Sort on Hadoop/Spark:

Problem Statement

In this programming assignment, we are implementing sorting on Hadoop and Spark in a multi-node environment.

A primarily goal of this assignment is to sort the large documents using Hadoop and Spark.

than memory size document. We need to work with data generated with the Gensort suite (http://www.ordinal.com/gensort.html) which generates the random data. Total 3 files with size 8GB, 20GB and 80GB has been generated and placed at path "/input" on Proton cluster. We need to use above generated files to execute our program. The program must be written in Java programming language. We also need to validate the output generated by our program with teravalidate. We also need to compare our program performance with the external sort implemented as part of PA2A assignment.

Program Methodology

1. Hadoop:
   For this part Java was used as a programming language. For sorting using Hadoop, 3 class files were created. One for main function, second for mapper function and third for reducer function.
   - Hadoop Sort:

     This class contains the main function for the code. Also, makes invocation to Mapper and Reducer classes. This class creates the object for Hadoop configurations and set their values. The configurations such as whether to use compression or not, which mapper class to use, which combiner class to use, which reducer class to use, what will be the output key and value class in use, etc are set in this class.

   - MapperClass:

     This is the mapper class for Map function in Hadoop. This class will read the input file and create the key-value pairs based on the parameter we pass. In input file, first 10 bytes are the key and remaining 90 bytes are the value. So, using same information we will create the key and values. After creating key-value pairs the data will be written intermediately into context space.

   - ReducerClass:
     This is the reducer class for Reduce function in Hadoop. This class will read the intermediate key-value pairs generated by Map function and apply the Reduce function. This class will create the sorted data and write into output file.

2. Spark:

For this part also, Java was used as a programming language. For sorting using Spark only single class file was created. This class has a main method implemented. This class also extends the pairFunction class of Spark. The overrides methods for functions call has been written. sortByKey function has been used to sort the values based on key. After the sorting data will be written to output file.

For both implementations, final output is also validated by teravalidate program to confirm that final sorted output is in order. If there is any problem with order it will give a 1st record which is out of order. Else it will give a checksum of records which means all records are in order. The performance of my program is presented in performance evaluation section.

Runtime Environment settings

The program has been executed on Proton cluster which has a following environment settings:

- Operating System:
  Ubuntu 16.04.4 LTS, Release: 16.04, Codename: Xenial

- Operating System kernel
  Linux proton 4.4.0-119-generic #143-Ubuntu SMP Mon Apr 2 16:08:24 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

- Java Compiler
  javac 1.8.0_162

- Make
  GNU Make 4.1

- Hadoop:
  Hadoop 2.9.0. 4 nodes with 4 cores each, 8GB memory and 80GB SSD

- Spark:
  Spark version 2.3.0
  Using Scala version 2.11.8, OpenJDK 64-Bit Server VM, 1.8.0_162

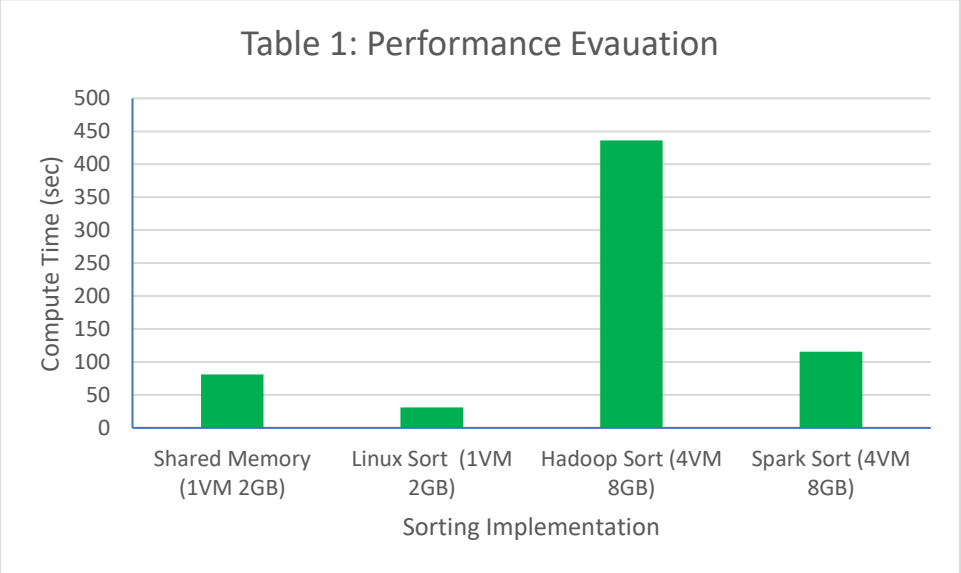
Performance Evaluation

This section presents the results of my program. This section contains three tables one for each input files (8GB, 20GB and 80GB).

- 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)
Compute Time (sec)	81	31	436.306	115.654
Data Read (GB)	4	4	8.0004	8
Data Write (GB)	4	4	8	8
I/O Throughput (MB/sec)	98.7654321	258.0645161	36.67242715	138.3436803
Speedup	NA	NA	0.185649521	0.700364881
Efficiency (%)	NA	NA	4.64123803	17.50912204

We can see that Spark's performance is way better than Hadoop's. It's giving speed up of 0.7 compared to Hadoop's 0.18. As speedup is more, it's also an efficient with efficiency of 17.50%.

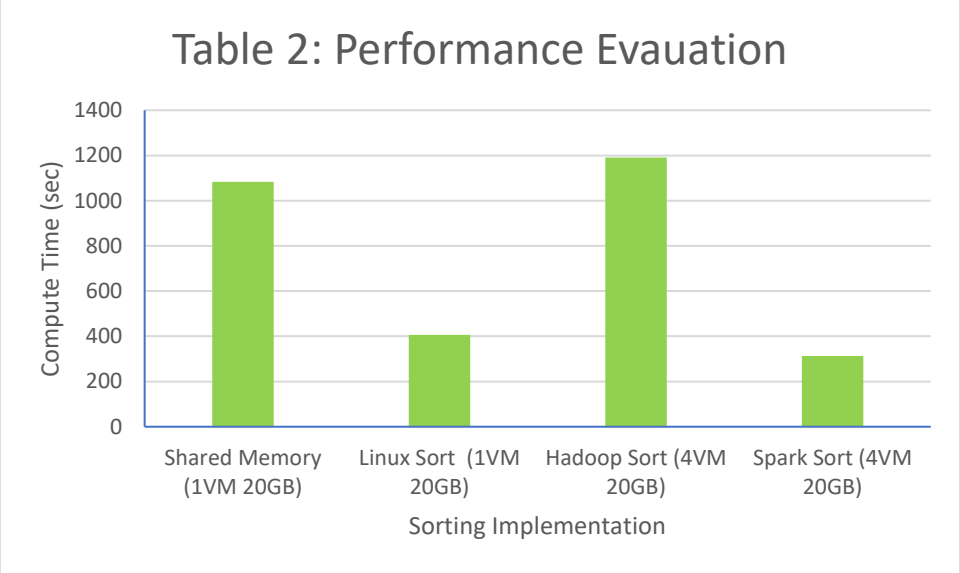


Above graph is for the visualization of compute time taken by each of the implementation. As we can see, Spark implementation is taken less time compare to Hadoop.

- 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)
Compute Time (sec)	1083	407	1191.054	313.132
Data Read (GB)	40	40	20.001	20
Data Write (GB)	40	40	20	20
I/O Throughput (MB/sec)	73.86888273	196.5601966	33.58453941	127.7416553
Speedup	NA	NA	0.909278673	3.458605317
Efficiency (%)	NA	NA	22.73196681	86.46513292

In this experiment too, Spark's performance is way better than Hadoop's. It's giving speed up of 3.458 compared to Hadoop's 0.9092. As speedup is more, it's also an efficient with efficiency of 86.46%.

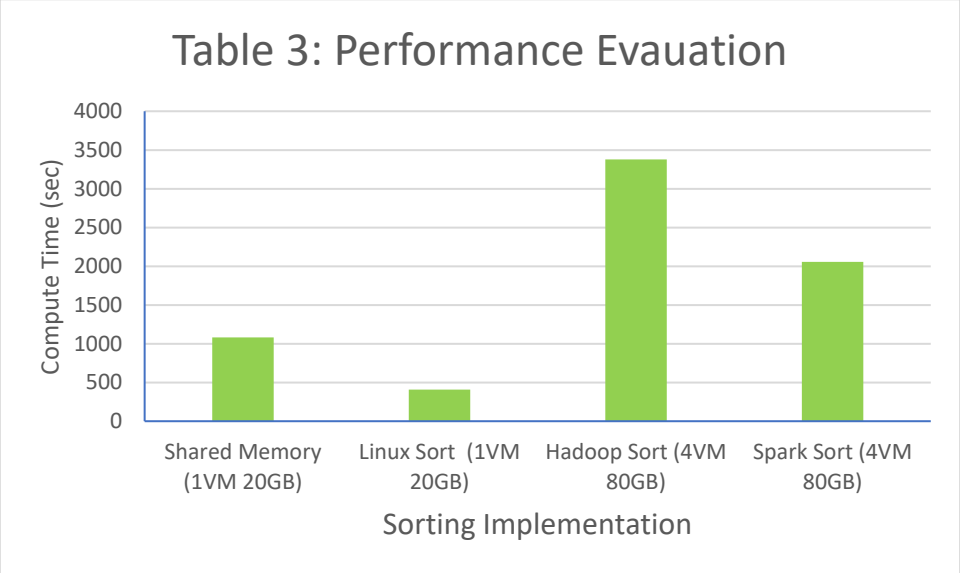


Above graph is for the visualization of compute time taken by each of the implementation. As we can see Spark implementation is taken less time compare to Hadoop.

- 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)
Compute Time (sec)	1083	407	3378.12	2056.711
Data Read (GB)	40	40	80.001	80
Data Write (GB)	40	40	80	80
I/O Throughput (MB/sec)	73.86888273	196.5601966	47.36391839	77.79410914

Speedup	NA	NA	0.320592519	0.526568876
Efficiency (%)	NA	NA	8.014812973	13.16422191



Above graph is for the visualization of compute time taken by each of the implementation. As we can see Spark implementation is taken less time compare to Hadoop.

The performance of the program of SharedMemory is less due to the hardware limitations (no. cores availability, no of disk heads, available memory at time of execution, etc). The difference in performance is because Spark can utilize the memory and disk both by keeping partial data in both. This will result in more cache hits. Spark performs well compared to Hadoop as well due to its implementation way of using memory. Also, the limitations stated earlier for my shared memory program can be the reason for having lower performance.

Conclusion:

- What conclusions can you draw?
  The mapReduce functions of Hadoop uses the disk for processing. Whereas, Spark uses memory and can use disk for processing. Spark is better than Hadoop for sorting large dataset due to more usage of memory at run time. Comparing the results of our experiments, we can say that, If the size of data is less than memory than SharedMemory sort is better. But in case the size of data is greater than memory than one should use Hadoop or Spark. Also, if the choice is to be made between Hadoop or Spark than Spark is better. This is based on our results.

- Which seems to be best at 1 node scale?

Comparing the results of our experiments, we can say that, If the size of data is less than memory then SharedMemory sort is better. But in case the size of data is greater than memory than one should use Hadoop or Spark. Also, if the choice is to be made between Hadoop or Spark than Spark is better. This is based on our results.

- How about 4 nodes?

  Again, by examine the data we have collected during our experiments, we can say that If the size of data is less than memory than SharedMemory sort is better. But in case the size of data is greater than memory than one should use Hadoop or Spark. Also, if the choice is to be made between Hadoop or Spark than Spark is better. This is based on our results.

- Can you predict which would be best at 100 node scale?

  The mapReduce functions of Hadoop uses the disk for processing. Whereas, Spark uses memory and can use disk for processing. As memory is faster than disk, program utilizing memory more than disk will be the faster. Now as Spark uses memory for processing it is faster than Hadoop. This primary difference in processing will be true irrespective of number of nodes used for execution. Thus, spark will also be faster and perform well than Hadoop on 100 node scale.

- How about 1000 node scales?

  Similar to 100 node scale, at 1000 node scale Spark will perform well compared to Hadoop due to usage of memory in processing.

- Compare your results with those from the Sort Benchmark (http://sortbenchmark.org), specifically the winners in 2013 and 2014 who used Hadoop and Spark.

2013, 1.42 TB/min

Hadoop
102.5 TB in 4,328 seconds
2100 nodes x
(2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks)
Thomas Graves
Yahoo! Inc.

2014, 4.27 TB/min

Apache Spark
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)
Reynold Xin, Parviz Deyhim, Xiangrui Meng,
Ali Ghodsi, Matei Zaharia
Databricks

Both the winners are using very different testing beds and input than our test bed and input. In our experiment, we are only sorting 80GB (Max) or file while winners sorting 100TB of input files. If we ignore the fact about testing environment and input file size and just simply compare the numbers

(results), we can say that our implementation is not as efficient as the winners have achieved. We can also say that the by implementing in correct way, we can sort TBs of data in a minute in Hadoop as well as Spark. The result of winners also indicates that Spark is better than Hadoop which was the conclusion we have drawn based on our experiments.

- what can you learn from the CloudSort benchmark, a report can be found at (http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf).

  The cloudsort benchmark is based on the external sort. The primary goal of this benchmark is to evaluate the effectiveness of Input/output intensive workloads on cloud. This benchmark provides the time taken by the implementation to sort the large file (typically 100TB or more) and the cost associated with the environment. This benchmark also helps us to find the best environments as well as cost-effective implementation of sorting.