International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.php

# Hadoop Vs Spark: An Integrated Big Data Analytics Architecture For Distributed Computing Frameworks

**S R V Prasad Reddy[1], Dr.K.Parthiban[2], Rajendrakumar Ramadass[3], Kakarla Hari Kishore[4]**
[1]Assistant Professor, Department of CSE in Data Science, Dayananda Sagar Academy of Technology and Management, 22 Mile, B M Kaval, Udayapura, Kanakapura Road, Bangalore – 560082, India.
[2]Assistant Professor, Department of Computer Science, N.K.R.Government Arts College for Women Namakkal – 637001, India.
[3]Assisstant Trainer, Electrical Section, Engineering Department, College of Engineering & Technology, University of Technology and Applied Sciences - Shinas, Sultanate of Oman.
[4]Professor, Electronics and Communication Engineering Department, Koneru Lakshmaiah Education Foundation, Guntur, A.P, India.
E-mail: mtechprasadreddy@gmail.com[1], rkparthiban2013@gmail.com[2], rajendrakumar.ramadass@utas.edu.om[3], kakarla.harikishore@kluniversity.in[4]
[*]Corresponding author Email: rkparthiban2013@gmail.com

***Abstract***
*In the digital age, decision-makers may now access enormous amounts of data. The phrase "big data" refers to databases that are difficult to handle using traditional tools and techniques due to their size, diversity, and dynamic nature. The last few years have seen an increase in the generation of data from multiple sources due to the introduction of cloud computing technology. Today's data processing equipment must be able to handle the enormous volumes of newly created information. For the DS &BDA sector in SC & L, we first propose a novel method of systematic review in this research. In order to classify the existing models and approaches, arrange their areas of practical usage, identify research needs, and propose potential future research methods; we then use the recommended methodology for an organized literature review on DS &BDA methods in the SC &L fields. The Adoop framework rose to prominence with its dispersed file structure and MapReduce programming methodology. On the other hand, Spark is a newly created framework for big data management and analysis that allows you to investigate an infinite number of the underlying properties of large data. The research compares Hadoop MapReduce with Spark's operating principles, effectiveness, and expense, simplicity of use, connectivity, data mining, disaster tolerance, and hygiene. Experimental observations of Hadoop MapReduce and Spark's performance have been made to ascertain their suitability for use in a variety of distributed computing scenarios.*

***Keywords:*** *Hadoop MapReduce, Spark, decentralized frameworks, parallel computing, big data, and big data analysis.*

## 1. INTRODUCTION

The phrase "Big Data" refers to a group of information sets that are so big and complex that they are hard to handle using standard data mining tools and techniques. Finding useful insights in large datasets and transforming them into an understandable format for later use is the main goal of big data analytics.

Acquisition, organization, preservation, research, distribution, transmission, analysis, and display are the main processes that make up big data [1]. There has been a notable increase in the awareness of the importance of this specific field in recent years. Its capacity to give organizations useful information and improved understanding of both organized and unorganized data is primarily to blame for this. This could therefore lead to better decision-making procedures that are founded on a more thorough comprehension of the topic. Big data analytics, as used in the business world, is the methodical analysis of large datasets, sometimes known as "big data," with the aim of uncovering previously undiscovered connections, market trends, customer preferences, and other relevant business insights. The use of big data analytics by businesses has been made easier by the convergence of recent developments in data analytics algorithms and techniques with modern technology improvements. Using big data analytics efficiently presents a number of important issues.

**Apache Spark:** The potential and efficiency of distributed information processing frameworks have been thoroughly investigated in earlier studies. The research has illustrated Spark's capacity to effectively handle enormous databases and its backing for real-time data dispensation. The Apache master/worker design is depicted in the first figure below (Figure 1).

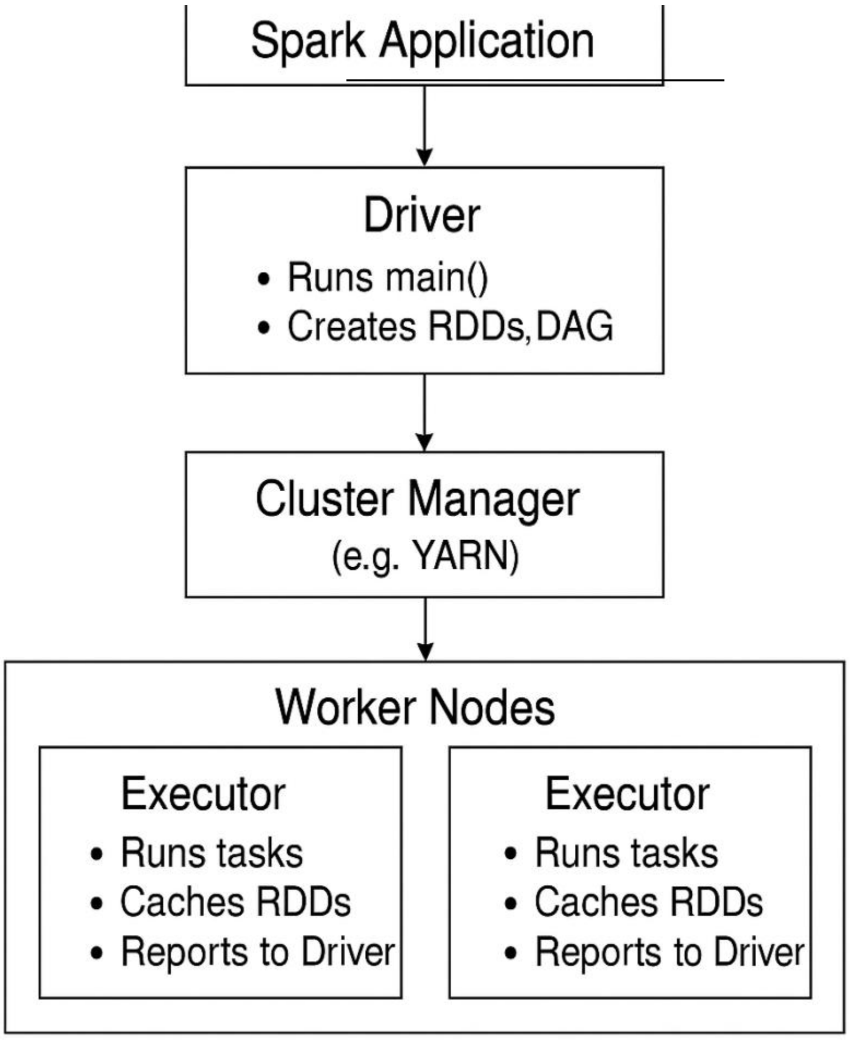

**Figure 1.** The planning of Apache Spark

It indicates that a single cluster management coordinator, which oversees several workers where executors operate, communicates with a driver program [2]. The executor can execute on different servers, which is an instance of vertical regrouping, or on the identical machine, also known as a horizontal clustered.

On the other hand, Apache Flink provides low processing latency, robust failure tolerance, and continuous data management as its top priority.

Numerous comparative analyses have assessed Flink's functionality and emphasized its benefits for managing real-time data streams and event-time processing; Flink architecture (Figure 2), which is constructed by the JobManager, Resource Manager, Distributor, and TaskManagers. At the same time, the JobManager manages checkpoint synchronization, job planning, and disaster recovery [3]. The ResourceManager manages the supply and allocation of resources in the Flink cluster [4]. Each JobMaster manages the way a specific JobGraph—a proxy for a Flink task—is executed.

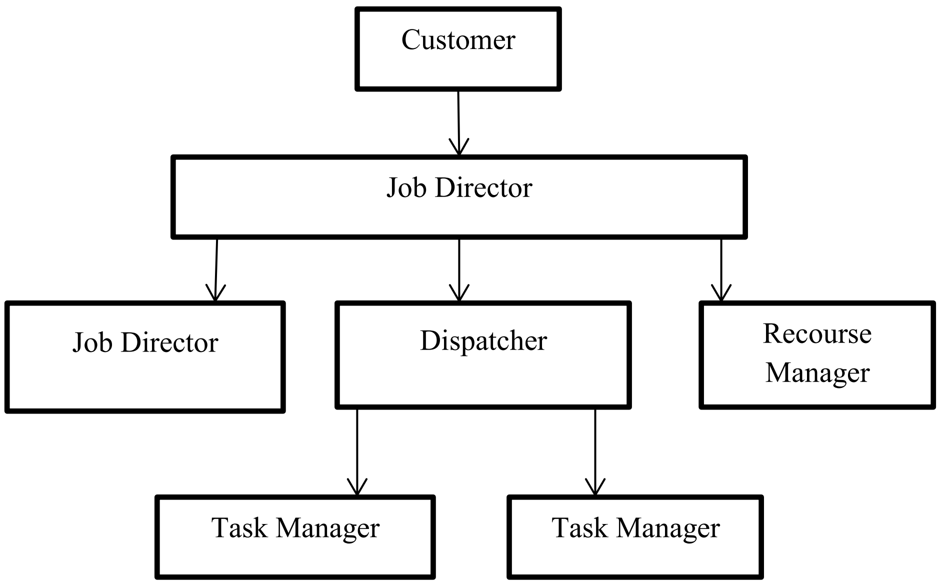

**Figure 2.** The architecture of Flink architecture

Similar to Hadoop, Spark is a programming framework; nevertheless, batch and rapid processing were considered when it was designed. Compared to Hadoop MapReduce, it operates in a fully distributed manner and is more capable of handling massive data issues thanks to its streaming API, which allows data processing to precede in short interval batches. Jobs in Spark can operate indefinitely until they are terminated by users or by any unrecoverable failure. Additionally, it can function in a solitary setting.

The remainder of the document is structured as follows: The literature is reviewed in Section 2, which also identifies the ways in which our work differs from previous research. The criteria for a distributed framework are covered in Section 3. The architectural details and needs of the frameworks are described in Section 4, which also talks about feature vectors and compares the frameworks in relation to those feature vectors. Lastly, we offer our conclusion in Section 5.

## 2. Related Works

The writers covered a variety of approaches to managing massive amounts of data, including bio-inspired computing, computing in the cloud, granular calculating, and quantum technology. Singh and Reddy [5] provided a comprehensive analysis of big data platforms, such as field programmable gate arrays (FPGAs), multicore CPUs, GPUs, peer-to-peer networks, Hadoop environment, and HPC clusters. A theoretical synopsis of Hadoop, Spark, and Tornado was also provided. The author evaluated the Spark and Storm platforms using five criteria: batch framework integration, fault tolerance, delays, computing model, and supported spoken languages. The main topics of discussion include open-source solutions for large data machine learning in the Hadoop ecosystem and some basic distinctions across stream processing systems. The writers talked about machine learning technologies and how well they work with MapReduce, Spark, Flink, Storm, and H2O. Several machine learning frameworks were assessed by them according to their extensibility, scalability, and usability.

Most methods begin with partitioning, which divides a big data set across several processing node locations, each of which works with the designated partitioned information. In a shared-nothing architecture, it is thought that every processing machine must perform the same set of tasks, despite certain variances in parallel processing. The majority of models integrate the results to create the final product after sending their partial output to the supervisor node for manufacturing [6]. Big data's three V's are essential components that led to the development of platforms and frameworks. The purpose of big data frameworks is to handle situations that are beyond the capabilities of conventional systems. The potential of big data tools and frameworks is particularly increased by the following characteristics: The first step in data distribution is to divide the data into smaller pieces and distribute them among the cluster's available nodes. In accordance with fault tolerance and shipping, the data is ready for parallel processing and stored in the distributed file systems (DFS).

A McKinsey analysis states that the Big Data era is upon us and will impact several industries, encompassing personal geographic information, sales, worldwide production, medical care, and public sector management. Distributed architectures for big data analytics in cloud computing include Hadoop, Spark, Apache Flink, Kafka, NoSQL databases like HBase, Cassandra, and MongoDB, packaged software like Docker, Kubernetes containerization technological advancements, server-less computing, cloud storage like Azure blob storage or S3, and managed analytics services like AWS EMR and Data Bricks for processing big data [7]. These platforms provide MLlib, RDDs, Spark SQL, job scheduling, streaming, distributed storage that is both scalable and fault-tolerant, and resource management. These technologies facilitate the management of enormous volumes of data and offer effective solutions for data handling and storing them.

Many studies have contrasted the Hadoop MapReduce and Apache Spark large data processing methods. They discovered that Spark performed noticeably better than Hadoop as far as of processing time and resource usage [8]. Detecting fall across a Hadoop/Spark cluster using deep recurrent neural networks (RNNs) demonstrates that deep RNNs with Spark's in-memory computation were quicker than Hadoop disk-based processing. This implies that while Hadoop MapReduce remains important at large scale for repetitive processing, Apache Spark is better suited for iterative machine learning workloads. Numerous individuals have investigated several frameworks and contrasted them based on the effectiveness, scalability, and performance of the large datasets.

Every CPU in a shared-memory system has access to the same memory region. Because data may be passed directly across processors without requiring explicit interaction, this architecture makes programming easier. In these systems, threads are usually used to coordinate parallel execution, frequently with the aid of frameworks such as OpenMP [9]. However, there are drawbacks to shared-memory systems as well, especially with regard to synchronicity. Inadequate coordination can result in data races or inconsistent outcomes when several threads try to write and read data to the same memory region at the same time. Developers must use barriers, locking mechanisms, or other synchronization methods to lessen these problems. Distributed-memory systems, however, employ a different strategy. Each processor in these systems has its local memory, and message passing is how processors interact with one another.

In general, there are three main architectural models for parallel computing systems: shared-memory, distributed-memory, and hybrid systems. Every CPU in a shared-memory system has access to the same memory region. Because data may be passed directly across processors without requiring explicit communication, this architecture makes programming easier. In these kinds of systems, threads are usually used to control parallel execution, frequently with the aid of frameworks like OpenMP [10]. However, there are drawbacks to shared-memory systems as well, especially with regard to synchronization. Inadequate coordination can result in data wars or inconsistent outcomes when several threads try to read and write to the same memory region at the same time. Designers must use barriers, locking mechanisms, or other synchronizing methods to lessen these problems.

## 3. Methods and Materials

### 3.1 Overview of Apache Spark

In this first section, we give an overview of the Apache Spark program and its main elements. We list some of the key characteristics that make Apache Spark a next-generation big data processing engine, following Hadoop's MapReduce [11]. We also provide an overview of a few industry contributions and case stories.

#### 3.1.1 The Apache Spark initiative

In 2013 [14], the Apache Software Foundation received the project as a donation. A number of research initiatives have significantly aided in the development and enhancement of Spark core and the primary upper-level libraries. For instance, the MLbase6 project served as the foundation for Spark's MLlib development, after which other projects began to contribute. Spark SQL began as part of the Shark project and later developed into a crucial Apache Spark module. Additionally [12], GraphX began as an AMPLab research project. Later, starting with version 0.9.0, it was incorporated into the Apache Spark project. Furthermore, a significant number of packages for Apache Spark have been contributed by both industry and academia. Furthermore, the individuals that developed Apache Spark also launched Databricks, a company that is closely involved in its creation.

#### 3.1.2 From Apache Spark to Hadoop's MapReduce

For huge data analytics, Apache Spark has emerged as the de facto standard, following Hadoop's MapReduce [13]. It combines an advanced programming environment for memory computation with a networked computing core machine as a platform. It has the same linear throughput and fault tolerance properties as MapReduce, but unlike the rigid map-then-reduce disk-based design, it has a multistage in-memory processing approach. With such an advanced model, Apache Spark is much faster and easier to use. It has strong APIs in multiple programming languages, including as Scala, Java, Python, SQL, and R, for carrying out complex distributed operations on scattered data. Additionally, Apache Spark significantly outperforms Hadoop's MapReduce by leveraging the memory of a computing cluster to reduce need on the supporting distributed file system. Additionally, it is thought of as a general-purpose machine that performs a variety of calculations and transcends batch operations.

#### 3.1.3 The industry's use of Apache Spark

Businesses from a variety of industries have quickly adopted Apache Spark since its original releases. Given Apache Spark's promise as a unified processing engine that interfaces with numerous storage systems, like as HDFS, Cassandra, HBase, and S3 [14], have produced dozens of functional community-contributed packages using Apache Spark. Many top-tier businesses, including IBM, have used Apache Spark as a core processor. There are numerous examples of Apache Spark being used for a variety of applications, such as Eyeview's planning and optimization of video advertising campaigns, Toyota Motor Sales' real-time classification and prioritization of social media interactions, NBC Universal's prediction of digital media offlining, and ING Banking's real-time anomaly detection.

### 3.2 The Apache Spark stack

Spark core and upper-level libraries are two of Apache Spark's primary components (Figure 3). Spark Core can retrieve data from any Hadoop data source and operates on many cluster managers. In addition, many packages have been created to work with both the higher-level libraries and Spark core. For a comprehensive rundown of the various structures and mediums, including Apache Spark, see the big data architecture.

#### 3.2.1 Spark core

Spark core serves as the foundation for Apache Spark. For managing large datasets, it provides the RDD API, a simple programming interface. Spark has R, Java, Python, and Scala APIs, even though its core is created in Scala. These APIs offer a wide range of activities (such transformations of information and movements) required for algorithm development in the higher-level components. Additionally, handling memory, job timetables, data hopping about, and fault recovery are among the primary features that Spark core provides for in-memory cluster computing. These characteristics enable the creation of a Spark application using the CPU, memory, and storage capabilities of an equipment cluster.

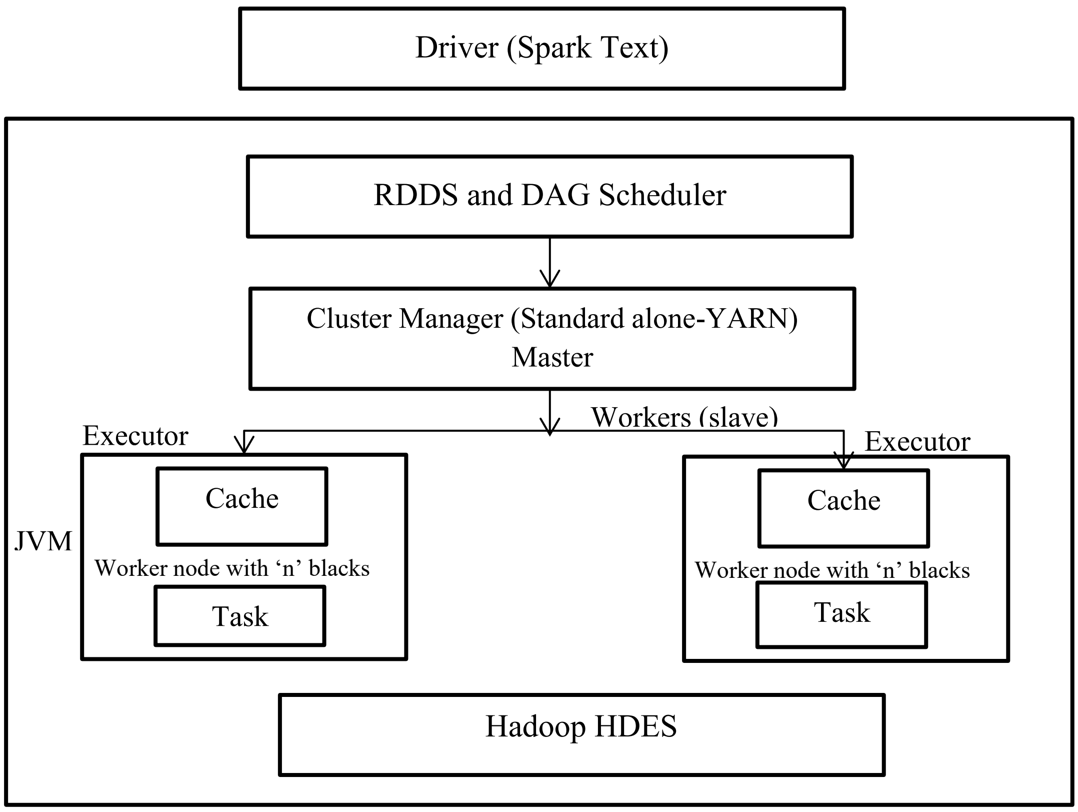

**Figure 3.** High-level Apache Spark stack architecture

**3.2.3 Spark submissions**

Five essential components are needed to run a Spark applicationa cluster supervisor, a driver program, staff, administrators, and procedures (Figure 4). A driver program is an application that uses Spark as a library to define a high-level control mechanism of the target computation. While a worker provides the program with CPU, memory, and storage space, an executer is a JVM (Java Virtual Machine) activity that Spark generates on each worker for a Spark application [15]. A job is a set of computations (like a data processing technique) that Spark does on the cluster and returns the results to the driver program. A Spark program can launch several jobs. A job is divided by Spark into a DAG, or directed acyclic graph, of stages, each of which consists of a group of tasks. The smallest workstation that Spark provides to an executor is called a task [16]. A SparkContext is the primary gateway to Spark's functionality, allowing the driver software to access Spark [17]. A connection to a computational cluster is represented by a Spark Context.

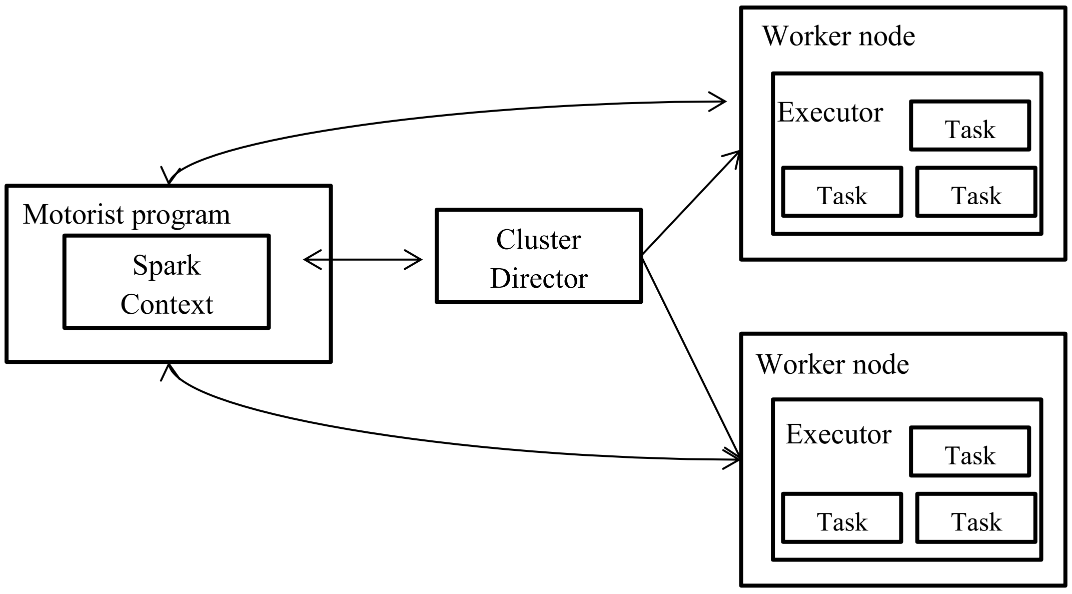

**Figure 4.** Essential components for executing a Spark application

**3.3 A Comparison between Spark and Hadoop**

Large volumes of massive datasets can be stored across clusters thanks to Hadoop, a well-known and useful open-source software framework that enables shared storage. The architecture of the system makes it possible to move from a single server to many nodes with ease [18]. Hadoop can process large data sets concurrently, yielding efficient outcomes. The two primary components of Hadoop are MapReduce and the Hadoop Distributed File System.

Files are distributed among multiple nodes for storage by the HDFS, which divides them into smaller sections called blocks. Data nodes, also called worker nodes, and name nodes, also called master nodes, are the two different types of nodes that make up the HDFS. These two different types of nodes are necessary for all actions, including typing, reading and removal. The following is a description of the HDFS workflow:

The name-node first asks for permission to access. The file name will be changed to a complete list of HDFS block identities if the proposal is accepted. The files themselves and the particular datanodes in charge of keeping the blocks connected to each file will be included in this compilation. The client will then receive the list of identifying (ID), allowing users to carry out further actions depending on the information they have obtained.

Two essential tasks are included in the MapReduce computation system: translators and decreases. The mappers will use the mapping function to analyze the files and convert them into distinct key-value pairs. After the key-value pairs are created, they are categorized by key in descending sequence and allocated to various partitions. A combiner, which is optional, can be thought of as a local reduction operation that makes it easier to pre-reduce data with matching keys, hence lessening the strain on input/output processes. The final key-value pairs will then be sent to a reducer after being divided into separate segments. A single action called shuffle must be included in order to build MapReduce.

"Move" is the process of transferring the mapper-generated data to the appropriate reducer. When the shuffling process is complete, the reducer initiates a series of replica threads known as fetchers. These threads are in charge of obtaining the map job's output files via the HTTP protocol; Consolidating the produced output into separate final files, which are then designated as the input variables for the reduction, is the next step. The output is then sent back to the Hadoop Distributed File System once the reducer has processed the data in accordance with the minimized function.

## 4. Implementation and Experimental Results

### 4.1 Datasets

Sentiment or opinion analysis of item-review data, which may be found online at sites like Amazon's product assessment site, indicates how a person feels and feels about a product. Since Amazon reviews are the most often used dataset for interactive attitude analysis, three distinct datasets concerning them have been employed in this work. Customers can ask and answer inquiries, share their own thoughts, and leave comments on any good on the website for Amazon Reviews [19]. This project will not be limited to concentrating only on the topic of a particular review. The first dataset is Amazon reviews per Kindle shop category; the second is Amazon reviews for emotional analysis; and the third is internet data: Amazon movie ratings. Additional details regarding the datasets are given in Table 1.

**Table 1. Features of the datasets along with their initial dimensions**

| No. | Appellation of the dataset | Size | No. of fields (attributes) | No. of rows |
|---|---|---|---|---|
| 1. | Amazon's reviews: category for Kindle stores | 674 MB | Eight | 865,754 |
| 2. | Sentiment analysis of Amazon reviews | 1.5GB | Three | 1,877,877 |
| 3. | Web data: movie reviews on Amazon | 7.99GB | Nine | 12,645,121 |
| 4. | Complete size | 11.01GB | One | 7,548,752 |

### 4.2 The architecture of the proposed system

Additionally, Hadoop and Spark are among the big data tools that have been deployed on the guest operating system. In distributed databases like Spark and Hadoop, input is processed in parallel by three nodes: a master and two slaves. Spark must then be linked with Hadoop in order to read and write data to and from HDFS. This combination will improve data processing abilities [20]. In other words, Hadoop stores the data across multiple nodes so that it can be viewed by all of the Spark nodes, and which process the data concurrently.

Table 2 shows the parameters of the datasets after the initial three steps of data preprocessing: choosing features, data scrubbing-1 [21], and data integrating.

**Table 2. Features of the datasets following the initial three data preparation processes**

| No. | Name of the dataset | Size | No. of grounds | No. of rows |
|---|---|---|---|---|
| 1. | Amazon reviews: category for Kindle stores | 582 MB | One | 846,987 |
| 2. | Sentiment analysis of comments on Amazon | 1.52GB | One | 3,964,864 |
| 3. | Web data: movie reviews on Amazon | 6.265GB | One | 7,085,942 |

| | | | | |
|---|---|---|---|---|
| 4. | files for aggregation (three datasets combined) | 8.743 | One | 13,033,643 |

### 4.3 Applications

This method was constructed in a very efficient way, making it straightforward to use to assess information of any size. The final usable dataset size was reduced from 13.9 GB to 8.3 GB after the first two phases of pre-processing and compilation. The implementation is separated into two methods: distributed data processing (data processing over several nodes) and central information processing (single node data handling).

Lastly, evaluating the developed model using the 20% of data that remains, and obtaining both good and negative outcomes.

### 4.4 Analysis and Results

Accuracy, recall, accuracy, f-measure, and execution time have all been calculated for each algorithm using the two aforementioned methods.

#### 4.4.1 Data processing at one place

Table 3 shows the outcomes of each of the aforementioned metrics along with an analysis of the algorithms. Using logistical regression and SVM classifiers, binary sentiment evaluation produced very good Naïve Bayes accuracy results and an excellent accuracy rate on training data. These findings suggest that the pre-processing procedures used had a major beneficial effect on both execution velocity and classification precision.

**Table 3. Results of central data dispensation**

| Classification algorithm | Correctness | Exactness | Reminiscence | F-meas. | Exec. time in min. |
|---|---|---|---|---|---|
| LR | 92.1% | 89.1% | 90.01% | 91.3% | 175.23 |
| SVM | 92.6% | 90.2% | 80.5% | 88.9% | 276.3 |
| NB | 81.5% | 93.5% | 85.9% | 88.2% | 234.86 |

#### 4.4.2 Processing data in a distributed manner

There have been some changes made to the datasets. As mentioned earlier, this alteration process has resulted in three different types of data: type (A), type (B), and type (C). In order to determine whether the system is more effective while working with a single huge dataset file or multiple datasets of the same size, this manipulation technique aims to exceed all other types in terms of accuracy and performance.

The results indicate that, as seen in Figure 5, all of the data types that were used produced results that were roughly equal.

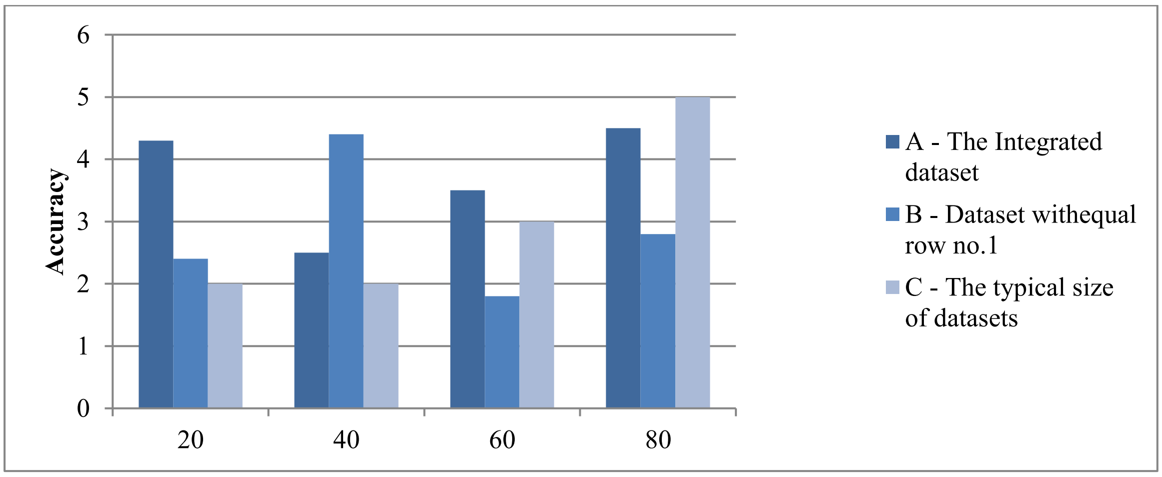

**Figure 5.** Comparison of the accuracy of the data types used in approach two

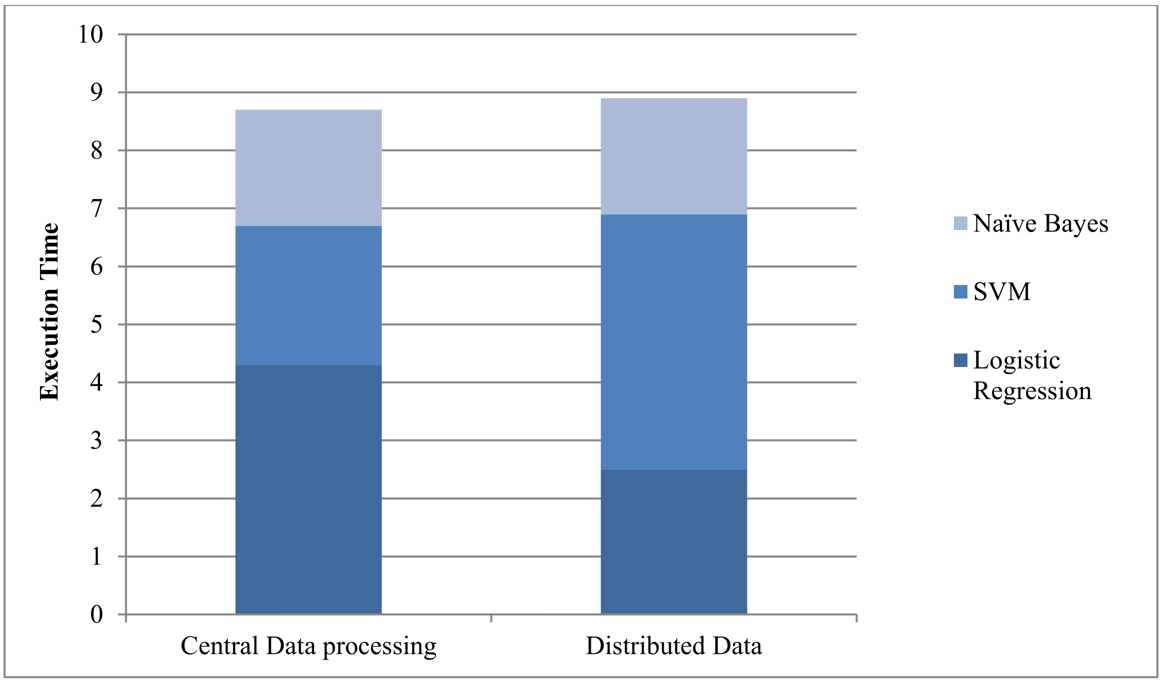

**Figure 6.** Disparities in execution times between the methods

Following the implementation of both central and distributed data processing strategies, a few quick comparisons between them are now required in order to determine which is the most effective. As shown in Figure 6, the experimental data indicates that the distributed system technique reduces the learning durations for all classifiers by almost half in comparison to the central approach.

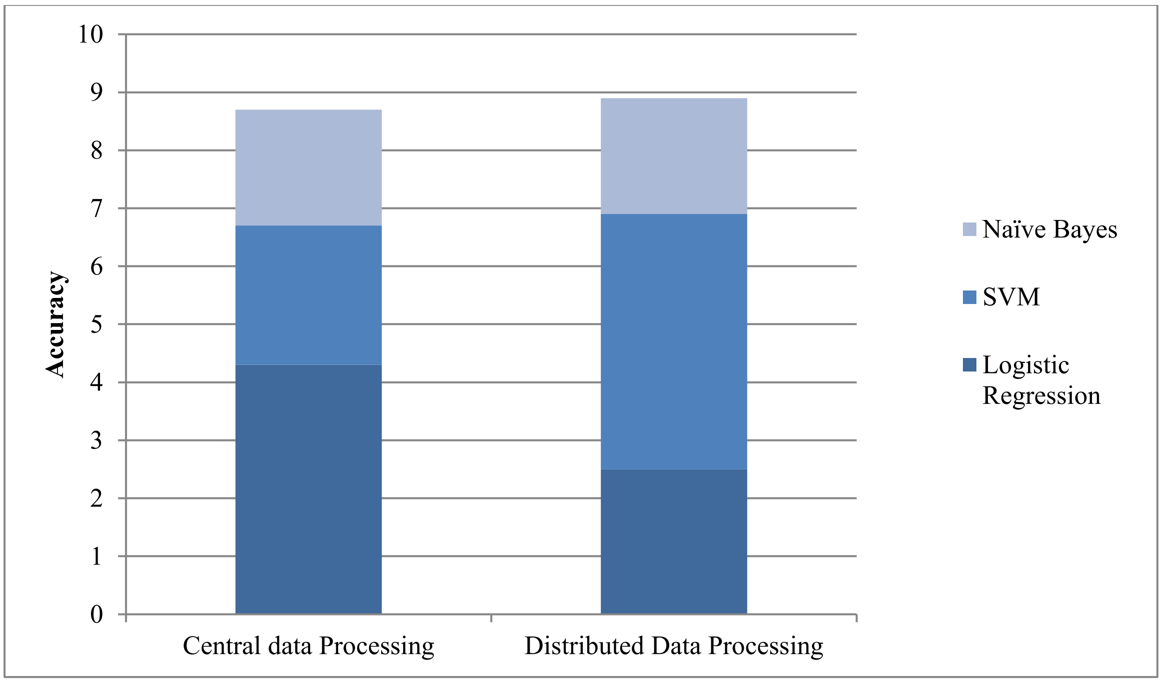

**Figure 7.** Accuracy findings for the methods

However, if one closely examines Figure 7, which shows the accuracy and F-measure rates, it is evident that the SVM algorithm and the logistic regression approach achieve the best accuracy and F-measure percentages in both centered and distributive ways. The Naïve Bayes method yielded the worst results compared to the other machine learning techniques, as seen by its lowest efficiency and F-measure frequencies.

## 5. CONCLUSION

The research presented here contrasts Hadoop with Spark according to their relative working theories, efficiency, expense, usability, interoperability, information mining, failure transparency, and security. To determine Hadoop and Spark's applicability under various distributed computing environment restrictions, experimental study has also been conducted to examine their performance.

This system uses both distributed information processing (multi-node) and central data processing (one-node). With the same accuracy ratio, the distributed processing strategy performed better than the central processing approach and cut the model construction execution time in half when compared to the central approach for all used algorithms.

Java and Scala are the programming languages used to write the system programs, and the built model includes the preparatory processes in an ML Pipeline phases figure along with the classification methods. Three different algorithm types were employed, and the results showed that the Logistic Regression model performed better than the Naïve Bayes and SVM classifiers in both approaches. The end results showed that the system could process huge volumes of data quickly, especially when the dispersed information processing technique was used. However, it can attain very good accuracy depending on the many phases of data pre-processing that are applied. Additionally, the system tests demonstrated that the techniques used for dataset adjustments, such as integrated information processing (one file) or segmented dataset processing (many files), had no effect on the accuracy and time efficiency measurements.

## REFERENCES

1. Ibtisum, S., Bazgir, E., Rahman, S. A., & Hossain, S. S. (2023). A comparative analysis of big data processing paradigms: MapReduce vs. Apache Spark. World Journal of Advanced Research and Reviews, 20(1), 1089–1098.
2. Gimaletdinova, A. (2024). An in-depth comparative study of distributed data processing frameworks: Apache Spark, Apache Flink, and Hadoop MapReduce. Вестник науки, 3(4), 364–377.
3. Yang, C. T., Chen, T. Y., Kristiani, E., & Wu, S. F. (2021). The implementation of data storage and analytics platform for big data lake of electricity usage with Spark. The Journal of Supercomputing, 77(6), 5934–5959.
4. Tekdogan, T., & Cakmak, A. (2021, August). Benchmarking Apache Spark and Hadoop MapReduce on big data classification. In Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing (pp. 15–20).
5. Khalid, M., & Yousaf, M. M. (2021). A comparative analysis of big data frameworks: An adoption perspective. Applied Sciences, 11(22), 11033.
6. Sultan, N. A., & Abdullaha, D. B. (2023). Comprehensive study on big data frameworks. International Journal of Applied Sciences and Technology, 5(1), 35–48.
7. Al-Atroshi, C., & Zeebaree, S. R. (2024). Distributed architectures for big data analytics in cloud computing: A review of data-intensive computing paradigm. The Indonesian Journal of Computer Science, 13(2).
8. Manikandan, K., Pamisett, V., Challa, S. R., Komaragiri, V. B., Challa, K., & Chava, K. (2025). Scalability and efficiency in distributed big data architectures: A comparative study. Metallurgical and Materials Engineering, 31(3), 40–49.
9. Singh, S., Alam, M. N., Kaur, B., Kaur, K., Kaur, S., & Hossain, S. (2025, February). Comparative analysis of Apache Hadoop and Apache Spark for business intelligence. In AIP Conference Proceedings (Vol. 3224, No. 1). AIP Publishing.
10. Mirza, N. M., Ali, A., & Ishak, M. K. (2024). The scheduling techniques in the Hadoop and Spark of smart cities environment: A systematic review. Bulletin of Electrical Engineering and Informatics, 13(1), 453–464.
11. Ramkumar, M. P., Reddy, P. B., Thirukrishna, J. T., & Vidyadhari, C. (2022). Intrusion detection in big data using hybrid feature fusion and optimization enabled deep learning based on Spark architecture. Computers & Security, 116, 102668.
12. Singh, D., & Gar, R. (2021). Hadoop integration for big data analytics. In Proceedings of 3rd International Conference on Computing Informatics and Networks: ICCIN 2020 (pp. 13–22). Springer Singapore.
13. Muvva, S. (2023). Optimizing Spark data pipelines: A comprehensive study of techniques for enhancing performance and efficiency in big data processing. Journal of Artificial Intelligence, Machine Learning and Data Science, 1(4), 1862–1865.
14. Shrotriya, L., Sharma, K., Parashar, D., Mishra, K., Rawat, S. S., & Pagare, H. (2023). Apache Spark in healthcare: Advancing data-driven innovations and better patient care. International Journal of Advanced Computer Science and Applications, 14(6).
15. Kekevi, U., & Aydın, A. A. (2022). Real-time big data processing and analytics: Concepts, technologies, and domains. Computer Science, 7(2), 111–123.
16. Azeem, M., Abualsoud, B. M., & Priyadarshana, D. (2023). Mobile big data analytics using deep learning and Apache Spark. Mesopotamian Journal of Big Data, 16–28.
17. Ahmad, L. G. (2024). Spark, Hadoop, and beyond: Exploring distributed frameworks for metaheuristic optimization in big data analytics.
18. Sundarakumar, M. R., Mahadevan, G., Natchadalingam, R., Karthikeyan, G., Ashok, J., Manoharan, J. S., & Velmurugadass, P. (2023). A comprehensive study and review of tuning the performance on database scalability in big data analytics. Journal of Intelligent & Fuzzy Systems, 44(3), 5231–5255.
19. Ma, C., Zhao, M., & Zhao, Y. (2023). An overview of Hadoop applications in transportation big data. Journal of Traffic and Transportation Engineering (English Edition), 10(5), 900–917.

20. Podhoranyi, M. (2021). A comprehensive social media data processing and analytics architecture by using big data platforms: A case study of Twitter flood-risk messages. Earth Science Informatics, 14(2), 913–929.
21. Yousfi, S., Rhanoui, M., & Chiadmi, D. (2021). Towards a generic multimodal architecture for batch and streaming big data integration. arXiv preprint arXiv:2108.04343.
22. Muralidharan, J. (2024). Optimization techniques for energy-efficient RF power amplifiers in wireless communication systems. SCCTS Journal of Embedded Systems Design and Applications, 1(1), 1–6. https://doi.org/10.31838/ESA/01.01.01
23. Velliangiri, A. (2024). Security challenges and solutions in IoT-based wireless sensor networks. Journal of Wireless Sensor Networks and IoT, 1(1), 8–14. https://doi.org/10.31838/WSNIOT/01.01.02
24. Rahim, R. (2024). Optimizing reconfigurable architectures for enhanced performance in computing. SCCTS Transactions on Reconfigurable Computing, 1(1), 11–15. https://doi.org/10.31838/RCC/01.01.03
25. Danh, N. T. (2025). Advanced geotechnical engineering techniques. Innovative Reviews in Engineering and Science, 2(1), 22–33. https://doi.org/10.31838/INES/02.01.03
26. Karthika, J. (2025). Sparse signal recovery via reinforcement-learned basis selection in wireless sensor networks. National Journal of Signal and Image Processing, 1(1), 44–51.