CS 553 - Cloud Computing, Spring 2018

PA 2 Part B - Performance Report

Sort on Hadoop & Spark

Saumil Pareshbhai Ajmera | A20397303 | sajmera4@hawk.iit.edu

1. Hadoop Sort for 8GB, 20GB and 80GB data sets

I have implemented the code in Java, where input parameters are input file path, output file path.

Implementation-

1) Here I have implemented my own *mapper* to split data into key value pairs holding first ten bytes of data and value as later bytes of data per line of file and *reducer* class to pass form a each key based output value for writing output in the file.
2) I have used file shards of 64MB (default) and ran with 6 *reducers*. Also I have *compressed* the map output for efficient data retrieval to reducer.
3) The *OutputGroupingValueComparator* will sort the data based on value having same key.
3) So in total 2 read operations and 2 write operations are performed on dataset from disk to memory.

2. Spark Sort for 8GB, 20GB and 80GB data sets

I have implemented the code in Java, where input parameters are input file path, output file path.

Implementation-

1) Here I have used RDD (Resilient Distributed Dataset) to hold the data and performing transformation and actions over the data set by storing its maximum capacity from tha available space in memory.
2) Finally it executes in form of lazy evaluation by forming a Directed Acyclic Graph to know which operations are performed in which sequence.
3) Also I have used memory of 2gb, number of executor as 16 and number of core per executor as 4.
3) So in total 2 read operations and 2 write operations are performed on dataset from disk to memory.

3. Evaluation and Calculation

Hadoop - Output files generated are validate whether they are sorted or not using TeraValidate which is an in built feature of Hadoop and it generate an output file which is obtained in my login node named part-r-0000 which needs to be deleted after each run as hadoop can't overwrite it.

Spark - Output files generated are validate whether they are sorted or not using TeraValidate which is an in built feature of Hadoop and it generate an output file which is obtained in my login node named part-r-0000 which needs to be deleted after each run as hadoop can't overwrite it.

Formula for calculating SpeedUp and Efficiency-

Speedup = Ts/Tp
Ts = serial execution time
Tp = parallel execution time

Efficiency = S/p
S = speedup
p = number of node

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 (sec)	43.77	20	248.402	220.69
Data Read (GB)	6	2	16	16
Data Write (GB)	6	2	16	16
I/O Throughput (MB/sec)	274.160	200	128.82	144.98
SpeedUp	NA	NA	0.18 (Shared) 0.08 (Linux)	0.19 (Shared) 0.09 (Linux)
Efficiency	NA	NA	0.04 (Shared) 0.02 (Linux)	0.04 (Shared) 0.02 (Linux)

Observation- As this is a weak scaling example performed on small dataset, by its definition depicts that as the number of resources increases the total workload also increases. Therefore each resource will have same amount of workload as it was initially so time should remain constant for particular resource, and total time also should remain constant as work will be done parallely. So time to sort data using Hadoop and Spark having 4VM but data is also quadrupled from Shared Memory and Linux sort i.e. 8GB so it is obvious that time also will increase in practical scenario according to Gustafson's law of speedup.

Below are the charts for plotted for the above table comparing the experimental results obtained.

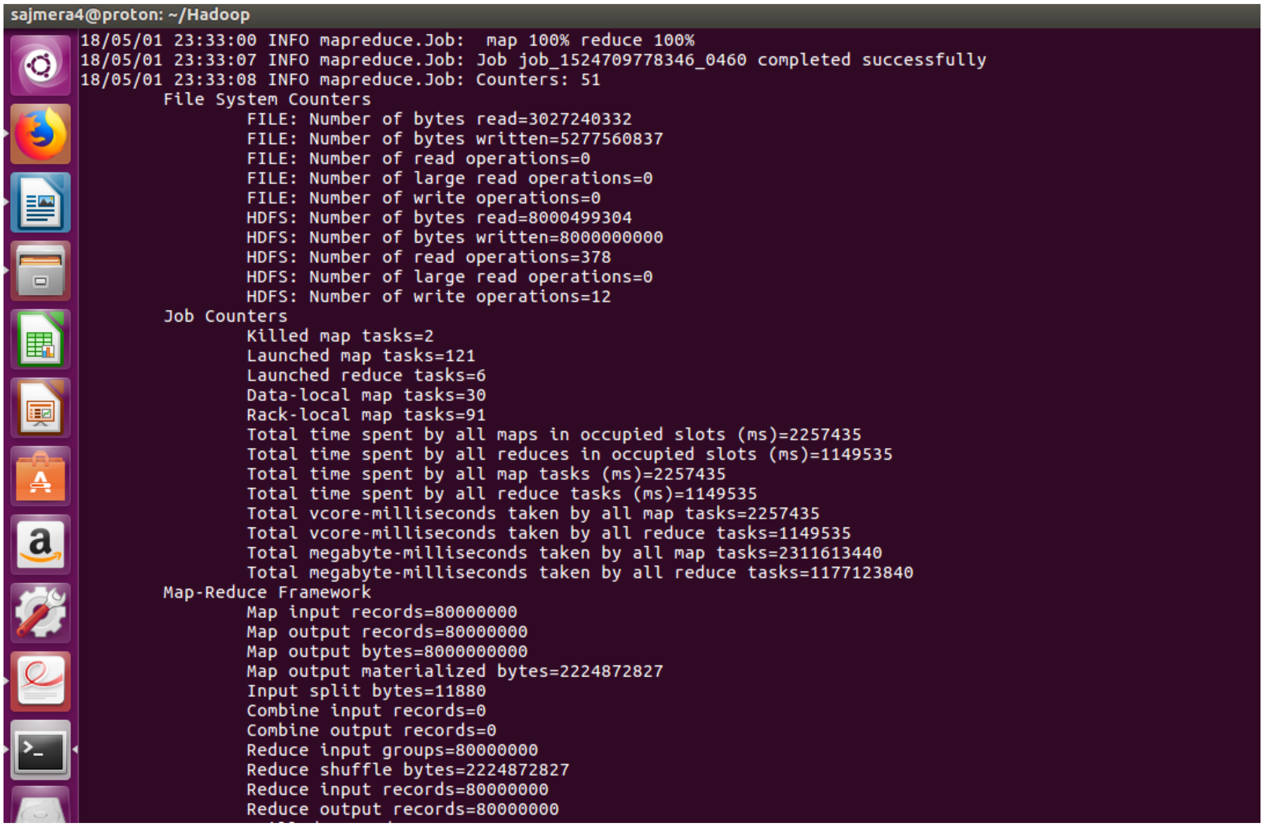


Figure 1: Hadoop 8GB Sorting

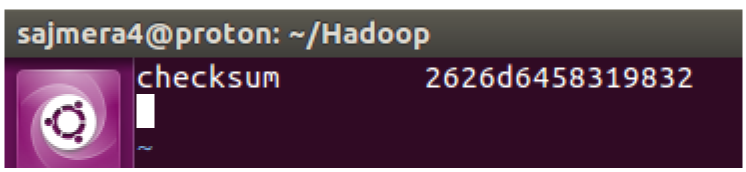


Figure 2: Hadoop 8GB TeraValidate Output

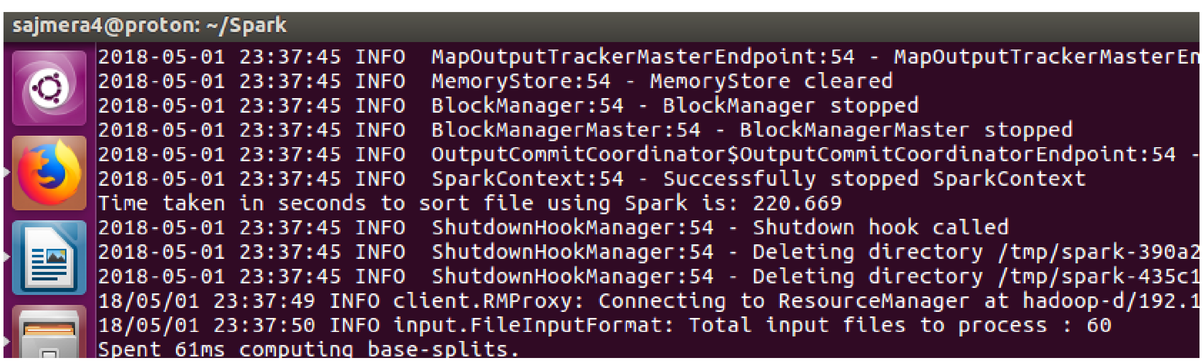


Figure 3: Spark 8GB Sorting

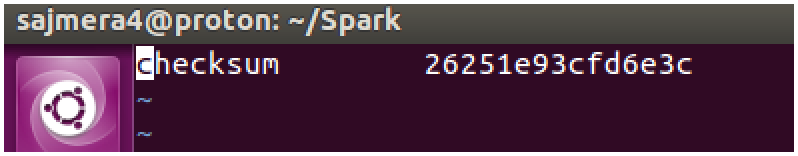
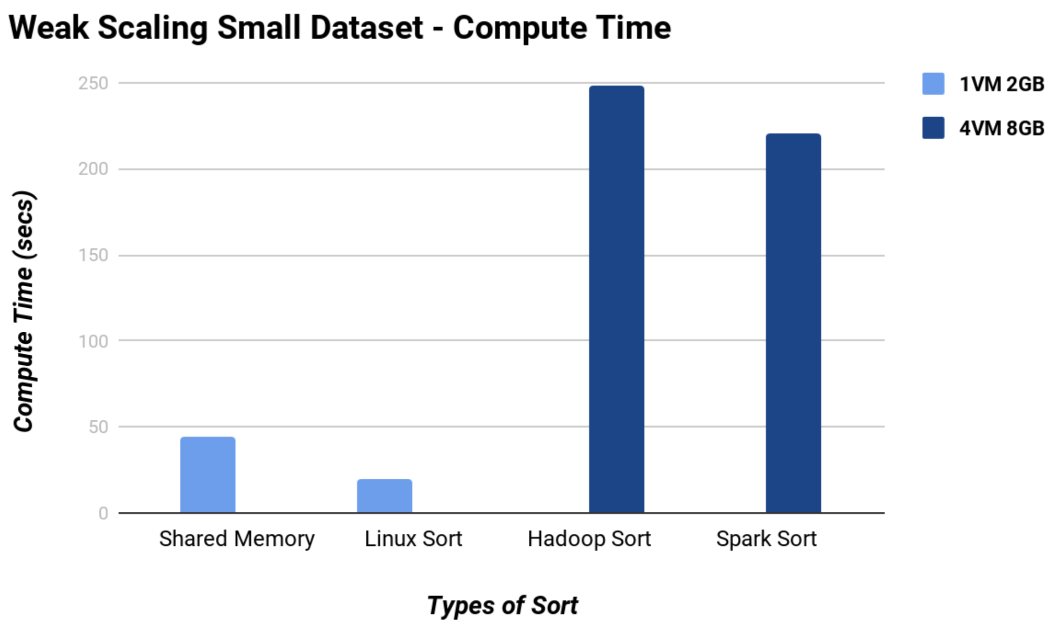
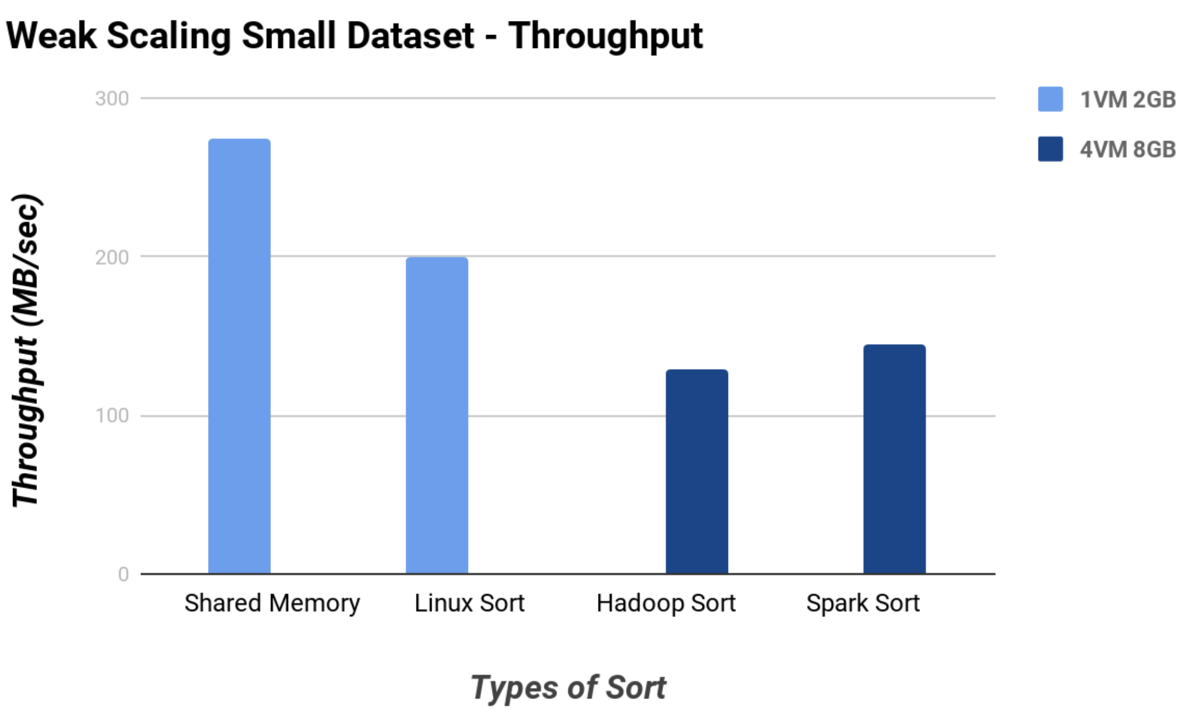
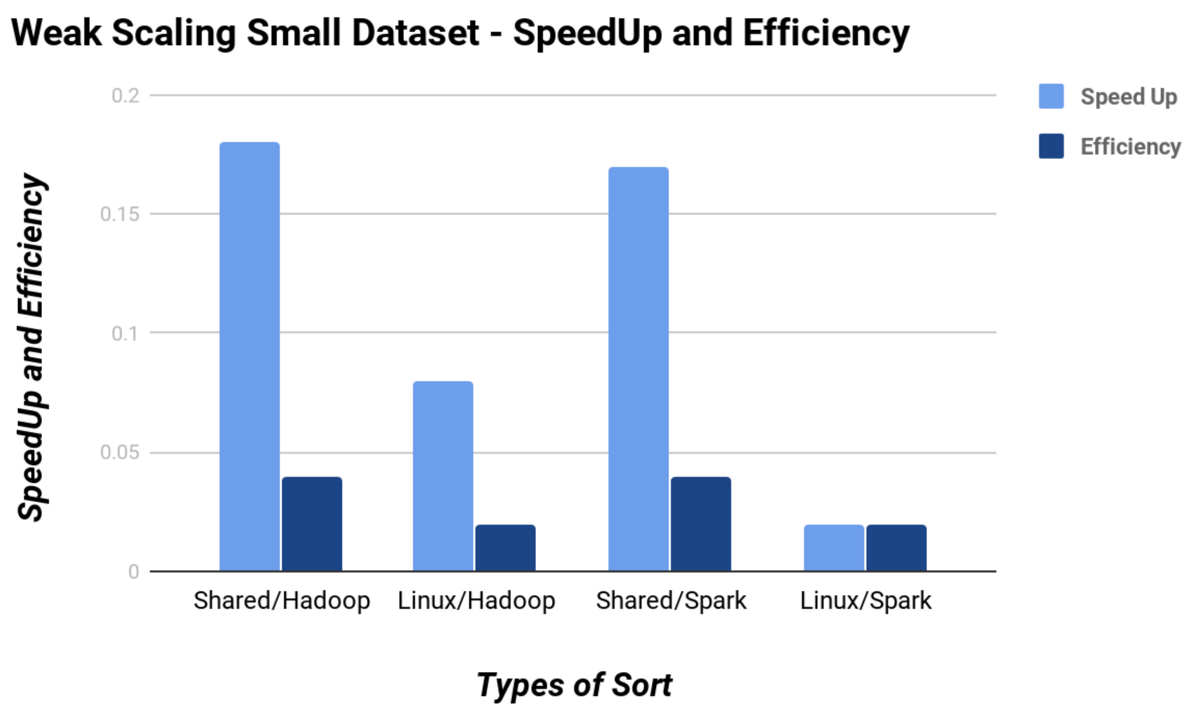


Figure 4: Spark 8GB TeraValidate Output







Conclusion- So we can see that we are not able to achieve high speed up and efficiency and ideally time must remain constant in weak scaling but in our case time also decreases, for both cases I can say that increasing workload on increasing resources leads to this result.

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

Experiment	**Shared Memory (1VM 20GB)**	**Linux Sort (1VM 20GB)**	**Hadoop Sort (4VM 20GB)**	**Spark Sort (4VM 20GB)**
Computation (sec)	822.809	401.34	601.57	308.08
Data Read (GB)	60	40	40	40
Data Write (GB)	60	40	40	40
I/O Throughput (MB/sec)	145.842	199.332	132.25	259.67
SpeedUp	NA	NA	1.36 (Shared) 0.66 (Linux)	2.67 (Shared) 1.30 (Linux)
Efficiency	NA	NA	0.34 (Shared) 0.16 (Linux)	0.66 (Shared) 0.32 (Linux)

Observation- As this is a strong scaling example, by its definition depicts that as the number of resources increases the total workload remains fixed. Therefore work gets divided as number of resources increases but there is one barrier beyond which we cannot achieve higher speedup practically as there will be some serialized part in program which can't be parallelized and also time synchronization between parallel process will add an extra overhead.

Here, we I am not getting linear speed up due to some of the slave completes its task early the other and waiting ideal for other. So speed up in not learner.

So time to sort data using Hadoop and Spark having 4VM but data with same data size as compared to Shared Memory and Linux sort i.e. 20GB should decrease significantly as per the Amdahl's law of speedup.

Below are the charts for plotted for the above table comparing the experimental results obtained.

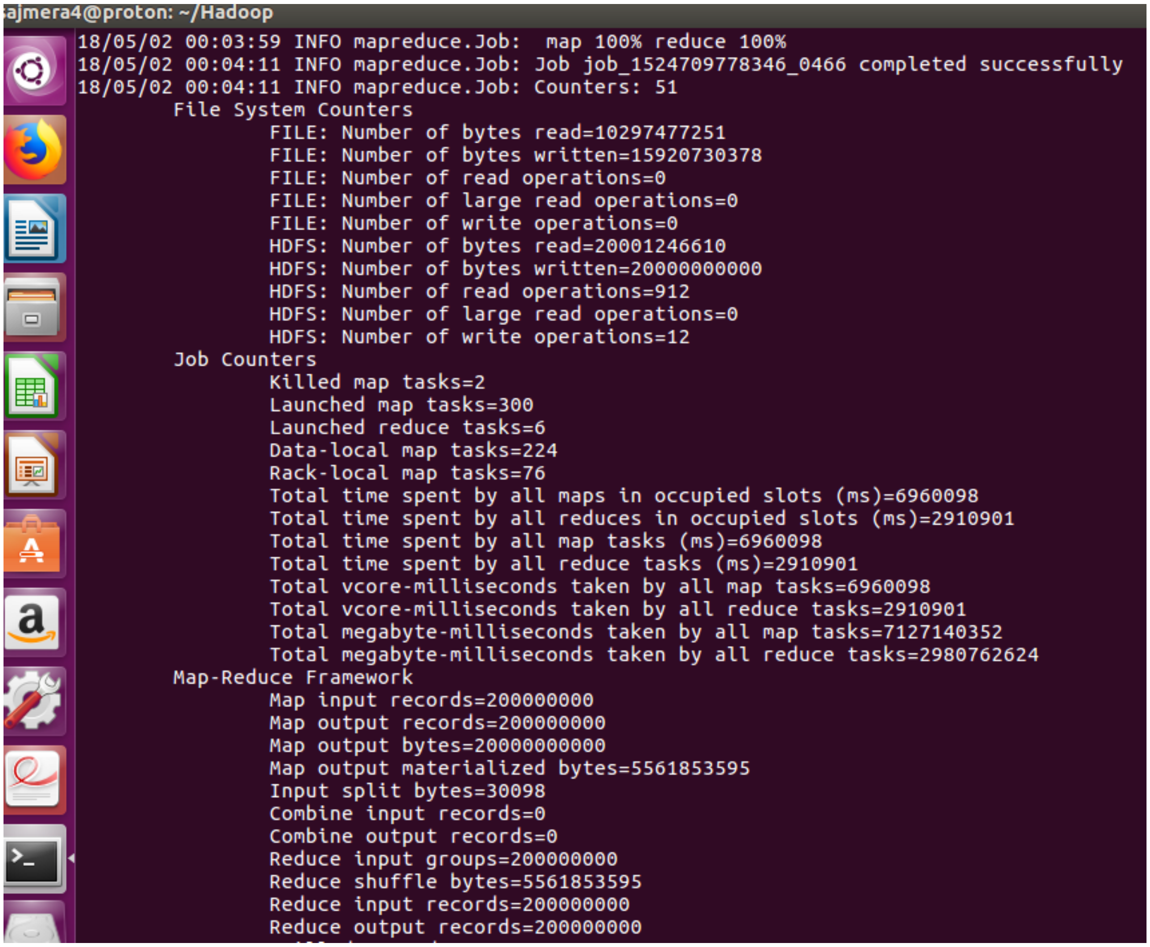


Figure 5: Hadoop 20GB Sorting

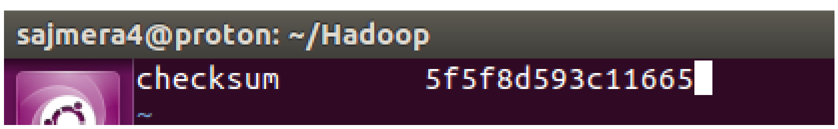


Figure 6: Hadoop 20GB TeraValidate Output

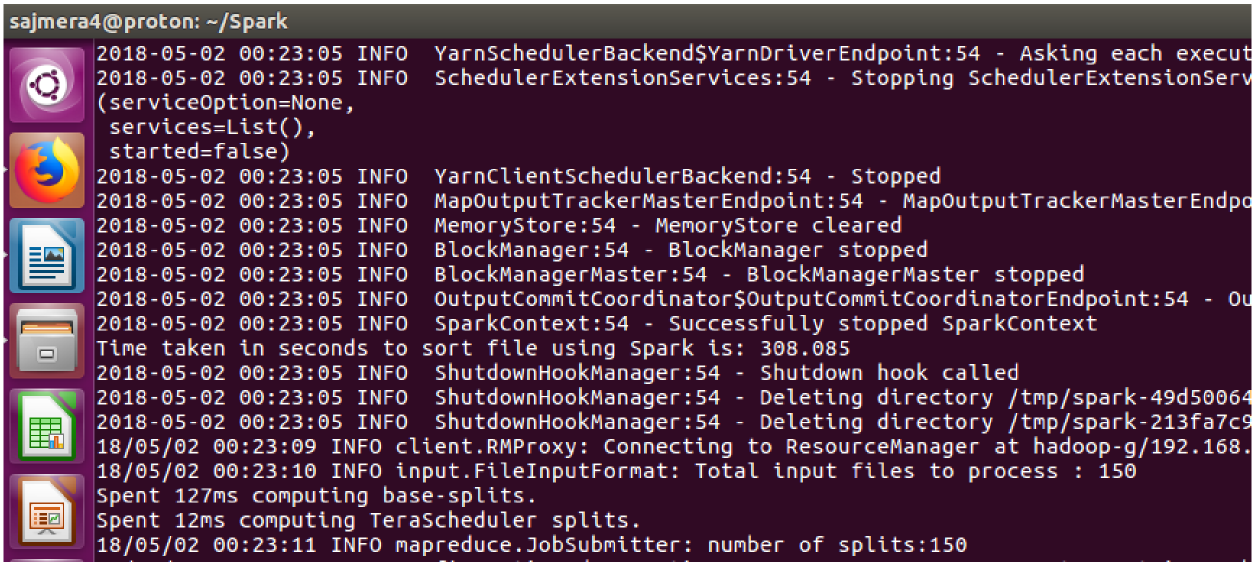


Figure 7: Spark 20GB Sorting

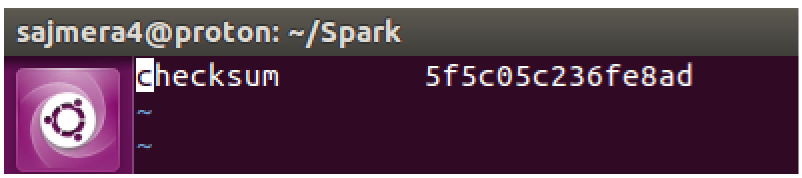
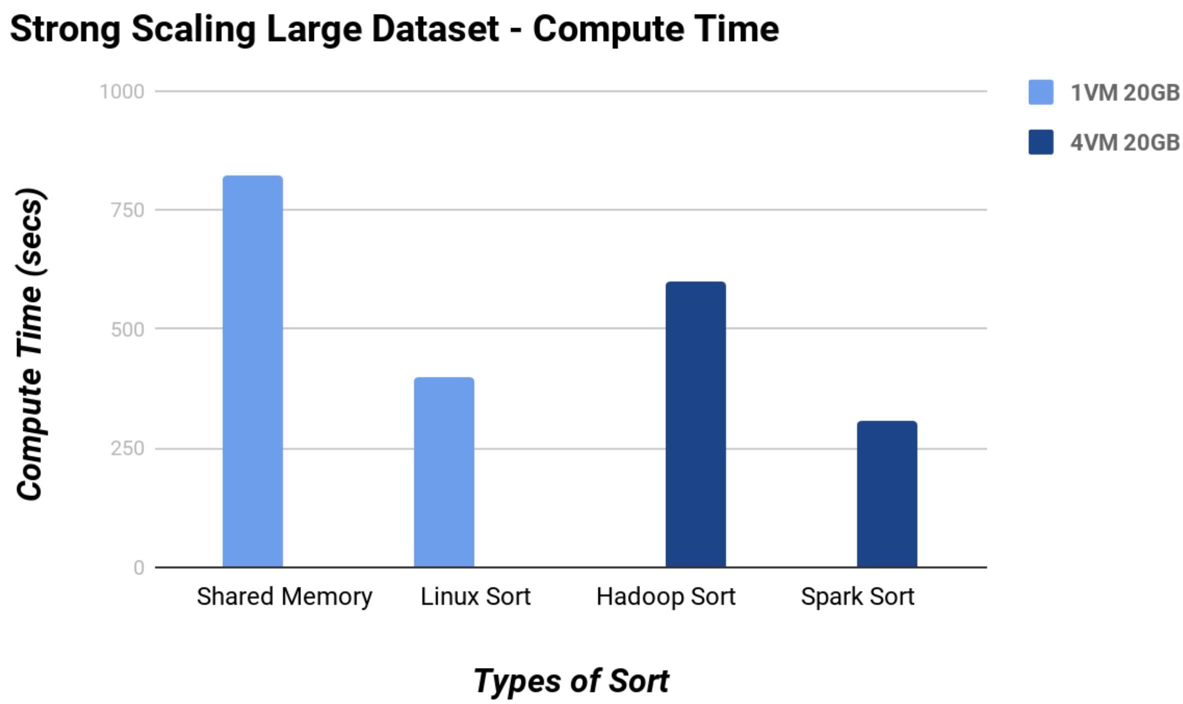
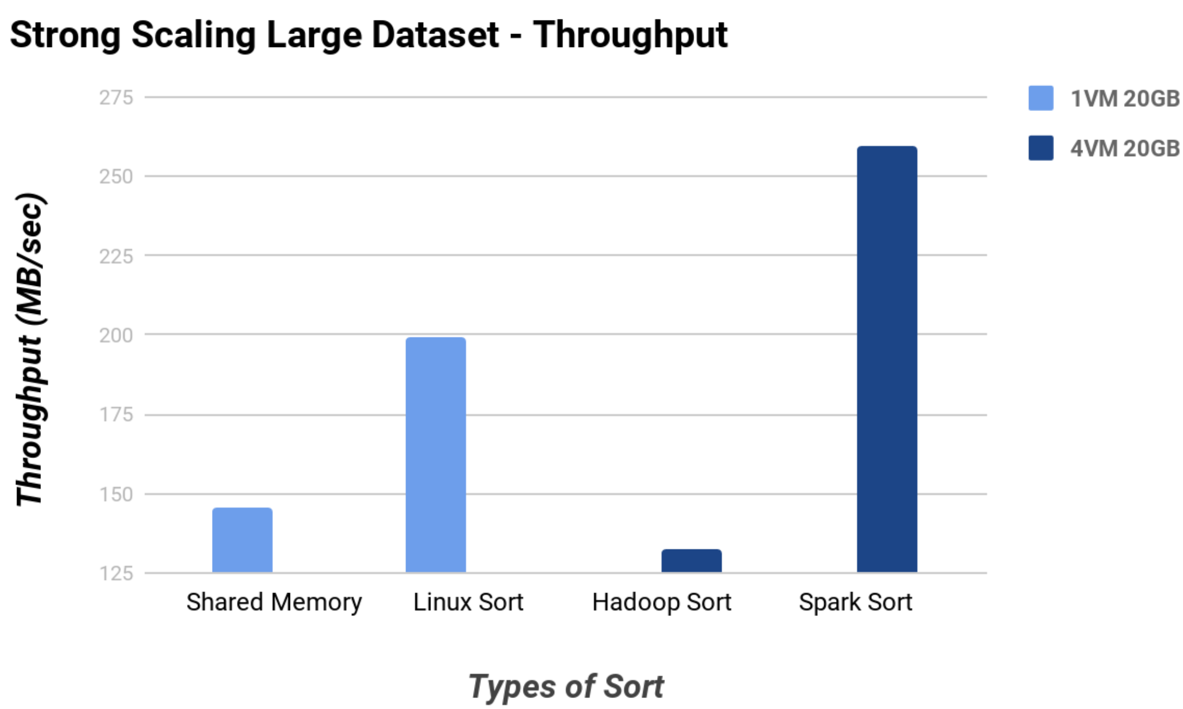
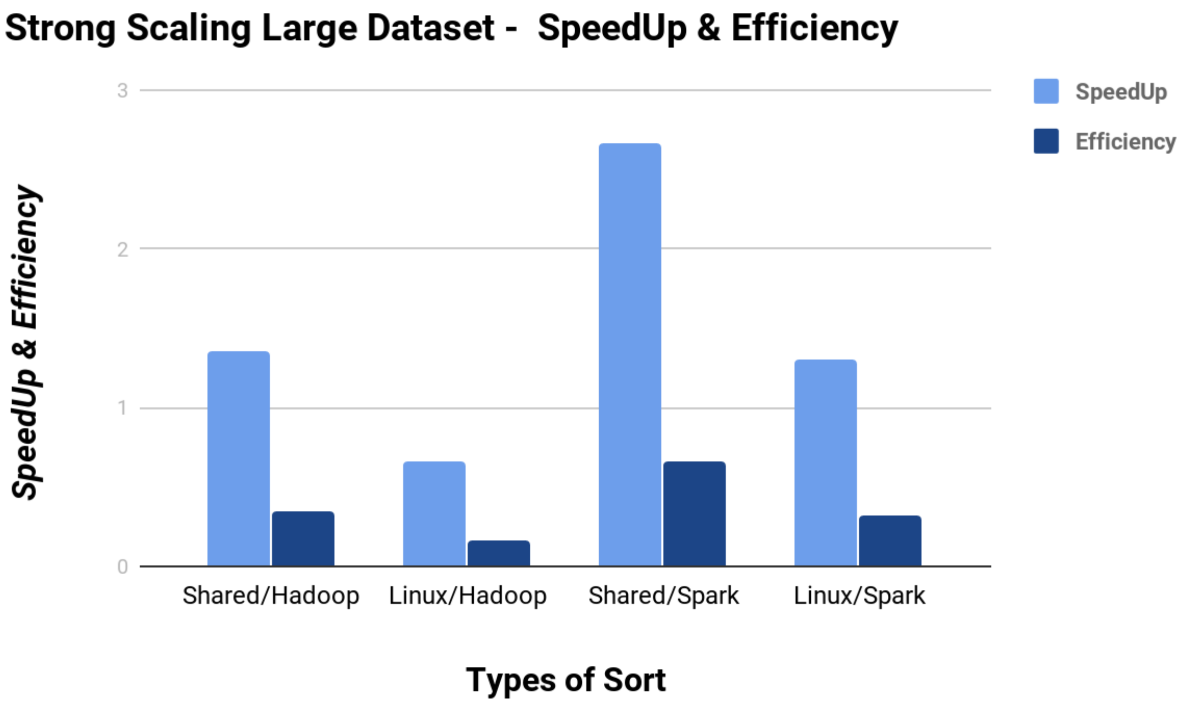


Figure 8: Spark 20GB TeraValidate Output


Strong Scaling Large Dataset - Compute Time
1000
750
500
250
0
Compute Time (secs)
Shared Memory
Linux Sort
Hadoop Sort
Spark Sort
Types of Sort
1VM 20GB
4VM 20GB


Strong Scaling Large Dataset - Throughput
275
250
225
200
175
150
125
Throughput (MB/sec)
Shared Memory
Linux Sort
Hadoop Sort
Spark Sort
Types of Sort
1VM 20GB
4VM 20GB



Conclusion- So we can see that we are able to achieve high speed up and efficiency and ideally time must decreases linearly in strong scaling but in our case time decreases but not linearly due to some overheads and physical machine capabilities which is not ideal, for both cases I can say that keeping constant workload on increasing resources leads to this result.

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

Experiment	Shared Memory (1VM 20GB)	Linux Sort (1VM 20GB)	Hadoop Sort (4VM 80GB)	Spark Sort (4VM 80GB)
Computation (sec)	822.809	401.34	2312.498	1342.34
Data Read (GB)	60	40	160	160
Data Write (GB)	60	40	160	160
I/O Throughput (MB/sec)	145.842	199.332	138.37	238.20
SpeedUp	NA	NA	0.35 (Shared) 0.17 (Linux)	0.61 (Shared) 0.29 (Linux)
Efficiency	NA	NA	0.08 (Shared) 0.04 (Linux)	0.15 (Shared) 0.07 (Linux)

Observation- As this is a weak scaling example performed on large dataset, by its definition depicts that as the number of resources increases the total workload also increases. Therefore each resource will have same amount of workload as it was initially so time should remain constant for particular resource, and total time also should remain constant as work will be done parallely.

So time to sort data using Hadoop and Spark having 4VM but data is also quadrupled from Shared Memory and Linux sort i.e. 80GB so it is obvious that time also will increase in practical scenario according to Gustafson's law of speedup.

Below are the charts for plotted for the above table comparing the experimental results obtained.

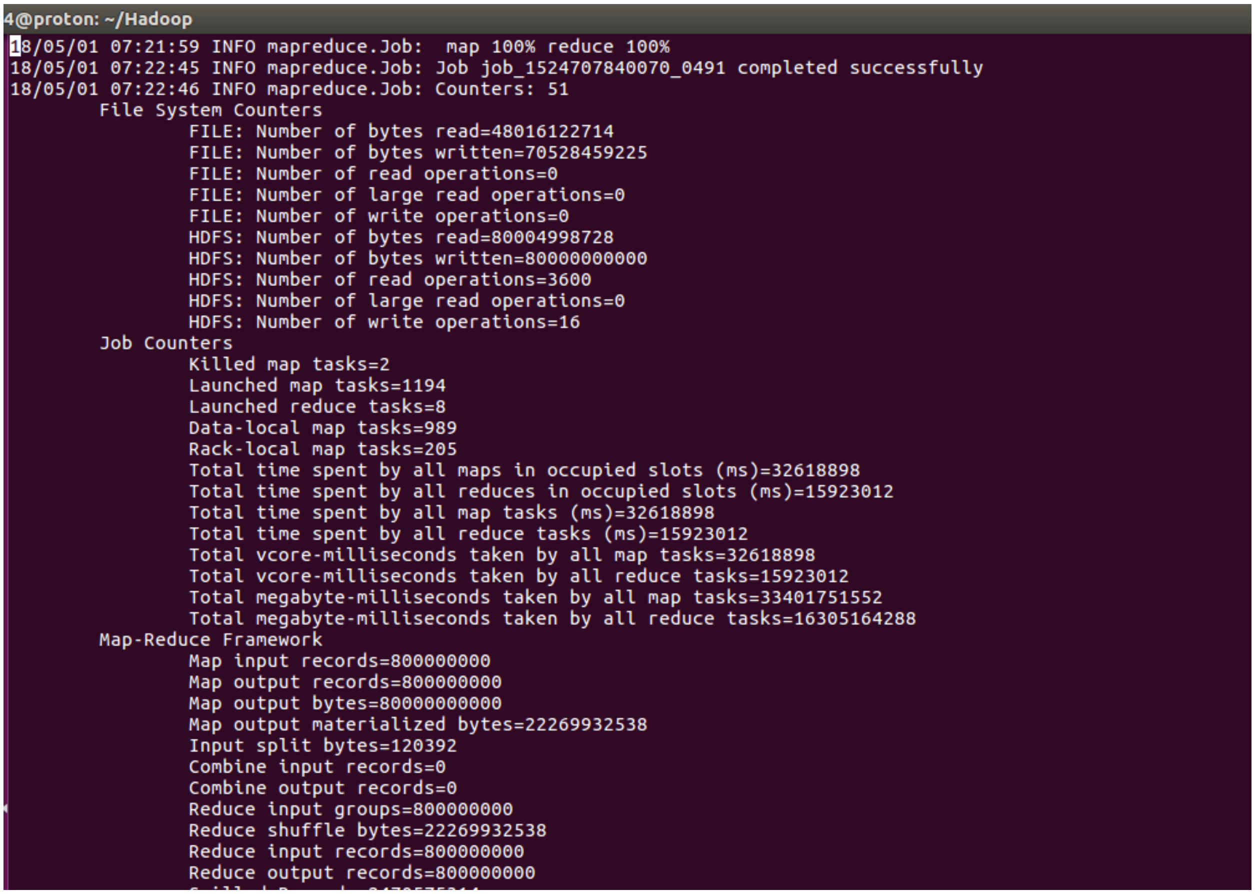


Figure 9: Hadoop 80GB Sorting

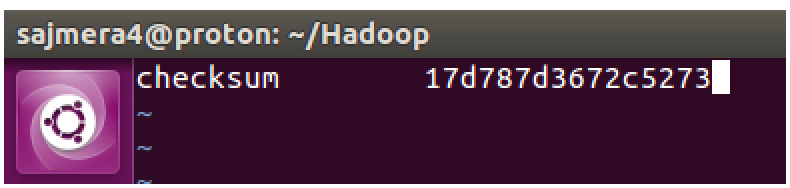


Figure 10: Hadoop 80GB TeraValidate Output

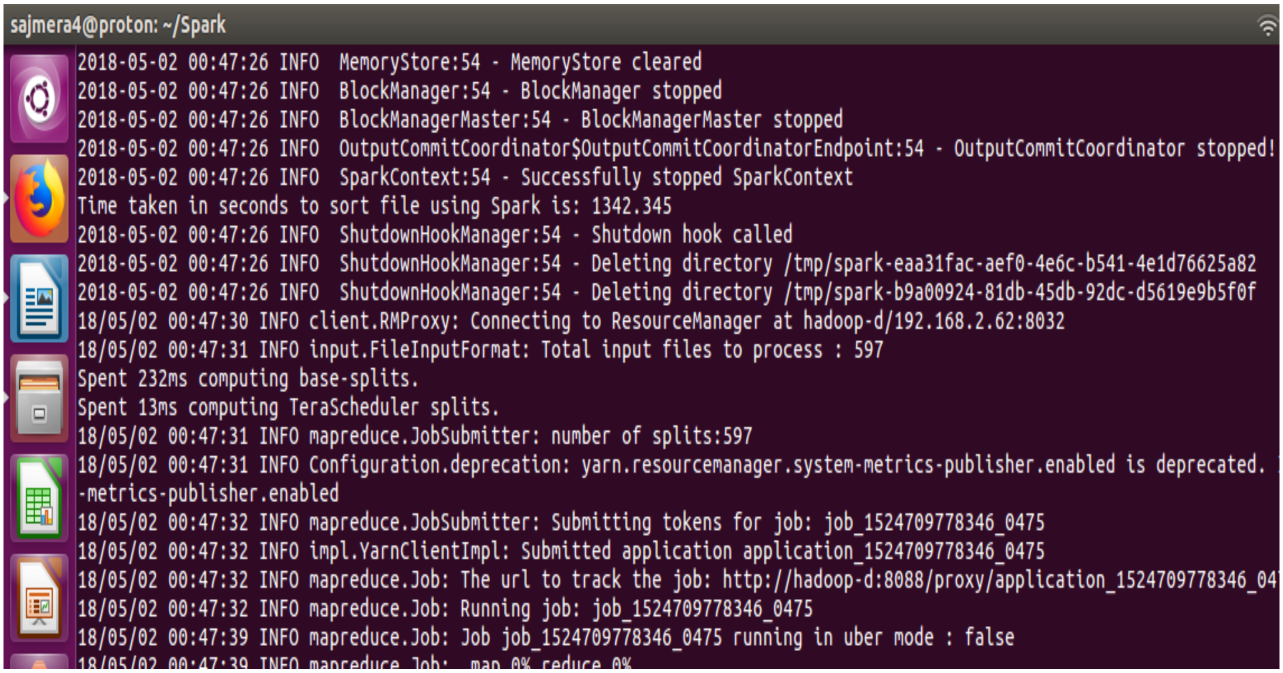


Figure 11: Spark 80GB Sorting

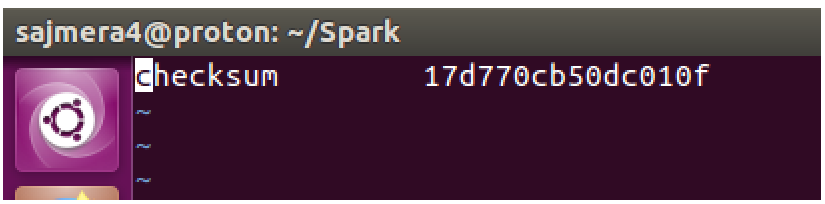
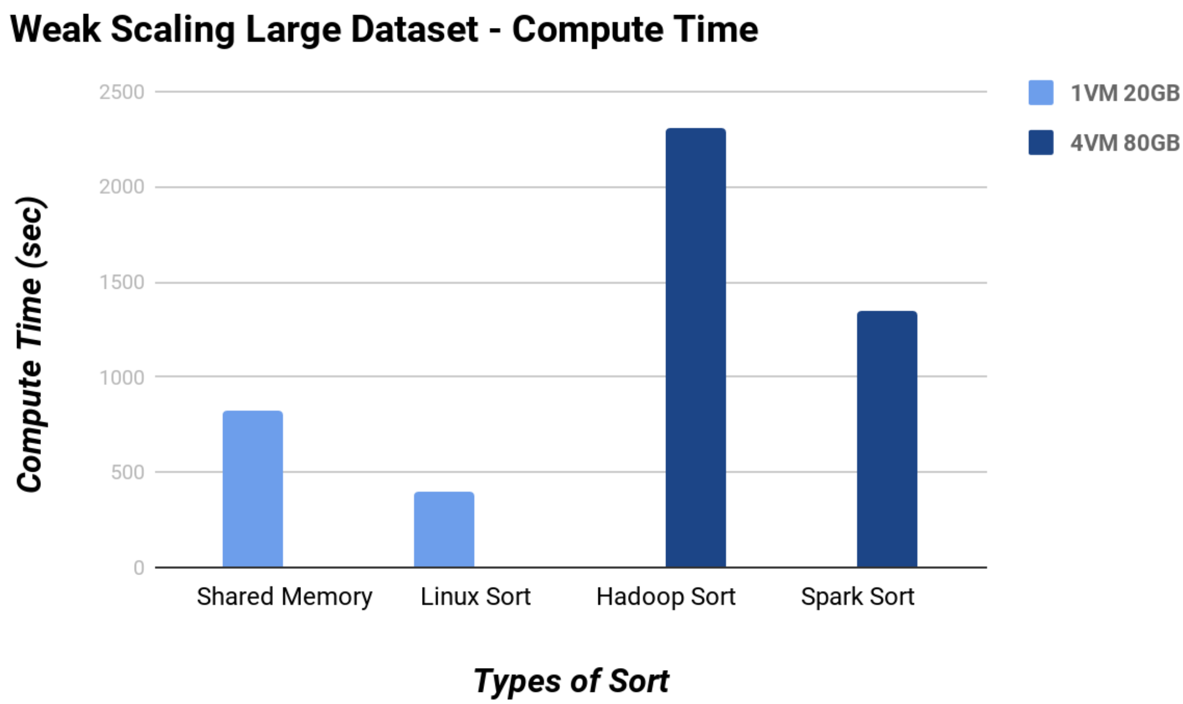
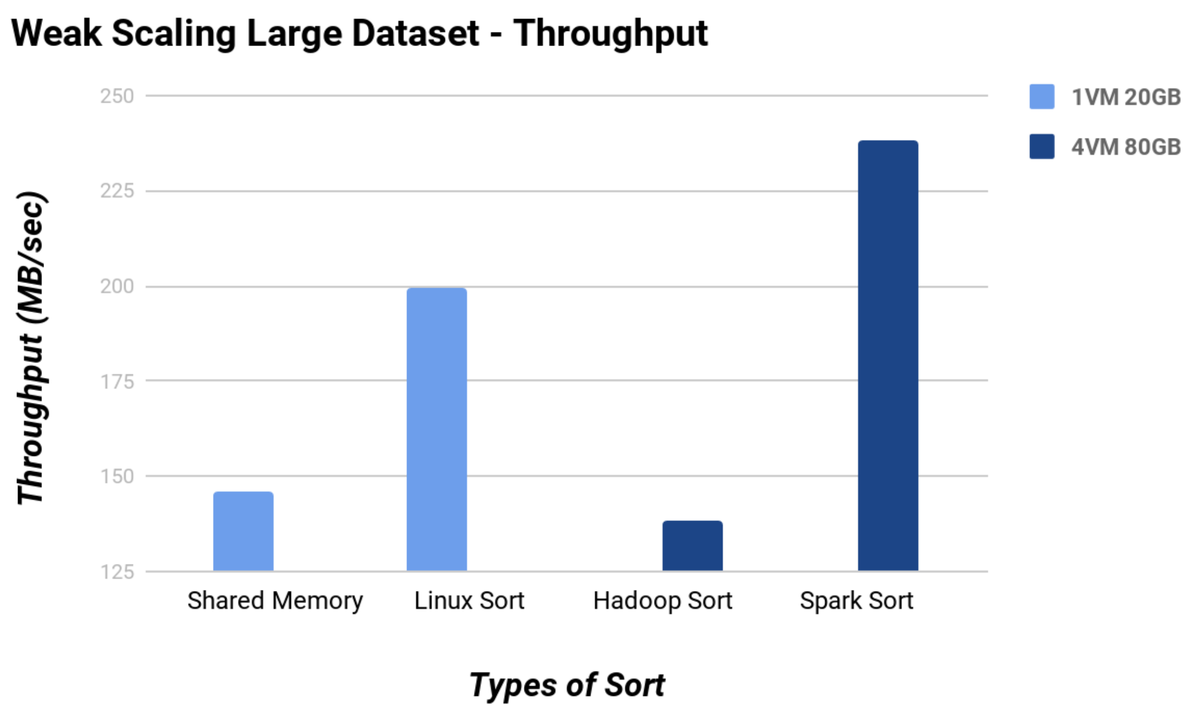
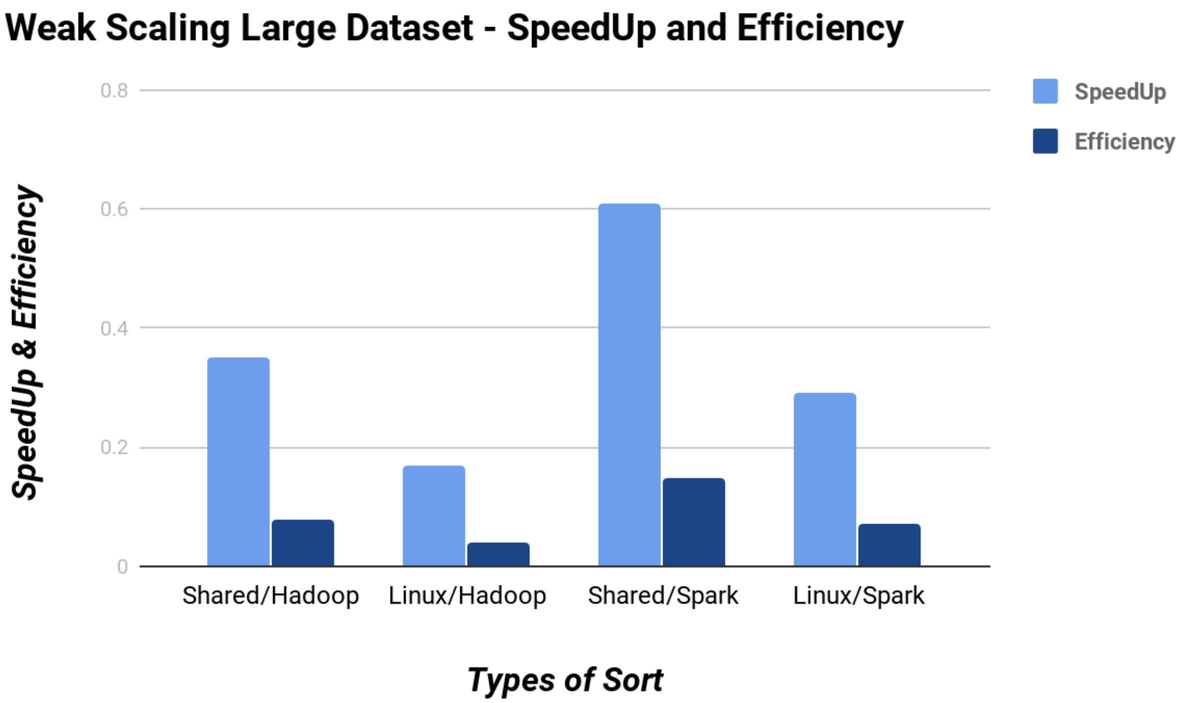


Figure 12: Spark 80GB TeraValidate Output


Weak Scaling Large Dataset - Compute Time
1VM 20GB
4VM 80GB
Compute Time (sec)
2500
2000
1500
1000
500
0
Shared Memory
Linux Sort
Hadoop Sort
Spark Sort
Types of Sort


Weak Scaling Large Dataset - Throughput
1VM 20GB
4VM 80GB
Throughput (MB/sec)
250
225
200
175
150
125
Shared Memory
Linux Sort
Hadoop Sort
Spark Sort
Types of Sort



Conclusion- So we can see that we are not able to achieve high speed up and efficiency and ideally time must remain constant in weak scaling but in our case time also decreases, for both cases I can say that increasing workload on increasing resources leads to this result.

Comparison with Sort Benchmarks winner in 2013 and 2014 (Hadoop and Spark)-

2013 Winner - Hadoop with 102.5 TB in 4,328 seconds 2100 nodes x (2 2.3Ghz hex core Xeon E5-2630, 64GB memory, 12x3TB disks)

2014 Winner - Apache Spark with 100 TB in 1,406 seconds 207 Amazon EC2 i2.8xlarge nodes x (32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD)

These are winners for sorting terabytes of data on distributed system scaled for achieving high throughput and minimize time with high I/O intensive application. With my test data of 80 GB and 4 Intel Skylake processors, 16GB memory, 80GB of SSD disk storage (data nodes), I have tried my best approach to match my output time and throughput with above result. But as of now result will be low compared to that as many factors comes into account like hardware type, code implementation and network bandwidth.

Final Conclusion-

So from this experiments I can conclude that if we have I/O intensive application then Hadoop is better and if we have memory intensive application then Spark very good choice.

There is significance visibility of strong scaling and weak scaling effect in my result which is as per theoretical explanation given by the Professor in the lecture.

1) For 1 node scale linux sort is the best sorting platform provided by linux.
2) For 4 nodes I achieved higher efficiency in Spark as compared to Hadoop as Spark does lazy evaluation and has less overhead than Hadoop’s map and reduce jobs.
3) But for 100 and 1000 scale, I can say that both will achieve higher speedup and efficiency as they are build for such application, system resiliency required, which can scale upto much higher resources. But we need to take into care the type of application as Hadoop works well on parallel processing application in which no interaction is required between one job’s output and one job’s input while Spark works well for iterative and machine learning application which Hadoop does not support.
4) From my point of view Spark works well for 100 and 1000 nodes due to its Resilient Distributed Dataset storage which faster than disk I/O operation and lazy evaluation technique which doesn't wait for other job worker’s task for their completion.