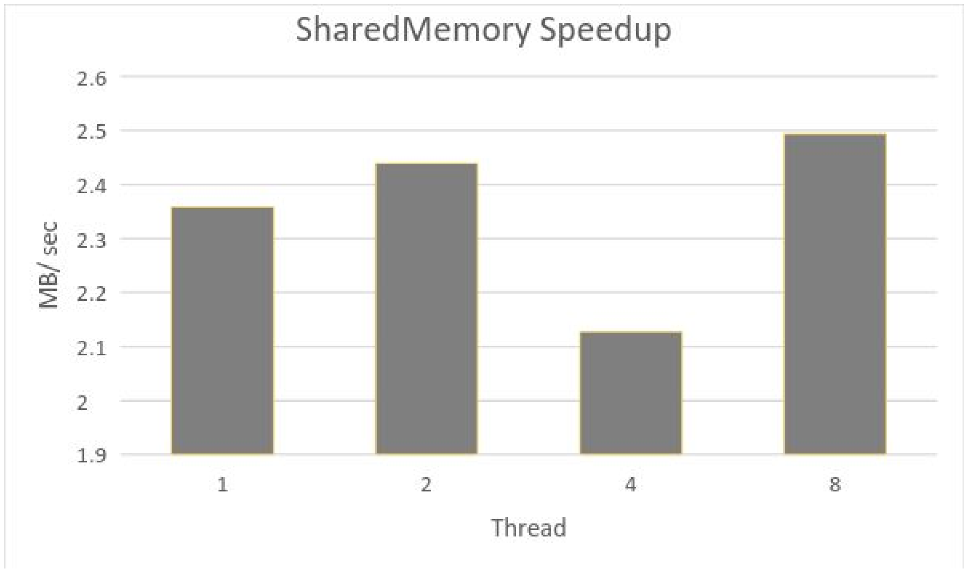


Chart 1: SharedMemory Timeline

Computing the throughput in mbps and plotting it against no of threads gave following curve of speedup:



Char 2 : Shared Memory Speedup

Conclusion:

SharedMemory gave great performance in 2 and 4 threads. As the instance used in these experiments have 2 virtual cores, we can see performance gain with each 2nd thread. Hence the throughput was gained and time was decreased.

3. Hadoop:

It is the second way in which the sort application is implemented. Hadoop can be thought of as a framework that will store and process huge data (big data) in a distributed environment of multi node cluster. It eases down the programming efforts from one node to multi node computer with local computing and storage enabled.

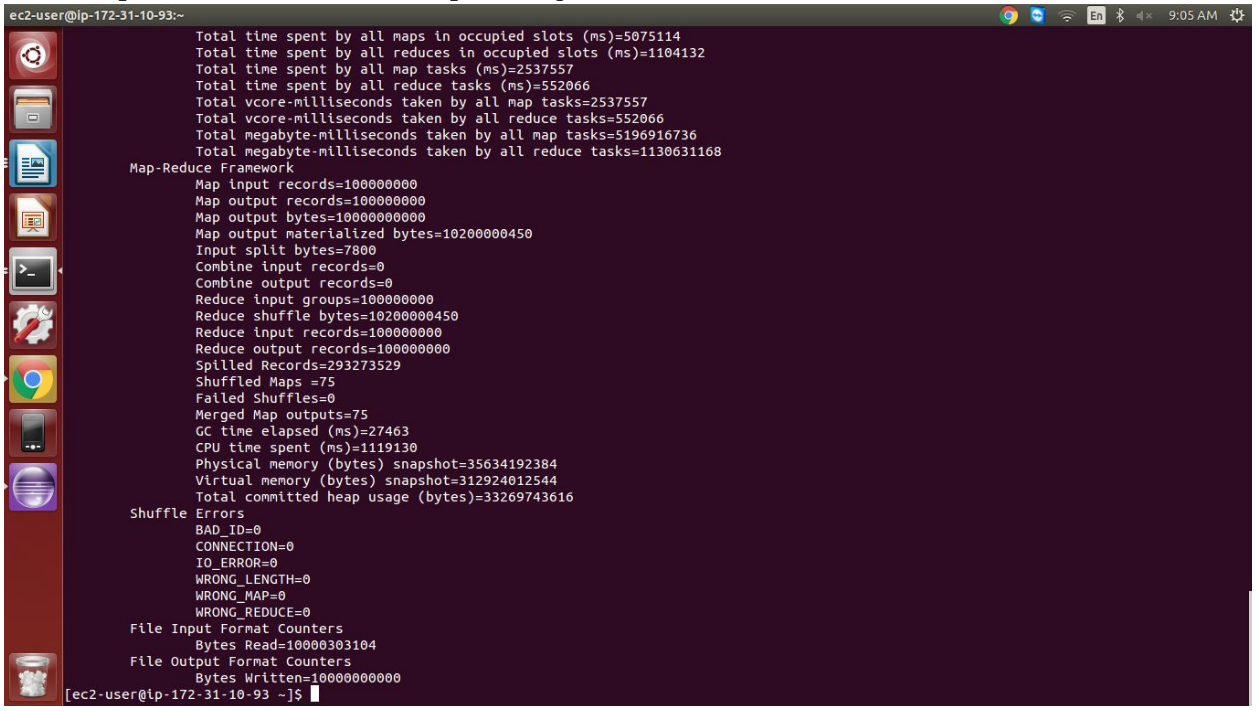
Here are 2 experiments on HadoopSort sequentially for 10GB and 100 GB data. For this a Map-Reduce style program is implemented which can be best fit for tera-sort Application

To perform the experiments, installed hadoop on one node cluster of AWS. Then generated 10GB data on the instance using gensort application. It took **1119.130** seconds for running the sort application implemented in hadoop.

Experiment 1: 10 GB on one node cluster:

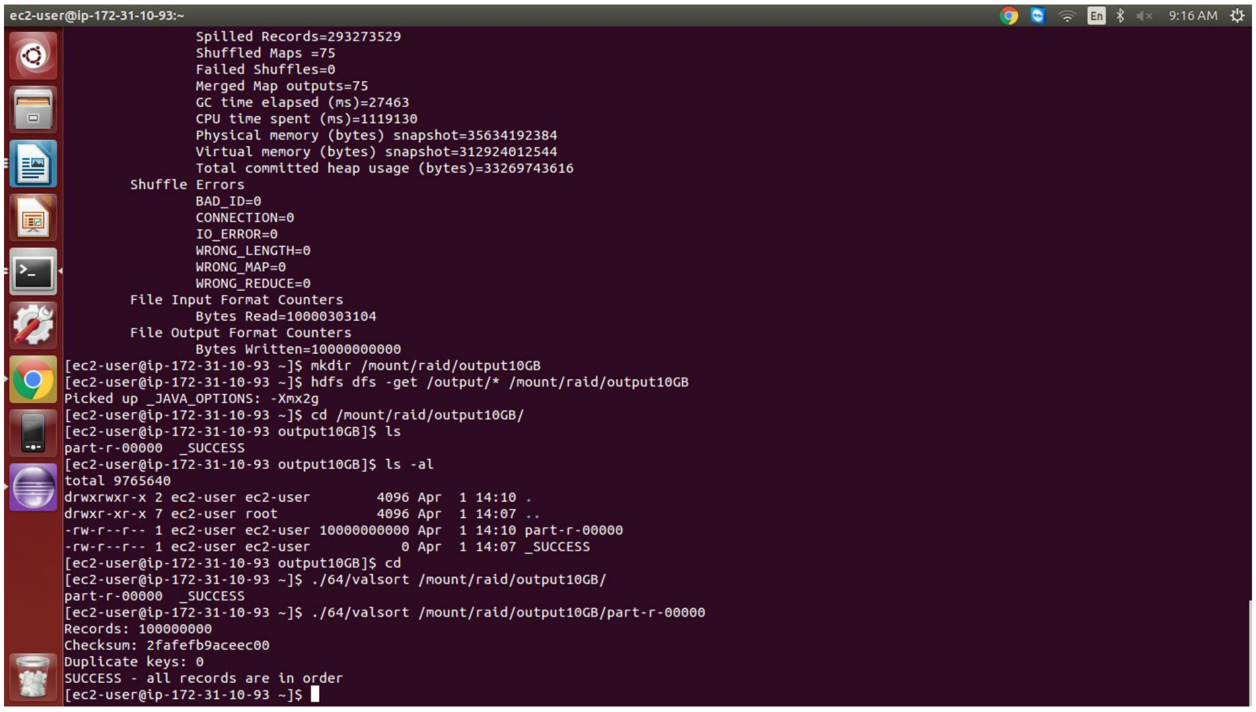
- For this, downloaded and configured hadoop on one node.
- We added 2 - 200GB of EBS volumes to the node. And configured RAID 0 on the disk giving one volume of 400GB.
- Generated 10 GB data on the node uing gensort
- Ran the HadoopSort.jar on this data using script and steps mentioned in readme.txt
- Fetched the output and performed valsort on it.
- It shown that all records were in order.

Following is the status after running HadoopSort on 10GB:



```
ec2-user@ip-172-31-10-93:~
                Total time spent by all maps in occupied slots (ms)=5075114
                Total time spent by all reduces in occupied slots (ms)=1104132
                Total time spent by all map tasks (ms)=2537557
                Total time spent by all reduce tasks (ms)=552066
                Total vcore-milliseconds taken by all map tasks=2537557
                Total vcore-milliseconds taken by all reduce tasks=552066
                Total megabyte-milliseconds taken by all map tasks=5196916736
                Total megabyte-milliseconds taken by all reduce tasks=1130631168
        Map-Reduce Framework
                Map input records=100000000
                Map output records=100000000
                Map output bytes=10000000000
                Map output materialized bytes=10200000450
                Input split bytes=7800
                Combine input records=0
                Combine output records=0
                Reduce input groups=100000000
                Reduce shuffle bytes=10200000450
                Reduce input records=100000000
                Reduce output records=100000000
                Spilled Records=293273529
                Shuffled Maps =75
                Failed Shuffles=0
                Merged Map outputs=75
                GC time elapsed (ms)=27463
                CPU time spent (ms)=1119130
                Physical memory (bytes) snapshot=35634192384
                Virtual memory (bytes) snapshot=312924012544
                Total committed heap usage (bytes)=33269743616
        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=10000303104
        File Output Format Counters
                Bytes Written=10000000000
[ec2-user@ip-172-31-10-93 ~]$
```

After fetching the output and performing valsort on it the output was:



```
ec2-user@ip-172-31-10-93:~
                Spilled Records=293273529
                Shuffled Maps =75
                Failed Shuffles=0
                Merged Map outputs=75
                GC time elapsed (ms)=27463
                CPU time spent (ms)=1119130
                Physical memory (bytes) snapshot=35634192384
                Virtual memory (bytes) snapshot=312924012544
                Total committed heap usage (bytes)=33269743616
        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=10000303104
        File Output Format Counters
                Bytes Written=10000000000
[ec2-user@ip-172-31-10-93 ~]$ mkdir /mount/raid/output10GB
[ec2-user@ip-172-31-10-93 ~]$ hdfs dfs -get /output/* /mount/raid/output10GB
Picked up _JAVA_OPTIONS: -Xmx2g
[ec2-user@ip-172-31-10-93 ~]$ cd /mount/raid/output10GB/
[ec2-user@ip-172-31-10-93 output10GB]$ ls
part-r-00000  _SUCCESS
[ec2-user@ip-172-31-10-93 output10GB]$ ls -al
total 9765640
drwxrwxr-x 2 ec2-user ec2-user        4096 Apr  1 14:10 .
drwxr-xr-x 7 ec2-user root            4096 Apr  1 14:07 ..
-rw-r--r-- 1 ec2-user ec2-user 10000000000 Apr  1 14:10 part-r-00000
-rw-r--r-- 1 ec2-user ec2-user           0 Apr  1 14:07 _SUCCESS
[ec2-user@ip-172-31-10-93 output10GB]$ cd
[ec2-user@ip-172-31-10-93 ~]$ ./64/valsort /mount/raid/output10GB/
part-r-00000  _SUCCESS
[ec2-user@ip-172-31-10-93 ~]$ ./64/valsort /mount/raid/output10GB/part-r-00000
Records: 100000000
Checksum: 2fafefb9aceec00
Duplicate keys: 0
SUCCESS - all records are in order
[ec2-user@ip-172-31-10-93 ~]$
```

Experiment 2: 100GB on 16 node cluster:

To perform this experiment, launched a cluster of 17 nodes (1 master and 16 slaves) on AWS. For this, we had to go through following modifications:

A. Launch the master node and change 5 files in hadoop/etc/hadoop folder:

1. hadoop-env.sh: This file is updated with JAVA_HOME environment variable.
2. core-site.xml: Updated with tmp directory location for hadoop and private ip, port number of master
3. hdfs-site.xml: Added configuration settings of HDFS daemons, Namenode, Secondary Namenode, Datanode giving directory locations for same.
4. yarn-site.xml: This file will have properties for yarn framework, node manager and resource manager
5. mapred-site.xml: Config settings for mappers and reducers.

B. Generate ssh key for allowing communication between cluster nodes.
C. Requested the AMI image of master node. Launched slaves on this AMI
D. Updated all slave files with self public IP
E. Updated slave file in master with self public IP and public IP of all slaves
F. Format hdfs
G. Start all services

This will start resourcemanager, nodemanager, datanode, namenode, Secondary namenode on master.

And it will start nodemanager and datanode on all slaves. After that the HadoopSort.jar was ran over the 100GB of data. It took 9630 seconds to complete the job.

Observations:

Table 2

Input Data Size	Time (sec)	MB per second
10GB dataset over 1 virtual node	1119.13 seconds	8.935512
100GB dataset over 16 virtual nodes	9630 seconds	10.38422

Analysis:

Following are the timeline and speedup graphs for HadoopSort.

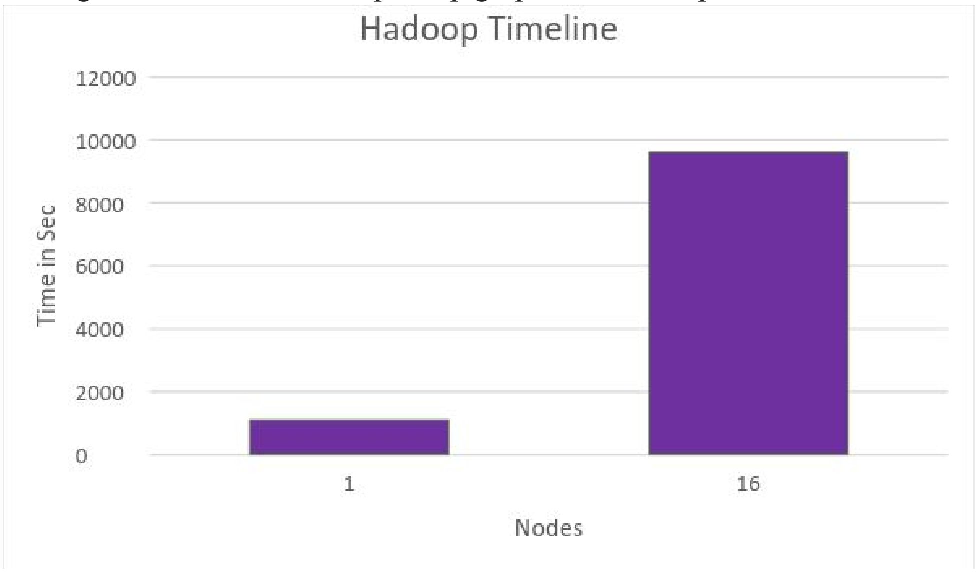


Chart 3. Hadoop Line Execution

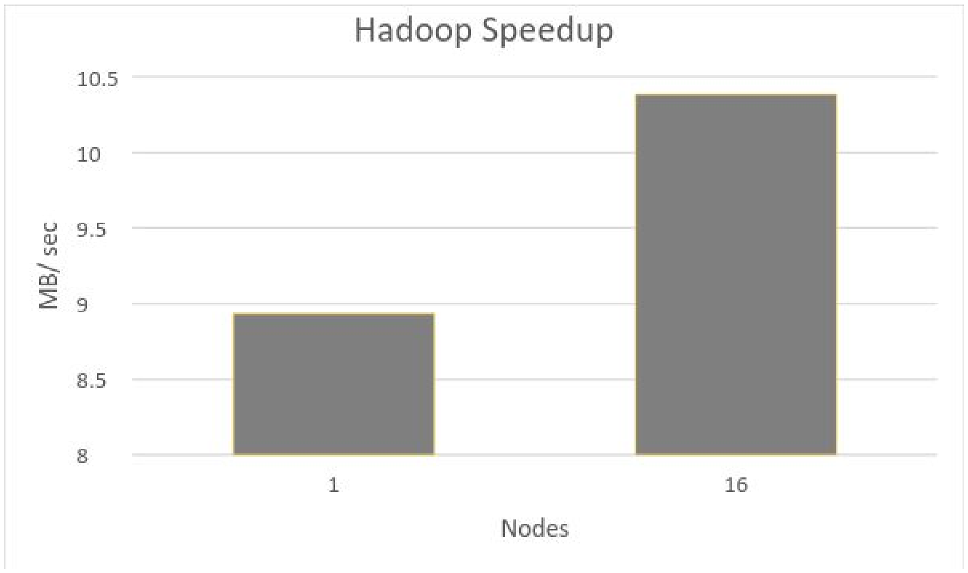


Chart 4 : Hadoop Speedup

Conclusion:

The graphs drawn above clearly shows speedup when taken from one node to multi-node taking advantage of multiple computers in the cluster. Hadoop has made use of these nodes to improve the speed of sorting.

4. Spark:

Spark is cluster computing framework by Apache, which covers drawbacks of Hadoop when it comes to iterative jobs on huge data and preserve information gained in intermediate levels. This increases performance of spark drastically.

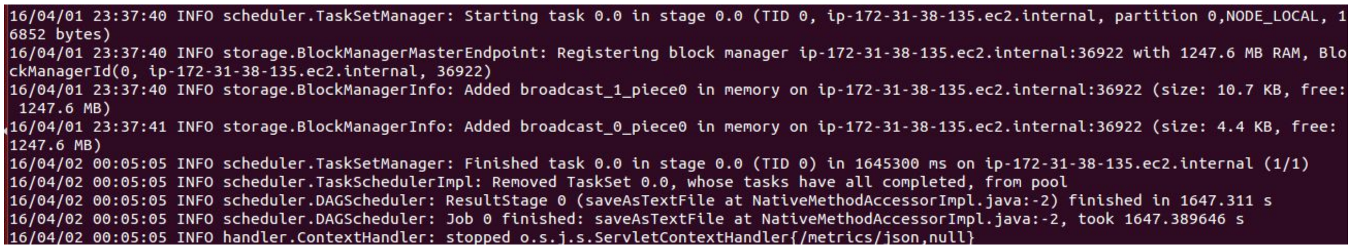
We are to perform 2 experiments on spark with same goals as above experiments.

For this:

- downloaded Apache Spark on local node and extracted on local host.
- generated Access key on AWS in Security Credentials
- Exported these Key ID and Secret Key from ec2 folder of spark.
- Generated cluster using command as given in readme.txt specifying number of slaves
- Ran the experiment SparkSort.py on the cluster.

Experiment 1: 10GB

-Similar to hadoop, generated 10 GB data on the master node and executed python program in spark using commands given in readme.txt.



```
16/04/01 23:37:40 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, ip-172-31-38-135.ec2.internal, partition 0,NODE_LOCAL, 1
6852 bytes)
16/04/01 23:37:40 INFO storage.BlockManagerMasterEndpoint: Registering block manager ip-172-31-38-135.ec2.internal:36922 with 1247.6 MB RAM, Blo
ckManagerId(0, ip-172-31-38-135.ec2.internal, 36922)
16/04/01 23:37:40 INFO storage.BlockManagerInfo: Added broadcast_1_piece0 in memory on ip-172-31-38-135.ec2.internal:36922 (size: 10.7 KB, free:
 1247.6 MB)
16/04/01 23:37:41 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on ip-172-31-38-135.ec2.internal:36922 (size: 4.4 KB, free:
1247.6 MB)
16/04/02 00:05:05 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 1645300 ms on ip-172-31-38-135.ec2.internal (1/1)
16/04/02 00:05:05 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
16/04/02 00:05:05 INFO scheduler.DAGScheduler: ResultStage 0 (saveAsTextFile at NativeMethodAccessorImpl.java:-2) finished in 1647.311 s
16/04/02 00:05:05 INFO scheduler.DAGScheduler: Job 0 finished: saveAsTextFile at NativeMethodAccessorImpl.java:-2, took 1647.389646 s
16/04/02 00:05:05 INFO handler.ContextHandler: stopped o.s.j.s.ServletContextHandler{/metrics/json,null}
```

Experiment 2: 100GB

- Launched a cluster of 16 slaves and 1 master for this experiment through same steps as above.
- Just like 10GB, generated 100GB of data using gensort on master node and ran SparkSort.py on it.

Observations:

Table 3

Input Data Size	Time	MB per second
10GB dataset over 1 virtual node	1647.3 seconds	6.07054
100GB dataset over 16 virtual nodes	2160 seconds	46.2963

Analysis:

The time and speed up for spark can be seen in below given charts:

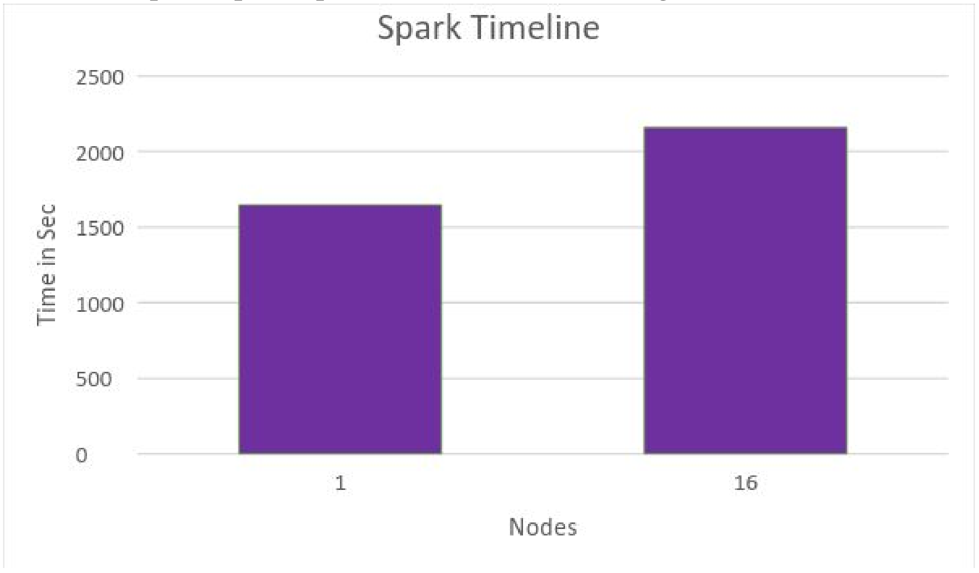


Chart 5: Spark Timline

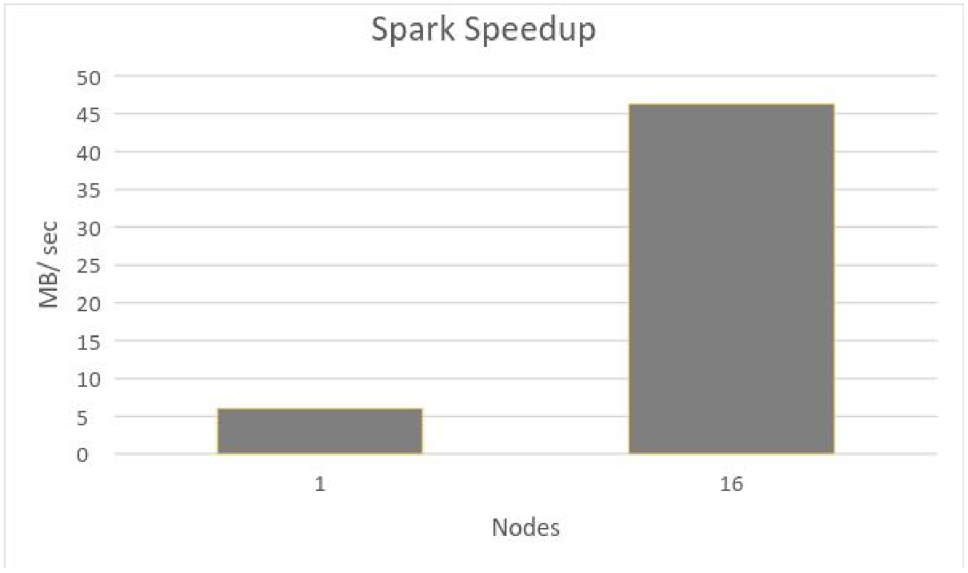


Chart 6: Spark Speedup

Conclusion:

Spark gave a huge performance gain when transformed from 1 node to 16 nodes. This is due to iterative behaviour of spark and nature of using the cache at the most possible times.

5. Answers:

A. Functions of files:

1. hadoop-env.sh: This file is updated with JAVA_HOME environment variable.
2. core-site.xml: Updated with tmp directory location for hadoop and private ip, port number of master
3. hdfs-site.xml: Added configuration settings of HDFS daemons, Namenode, Secondary Namenode, Datanode giving directory locations for same.
4. yarn-site.xml: This file will have properties for yarn framework, node manager and resource manager
5. mapred-site.xml: Config settings for mappers and reducers.

B. Multinode Hadoop installaion:

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

→ Master: This is the one node which has responsibilities of assigning tasks and distributing the data amongst slave nodes. It plays important role in parallel computations on data.

→ Slave: It can be thought of a normal node which takes job from master, input data to process and returns the result to the master. It works independently to other slaves. Hoever, it is dependent on master node to get assigned to the job.

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

→ Setting the same port on shared environment will become ambigious when it will come to identify the processes.
→ Moreover, if we do so, then finding where the things goes wrong will be near to impossible. It will make it critical for nodes to communicate amongst each other

3) How can we change the number of mappers and reducers from the configuration file?
→ We will need to specify it in the mapred-site.xml
4) Best at one node scale:
→ Hadoop is best at one node scale.
5) Best at 16 nodes scale:
→ Spark is best at one node scale.
6) Best at 100 node scale:
→ Spark will be very good at 100 nodes as it has exponential performance gain in comparison with Hadoop and Shared Memory
7) Best at 1000 node scale:
→ As the nodes goes higher and higher, we will surely gain performance in Spark then Hadoop.
8) Hadoop Spark Benchmark Comparison
→ Speed: 1.42 TB/min with 2100 nodes -- Daytona Hadoop and Indy Hadoop.
Performance acheived: 4.27 TB/min with 207 AWS EC2 Instances.
→ We gained a performance at 16 nodes which was much lesser than this. It shows the nature of spark as it gains performance with increasing number of nodes and size.

6. Performance evaluation:

1> Hadoop Vs. Spark:

A. Time taken:

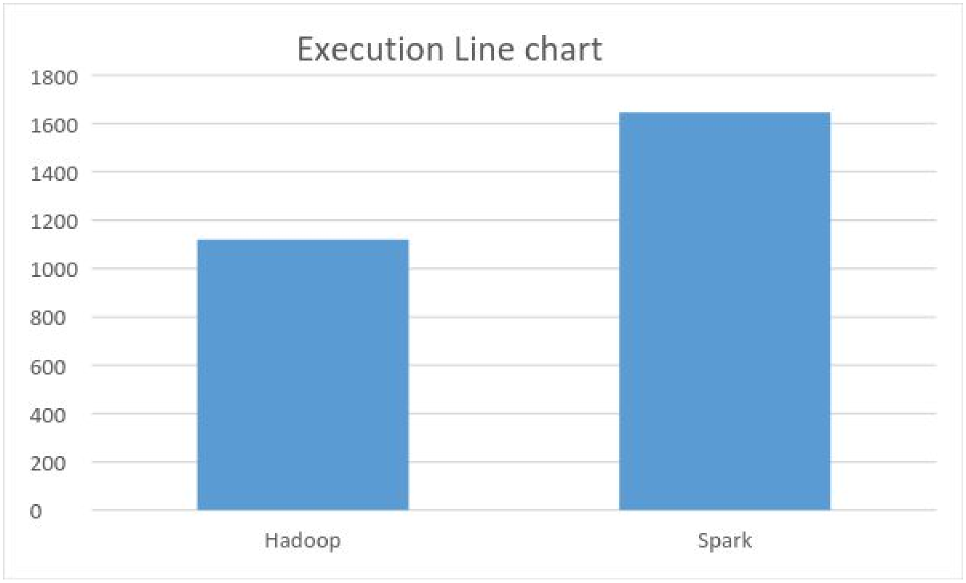


Chart 7: Time

As we can clearly see, Spark saves a lot of time as compared to Hadoop. The reason is, spark has lesser number of Disk read and write as it uses the main memory most of the times. Also, it assigns tasks to the slaves where the data is closer to them reducing network latency.

B. Speedup:

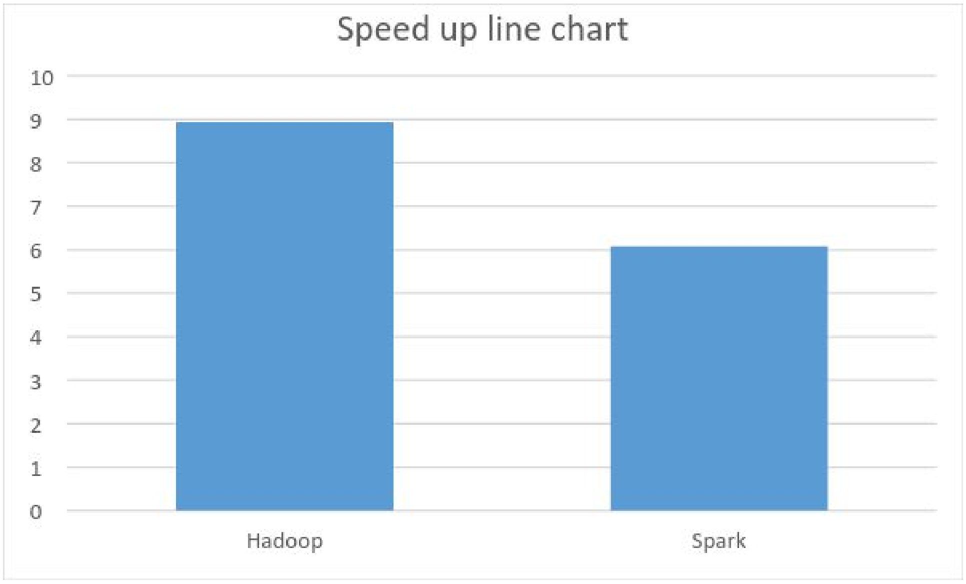


Chart 8: Speed UP

As we can see, Spark gives much better speed as compared to the hadoop. This is as discussed earlier, the spark nature of making maximum usage of main memory.

C. Speed UP (Shared Memory Vs Hadoop Vs. Spark)

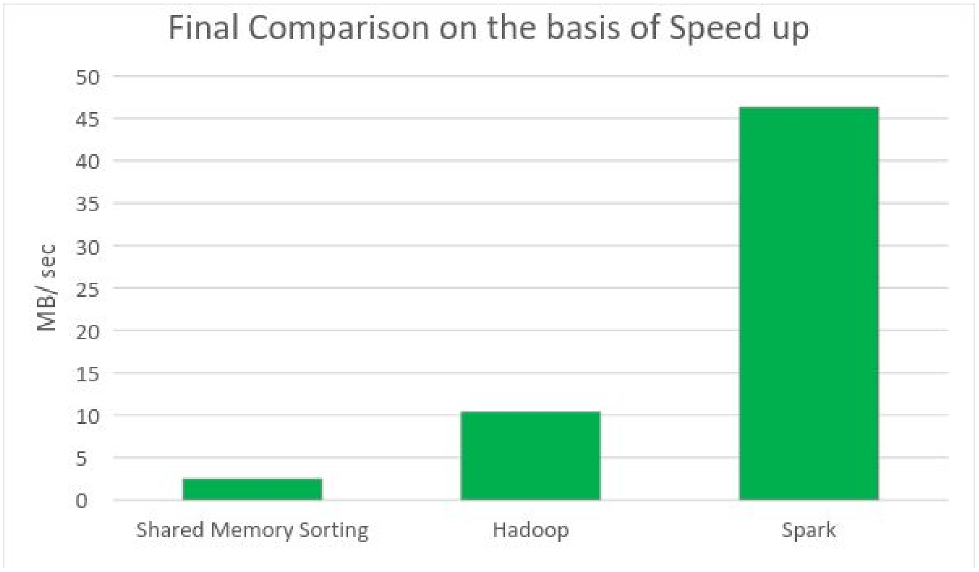


Chart 9: Overall Speed up

When compared the performance of all 3 ways, we can clearly see, Spark performs way better than both the other ways. Also, Hadoop makes good usage of multinodes giving better performance than SharedMemory.