Report

- **Problem:**

  The problem here is to sort a large dataset which varies from 8GB to 80GB in size and the environment that needs to be used is Hadoop and spark for achieving faster speed. Also, compare results for shared memory (1 VM) results with Hadoop Terasort(4 VM) and Spark Terasort(4 VM).

- **Solution**:
  - As we know we cannot use in memory sort for such large data files as there is limited memory that we have, and such large files cannot be accommodated at once.
  - Hence, we use external sort algorithms to deal with such kind of sorting.
  - Since the file is generated using gensort, I have sorted the 100 bytes using by dividing first 10 bytes as key and last 90 bytes as value.
  - Only sorting based on key gives the correct result and also improves performance.

- **Methodology**:
  - Programming language used: **JAVA**
  - Algorithm used:
    1. Hadoop: TeraByte sort – It samples the input data and uses map/reduce to sort the data into a total order.
       - It uses MapReduce pair to sort very large datasets.
       - During the Map phase, each mapper generates a key which is 10 bytes long.
       - The rest of 90 bytes are treated as value to that key.
       - The map phase results in collection which is sorted based on the keys in ascending/descending order.
       - Multiple reducers are used to increase the implementation speed to great extent.
       - The main advantage of Hadoop framework is that the intermediate steps are not stored at any moment and hence reduce phase does not do any work at this time.
    2. Spark: Use of in-memory sort to sort the data.
       - Spark takes input file which should reside on HDFS directory.
       - It takes output directory as an input where output files are stored.
       - Each record from the file is read and first 10 bytes of the records are the keys.
       - Use of sortByKey to sort the records based on the key values in ascending/descending order.
       - Use of FlatMap to merge the key and values back to record.

- The file is then saved on hdfs directory.
- Spark is considered to be 10x faster due to it's in-memory algorithm implementation.

- Input:
  1. The input file is generated using gensort.
  2. Following command generates file with 1000 records where each record corresponds to 100 bytes each (98 bytes of data and 2 bytes for "\r\n").
     - gensort -a 1000 sample.txt
     - so the file size will be 1000*100 = 100000 bytes.
  3. To validate the output, we make use of TeraValidate to validate whether output file is sorted or not and also for duplicity check along with checksum.
- Also, first 10 bytes of each line in the input file are considered as key and the sorting is done based on that part to increase sorting behavior.
- There are two read operations performed and two write operations performed on the disk.

- **Performance Evaluation**:

  - This section will describe the performance of shared memory vs Hadoop sort vs Spark sort program.
    - Number of Nodes used: 1, **4**
    - Input file size:
      1. **2GB**
      2. **8GB**

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)**
Compute Time (sec)	**84.462**	**27**	**370.914**	**392.768**
Data Read (GB)	4	4	16	16
Data Write (GB)	4	4	16	16
I/O Throughput (MB/sec)	94.71715091	**296.2962963**	86.27336795	81.47303243
Speedup	N/A	N/A	1.097872416	1.16255831
Efficiency	N/A	N/A	0.274468104	0.290639578

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

Inference:

- From the above table we can see that Spark Sort has more speedup as compared to Hadoop sort.
- Also, spark shows out more efficiency than Hadoop sort.

- From the above table for weak scaling, where data load is 2Gb on each nodes for spark and Hadoop sort, we see that Hadoop sort has a better throughput as compared to spark sort due to less data set size and also due to use of more reducers for map/reduce functionality.

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)**
Compute Time (sec)	867.871	395	1137.307	1398.715
Data Read (GB)	40	40	40	40
Data Write (GB)	40	40	40	40
I/O Throughput (MB/sec)	92.17959812	202.5316456	70.34160521	57.19535431
Speedup	N/A	N/A	1.310456277	1.611662332
Efficiency	N/A	N/A	0.327614069	0.402915583

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

Inference:

- From the above table we can see that Spark Sort has more speedup as compared to Hadoop sort.
- Also, spark shows out more efficiency than Hadoop sort.
- From the above table for Strong Scaling where the fixed data load of 20GB is shared on 4VM's as compared to that of 1VM in shared memory.
- Hadoop sort gives better performance here because of compression and use of more number of reducers to perform Map/Reduce operation.

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)**
Compute Time (sec)	867.871	395	3288.64	3098.27
Data Read (GB)	40	40	160	160
Data Write (GB)	40	40	160	160
I/O Throughput (MB/sec)	92.17959812	202.5316456	97.30466089	103.2834453
Speedup	N/A	N/A	0.94732973	0.892491511
Efficiency	N/A	N/A	0.236832432	0.223122878

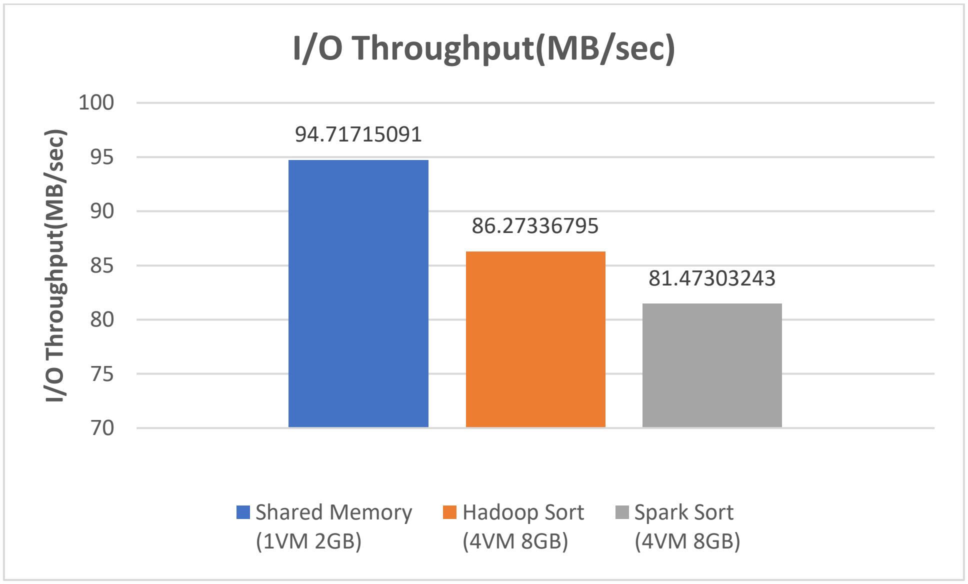
Table III: Performance evaluation of sort (weak scaling – large dataset)

Inference:

- From the above table, we can clearly make out that Spark has faster computation time as compared to Hadoop sort.

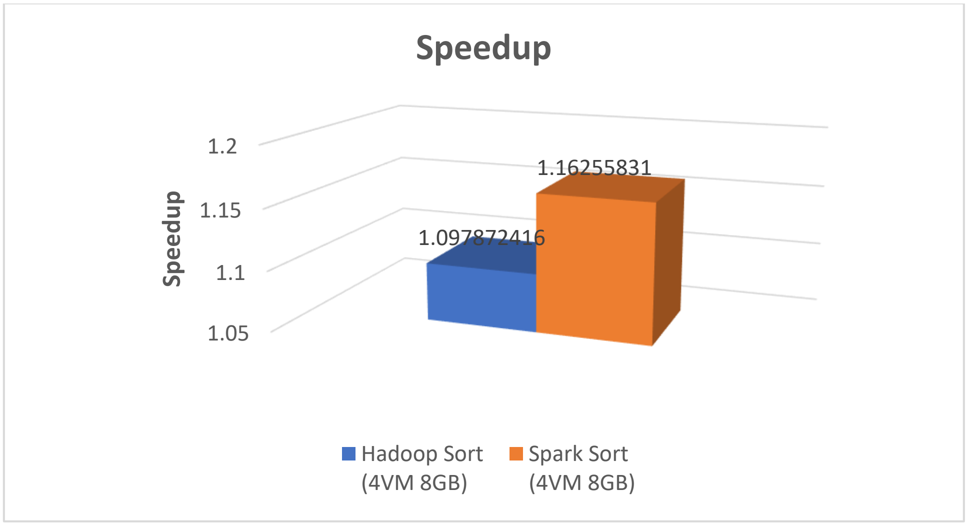
- We have made use of weak scaling where the data size is increased with the number of nodes.
- Hadoop gives better throughput here as compared to Spark, probably due to use of less executors for spark.
- The Spark has almost similar efficiency to that of Hadoop sort.

- **Graphical Visualization**:



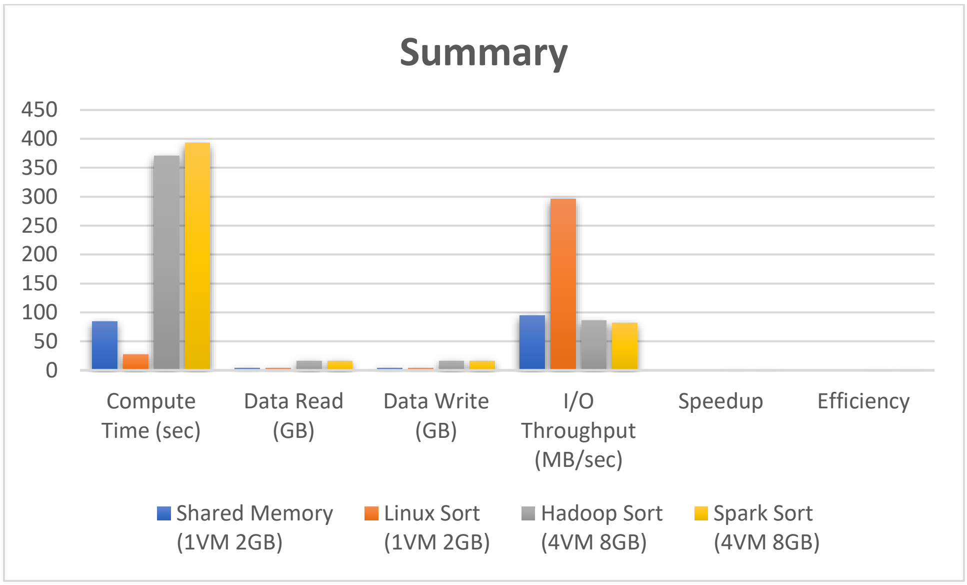
Throughput: Performance evaluation of sort (weak scaling – small dataset)

✓ As we see that throughput is maximum for Shared memory here for small dataset.
✓ This is the case because of use of multi-threaded implementation.



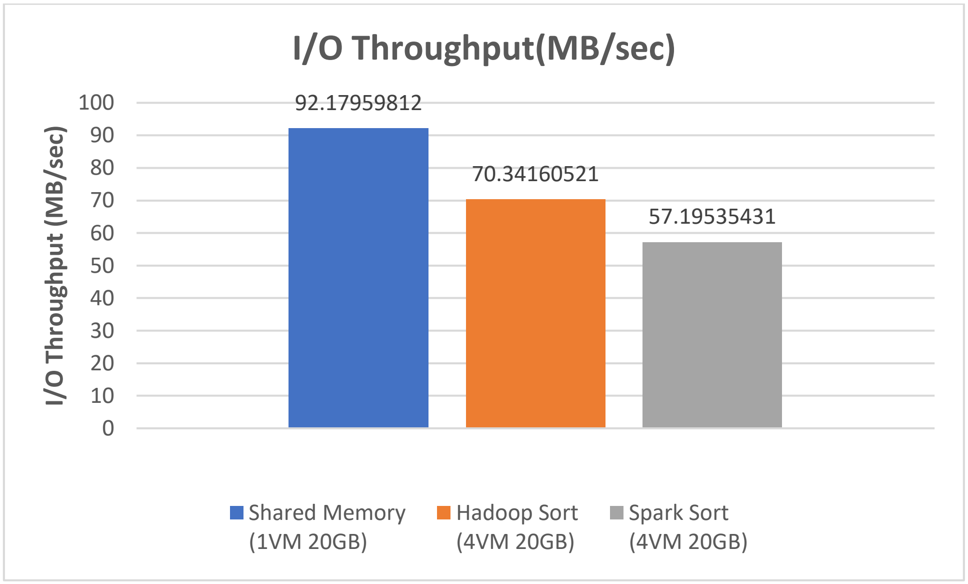
Speedup: Performance evaluation of sort (weak scaling – small dataset)

- ✓ The above graph shows that Spark has more speedup as compared to Hadoop.
- ✓ Spark has in-memory sort which leads to increase in speed of execution.



<u>Summary: Performance evaluation of sort (weak scaling – small dataset)</u>

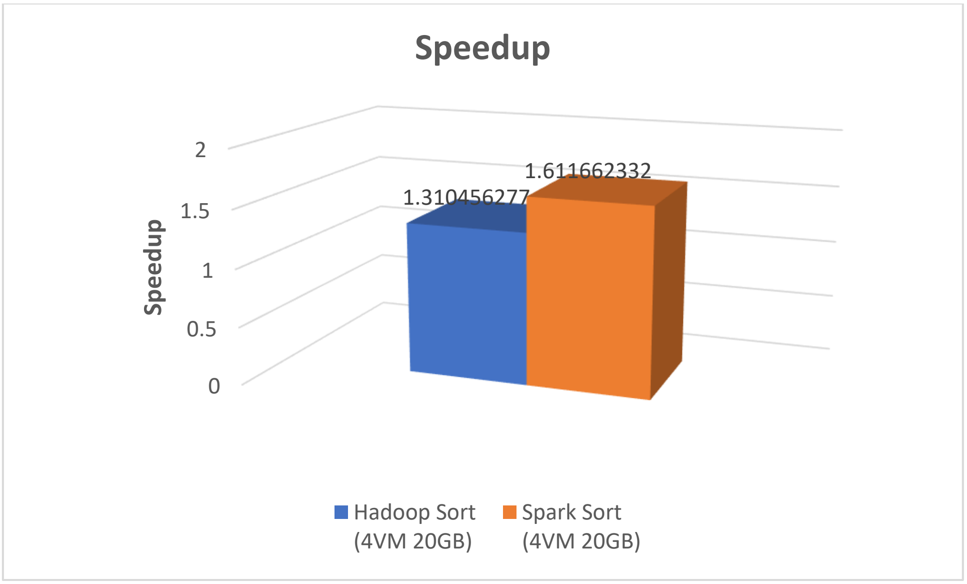
- ✓ The above graph shows the summary of all the sorting types that we have implemented for 2Gb shared memory and Hadoop and spark using weak scaling.



<u>Throughput: Performance evaluation of sort (strong scaling – large dataset)</u>

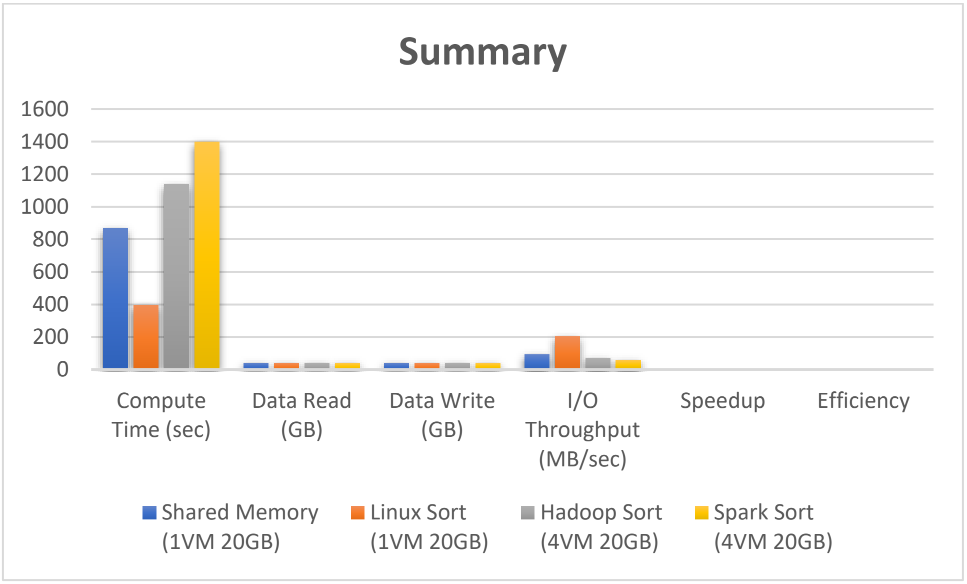
- ✓ As we see that throughput is maximum for Shared memory here for moderate dataset.

- ✓ This is the case because of use of multi-threaded implementation.



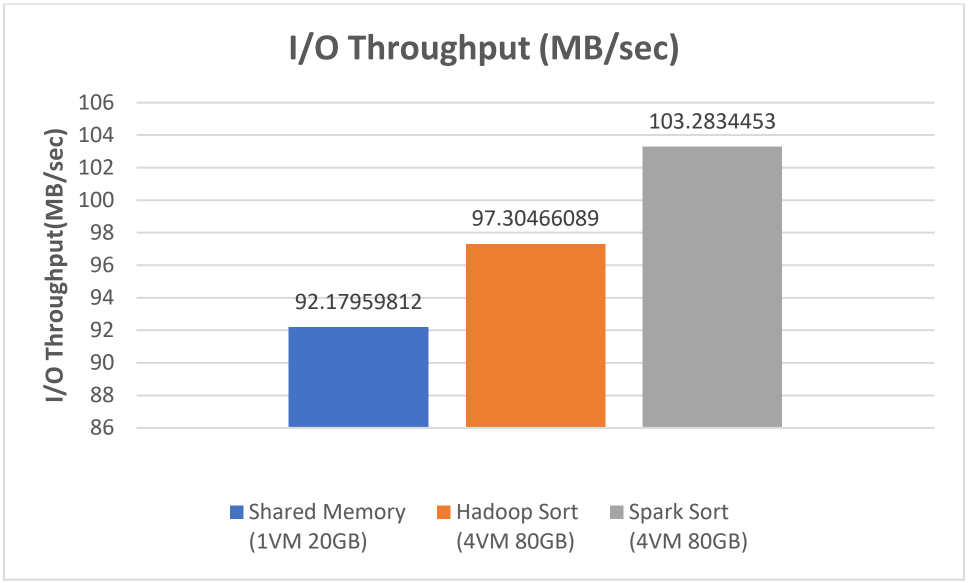
<u>Speedup:</u> <u>Performance evaluation of sort (strong scaling – large dataset)</u>

- ✓ The above graph shows that Spark has more speedup as compared to Hadoop.
- ✓ Spark has in-memory sort which leads to increase in speed of execution.



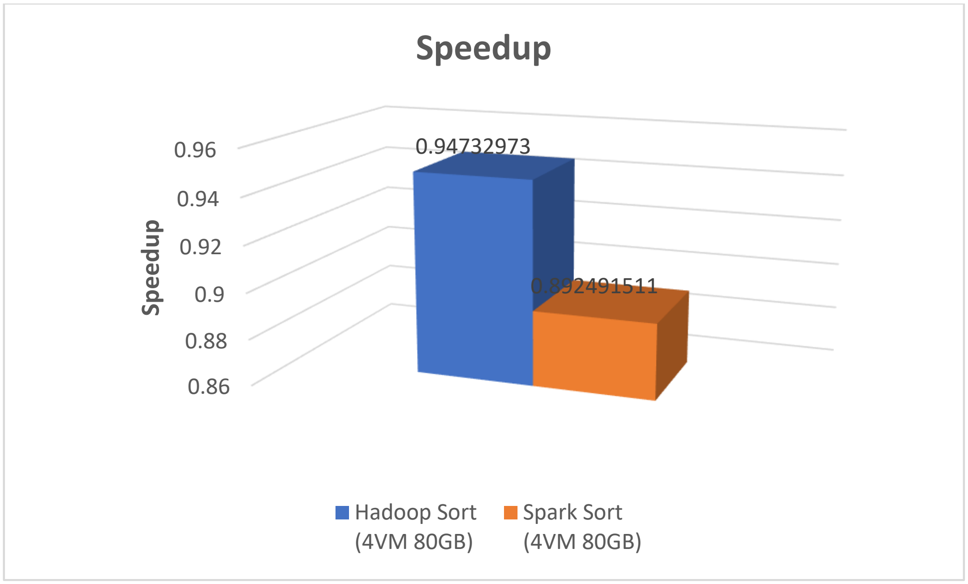
<u>Summary: Performance evaluation of sort (strong scaling – large dataset)</u>

- ✓ The above graph shows the summary of all the sorting types that we have implemented for 20Gb shared memory and Hadoop and spark using weak scaling.



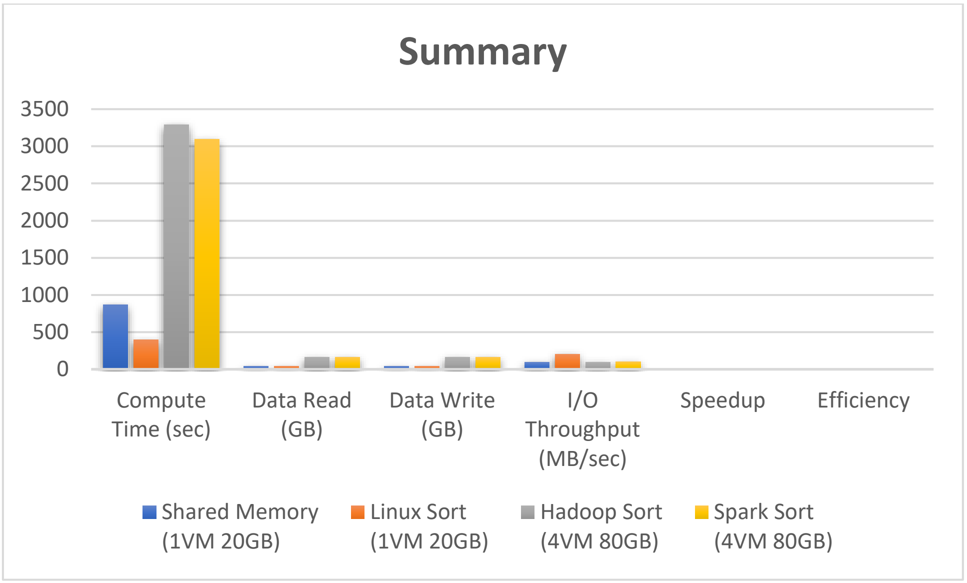
<u>**Throughput: Performance evaluation of sort (weak scaling – large dataset)**</u>

- ✓ As we see that throughput is maximum for Spark memory here for Large dataset.
- ✓ This is the case because of use of in-memory sort by spark.
- ✓ Spark's performance increases as the size of the dataset increases.
- ✓ Hadoop is good for small and moderate datasets.



<u>**Speedup: Performance evaluation of sort (weak scaling – large dataset)**</u>

- ✓ The above graph shows that Hadoop has more speedup as compared to Spark.
- ✓ This comparison is with respect to shared memory 1VM results.

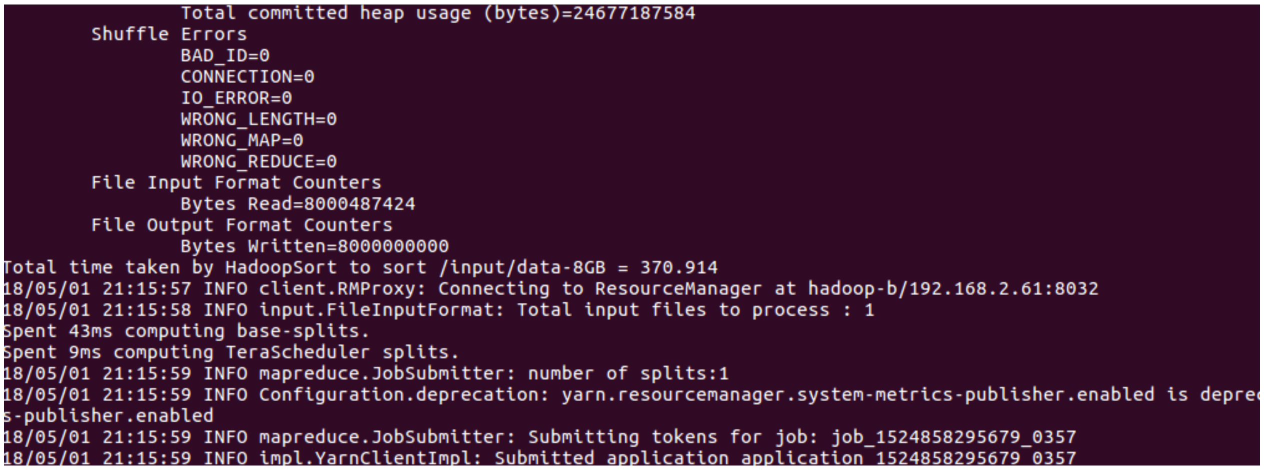


<u>**Summary: Performance evaluation of sort (weak scaling – large dataset)**</u>

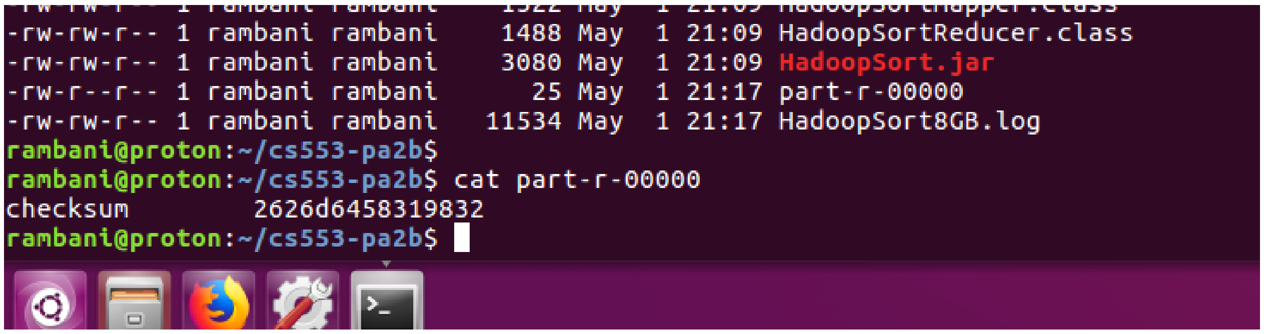
✓ The above graph shows the summary of all the sorting types that we have implemented for 80Gb shared memory and Hadoop and spark using weak scaling.

<u>Output</u>:

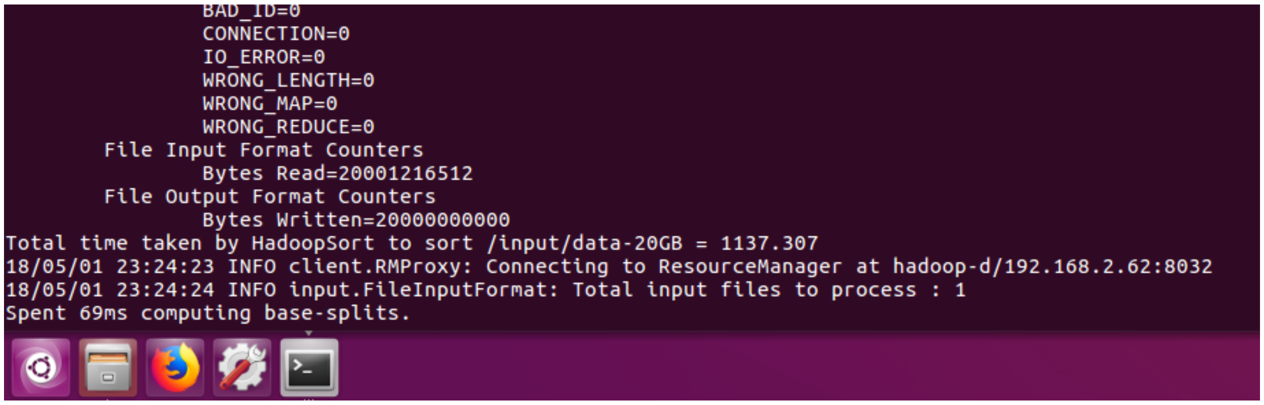
1. Hadoop Sort:



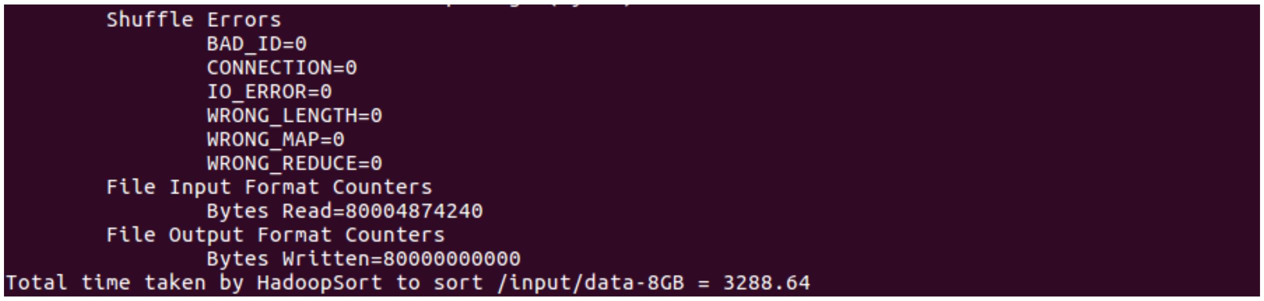
```
                Total committed heap usage (bytes)=24677187584
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=8000487424
        File Output Format Counters
                Bytes Written=8000000000
Total time taken by HadoopSort to sort /input/data-8GB = 370.914
18/05/01 21:15:57 INFO client.RMProxy: Connecting to ResourceManager at hadoop-b/192.168.2.61:8032
18/05/01 21:15:58 INFO input.FileInputFormat: Total input files to process : 1
Spent 43ms computing base-splits.
Spent 9ms computing TeraScheduler splits.
18/05/01 21:15:59 INFO mapreduce.JobSubmitter: number of splits:1
18/05/01 21:15:59 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is depre
s-publisher.enabled
18/05/01 21:15:59 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524858295679_0357
18/05/01 21:15:59 INFO impl.YarnClientImpl: Submitted application application_1524858295679_0357
```



```
-rw-rw-r-- 1 rambani rambani    1488 May  1 21:09 HadoopSortReducer.class
-rw-rw-r-- 1 rambani rambani    3080 May  1 21:09 HadoopSort.jar
-rw-r--r-- 1 rambani rambani      25 May  1 21:17 part-r-00000
-rw-rw-r-- 1 rambani rambani   11534 May  1 21:17 HadoopSort8GB.log
rambani@proton:~/cs553-pa2b$
rambani@proton:~/cs553-pa2b$ cat part-r-00000
checksum        2626d6458319832
rambani@proton:~/cs553-pa2b$
```

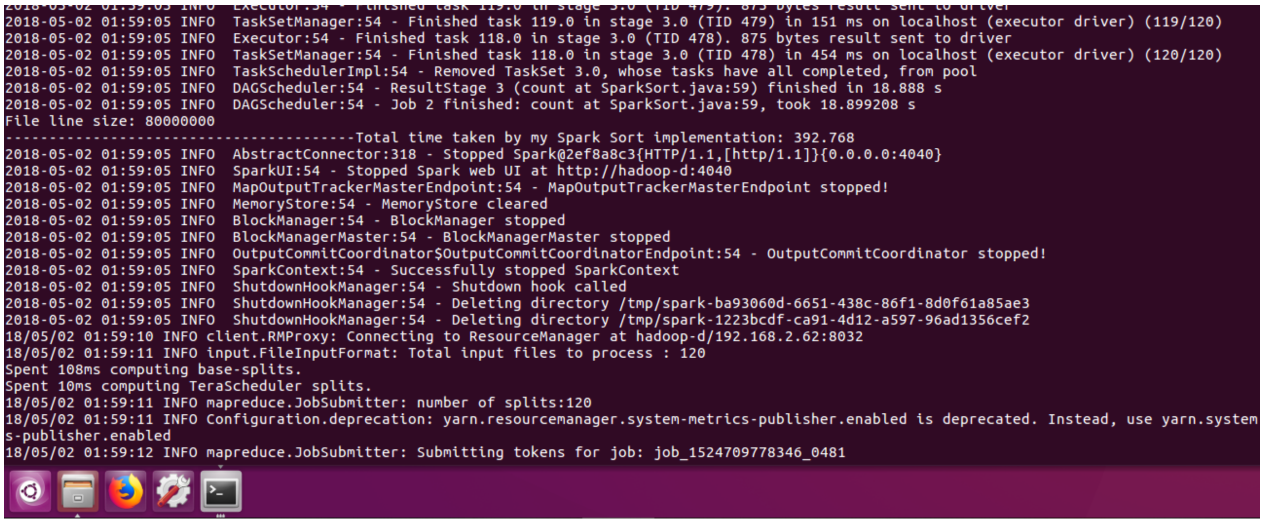


```
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=20001216512
        File Output Format Counters
                Bytes Written=20000000000
Total time taken by HadoopSort to sort /input/data-20GB = 1137.307
18/05/01 23:24:23 INFO client.RMProxy: Connecting to ResourceManager at hadoop-d/192.168.2.62:8032
18/05/01 23:24:24 INFO input.FileInputFormat: Total input files to process : 1
Spent 69ms computing base-splits.
```

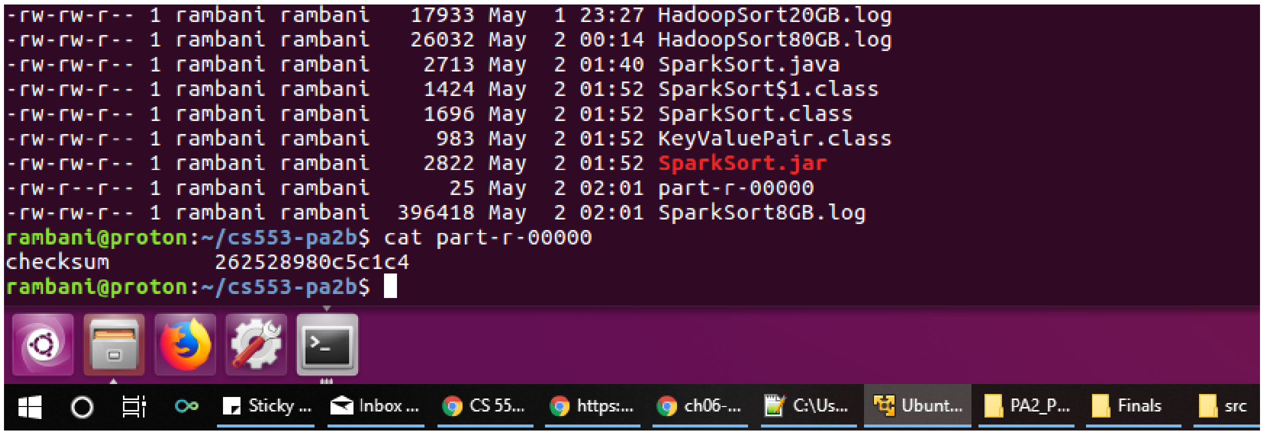


```
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=80004874240
        File Output Format Counters
                Bytes Written=80000000000
Total time taken by HadoopSort to sort /input/data-8GB = 3288.64
```

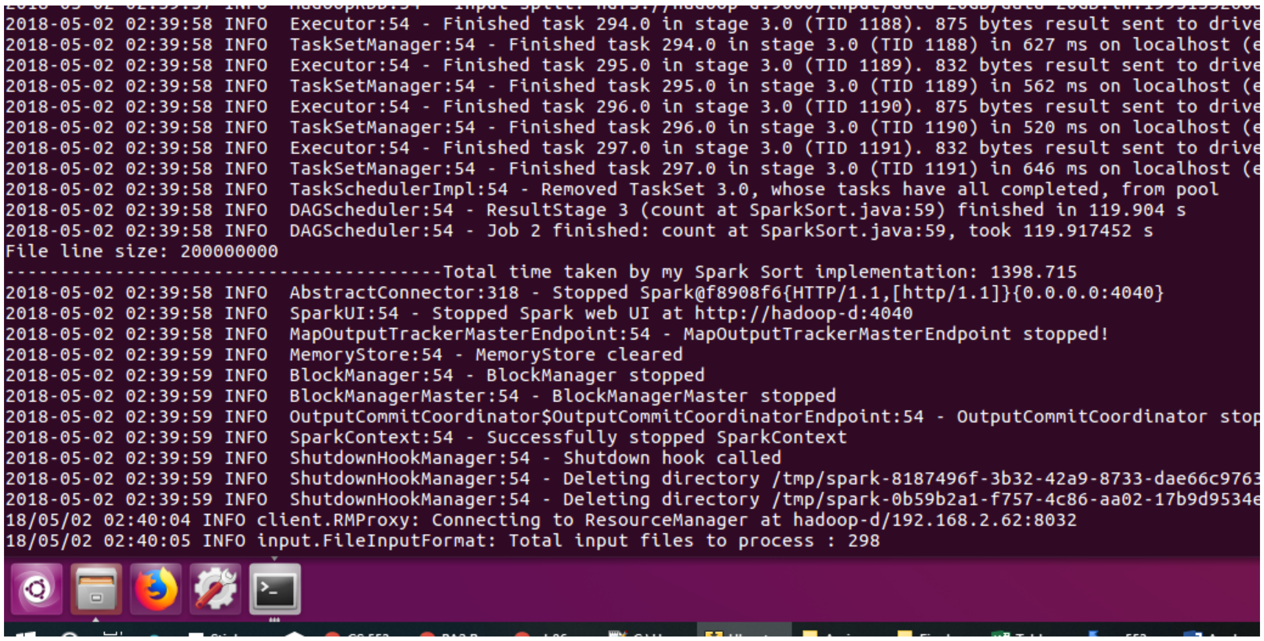
2. Spark Sort:



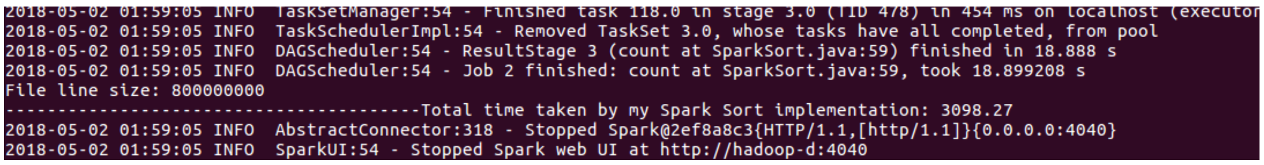
```
2018-05-02 01:59:05 INFO  TaskSetManager:54 - Finished task 119.0 in stage 3.0 (TID 479) in 151 ms on localhost (executor driver) (119/120)
2018-05-02 01:59:05 INFO  Executor:54 - Finished task 118.0 in stage 3.0 (TID 478). 875 bytes result sent to driver
2018-05-02 01:59:05 INFO  TaskSetManager:54 - Finished task 118.0 in stage 3.0 (TID 478) in 454 ms on localhost (executor driver) (120/120)
2018-05-02 01:59:05 INFO  TaskSchedulerImpl:54 - Removed TaskSet 3.0, whose tasks have all completed, from pool
2018-05-02 01:59:05 INFO  DAGScheduler:54 - ResultStage 3 (count at SparkSort.java:59) finished in 18.888 s
2018-05-02 01:59:05 INFO  DAGScheduler:54 - Job 2 finished: count at SparkSort.java:59, took 18.899208 s
File line size: 80000000
----------------------------------------Total time taken by my Spark Sort implementation: 392.768
2018-05-02 01:59:05 INFO  AbstractConnector:318 - Stopped Spark@2ef8a8c3{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-05-02 01:59:05 INFO  SparkUI:54 - Stopped Spark web UI at http://hadoop-d:4040
2018-05-02 01:59:05 INFO  MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!
2018-05-02 01:59:05 INFO  MemoryStore:54 - MemoryStore cleared
2018-05-02 01:59:05 INFO  BlockManager:54 - BlockManager stopped
2018-05-02 01:59:05 INFO  BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-02 01:59:05 INFO  OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator stopped!
2018-05-02 01:59:05 INFO  SparkContext:54 - Successfully stopped SparkContext
2018-05-02 01:59:05 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-05-02 01:59:05 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-ba93060d-6651-438c-86f1-8d0f61a85ae3
2018-05-02 01:59:05 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-1223bcdf-ca91-4d12-a597-96ad1356cef2
18/05/02 01:59:10 INFO client.RMProxy: Connecting to ResourceManager at hadoop-d/192.168.2.62:8032
18/05/02 01:59:11 INFO input.FileInputFormat: Total input files to process : 120
Spent 108ms computing base-splits.
Spent 10ms computing TeraScheduler splits.
18/05/02 01:59:11 INFO mapreduce.JobSubmitter: number of splits:120
18/05/02 01:59:11 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system
s-publisher.enabled
18/05/02 01:59:12 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1524709778346_0481
```



```
-rw-rw-r-- 1 rambani rambani   17933 May  1 23:27 HadoopSort20GB.log
-rw-rw-r-- 1 rambani rambani   26032 May  2 00:14 HadoopSort80GB.log
-rw-rw-r-- 1 rambani rambani    2713 May  2 01:40 SparkSort.java
-rw-rw-r-- 1 rambani rambani    1424 May  2 01:52 SparkSort$1.class
-rw-rw-r-- 1 rambani rambani    1696 May  2 01:52 SparkSort.class
-rw-rw-r-- 1 rambani rambani     983 May  2 01:52 KeyValuePair.class
-rw-rw-r-- 1 rambani rambani    2822 May  2 01:52 SparkSort.jar
-rw-r--r-- 1 rambani rambani      25 May  2 02:01 part-r-00000
-rw-rw-r-- 1 rambani rambani  396418 May  2 02:01 SparkSort8GB.log
rambani@proton:~/cs553-pa2b$ cat part-r-00000
checksum        262528980c5c1c4
rambani@proton:~/cs553-pa2b$ 
```



```
2018-05-02 02:39:58 INFO  Executor:54 - Finished task 294.0 in stage 3.0 (TID 1188). 875 bytes result sent to drive
2018-05-02 02:39:58 INFO  TaskSetManager:54 - Finished task 294.0 in stage 3.0 (TID 1188) in 627 ms on localhost (e
2018-05-02 02:39:58 INFO  Executor:54 - Finished task 295.0 in stage 3.0 (TID 1189). 832 bytes result sent to drive
2018-05-02 02:39:58 INFO  TaskSetManager:54 - Finished task 295.0 in stage 3.0 (TID 1189) in 562 ms on localhost (e
2018-05-02 02:39:58 INFO  Executor:54 - Finished task 296.0 in stage 3.0 (TID 1190). 875 bytes result sent to drive
2018-05-02 02:39:58 INFO  TaskSetManager:54 - Finished task 296.0 in stage 3.0 (TID 1190) in 520 ms on localhost (e
2018-05-02 02:39:58 INFO  Executor:54 - Finished task 297.0 in stage 3.0 (TID 1191). 832 bytes result sent to drive
2018-05-02 02:39:58 INFO  TaskSetManager:54 - Finished task 297.0 in stage 3.0 (TID 1191) in 646 ms on localhost (e
2018-05-02 02:39:58 INFO  TaskSchedulerImpl:54 - Removed TaskSet 3.0, whose tasks have all completed, from pool
2018-05-02 02:39:58 INFO  DAGScheduler:54 - ResultStage 3 (count at SparkSort.java:59) finished in 119.904 s
2018-05-02 02:39:58 INFO  DAGScheduler:54 - Job 2 finished: count at SparkSort.java:59, took 119.917452 s
File line size: 200000000
----------------------------------------Total time taken by my Spark Sort implementation: 1398.715
2018-05-02 02:39:58 INFO  AbstractConnector:318 - Stopped Spark@f8908f6{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-05-02 02:39:58 INFO  SparkUI:54 - Stopped Spark web UI at http://hadoop-d:4040
2018-05-02 02:39:58 INFO  MapOutputTrackerMasterEndpoint:54 - MapOutputTrackerMasterEndpoint stopped!
2018-05-02 02:39:59 INFO  MemoryStore:54 - MemoryStore cleared
2018-05-02 02:39:59 INFO  BlockManager:54 - BlockManager stopped
2018-05-02 02:39:59 INFO  BlockManagerMaster:54 - BlockManagerMaster stopped
2018-05-02 02:39:59 INFO  OutputCommitCoordinator$OutputCommitCoordinatorEndpoint:54 - OutputCommitCoordinator stop
2018-05-02 02:39:59 INFO  SparkContext:54 - Successfully stopped SparkContext
2018-05-02 02:39:59 INFO  ShutdownHookManager:54 - Shutdown hook called
2018-05-02 02:39:59 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-8187496f-3b32-42a9-8733-dae66c9763
2018-05-02 02:39:59 INFO  ShutdownHookManager:54 - Deleting directory /tmp/spark-0b59b2a1-f757-4c86-aa02-17b9d9534e
18/05/02 02:40:04 INFO client.RMProxy: Connecting to ResourceManager at hadoop-d/192.168.2.62:8032
18/05/02 02:40:05 INFO input.FileInputFormat: Total input files to process : 298
```



```
2018-05-02 01:59:05 INFO  TaskSetManager:54 - Finished task 118.0 in stage 3.0 (TID 478) in 454 ms on localhost (executor
2018-05-02 01:59:05 INFO  TaskSchedulerImpl:54 - Removed TaskSet 3.0, whose tasks have all completed, from pool
2018-05-02 01:59:05 INFO  DAGScheduler:54 - ResultStage 3 (count at SparkSort.java:59) finished in 18.888 s
2018-05-02 01:59:05 INFO  DAGScheduler:54 - Job 2 finished: count at SparkSort.java:59, took 18.899208 s
File line size: 800000000
----------------------------------------Total time taken by my Spark Sort implementation: 3098.27
2018-05-02 01:59:05 INFO  AbstractConnector:318 - Stopped Spark@2ef8a8c3{HTTP/1.1,[http/1.1]}{0.0.0.0:4040}
2018-05-02 01:59:05 INFO  SparkUI:54 - Stopped Spark web UI at http://hadoop-d:4040
```

- **<u>Challenges</u>**:
    - Clusters were very instable. And since we were asked to implement directly on cluster by giving us less number of days for the assignment, we faced lot of problems like we had to wait a lot for the nodes to be assigned as there were initially only 4 nodes for entire class of ~90-100 students, also, the nodes had many issues like my code ran on certain nodes and other nodes threw error.
    - From coding perspective, I faced issue while compressing data for 80GB.
    - Also, increasing number of reducers led to errors in output file where TeraValidate threw error.
    - Spark code ran slower than Hadoop for small files and I investigated a lot to fix this issue.
    - Another problem that was faced is that the slurm jobs were getting timed out in case of 80GB operations, I had to make my code more efficient in order to overcome this difficulty.

- **<u>Performance Comparison</u>**:
    - Spark is the winner for sorting large datasets at large number of nodes because it performs in-memory sorting which increases the performance to great exetent.
    - Also comparing the results for **2013 & 2014 Sort Benchmark winners**:
        1. 2013: Hadoop
            - **System info**: 2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)
            - **Throughput**: 4328 seconds to sort 102.5 TB data which is 23.68 GB/Sec
            - If we compare our throughput which is around 55% of their value.
        2. 2014: Spark
            - **System info**: 207 nodes x(32 vCores - 2.5Ghz Intel Xeon E5-2670 v2, 244GB memory, 8x800 GB SSD
            - **Throughput**: 1406 seconds to sort 100 TB data which is around 71.12 GB/Sec
            - By comparing the throughput with our results which comes to around 60% of their value.
- **<u>Cloudsort Benchmark</u>**:
    - The report found at the mentioned link represents a benchmark called cloudsort which is mainly for Input/Output intensive applications.

- It calculates minimum cost for sorting a given number of records for any public cloud.
- It implements total-cost-of-ownership benchmark which is mainly based on an external sort.
- It is better than many other benchmarks in variety of ways like Affordability and accessibility.

- **Conclusion**:
  - Spark gives the best performance as we increase the number of nodes.
  - At 1 node, shared memory gives the best results due to easy memory access which results in faster operations.
  - At 4 nodes, Hadoop and Spark will give almost similar results as Hadoop performs almost good as Spark if the number of reducers are properly managed.
  - At 100 and 1000 nodes, Spark performs best and way better than Hadoop implementation.
  - Also, as we increase the dataset size, Spark gives better performance as compared to Hadoop Sort and Shared memory sort.
  - Shared Memory sort is best for 1Vm whereas Spark is best for more number of nodes and dataset size.
  - Hadoop is good for moderate dataset size and number of nodes.
  - Spark has best speedup as compared to Hadoop for most of the scenarios.