A Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using HiBench

Nasim Ahmed (nasim751@yahoo.com)
Massey University Institute of Natural and Mathematical Sciences https://orcid.org/0000-0001-5663-0042
Andre L. C. Barczak
Massey University - Albany Campus
Teo Susnjak
Massey University - Albany Campus
Mohammad Rashid
Massey University - Albany Campus

Research

Keywords: HiBench, BigData, Hadoop, MapReduce, Benchmark, Spark

Posted Date: December 2nd, 2020

DOI: https://doi.org/10.21203/rs.3.rs-43526/v2

Version of Record: A version of this preprint was published on December 14th, 2020. See the published version at https://doi.org/10.1186/s40537-020-00388-5.

RESEARCH

A Comprehensive Performance Analysis of Apache Hadoop and Apache Spark for Large Scale Data Sets Using HiBench

N. Ahmed[1*†], Andre L. C. Barczak[1], Teo Susnjak[1] and Mohammed A. Rashid[2]

*Correspondence:
nasim751@yahoo.com
[1]School of Natural and Computational Sciences, Massey University, Albany, 0745 Auckland, New Zealand
Full list of author information is available at the end of the article
†Equal contributor

Abstract

Big Data analytics for storing, processing, and analyzing large-scale datasets has become an essential tool for the industry. The advent of distributed computing frameworks such as Hadoop and Spark offers efficient solutions to analyze vast amounts of data. Due to the application programming interface (API) availability and its performance, Spark becomes very popular, even more popular than the MapReduce framework. Both these frameworks have more than 150 parameters, and the combination of these parameters has a massive impact on cluster performance. The default system parameters help the system administrator deploy their system applications without much effort, and they can measure their specific cluster performance with factory-set parameters. However, an open question remains: can new parameter selection improve cluster performance for large datasets? In this regard, this study investigates the most impacting parameters, under resource utilization, input splits, and shuffle, to compare the performance between Hadoop and Spark, using an implemented cluster in our laboratory. We used a trial-and-error approach for tuning these parameters based on a large number of experiments. In order to evaluate the frameworks of comparative analysis, we select two workloads: WordCount and TeraSort. The performance metrics are carried out based on three criteria: execution time, throughput, and speedup. Our experimental results revealed that both system performances heavily depends on input data size and correct parameter selection. The analysis of the results shows that Spark has better performance as compared to Hadoop when data sets are small, achieving up to 2 times speedup in WordCount workloads and up to 14 times in TeraSort workloads when default parameter values are reconfigured.

Keywords: HiBench; BigData; Hadoop; MapReduce; Benchmark; Spark

1 Introduction

Hadoop [1] has become a very popular platform in the IT industry and academia for its ability to handle large amounts of data, along with extensive processing and analysis facilities. Different users produce these large datasets, and most of data are unstructured, increasing the requirements for memory and I/O. Besides, the advent of many new applications and technologies brought much larger volumes of complex data, including social media, e.g., Facebook, Twitter, YouTube, online shopping, machine data, system data, and browsing history [2]. This massive amount of digital data becomes a challenging task for the management to store, process, and analyze.

The conventional database management tools are unable to handle this type of data [3]. Big data technologies, tools, and procedures allowed organizations to capture, process speedily, and analyze large quantities of data and extract appropriate information at a reasonable cost.

Several solutions are available to handle this problems [4]. Distributed computing is one possible solution considered as the most efficient and fault-tolerant method for companies to store and process massive amounts of data. Among this new group of tools, MapReduce and Spark are the most commonly used cluster computing tools. They provide users with various functions using simple application programming interfaces (API). MapReduce is a framework used for distributed computing used for parallel processing and designed purposely to write, read, and process bulky amounts of data [1, 5, 6]. This data processing framework is comprised of three stages: Map phase, Shuffle phase and Reduce phase. In this technique, the large files are divided into several small blocks of equal sizes and distributed across the cluster for storage. MapReduce and Hadoop distributed file systems (HDFS) are core parts of the Hadoop system, so computing and storage work together across all nodes that compose a cluster of computers [7].

Apache Spark is an open-source cluster-computing framework [8]. It is designed based on the Hadoop and its purpose is to build a programing model that "fits a wider class of applications than MapReduce while maintaining the automatic fault tolerance" [9]. It is not only an alternative to the Hadoop framework but it also provides various functions to process real streaming data. Apart from the map and reduce functions, Spark also supports MLib1, GraphX, and Spark streaming for big data analysis. Hadoop MapReduce processing speed is slow because it requires accessing disks for reads and writes. On the other hand, Spark uses memory to store data reducing the read/write cycle [1]. In this paper, we have addressed the above mentioned critical challenges. According to our knowledge, none of the previous works have addressed those challenges. Our proposed work will help the system administrators and researchers to understand the system behavior when processing large scale data sets. The main contributions of this paper are as follows:

- We introduced a comprehensive empirical performance analysis between MapReduce and Spark frameworks by correlating resource utilization, splits size, and shuffle behavior parameters.
- We accomplished comprehensive comparison work between Hadoop and Spark where large scale datasets (600GB) are used for the first time. The experiments present the various aspects of cluster performance overhead. We applied two Hibenchmark workloads to test the efficiency of the system under MapReduce and Spark, where the data sets are repeatedly changing.
- We selected several parameters covering different aspects of system behavior. Multiple parameters are used to tune job performance. The results of the analysis will facilitate job performance tuning and enhance the freedom to modify the ideal parameters to enhance job efficiency.
- We measured the scaleability of the experiment by repeating the experiment three times, getting the average execution time for each job. Besides, we investigate the system execution time, maximum sustainable throughput and speedup.

- We used a real cluster capable of handling large scale data set (600GB) with benchmarking tools for a comprehensive evaluation of MapReduce and Spark.

The remainder of the paper is organized as follows: Section 2 presents a critical review of related research works, and then describes Hadoop and Spark systems. The difference between Hadoop and Spark is explained in Section 3. The experimental setup is presented in Section 4. In Section 5, we explain the chosen parameters and tuning approach. Section 6 presents the performance analysis of the results and finally, we conclude in Section 7.

2 Related Work

Shi et al. [10] proposed two profiling tools to quantify the performance of the MapReduce and Spark framework based on a micro-benchmark experiment. The comparative study between these frameworks are conducted with batch and iterative jobs. In their work, the authors consider three components: shuffle, executive model, and caching. The workloads, Wordcount, k-means, Sort, Linear Regression, and PageRank, are chosen to evaluate the system behavior based on CPU bound, disk-bound, and network bound [11]. They disabled map and reduce function for all workloads apart of a Sort. For the Sort, the reduce task is configured up to 60 map tasks, and the reduce task conFigured to 120. The map output buffer is allocated to 550MB to avoid additional spills for sorting the map output. Spark intermediate data are stored in 8 disks where each worker is configured with four threads. The authors claim that Spark is faster than MapReduce when WordCount runs with different data sets (1GB, 40GB, and 200GB). The TeraSort is used by sort-by-key() function. They have found that Spark is faster than MapReduce when the data set is smaller (1GB), but Mapreduce is nearly two times faster than Spark when the data set is of bigger sizes (40GB or 100GB). Besides, Spark is one and a half times faster than MapReduce with machine learning workloads such as K-means and Linear Regression. It is claimed that in a subsequent iteration, Spark is five times faster than MapReduce due to the RDD caching and Spark-GraphX is four times faster than MapReduce.

Li et al. [12] proposed a spark benchmarking suite [13], which significantly enhances the optimization of workload configuration. This work has identified the distinct features of each benchmark application regarding resource consumption, the data flow, and the communication pattern that can impact the job execution time. The applications are characterized based on extensive experiments using synthetic data sets. There are ten different workloads such as Logistic Regression, Support Vector Machine, Matrix Factorization, Page Rank, Tringle Count, SVD++, Hive, RDD Relation, Twitter, and PageView used with different input data sizes. An eleven nodes virtual cluster is used to analyze the performance of the workloads. The workload analysis is carried out concerning CPU utilization, memory, disk, and network input/output consumption at the time of job execution. They have found that most of the workloads spend more than 50% execution time for MapShuffle-Tasks except logistic regression. They concluded that the job execution time could be reduced while increasing task parallelism to leverage the CPU utilization fully.

Marcue et al. [14] present the comparative analysis between Spark and Flink frameworks for large scale data analysis. This work proposed a new methodology

for iterative workloads (K-Means, and Page Rank) and batch processing workloads (WordCount, Grep, and TeraSort) benchmarking. They considered four most important parameters that impact scalability, resource consumption, and execution time. Grid 5000 [15] has used upto 100 nodes cluster deploying Spark and Flink. They have recommended that Spark parameter (i.e., parallelism and partitions) configuration is sensitive and depends on data sets, while the Flink is highly extensive memory oriented.

Samadi et al. [7] has investigated the criteria of the performance comparison between Hadoop and Spark framework. In his work, for an impartial comparison, the input data size and configuration remained the same. Their experiment used eight benchmarks of the HiBench suite [13]. The input data was generated automatically for every case and size, and the computation was performed several times to find out the execution time and throughput. When they deployed microbenchmark (Short and TeraSort) on both systems, Spark showed higher involvement of processor in I/Os while Hadoop mostly processed user tasks. On the other hand, Spark's performance was excellent when dealing with small input sizes, such as micro and web search (Page Rank). Finally, they concluded that Spark is faster and very strong for processing data in-memory while Hadoop MapReduce performs maps and reduces function in the disk.

In another paper, Samadi et al. [9] proposed a virtual machine based on Hadoop and Spark to get the benefit of virtualization. This virtual machine's main advantage is that it can perform all operations even if the hardware fails. In this deployment, they have used Centos operating system built a Hadoop cluster based on a pseudo-distribution mode with various workloads. In their experiments, they have deployed the Hadoop machine on a single workstation and all other demos on its JVM. To justify the big data framework, they have presented the results of Hadoop deployment on Amazon Elastic Computing (EC2). They have concluded that Hadoop is a better choice because Spark requires more memory resources than Hadoop. Finally, they have suggested that the cluster configuration is essential to reduce job execution time, and the cluster parameter configuration must align with Mappers and Reducers.

The computational frameworks, namely Apache Hadoop and Apache Spark, were investigated by [16]. In this investigation, the Apache webserver log file was taken into consideration to evaluate the two frameworks' comparative performance. In these experiments, they have used Okeanos's virtualized computing resources based on infrastructures as a Service (IaaS) developed by the Greek Research and Technology Network [16]. They proposed a number of applications and conducted several experiments to determine each application's execution time. They have used various input files and the slave nodes to find out the execution time. They have found that the execution time is proportional to the input data size. They have concluded that the performance of Spark is much better in most cases as compared to Hadoop.

Satish and Rohan [17] have shown a comparative performance study between Hadoop MapReduce and Spark-based on the K-means algorithm. In this study, they have used a specific data set that supports this algorithm and considered both single and double nodes when gathering each experiment's execution time. They have concluded that the Spark speed reaches up to three times higher than the

MapReduce, though Spark performance heavily depends on sufficient memory size [18].

Lin et al. [19] have proposed a unified cloud platform, including batch processing ability over standalone log analysis tools. This investigation has considered four different frameworks: Hadoop, Spark, and warehouse data analysis tools Hive and Shark. They implemented two machine learning algorithms (K-means and PageRank) based on this framework with six nodes to validate the cloud platform. They have used different data sizes as inputs. In the case of K-means, as the data size increased and exceed memory size, the latency schedule and overall Spark performance degraded. However, the overall performance was still six times higher than Hadoop on average. On the other hand, Shark shows significant performance improvement while using queries directly from disk.

Petridis et al. [20] have investigated the most important Spark parameters shown in table 4 and given a guideline to the developers and system administrators to select the correct parameter values by replacing the default parameter values based on trial-and-error methodology. Three types of case studies with different categories such as Shuffle Behavior, Compression and Serialization, and Memory Management parameters were performed in this study. They have highlighted the impact of memory allocation and serialization when the number of cores and default parallelism values change. Therefore, there are 12 parameters chosen with three benchmarking applications: sort-by-key, shuffling, and k-means. The sort-by-key experiments used both 1 million and 1 billion key-values of lengths 10 and 90 bytes and the optimal degree of partition is set to 640. The Hash performance is increased to 127 seconds, which is 30 seconds faster than the default parameter, and shuffle.file.buffer is increased by 140 seconds. The rest of the parameters do not play any important role in improving the performance. For another Shuffling experiment, they used a 400GB dataset. The Hash shuffle performance is degraded by 200 seconds, and Tungsten-Sort speed is increased by 90 seconds. By decreasing the buffer size from 32KB to 15KB, the system performance was degraded by about 135s, which is more than 10% from the primary selection. For K-means, they used two sizes of data input (100 MB and 200 MB). They have not found significant k-means performance improvement by changing the parameters. Therefore, they have concluded that based on their methodology, the speedup achievement is 10-fold. However, the main challenges of tuning Hadoop and Spark configuration parameters are due to the complicated behavior of distributed large scale systems while the parameter selection is not always trivial for the system administrators. Inappropriate combination of parameter values can affect the overall system performance. Inappropriate combination of parameter values can affect the overall system performance.

The published literature in Table 1 presents some empirical studies. None of these studies have considered larger data sizes (600GB), more parameters, and real clusters. In our study, we chose a conventional trial-and-error approach [20], larger data set, and 18 important parameters (listed in tables 3 and 4)from resource utilization, input splits, and shuffle category.

3 Difference Between Hadoop and Spark

Hadoop [21] is a very popular and useful open-source software framework that enables distributed storage, including the capability of storing a large amount of

Table 1 Published Related Work

Author's	Date	Workloads	Data Size	Parameters	Hardware
Lin.et.al. [19]	2013	K-means PageRank	10,000 to 20 mil points 1 mil to 10 mil points	Log Analysis	Nodes- 6, 2 CPU cores 4GB memory per node Nodes- 4, 16 CPU cores 48GB memory per node
Satish & Rohan [17]	2015	K-means	62MB - 1240MB	Default	Virtual machine Nodes- 2, 4GB RAM and 500GB (HD)
Yasir Samadi. et.al. [7]	2016	Micro Benchmarks Web Search SQL Machine Learning	183MB - 328MB 5000 to 12*10e4 pages	3	Virtual machine Disk(SDD)- 40GB
Petridis.et.al. [20]	2017	K-means Shuffling and Sort-by-Key	400GB	12	Barcelona Supercomputing Center
Mavridis.et.al. [16]	2017	Spark SQL and Spark Hive	1.1GB, 1.5GB and 11GB	Log Analysis	Virtual machine- 6 Memory- 8GB Master node- 8cores Salve node- 4cores
Yasir Samadi. et.al. [9]	2018	Micro Benchmarks Web Search SQL Machine Learning	1GB, 5GB and 8GB	3	Virtual machine Disk(SDD) - 40GB
Proposed Experiments	**2020**	**WordCount and TeraSort**	**50GB - 600GB**	**18**	**SNCC, Production Cluster** **CPU cores - 80** **Total Storage - 60TB** **Master node - 1** **Slaves nodes - 9**

big datasets across clusters. It is designed in such a way that it can scale up from a single server to thousands of nodes. Hadoop processes large data concurrently and produces fast results. With Hadoop, the core parts are Hadoop Distributed File System (HDFS) and MapReduce.

HDFS [22] splits the files into small pieces into blocks and saves them into different nodes. There are two kinds of nodes on HDFS: data-nodes (worker) and name-nodes (master nodes) [23, 24]. All the operations, including delete, read, and write, are based on these two types of nodes. The workflow of HDFS is like the following flow: firstly, the name-node asks for access permission. If accepted, it will turn the file name into a list of HDFS block IDs, including the files and the data-nodes that saved the blocks related to that file. The ID list will then be sent back to the client, and the users can do further operations based on that.

MapReduce [25] is a computing framework that includes two operations: Mappers and Reducers. The mappers will process files based on the map function and transfer them into the new key-value pairs [26]. Next, the new key-value pairs are assigned to different partitions and sorted based on their keys. The combiner is optional and can be recognized as a local reduces operation which allows counting the values with the same key in advance to reduce the I/O pressure. Finally, partitions will divide the intermediate key-value pairs into different pieces and transfer them to a reducer. MapReduce needs to implement one operation: shuffle. Shuffle means transferring the mapper output data to the proper reducer. After the shuffle process is finished, the reducer starts some copy threads (Fetcher) and obtains the output files of the

map task through HTTP [27]. The next step is merging the output into different final files, which are then recognized as reducer input data. After that, the reducer processes the data based on the reduced function and writes the output back to the HDFS. Figure 1 depicts a Hadoop MapReduce architecture.

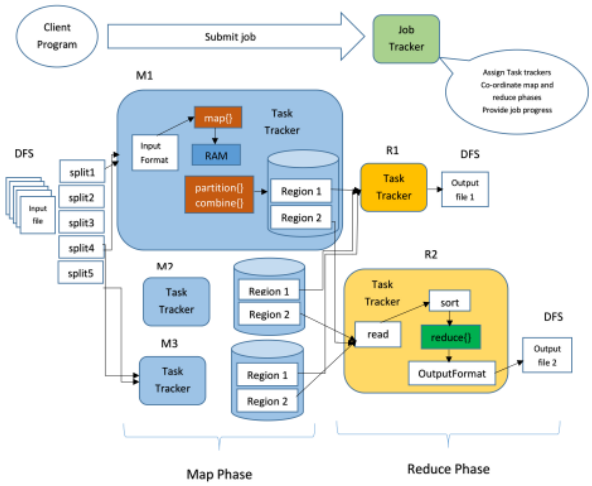


Figure 1 Hadoop MapReduce architecture [1]

Spark became an open-source project from 2010. Zahari has developed this project at UC Berkely's AMPLab in 2009 [28, 4]. Spark offers numerous advantages for developers to build big data applications. Spark proposed two important terms: Resilient Distributed Datasets (RDD) and Directed Acyclic Graph (DAG). These two techniques work together perfectly and accelerate Spark up to tens of times faster than Hadoop under certain circumstances, even though it usually only achieves a performance two to three times more quickly than MapReduce. It supports multiple sources that have a fault tolerance mechanism that can be cached and supports parallel operations. Besides, it can represent a single dataset with multiple partitions. When Spark runs on the Hadoop cluster, RDDs will be created on the HDFS in many formats supported by Hadoop, likewise text and sequence files. The DAG scheduler [29] system expresses the dependencies of RDDs. Each spark job will create a DAG and the scheduler will drive the graph into the different stages of tasks then the tasks will be launched to the cluster. The DAG will be created in both maps and reduce stages to express the dependencies fully. Figure 2 illustrates the iterative operation on RDD. Theoretically, limited Spark memory causes the performance to slow down.

4 Experimental Setup

4.1 Cluster Architecture

In the last couple of years, many proposals came from different research groups about the suitability of Hadoop and Spark frameworks when various types of data of different sizes are used as input in different clusters. Therefore, it becomes necessary to study the performance of the frameworks and understand the influence of various parameters. For the experiments, we will present our cluster performance

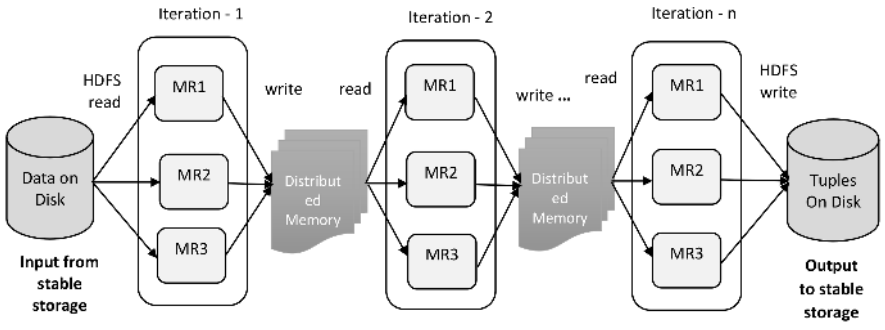


Figure 2 Spark workflow [30]

based on MapReduce and Spark using the HiBench suite [22, 31]. In particular, we have selected two Hibench workloads out of thirteen standard workloads to represent the two types of jobs, namely WordCount (aggregation job) [32], and TeraSort (shuffle job) [33] with large datasets. We selected both the workloads because of their complex characteristics to study how efficiently both the workloads analyze the cluster performance by correlating MapReduce and Spark function with a combination of groups of parameters.

4.2 Hardware and Software Specification

The experiments were deployed in our own cluster. The cluster is configured with 1 master and 9 slaves nodes which is presented in fig.3. The cluster has 80 CPU cores and 60TB local storage. The implemented hardware is suitable for handling various difficult situations in Spark and MapReduce.

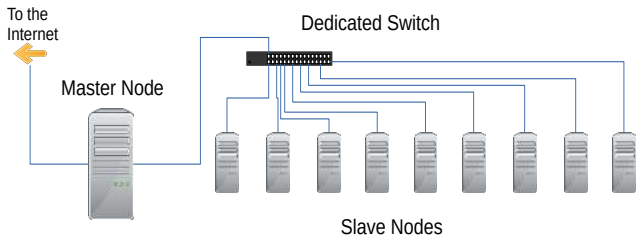


Figure 3 Hadoop Cluster Nodes

The detailed Hadoop cluster and software specifications are presented in Table 2. All our jobs run in Spark and MapReduce. We have selected Yarn as a resource manager, which can help us monitor each working node's situation and track the details of each job with its history. We have used *Apache Ambari* to monitor and profile the selective workloads running on Spark and MapReduce. It supports most of the Hadoop components, including HDFS, MapReduce, Hive, Pig, Hbase, Zookeeper, Sqoop, and Hcatalog" [34]. Besides, Ambari supports the user to control the Hadoop cluster on three aspects, namely provision, management, and monitoring.

Table 2 Experimental Hadoop Cluster

Server Configuration	Processor	2.9 GHz
	Main Memory	64 GB
	Local Storage	10 TB
Node Configuration	CPU	Intel(R) Xeon(R) CPU E3-1231 v3 @ 3.40GHz
	Main Memory	32 GB
	Number of Nodes	10
	Local Storage	6 TB each, 60TB total
	CPU cores	8 each, 80 total
Software	Operating System	Ubuntu 16.04.2 (GNU/Linux 4.13.0-37-generic x86 64)
	JDK	1.7.0
	Hadoop	2.4.0
	Spark	2.1.0
Workload	Micro Benchmarks	WordCount, and TeraSort

4.3 Workloads

As stated above, in this study we chose two workloads for the experiments [32, 33]:

WordCount: The wordCount workload is map-dependent, and it counts the number of occurrences of separate words from text or sequence file. The input data is produced by *RandomTextWriter*. It splits into each word by using the map function and generates intermediate data for the reduce function as a key-value [35]. The intermediate results are added up, generating the final word count by the reduce function.

TeraSort: The TeraSort package was released by Hadoop in 2008 [36] to measure the capabilities of cluster performance. The input data is generated by the *TeraGen* function which is implemented in Java. The TeraSort function does the sorting using the MapReduce, and the TeraValidate function is used to validate the output of the sorted data. For both workloads, we used up to 600 GB of synthetic input data generated using a string concatenation technique.

5 The Parameters of Interest and Tuning Approach

Tuning parameters in Apache Hadoop and Apache Spark is a challenging task. We want to find out which parameters have important impacts on system performance. The configuration of the parameters needs to be investigated according to work-load, data size, and cluster architecture. We have conducted a number of experiments using Apache Hadoop and Apache Spark with different parameter settings. For this experiment, we have chosen the core MapReduce and Spark parameter setting from resource utilization, input splits and shuffle groups. The selected tuned parameters with their respective tuned values on the map-reduce and Spark category are shown in Tables 3 and 4.

6 Results and Discussion

In this section, the results obtained after running the jobs are evaluated. We have used synthetic input data and used the same parameter configuration for a realistic comparison. Each test was repeated 3 times, and the average runtime was plotted in each graph. For both frameworks, we show the execution time, throughput, and speedup to compare the two frameworks and visualize the effects of changing the default parameters..

Table 3

Configuration Parameters Category	Hadoop	Tuned Values
Resource Utilization	mapreduce.reduce.memory	8GB
	mapred.reduce.task	16,384MB, 25,600MB
	mapreduce.reduce.cpu.vcores	4
Input Split	mapred.min.split.size, mapred.max.split.size	128MB (default), 256MB, 512MB, 1024MB
Shuffle	i/o.sort.mb	25, 50, 75, 100
	i/o.sort.factor	512, 1024, 1536, 2047
	mapreduce.reduce.shuffle.parallelcopies	50, 100, 150, 200
	mapreduce.task.io.sort.factor	15, 30, 45, 60

Table 4

Configuration Parameters Category	Spark	Tuned Values
Resource Utilization	num-executors	50
	executor-cores	4
	executor-memory	8GB
Input Split	spark.hadoop.MapReduce.input .fileinputformat.split.minsize	128MB (default), 256MB, 512MB, 1024MB
Shuffle	spark.shuffle.file.buffer	16k, 32k (default), 48k, 64k
	spark.reducer.maxSizeInFlight	32M, 48M (default), 64M, 96M
	spark.hadoop.dfs.replication	1
	spark.default.parallelism	80, 100, 200, 300

6.1 Execution Time

The execution time is affected by the input data sizes, the number of active nodes, and the application types. We have fixed the same parameters for the fair comparative analysis, such as the number of executors to 50, executor memory to 8GB, executor cores to 4.

Figures 4-a and 4-b show how MapReduce and Spark execution time depend on the datasets' size and the different input splits and shuffle parameters. The execution time of MapReduce WordCount workload with the default input split size (128MB) and shuffle parameter (*sort.mb* 100, *sort.factor* 2047) obtained better execution time for entire data sizes compared to other parameters. Hadoop Map and Reduce function behave better because of their faster execution time and overlooked container initialization overhead for specific workload types. This result suggests that the default parameter is more suitable for our cluster when using data sizes from 50GB to 600GB.

In fig.4-c the default input splits of Spark is 128MB. Previously, we have mentioned that the number of executors, executor memory, and executor cores are fixed. From the above fig.4-c,we see that the execution time of input split size 256MB outperforms the default set up until 450GB data sizes. In fact, the default splits size (128MB) is more efficient when the data size is larger than the 450GB. Notably, we can see that the default parameter shows better execution performance when the data set reaches 500GB or above. The new parameter values can improve the processing efficiency by 2.2% higher than the default value (128MB). Table 5 presents the experimental data of WordCount workload between MapReduce and Spark while the default parameters are changing.

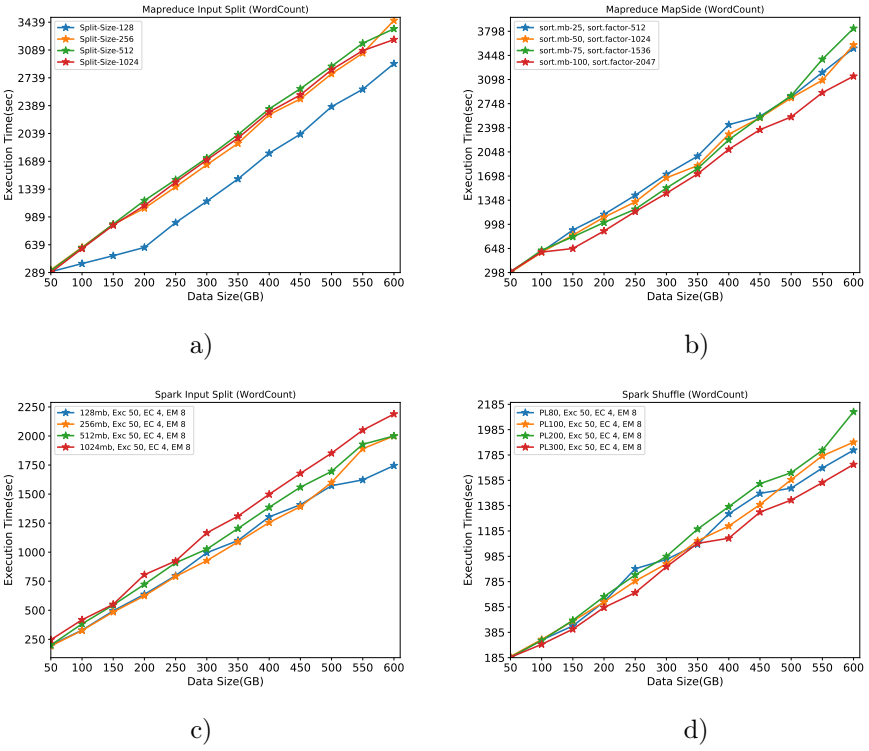


Figure 4 The performance of the WordCount application with a varied number of input splits and shuffle tasks.

For the Spark shuffle parameter, we have chosen the default serializer, the *(JavaSerializer)* because of the simplicity and easy control of the performance of the serialization [37]. In this category, the serializer is PL100 object [38]. We can see from figure 4-d that the improvement rate is significantly increased when we set the PL value to 300. It is evident that the best performance is achieved for sizes larger than 400GB. Also, it shows that when tuning the PL value to 300, the system can achieve a 3% higher improvement for the rest of the data sizes. Consequently, we can conclude that input splits can be considered an important factor in enhancing Spark WordCount jobs' efficiency when executing small datasets.

Figure 5-a is comparing MapReduce TeraSort workloads based on input splits that include default parameters. In this analysis, we have set (Red_Task and InSp) value fixed with default split size 128MB. We have changed the parameter values and tested whether the splits' size can keep the impact on the runtime. So, for this reason, we have selected three different sizes: 256MB, 512MB, and 1024MB. We have observed that with a split size of 256MB, the execution performance is increased by around 2% in datasets with up to 300GB. On the contrary, when the data sizes are larger than 300GB, the default size outperforms split size equals 512MB. Moreover, we have noticed that the improvement rates are similar when the data sizes are smaller than 200GB.

Figure 5-b illustrates the execution performance with the MapReduce shuffle parameter for the TeraSort workload. We have seen that the average execution

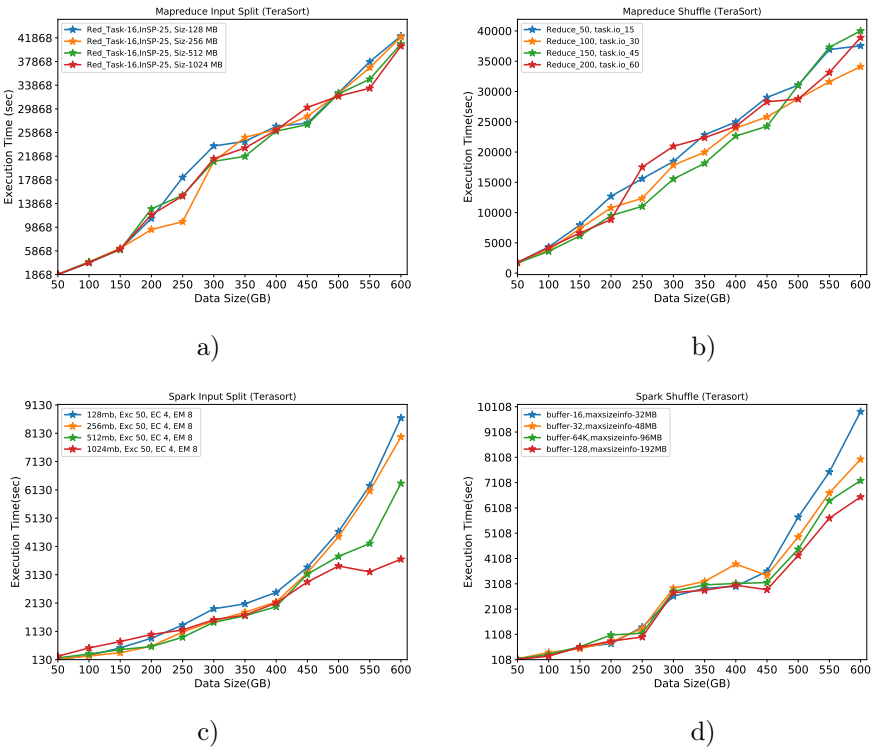


Figure 5 The performance of the TeraSort application with a varied number of input splits and shuffle tasks.

time behaves linearly for sizes up to 450GB when the parameter change to (*Reduce_150* and *task.io_45*) as compared to the default configuration (*Reduce_100* and *task.io_30*). Besides, We have also noticed that the default configuration is outperforming all other settings when the data sizes are larger than 450GB. So, we can conclude that by changing the shuffled value, the system execution performance improves by 1%. In general, this is very unlikely that the default size has optimum performance for larger data sizes.

Figure 5-c illustrates the Spark input split parameter execution performance analysis for the TeraSort workload. The Spark executor memory, number of executors, and executor memory are fixed while changing the block size to measure the execution performance. Apart from the default block size (128MB), there are 3 pairs (256MB, 512MB, and 1024MB) of block size is taken into this consideration. Our results revealed that the block size 512 MB and 1024MB present better runtime for sizes up to 500GB data size. We have also observed a significant performance improvement achieved by the 1024 block size, which is 4% when the data size is larger than 500GB. Thus, we can conclude that by adding the input splits block size for large scale data size; Spark performance can be increased.

Figure 5-d shows Spark shuffle behaviour performance for TeraSort workloads. We have taken two important default parameters (*buffer*=32, *spark.reducer.maxSizeIn Flight*=48MB) into our analysis. We have found that when the buffer and *maxSizeInFlight* are increased by 128 and 192, the execution performance increased

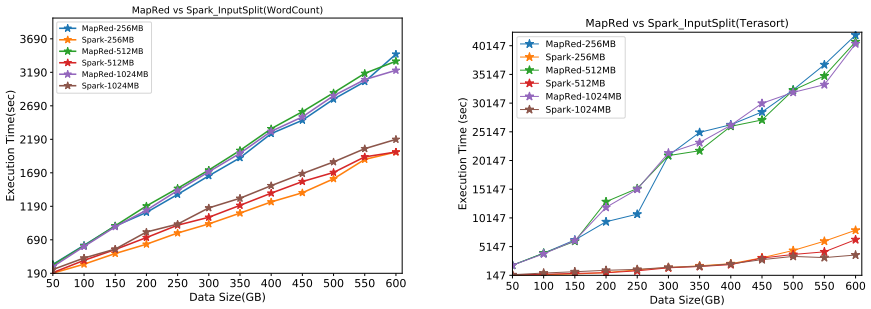
Table 5 The best execution time of MapReduce and Spark with WordCount workload

	split sizes (MB)	execution time (sec)
MapReduce input splits (WordCount)	128	2376
Spark input splits (WordCount)	256	1392
MapReduce shuffle (WordCount)	100	2371
Spark shuffle (WordCount)	300	1334

Table 6 The best execution time of MapReduce and Spark with Terasort workload

	split sizes (MB)	execution time (sec)
MapReduce input splits (TeraSort)	256	21014
Spark input splits (TeraSort)	512 & 1024	3780 & 3439
MapReduce shuffle (TeraSort)	150 & 45	24250
Spark shuffle (TeraSort)	128 & 192	6540

proportionally up to 600GB data sizes. Our results show that the default execution is equal, with a tested value of up to 200GB data sizes. The possible reason for this performance improvement is the larger number of splits size for different executors. Table 6 presents the experimental data of the TeraSort workload between MapReduce and Spark, while the default parameters are changing.



a) b)

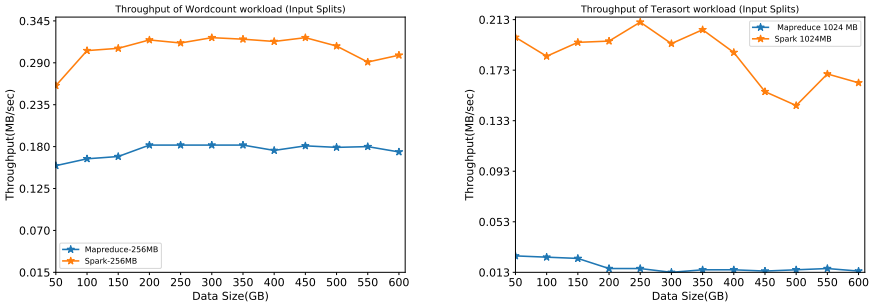
Figure 6 The comparison of Hadoop and Spark with WordCount and TeraSort workload with varied input splits and shuffle tasks.

Figure 6-a illustrates the comparison between Spark and MapReduce for WordCount and TeraSort workloads after applying the different input splits. We have observed that Spark with WordCount workloads shows higher execution performance by more than 2 times when data sizes are larger than 300GB for WordCount workloads. For the smaller data sizes, the performance improvement gap is around 10 times. Fig. 6-b shows a TeraSort workload for MapReduce and Spark. We can see that Spark execution performance is linear and proportionally larger as the data size increase. Also, we noticed that the runtime for MapReduce jobs are not as linear in relation to the data size as Spark jobs. The possible reason could be unavoidable job action on the clusters and as a result that the dataset is larger than the available RAM. So, we conclude that MapReduce has slower data sharing capabilities and a longer time to the read-write operation than Spark [4].

6.2 Throughput

The throughput metrics are all in MB per second. For this analysis, we only considered the best results from each category. We have observed that MapReduce

throughput performance for the TeraSort workload is decreasing slightly as the data size crosses beyond 200GB. Besides, for the WordCount workload, the MapReduce throughput is almost linear. For the Spark TeraSort workload, it can be observed that the throughput is not constant, but for the WordCount workload, the throughput is almost constant. In this analysis, the main focus was to present the throughput difference between WordCount and TeraSort workload for MapReduce and Spark. We found that WordCount workload remains almost stable for most of the data sizes, and concerning the TeraSort workload, MapReduce remain stable than Spark (see Figure 7).



a) b)

Figure 7 Throughput of WordCount and TeraSort workload.

6.3 Speedup

Figures 8(a, b, c) show the Spark's speed up compared to MapReduce. Figures 8(a and b) depicts individual workload speedup. The best results are taken into this consideration from each category in order to get a speedup. From the above figures, we can see that as the data size increases, WordCount workload speedup decreases with some non-linearity. Besides, we can see that the TeraSort speedup decreases when data reaches sizes larger than 300GB. Notably, as the data size increases to more than 500GB for both workloads, the speedup starts to increase. Figure 8(c) illustrates the speedup comparison between the workloads. It can be seen that the TeraSort workload outperforms WordCount workload and achieves an all-time maximum speedup of around 14 times. The literature presents that Spark is up to ten times faster than Hadoop under certain circumstances and in normal conditions, and it only achieves a performance two to three times faster than MapReduce [39]. However, this study found that Spark performance is degraded when the input data size is big.

7 Conclusion

This article presented the empirical performance analysis between Hadoop and Spark based on a large scale dataset. We have executed WordCount and Terasort workloads and 18 different parameter values by replacing them with default set-up. To investigate the execution performance, we have used trial-and-error approach for tuning these parameters performing number of experiments on nine node cluster with a capacity of 600GB dataset. Our experimental results confirm that both

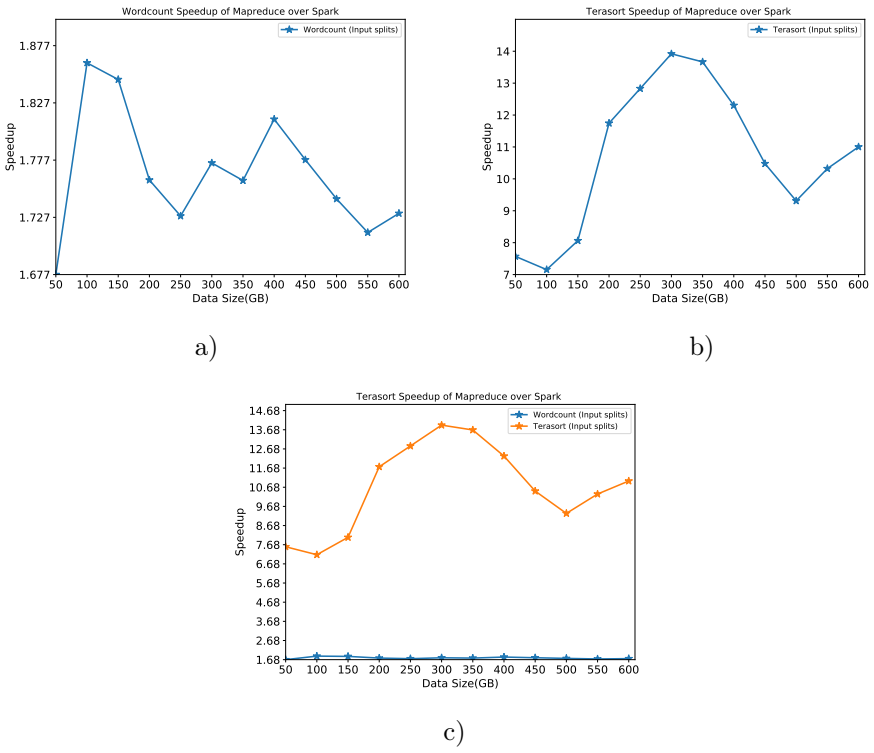


Figure 8 Spark over MapReduce speedup on input splits and shuffle.

Hadoop and Spark systems performance heavily depends on input data size and right parameter selection and tuning. We have found that Spark has better performance as compared to Hadoop by 2 times with WordCount work load and 14 times with Tera-Sort workloads respectively when default parameters are tuned with new values. Further more, the throughput and speedup results show that Spark is more stable and faster than Hadoop because of Spark data processing ability in memory instead of store in disk for the map and reduced function. We have also found that Spark performance degraded when input data was larger.

As future work, we plan to add and investigate 15 HiBench workloads, consider more parameters under resource utilization, parallelization, and other aspects, including practical data sets. The main focus would be to analyze the job performance based on auto-tuning techniques for MapReduce and Spark when several parameter configurations replace the default values.

Author's contributions

Ahmed was the main contributor of this work. He has done an initial literature review, data collection, experiments, prepare results, and drafted the manuscript. Andre and Teo deployed and configured the physical Hadoop cluster. Andre also worked closely with Ahmed to review, analyze, and manuscript preparation. Teo and Rashid helped to improve the final paper.

Acknowledgements

The authors acknowledge Sibgat Bazai for his valuable suggestions.

Availability of data and materials

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Funding
This work was not funded.

Author details
[1]School of Natural and Computational Sciences, Massey University, Albany, 0745 Auckland, New Zealand. [2]School of Natural and Computational Sciences, Massey University,, 0745 Auckland, New Zealand. [3]School of Natural and Computational Sciences, Massey University,, 0745 Auckland, New Zealand. [4]Department of Mechanical and Electrical Engineering, Massey University,, 0745 Auckland, New Zealand.

References

1. Apache Hadoop Documentation 2014. http://hadoop.apache.org/
2. Verma, A., Mansuri, A.H., Jain, N.: Big data management processing with hadoop mapreduce and spark technology: A comparison. In: 2016 Symposium on Colossal Data Analysis and Networking (CDAN), pp. 1–4 (2016). IEEE
3. Management Association, I.R.: Big Data: Concepts, Methodologies, Tools, and Applications. IGI Global, United States (2016)
4. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., Mccauley, M., Franklin, M., Shenker, S., Stoica, I.: Fast and interactive analytics over hadoop data with spark. Usenix Login **37**, 45–51 (2012)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)
6. Wang, G., Butt, A.R., Pandey, P., Gupta, K.: Using realistic simulation for performance analysis of mapreduce setups. In: Proceedings of the 1st ACM Workshop on Large-Scale System and Application Performance, pp. 19–26 (2009)
7. Samadi, Y., Zbakh, M., Tadonki, C.: Comparative study between hadoop and spark based on hibench benchmarks. In: 2016 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech), pp. 267–275 (2016). IEEE
8. Ahmadvand, H., Goudarzi, M., Foroutan, F.: Gapprox: using gallup approach for approximation in big data processing. Journal of Big Data **6**(1), 20 (2019)
9. Samadi, Y., Zbakh, M., Tadonki, C.: Performance comparison between hadoop and spark frameworks using hibench benchmarks. Concurrency and Computation: Practice and Experience **30**(12), 4367 (2018)
10. Shi, J., Qiu, Y., Minhas, U.F., Jiao, L., Wang, C., Reinwald, B., Özcan, F.: Clash of the titans: Mapreduce vs. spark for large scale data analytics. Proceedings of the VLDB Endowment **8**(13), 2110–2121 (2015)
11. Veiga, J., Expósito, R.R., Pardo, X.C., Taboada, G.L., Tourifio, J.: Performance evaluation of big data frameworks for large-scale data analytics. In: 2016 IEEE International Conference on Big Data (Big Data), pp. 424–431 (2016). IEEE
12. Li, M., Tan, J., Wang, Y., Zhang, L., Salapura, V.: Sparkbench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM International Conference on Computing Frontiers, pp. 1–8 (2015)
13. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S.: Bigdatabench: A big data benchmark suite from internet services. In: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 488–499 (2014). IEEE
14. Marcu, O.-C., Costan, A., Antoniu, G., Pérez-Hernández, M.S.: Spark versus flink: Understanding performance in big data analytics frameworks. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 433–442 (2016). IEEE
15. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., *et al.*: Grid'5000: a large scale and highly reconfigurable experimental grid testbed. The International Journal of High Performance Computing Applications **20**(4), 481–494 (2006)
16. Mavridis, I., Karatza, E.: Log file analysis in cloud with apache hadoop and apache spark (2015)
17. Gopalani, S., Arora, R.: Comparing apache spark and map reduce with performance analysis using k-means. International journal of computer applications **113**(1) (2015)
18. Gu, L., Li, H.: Memory or time: Performance evaluation for iterative operation on hadoop and spark. In: 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, pp. 721–727 (2013). IEEE
19. Lin, X., Wang, P., Wu, B.: Log analysis in cloud computing environment with hadoop and spark. In: 2013 5th IEEE International Conference on Broadband Network & Multimedia Technology, pp. 273–276 (2013). IEEE
20. Petridis, P., Gounaris, A., Torres, J.: Spark parameter tuning via trial-and-error. In: INNS Conference on Big Data, pp. 226–237 (2016). Springer
21. Landset, S., Khoshgoftaar, T.M., Richter, A.N., Hasanin, T.: A survey of open source tools for machine learning with big data in the hadoop ecosystem. Journal of Big Data **2**(1), 24 (2015)
22. HiBench Benchmark Suite. https://github.com/intel-hadoop/HiBench
23. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10 (2010). IEEE

24. Luo, M., Yokota, H.: Comparing hadoop and fat-btree based access method for small file i/o applications. In: International Conference on Web-Age Information Management, pp. 182–193 (2010). Springer
25. Taylor, R.C.: An overview of the hadoop/mapreduce/hbase framework and its current applications in bioinformatics. In: BMC Bioinformatics, vol. 11, p. 1 (2010). Springer
26. Vohra, D.: Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools. Apress, California (2016)
27. Lee, K.-H., Lee, Y.-J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with mapreduce: a survey. AcM sIGMoD Record **40**(4), 11–20 (2012)
28. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster computing with working sets. HotCloud **10**, 95 (2010)
29. Kannan, P.: Beyond hadoop mapreduce apache tez and apache spark. San Jose State University. URL: http://www. sjsu. edu/people/robert. chun/courses/CS259Fall2013/s3/F. pdf (02.08. 2016) (2015)
30. Spark Core Programming. https://www.tutorialspoint.com/apache_spark/apache_spark_rdd.htm
31. HiBench Benchmark Suit. https://github.com/intel-hadoop/HiBench
32. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The hibench benchmark suite: Characterization of the mapreduce-based data analysis. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010), pp. 41–51 (2010). IEEE
33. Chen, C.-O., Zhuo, Y.-Q., Yeh, C.-C., Lin, C.-M., Liao, S.-W.: Machine learning-based configuration parameter tuning on hadoop system. In: 2015 IEEE International Congress on Big Data, pp. 386–392 (2015). IEEE
34. Ambari. https://ambari.apache.org/
35. Xiang, L.-H., Miao, L., Zhang, D.-F., Chen, F.-P.: Benefit of compression in hadoop: A case study of improving io performance on hadoop. In: Proceedings of the 6th International Asia Conference on Industrial Engineering and Management Innovation, pp. 879–890 (2016). Springer
36. O'Malley, O.: Terabyte sort on apache hadoop. Report, Yahoo! (2008). http://sortbenchmark.org/YahooHadoop.pdf
37. Apache Tuning Spark 1.1.1. https://spark.apache.org/docs/1.1.1/tuning.html
38. Spark Configuration. https://spark.apache.org/docs/latest/configuration.html
39. Rathore, M.M., Son, H., Ahmad, A., Paul, A., Jeon, G.: Real-time big data stream processing using gpu with spark over hadoop ecosystem. International Journal of Parallel Programming **46**(3), 630–646 (2018)

Figures

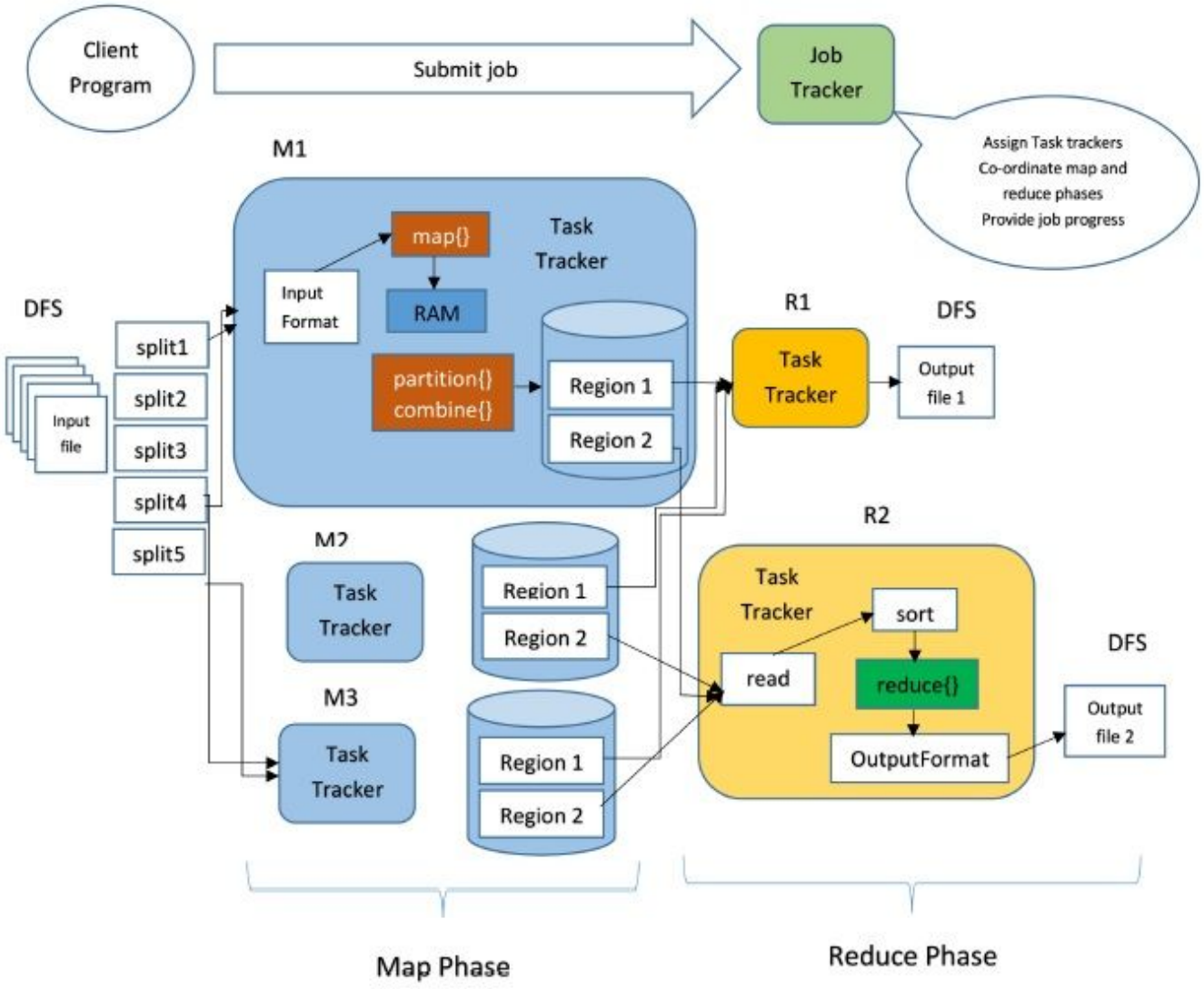


Figure 1

Hadoop MapReduce architecture [1]

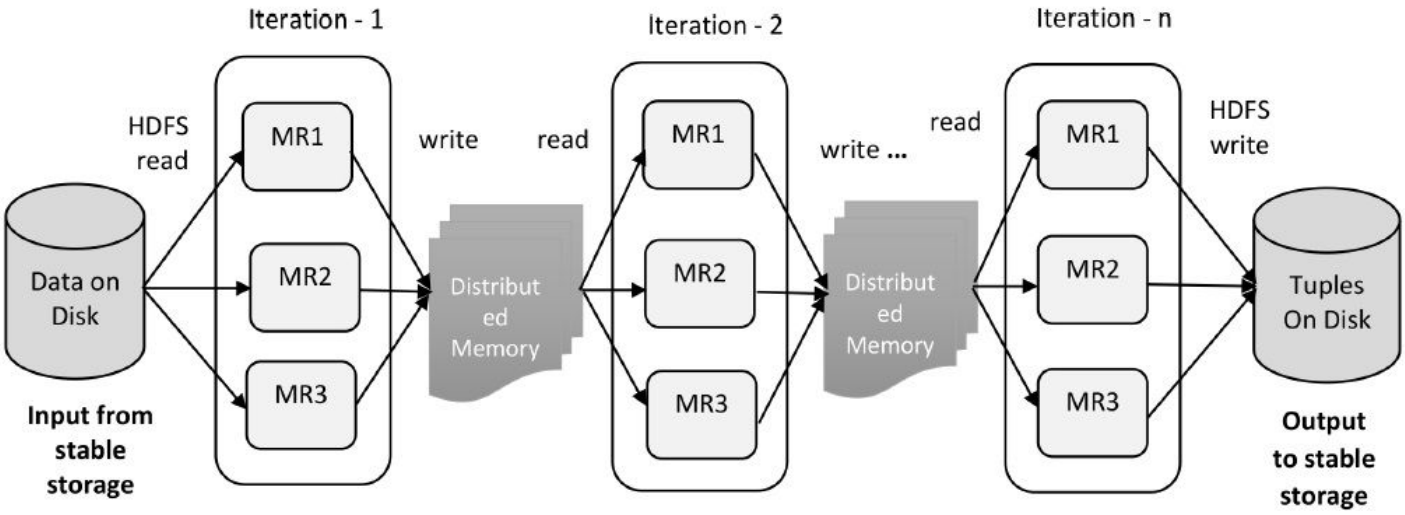


Figure 2

Spark workflow [30]

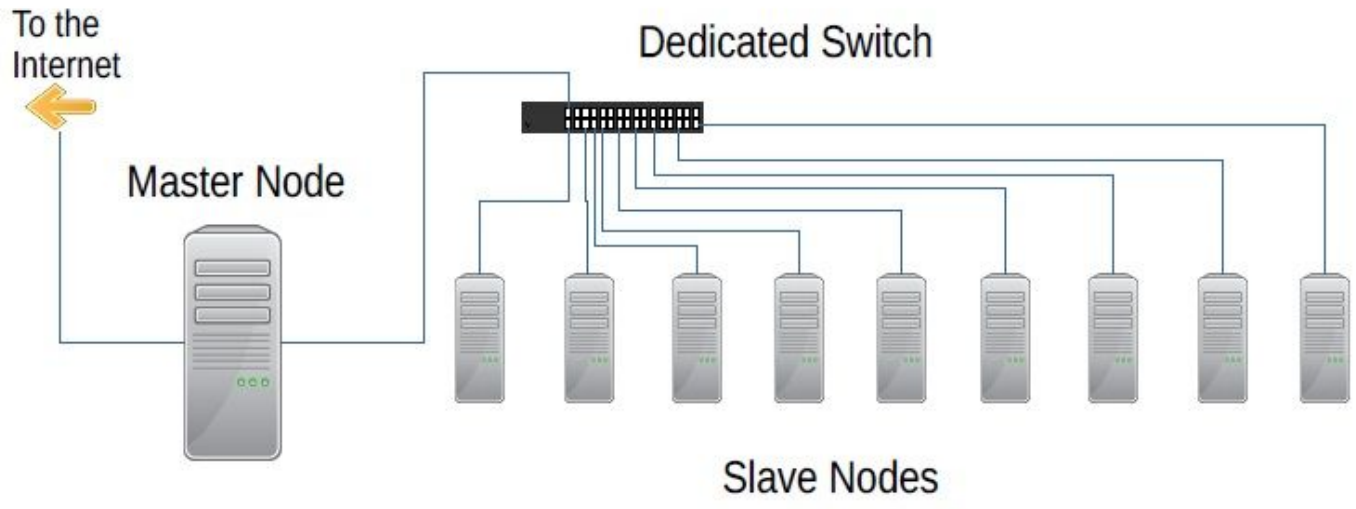


Figure 3

Hadoop Cluster Nodes

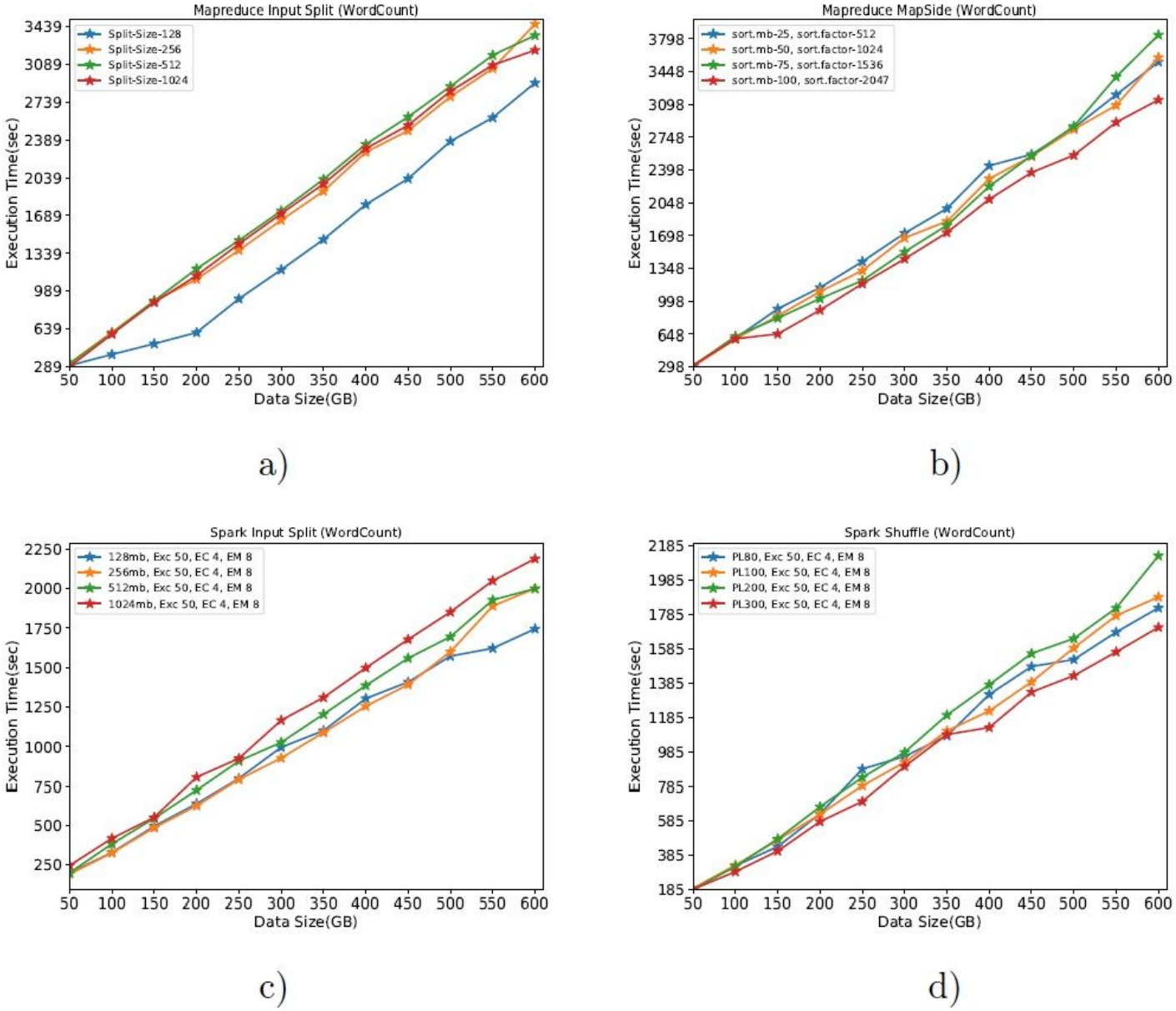


Figure 4

The performance of the WordCount application with a varied number of input splits and shuffle tasks.

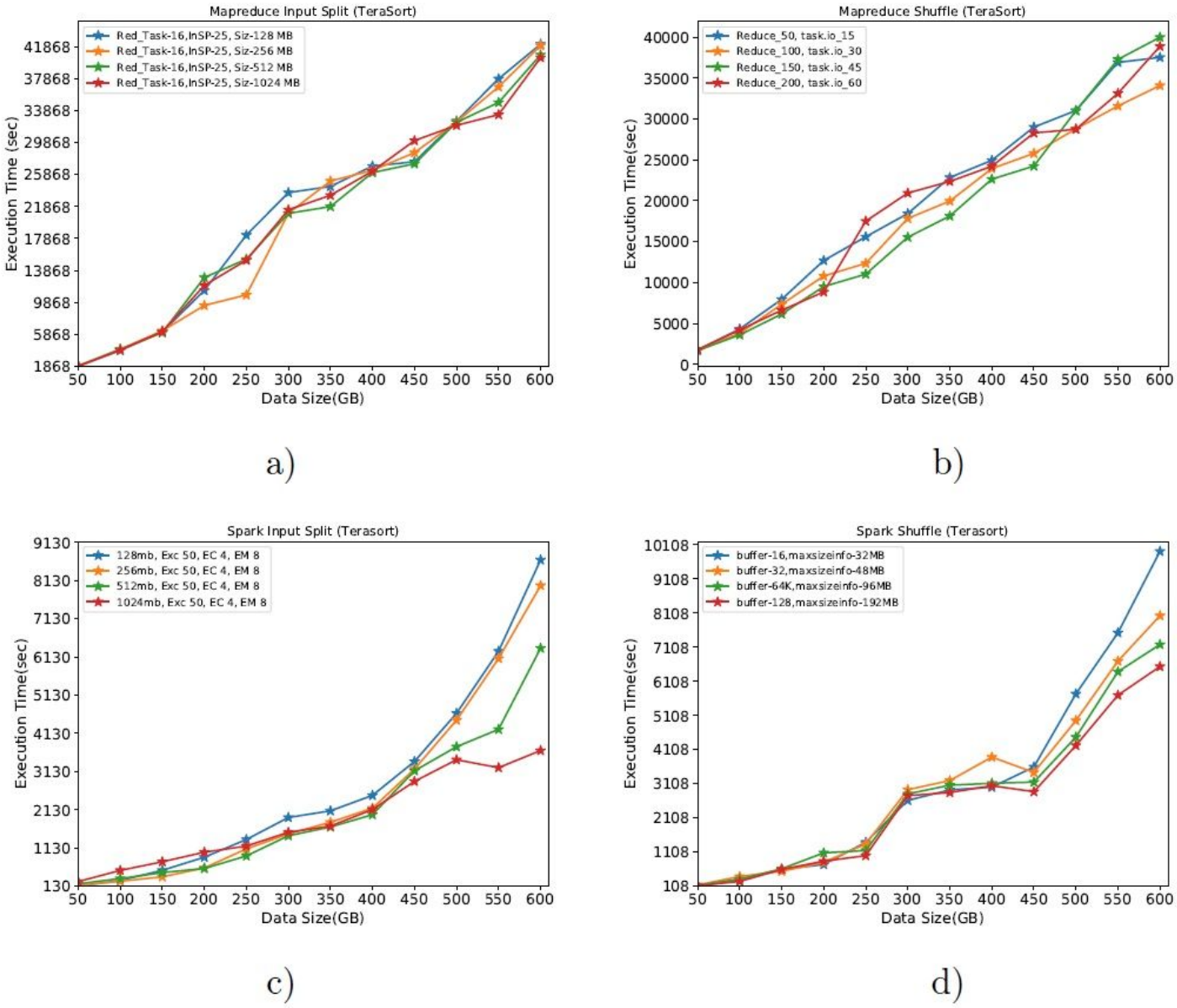


Figure 5

The performance of the TeraSort application with a varied number of input splits and shuffle tasks.

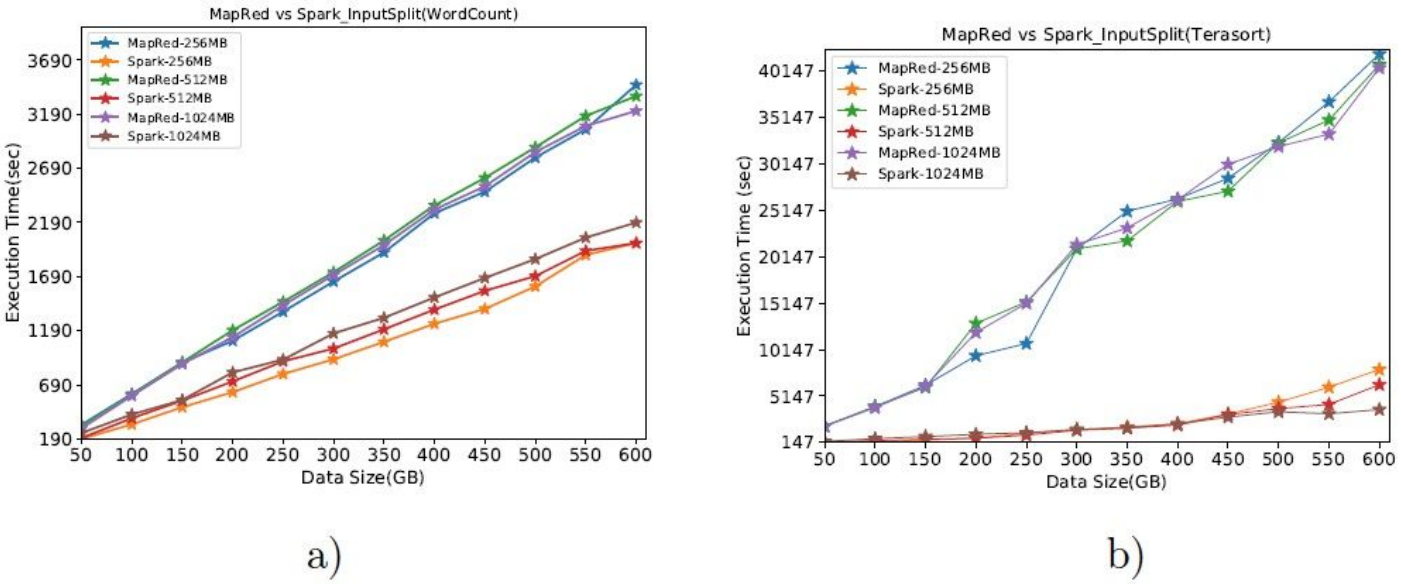


Figure 6

The comparison of Hadoop and Spark with WordCount and TeraSort workload with varied input splits and shuffle tasks.

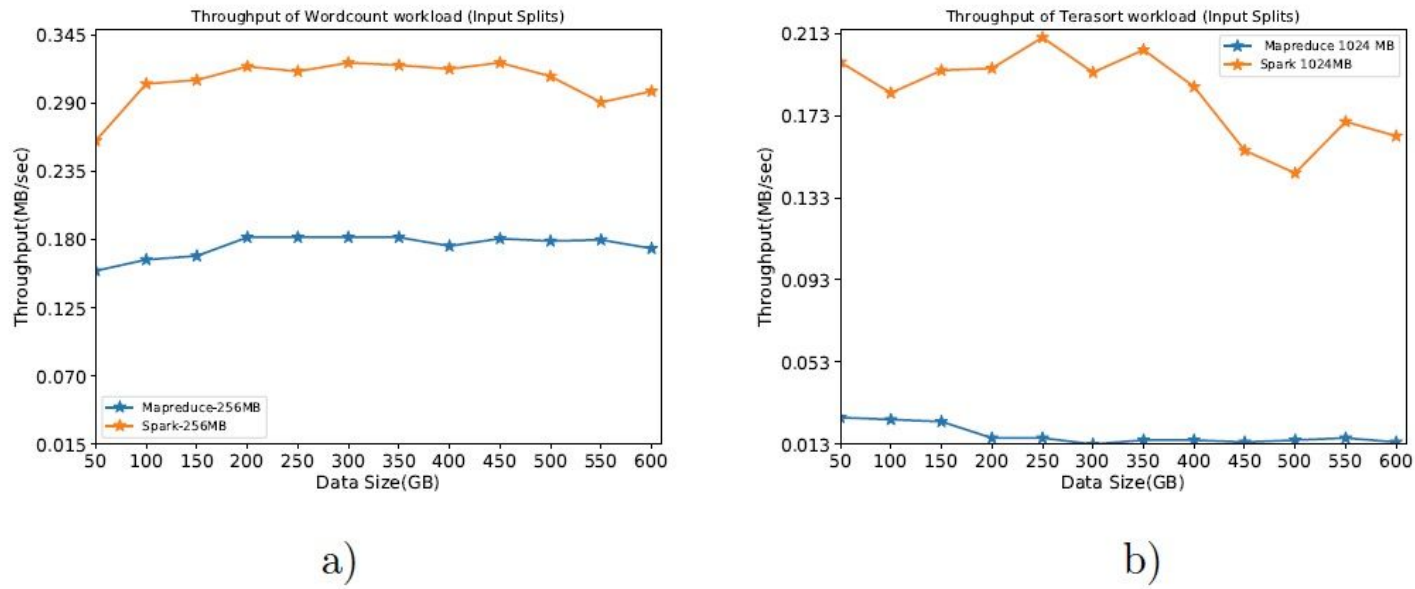


Figure 7

Throughput of WordCount and TeraSort workload.

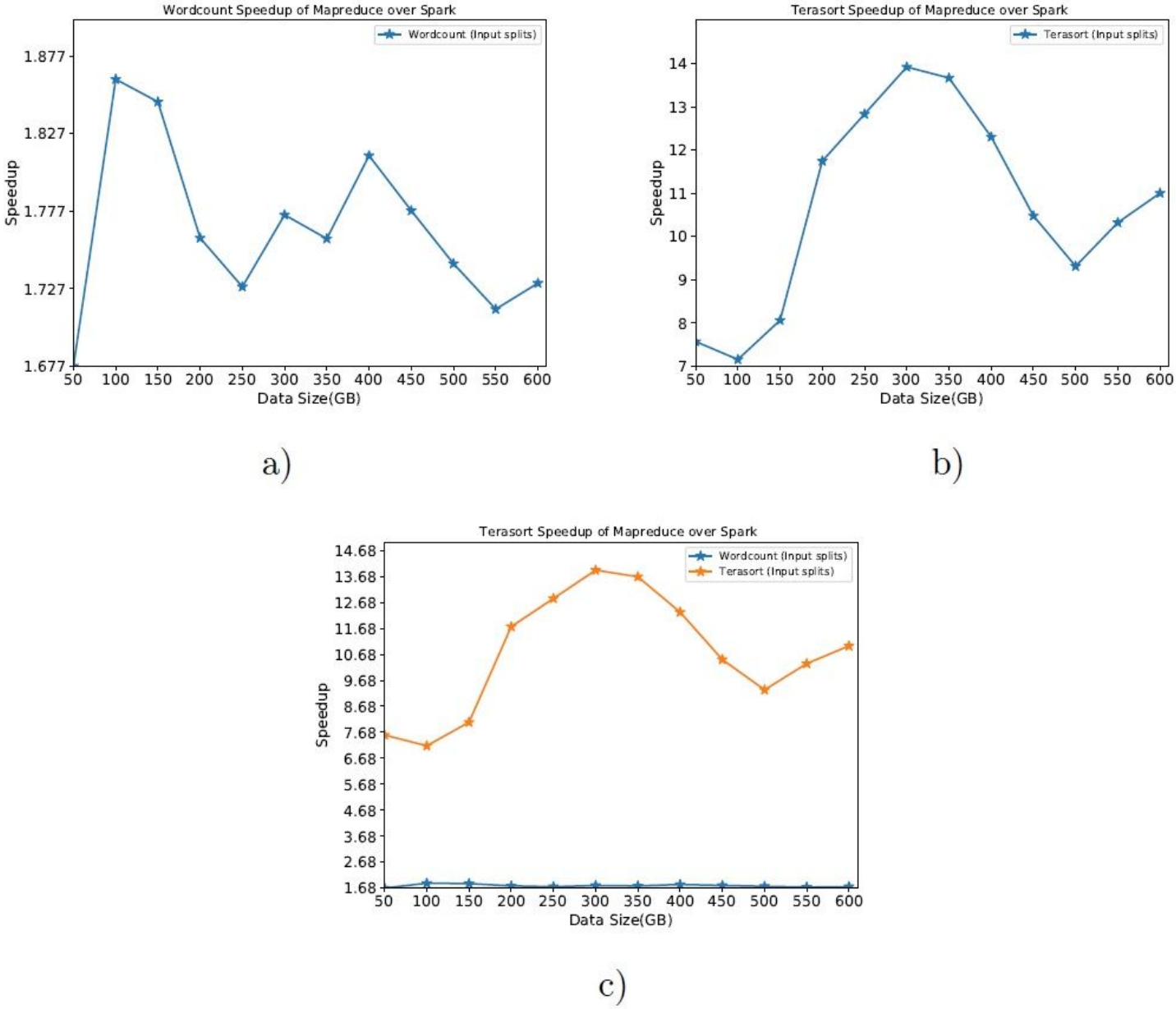


Figure 8

Spark over MapReduce speedup on input splits and shuffle.