Jurnal Informasi dan Teknologi

https://jidt.org/jidt

2025 Vol. 7 No. 4 Hal: 1-6 e-ISSN: 2714-9730

Analysis of Apache Hadoop Architecture in Supporting Large-Scale Data Processing

TNM Dhuha[1]✉, Asrianda[2], Muhammad Fikry[3]

[1,2,3]Department of Information Technology, Faculty of Engineering, Universitas Malikussaleh, Indonesia

teuku.257110201013@mhs.unimal.ac.id

Abstract

The rapid development of information technology has led to the exponential growth of data generated from various sectors, such as healthcare services, social media, information systems, and other digital activities. This condition has given rise to the concept of big data, which cannot be optimally processed using conventional data processing technologies. Therefore, distributed computing platforms are required to efficiently handle large-scale data storage and processing. Apache Hadoop is one of the widely used big data technologies due to its distributed architecture that supports scalability, parallel processing, and fault tolerance. This study aims to analyze the architecture of Apache Hadoop and explain the role of each of its components in supporting large-scale data processing. The research method employed is a qualitative literature study, conducted through the review of books, scientific articles, and related publications on Hadoop. The results indicate that Hadoop consists of three main components: the Hadoop Distributed File System as a distributed storage system, MapReduce as a programming model for parallel data processing, and Yet Another Resource Negotiator, which functions in cluster resource management and scheduling. The integration of these components enables Hadoop to manage large-scale data in a reliable and distributed manner. However, Hadoop has limitations related to its batch-based processing model, which is less suitable for real-time processing needs, thus requiring consideration of complementary technologies according to application requirements.

Keywords: *Big Data, Apache Hadoop, Hadoop Distributed File System, Mapreduce, Resource Management.*

1. Introduction

The rapid advancement of technology has led to a significant increase in data generation, both consciously and unconsciously, by various parties. This data is produced across multiple sectors, including healthcare services, social media, device and server activity logs, and financial markets, with an extremely high growth rate. The accumulation of data from these diverse sources results in datasets characterized by large volume, high velocity, and high variety, commonly referred to as big data [1].

Big data represents an abstract concept that refers to large-scale and complex data integration that cannot be efficiently stored, processed, or analyzed using conventional data management tools. It is commonly characterized by the five Vs: volume, which reflects the massive scale of data generation; velocity, which emphasizes the need for timely data collection and analysis; variety, which includes structured, semi-structured, and unstructured data such as text, images, audio, and video; veracity, which relates to data accuracy and reliability; and value, which highlights the potential insights and benefits that can be extracted from data despite the typically low value density of raw data [2][3]. These characteristics distinguish big data from traditional datasets and underscore the increasing complexity of modern data environments.

Big data processing is a complex task and cannot be treated in the same manner as small-scale data processing. As data volume, velocity, and complexity continue to increase, traditional data processing methods, models, and technologies become insufficient due to limitations in performance, storage capacity, and processing capabilities. Conventional analytical approaches are often unable to adapt to the scale and heterogeneity of big data, thereby limiting their effectiveness in supporting data-driven decision making. This condition necessitates the development of new analytical models, theories, and high-performance computing platforms capable of supporting large-scale data storage, processing, and analysis in complex and dynamic environments [4][5].

Apache Hadoop is one of the technologies developed to address these challenges. Hadoop is a system that integrates distributed file storage and distributed computational processing, operating on cloud server infrastructure. The distributed file storage component in Hadoop is known as the Hadoop Distributed File System (HDFS), while distributed computation is carried out using the MapReduce framework [6]. Hadoop enables efficient data processing by leveraging distributed computing resources, thereby enhancing system performance and scalability.

Receipt: 26-05-2025 | Revision: 29-10-2025 | Publish: 30-12-2025 | doi: 10.60083/jidt.vi0.711

Hadoop is designed to process large-scale data through three core components: HDFS, MapReduce, and Yet Another Resource Negotiator (YARN). HDFS functions as a reliable and scalable distributed storage system, MapReduce provides a batch-oriented parallel data processing model, and YARN is responsible for cluster resource management [7]. MapReduce offers a high level of abstraction by allowing users to define parallel processing tasks without managing low-level details such as data distribution, load balancing, and fault tolerance. In this model, input data are divided into multiple segments processed concurrently during the map phase, and intermediate results are aggregated during the reduce phase to produce the final output [8]. This abstraction enables MapReduce to efficiently process large volumes of data in a distributed environment while maintaining system reliability.

Although Hadoop has been widely implemented, a comprehensive understanding of its architecture and the role of each component in supporting large-scale data processing remains limited. Most previous studies focus primarily on Hadoop implementations, while in-depth analyses of Hadoop's architecture and the contributions of its individual components are still relatively scarce. Therefore, this study aims to analyze the architecture of Apache Hadoop and explain how each component contributes to supporting large-scale data processing. The results of this analysis are expected to provide a clearer understanding of Hadoop's role as a big data technology and serve as a reference for future development and utilization of large-scale data processing technologies.

2. Research Methods

This study employs a qualitative research method with a literature review approach. This method was selected because it aligns with the research objective of analyzing the architecture of Apache Hadoop and the role of each of its components in supporting large-scale data processing, based on theoretical studies and previous research, without conducting direct experiments or field data collection.

3. Results and Discussion

3.1. Apache Hadoop Architecture

Apache Hadoop is a distributed framework designed to handle large-scale data processing by utilizing a collection of computers known as a cluster. As illustrated in Figure 1, the Hadoop cluster architecture consists of a MasterNode and multiple WorkerNodes. The MasterNode is responsible for running the NameNode and ResourceManager components, while the WorkerNodes execute the DataNode, NodeManager, and MapReduce processes. The NameNode and DataNode function as managers of the HDFS storage system, whereas the ResourceManager and NodeManager are responsible for resource allocation and distribution to ensure efficient cluster utilization [9].

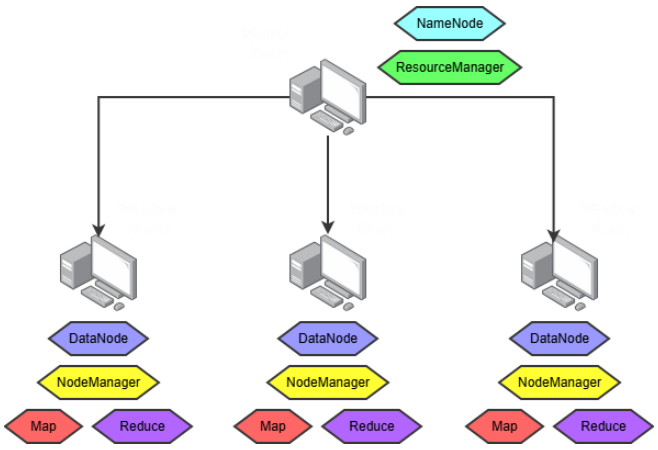

Figure 1. Hadoop Cluster Architecture

The division of roles between the MasterNode and WorkerNodes enables Hadoop to operate in a distributed and coordinated manner. This separation of functions allows data storage and processing to be performed in parallel, preventing workload concentration on a single node. Consequently, this architecture directly contributes to improved performance and scalability in handling large-scale data. Furthermore, Hadoop's architecture is designed to overcome the limitations of single-machine computing systems that are unable to efficiently manage massive data volumes. In a Hadoop cluster, data is stored using HDFS, while data processing is carried out through the MapReduce programming model [10].

In practice, Hadoop can be deployed in three modes: standalone mode, which runs within a single Java process; pseudo-distributed mode, which operates on a single machine using multiple separate daemons; and fully

distributed or cluster mode, which utilizes multiple machines within a cluster. In cluster mode, one node acts as the NameNode that manages metadata and file system namespaces, while the remaining nodes function as DataNodes and TaskTrackers to store data and execute MapReduce processes [11].

3.2. The Role of HDFS in Supporting Large-Scale Data Storage

Hadoop supports large-scale data storage through the Hadoop Distributed File System (HDFS). HDFS is developed using Java to provide reliable and consistent storage for massive datasets while enabling high-throughput data access. Data is stored in blocks that are replicated and distributed across multiple nodes within the cluster. This transparent data distribution mechanism simplifies storage management and reduces administrative overhead in big data systems [12].

The replication mechanism in HDFS also provides fault tolerance by maintaining multiple copies of data blocks to anticipate hardware or storage failures. In addition, HDFS supports NameNode failure handling through the use of a Secondary NameNode or by backing up metadata to multiple file systems [13]. As shown in Figure 2, the NameNode manages file system metadata, including block locations and file structure, while DataNodes store the data blocks physically. This separation of responsibilities allows HDFS to efficiently manage large-scale data without overburdening a single node. These characteristics make HDFS highly suitable for supporting large-scale data processing while maintaining performance and storage reliability.

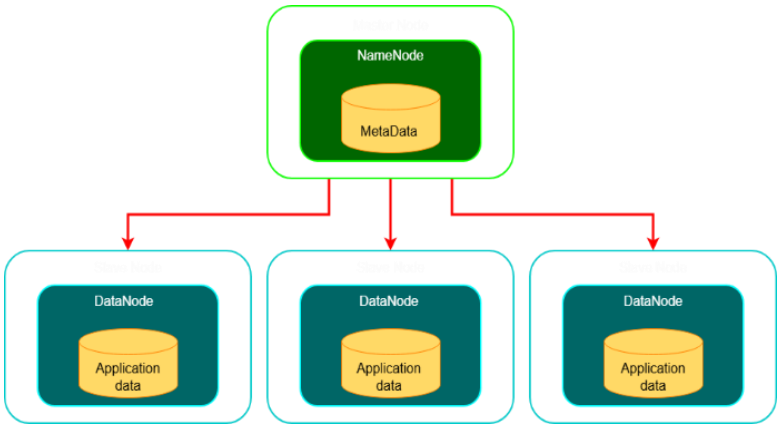

Figure 2. HDFS Components

Furthermore, HDFS applies the concept of data locality by assigning processing tasks to nodes where the corresponding data blocks are stored. This strategy reduces data transfer across the network and enhances processing performance in large-scale distributed systems [14].

3.3. The Role of MapReduce in Distributed Data Processing

MapReduce is one of the most widely used data processing models for Big Data applications. Hadoop represents the industry-standard implementation of the MapReduce framework and provides a robust platform for handling data-intensive applications such as web crawling, data mining, and web indexing. MapReduce also supports distributed and parallel processing of large-scale structured and unstructured datasets stored in HDFS, enabling efficient computation across clusters of commodity hardware [15].

One of the key strengths of Hadoop MapReduce is its built-in fault-tolerance mechanism, which enables reliable execution in large-scale distributed environments. MapReduce allowing failed or slow-running tasks to be automatically re-executed without affecting the overall job execution. Intermediate computation results are persistently stored on disk, enabling the system to recover and resume processing from the most recent successful execution state in the event of node or task failures, making MapReduce well suited for batch-oriented processing of massive datasets on clusters composed of commodity hardware [16]. Parallel processing also enables system recovery from partial server failures during execution, and as MapReduce clusters have become increasingly widespread, task scheduling has emerged as a critical factor in determining system performance [17].

The MapReduce data processing workflow in Apache Hadoop is illustrated in Figure 3. The process begins with input data being split into smaller chunks for parallel processing. During the mapping phase, each data chunk is processed into key–value pairs. These results then enter the shuffling phase, where data is grouped based on identical keys before being forwarded to the reducing phase. Finally, in the reducing phase, grouped data is processed to generate the final output, which is stored as the processing result. This workflow demonstrates how MapReduce enables efficient and distributed processing of large-scale data. Nevertheless, MapReduce is primarily designed for batch-oriented workloads, making it less suitable for low-latency or real-time data processing scenarios.

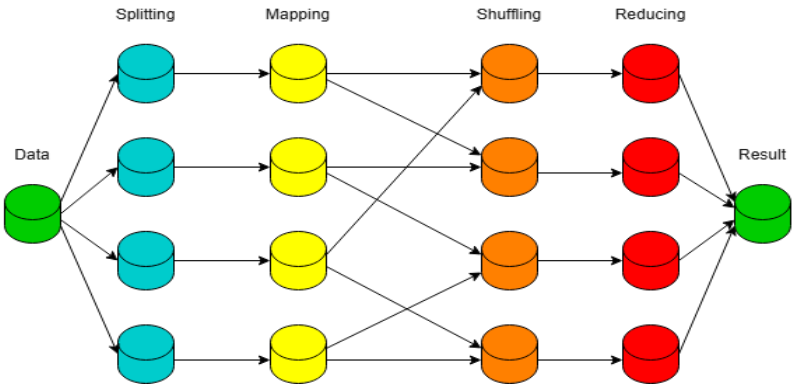

Figure 3. MapReduce Process

3.4. The Role of YARN in Resource Management and Scheduling

YARN is a core component of Hadoop that manages resources and schedules applications within a distributed computing cluster. Its primary function is to allocate resources such as CPU and memory efficiently among applications running on Hadoop. YARN, also known as MapReduce 2, was developed to address scalability limitations in the first-generation MapReduce, particularly in very large clusters. YARN enhances Hadoop's ability to manage resources and support the concurrent execution of diverse applications [18].

Figure 4 illustrates the YARN architecture and its operational processes. The process begins when a client submits an application execution request to the ResourceManager, which consists of a Scheduler and an Application Manager. The Scheduler allocates containers based on available cluster resources, while the Application Manager oversees the application lifecycle. NodeManagers running on each node manage local resources and monitor container execution. Applications run within containers, including the ApplicationMaster, which coordinates task execution and communicates with the ResourceManager to request additional resources. This integration enables YARN to support parallel application execution while improving Hadoop's scalability and overall performance.

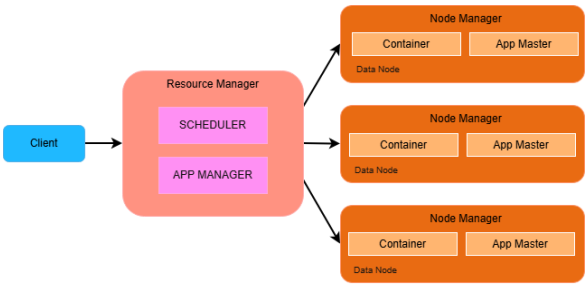

Figure 4. YARN Architecture

Beyond scalability, YARN significantly improves resource utilization efficiency within Hadoop clusters through its flexible resource allocation mechanism, which reduces resource wastage commonly found in traditional MapReduce systems. Although challenges related to centralized resource management remain, YARN continues to be a fundamental component for large-scale data processing due to its ability to enhance performance, flexibility, and support for multiple concurrent applications [18]. Recent studies further demonstrate that YARN can be deployed in container-based and serverless-oriented environments to enable dynamic and application-specific resource management. By providing customized YARN clusters for individual applications or users, this approach improves resource isolation, scalability, and overall system performance compared to conventional centralized YARN deployments, reinforcing YARN's relevance in modern Big Data architectures that require elastic resource allocation and efficient multi-tenant support [19].

3.5. Limitations of Apache Hadoop

Apache Hadoop offers substantial advantages in large-scale data processing through its distributed architecture, which supports high scalability, parallel processing, and fault tolerance. This technology enables efficient storage and processing of massive datasets using clusters of commodity hardware. However, Hadoop also has limitations, particularly due to its batch-oriented processing model, which results in high latency and makes it less suitable for real-time applications. In addition, Hadoop MapReduce relies heavily on disk-based input/output operations, where intermediate results are repeatedly written to and read from disk during the map and reduce phases. This disk-intensive processing introduces significant performance overhead, especially for large-scale and iterative workloads. Furthermore, Hadoop performance is highly dependent on configuration parameters, and suboptimal

default settings can lead to inefficient resource utilization and increased execution time as data volume grows, thereby increasing system complexity and operational overhead [20].

As data processing demands continue to evolve, alternative technologies such as Apache Spark have emerged. Apache Spark is an open-source framework designed for large-scale data processing, with its primary advantage being in-memory computation. Spark utilizes the Resilient Distributed Dataset (RDD) concept, which allows intermediate results to be stored in memory, thereby significantly improving processing speed and efficiency, particularly for iterative and real-time analytics [21].

4. Conclusion

This study analyzed the architecture of Apache Hadoop and the roles of its core components in supporting large-scale data processing. Apache Hadoop provides a distributed framework that enables scalable, fault-tolerant, and parallel processing of massive datasets using clusters of commodity hardware. Its architecture effectively addresses the challenges of modern data environments by integrating distributed storage, parallel computation, and centralized resource management.

The results show that HDFS ensures reliable and scalable data storage through replication and data locality, MapReduce supports fault-tolerant batch-oriented parallel processing, and YARN enhances resource utilization by enabling flexible and concurrent application execution. The integration of these components allows Hadoop to process large volumes of data efficiently in distributed environments.

However, Apache Hadoop is limited by its batch-based and disk-intensive processing model, which leads to higher latency and reduces its suitability for real-time and iterative workloads. Despite these limitations, Hadoop remains effective for large-scale batch analytics and data-intensive applications where reliability and scalability are prioritized. Future research may explore hybrid architectures or integration with other big data frameworks to improve processing flexibility and performance.

References

[1] R. Rawat and R. Yadav, "Big data: Big data analysis, issues and challenges and technologies," IOP Conference Series: Materials Science and Engineering, vol. 1022, no. 1, p. 012014, 2021, doi: 10.1088/1757-899X/1022/1/012014.

[2] K. Batko and A. Ślęzak, "The use of big data analytics in healthcare," Journal of Big Data, vol. 9, no. 3, 2022, doi: 10.1186/s40537-021-00553-4.

[3] J. Yang, Y. Li, Q. Liu, et al., "Brief introduction of medical database and data mining technology in big data era," Journal of Evidence-Based Medicine, vol. 13, pp. 57–69, 2020, doi: 10.1111/jebm.12373.

[4] J. Wang, Y. Yang, T. Wang, R. Sherratt, and J. Zhang, "Big data service architecture: A survey," Journal of Internet Technology, vol. 21, no. 2, pp. 393–405, 2020. [Online]. Available: https://jit.ndhu.edu.tw/article/view/2261/2274

[5] T. Lyu, P. Wang, Y. Gao, and Y. Wang, "Research on the big data of traditional taxi and online car-hailing: A systematic review," Journal of Traffic and Transportation Engineering (English Edition), vol. 8, no. 1, pp. 1–34, 2021, doi: 10.1016/j.jtte.2021.01.001.

[6] G. Karya and V. S. Moertini, "Exploration of Hadoop big data technology for community-based application systems," Jurnal Rekayasa Sistem dan Teknologi Informasi, vol. 1, no. 2, 2017, doi: 10.29207/resti.v1i2.65.

[7] S. R. Julakanti, N. S. K. Sattiraju, and R. Julakanti, "Creating high-performance data workflows with Hadoop components," NeuroQuantology, vol. 19, no. 11, 2021, doi: 10.48047/nq.2021.19.11.NQ21326.

[8] S. Hedayati, N. Maleki, T. Olsson, et al., "MapReduce scheduling algorithms in Hadoop: A systematic study," Journal of Cloud Computing, vol. 12, p. 143, 2023, doi: 10.1186/s13677-023-00520-9.

[9] F. D. Utami and F. D. Astuti, "Comparison of Hadoop MapReduce and Apache Spark in big data processing with Hgrid247-DE," Journal of Applied Informatics and Computing, vol. 8, no. 2, pp. 390–399, 2024, doi: 10.30871/jaic.v8i2.8557.

[10] N. H. Wicaksana, F. X. Arunanto, and H. Studiawan, "Implementation of transfer rate management in SDN-based HDFS processes," Jurnal Teknik ITS, vol. 5, no. 2, pp. 576–579, 2016, doi: 10.12962/j23373539.v5i2.18976.

[11] S. Petrova and S. Ivanov, "Integration of a distributed Hadoop system into the infrastructure of a technology startup company," Izvestia Journal of the Union of Scientists – Varna. Economic Sciences Series, vol. 9, no. 2, pp. 76–84, 2020, doi: 10.36997/IJUSV-ESS/2020.9.2.76.

[12] O. Azeroual and R. Fabre, "Processing big data with Apache Hadoop in the current challenging era of COVID-19," Big Data and Cognitive Computing, vol. 5, no. 1, p. 12, 2021, doi: 10.3390/bdcc5010012.

[13] S. Landset, T. M. Khoshgoftaar, A. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the Hadoop ecosystem," Journal of Big Data, vol. 2, 2015, doi: 10.1186/s40537-015-0032-1.

[14] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in Proc. IEEE 26th Symp. on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 2010, pp. 1–10, doi: 10.1109/MSST.2010.5496972.

[15] M. Saadoon, S. H. A. Hamid, H. Sofian, et al., "Experimental analysis in Hadoop MapReduce: A closer look at fault detection and recovery techniques," Sensors, vol. 21, no. 11, p. 3799, 2021, doi: 10.3390/s21113799.

[16] S. Ketu, P. K. Mishra, and S. Agarwal, "Performance analysis of distributed computing frameworks for big data analytics: Hadoop vs Spark," Computación y Sistemas, vol. 24, no. 2, pp. 669–686, 2020, doi: 10.13053/cys-24-2-3401.

[17] L. Thomas and R. Syama, "Survey on MapReduce scheduling algorithms," International Journal of Computer Applications, vol. 95, no. 23, pp. 9–13, 2014, doi: 10.5120/16733-6903.

[18] M. Timothy and O. J. Abiodun, "A fault-tolerance model for Hadoop rack-aware resource management system," Journal of Computer Science and Engineering (JCSE), vol. 4, no. 1, pp. 15–24, 2023.

[19] Ó. Castellanos-Rodríguez, R. R. Expósito, J. Enes, G. L. Taboada, and J. Touriño, "Serverless-like platform for container-based YARN clusters," Future Generation Computer Systems, vol. 155, pp. 256–271, 2024, doi: 10.1016/j.future.2024.02.013.

[20] N. Ahmed, A. L. C. Barczak, T. Susnjak, et al., "A comprehensive performance analysis of Apache Hadoop and Apache Spark for large-scale data sets using HiBench," Journal of Big Data, vol. 7, p. 110, 2020, doi: 10.1186/s40537-020-00388-5.

[21] R. Guo, Y. Zhao, Q. Zou, X. Fang, and S. Peng, "Bioinformatics applications on Apache Spark," GigaScience, vol. 7, no. 8, p. giy098, 2018, doi: 10.1093/gigascience/giy098.