

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 the 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-

$$\text{Speedup} = T_s / T_p$$

T_s = serial execution time

T_p = parallel execution time

$$\text{Efficiency} = S / p$$

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 parallelly. 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.

```

sajmera4@proton: ~/Hadoop
18/05/01 23:33:00 INFO mapreduce.Job: map 100% reduce 100%
18/05/01 23:33:07 INFO mapreduce.Job: Job job_1524709778346_0460 completed successfully
18/05/01 23:33:08 INFO mapreduce.Job: Counters: 51
File System Counters
  FILE: Number of bytes read=3027240332
  FILE: Number of bytes written=5277560837
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=8000499304
  HDFS: Number of bytes written=8000000000
  HDFS: Number of read operations=378
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=12
Job Counters
  Killed map tasks=2
  Launched map tasks=121
  Launched reduce tasks=6
  Data-local map tasks=30
  Rack-local map tasks=91
  Total time spent by all maps in occupied slots (ms)=2257435
  Total time spent by all reduces in occupied slots (ms)=1149535
  Total time spent by all map tasks (ms)=2257435
  Total time spent by all reduce tasks (ms)=1149535
  Total vcore-milliseconds taken by all map tasks=2257435
  Total vcore-milliseconds taken by all reduce tasks=1149535
  Total megabyte-milliseconds taken by all map tasks=2311613440
  Total megabyte-milliseconds taken by all reduce tasks=1177123840
Map-Reduce Framework
  Map input records=800000000
  Map output records=800000000
  Map output bytes=8000000000
  Map output materialized bytes=2224872827
  Input split bytes=11880
  Combine input records=0
  Combine output records=0
  Reduce input groups=80000000
  Reduce shuffle bytes=2224872827
  Reduce input records=800000000
  Reduce output records=800000000

```

Figure 1: Hadoop 8GB Sorting

```

sajmera4@proton: ~/Hadoop
checksum 2626d6458319832

```

Figure 2: Hadoop 8GB TeraValidate Output

```

sajmera4@proton: ~/Spark
2018-05-01 23:37:45 INFO MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint:54
2018-05-01 23:37:45 INFO MemoryStore:54 - MemoryStore cleared
2018-05-01 23:37:45 INFO BlockManager:54 - BlockManager stopped
2018-05-01 23:37:45 INFO BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-01 23:37:45 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54
2018-05-01 23:37:45 INFO SparkContext:54 - Successfully stopped SparkContext
Time taken in seconds to sort file using Spark is: 220.669
2018-05-01 23:37:45 INFO ShutdownHookManager:54 - Shutdown hook called
2018-05-01 23:37:45 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-390a2
2018-05-01 23:37:45 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-435c1
18/05/01 23:37:49 INFO client.RMProxy: Connecting to ResourceManager at hadoop-d/192.1
18/05/01 23:37:50 INFO input.FileInputFormat: Total input files to process : 60
Spent 61ms computing base-splits.

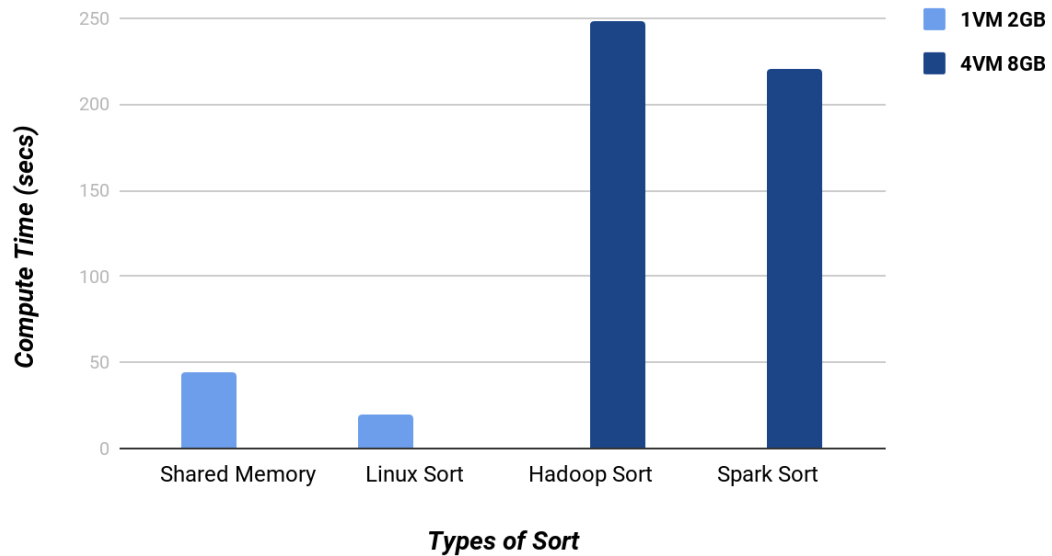
```

Figure 3: Spark 8GB Sorting

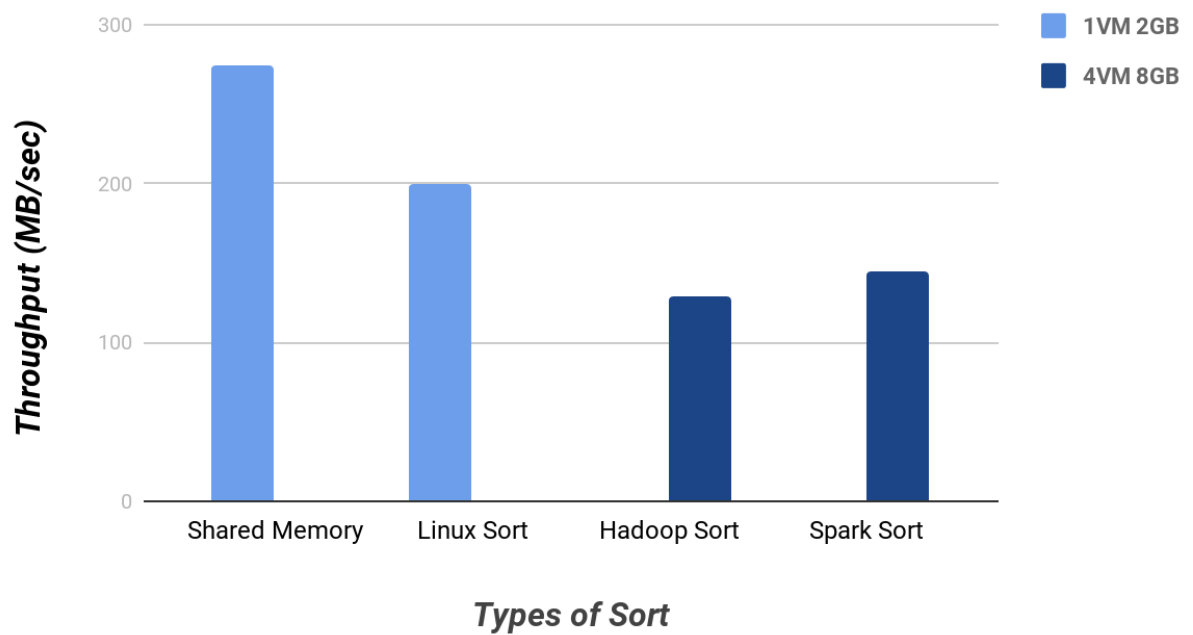
```
sajmera4@proton: ~/Spark
checksum 26251e93cfd6e3c
```

Figure 4: Spark 8GB TeraValidate Output

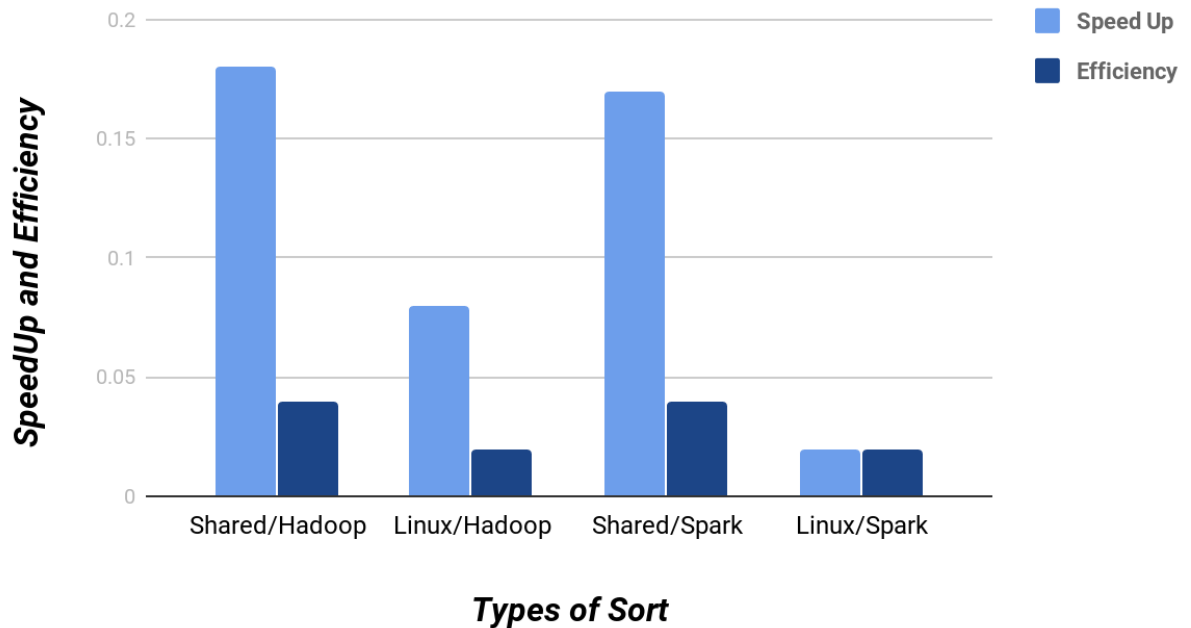
Weak Scaling Small Dataset - Compute Time



Weak Scaling Small Dataset - Throughput



Weak Scaling Small Dataset - SpeedUp and Efficiency



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.

```
sajmera4@proton: ~/Hadoop
18/05/02 00:03:59 INFO mapreduce.Job: map 100% reduce 100%
18/05/02 00:04:11 INFO mapreduce.Job: Job job_1524709778346_0466 completed successfully
18/05/02 00:04:11 INFO mapreduce.Job: Counters: 51
  File System Counters
    FILE: Number of bytes read=10297477251
    FILE: Number of bytes written=15920730378
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=20001246610
    HDFS: Number of bytes written=20000000000
    HDFS: Number of read operations=912
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=12
  Job Counters
    Killed map tasks=2
    Launched map tasks=300
    Launched reduce tasks=6
    Data-local map tasks=224
    Rack-local map tasks=76
    Total time spent by all maps in occupied slots (ms)=6960098
    Total time spent by all reduces in occupied slots (ms)=2910901
    Total time spent by all map tasks (ms)=6960098
    Total time spent by all reduce tasks (ms)=2910901
    Total vcore-milliseconds taken by all map tasks=6960098
    Total vcore-milliseconds taken by all reduce tasks=2910901
    Total megabyte-milliseconds taken by all map tasks=7127140352
    Total megabyte-milliseconds taken by all reduce tasks=2980762624
  Map-Reduce Framework
    Map input records=200000000
    Map output records=200000000
    Map output bytes=20000000000
    Map output materialized bytes=5561853595
    Input split bytes=30098
    Combine input records=0
    Combine output records=0
    Reduce input groups=200000000
    Reduce shuffle bytes=5561853595
    Reduce input records=200000000
    Reduce output records=200000000
```

Figure 5: Hadoop 20GB Sorting

```
sajmera4@proton: ~/Hadoop
checksum 5f5f8d593c11665
```

Figure 6: Hadoop 20GB TeraValidate Output

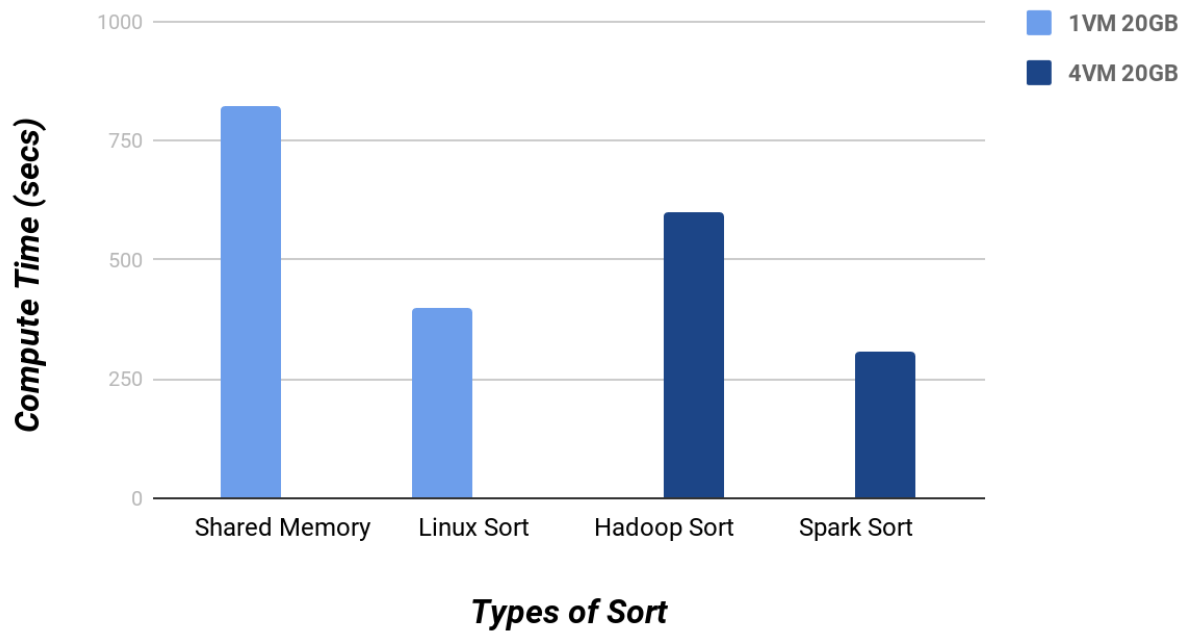

```
sajmera4@proton: ~/Spark
2018-05-02 00:23:05 INFO YarnSchedulerBackend$YarnDriverEndpoint:54 - Asking each execut
2018-05-02 00:23:05 INFO SchedulerExtensionServices:54 - Stopping SchedulerExtensionServ
(serviceOption=None,
services=List(),
started=false)
2018-05-02 00:23:05 INFO YarnClientSchedulerBackend:54 - Stopped
2018-05-02 00:23:05 INFO MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpo
2018-05-02 00:23:05 INFO MemoryStore:54 - MemoryStore cleared
2018-05-02 00:23:05 INFO BlockManager:54 - BlockManager stopped
2018-05-02 00:23:05 INFO BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-02 00:23:05 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - Ou
2018-05-02 00:23:05 INFO SparkContext:54 - Successfully stopped SparkContext
Time taken in seconds to sort file using Spark is: 308.085
2018-05-02 00:23:05 INFO ShutdownHookManager:54 - Shutdown hook called
2018-05-02 00:23:05 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-49d50064
2018-05-02 00:23:05 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-213fa7c9
18/05/02 00:23:09 INFO client.RMPProxy: Connecting to ResourceManager at hadoop-g/192.168.
18/05/02 00:23:10 INFO input.FileInputFormat: Total input files to process : 150
Spent 127ms computing base-splits.
Spent 12ms computing TeraScheduler splits.
18/05/02 00:23:11 INFO mapreduce.JobSubmitter: number of splits:150
```

Figure 7: Spark 20GB Sorting

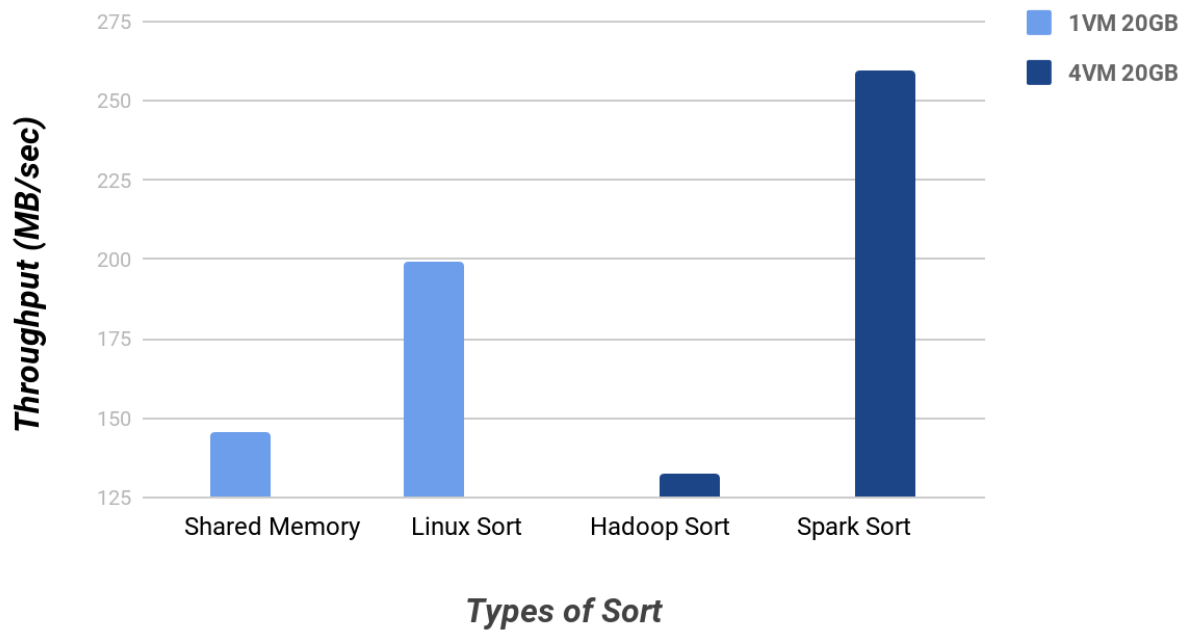
```
sajmera4@proton: ~/Spark
checksum 5f5c05c236fe8ad
~
~
```

Figure 8: Spark 20GB TeraValidate Output

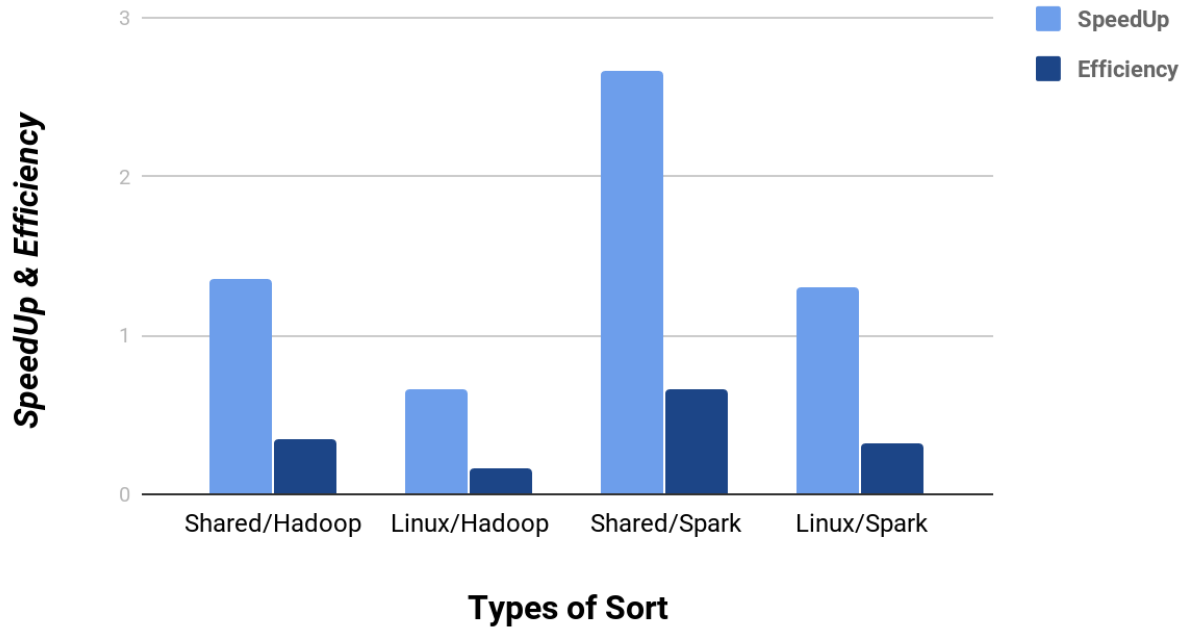
Strong Scaling Large Dataset - Compute Time



Strong Scaling Large Dataset - Throughput



Strong Scaling Large Dataset - SpeedUp & Efficiency



Conclusion- So we can see that we are able to achieve high speed up and efficiency and ideally time must decrease 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 parallelly.

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.

```

sajmera4@proton: ~/Hadoop
18/05/01 07:21:59 INFO mapreduce.Job: map 100% reduce 100%
18/05/01 07:22:45 INFO mapreduce.Job: Job job_1524707840070_0491 completed successfully
18/05/01 07:22:46 INFO mapreduce.Job: Counters: 51
  File System Counters
    FILE: Number of bytes read=48016122714
    FILE: Number of bytes written=70528459225
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=80004998728
    HDFS: Number of bytes written=80000000000
    HDFS: Number of read operations=3600
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=16
  Job Counters
    Killed map tasks=2
    Launched map tasks=1194
    Launched reduce tasks=8
    Data-local map tasks=989
    Rack-local map tasks=205
    Total time spent by all maps in occupied slots (ms)=32618898
    Total time spent by all reduces in occupied slots (ms)=15923012
    Total time spent by all map tasks (ms)=32618898
    Total time spent by all reduce tasks (ms)=15923012
    Total vcore-milliseconds taken by all map tasks=32618898
    Total vcore-milliseconds taken by all reduce tasks=15923012
    Total megabyte-milliseconds taken by all map tasks=33401751552
    Total megabyte-milliseconds taken by all reduce tasks=16305164288
  Map-Reduce Framework
    Map input records=800000000
    Map output records=800000000
    Map output bytes=80000000000
    Map output materialized bytes=22269932538
    Input split bytes=120392
    Combine input records=0
    Combine output records=0
    Reduce input groups=800000000
    Reduce shuffle bytes=22269932538
    Reduce input records=800000000
    Reduce output records=800000000

```

Figure 9: Hadoop 80GB Sorting

```

sajmera4@proton: ~/Hadoop
checksum 17d787d3672c5273
~
~

```

Figure 10: Hadoop 80GB TeraValidate Output

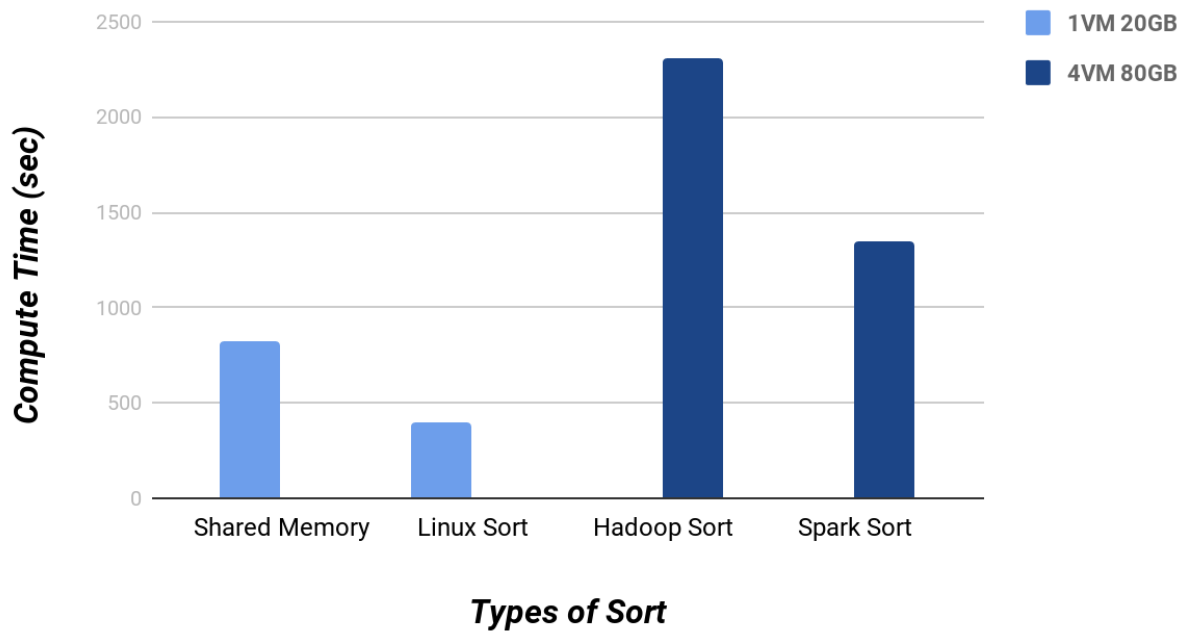
```
sajmera4@proton: ~/Spark
2018-05-02 00:47:26 INFO MemoryStore:54 - MemoryStore cleared
2018-05-02 00:47:26 INFO BlockManager:54 - BlockManager stopped
2018-05-02 00:47:26 INFO BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-02 00:47:26 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator stopped!
2018-05-02 00:47:26 INFO SparkContext:54 - Successfully stopped SparkContext
Time taken in seconds to sort file using Spark is: 1342.345
2018-05-02 00:47:26 INFO ShutdownHookManager:54 - Shutdown hook called
2018-05-02 00:47:26 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-eaa31fac-aef0-4e6c-b541-4e1d76625a82
2018-05-02 00:47:26 INFO ShutdownHookManager:54 - Deleting directory /tmp/spark-b9a00924-81db-45db-92dc-d5619e9b5f0f
18/05/02 00:47:30 INFO client.RMProxy: Connecting to ResourceManager at hadoop-d/192.168.2.62:8032
18/05/02 00:47:31 INFO input.FileInputFormat: Total input files to process : 597
Spent 232ms computing base-splits.
Spent 13ms computing TeraScheduler splits.
18/05/02 00:47:31 INFO mapreduce.JobSubmitter: number of splits:597
18/05/02 00:47:31 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated.
-metrics-publisher.enabled
18/05/02 00:47:32 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524709778346_0475
18/05/02 00:47:32 INFO impl.YarnClientImpl: Submitted application application_1524709778346_0475
18/05/02 00:47:32 INFO mapreduce.Job: The url to track the job: http://hadoop-d:8088/proxy/application_1524709778346_04
18/05/02 00:47:32 INFO mapreduce.Job: Running job: job_1524709778346_0475
18/05/02 00:47:39 INFO mapreduce.Job: Job job_1524709778346_0475 running in uber mode : false
18/05/02 00:47:39 INFO mapreduce.Job: map 0% reduce 0%
```

Figure 11: Spark 80GB Sorting

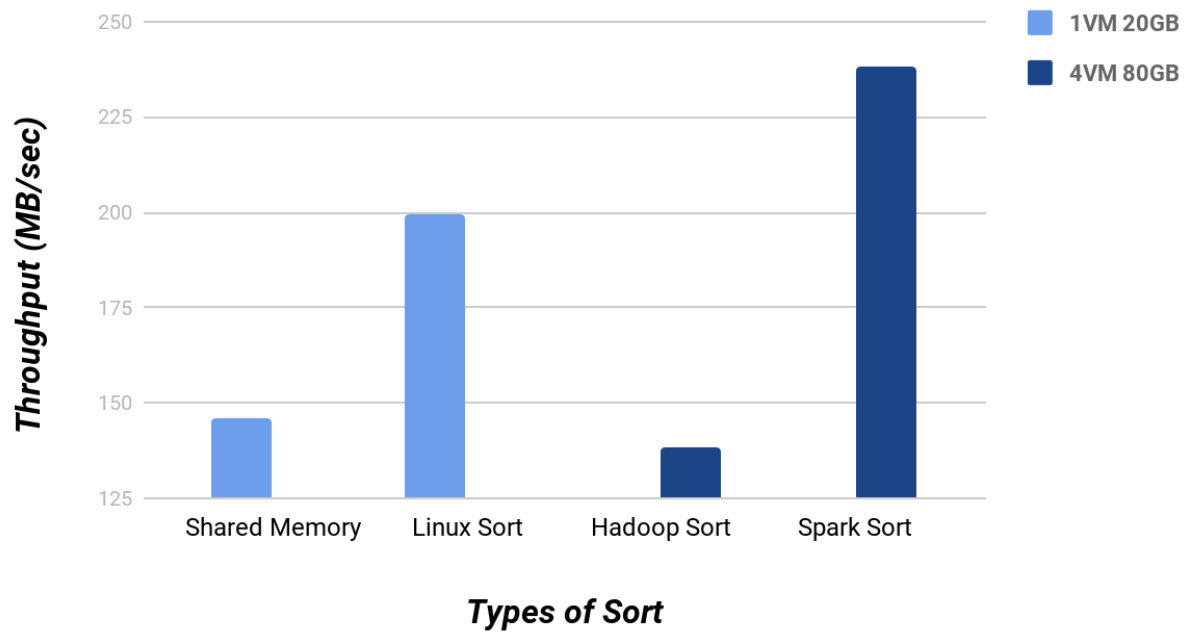
```
sajmera4@proton: ~/Spark
checksum 17d770cb50dc010f
~
~
~
```

Figure 12: Spark 80GB TeraValidate Output

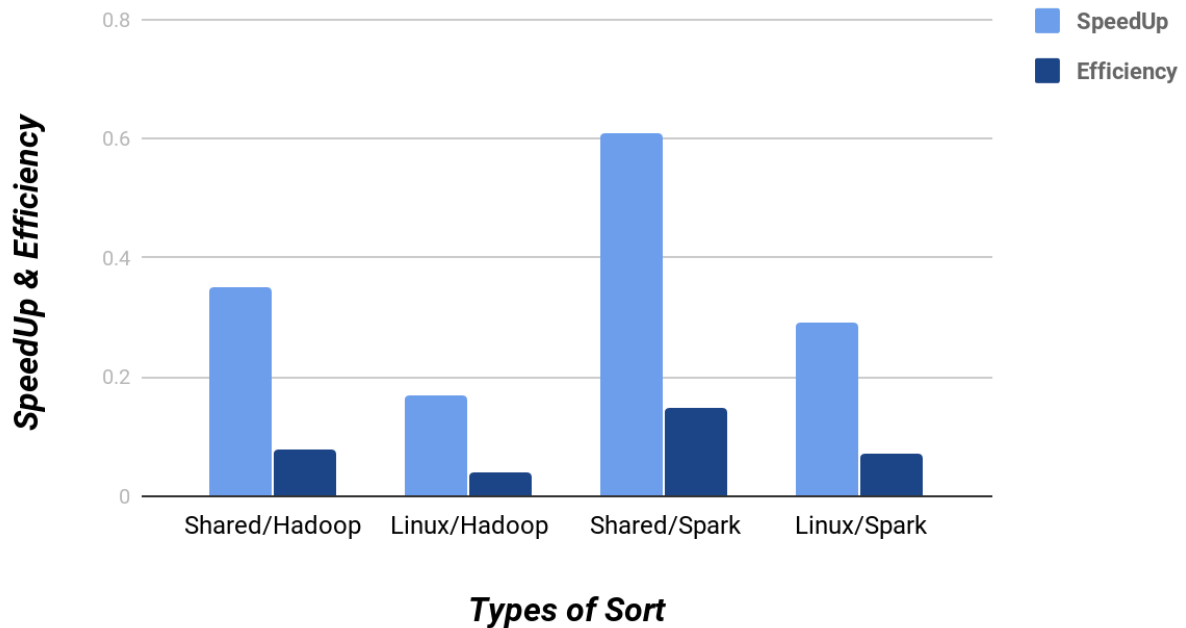
Weak Scaling Large Dataset - Compute Time



Weak Scaling Large Dataset - Throughput



Weak Scaling Large Dataset - SpeedUp and Efficiency



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.