STM JOURNALS

Recent Trends in
Parallel Computing

ISSN: 2393-8749
Volume 11, Issue 3, 2024
September–December
DOI (Journal): 10.37591/RTPC

https://journals.stmjournals.com/rtpc

Review RTPC

Decoding Big Data: A Practical Comparison Between Hadoop and Spark

Saunved Palve[1,*], Ammar Abdulhussain[1], Om Sawant[1], Ayush Yadav[1], Aditya Kasar[2]

Abstract

This paper conducts a comprehensive comparison of Apache Hadoop and Apache Spark, two essential frameworks in the big data era. The rapid expansion of data possesses challenges in terms of volume, variety, and velocity, which necessitate advanced processing solutions. Hadoop, utilizing its MapReduce paradigm, provides scalable and fault-tolerant storage, whereas Spark, built upon Hadoop, introduces in-memory processing to increase speed and flexibility. This study includes a detailed examination of their features, strengths, and limitations, offering insights through experiments in statistical analysis, machine learning, and database operations. The work contributes valuable perspectives for both practitioners and researchers, enabling them to make informed decisions in the ever-changing landscape of big data analytics. Our experiments reveal that Apache Spark achieves a notable average speedup of 41.57% over Apache Hadoop in statistical and machine learning applications, underscoring its superiority in big data analytics. However, it is important to note that Hadoop excels in database management systems, demonstrating superior performance in particular scenarios.

Keywords: Hadoop, MapReduce, Spark, Sqoop, average word length, machine learning, database management (DBMS)

INTRODUCTION

Big data is another name for the rapid increase in information that has ushered in a new era in which possibilities are intertwined with difficulties. This vast amount of data can reveal new insights never known before, stimulate innovative thinking, and help change behavior [1]. Processing or analyzing it traditionally becomes difficult because of the volume, variety, and velocity at which big data arrives.

Several methods have been proposed for overcoming these challenges. First, data collected from social media sites such as Facebook and sensor networks that monitor physical conditions such as temperature or financial transactions daily cannot be easily managed by traditional systems. Second, a variety of data exists, such as structured databases to unstructured social media posts or images, making the standardization of management techniques more difficult. Finally, conventional methods cannot perform fast, continuous data processing with live data streaming and hence require real-time processing.

Addressing these challenges is vital for unlocking the full capabilities of big data. Thus, distributed computing platforms, such as Apache Hadoop and Apache Spark, play an important role by providing scalable and efficient storage solutions, as well as processing capacities for handling big data.

***Author for Correspondence**
Saunved Palve
E-mail: saunved.palve@gmail.com

[1]Student, Department of Computer Engineering, Shri Vile Parle Kelavani Mandal's Narsee Monjee Institute of Management Studies (SVKM'S NMIMS), Navi Mumbai, Maharashtra, India
[2]Assistant Professor, School of Technology, Shri Vile Parle Kelavani Mandal's Narsee Monjee Institute of Management Studies (SVKM'S NMIMS), Navi Mumbai, Maharashtra, India

Received Date: June 24, 2024
Accepted Date: August 08, 2024
Published Date: September 16, 2024

Citation: Saunved Palve, Ammar Abdulhussain, Om Sawant, Ayush Yadav, Aditya Kasar. Decoding Big Data: A Practical Comparison Between Hadoop and Spark. Recent Trends in Parallel Computing. 2024; 11(3): 15–23p.

Enter Apache Spark, a game changer in the world of big data. Developed in 2009 and opened in 2010, Spark is based on Hadoop and offers many key features. In terms of memory usage, Spark uses memory for intermediate data, significantly increasing the processing speed compared with traditional disk-based methods such as MapReduce in Hadoop. Spark uses engines such as Spark SQL for SQL-like queries, Spark Streaming pipelines that provide real-time data, and MLlib for machine learning (ML) algorithms, making it simple and versatile for various big data applications [3]. The ability of Spark to rapidly handle batch and real-time processing at an incredible speed and performance makes it a popular choice for big data analytics. Today, Hadoop and Spark often coexist, with Hadoop providing robust storage infrastructure and Spark providing the advanced techniques needed to deal with modern variety and demanding big data. This study aimed to provide a critical comparison of these two systems, focusing on their main characteristics, strengths, and limitations.

LITERATURE REVIEW

As a fundamental part of Hadoop, MapReduce is one of the major tools for big data processing worldwide and has been widely examined as a distributed programming framework [2]. It can be applied to various domains such as machine learning, stream processing, and file deduplication. Despite its shortcomings, MapReduce continues to be investigated further for specific use cases and security issues regarding how it can be improved for performance optimization.

This study focused on improvements aimed at enhancing performance and energy efficiency, as evidenced by Hong et al. [4], and concentrating on data organization in Hadoop. Karun and Chitharanjan [5] contrast the cluster setups, providing important pointers for the design of energy-efficient systems. Furthermore, Salloum et al. [6] presented a practical performance model based on relative computational complexity (RCC). In addition, Wang and Khan [7] examine In-Storage Computing (ISC) integration, showing that Hadoop can effectively optimize big data processing.

Turning to Spark, Shaikh et al. [8] addressed MLlib, Spark's machine learning library, along with its emphasis on speed and adaptability. Building on this, García-Gil et al. [9] provides a comprehensive overview of big data analytics using Spark. Finally, Joshi et al. [10] introduced a simulation-based workload predictive model that opens up more possibilities for optimizing I/O cost predictions. Aravinth et al. [11] presented the importance of Sqoop as an interface software that runs on the command line and is an important aspect in the transfer of data from relational databases to Hadoop. This means that it can perform both the import and export functions. Hence, Sqoop has become significant because it enables industries to handle big datasets efficiently by acting as a bridge between traditional database systems and Hadoop.

In a comparative analysis, Benlachmi et al. [12] Spark versus Apache Flink analysis focuses on machine learning aspects and shows where there is no clear understanding of what comparative advantage refers to. However, the reviews show that, despite the limitations of using these tools, it is possible to create real-time analytics and emerging technologies that are capable of handling large amounts of data with flexibility by Spark. The growing importance of Apache Hadoop compared with Apache Spark in the big data processing field is demonstrated by comparative study, as challenges still exist, and research continues.

COMPARISON BETWEEN HADOOP AND SPARK

This section discusses the disparities between Hadoop and Spark, which use frameworks for processing large amounts of data. We also discuss their attributes, architectures, and security concerns.

Hadoop

Hadoop, which was developed by Google, uses the Hadoop Distributed File System (HDFS) to store large datasets across clusters of standard hardware. This system can be easily expanded by adding extra

storage via more nodes in the cluster and ensures data access by duplicating values if a node goes down. The best part about it is that Hadoop uses commercial off-the-shelf hardware, inexpensive as compared to supercomputers. The MapReduce programming model in a Hadoop cluster organizes large dataset computations into parallel tasks, where each task is divided into small pieces that are processed simultaneously at different nodes. The "map" function changes data into key-value pairs followed by the Shuffle Phase where keys and values are reorganized and indexed to merge identical values. Then finally, "reduce" takes all those key-value pairs with similar keys and combines these pairs' corresponding values to come up with the final results.

Security

There is strict control over access to Hadoop's distributed storage system (HDFS) through the use of Access Control Lists (ACLs). For example, using ACLs, one can decide on who among users or groups is allowed to see what kind of data and do what, such as read, write, or execute. This extreme control helps prevent unauthorized entry and manipulation of data.

Moreover, Hadoop addresses data security in a very strong manner both at rest and during transit using encryption. Encryption encompasses the scrambling of information on storage equipment using cryptographic keys, which makes it unreadable to unauthorized people unless properly authorized. Similarly, the encryption of data in the transit safeguards information is passed from node to node within a cluster against interception that may be attempted.

Additionally, Hadoop YARN (Yet Another Resource Negotiator) employs resources and node managers to monitor the cluster resources. Some mechanisms like SASL (Simple Authentication and Security Layer), can be used for secure interaction between these constituents; hence, confidentiality and integrity of communication channels are ensured.

Spark

Spark is a framework designed for big data processing that prioritizes in-memory computations, thereby improving efficiency compared to MapReduce. It is particularly effective for handling iterative and interactive workloads for data. This memory approach by Spark reduces the disk I/O, resulting in faster execution times than Hadoop MapReduce. In addition, Spark provides flexibility by enabling complex data processing workflows beyond a simple two-stage MapReduce model. This allows the implementation of such iterative algorithms and transformations of data that are represented as a Directed Acyclic Graph (DAG). Moreover, Spark enables real-time processing by dealing with micro-batches of data, which makes near-real-time processing possible in streaming data pipelines. Distributed processing is performed using the master-worker structure of the Spark. Task execution coordination is managed by the Spark driver, which acts as the central coordinator, manages task dependencies in the DAG, and communicates with the worker nodes. In-memory computations handled by these processes on worker nodes are executed based on instructions from the driver node, where each task has been assigned to be executed and data may be moved between them, if needed, during processing. Communication between the Spark driver and executors occurs through a messaging system, where the driver assigns tasks to executors and executors report their progress or completion back to the driver.

Security

Spark's default security features are minimal, as it relies solely on shared secret passwords for authentication, making it susceptible to brute-force attacks. Furthermore, Spark does not have built-in authorization or encryption capabilities. However, when deployed on YARN in a secure Hadoop environment, Spark can benefit from security measures. It can leverage Kerberos authentication and HDFS access control in its applications. In addition, tools such as Apache Sentry can offer more precise authorization for Spark data access within the HDFS.

Table 1. Key differences between Apache Hadoop and Spark.

Feature	Hadoop MapReduce	Apache Spark
Developed at	Google	UC Berkeley
Processing Mode	Batch processing	Real-time and streaming processing
Written in	Java	Scala
Intermediate results	Stored on hard disk	Stored in-memory
Programming Language	Java, C, C++, Ruby, Groovy and Python	R, Scala, Java, Python, and SQL
Storage	Disk only	Disk or primary memory
Speed	Write and read from the disk (slower)	Write and read from memory (faster)
Real-time	Not suitable	Suitable
Machine Learning (ML)	No ML library	Has an inbuilt ML library "MLlib"

However, the standalone deployments of Spark require significant manual configurations to ensure security. This may involve implementing custom authentication mechanisms, integrating them with external security services, and potentially encrypting data, both at rest and during transit. Table 1 has been referenced from Benbrahim et al. [13] for key distinctions between Hadoop MapReduce and Apache Spark.

EXPERIMENTAL SETUP AND METHODOLOGY

This section provides an overview of the system configuration, framework configuration, and the various experiments performed.

Software and Hardware Configurations

The experiments were conducted in a controlled environment using a virtual machine (detailed specifications are listed in Tables 2 and 3). Open-source tools, such as Apache Hadoop and Spark, facilitate data processing and analysis machines (detailed specifications in Table 4).

Ubuntu 23.10 (64-bit) was chosen as the operating system because of its stability and compatibility with the selected software stack. The Oracle VM VirtualBox serves as the virtualization platform, facilitating the creation and management of the virtual machine environment. A data transfer tool was utilized to enable seamless data transfer between relational databases and the distributed file system used by Hadoop Sqoop.

Table 2. Hardware configurations.

AMD Ryzen 7 7840Hs w/ Radeon 780M Integrated GPU
16 GB 5200 MHz RAM (Memory)
NVIDIA GeForce RTX 4050 Laptop GPU

Table 3. Virtual machine configuration.

(Ubuntu 64-bit) version: Ubuntu 23.10
Base Memory (assigned) 12056MB
Processors 6 Cores

Table 4. Software configuration.

Java (OpenJDK) 11.0.22
Hadoop 3.3.6
Spark 3.5.0
Sqoop 1.4.7

METHODOLOGY AND RESULT

To achieve our work goals, we deployed Java code on both Apache Hadoop and Apache Spark. Within the Hadoop framework, we followed the MapReduce paradigm, whereas in Spark, we implemented the master-worker model. To store our data, we utilized the HDFS with a block size of 128 MB.

Our experimental methodology had three separate applications. The first focused on statistical analysis, which considered the average length of words starting with each letter of the alphabet. Next, we used a machine learning application (with simple linear regression). To make this comparison fair between the two systems, we did not use the MLlib library in Spark and performed a balanced appraisal. In our third application, we ran database management (DBMS) operations. In Hadoop, we used Sqoop for data transfer in our DBMS operation, whereas Spark operated independently without any support from it [14].

In addition, we conducted tests using three different datasets, varying both in size and content, for the average length of words, starting with each letter and the machine learning code. Our datasets were obtained from external repositories and were refined according to our experimental requirements. However, we slightly modified the DBMS code, as discussed in subsequent sections.

Most importantly, all these experiments were performed using a standalone machine. Upon executing each code, we timed its execution and used it during the evaluation process. These execution times formed our primary measurements for comparative analysis [15].

Average Length of Words Analysis

The purpose of this experiment was to compare the performance of Apache Hadoop and Apache Spark in computing the average word length, starting with each letter. The input consisted of three datasets: 1 million, 100,000, and 10,000 words, and the execution times for each dataset were measured and compared.

The findings in Tables 5 and 6 demonstrate that Spark consistently outperforms Hadoop in terms of execution time across all dataset sizes [16]. As the data size increases, the margin between the two frameworks also increases. For the 1-million-word dataset, Spark recorded an increase in speed of approximately 40%. This can be attributed to the use of memory processing capabilities by Spark, which provides many advantages over Hadoop's disk-based MapReduce paradigm, particularly for iterative and memory-intensive tasks, as shown in Figure 1.

These two implementations produced the same result numbers but differed in code structure and complexity. The MapReduce framework was used to write Hadoop code, necessitating the creation of separate mapper and reducer classes. This approach has often involved more verbose and boilerplate code than that used in Spark, where Java lambda expressions and functional programming constructs take advantage of making it concise and readable [17].

Table 5. Hadoop average word length.

File (words)	Size	Time (seconds)
10,000	75 KB	8s
100,000	804 KB	9s
1,000,000	9.90 MB	11s

Table 6. Spark average word length.

File (words)	Size	Time (seconds)
10,00	75 KB	5s
100,000	804 KB	6s
1,000,000	9.90 MB	8s

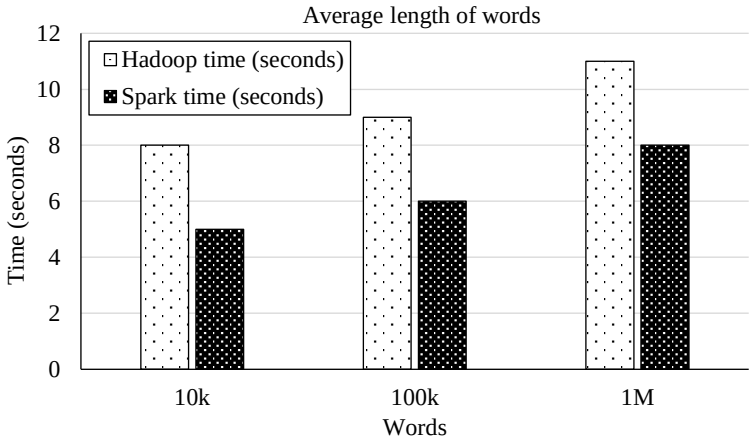

Figure 1. Time comparison of Hadoop and Spark on the average word length.

Machine Learning Application: Simple Linear Regression

The purpose of this experiment was to compare the performance of Apache Spark in implementing a simple linear regression model without using the MLlib library, to ensure a fair comparison with the MapReduce approach used in Hadoop. The execution times were evaluated on different datasets with sizes 1 kb, 96 kb, and 2.82 Mb as shown in Table 7.

Similar to the previous experiment, Spark performed better than Hadoop across all dataset sizes in terms of execution time as seen from the previous experiment. Although there was a small margin between them, Spark improved slightly over time, particularly when dealing with large datasets. This can be attributed to efficient in-memory processing capability and parallel calculations across multiple nodes. It is worth mentioning that the use of MLlib would further enhance this effect [18].

It should be noted that both implementations achieved the same computational result, which calculated the slope and y-intercept for the linear regression model, although their coding structures were significantly different, as shown in Figure 2. While the code for Hadoop applied MapReduce concept needed separate mapper and reducer classes to perform distributed computation However, this approach can be more intricate or wordy compared to writing Spark code (SparkSimpleLinearRegression.java). The Spark program uses functional programming concepts, such as lambda expressions, and decreases operations, resulting in a clearer and easier presentation.

Database Management System Operations

The objective of this test was to compare the use of Apache Spark and Sqoop-based Hadoop methodology for executing a simple "SELECT *" command in a MySQL database (Table 8). Spark proved to be a flexible and scalable solution, whereas Hadoop using the Sqoop approach had a better implementation time for this application. These performance differences can be attributed to several factors. First, data are transferred between the database and the Spark cluster in Spark, which can cause additional overhead when Sqoop fetches data directly from the database to HDFS. In addition, Sqoop can use a much better I/O performance time by moving large datasets between databases and HDFS with relative ease.

Table 7. Simple linear regression results.

Size	Hadoop time (seconds)	Spark time (seconds)
1 KB	9s	5s
96 KB	11s	5s
2.82 MB	15s	7s

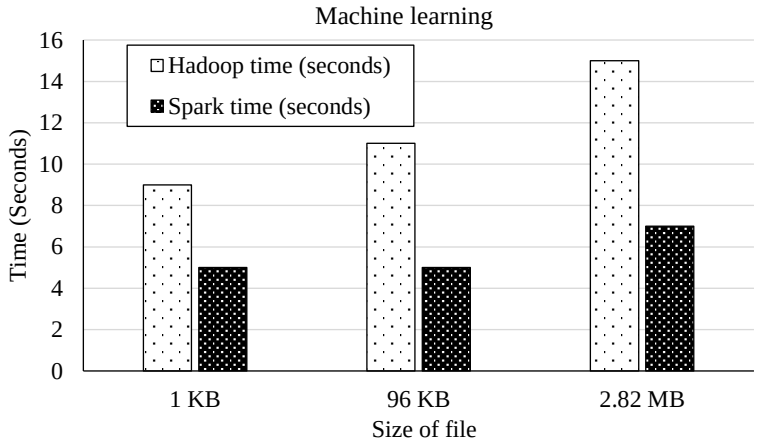

Figure 2. Time comparison of Hadoop and Spark on simple linear regression.

Table 8. DBMS results.

Hadoop	Spark
2.6 sec	15.5 sec

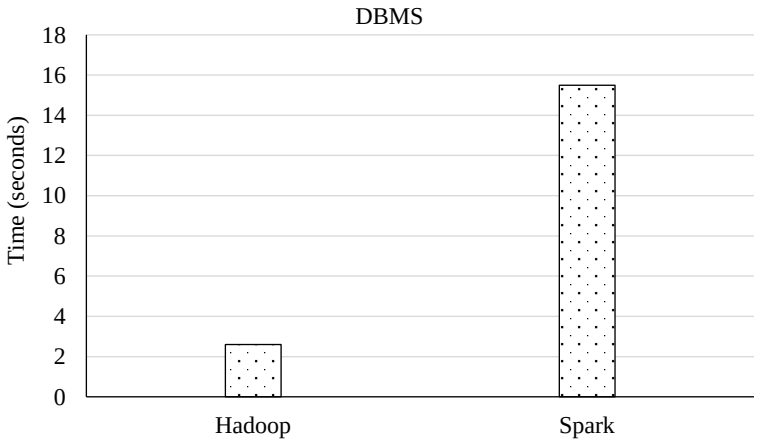

Figure 3. Time Comparison of Hadoop and Spark on DBMS Operation.

Regarding code design, Spark provides SQL functions that allow accessing databases, reading data in data frames, and displaying results. This approach abstracts high-level manipulation of data but can be a bottleneck compared to direct access to a database. Alternatively, Sqoop provides a tool that allows data to be transferred from traditional relational databases to HDFS. It should be noted that Sqoop provides connectivity to all databases and therefore provides direct interaction as well as flexibility to meet these specific functional requirements quickly, as shown in Figure 3.

CONCLUSION AND LIMITATIONS

Conclusion

This study demonstrates the changing dynamics of big data applications, focusing on the functioning of Apache Hadoop and Apache Spark. These applications provide solutions to meet the growing challenges of big data analysis, such as data volume, variety, and speed. Hadoop uses the new MapReduce method for reliable storage and fault tolerance management, whereas Spark builds upon the foundation of Hadoop, which introduces efficient in-memory processing to improve speed and modifications. The experiments conducted in this work include the average word length and machine learning, which show that Spark performs better than Hadoop, particularly when conducting memory-intensive tasks. By conducting this study, we can say that Spark's design, readability, and ease of writing

help increase its preference over Hadoop. However, it is necessary to consider the safety features, their maintenance, and their optimizations when choosing a suitable framework. This study provides key points for researchers working in the field of big data analytics by presenting an overview of the strengths and weaknesses of Hadoop and Spark. The limitations of this study open up a path for future research. Conducting more in-depth research on these frameworks would help advance big data-processing technology.

Limitations

Although this study provides key insights into the comparison between Apache Hadoop and Apache Spark, it is necessary to identify some limitations. First, it was performed on a standalone machine, which allowed controlled but possibly imprecise studies that may not accurately reflect large distributed systems. In addition, the analysis mainly focused on specific applications such as average word length analysis and simple linear regression for machine learning, which could overlook the full range of capabilities offered by these frameworks.

Acknowledgments

We thank Dr. Yogesh Jadhav and Prof. Preeti Godbole for their invaluable guidance and support throughout this study.

REFERENCES

1. Anjum B. MapReduce–The scalable distributed data processing solution. In: Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms. 2018. p. 173–90.
2. Naga Malleswari TYJ, Vadivu G. MapReduce: A technical review. Indian J Sci Technol. 2016;9:1–6. DOI: 10.17485/ijst/2016/v9i1/78964.
3. Feller E, Ramakrishnan L, Morin C. On the performance and energy efficiency of Hadoop deployment models. 2013 IEEE International Conference on Big Data, Silicon Valley, CA, USA. 2013. pp. 131–6. DOI: 10.1109/BigData.2013.6691564.
4. Hong Z, Xiao-Ming W, Jie C, Yan-Hong M, Yi-Rong G, Min W. An optimized model for MapReduce based on Hadoop. TELKOMNIKA. 2016;14:1552–8.
5. Karun AK, Chitharanjan K. A review on Hadoop—HDFS infrastructure extensions. 2013 IEEE Conference on Information & Communication Technologies, Thuckalay, India. 2013. pp. 132–7. DOI: 10.1109/CICT.2013.6558077.
6. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ. Big data analytics on Apache Spark. Int J Data Sci Anal. 2016;1:145–64. DOI: 10.1007/s41060-016-0027-9.
7. Wang K, Khan MMH. Performance prediction for Apache Spark platform. 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, New York, NY, USA. 2015. pp. 166–73. DOI: 10.1109/HPCC-CSS-ICESS.2015.246.
8. Shaikh E, Mohiuddin I, Alufaisan Y, Nahvi I. Apache Spark: A big data processing engine. 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), Manama, Bahrain. 2019. pp. 1–6. DOI: 10.1109/MENACOMM46666.2019.8988541.
9. García-Gil D, Ramírez-Gallego S, García S, Herrera F. A comparison on scalability for batch big data processing on Apache Spark and Apache Flink. Big Data Anal. 2017;2:1–11. DOI: 10.1186/s41044-016-0020-2.
10. Joshi A, Luo Y, John LK. Applying statistical sampling for fast and efficient simulation of commercial workloads. IEEE Trans Comput. 2007;56:1520–33. doi: 10.1109/TC.2007.70748.
11. Aravinth SS, Begam AH, Shanmugapriyaa S, Sowmya S, Arun E. An efficient HADOOP frameworks SQOOP and Ambari for big data processing. Int J Innov Res Sci Technol. 2015;1: 252–5.

12. Benlachmi Y, El Yazidi A, Hasnaoui ML. A comparative analysis of Hadoop and Spark frameworks using word count algorithm. Int J Adv Comput Sci Appl. 2021;12:778–88.
13. Benbrahim H, Hachimi H, Amine A. Comparison between Hadoop and Spark. Proceedings of the International Conference on Industrial Engineering and Operations Management; 2019 Mar 5-7; Bangkok, Thailand. IEOM Society International; 2019. p. 690–701.
14. Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, et al. Mllib: Machine learning in Apache Spark. J Mach Learn Res. 2016;17:1–7.
15. Shi J, Qiu Y, Minhas UF, Jiao L, Wang C, Reinwald B, Özcan F. Clash of the Titans: MapReduce vs. Spark for large scale data analytics. Proc VLDB Endow. 2015 Sep;8(13):2110–21. DOI: 10.14778/2831360.2831365.
16. Tekdogan T, Cakmak A. Benchmarking Apache Spark and Hadoop MapReduce on big data classification. Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing. New York, NY, USA: Association for Computing Machinery; 2021. p. 15–20. DOI: 10.1145/3481646.3481649.
17. Pavan DKU, Nachappa DMN, Srinivasu DSN. Sqoop usage in Hadoop distributed file system and observations to handle common errors. Int J Recent Technol Eng. 2020;9:452–4. DOI: 10.35940/ijrte.D4980.119420.
18. Verma A, Mansuri AH, Jain N. Big data management processing with Hadoop MapReduce and Spark technology: A comparison. 2016 Symposium on Colossal Data Analysis and Networking (CDAN), Indore, India. 2016. pp. 1–4. DOI: 10.1109/CDAN.2016.7570891.

