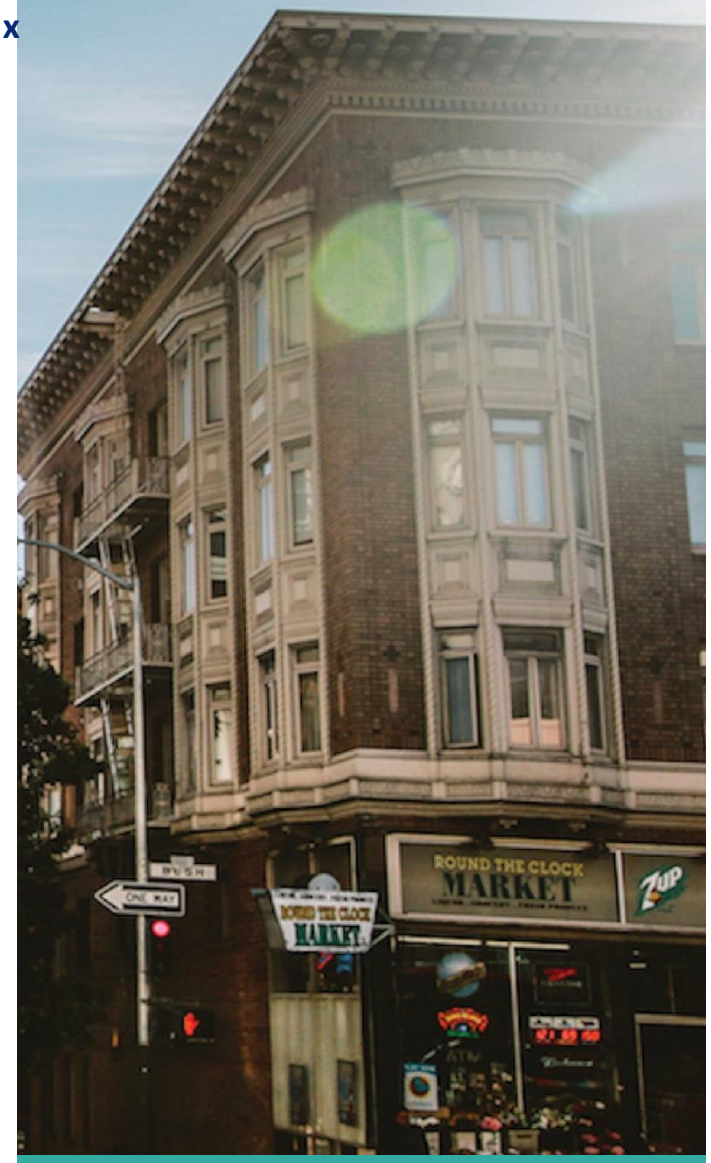
Sort using Spark and Hadoop



APRIL 29

CS553

Authored by: Chetan Gupta (A20378854)

Introduction

This programming assignment covers sort through Hadoop and Spark on multiple nodes. We have used Linux system to implement your application and used the Proton Cluster accessible at 216.47.142.37; each Hadoop group (cluster) has 4 nodes, each node having 4-cores, 8GB of memory, and 80GB of SSD storage.

AIM OF THE EXPERIMENT:

Firstly, the aim of this assignment performing the following tasks on these two configurations:

1. Hadoop Terasort
2. Spark Terasort

1.1 Hadoop Tera sort: -

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking

Hadoop MapReduce – an implementation of the MapReduce programming model for large-scale data processing.

Implementation:

The Hadoop map-reduce program functions in 3 parts:

1. Mapper Class
2. Reducer Class
3. Driver Class

MAPPER CLASS:

• In this class the intermediate key-value pairs are generated. The string are broken into tokens by importing the class StringTokenizer.

• Mapper has 4 variables - key-in, value-in and key-out ,value-out.

• Firstly, the framework first call setup is called and then the combiner is used for the local grouping of the intermediate key-value pairs. The result of this is forwarded to the reducer.

REDUCER CLASS:

• The reducer also has 4 variables and here the inputs are obtained from the output of the mapper.

• The tokens are sorted and shuffles according to the values in the key-value pairs and pushes to the output collector the key and the value.

DRIVER CLASS:

• Map reduce job in Hadoop is triggered using the driver class.

• It is in this class that all the details like job, the output keys and values and the mapper ad reducer classes are given.

1.2 Spark Tera sort: -

Apache Spark has as its architectural foundation the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API.

Analysis

Table 1: Performance evaluation of sort (weak scaling – small dataset)

Experiment	Shared Memory(1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 8GB)	Spark Sort (4VM 8GB)
Computation Time (sec)	58.834	27.876	752	413
Data Read (GB)	4	2	16	16
Data Write (GB)	4	2	16	16
I/O Throughput (MB/sec)	139.2392154	146.9364328	43.57446809	79.34140436
Speedup	0.236903831	0.843019085	0.296553191	0.539970944
Efficiency	5.922595778	21.07547711	29.65531915	53.99709443

Table 2: Performance evaluation of sort (strong scaling – large dataset)

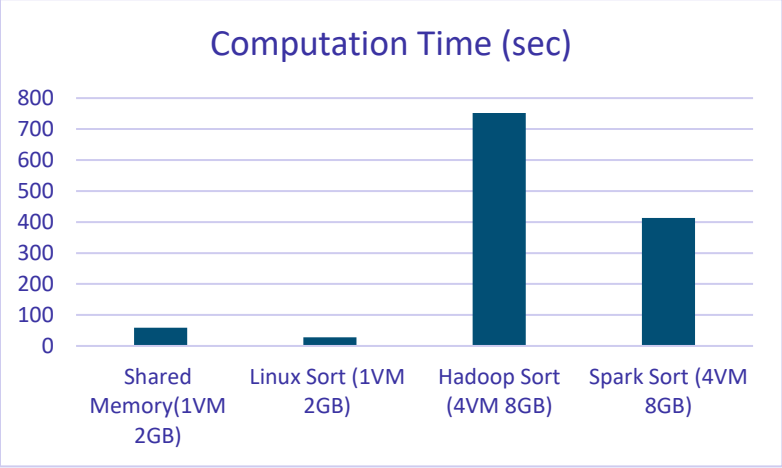
Experiment	Shared Memory(1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 20GB)	Spark Sort (4VM 20GB)
Computation Time (sec)	646.486	447.88	2253	1133
Data Read (GB)	40	20	32	32
Data Write (GB)	40	20	32	32
I/O Throughput (MB/sec)	126.7158144	91.45306779	29.08832668	57.84289497
Speedup	0.346395746	0.785994574	0.318068353	0.632487202
Efficiency	8.65989364	19.64986436	31.80683533	63.24872021

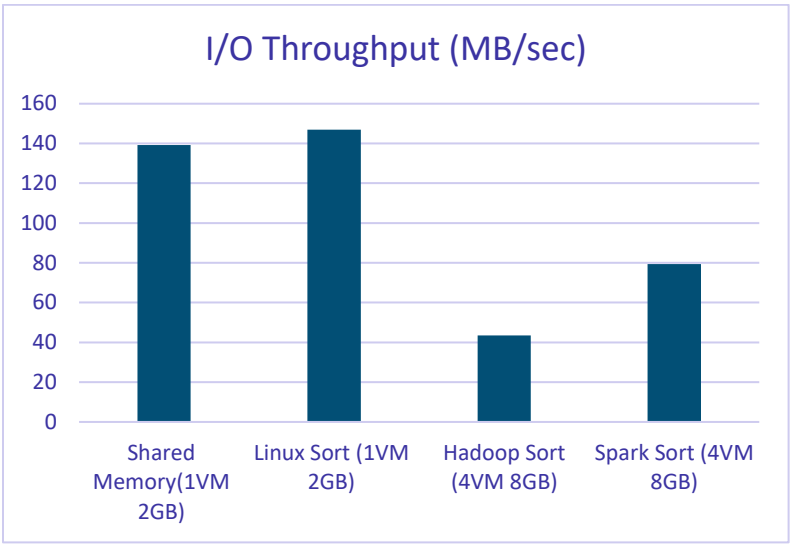
Table 3: Performance evaluation of sort (weak scaling – large dataset)

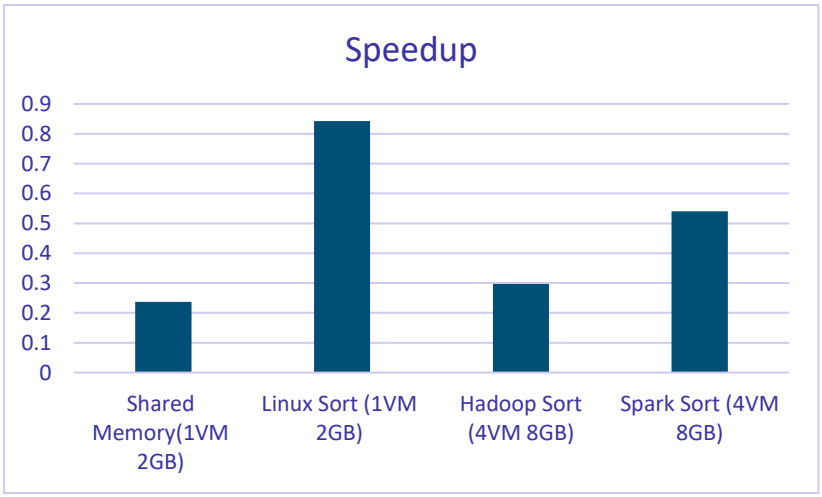
Experiment	Shared Memory(1VM 2GB)	Linux Sort (1VM 2GB)	Hadoop Sort (4VM 80GB)	Spark Sort (4VM 80GB)
Computation Time (sec)	646.486	447.88	3487	1826
Data Read (GB)	40	20	160	160
Data Write (GB)	40	20	160	160
I/O Throughput (MB/sec)	126.7158144	91.45306779	93.97189561	71.78094195
Speedup	0.346395746	0.649546084	0.741595641	0.784893757
Efficiency	34.63957456	64.95460838	74.1595641	78.48937568

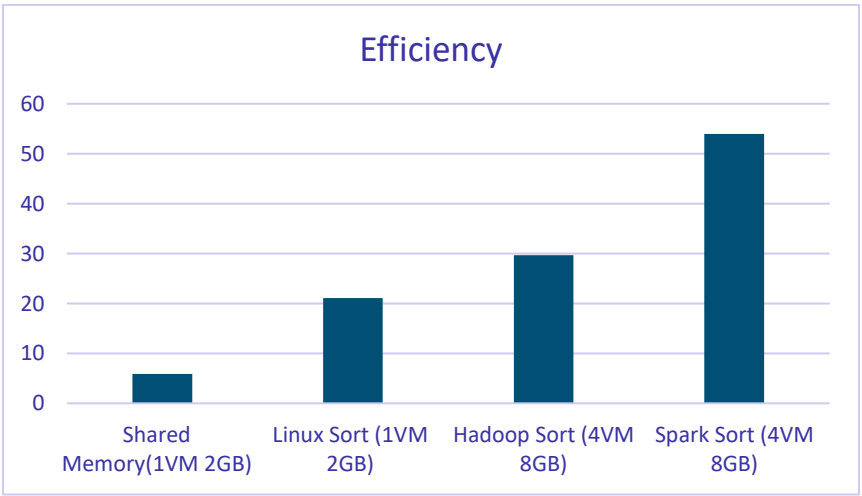
Graphs

1 Performance *evaluation of sort (strong scaling – large dataset) 8GB*

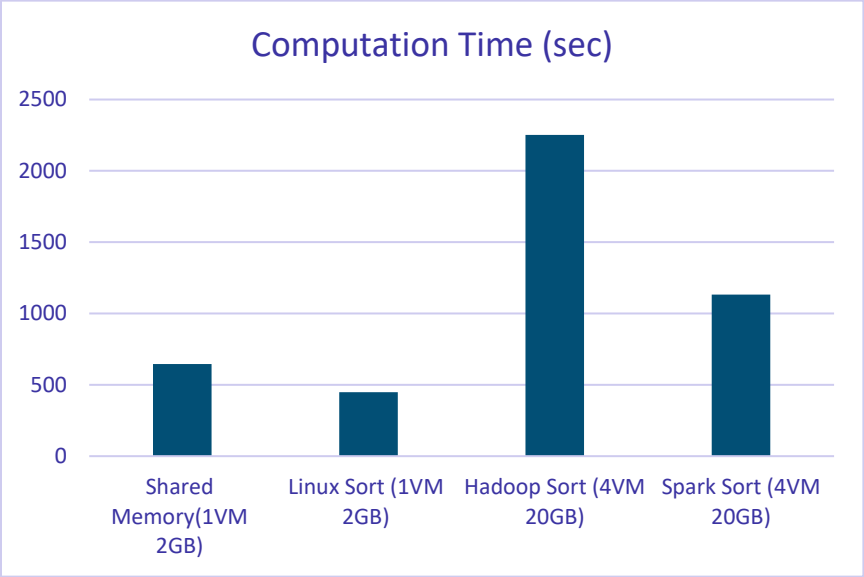


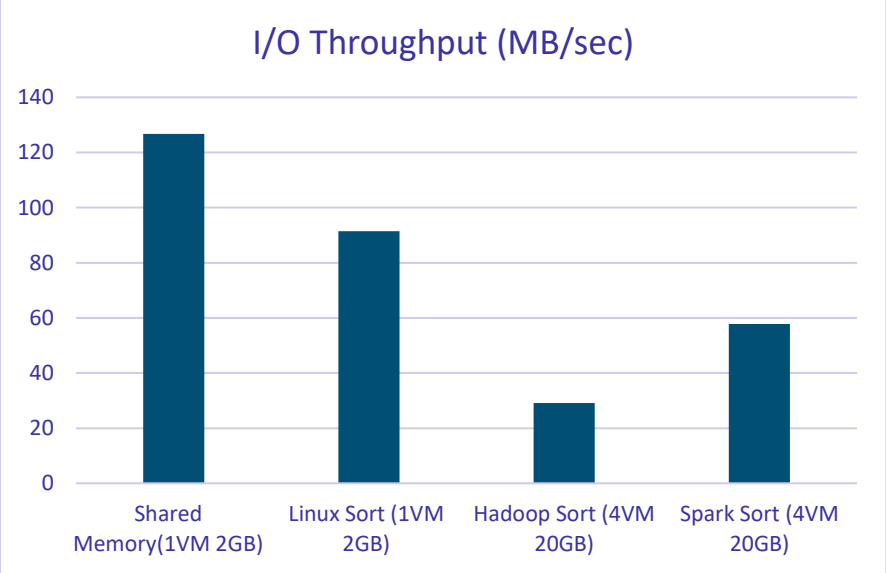

I/O Throughput (MB/sec)
160
140
120
100
80
60
40
20
0
Shared Memory(1VM 2GB)
Linux Sort (1VM 2GB)
Hadoop Sort (4VM 8GB)
Spark Sort (4VM 8GB)

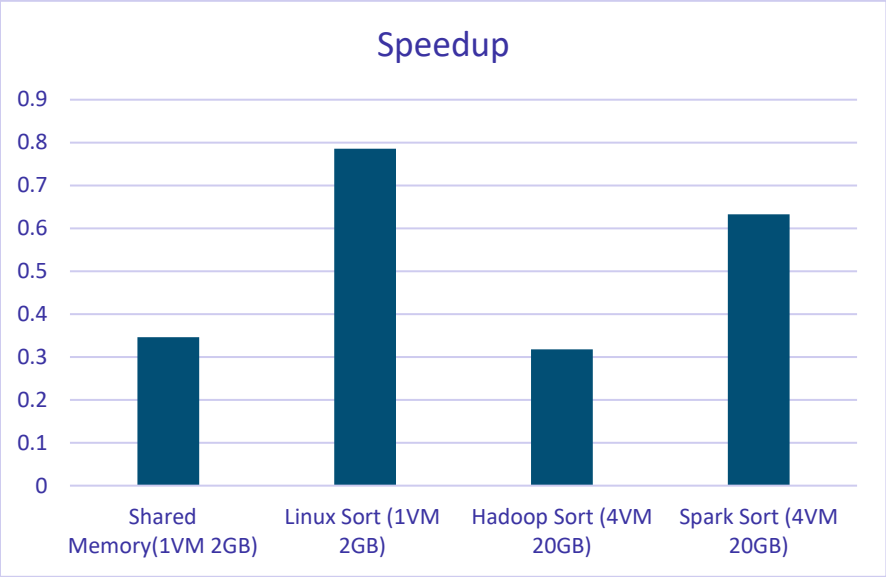

Speedup
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Shared Memory(1VM 2GB)
Linux Sort (1VM 2GB)
Hadoop Sort (4VM 8GB)
Spark Sort (4VM 8GB)

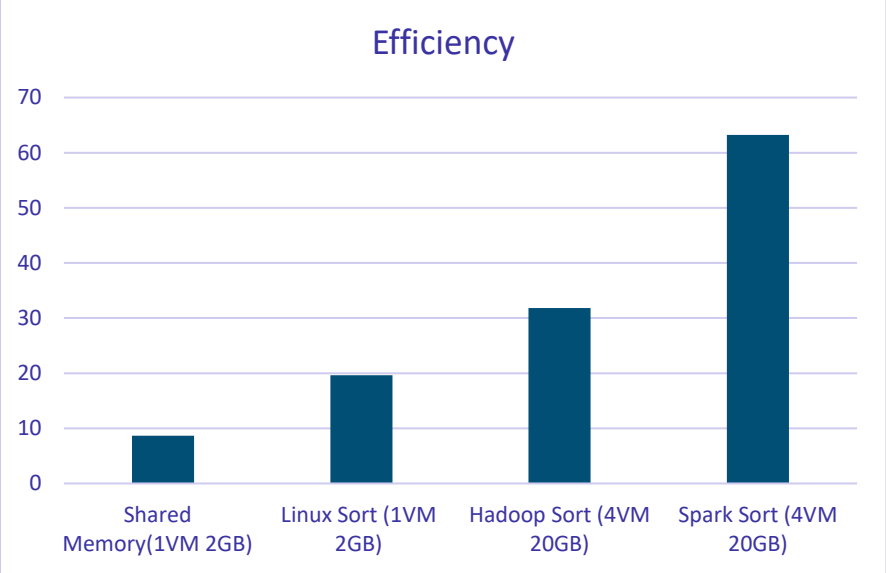

Efficiency
60
50
40
30
20
10
0
Shared Memory(1VM 2GB)
Linux Sort (1VM 2GB)
Hadoop Sort (4VM 8GB)
Spark Sort (4VM 8GB)

2 Analysis *Performance evaluation of sort (strong scaling – large dataset)(* 20 GB*)*

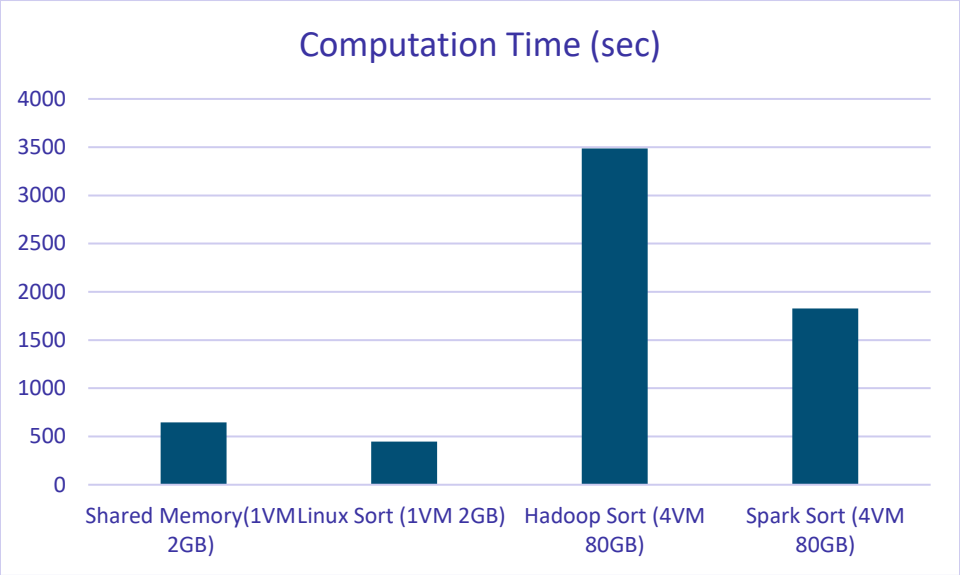


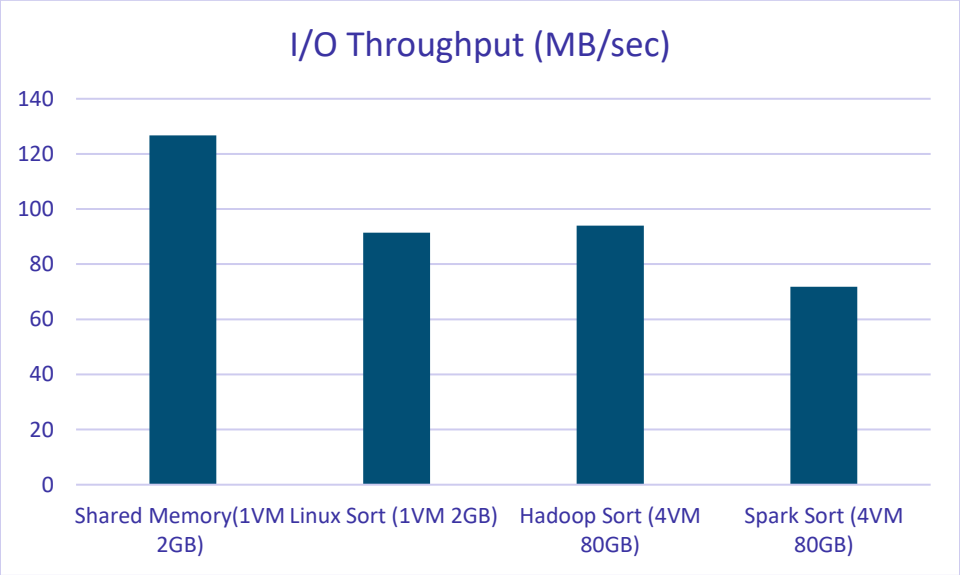


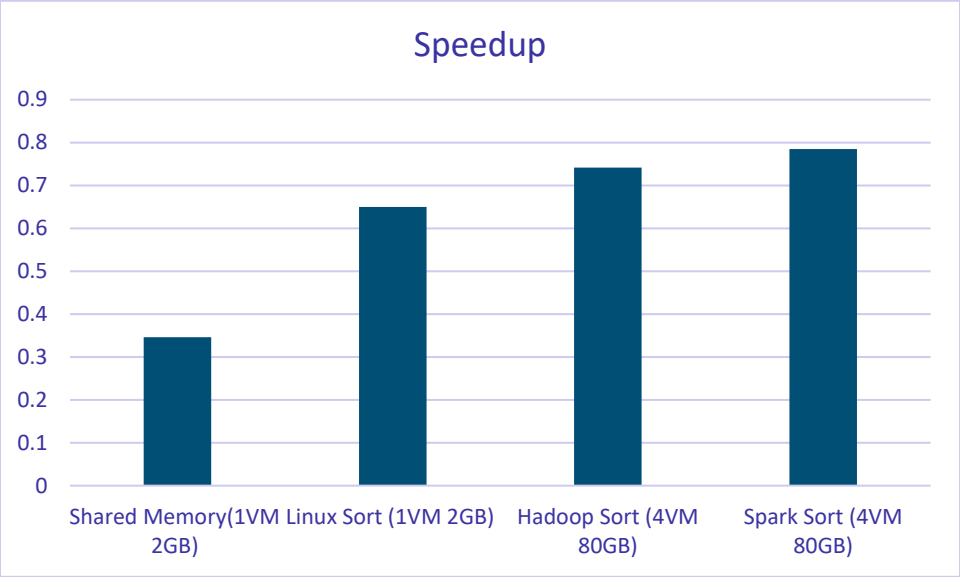

Speedup
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Shared Memory(1VM 2GB)
Linux Sort (1VM 2GB)
Hadoop Sort (4VM 20GB)
Spark Sort (4VM 20GB)

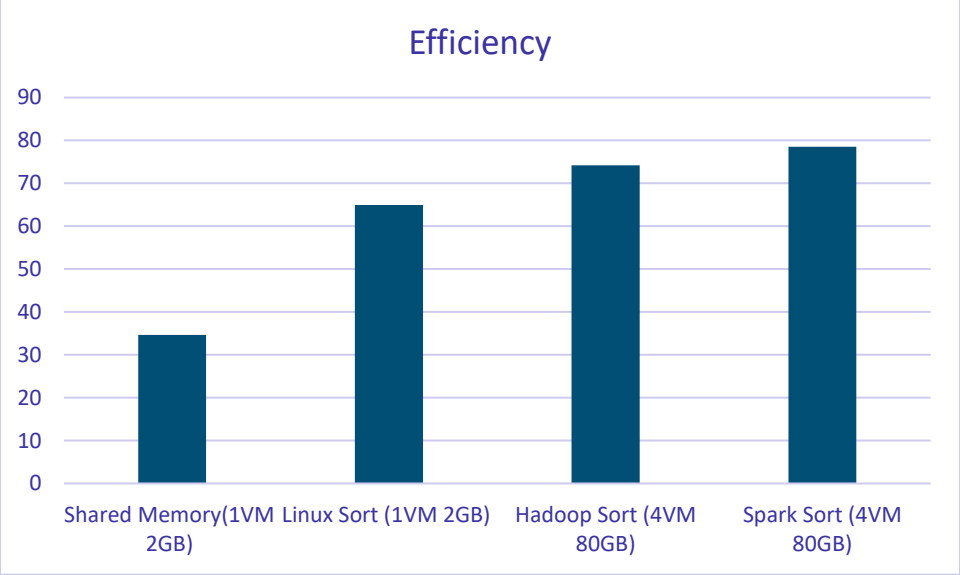

Efficiency
70
60
50
40
30
20
10
0
Shared Memory(1VM 2GB)
Linux Sort (1VM 2GB)
Hadoop Sort (4VM 20GB)
Spark Sort (4VM 20GB)

3 *Performance evaluation of sort (weak scaling – large dataset)80 GB*









Conclusion

Here from the graphs and the experiments we performed we found that Spark is better than Hadoop, as per our study we found that, Hadoop MapReduce, read and write from the disk as a result it slows down the computation. While Spark can run on top of Hadoop and provides a better computational speed solution. Hence, the differences between Apache Spark vs. Hadoop MapReduce shows that Apache Spark is much-advance cluster computing engine than MapReduce. The greater throughput on memory shared is not to be mistaken as a positive over Hadoop and spark. The throughput is higher because we are writing and reading the files multiple times in the memory shared while the spark and Hadoop uses less reads and writes. Thus, when the trough put is calculated as the

amount used in reads and writes divided by the total time taken for the programs to run it does mislead to have higher throughput for the memory shared with respective to other technologies which we have implemented.

Which seems to be best at 1 node scale? How about 4 nodes?

park performs better than Shared Memory or Hadoop for 1 node experiments. If we work with 4 nodes, spark would still perform better as the computations are performed within memory., including the data transfer. For working with 100 and 1000 node scales, Spark works best, as it's build on functional programming language, which works best in the distributed environment, but we have implemented the application in JAVA. The in-memory batch processing and reduction in disk read/write makes spark about 10 to 1000 times faster than MapReduce