iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** VI **Month of publication:** June 2021

DOI: https://doi.org/10.22214/ijraset.2021.35540

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 9 Issue VI Jun 2021- Available at www.ijraset.com

Performance Evaluation of Map Reduce vs. Spark framework on Amazon Machine Image for TeraSort Algorithm

Gangadhara Rao Kommu
Information Technology, Chaitanya Bharathi Institute of Technology

Abstract: ***TeraSort is one of Hadoop's widely used benchmarks. Hadoop's distribution contains both the input generator and sorting implementations: the TeraGen generates the input and TeraSort conducts the sorting. We focus on the comparison of TeraSort algorithm on the different distributed platforms with different configurations of the resources. We have considered the parameters of measure of efficiency as Compute Time, Data Read, Data Write, Compute Time, and Speedup. We have conducted experiments using Hadoop map reduce and Spark (Java). We empirically evaluate the performance of TeraSort algorithm on Amazon EC2 Machine Images, and demonstrate that it achieves 3.95 × - 2.4 × speedup, compared with TeraSort, for typical settings of interest.***

Keywords: TeraSort, Amazon Machine Image (AMI), Hadoop, Spark and Java

I. INTRODUCTION

A. First we use hadoop-mapreduce-examples-2.7.4.jar (hadoop sample program) to generate 128GB and 1TB file in HDFS directly. Then we use map reduce program to sort each file.

B. We split the program into three parts.

C. Main function: it has main function to run the program. In the beginning [1] , it created a job instance then start to set mapper, combiner and reducer function in this job. Finally, it implements each function to sorting the data. To achieve the peak performance, the program also calls combiner, which grouped data in the map phase.

D. Mapper function: it split each line of data in <key, value> pair, which key is first 10 characters, and value is remaining characters. Then it passed all <key, value> pair on combiner function.

E. Reducer function: it took the grouped data from combiner as input and sorting all of the values associated with that key.

II. CONFIGURATION SETUP

A. Amazon machine image (AMI) Specifications

1) Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-82f4dae7

2) Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical

3) With 128GB terasort, we used Amazon ubuntu image 1 x i3.large instance.

4) i3.large (9 ECUs, 2 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 15.25 GiB memory, EBS only) 1 x 475 (SSD)

5) With 1TB terasort, we used Amazon ubuntu image 1 x i3.4xlarge instance.

6) i3.4xlarge (53 ECUs, 16 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 122 GiB memory, EBS only)

7) With 1TB terasort in cluster, we used Amazon ubuntu image 8 x i3.large instance. i3.large (9 ECUs, 2 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 15.25 GiB memory, EBS only) 1 x 475 (SSD)

8) Environment setting: java-8-openjdk-amd64, hadoop-2.7.4

9) We mounted the EBS volume to boot and combine the disks into a RAID-0 to achieve the best possible performance.

III. EXPERIMENTS

A. Experiment -1

1) *Experiment Setup:* Before running 128GB data set, we installed java environment and hadoop package. Then we start to set each hadoop configuration file hdfs-site.xml, mapred-site.xml, core-site.xml and yarn-site.xml. Next we setup environment variables in ./bashrc and hadoop-env.sh. Then we can format namenode and start hadoop framework. We started testing small dataset like 1GB file to check hadoop is mounted successfully. Then we tested 128GB [2] using 1 mapper and 1 reducer, but it was fail due to lack of disk size. So we mounted 300GB EBS to accommodate datanode and namenode files and 400 GB EBS for temperate data which is generated from map reduce program. The result is shown in Figure 2 to compare experiment 2 results.

B. Experiment -II

1) Experimental Setup: Different preparation with 128GB in hadoop configuration.[3] Regarding hdfs-site.xml, we change block size from 64MB to 1GB. Regarding mapred-site.xml, we enlarge amount memory of mapper and reducer from 1024MB to 2GB. According the 1TB file size, we also need to enlarge EBS volume to accommodate temperate files and datanode/namenode files. To further explain, we put datanode/namenode files inside 2TB disk and temperate files inside 3TB disk.

C. Experiment-III

1) Experimental Setup: We needed to set up the basic hadoop environment, java and hadoop in 8 nodes first, so we used install script to run each node to make sure their configuration are the same. We also set up password-less login between one namenode and seven data nodes. Before running 1TB file size, we test 1 GB file to make sure mapreduce can work well in the cluster. Then we started generate 1TB file in HDFS, which can avoid lack of disk size problem in local nodes.[4] However, it was fails to sort successfully when Yarn resource manager is out of resource to give reducer. Therefore, we enlarged RAM allocation setting in yarn-site.xml and mapred-site.xml to solve this issue. Also, we increase the number of mapper and reducer from 1 to 2, which can sortmore efficiently. Replication number also changes from 1 to 3, which can achieve fault tolerance

The result of experiment is shown in below table:

	file size		
node number		128GB	1TB
	1	20115 sec	67268 sec
	8	N/A	33268 sec

Table 1: 128GB and 1TB terasort with different nodes

IV. PERFORMANCE CALCULATION AMONG EXPERIMENT 1, 2 AND 3

A. [1xi3.large 128GB]

1) Compute Time (sec): 20115 seconds This time is counted directly within the program.

2) Data Read (GB):61278 GB

3) This size is counted directly within the result-File system Counters.

4) Data Write (GB) : 128 GB This size is counted directly within the result-File system Counters.

5) I/O Throughput (MB/sec) : Since total computing time is 20115 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time=(61278+128)x 1000 /20115 = 3050 MB/sec

B. [1xi3.4xlarge 1TB]

1) Compute Time (sec) : 67268 seconds. This time is counted directly within the program.

2) Data Read (GB):3727284 GB This size is counted directly within the result-File system Counters.

3) Data Write (GB):1000 GB This size is counted directly within the result-File system Counters.

4) I/O Throughput (MB/sec):Since total computing time is 67268 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time (3727284+1000) x 1000 / 67268 = 55424 MB/sec

C. [8xi.large 1TB]

1) Compute Time (sec) : 33268 seconds. This time is counted directly within the program.

2) Data Read (GB):465 GB This size is counted directly within the result-File system Counters.

3) Data Write (GB):1000 GB This size is counted directly within the result-File system Counters.

4) I/O Throughput (MB/sec): Since total computing time is 33268 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time

(465+1000) x 1000 / 33268 = 44 MB/sec

*Speedup (weak scale) = 8x * efficiency*

=8 x (33268/67268) = 3.95X

Efficiency (weak scale) =1TB 1 node compute time / 1TB 8 nodes compute time

= 33268/67268 = 49%

V. SPARK TERASORT

First we use hadoop-mapreduce-examples-2.7.4.jar (hadoop sample program) to generate 128GB and 1TB file in HDFS directly. Then we use spark RDD program tosort each file.[6]

A. We Split the Program into Four Parts.

1) SC.textFile function: Spark use textFile to store file in RDD format.
2) Map function: Spark use flatmap to split each in line of data, then use map to convert each line to <key, value> pair, which key is first 10 characters, and valueis remaining characters.
3) Reduce function: Spark use sortByKey to group key for each <key, value> pair, than use reduce action to change each <key, value> pair to line string object.
4) SC.saveAsTextFile: Finally Spark use parallelize to transform object into RDD format, then use saveAsTextFile to store in external file system.

B. System Configuration

Amazon machine image (AMI)

1) Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-82f4dae7
2) Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical
3) With 128GB terasort, we used Amazon ubuntu image 1 x i3.large instance.
4) i3.large (9 ECUs, 2 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 15.25 GiB memory, EBS only) 1 x 475 (SSD)
5) With 1TB terasort, we used Amazon ubuntu image 1 x i3.4xlarge instance
6) i3.4xlarge (53 ECUs, 16 vCPUs, 2.3 GHz, Intel Broadwell E5-2686v4, 122 GiB memory, EBS only)
7) Environment setting: java-8-openjdk-amd64, hadoop-2.7.4,spark-2.2.0-bin-hadoop2.7, python3
8) We mounted the EBS volume to boot and combine the disks into a RAID-0 to achieve the best possible performance.

a) *Experiment 1:* We use previous hadoop experiment environment and then install spark and python package. Then we changed environment variable in./bashrc. Now we ready to run Spark sorting.

b) *Experiment 2:* Different preparation with 128GB is spark configuration. Regarding spark-defaults.conf, we set spark.kryoserializer.buffer.max 2045MB in order to solve insufficient memory problem. we change block size from 64MB to 1GB. Also we increase executor memory and core, with 8g and 5 respectively, in order to get the good performance more efficiently.[7].

The result of experiment is shown in below table:

	file size		
node number		128GB	1TB
	1	10835 sec	36240 sec

Table 2: 128GB and 1TB terasort

VI. PERFORMANCE CALCULATION AMONG EXPERIMENT 1 AND 2

A. [1xi3.large 128GB]

1) Compute Time (sec):10835 seconds This time is counted directly within the program.
2) Data Read (GB):128 GB This size is counted directly within the result-File system Counters.
3) Data Write (GB) : 128 GB This size is counted directly within the result-File system Counters.
4) I/O Throughput (MB/sec) : Since total computing time is 10835 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time =(61278+128)x 1000 /20115 = 3050 MB/sec

B. [1xi3.4xlarge 1TB]

1) Compute Time (sec) : 36240 seconds. This time is counted directly within the program.
2) Data Read (GB):1000 GB This size is counted directly within the result-File system Counters.
3) Data Write (GB):1000 GB This size is counted directly within the result-File system Counters.
4) I/O Throughput (MB/sec):Since total computing time is 36240 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time (3727284+1000) x 1000 / 67268 = 55424 MB/sec

C. [8xi.large 1TB]

1) Compute Time (sec)33268 seconds. This time is counted directly within the program.
2) Data Read (GB):465 GB This size is counted directly within the result-File system Counters.
3) Data Write (GB):1000 GB This size is counted directly within the result-File system Counters.
4) I/O Throughput (MB/sec): Since total computing time is 33268 seconds, the I/O Throughput (MB/sec) = total data size (read and write)/total compute time (465+1000) x 1000 / 33268 = 44 MB/sec

*Speedup (weak scale) = 8x * efficiency=8 x (10835/33268) = 2.4X*
Efficiency (weak scale) =128GB compute time / 1TB compute time= 10835/33268 = 30%

Experiment (instance/dataset)	Hadoop TeraSort	Spark TeraSort
Compute Time (sec) [1xi3.large 128GB]	20114	10835
Data Read (GB) [1xi3.large 128GB]	61278	61278
Data Write (GB) [1xi3.large 128GB]	128	128
I/O Throughput (MB/sec) [1xi3.large 128GB]	3050	5667
Compute Time (sec) [1xi3.4xlarge 1TB]	67268	36240
Data Read (GB) [1xi3.4xlarge 1TB]	3727284	3727284
Data Write (GB) [1xi3.4xlarge 1TB]	1000	1000
I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]	55424	102877
Compute Time (sec) [8xi3.large 1TB]	40000	
Data Read (GB) [8xi3.large 1TB]	465	
Data Write (GB) [8xi3.large 1TB]	1000	
I/O Throughput (MB/sec) [8xi3.large 1TB]	44	
Speedup (weak scale)	3.95	2.4
Efficiency (weak scale)	49%	30%

Table 3: Performance evaluation of TeraSort

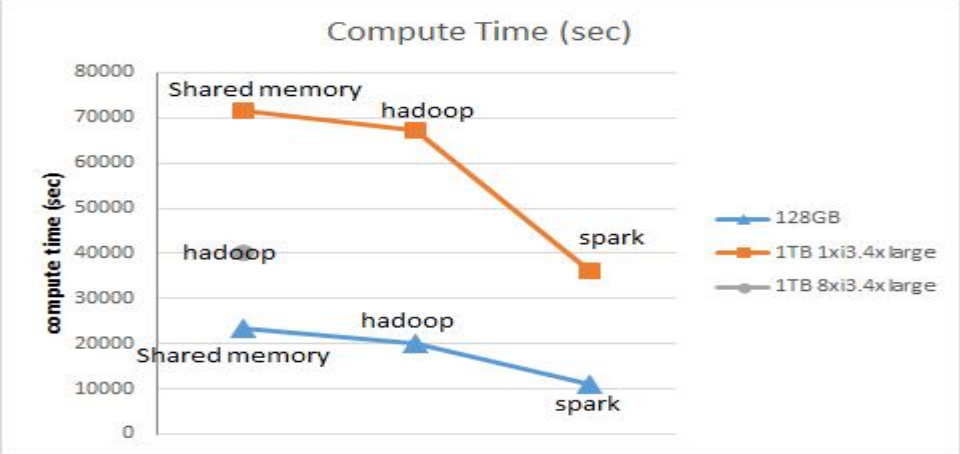


Fig: 1 Comparing computing time and throughput

VII. CONCLUSIONS

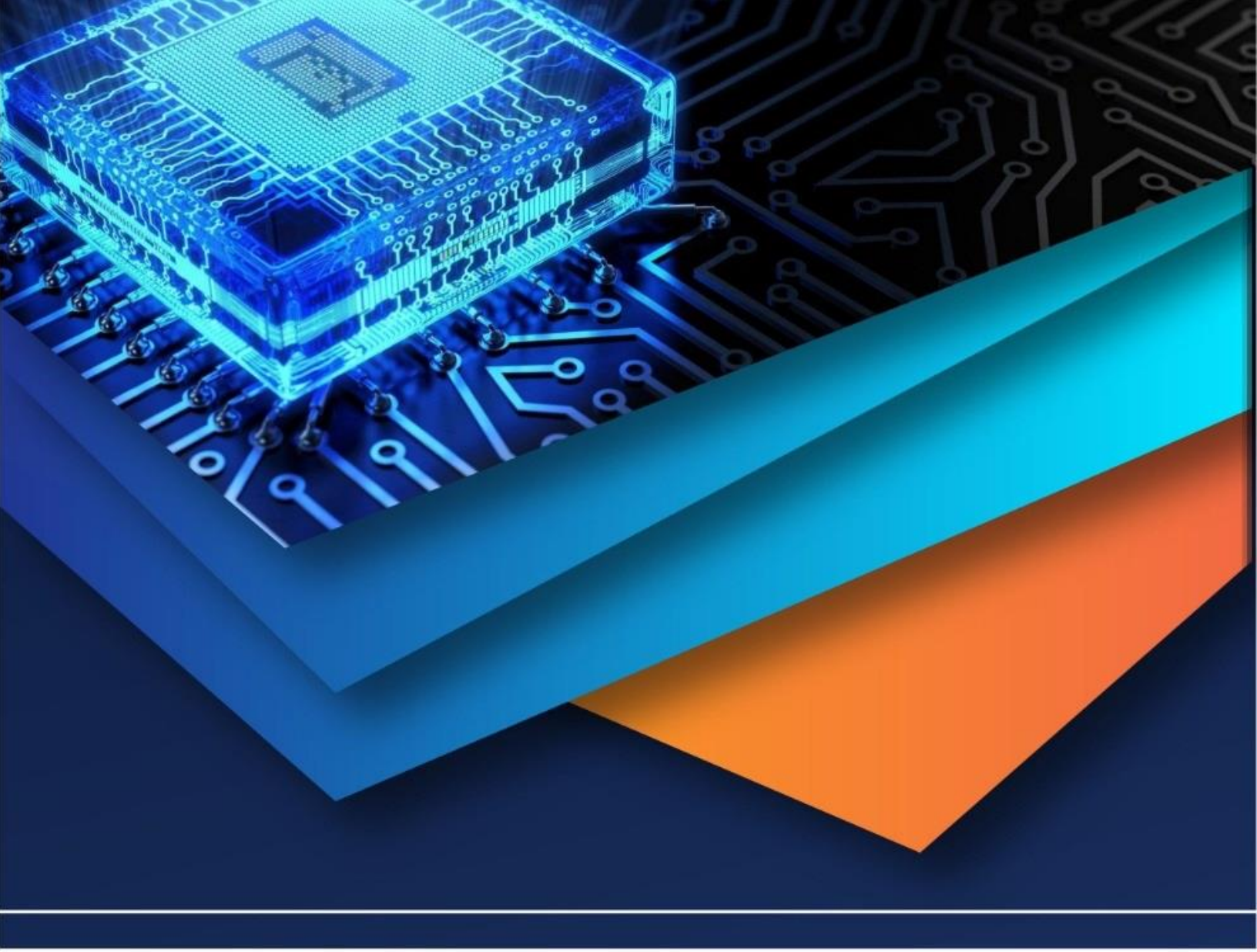
Spark performs best at 1 node compare to shared-memory and Hadoop compare to Throughput Comparison graph. Regarding our experiment, Spark would be best for 100 node scale because it stores the intermediate data so that we can directly use those data when we want at any time but the only constraint is that spark consumes a huge amounts ofmemory.

VIII. ACKNOWLEDGMENT

The heading of the Acknowledgment section and the References section must not be numbered.
Causal Productions wishes to acknowledge Michael Shell and other contributors for developing and maintaining the IEEE LaTeX style files which have been used in the preparation of this template. To see the list of contributors, please refer to the top of file IEEETran.cls in the IEEE LaTeX distribution.

REFERENCES

[1] Apache Mesos. http://mesos.apache.org.
[2] Cloudera: Data access based on Locality and Storage Type. http://blog.cloudera.com/blog/2014/08/ new-in-cdh-5-1-hdfs-read-caching/.
[3] Cloudera Spark Roadmap. https:// 2s7gjr373w3x22jf92z99mgm5w-wpengine.netdna-ssl. com/wp-content/uploads/2015/09
[4] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics." in CIDR, vol. 11, 2011, pp. 261–272.
[5] H. Herodotou and S. Babu, "Profiling, what-if analysis, and cost-based optimization of mapreduce programs," Proc. of the VLDB Endowment, vol. 4, no. 11, pp. 1111–1122, 2011.
[6] D. Boyd and K. Crawford, "Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon," Information, communication & society, vol. 15, no. 5, pp. 662–679, 2012
[7] O. Olmez and A. Ramamoorthy, "Fractional repetition codes with flexible repair from combinatorial designs," IEEE Trans. on Info. Th., vol. 62, no. 4, pp. 1565 –1591, 2016.
[8] "Repository of TeraSort for prior implementation." [Online]. Available: https://github.com/AvestimehrResearchGroup/Coded-TeraSort/ tree/IgnoreMemoryTime



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY