Journal of Autonomous Intelligence (2024) Volume 7 Issue 1
doi: 10.32629/jai.v7i1.746

ORIGINAL RESEARCH ARTICLE

An enhanced distributed framework for real-time performance testing of large scale IoT dataset using big data analytic tools

Vijay Hasanpuri*, Chander Diwaker

Department of Computer Science & Engineering, University Institute of Engineering and Technology, Kurukshetra University, Kurukshetra, Haryana 136119, India

*** Corresponding author:** Vijay Hasanpuri, uietphd2124vijayhasanpuri@kuk.ac.in

ABSTRACT

The demand for analyzing enormous IoT datasets is rising in parallel with the popularity of the IoT. There are considerable obstacles to effective processing and analysis due to the amount, velocity, and variety of IoT data. In this research, we present a distributed system that makes use of big data analytic tools like Apache Hive, Spark, and Hadoop to efficiently test the performance of massive IoT datasets. The framework addresses the lack of a comprehensive solution by providing a scalable and fault-tolerant architecture. We discuss the motivation behind real-time performance testing in the context of big data analytics for IoT datasets and highlight the need for a distributed framework. A literature review is conducted to explore existing performance testing frameworks, big data analytic tools, and approaches for performance testing big data analytics. The proposed framework's key components, including dataset generation, test scenario specification, cluster configuration, performance metrics collection, analysis and visualization modules, and implementation details, including tool choices, are discussed. An experimental evaluation is conducted to validate the framework's performance, and it is suggested to incorporate blockchain technology. Overall, the proposed framework offers a comprehensive solution for real-time performance testing of large-scale IoT datasets, providing organizations and researchers with a valuable tool to ensure efficient and reliable IoT data processing and analysis.

Keywords: big-data; IoT; Hadoop; MapReduce; Apache Hive; benchmark; Spark; WordCount; TeraSort

ARTICLE INFO

Received: 18 June 2023
Accepted: 31 July 2023
Available online: 9 October 2023

1. Introduction

The rapid growth of the IoT has revolutionized various industries, generating vast amounts of data from interconnected devices. Analyzing large-scale IoT datasets presents unique challenges due to the volume, velocity, and variety of the data[1]. The term "big data analytics" was coined to describe the method used to derive actionable intelligence from massive data stores[2]. Finding hidden patterns, correlations, and trends in large amounts of data requires the application of sophisticated analytic methods such as statistical analysis, machine learning, data mining, and predictive modelling. Data-driven decisions, a competitive edge, and industry-wide innovation are all possible thanks to big data analytics[3,4]. The performance, scalability, and responsiveness of a system under defined workloads and conditions are the primary emphasis of performance testing, an integral part of software testing. To ensure the system performs as expected, it is necessary to monitor and analyze key performance indicators such as response times, throughput, resource utilization, and scalability[4]. When it comes to the processing and analysis of massive datasets, performance testing plays a critical role

in the context of big data analytics. The system's responsiveness to massive amounts of data and real-time analytics may be validated, and resource allocation, performance bottlenecks, and system responsiveness can all be optimized for the business. Big data analytics rely heavily on distributed clusters[5]. They are made up of a network of computers called nodes that can handle and analyze massive amounts of data simultaneously. To meet the huge data processing needs of big data analytics, distributed clusters provide the scalability and computational capacity required. Distributed clusters are widely used for processing and analyzing massive data, and popular distributed computing frameworks include Apache Hadoop, Apache Hive, and Apache Spark. Frameworks like this facilitate efficient parallel processing and fault tolerance[6–8] by distributing data and computational workloads across numerous nodes. Using big data analytic technologies like Apache Hive, Apache Spark, and Apache Hadoop, this study proposes a distributed architecture for performance testing of large-scale IoT datasets[9].

1.1. Motivation

The increasing adoption of IoT devices has resulted in a tremendous influx of data, making efficient processing and analysis of large-scale IoT datasets crucial. Performance testing plays a vital role in ensuring the efficiency and effectiveness of data processing and analysis. It helps organizations optimize resource utilization, identify performance bottlenecks, ensure real-time responsiveness, and validate data integrity. However, existing performance testing approaches lack comprehensive frameworks specifically designed for the unique challenges of analyzing large-scale IoT datasets. This motivates the need for a dedicated framework that incorporates big data analytic tools to address these challenges[7–10].

1.2. Organization of the paper

In this paper Section 1 gives a brief introduction to big data analytic tools and its performance for IoT datasets, and motivation for writing this particular paper. While section 2 provides a comprehensive literature review of related works in the field of performance testing for IoT datasets and big data analytics. It highlights the existing research gaps and identifies the need for a dedicated framework. Section 3 includes the bigdata analysis tools used for this work with their architecture and workflow. Furthermore, section 4 presents the proposed enhanced distributed framework for performance testing of large-scale IoT datasets. The framework architecture, components, and integration with Apache Hive, Apache Spark, and Apache Hadoop are described in detail. Section 5 discusses the implementation details of the framework and the experimental setup used to evaluate its performance. It presents the performance metrics and benchmarks used for analysis. Section 6 presents the experimental results and performance evaluation of the framework. It compares the performance of the proposed framework with existing approaches and discusses the findings. Finally, Section 7 concludes the paper by summarizing the contributions of the research and emphasizing the significance of the proposed framework for performance testing of large-scale IoT datasets using big data analytic tools with suitable future directions.

Contribution:

This paper's significant contributions include:

- Improves performance assessment of massive Internet of Things datasets proposed in this research using a distributed cluster-based system.
- This work performs to leverage big data analytic tools such as Apache Hive, Apache Spark, and Apache Hadoop to handle the scale, complexity, and real-time requirements (such as smart city data, healthcare data, agricultural data etc.) of IoT data analysis.
- Analyses the big data analysis tools such as Hive, Spark and Hadoop performance in terms of execution time or response time and throughput with two workloads known as TeraSort and WordCount, while dealing with parameters for input split and shuffle.

2. Related work

The literature review presented in this section serves as a critical foundation for our research, exploring the vast body of knowledge surrounding “performance testing of bigdata analytic tools using IoT datasets”, and “distributed and cluster based performance testing frameworks”. By examining and synthesizing previous studies, we can build upon existing knowledge and contribute new insights to the field.

One the basis of micro-benchmark experiment, Shi et al.[11] suggested two profiling tools to measure the performance of the Apache MapReduce (Hadoop) and Spark frameworks. Iterative and Batch jobs are used in the comparative analysis of these frameworks. Shi et al.[11] conducted a research which takes into account three elements: caching, executive model, and shuffling. On the basis of CPU, network bounds and disc, there were some workloads selected such as k-means, Linear Regression (LR), Wordcount, Sort, Linear Regression, and PageRank, to assess the system behaviour[12]. The map and reduce functions were disabled for any workloads other than a Sort. Up to 60 map and 120 reduce tasks can be defined for the sort’s reduce task. To prevent further spills when sorting the map data, 550 MB are set aside for the map output buffer. Eight discs are used to hold the intermediate data produced by Spark, and four threads are configured for each worker. Spark runs WordCount with different data sets (1, 40, and 200 GB) quicker than MapReduce[12]. The function sort-by-key() makes advantage of the TeraSort. To find out how each language affects the performance of the systems as a whole, Thiruvathukal et al.[13] looked at the significance and implications of the Java Virtual Machine (JVM) and languages like Scala and Python and proposed a thorough benchmarking test for cloud-based applications and the MPI messaging protocol while taking common parallel analysis into account. The benchmark methodologies that are being suggested are intended to mimic a typical picture analysis. As a result, Thiruvathukal et al.[13] provided two clusters: a big supercomputer cluster with one node that uses THETA and a mid-sized cluster with 126 nodes that runs COOLEY. Significantly, it increased the settings of some key Spark parameters, including the executor and driver memory. The COOLEY and THETA frameworks have been suggested as being advantageous for current research projects and high-performance computing (HPC) environments.

The comparison of the Apache Flink and Apache Spark frameworks for large-scale data processing is presented by Marcue et al.[14], and also presented a novel methodology for benchmarking batch processing workloads such as Grep, TeraSort, and WordCount as well as iterative workloads such as Page Rank and k-means. These workloads took into account the four most crucial factors that impact consumption of resources, scalability, and execution time. Up to 100 nodes have been employed in a Spark and Flink cluster by Grid 5000[15]. The criteria for comparing the performance of the Hadoop and Spark frameworks have been looked into by Samadi et al.[16]. The amount and configuration of the input data stayed constant in his work to allow for an unbiased comparison. Eight HiBench suite benchmarks were used in their experiment. For each case and size, the input data was automatically created, and the algorithm was run multiple times to determine the execution time and throughput. Spark demonstrated a larger participation of the CPU in I/Os when microbenchmarks (Short and TeraSort) were deployed on both systems, whereas Hadoop processed mostly user workloads. Samadi et al.[16] concluded that Spark, which performs maps and reduces on disc, is faster and far more powerful than Hadoop MapReduce for processing data in memory.

Samadi et al.[17] suggested a virtual machine based on Hadoop and Spark in another paper to take use of virtualization. Mavridis and Karatza[18] looked at the computing frameworks, specifically Apache Hadoop and Apache Spark. The Apache webserver log file was examined in this inquiry to compare the performance of the two frameworks. To determine the execution time or response time of each application, proposed a variety of them and ran numerous experiments. A performance comparison between MapReduce and Spark on the basis k-means algorithm which was presented by Gopalani and Arora[19]. A data set designed for this method and accounted for both single and double nodes when determining how long each trial would take to complete. According to the findings, Spark can be up to three times faster than MapReduce, but only if sufficient memory

is available[20]. A unified cloud platform with batch processing capabilities above independent log analysis tools has been presented by Lin et al.[21]. Various data input sizes were employed. When using k-means, the delay schedule and overall Spark performance suffered as the data size increased and overflowed the available RAM. Even still, Hadoop's average performance was still six times lower than the overall performance. However, when using disk-based queries, Shark exhibits a considerable performance gain. Petridis et al.[22] have examined the default parameters of spark. Developers and system administrators were provided with a guide to help and replace the default parameter values with the proper parameter values using a trial-and-error approach. Pajooh et al.[23], also had a significant discussion on the relation of IoT and Blockchain technology and proposed a layer-based distributed data storage concept and implementation of a large-scale IoT system powered by blockchain technique.

Some empirical researches are included in the published literature listed in **Table 1**. None of these researches have taken into account real clusters, greater data quantities (700 GB), and additional factors. In our analysis, we used a larger data set, and 27 key factors from the resource utilization, input splits, and shuffle category.

Table 1. Comparison of existing works based on workload, dataset size and parameters used for execution.

Author	Year	Workload and tools used	Datasize	Parameters
Lin X et al.[21]	2013	Page rank, k-means	10,000 to 20 million (m) points	Log analysis
Gopalani S and Arora R[19]	2015	K-mean square	1240 MB	Default
Samadi Y et al.[16]	2016	Machine learning (ML), WebSearch SQL, Micro benchmarks	328 MB	3 parameters
Petridis P et al.[22]	2017	Sort-by-key and k-mean shuffling	400 GB	12 parameters
Mavridis I and Karatza E[18]	2017	Apache Spark, hive, SQL	11 GB	Log analysis
Samadi Y et al.[17]	2018	Machine learning (ML), WebSearch SQL, Micro benchmarks, Apache Spark, Hadoop	1, 5, and 8 GB	3 parameters
Ahmad N et al.[24]	2020	HiBench, TeraSort, WordCount, Apache Spark, Apache Hive	600 GB	18 parameters
de Oliveira BFP et al.[25]	2022	Query processing, Map reduce, hive, pig	300 GB	8 parameters
Proposed work	2023	HiBench, TeraSort, WordCount, Apache Hive, Apache Spark, Apache Hadoop	700 GB (IoT Dataset)	27 parameters

3. Big data analytics tools

Hadoop, Spark, and Hive are three popular big data analytic tools that are widely used for processing and analyzing large-scale datasets. Each tool has its own architecture and workflow that contribute to their effectiveness in handling big data[26].

Hadoop is a free software platform for distributed file storage and computation. The Hadoop Distributed File System (HDFS) and the MapReduce processing engine form the backbone of its architecture. HDFS is responsible for distributed storage, breaking data into blocks and replicating them across multiple nodes in a cluster for fault tolerance. The workflow of Hadoop involves data ingestion into HDFS, followed by the execution of MapReduce jobs that process and analyze the data in parallel across the cluster. **Figure 1** depicts the effective architecture for MapReduce.

Its architecture includes the Spark Core engine, which provides the foundation for distributed processing, and additional libraries such as Spark SQL, Spark Streaming, and Spark Mllib for various data processing tasks. Spark leverages in-memory computing to achieve high performance and offers a more flexible programming model compared to Hadoop[27]. Users can then perform transformations and actions on these distributed datasets using Spark's API. Spark automatically optimizes the execution plan and processes the data in memory, resulting in faster processing and analysis. The workflow of Apache Spark is depicted in **Figure 2**.

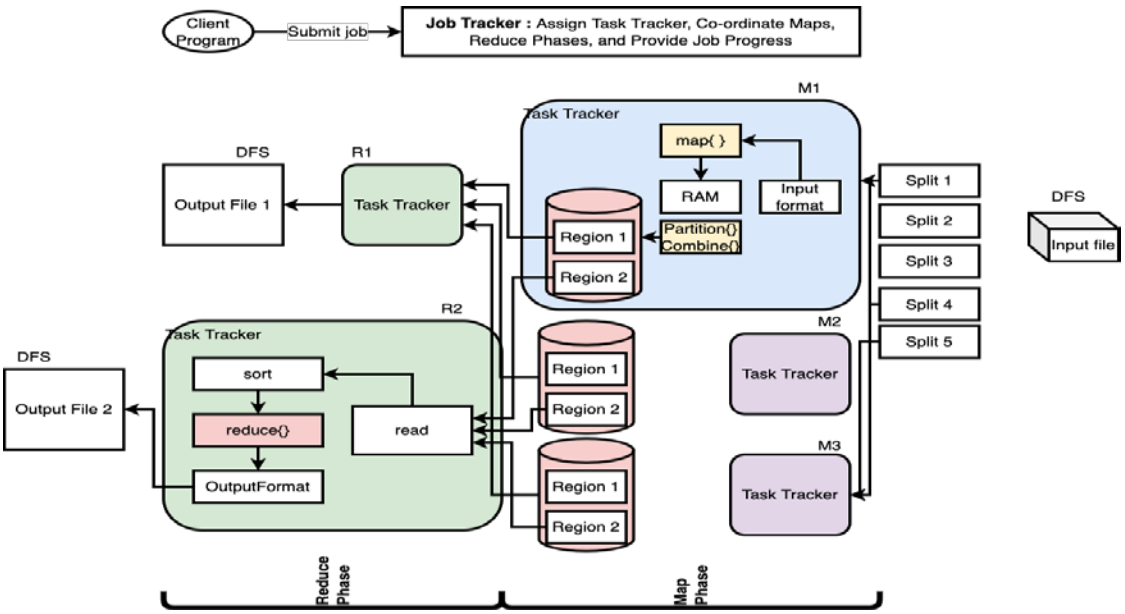


Figure 1. An effective architecture of Hadoop MapReduce[23].

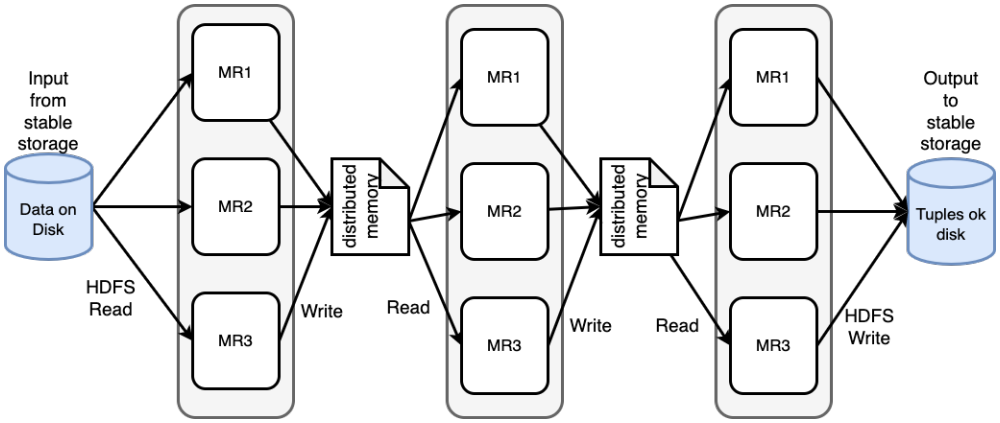


Figure 2. Distributed memory based Apache Spark workflow[23].

Apache Hive is an extension to Hadoop that serves as a data warehouse. Its architecture (shown in **Figure 3**) includes the Hive Metastore, which stores metadata about Hive tables, and the Hive Query Processor, responsible for parsing and executing HiveQL queries. Hive translates HiveQL queries into MapReduce or Spark jobs for execution on the underlying distributed cluster[28]. The workflow of Hive begins with data ingestion into HDFS, followed by the creation of Hive tables and the definition of their structure using HiveQL. Users can then write HiveQL queries to perform data transformations, aggregations, and analysis. Hive processes these queries by generating the necessary MapReduce or Spark jobs and submits them to the Hadoop or Spark cluster for execution.

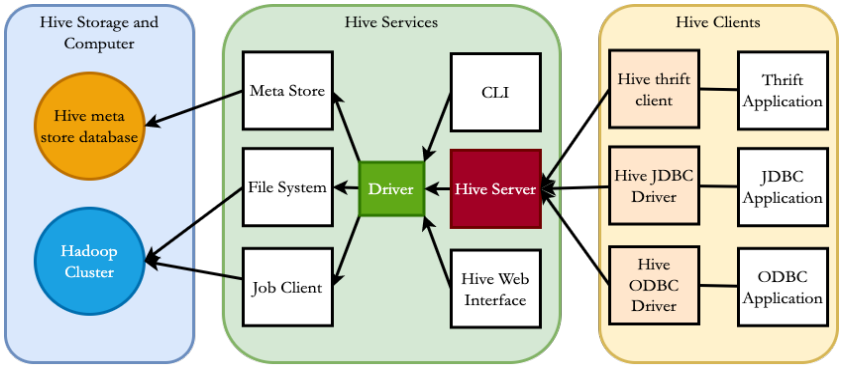


Figure 3. Apache Hive architecture[8].

In summary, Hadoop provides distributed storage and processing capabilities through its HDFS and MapReduce components, Spark offers fast and flexible cluster computing with in-memory processing, and Hive provides a data warehousing infrastructure with a SQL-like query language on top of Hadoop. Each tool has its own architecture and workflow that enables efficient processing and analysis of large-scale IoT datasets[29].

4. Proposed framework

The proposed framework aims to address the challenges in performance testing of large-scale IoT datasets using big data analytic tools. It provides a comprehensive and distributed cluster-based approach to efficiently process and analyze the data. The proposed framework uses blockchain technology for securing data in clusters. By incorporating blockchain technique into IoT data security, real-time performance testing is enhanced by ensuring transparent audit trails, decentralized consensus and tamper resistant data integrity. Blockchain also provides a reliable and robust framework for monitoring and validating real-time IoT dataset. The architecture of the framework consists of multiple key components that work together to facilitate performance testing and analysis. Those key components are:

Dataset generation module: The dataset generation module is responsible for generating realistic and representative datasets for performance testing. It takes into account the characteristics of IoT data, such as volume, variety, and velocity, and generates synthetic or real-world datasets that closely resemble the production data. This module ensures that the generated datasets are diverse and representative of the actual data to accurately simulate real-world scenarios.

Test scenario specification module: The test scenario specification module allows users to define and configure the performance testing scenarios. Users can specify various parameters such as the workload intensity, data distribution, and query types to be executed on the datasets. This module provides flexibility in defining custom test scenarios that mimic real-world usage patterns and workload variations.

Cluster configuration module: The cluster configuration module enables users to configure and manage the distributed cluster environment for performance testing. It provides options to specify the number of nodes, their configurations, and the allocation of computing resources. This module ensures that the cluster is properly set up to handle the large-scale IoT datasets and optimize the utilization of resources during testing.

Performance metrics collection module: The performance metrics collection module is responsible for collecting and monitoring various performance metrics during the testing process. This module enables real-time performance monitoring and provides insights into the performance of the system under different workloads and conditions.

Analysis and visualization module: The analysis and visualization module processes the collected real time performance metrics and provides meaningful insights for performance evaluation. It leverages advanced analytics algorithms and visualization techniques to identify performance bottlenecks, analyze system behavior, and detect anomalies. This module helps in interpreting the performance results and facilitates decision-making for optimizing system performance.

Figure 4 depicts the steps for workflow of proposed framework. The proposed framework follows a well-defined workflow to conduct real-time performance testing of large-scale IoT datasets. The workflow includes the following steps:

1) Dataset generation: The framework generates synthetic or real-world datasets that closely resemble the production data, considering the volume, variety, and velocity of IoT data.
2) Test scenario specification: Users define and configure the test scenarios, specifying parameters such as workload intensity, data distribution, and query types to be executed on the datasets.

3) Cluster configuration: Users configure the distributed cluster environment, specifying the number of nodes, their configurations, and resource allocation to optimize performance.
4) Performance testing execution: The framework executes the defined test scenarios on the generated datasets using the configured cluster. It collects performance metrics during the testing process.
5) Performance metrics collection: The framework captures performance metrics such as response time, throughput, resource utilization, and system latency during the testing process.
6) Analysis and visualization: The framework processes the collected performance metrics and provides analysis and visualization capabilities to interpret the results and identify performance bottlenecks.
7) Optimization and iteration: Based on the analysis, users can optimize the system configuration, workload distribution, or other parameters to improve performance. The process can be iterated to achieve desired performance goals.

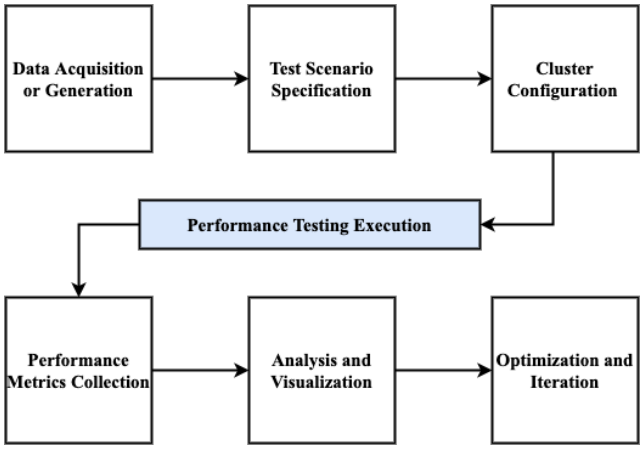


Figure 4. Proposed framework workflow.

By following this workflow, the proposed framework enables efficient and effective performance testing of large-scale IoT datasets using big data analytic tools. It provides insights into system performance and facilitates the optimization of resources and algorithms for efficient processing and analysis of IoT data.

5. Experimental evaluation

5.1. Extraction of parameter tools and technologies

In the proposed framework for performance testing of large-scale IoT datasets, careful consideration is given to selecting the appropriate tools and technologies. The choice of tools depends on the specific requirements and objectives of the performance testing. However, some commonly chosen tools for this purpose include Apache Hive, Apache Spark, and Apache Hadoop.

Apache Hive: Apache Hive is selected for its ability to provide a SQL-like interface and its compatibility with Hadoop. It allows users to write queries in HiveQL, which can be translated into MapReduce or Spark jobs for distributed processing. This makes it suitable for analyzing and querying large-scale IoT datasets efficiently.

Apache Spark: Apache Spark is chosen for its in-memory computing capabilities and high-performance distributed processing. It offers flexible APIs for data processing, machine learning, and streaming analytics. Spark's ability to handle real-time streaming data and its integration with other big data tools make it a valuable choice for the proposed framework.

Apache Hadoop: Apache Hadoop is selected for its distributed storage and processing capabilities. Hadoop's MapReduce framework enables parallel processing of data, making it suitable for handling the volume and complexity of IoT datasets.

In the experimental evaluation of the proposed framework for real time performance testing of large-scale IoT datasets, a suitable test environment is set up. The test environment consists of the necessary hardware and software components to replicate a real-world scenario. The hardware setup includes a distributed cluster of machines or virtual machines to simulate a scalable and distributed computing environment. The number of nodes in the cluster can vary depending on the desired scale and complexity of the IoT datasets. To provide security in data, blockchain techniques is incorporated in this work through the tool named as Hyperledger Fabric. The software setup involves installing and configuring the selected big data analytic tools such as Apache Hive, Apache Spark, and Apache Hadoop. The versions of these tools should be compatible and properly integrated with each other. Additionally, any required dependencies or libraries are installed to support the performance testing scenarios. The experimental setup details of shown in **Table 2**.

Table 2. Experimental cluster.

Server configuration	Processor	2.9 GHz
	Local storage	1 TB
	Primary memory	16 GB
Node configuration	No. of nodes	10
	Local storage	1 TB (each), 10 TB (total)
	Primary memory	16 GB
	CPU	Intel ® Xeon ® CPU @ 3.4 GHz
	Cores	8 (each), 80 (total)
Software	OS	Windows 10
	JDK	8
	Apache Hive	3.1.2
	Apache Hadoop	3.2.1
	Apache Spark	3.0.3
Workload	Micro benchmarks	TeraSort and WordCount

5.2. Workloads

As previously mentioned, for the trials in this study, we selected two workloads:

WordCount: This job counts the occurrences of distinct words in a text or sequence file and is map-dependent. RandomTextWriter is used to generate the input data.

TeraSort: In order to gauge cluster performance, Hadoop published the TeraSort package in 2008.

5.3. Performance metrics and evaluation criteria

To evaluate the performance of the framework, various performance metrics and evaluation criteria are considered. These metrics provide insights into the efficiency, scalability, and reliability of the framework in processing and analyzing large-scale IoT datasets[25].

Common performance metrics include:

Execution time or response time: The time taken to complete the performance testing scenarios.

$$response\ time = completion\ time - arrival\ time$$

Throughput: The number of records or operations processed per unit of time.

$$throughput = \frac{number\ of\ requests}{total\ time}$$

Resource utilization: The utilization of CPU, memory, and disk I/O during the performance testing.

$$Resource\ Utilization\ (RU) = \frac{resource\ usage}{total\ resource\ capacity} \times 100$$

Scalability: The ability of the framework to handle increasing workloads by adding more nodes to the distributed cluster.

$$Scalability\ (S) = \frac{new\ performance - initial\ performance}{initial\ performance}$$

Fault tolerance: The ability of the framework to recover from failures and continue processing without data loss.

$$Fault\ tolerance\ (FT) = \frac{Total\ successful\ requests}{requests} \times 100$$

Data consistency: The consistency of the results obtained from the performance testing scenarios.

The evaluation criteria are defined based on the expected performance targets and the specific requirements of the IoT data processing and analysis tasks. These criteria serve as benchmarks to assess the effectiveness and efficiency of the framework. The above mentioned performance metrices are standard metrices which are obtained by various researchers for their analysis. In this manuscript, we applied only response time, throughput and scalability. In future, we further apply the remaining metrices.

6. Results and discussion

The analysis involves comparing the performance metrics against the evaluation criteria and identifying any bottlenecks or areas for improvement. It may include examining the impact of different dataset sizes, cluster configurations, and workload variations on the performance of the framework. The results and analysis provide valuable insights into the effectiveness and efficiency of the proposed framework for performance testing of large-scale IoT datasets.

6.1. Execution time

The execution time varies depending on factors such as the quantity of the input data, the number of active nodes, and the type of applications being run. We used the same parameters throughout, such as a fixed number of 50 executors, 8 GB of RAM per executor, and 4 cores per executor, to ensure a fair comparison.

Figure 5a and **5b** show how the size of the datasets and different input split and shuffle parameters affect how quickly MapReduce, Hive, and Spark are executed. This operation makes use of the shuffle parameter (sort.mb 100, sort.factor 2047) and the default input split size (128 MB). **Figure 5c**'s default input split for Spark is 128 MB. According to **Figure 5c**, input split sizes of 256 MB perform better than the default configuration up to 450 GB of data sizes. **Figure 5d** demonstrates how the improvement rate significantly rises when the PL value is set to 300. Similar to **Figure 5e**, Apache Hive's input split and shuffle parameter dataset is shown in **Figure 5f**. It has been shown that Hive performs less well than the other two analytic tools.

Input split with default parameters are used in **Figure 6a** to compare MapReduce TeraSort workloads. In this analysis, Red_Task and InSp have fixed values with a 128 MB default split size. We experimented with several parameter values to see if the size of the divides would still have an effect on the runtime. **Figure 6b** depicts the performance of the TeraSort workload during execution with the MapReduce shuffle parameter. When the parameters are changed from the default configuration of Reduce_100 and task.io_30, the average execution time exhibits linear behaviour for sizes up to 450 GB. In **Figure 6c**, we see the results of a performance analysis of the Spark input split parameter execution on the TeraSort workload. Spark's shuffle behaviour performance for TeraSort workloads is shown in **Figure 6d**. Our investigation includes the two most commonly used default values (buffer = 32 and spark.reducer.maxSizeIn Flight = 48 MB). The execution performance was enhanced up to 700 GB of data volumes when the buffer and maxSizeInFlight were increased

by 128 and 192, respectively. The performance shuffle and input split values for TeraSort workload are also shown in **Figure 6e,f**.

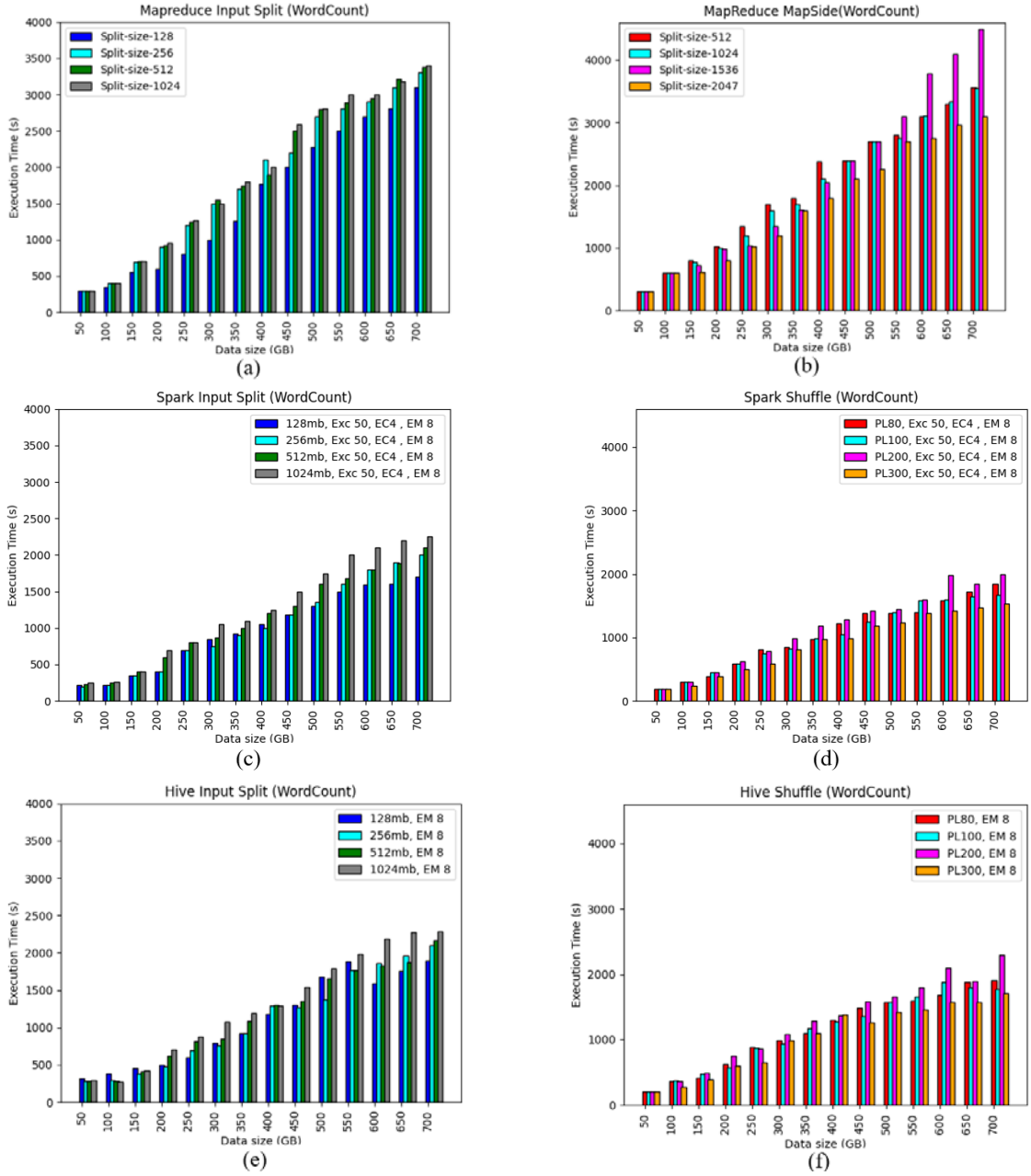
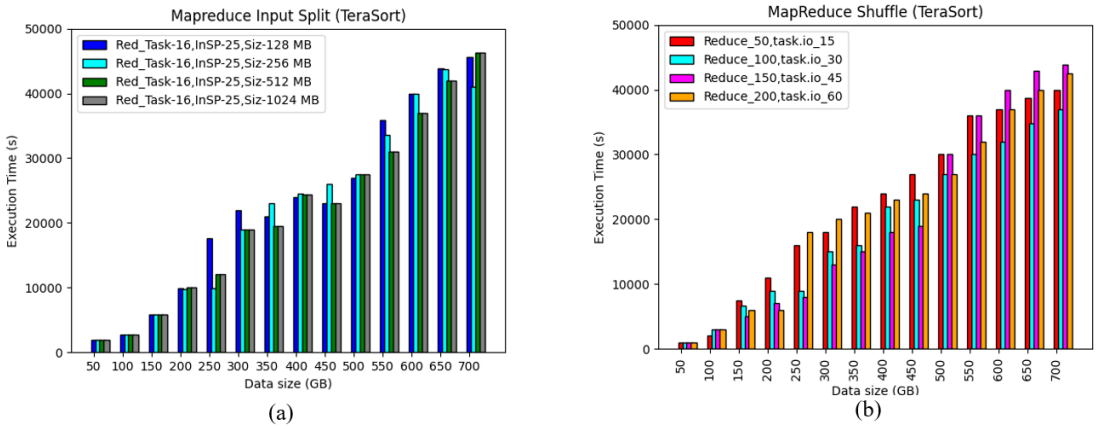


Figure 5. The performance of the WordCount application with a varied number of input splits and shuffle tasks.



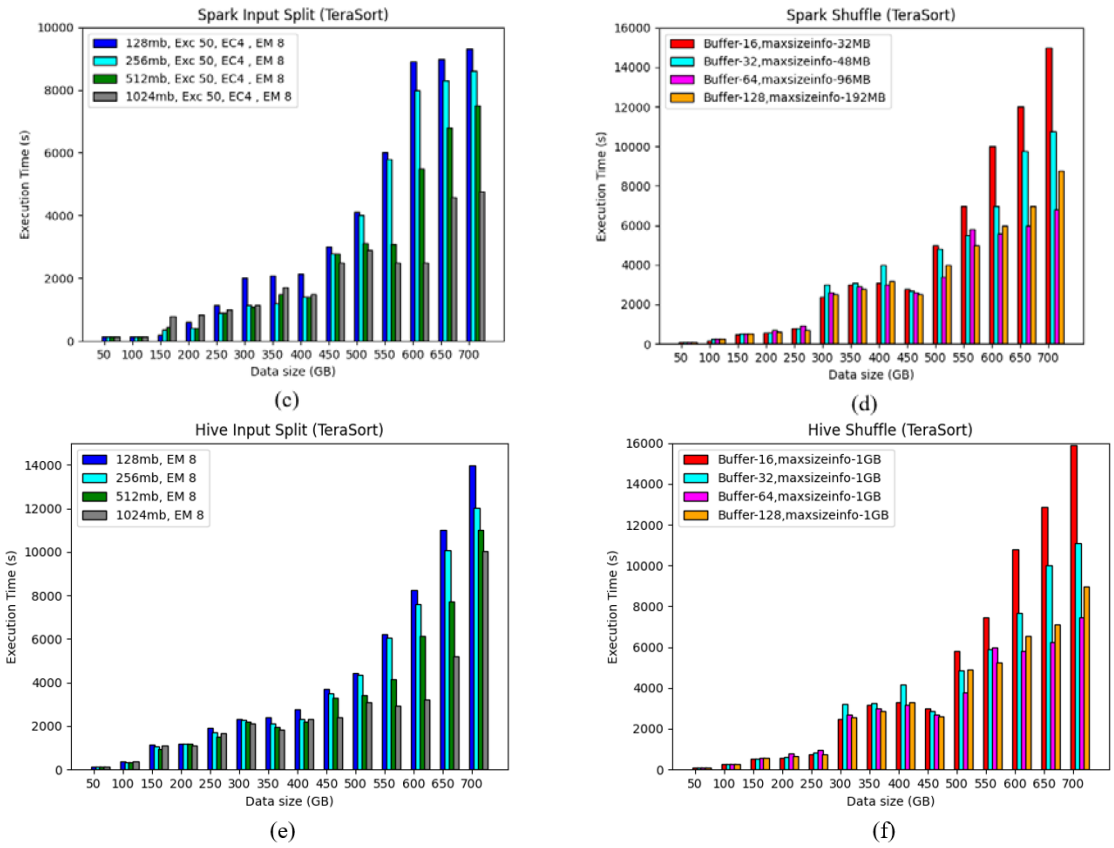


Figure 6. The performance of the TeraSort application with a varied number of input splits and shuffle tasks.

After applying different input splits, **Figure 7** compares Hive, Spark, and MapReduce on the WordCount and TeraSort workloads. We observe that Spark's execution performance improves by more than 2 times with WordCount workloads and data quantities greater than 300 GB.

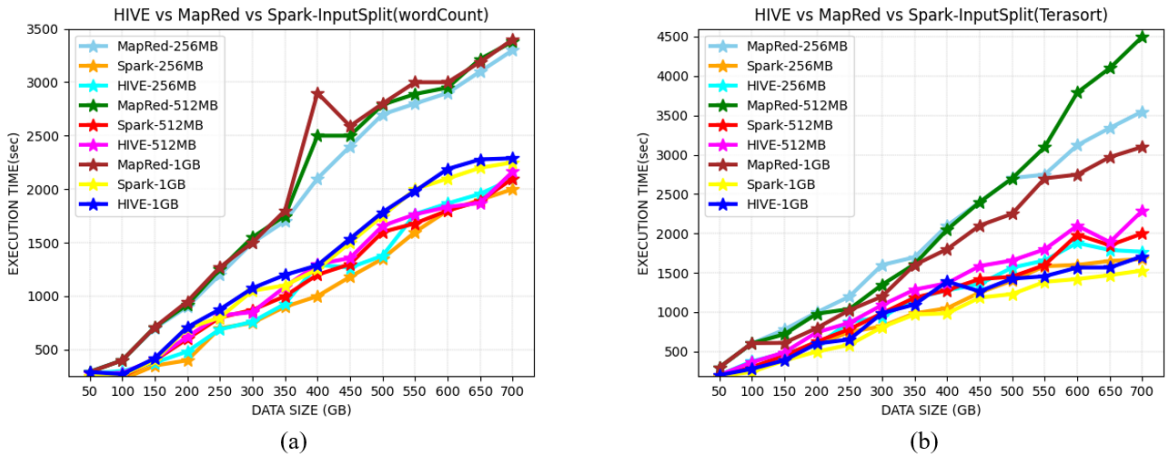
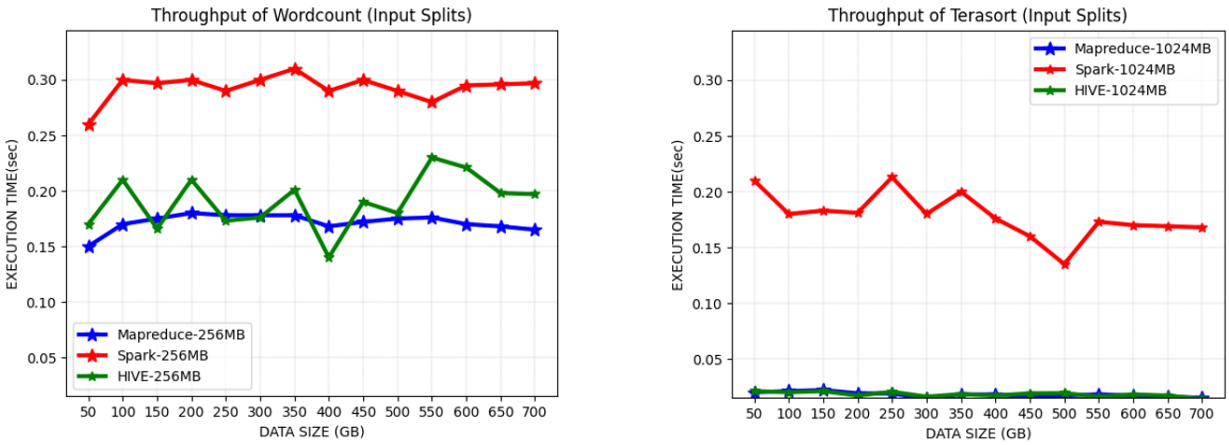


Figure 7. Comparison of Hadoop and Spark with **(a)** WordCount and **(b)** TeraSort workload with varied input splits and shuffle tasks.

6.2. Throughput

All throughput measurements are in megabytes per second. Only the top outcomes from each category were taken into account for this study. We discovered that MapReduce throughput performance for the TeraSort operation decreases when data size increases over 200 GB. It can be seen that the throughput for the Spark TeraSort job fluctuates, whereas the throughput for the WordCount workload is nearly constant. The primary goal of this investigation was to show the throughput variation between the TeraSort and WordCount workloads

for MapReduce and Spark. For the majority of the data sizes, we discovered that the WordCount workload is nearly stable, and that MapReduce is more stable than Spark for the TeraSort task (**Figure 8**).



(a) (b)

Figure 8. Throughput comparison in perspective of **(a)** WordCount and **(b)** TeraSort workload.

7. Conclusion and future direction

In conclusion, the proposed enhanced distributed cluster-based framework for performance testing of large-scale IoT datasets using big data analytic tools addresses the challenges associated with processing and analyzing IoT data. It offers a scalable, fault-tolerant, and comprehensive solution for evaluating the performance of IoT analytics systems. Through the literature review, implementation details, experimental evaluation, and use case discussions, the paper has provided a thorough understanding of the framework's architecture, workflows, and applicability in real-world scenarios. The key contributions, implications, and applications of the framework have been highlighted, paving the way for further advancements in IoT data analytics research and applications. As the IoT continues to grow and generate vast amounts of data, the need for efficient and reliable performance testing frameworks becomes increasingly critical. The proposed framework offers a valuable tool for organizations and researchers working with large-scale IoT datasets, enabling them to unlock the full potential of IoT analytics and derive meaningful insights from their data.

Future research can focus on expanding the framework’s compatibility with a wider range of big data analytic tools. This would provide more flexibility to users and enable the framework to address diverse IoT analytics requirements. Investigating optimization techniques to improve the framework’s performance and resource utilization can be an interesting avenue for future research. Techniques such as data partitioning, load balancing, and query optimization can be explored to enhance the efficiency and scalability of the framework. Conducting real-world use-case validations of the framework on different IoT datasets and analytics scenarios can provide further insights into its effectiveness and applicability. This would involve testing the framework in various industry domains and evaluating its performance and usability in practical IoT analytics scenarios. Future research can explore the integration of advanced analytics techniques, such as machine learning and deep learning, into the framework. This would enable the framework to support more sophisticated analysis tasks and provide deeper insights into the IoT data. By addressing these future directions, the proposed framework can continue to evolve and meet the growing demands of performance testing for large-scale IoT datasets using big data analytic tools, contributing to the advancement of IoT analytics research and applications.

Author contributions

Conceptualization, VH; methodology, VH; software, VH; validation, VH and CD; formal analysis, CD; investigation, CD; resources, CD; data curation, CD; writing—original draft preparation, VH; writing—review and editing, CD. All authors have read and agreed to the published version of the manuscript.

Conflict of interest

The authors declare no conflict of interest.

References

1. Hasanpuri V, Diwaker C. Comparative analysis of techniques for big-data performance testing. In: Proceedings of the 2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC); 25–27 November 2022; Solan, Himachal Pradesh, India. pp. 292–297.
2. Bhardwaj A, Singh R, Deep V, Sharma P. BDT3V—A Technique for big data testing considering 3V's. In: Proceedings of the 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT); 16–18 August 2018; Bangalore, India. pp. 222–225.
3. Jankatti S, Raghavendra BK, Raghavendra S, Meenakshi M. Performance evaluation of Map-reduce jar pig hive and spark with machine learning using big data. *International Journal of Electrical and Computer Engineering* 2020; 10(4): 3811. doi: 10.11591/ijece.v10i4.pp3811-3818
4. Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. *Journal of Systems and Software* 2017; 125: 133–151. doi: 10.1016/j.jss.2016.11.037
5. Chou SC, Yang CT. A high-performance data accessing and processing system for campus real-time power usage. *International Journal of Informatics and Information Systems* 2020; 3(3): 128–135. doi: 10.47738/ijiis.v3i3.98
6. Awasthy N, Valivarthi N. Evolution of hadoop and big data trends in smart world. In: Awasthi S, Sanyal G, Travieso-Gonzalez CM, et al. (editors). *Sustainable Computing: Transforming Industry 4.0 to Society 5.0*. Springer International Publishing; 2023. pp. 99–127.
7. Wu X, He Y. Optimization of the join between large tables in the spark distributed framework. *Applied Sciences* 2023; 13(10): 6257. doi: 10.3390/app13106257
8. Adamov A. Large-scale data modelling in hive and distributed query processing using MapReduce and tez. *arXiv* 2023; arXiv:2301.12454. doi: 10.48550/arXiv.2301.12454
9. Yu P, Tao Y, Zhang J, Jin Y. Design and implementation of a cloud-native platform for financial big data processing course. In: Hong W, Weng Y (editors). *Computer Science and Education*, Proceedings of the 17th International Conference, ICCSE 2022; 18–21 August 2022; Ningbo, China. Springer Nature Singapore; 2022. pp. 180–193.
10. Ahmed S, Abdel-Hamid Y, Hefny HA. Traffic flow prediction using big data and gis: a survey of data sources, frameworks, challenges, and opportunities. *International Journal of Computing and Digital Systems* 2023; 14(1): 613–632. doi: 10.12785/ijcds/140147
11. Shi J, Qiu Y, Minhas UF, et al. Clash of the titans: MapReduce vs. spark for large scale data analytics. *Proceedings of the VLDB Endowment* 2015; 8(13): 2110–211. doi: 10.14778/2831360.2831365
12. Veiga J, Expósito RR, Pardo XC, et al. Performance evaluation of big data frameworks for large-scale data analytics. In: Proceedings of the 2016 IEEE international conference on Big Data; 5–8 December 2016; Washington, DC, USA. pp. 424–431.
13. Thiruvathukal GK, Christensen C, Jin X, et al. A benchmarking study to evaluate apache spark on large-scale supercomputers. *arXiv* 2019; arXiv:1904.11812. doi: 10.48550/arXiv.1904.11812
14. Marcu OC, Costan A, Antoniu G, Pérez-Hernández MS. Spark versus flink: Understanding performance in big data analytics frameworks. In: 2016 IEEE international conference on cluster computing (CLUSTER); 12–16 September 2016; Taipei, Taiwan. pp. 433–442.
15. Bolze R, Cappello F, Caron E, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications* 2006; 20(4): 481–494. doi: 10.1177/1094342006070078
16. Samadi Y, Zbakh M, Tadonki C. Comparative study between hadoop and spark based on hibench benchmarks. In: Proceedings of the 2016 2nd international conference on cloud computing technologies and applications (CloudTech); 24–26 May 2016; Marrakech, Morocco. pp. 267–275.
17. Samadi Y, Zbakh M, Tadonki C. Performance comparison between hadoop and spark frameworks using hibench benchmarks. *Concurrency and Computation: Practice and Experience* 2018; 30(12): e4367. doi: 10.1002/cpe.4367
18. Mavridis I, Karatza E. Log file analysis in cloud with apache hadoop and apache spark. In: Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015); 10–11 September 2015; Krakow, Poland.

19. Gopalani S, Arora R. Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications* 2015; 113(1): 8–11. doi: 10.5120/19788-0531
20. Gu L, Li H. Memory or time: Performance evaluation for iterative operation on hadoop and spark. In: Proceedings of the 2013 IEEE 10th international conference on high performance computing and communications & 2013 IEEE international conference on embedded and ubiquitous computing; 13–15 November 2013; Zhangjiajie, China. pp. 721–727.
21. 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; 17–19 November 2013; Guilin, China. pp. 273–276.
22. Petridis P, Gounaris A, Torres J. Spark parameter tuning via trial-and-error. *arXiv* 2016; arXiv:1607.07348. doi: 10.48550/arXiv.1607.07348
23. Pajooh HH, Rashid MA, Alam F, Demidenko S. IoT Big Data provenance scheme using blockchain on Hadoop ecosystem. *Journal of Big Data* 2021; 8: 114. doi: 10.1186/s40537-021-00505-y
24. Ahmed N, Barczak AL, Susnjak T, Rashid MA. A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data* 2020; 7(1): 110. doi: 10.1186/s40537-020-00388-5
25. de Oliveira BFP, Valente ASO, Victorino M, et al. Analysis of the influence of modeling, data format and processing tool on the performance of hadoop-hive based data warehouse. *Journal of Information and Data Management* 2022; 13(3). doi: 10.5753/jidm.2022.2516
26. Gupta P, Sharma A, Grover J. Rating based mechanism to contrast abnormal posts on movies reviews using MapReduce paradigm. In: Proceedings of the 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO); 7–9 September 2016; Noida, India. pp. 262–266.
27. Gupta D, Rani R. A study of big data evolution and research challenges. *Journal of information science* 2019; 45(3): 322–340. doi: 10.1177/0165551518789880
28. Kumar A, Sharma S, Goyal N, et al. Secure and energy-efficient smart building architecture with emerging technology IoT. *Computer Communications* 2021; 176: 207–217. doi: 10.1016/j.comcom.2021.06.003
29. Kumar CNS, Reddy KS. An experimental analysis of the applications of datamining methods on bigdata. *Journal of Autonomous Intelligence* 2019; 2(3): 30–38. doi: 10.32629/jai.v2i3.59