
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2024 SCPE. Volume 25, Issues 3, pp. 1373–1386, DOI 10.12694/scpe.v25i3.2687

PERFORMANCE COMPARISON OF APACHE SPARK AND HADOOP FOR MACHINE LEARNING BASED ITERATIVE GBTR ON HIGGS AND COVID-19 DATASETS

PIYUSH SEWAL* AND HARI SINGH†

Abstract. In the realm of distributed computing frameworks, such as Apache Spark and MapReduce Hadoop, the efficacy of these frameworks varies across diverse applications and algorithms contingent upon distinctive evaluation metrics and critical parameters. This research paper diligently scrutinizes the extant body of research that compares these two frameworks concerning said evaluation metrics and parameters. Subsequently, it conducts empirical investigations to authenticate the performance of these frameworks in the context of an iterative Gradient Boosting Tree Regression (GBTR) algorithm. Remarkably, the comparative analyses in previous studies encompass a spectrum of iterative machine learning regression and classification techniques, batch processing, SQL, and Graph processing algorithms. Furthermore, numerous investigations have explored the application of machine learning algorithms encompassing logistic regression, Page Rank, K-Means, KNN, and the HiBench suite. This paper presents the comparison between the two distributed computing platforms on iterative GBTR for classification task on the HIGGS dataset from the physics domain and for the regression task on the Covid-19 dataset from the healthcare domain. The empirical findings corroborate that Apache Spark exhibits superior execution speed in iterative tasks when the available physical memory significantly exceeds the dataset size. Conversely, Hadoop outperforms Spark when dealing with substantial datasets or constrained physical memory resources.

Key words: MapReduce Hadoop, Spark, Machine Learning, Iterative, In-memory computation, Gradient Boost Tree Regression, Covid-19

1. Introduction. Today, a huge amount of data is being generated from different sources like social media platforms, IoT devices, sensors and digital devices. This data is popularly known as "Big Data" which is being generated in different forms like structured, semi-structured and unstructured [1]. As a result, the key challenge is not only to store this huge amount of data but also to process the data to gain useful insights and knowledge discovery. At present, there are several distributed data processing frameworks available such as Hadoop MapReduce, Spark, Flink, Storm, Samza etc.[2, 3].

Among these frameworks, Apache Spark and MapReduce Hadoop are two popular open-source frameworks that are widely used by different enterprises for processing large-scale data. Google's MapReduce aimed for scalability, security and fault tolerance for big data processing. The Apache Hadoop, an open-source implementation of the MapReduce model is a disk-based data processing framework suitable for batch processing jobs. However, it faced the issue of high disk access in each epoch which resulted in high I/O costs due to its inability to reuse intermediate results during the execution phase. Hence it resulted in low performance for iterative jobs. The performance of Apache Hadoop for spatial data is improved by indexing [4, 5]. The Apache Spark framework having the in-memory computational capability, overcame this limitation with a special type of data structure known as Resilient Distributed Datasets (RDDs) that supports reusability and is capable to store intermediate results in the physical memory of the system. A critical analysis of Hadoop and Spark [6], along with the high accuracy, scalability, and execution efficiency of distributed Spark MLib regression algorithms [7], as well as the performance prediction of Spark workloads using I/O parameters [8], covered in prior studies, sheds light on distributed processing frameworks.

In this study, firstly, a detailed literature survey is carried out that compares the two distributed computing frameworks – Apache Spark and MapReduce Hadoop on performance evaluation metrics and key parameters. Secondly, the two frameworks are compared through experimental work on the iterative Gradient Boost Tree Regression algorithm on metric execution time, the effect of memory size and varying dataset size on execu-

*CSE & IT Department, Jaypee University of Information Technology, Solan, HP, India. (piyush.sewal@gmail.com)
†CSE & IT Department, Jaypee University of Information Technology, Solan, HP, India. (hsrawat2016@gmail.com)

tion time. The experimental work uses the HIGGS dataset [9] and the Covid-19 datasets [10] on clusters of varying sizes.

The rest of the paper is as follows. The related work is presented in Section 2. Section 3 compares experimental results for the iterative Gradient Boost Tree Regression algorithm under different cluster configurations and datasets. Finally, Section 4 concludes the paper.

2. Related Work. This section presents a detailed literature review on the distributed computing frameworks – Apache Spark and MapReduce Hadoop. The review is carried out in two different sets of parameters in sub-section 2.1 and 2.2.

2.1. Review of evaluation metrics. This sub-section presents the review of evaluation metrics execution speed, memory usage, cluster size, data characteristics, CPU utilization and network usage. It is presented in Table 2.1 along with its description in the text. The issues of latency, I/O and deserialization cost are analyzed in a distributed memory structure Resilient Distributed Datasets (RDDs) in the Spark that uses in-memory computations. Then Spark and Hadoop are compared on Logistic Regression, K-means and Page Rank algorithms. The experimental results validate the high execution speed of Spark than Hadoop for iterative and graph-based applications. It is also observed that Spark recovers the lost RDD partitions very quickly in case of node failures. However, the performance of the Spark degrades more than the Hadoop when memory is not sufficient [11]. Similar results were obtained when researchers compared the Hadoop and Spark frameworks using the Page Rank algorithm [12].

The running time of the K-Means algorithm of the HiBench benchmark on Hadoop and Spark clusters with different sizes of memories allocated to data nodes shows that Spark performs better as long as the memory size is sufficient enough for the data size [13]. Another similar work evaluated the two frameworks on execution speed for the K-Means algorithm using sensor datasets of varying size on different cluster sizes and obtained similar results [14]. In another paper, the K-Means algorithm is applied to the satellite images dataset with modified values of K in three phases which include the Initialization Phase, Clustering Phase and Validation Phase [15]. The experimental results show that the speed-up performance and scalability of the algorithm is improved on both the Spark and MapReduce clusters. In another research work, the K-Means algorithm and Page-Rank algorithm show that the Spark performance improves for the iterative algorithm of data reuse [16]. The Spark is about 40 times faster than Hadoop for a data quantity of about 40 thousand points. However, the Spark performance declines and then saturates but it remains 8 times fast as compared to Hadoop with an increase in data quantity. The results also show that Spark has a significant performance for iterative jobs with a low latency schedule when the size of the dataset exceeds the memory size. In another work, the authors compare the two frameworks on execution time, CPU utilization, memory and network usage for the K-Nearest Neighbor (KNN) algorithm on different size datasets and cluster configurations. The Spark is observed to have better execution speed and CPU utilization but memory size is the bottleneck. The Hadoop is observed to consume more network resources but the memory size does not create any performance problems [17].

The Spark is reported to perform well on the HiBench benchmark suit when different datasets are used [18]. The authors classified thirteen workloads benchmark suites into four categories Micro Benchmarks, Web Search, SQL and Machine Learning. Among these, eight benchmarks Aggregation, Bayesian, Join, Pagerank, Scan, Sleep, Sort and Terasort are taken into consideration for measuring the performance. Then the performance is evaluated using three metrics execution time, throughput and speed-up. It is observed that the execution time is very less in Spark as compared to Hadoop with a factor of 18. The reason behind the more execution time in Hadoop is due to multiple object creation for single input, slow data sharing due to replication and serialization and disk-based I/O. Similarly, the throughput and speedup are also better for the Spark cluster than the Hadoop cluster.

The wordcount program is used to compare the two frameworks for different size datasets [19]. The experimental results show that Hadoop takes more time as compared to Spark irrespective of the size of the dataset. The Spark counts the occurrence of each word in less time due to its in-memory computational capabilities.

In a similar work, the wordcount program on four different datasets validates the better performance of Spark over Hadoop on execution time [23]. In another work, the wordcount program is implemented on a publically available word file and logistic regression is applied to a dataset of bankruptcy conditions of companies

[20]. The experimental results show that Spark performs better than MapReduce for both normal and iterative queries with a performance ratio of 2.8 and 2.2 for wordcount and logistic regression respectively. The study is performed on a single machine and does not consider varying cluster sizes. Another work on the same algorithm with a cluster of fixed size validated the high performance of Spark for iterative and streaming data processing whereas Hadoop is found suitable for batch data processing [24].

In another research work, the authors compared Hadoop and Spark on streaming data for comparing the execution time. The Spark engine is used to process twitter's tweets in a short interval of less than a second. The results indicate that Spark is better than Hadoop for streaming data processing due to its in-memory processing, lower disk-access rate and event-driven task scheduling [22]. In another work, Spark outperformed MapReduce for processing streams of text and video on GPU for execution time and throughput [25]. The video data is captured from Youtube with different road and traffic scenarios. The text data is collected from sensors and social networks by using Apache Spark Streaming. For video-related big data, the processing time of large videos increased considerably in MapReduce Hadoop in comparison to Spark.

In another work, the authors used the Apriori algorithm in three different execution approaches IMRAprioriAcc (Improved MapReduce Apriori Accelerated), DPC (Dynamic Passes Combined-Counting) and CPA (Complete Parallel Apriori) along with their adaption on Spark with different size datasets and varying cluster configuration [21]. Four performance metrics runtime, speed-up, size-up and scale-up are used for the performance evaluation of the Hadoop MapReduce and Spark. The experiment results of the work validate the better performance of Spark over Hadoop MapReduce. The implementation of CPA with MapReduce gives better results than Spark when the size of the dataset is large and physical memory is not sufficient. In recent works, the authors compared various machine learning regression algorithms [26] and Hadoop and Spark for execution time and throughput using different size Covid-19 datasets on a fixed-size cluster [27]. In the latter, the experiment results validate the high execution speed of Spark for small datasets.

2.2. Comparison of key parameters. This sub-section presents the comparison of key parameters data processing, performance, latency, fault-tolerance, scalability, security, cost, scheduling, resource management, inbuilt capabilities, usability and language support. It is presented in Table 2.2 along with its description in the text. The Hadoop is best suited for batch data processing as it uses MapReduce which splits large datasets among various clusters and processes these in parallel [28]. In the MapReduce architecture, data passes in four phases which are splitting, mapping, shuffling and reducing. On the other hand, Spark is suitable for iterative and live streaming data which is mostly in the unstructured form. It uses the concept of in-memory processing of data and works with the help of RDDs to perform various operations [29, 30]. Spark creates DAG (Directed Acyclic Graph) that contains vertices and edges where vertices represent RDDs and edges represent the operations to be performed on the RDDs [11].

The MapReduce Hadoop is not efficient for iterative operations because it cannot keep reused data and state information during execution [17]. It is a high-latency framework in which integrative mode is not available. Thus it persists intermediate data onto the disk that further results in slow data processing. However, Spark is a low-latency computing framework and it can process data interactively. It results in faster processing of data as it reads the disk only once and then all the intermediate operations are performed within the RAM [2, 11].

The MapReduce Hadoop accesses disk for data storage and processing which comparatively results in slower processing whereas the Spark performs in-memory data processing that results in faster processing of data [12, 17, 31, 32, 33, 34]. The processing capabilities of the Spark are affected significantly by the size of available memory in comparison to the Hadoop [13]. The memory utilization is proven better in Hadoop whereas CPU utilization is proven better in Spark [17].

The Hadoop uses the concept of 3X replication and erasure coding for backing up data in case of any node failure. 3X replication generates 200% overhead in storage space as compared to just 50% in the case of erasure coding. On the other side, Spark uses DAG to rebuild the data using RDD across the nodes which also avoids storage space overhead. Hence both platforms have good fault tolerance mechanisms [11, 35].

Scalability is another important parameter in big data processing frameworks. Nodes and disks can be added easily on the fly and the latest versions of Hadoop are capable to add more than ten thousand nodes at a time. On the other hand, scalability is a bit challenging in Spark because it depends upon the computational capabilities of machines which may be different. However, Spark also supports thousands of nodes in a cluster[33, 36, 37].

Table 2.1: Performance comparison of Hadoop and Spark on various applications/algorithms

Application	**System Configuration**				**Dataset Size**	**Dataset Type**	**Evaluation Parameters**						**Better Performer**
	RAM (GB)	**Disk (GB)**	**Disk Type**	**Cluster Nodes**			**A**	**B**	**C**	**D**	**E**	**F**	
K-Means, Page Rank [16]	4	NA	NA	4	10K to 20M points	Log data	✓	x	✓	x	x	x	Spark
Page Rank, Logistic Regression, K -Means [11]	15	NA	NA	4	54GB	Wikipedia dump	✓	✓	✓	x	x	x	Spark
Page Rank [12]	4	NA	NA	8	NA	Graph data	✓	x	✓	✓	x	x	Spark
K-Means [14]	4	500	HDD	2	64 MB, 1240 MB	Sensor data	✓	✓	✓	x	x	x	Spark
Multiple K-Means++ [15]	8	NA	NA	5	1GB to 4GB	Satelite images data	✓	x	✓	x	x	x	Spark & Hadoop
Wordcount, Logistic Regression [20]	8	1000	HDD	1	NA	Text and numeric data	✓	x	✓	x	x	x	Spark
Hibench suit (8 benchmarks) [18]	4	40	SSD	1	Different for each benchmark	NA	✓	x	✓	✓	✓	x	Spark
K-Means [13]	16	500	HDD	3	1GB to 8GB	NA	✓	✓	✓	✓	x	x	Spark & Hadoop
Apriori [21]	8	500	HDD	20	NA	Synthetic data	✓	✓	✓	x	x	x	Spark & Hadoop
Flume, Spark streaming [22]	5	40	NA	1	NA	Twitter data	x	x	✓	x	x	x	Spark
Wordcount [19]	4	1000	HDD	4	1 MB to 300 MB	Text data	✓	x	✓	x	x	x	Spark
Wordcount, Logistic Regression, K-Means [19]	NA	NA	NA	1	500 MB to 40 GB	Wikipedia and Enron	✓	x	✓	x	x	x	Spark
K-NN [17]	4	NA	NA	6	8 GB	CSV data	✓	✓	✓	✓	✓	✓	Spark& Hadoop
Wordcount [23]	2	NA	NA	3	34 MB to 202 MB	Text data	✓	x	✓	x	x	x	Spark

(Abbreviations used: A: Varing dataset size, B: Varing cluster size, C: Execution time, D: Memory Usage, E: System Throughput, F: Networking, GB: GigaByte, MB: MegaByte RAM: Random Access Memory, HDD: Hard Disk Drive, SSD: Solid State Drive, NA: Not Available.)

Security is always a major concern while processing large datasets and Hadoop address this issue very effectively. The Hadoop supports ACLs, SLAs, LDAP and Kerberos which makes it extremely secure. The Spark does not provide such level of security and its security is turned off by default. However, Spark provides authentication with the help of event logging or shared secrets which is not sufficient. Thus, Spark integrates with Hadoop to achieve a significant security level [38].

Along with some key performance parameters, cost also plays an important role in the big data processing. Although both Hadoop and Spark are open-source platforms, when hardware resources are considered, Hadoop is less expensive as it relies on disks for storage and processing whereas Spark performs in-memory processing which slightly increases the data processing cost [20, 33, 36].

Table 2.2: Comparison between Hadoop and Spark on key parameters

Parameters	**Hadoop**	**Spark**
Data Processing [6, 11, 28, 29, 30]	Suitable for batch processing	Best for iterative and streaming data processing
Latency [2, 11]	High	Low
Performance [11, 12, 13, 17, 21, 24, 34]	Comparatively slower	Comparatively faster
Fault Tolerance [6, 11, 35]	Supported	Supported
Scalability [33, 36, 37]	Easily scalable	Scalability is a bit challenging
Security [33, 36, 38]	Extremely secure	Comparatively less secure
Cost [20, 33, 36]	Less expansive	A bit more expensive
Scheduling and Resource Management [33, 36, 39]	Use external solutions	Built-in solutions available
In-built capabilities [6, 22, 24, 40, 41, 42]	HDFS, YARN, MapReduce	Spark Core, Spark SQL, Spark Streaming, SparkML, GraphX
Usability and Language Support [33, 36]	A bit difficult	User friendly

For scheduling and resource management, Hadoop uses external solutions which include ResourceManager, NodeManager and YARN for resource management, CapacityScheduler and FairScheduler for resource allocation, and workflow scheduling is done by Oozie [39]. On the other hand, Spark has in-built support for job scheduling, resource management and monitoring. Spark uses DAG Scheduler for dividing the operations into stages and each stage consists of various tasks that need to be done by the Spark computation engine. Both Hadoop and Spark are equipped with some in-built components. In Hadoop, in-built components include HDFS which is used as a file system, YARN for resource management and MapReduce as the processing engine. On the other side, Spark has Spark core as the processing engine, Spark streaming for near real-time data processing, GraphX for graph processing and Spark SQL for structured data processing. Both Hadoop and Spark support machine learning libraries [24]. Hadoop uses the external library Mahout for machine learning whereas Spark has a in-built library MLlib. The experimental results show that due to the in-memory processing of Spark, MLlib is much faster than Apache Mahout but when data is extremely large MLlib sometimes crashes because of insufficient memory whereas Mahout process the data continuously even with a slow speed [40, 41, 42].

Lastly, if we compare the usability and language support of both platforms, Hadoop was developed in Java language whereas Apache Spark was developed in Scala language. If we consider the usability of both data processing platforms, Hadoop has limited language support and it uses Java and Python languages for MapReduce applications. Spark on the other hand is more user-friendly and allows interactive shell mode. Spark has large language support and its APIs can be written in Scala, Python, R, Java and Spark SQL [33, 36].

3. Performance Comparison of the Apache Spark and MapReduce Hadoop on the iterative GBTR. This section covers the performance comparison of Hadoop and Spark frameworks on the iterative GBTR. The performance of both frameworks has been analyzed on the benchmark dataset HIGGS [9] and two real-life Covid-19 [10] datasets. This section first briefs the GBTR algorithm and then presents experimental results and discussions.

3.1. Gradient Boost Tree Regression Background. The GBTR improves the mistakes of the previous learner with the help of the next learner. This algorithm is similar to the AdaBoost algorithm (adaptive boosting) and uses an ensemble tree for predicting the target variables. However, the depth of the tree is more than one here. During the implementation of the algorithm, firstly a base model is initialised with constant values and an average value of the target variable is calculated using the following equation:

$$F_0(x) = arg_\gamma min \sum_{i=1}^{n} L(y_i, \gamma) \tag{3.1}$$

Here L denotes the loss function, y_i is the observed value, γ is the predicted value and arg_γmin is the predicted value for which the value of the loss function is minimum. The Loss function L for the target variable can be

Algorithm 1: Gradient Boost Tree Regression for predicting the target variable

Data: Dataset file (in CSV format)
Result: Final Prediction concerning target variable
Description: This algorithm is used to predict the value of target variable using the Gradient Boost Tree Regression algorithm.
Step 1. Choose the input dataset (HIGGS/Covid-19) and select the IndependentFeatures and DepenedentFeatures as input values;
Step 2. Intialize a base model with IndependentFeatures and DependentFeatures;
$TargetVariable \leftarrow DependentVariable$;
and compute y using equation 3.1 where y is the average value of TargetVariable;
Step 3. while $Residuals = Null$ **do** Compute Residuals using equation 3.1 and 3.2 ;
$Residuals \leftarrow ActualOutputValue - PredictedOutputValue$;
end while
Step 4. Initialize an empty ensemble list: Ensemble = [] ;
Initialize i = 1 ;
while $i \leq N$ **do** /* Where, N is the number specified by hyperparameter tuning */ Construct $DecisionTree_i$;
$PredictTargetValue_i$ using $DecisionTree_i$ within the ensemble;
Compute New $Residual_i$ using equation 3.3 ;
$AddDecisionTree_i$ to the ensemble ;
$i \leftarrow i + 1$
end while
Step 5. Use all $DecisionTree_i$ within the ensemble for final prediction as to the value of TargetVariable;

calculated as:

$$L = \frac{1}{n} \sum_{i=1}^{n} L(y_i - \gamma_i)^2 \tag{3.2}$$

Here residuals are calculated by taking the difference between the actual and predicted values. The predicted value is the average value which is calculated in the first step of the base model. After this, a decision tree is constructed and new values of the target variable are predicted. Mathematically, pseudo residuals can be calculated using the following equation:

$$\gamma_i m = \left[\frac{L(y, F(x_i))}{F(x_i)} \right] \forall i \in \{1 to N\}, WhereF(x) = F_{m-1}(x) \tag{3.3}$$

Here F(x) is used to calculate the value of the updated model by using the previous model $F_{m-1}(x)$. To prevent low bias and high variance, a learning rate variable (between 0 to 1) is multiplied with newly calculated residuals which is important to use for improving the accuracy of the model in long run. With each successive step, new residuals are calculated again and steps are repeated till the value matches the value of the hyperparameter. In the end, all the decision trees are used within the ensemble and the final prediction is calculated concerning the target variable. An algorithm to demonstrate the working of GBTR has been presented in Algorithm 1.

3.2. Results and Analysis on the Benchmark Dataset - HIGGS. In the first level of the execution stage, the Spark and Hadoop frameworks are tested on the HIGGS dataset [9]. The dataset has been produced using Monte Carlo simulations and contains 11 million samples with 28 features for each. The first 21 features are the kinematic properties and the last 7 features are the functions of the first 21 features. In our execution environment, we have used five nodes cluster where each node has 4 GB RAM, 512 GB HDD, Intel Core i5

Table 3.1: Details of HIGGS sample datasets and cluster configuration

HIGGS Dataset Samples	Sample Size (MB)	Number of Records	Number of Features	Cluster Size
H1	20	30000	28	5
H2	40	60000	28	5
H3	80	120000	28	5
H4	160	240000	28	5
H5	320	480000	28	5
H6	6400	960000	28	5
H7	1000	1375000	28	5
H8	2000	2750000	28	5
H9	4000	5500000	28	5
H10	8000	1100000	28	5

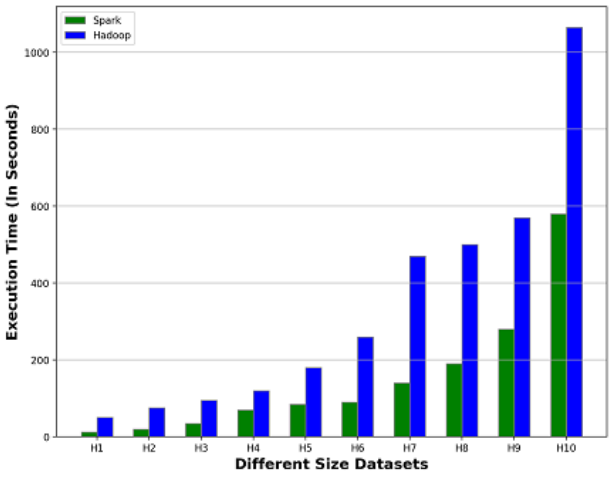

Fig. 3.1: Comparison of the Execution time of GBTR algorithm on varying size datasets on a fixed cluster size=5

processor, Hadoop 3.2 and Spark 3.2. All the systems are connected with 100Mbps local area network. The performance is evaluated using two different scenarios which are covered in the next sections.

3.2.1. Varying size datasets on a fixed cluster size. Ten different samples of HIGGS datasets (H1 to H10) with sizes from 20 MB to 8000 MB are used for the execution. The GBTR is used for execution on Hadoop and Spark clusters on a cluster size of five. The details are presented in Table 3.1.

It is clear from FIG. 3.1 that in the case of small size datasets, the dominance of Spark over Hadoop is nearly 4 to 5 times but as the size of the dataset keeps increasing, this dominance starts reducing. In the case of samples H9 and H10, it is observed that the Spark is 1.5x to 2x faster than the Hadoop. So, the Hadoop is not suitable for small size datasets but when the dataset size is large enough then the Hadoop performs well. On the other hand, Spark is good for small size datasets but if the size of the dataset is large then the Spark either need sufficient physical memory or its performance will start degrading. So the size of the dataset plays an important role in the performance evaluation of Hadoop and Spark.

3.2.2. Fixed size dataset on varying cluster size. In the second scenario, a sample HIGSS dataset of 320 MB is used for experimentation on five different cluster configurations C1, C2, C3, C4 and C5 having one,

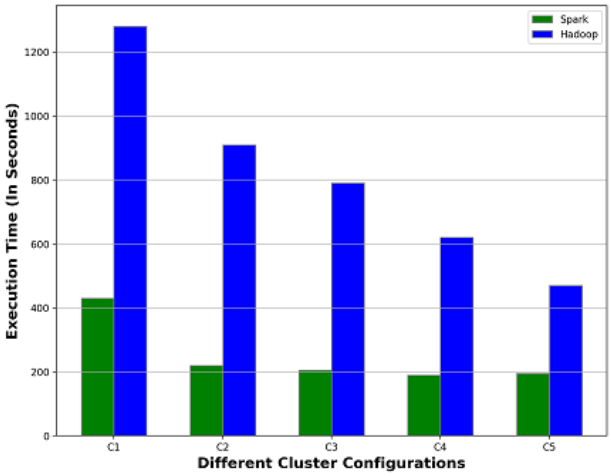

Fig. 3.2: Comparison of the Execution time of the GBTR algorithm on different cluster configurations with the same size dataset

two, three, four and five machines respectively. It observed the impact of varying system configurations on the execution time of the GBTR algorithm with the same dataset. It is clear from FIG. 3.2 that the execution time of both Hadoop and Spark is very high in C1.

Although, with the increase in system configuration, additional resources are available for execution that results in a decrease in execution time for parallel tasks which can be validated from the execution statistics of configuration C2. The execution time of the Spark does not vary in the case of C3, C4 and C5. This is because Spark got sufficient resources for execution and after a certain limit of computational resources there is no effect on execution time. The execution time of the Spark in C5 is slightly more than in C4. On the other hand, the execution time of Hadoop gradually decreases from C1 to C5. It means that the allocation of additional resources in Hadoop is helping in reducing the execution time. So it can be concluded from this experiment that granting the additional resources in distributed processing framework can be useful in reducing execution time but after the saturation point, the execution time will remain constant or it may start increasing and that saturation point of cluster configuration is directed related with dataset size and execution algorithm.

However, this fact cannot be neglected that the execution time of Spark is still less than Hadoop. The main reason behind this is the capability of Spark to perform in-memory computations with the help of RDDs. The intermediate operations performed on RDDs can be visualized through a graph which is known as Directed Acyclic Graph (DAG). The DAG is basically a set of vertices and edges where vertices represent the RDDs and edges represent the operations performed on RDDs. The Spark RDDs splits into stages by job scheduler on the basis of various transformations. During the execution phase of Spark framework, the GBTR application is divided into 203 stages. Each stage performs the transformations operation on intermediate RDDs and finally performs the action operation in the last stage. Stages 1 to 6 perform distinct operations on RDDs and then Stage 7 to 200 performs the same operations as Stage 5 and 6 but on different intermediate RDDs. Then all the intermediate orations are consolidated in Stage 201, Stage 202 shuffles the final results and Stage 203 shows the final results of GBTR on the console. An overview of Spark stages for GBTR application in the form of DAG has been presented in FIG. 3.3.

3.3. Results and Analysis on the Real-life Covid-19 Datasets. The performance evaluation of Hadoop and Spark clusters was conducted using the iterative GBTR algorithm on two datasets, Dataset-1 and Dataset-2, of varying sizes from the Covid-19 datasets of India and the world available on Kaggle [10]. After data pre-processing phase, data cleaning and data transformation, the experiment is performed on one, two and five machines equipped with 64-bit Windows OS, 4 GB RAM, 512 GB HDD, Apache Spark, Hadoop, Mahout, Python, Java, Eclipse and Anaconda Navigator respectively. The same system configuration is used for Hadoop

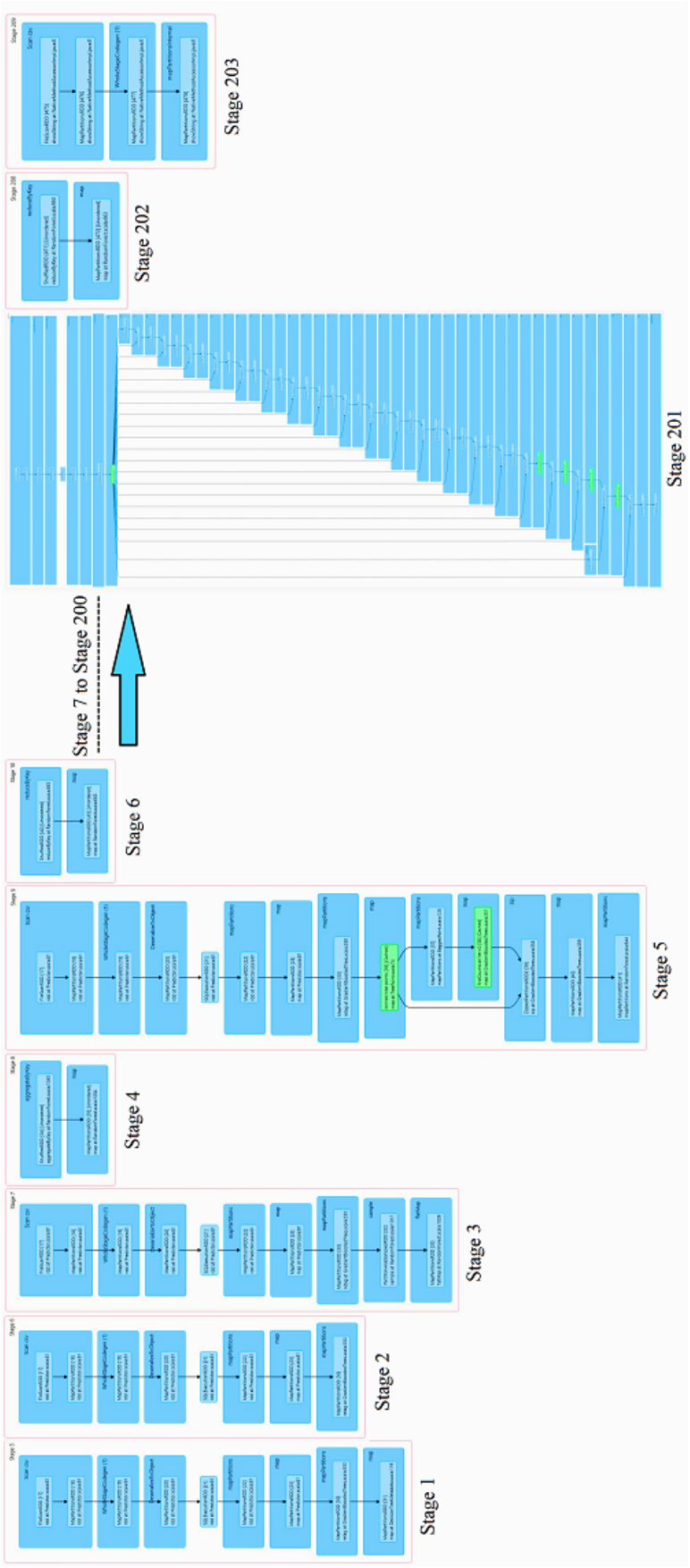

Fig. 3.3: DAG visualization of GBTR Algorithm at stage level during execution phase

Table 3.2: Cluster Specification of Hadoop and Spark

Title	**Hadoop Cluster**			**Spark Cluster**		
ClusterSize (Nodes)	1	2	5	1	2	5
Master Node	1	1	1	1	1	1
Worker Nodes	1	1	4	1	1	4
Number of CPU Cores	4	8	20	4	8	20
Total Internal Memory (GB)	4	8	20	4	8	20
Total Secondary Memory (GB)	512	1024	2560	512	1024	2560
Secondary Memory Type	HDD			HDD		
Processing Framework & Version	Hadoop 3.2			Spark 3.2		
Machine Learning Library	Mahout			Spark MLib		
Scala Version	2.12.15			2.12.15		
Java Version	11.0.13			11.0.13		
IDE	Eclipse			Jupyter Notebook		
API	Java			Python		
OS	Windows 10			Windows 10		
Processor	Intel Core i5-6500 CPU @ 3.20 GHz					

Table 3.3: Comparison of Hadoop and Spark for GBTR Algorithm on different size datasets and cluster configurations

Dataset	**Execution Case**	**No. of Records**	**Cluster Size**	**Execution Time (sec)**	
				Hadoop (Mahout)	**Spark (MLib)**
Dataset 1	Case I	560	1 Node	541	220
			2 Node	317	18.6
			5 Node	206	17.2
	Case II	18110	1 Node	754	311
			2 Node	429	22.4
			5 Node	293	17.8
Dataset 2	Case III	494	1 Node	821	371
			2 Node	472	21.9
			5 Node	325	18.8
	Case IV	306429	1 Node	1019	489
			2 Node	589	24.2
			5 Node	382	21.7

and Spark as given in Table 3.2.

In Case-I, 18,110 records from dataset-1 were grouped based on the number of days, ranging from day 1 to day 560. Consequently, day-wise data was consolidated, reducing the record count to 560, and the algorithm was executed under three distinct cluster configurations. Case-II involved the use of dataset-1 in its entirety, encompassing all 18,110 records. In Case-III, a similar day-wise grouping approach as in Case-I was followed, but this time with 306,429 records from dataset-2, resulting in a dataset of 494 records for algorithm execution. Finally, in Case-IV, the complete dataset-2, consisting of all 306,429 records, was employed.

Findings, illustrated in Table 3.3, highlight that across all four cases, the execution time for both Hadoop and Spark is notably high when executed on a single node. However, statistical analysis clearly reveals that Hadoop exhibits higher execution times than Spark, as indicated in FIG. 3.4. This disparity arises because

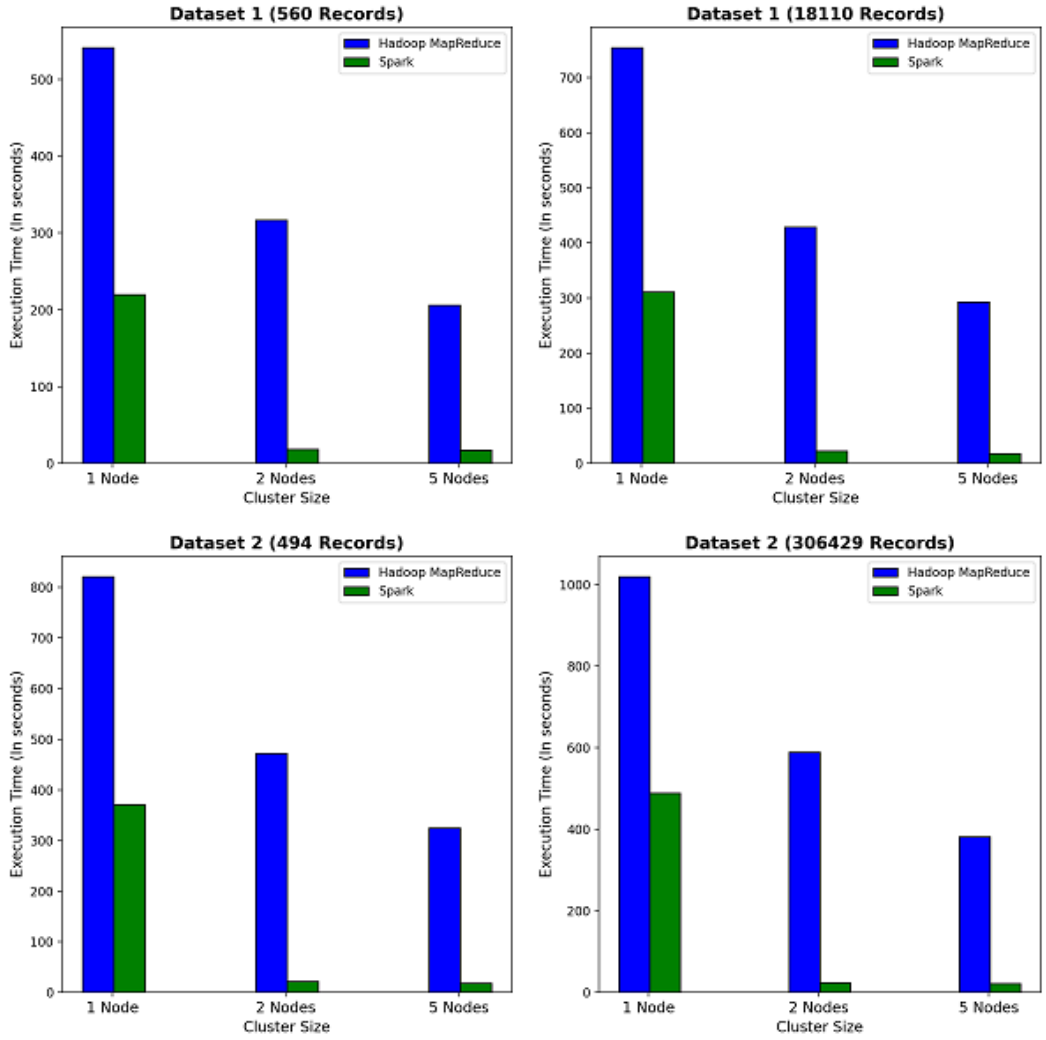

Fig. 3.4: Comparison of the execution time of Hadoop and Spark for Gradient Boosting Algorithm under different cluster configurations

Hadoop accesses the disk multiple times during iterative tasks, leading to increased latency. In contrast, Spark leverages in-memory data processing, creating RDDs and executing transformations and actions without repeated disk reads. An important observation is that the addition of nodes to the cluster, from one to two and then five, results in a sharp reduction in Spark's execution time. Conversely, Hadoop MapReduce exhibits a gradual decrease in execution time under similar conditions, as depicted in FIG. 3.4. This phenomenon is attributed to the increased number of nodes, which also augments available physical memory, contributing to Spark's enhanced processing speed. It's noteworthy that the execution time remains consistent for Spark clusters with 2 nodes and 5 nodes, indicating that the resources of the two-node cluster suffice for processing the dataset, and additional resources do not impact execution time. Conversely, while disk space increases for Hadoop, it has limited utility in expediting data processing tasks, particularly with small datasets. However, in the context of batch processing of larger datasets, Hadoop surpasses Spark when physical memory resources are constrained.

4. Conclusions and Future Work. In summary, this paper presents an extensive comparative analysis of evaluation metrics and critical parameters between Apache Spark and MapReduce Hadoop across a diverse array of algorithms, including K-Means, Page Rank, Word Count, Logistic Regression, Apriori, and the HiBench Suite. Our analysis reveals that both Hadoop and Spark exhibit commendable data processing capabilities, efficient scalability, and robust fault tolerance mechanisms. Nonetheless, it is worth noting that Hadoop excels in batch data processing, albeit at the cost of frequent disk memory access, resulting in increased disk latency and relative sluggishness. Conversely, Spark emerges as the preferred choice for iterative and streaming data processing, owing to its in-memory computational prowess, translating into superior performance vis-à-vis Hadoop. While Spark consistently outperforms Hadoop across most scenarios, it is prudent to acknowledge Hadoop's superior performance in scenarios characterized by substantial data sizes and constrained physical memory resources. Finally, our literature review findings find empirical validation through an experimental examination that compares these two frameworks, employing the iterative Gradient Boost Tree Regression algorithm.

This validation encompasses a two-tiered approach, the initial phase employing benchmark HIGGS datasets across distinct scenarios, encompassing varying dataset sizes under identical system configurations, as well as uniform dataset sizes across divergent cluster configurations. In the second phase of execution, both Hadoop and Spark are applied to actual Covid-19 datasets, illustrating practical scenarios for both frameworks and shedding light on their respective advantages for future endeavours in constructing real-world application models. The ensuing performance evaluations affirm Spark's consistent superiority over Hadoop across all scenarios, with a marked reduction in execution time as cluster size increases for Spark, whereas Hadoop MapReduce exhibits linear execution time degradation under analogous conditions.

The findings of this research paper can guide future research efforts by highlighting the suitability of Apache Spark and MapReduce Hadoop for specific use cases. Researchers can consider these frameworks' strengths and weaknesses when choosing platforms for various data processing needs, considering factors such as data size and type, memory resources, nature of algorithm, and processing requirements. Additionally, the paper's experimental validation using real-world datasets provides practical insights into the performance of these frameworks, aiding future endeavours in building real-world application models.

REFERENCES

[1] P. Muthulakshmi and S. Udhayapriya,*A Survey on big data issues and challenges*, Int. J. Comput. Sci. Eng., Vol. 6, No. 6, pp. 1238–1244, 2018, doi: 10.26438/ijcse/v6i6.12381244.

[2] T. R. Rao, P. Mitra, R. Bhatt, and A. Goswami,*The big data system, components, tools, and technologies: a survey*, Knowl. Inf. Syst., Vol. 60, No.3, pp. 1165–1245, 2019, doi: 10.1007/s10115-018-1248-0.

[3] C. Dobre and F. Xhafa,*Parallel programming paradigms and frameworks in Big Data Era*, Int. J. Parallel Program., Vol. 42, No.5, pp. 710-738, 2014, doi: 10.1007/s10766-013-0272-7.

[4] H. Singh and S. Bawa,*A mapreduce-based efficient H-bucket PMR quadtree spatial index*, Computer System Science and Engineering, Vol. 32, No. 5, pp. 405–415, 2017.

[5] H. Singh and S. Bawa,*IGSIM: An improved integrated Grid and MapReduce-Hadoop architecture for spatial data: Hilbert TGS R-Tree-based IGSIM*, Concurrency Computation : Practice and Experience, John Wiley & Sons, Vol. 31, Iss. 17, 2019, doi:https://doi.org/10.1002/cpe.5202.

[6] P. Sewal and H. Singh, *A Critical Analysis of Apache Hadoop and Spark for Big Data Processing*, in 2021 6th International Conference on Signal Processing, Computing and Control (ISPCC), pp. 308–313, 2021, doi: 10.1109/ISPCC53510.2021.9609518.

[7] P.Sewal and H. Singh,*Analyzing distributed Spark MLlib regression algorithms for accuracy, execution efficiency and scalability using best subset selection approach*, Multimedia Tools and Applications, 2023, doi: 10.1007/s11042-023-17330-5.

[8] P. Sewal and H. Singh, *A Machine Learning Approach for Predicting Execution Statistics of Spark Application*, in the proceedings of the 2022 7th Int. Conf. Parallel, Distrib. Grid Comput., pp. 331–336, 2022, doi: 10.1109/PDGC56933.2022.10053356.

[9] Daniel Whiteson, *UCI Machine Learning Repository: HIGGS Data Set*, https://archive.ics.uci.edu/ml/datasets/HIGGS (accessed Dec. 05, 2022).

[10] Kaggle, *Your Machine Learning and Data Science Community*,https://www.kaggle.com/ (accessed Mar. 23, 2022).

[11] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker andI. Stoica, *Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing*, in Proceedings of NSDI 2012: 9th USENIX Symposium on Networked Systems Design and Implementation, pp. 15–28., 2012

[12] L. Gu and H. Li, *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, IEEE, pp. 721–727, 2013, doi: 10.1109/HPCC.and.EUC.2013.106.

[13] S. Han, W. Choi, R. Muwafiq, and Y. Nah, *Impact of Memory Size on Bigdata Processing based on Hadoop and Spark*, in Proceedings of the International Conference on Research in Adaptive and Convergent Systems, New York, NY, USA: ACM, pp. 275–280, 2017 doi: 10.1145/3129676.3129688.

[14] S. Gopalani and R. Arora, *Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means*, Int. J. Comput. Appl., Vol. 113, No. 1, pp. 8–11, Mar. 2015, doi: 10.5120/19788-0531.

[15] T. Sharma, D. V. Shokeen and D. S. Mathur, *Multiple K Means++ Clustering of Satellite Image Using Hadoop MapReduce and Spark*, Int. J. Adv. Stud. Comput. Sci. Eng., Vol. 5, No. 4, pp. 23–31, May 2016, [Online]. Available: http://arxiv.org/abs/1605.01802

[16] X. Lin, P. Wang, and B. Wu, *Log analysis in cloud computing environment with Hadoop and Spark*, Proc. 2013 5th IEEE Int. Conf. Broadband Netw. Multimed. Technol. IEEE IC-BNMT(2013), pp. 273–276, 2013, doi: 10.1109/ICB-NMT.2013.6823956.

[17] A. Mostafaeipour, A. J. Rafsanjani, M. Ahmadi, and J. A. Dhanraj, *Investigating the performance of Hadoop and Spark platforms on machine learning algorithms*, J. Supercomput., vol. 77, no. 2, pp. 1273–1300, 2021, doi: 10.1007/s11227-020-03328-5.

[18] Y. Samadi, M. Zbakh, and C. Tadonki, *Comparative study between Hadoop and Spark based on Hibench benchmarks*, Proc. 2016 Int. Conf. Cloud Comput. Technol. Appl. CloudTech 2016, pp. 267–275, 2017, doi: 10.1109/CloudTech.2016.7847709.

[19] A. Singh, A. Khamparia, and A. K. Luhach, *Performance comparison of Apache Hadoop and Apache Spark*, in Proceedings of the Third International Conference on Advanced Informatics for Computing Research - ICAICR '19, New York, New York, USA: ACM Press, pp. 1–5, 2019. doi: 10.1145/3339311.3339329.

[20] A. V. Hazarika, G. J. S. R. Ram, and E. Jain, *Performance comparision of Hadoop and spark engine*, in 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), IEEE, pp. 671–674, 2017. doi: 10.1109/I-SMAC.2017.8058263.

[21] E. P. S. Castro, T. D. Maia, M. R. Pereira, A. A. A. Esmin, and D. A. Pereira, *Review and comparison of Apriori algorithm implementations on Hadoop-MapReduce and Spark*, Knowl. Eng. Rev., Vol. 33, No. e9, pp. 1–25, Jul. 2018, doi: 10.1017/S0269888918000127.

[22] K. Aziz, D. Zaidouni, and M. Bellafkih, *Real-time data analysis using Spark and Hadoop*, Proc. 2018 Int. Conf. Optim. Appl. ICOA 2018, pp. 1–6, 2018, doi: 10.1109/ICOA.2018.8370593.

[23] Y. Benlachmi, A. El Yazidi, and M. L. Hasnaou, *A Comparative Analysis of Hadoop and Spark Frameworks using Word Count Algorithm*, Int. J. Adv. Comput. Sci. Appl., Vol. 12, No. 4, pp. 778–788, 2021, doi: 10.14569/IJACSA.2021.0120495.

[24] S. Ketu, P. K. Mishra, and S. Agarwal, *Performance Analysis of Distributed Computing Frameworks for Big Data Analytics: Hadoop Vs Spark*, Comput. y Sist., vol. 24, no. 2, pp. 669–686, 2020, doi: 10.13053/CyS-24-2-3401.

[25] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, *Real-Time Big Data Stream Processing Using GPU with Spark Over Hadoop Ecosystem*, Int. J. Parallel Program., vol. 46, no. 3, pp. 630–646, 2018, doi: 10.1007/s10766-017-0513-2.

[26] H.Singh and S. Bawa, *Predicting Covid-19 statistics using machine learning regression models Li-MuLi-Poly*, Multimedia Systems, Vol. 28, pp. 113-120, 2022, doi: 10.1007/s00530-021-00798-2.

[27] M. M. George and P. S. Rasmi, *Performance Comparison of Apache Hadoop and Apache Spark for COVID-19 data sets*, 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), pp. 1659–1665, Feb. 2022, doi: 10.1109/ICSSIT53264.2022.9716232.

[28] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Commun. Acm, Vol. 51, No. 1, pp. 107–113, 2008, doi: 10.1145/1330000/1327492.

[29] S. Shahrivari, *Beyond batch processing: Towards real-time and streaming big data*, Computers, Vol. 3, No. 4. MDPI AG, pp. 117–129, Dec. 01, 2014. doi: 10.3390/computers3040117.

[30] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, *Big data analytics on Apache Spark*, International Journal of Data Science and Analytics, Vol. 1, No. 3–4. Springer International Publishing, pp. 145–164, 2016. doi: 10.1007/s41060-016-0027-9.

[31] J. G. Shanahan and L. Dai, *Large Scale Distributed Data Science using Apache Spark*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, pp. 2323–2324, 2015. doi: 10.1145/2783258.2789993.

[32] S. Pan, *The Performance Comparison of Hadoop and Spark*,Culminating Proj. Comput. Sci. Inf. Technol. 7, 2016, [Online]. Available: https://repository.stcloudstate.edu/csit_etds/7/

[33] Apache SparkTM, *Unified Analytics Engine for Big Data*, https://spark.apache.org/ (accessed Jan. 05, 2023).

[34] J. Shi et al., *Clash of the titans: Mapreduce vs. spark for large scale data analytics*, Proc. VLDB Endow., Vol. 8, No. 13, pp. 2110–2121, 2015, doi: 10.14778/2831360.2831365.

[35] Y. Liu and W. Wei, *A Replication-Based Mechanism for Fault Tolerance in MapReduce Framework*, Math. Probl. Eng., Vol. 2015, pp. 1–7, 2015, doi: 10.1155/2015/408921.

[36] Apache Hadoop, *https://hadoop.apache.org/ (accessed Dec. 23, 2023).*

[37] H. Singh and S. Bawa, *A MapReduce-based scalable discovery and indexing of structured big data*, Futur. Gener. Comput. Syst., Vol. 73, pp. 32–43, Aug. 2017, doi: 10.1016/j.future.2017.03.028.

[38] B. Saraladevi, N. Pazhaniraja, P. V. Paul, M. S. S. Basha, and P. Dhavachelvan, *Big data and Hadoop-A study in security perspective*, Procedia Comput. Sci., Vol. 50, pp. 596–601, 2015, doi: 10.1016/j.procs.2015.04.091.

[39] V. K. Vavilapalli et al., *Apache Hadoop YARN*, in Proceedings of the 4th annual Symposium on Cloud Computing, New York, NY, USA: ACM, Oct. 2013, pp. 1–16. doi: 10.1145/2523616.2523633.

[40] K. Aziz, D. Zaidouni, and M. Bellafkih, *Big Data Processing using Machine Learning algorithms: MLlib and mahout use case*, in ACM International Conference Proceeding Series, 2018, pp. 2–7. doi: 10.1145/3289402.3289525.

[41] M. Assefi, E. Behravesh, G. Liu, and A. P. Taft, *Big data machine learning using apache spark MLlib*, Proc. - 2017 IEEE Int. Conf. Big Data, Big Data 2017, Vol. 2018-Janua, pp. 3492–3498, 2017, doi: 10.1109/BigData.2017.8258338.
[42] X. Meng et al., *MLlib: Machine learning in Apache Spark*, J. Mach. Learn. Res., Vol. 17, pp. 1–7, 2016.
[43] A. Sarkar, J. Guo, N. Siegmund, and S. Apel, *Cost-Efficient Sampling for Performance Prediction of Configurable Systems*, 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 342–352, 2015, doi: 10.1109/ASE.2015.45.
[44] M. Last, *Improving data mining utility with projective sampling*, Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., pp. 487–495, 2009, doi: 10.1145/1557019.1557076.

Edited by: Chiranji Lal Chowdhary
Special issue on: Scalable Machine Learning for Health Care: Innovations and Applications
Received: Oct 12, 2023
Accepted: Mar 12, 2024