EAI Endorsed Transactions
on Scalable Information Systems
Research Article EAI.EU

The Cutting-Edge Hadoop Distributed File System: Unleashing Optimal Performance

Anish Gupta[1], P. Santhiya[2], C. Thiyagarajan[3], Anurag Gupta[4], Manish Kr. Gupta[5*], Rajendra Kr. Dwivedi[6]

[1]Department of Computer Science & Engineering, Chandigarh Engineering College, Chandigarh Group of Colleges, Jhanjeri, Mohali, Punjab, India

[2]Department of Computer Science & Engineering, School of Computing, Sathyabama Institute of Science and Technology Semmancheri, Chennai, Tamilnadu, India

[3]Department of Artificial Intelligence and Machine Learning, Panimalar Engineering College Chennai, Tamilnadu, India

[4]Department of Computer Science & Engineering, ABESEC, Ghaziabad, Uttar Pradesh, India

[5]Department of Computer Science & Engineering, Madan Mohan Malaviya University of Technology, Gorakhpur, UP, India

[6]Department of Information Technology & Computer Application, Madan Mohan Malaviya University of Technology, Gorakhpur, UP, India

Abstract

Despite the widespread adoption of 1000-node Hadoop clusters by the end of 2022, Hadoop implementation still encounters various challenges. As a vital software paradigm for managing big data, Hadoop relies on the Hadoop Distributed File System (HDFS), a distributed file system designed to handle data replication for fault tolerance. This technique involves duplicating data across multiple DataNodes (DN) to ensure data reliability and availability. While data replication is effective, it suffers from inefficiencies due to its reliance on a single-pipelined paradigm, leading to time wastage. To tackle this limitation and optimize HDFS performance, a novel approach is proposed, utilizing multiple pipelines for data block transfers instead of a single pipeline. Additionally, the proposed approach incorporates dynamic reliability evaluation, wherein each DN updates its reliability value after each round and sends this information to the NameNode (NN). The NN then sorts the DN based on their reliability values. When a client requests to upload a data block, the NN responds with a list of high-reliability DN, ensuring high-performance data transfer. This proposed approach has been fully implemented and tested through rigorous experiments. The results reveal significant improvements in HDFS write operations, providing a promising solution to overcome the challenges associated with traditional HDFS implementations. By leveraging multiple pipelines and dynamic reliability assessment, this approach enhances the overall performance and responsiveness of Hadoop's distributed file system.

Keywords: Hadoop, HDFS, DataNode, NameNode, Write operation, Read operation

Received on 04 April 2025, accepted on 09 October 2025, published on 13 October 2025

doi: 10.4108/eetsis.9027

*Corresponding author. Email: mkgcse@mmmut.ac.in

1. Introduction

In the digital age, the proliferation of data has reached unprecedented levels, creating an information landscape of colossal proportions known as “Big Data.” The explosion of data generated from various sources, presents both tremendous opportunities and significant challenges. Effectively harnessing and extracting valuable insights from these massive datasets have become a pivotal focus for

researchers, industry professionals, and policymakers alike. In recent years, Hadoop has emerged as a dominant software paradigm for handling big data in large-scale distributed environments. With the ever-increasing volume of data, companies have been adopting 1000-node Hadoop clusters to tackle the challenges of data management and analysis. The foundation of Hadoop lies in its distributed file system, known as the Hadoop Distributed File System (HDFS), which ensures fault tolerance through data replication across multiple DNs, thus achieving reliability and availability. DFS is a client/server architecture that enables clients to access data stored on servers. In this architecture, DFS software is deployed on both the servers and clients, facilitating the connection of files located on various file servers into a unified Namespace. The primary goal of DFS is to enhance data availability by providing a seamless and transparent experience for clients.

In the rapidly evolving landscape of data management, Distributed File Systems (DFSs) have emerged as critical paradigms that enable efficient and seamless access to data across distributed environments. These innovative systems bridge the gap between clients and servers, providing a unified Namespace and enhancing data availability for users. In this context, Table 1 presents a comprehensive overview of various types of DFSs, each designed to address specific challenges and requirements in the field of data distribution and access. One of these notable DFSs is the Network File System (NFS) [1]. The NFS protocol became publicly available for use by other vendors. Another significant DFS is the Andrew File System (AFS), as presented by Howard [2]. Inspired by Carnegie Mellon University, AFS was specifically designed to cater to distributed workstation environments, providing seamless data access and management. The Google File System (GFS) takes inspiration from Ghemawat et al.'s work [3]. Comprising a single master and multiple segment servers, GFS efficiently handles requests from multiple clients. With the rise of object-oriented computing environments, Martini et al. developed XtreemFS, a file system specifically tailored for Grid environments [4]. XtreemFS offers enhanced capabilities for managing data in complex and distributed Grid settings. Each of these DFSs brings unique features and advantages to the table, catering to diverse data management needs in today's complex computing landscape.

Table 1: Different types of Distributed File System

Features	NFS	AFS	GPS	XFS	HDFS
Reliability	--	H	H	H	H
Flexibility	--	L	H	H	H
Scalability	L	H	H	H	H
Transparency	--	L	H	H	H
Security	H	H	L	L	L

H= High, L= Low

Despite the widespread implementation of Hadoop and HDFS, several challenges persist in their utilization, including concerns related to security, fault tolerance, and flexibility. One prominent issue is the time-consuming nature of the data replication technique due to its reliance on a single-pipelined paradigm. This limitation can hinder the overall performance and responsiveness of HDFS write operations.

To address these challenges and optimize HDFS performance, this paper proposes a novel approach employing multiple pipelines for data block transfers, rather than a single pipeline. Additionally, the proposed approach introduces dynamic reliability evaluation, where each DN updates its reliability value after each round and communicates this information to the NN. The NN then sorts the DNs based on their reliability values, allowing for efficient data management.

1.1. Motivation and Research Problems

With the exponential growth of big data, organizations increasingly rely on the Hadoop Distributed File System (HDFS) to manage large-scale distributed storage efficiently. However, despite its widespread adoption, HDFS still faces performance bottlenecks during data replication and write operations due to its traditional single-pipeline mechanism. This sequential data transfer model leads to increased latency, reduced throughput, and inefficient utilization of available network bandwidth. Additionally, the random selection of DataNodes (DNs) during replication often overlooks their varying reliability and performance, resulting in potential data transfer failures and uneven load distribution. Therefore, there is a strong motivation to enhance the efficiency and reliability of HDFS by introducing a multi-pipeline architecture and a dynamic reliability evaluation mechanism. The key research problems addressed in this study include optimizing the data replication process to minimize execution time, improving throughput without compromising fault tolerance, and developing an adaptive reliability model that ensures consistent performance in heterogeneous Hadoop environments.

2. Background

This section provide a detail background on big data, HDFS, read operation in HDFS and write operation in HDFS

2.1. Big Data

Big data is a term that refers to the massive volume of structured and unstructured data generated by various sources, which cannot be effectively managed and processed using traditional data processing techniques [5] It has become a driving force behind modern-day technological advancements and decision-making processes across various domains. The concept of big data encompasses the three V's: volume, velocity, and variety [6] Volume refers to the sheer amount of data generated daily, velocity denotes the speed at which data is produced and must be processed, and variety represents the diverse formats and types of data. Big data has opened up new avenues for research, particularly in data-driven fields like artificial intelligence, machine learning, and predictive analytics [7]. The ability to harness vast amounts of data has led to significant breakthroughs in areas such as natural language processing, computer vision, and personalized medicine. In addition to its applications in research and technology, big data has also transformed industries such as healthcare, finance, and marketing. By analyzing large datasets, organizations can gain valuable insights into customer behavior, market trends, and investment opportunities [8]. However, with the potential benefits of big data come significant challenges, including data privacy, security, and scalability [9]. Researchers and practitioners must address these issues to ensure the responsible and ethical use of big data. In conclusion, big data continues to reshape the world by providing opportunities for innovative research and driving transformative changes in various sectors. Its potential remains vast, and ongoing research efforts aim to address the challenges associated with its implementation and usage.

2.2. HDFS

HDFS is a crucial component of the Apache Hadoop ecosystem, designed to store and manage large-scale distributed data across a cluster of commodity hardware. It is designed for handling big data efficiently, providing fault tolerance, scalability, and high throughput for data-intensive applications [10]. HDFS operates on a master/slave architecture, consisting of a single NN that acts as the central metadata repository and multiple DNs that store the actual data. The data is divided into blocks, typically 128 MB or 256 MB in size, which are replicated across different DNs to ensure fault tolerance. One of the primary goals of HDFS is to facilitate data locality, wherein data processing tasks are executed on the nodes where the data is stored. This approach minimizes data movement across the network, leading to improved performance and reduced latency [11]. HDFS is widely adopted in various industries for its ability to handle vast amounts of data efficiently. It has been utilized in large-scale data-intensive applications, including web search engines, social media platforms, and log processing systems [12]. The fault tolerance mechanism of HDFS is achieved through data replication, where each block is replicated across multiple DNs. If a DN fails, the NN automatically instructs other DNs to replicate the lost data, ensuring data integrity and continuous availability [13]. Although HDFS offers several benefits, it also faces certain challenges, such as handling small files and supporting real-time data processing. Efforts have been made to address these limitations through optimizations and the integration of complementary technologies.

Table 2: Abbreviation used

Abbreviation	Full Form	Abbreviation	Full Form
reli	reliability	adpt	Adapt
mxreli	maxReliability	mnreli	minReliability
DNS	DataNodeStatus	RDNS	Repair DataNodeStatus

2.3. Read operation in HDFS

HDFS read operation involves retrieving data from the distributed storage system. The process of reading data from HDFS involves communication between the client and the NN to determine the DNs containing the required data. As mentioned in [14], to ensure security and authentication, the NN issues a token to the client, which allows the client to access and read data from the DNs. The process of reading data from HDFS, as described in the given steps and in Figure 1, involves the client's interaction with the NN and the DN. The steps can be summarized as follows:

1. **Client Request:** The client communicates with the NN by using the API, requesting the location of the required data block.

2. **Privilege Check:** The NN checks the client's privilege to access the requested data, verifying if the client is an authorized user.

3. **Authorization Response:** If the client is authorized, the NN responds by providing the data location and a security token for authentication purposes. If the client is not authorized, the NN rejects the request.

4. **DN Permission:** After receiving the data location and security token, the client communicates directly with the DN where the required block is stored. The DN checks the security token to ensure the client's authentication and provides the client with permission to read the required data block.

5. **Data Reading:** With the necessary permissions granted, the client opens an input stream and starts reading the data directly from the DN.

6. **DN Failure Handling:** In the event of a DN failure (e.g., due to a hardware issue), the client returns to the NN to obtain an alternative location for the same data block. The NN provides the client with a new location, allowing the client to resume reading the data from another available DN.

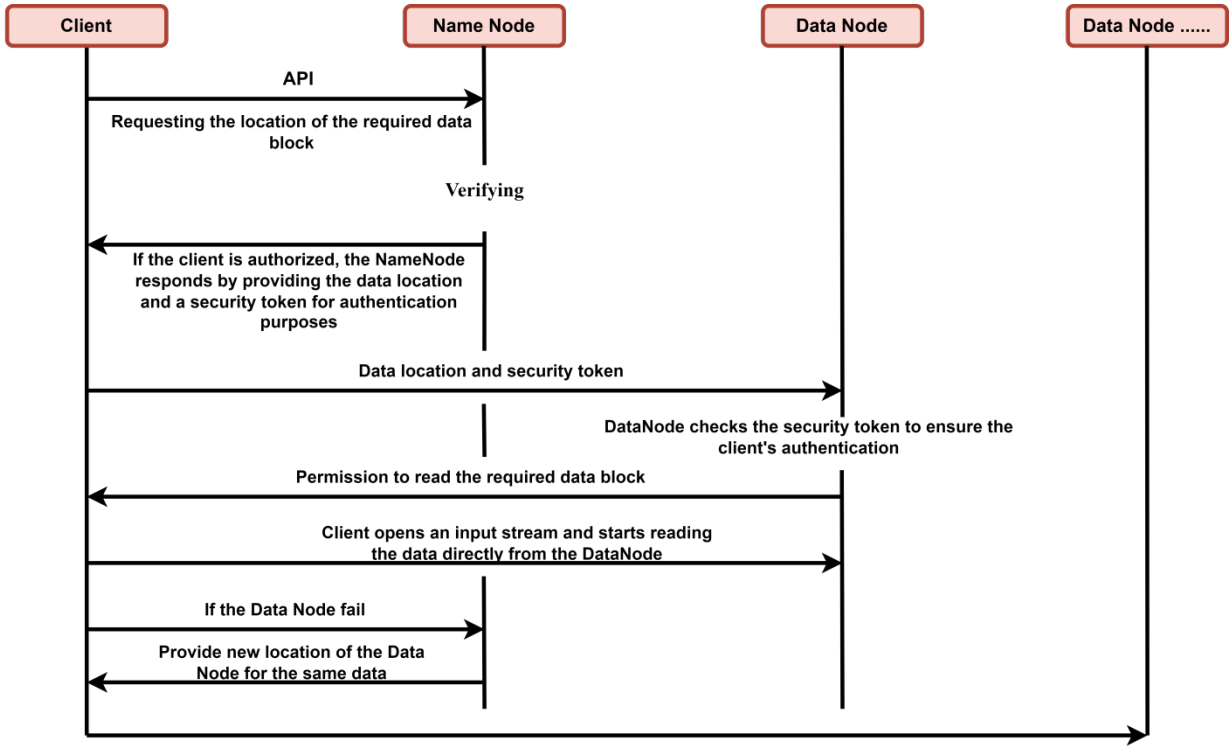

Figure 1. Read operation in HDFS

Overall, this process ensures secure and authorized access to the data stored in HDFS while providing fault tolerance in case of DN failures, resulting in a robust and reliable data retrieval mechanism.

2.4. Write operation in HDFS

The writing operation in Hadoop HDFS involves the client's interaction with the NN and the DNs. As shown in Figure 2 and based on the steps described in the provided text, the writing operation can be summarized as follows:

1. **Client Request:** The client contacts the NN through the API to obtain the location of the DN where it should start writing the data.
2. **Data Writing:** The client initiates the data writing process to the designated DN by utilizing the File FS data output stream.
3. **Replication:** Once the client finishes writing the first data block, the first DN replicates the same

block to other DNs to ensure data redundancy and fault tolerance. This replication process creates multiple copies of the block across different DNs in the HDFS cluster.

4. **DN-to-DN Copying:** After the replication is completed, the DN that possesses the replicated block starts copying the block to the next DN in the sequence. This process continues until all the necessary DNs have a copy of the data block.

The writing operation in HDFS is designed to achieve data reliability and fault tolerance through data replication. By creating multiple copies of the data blocks across different DNs, HDFS can tolerate the failure of individual nodes and ensure the availability of data even in the presence of hardware failures.

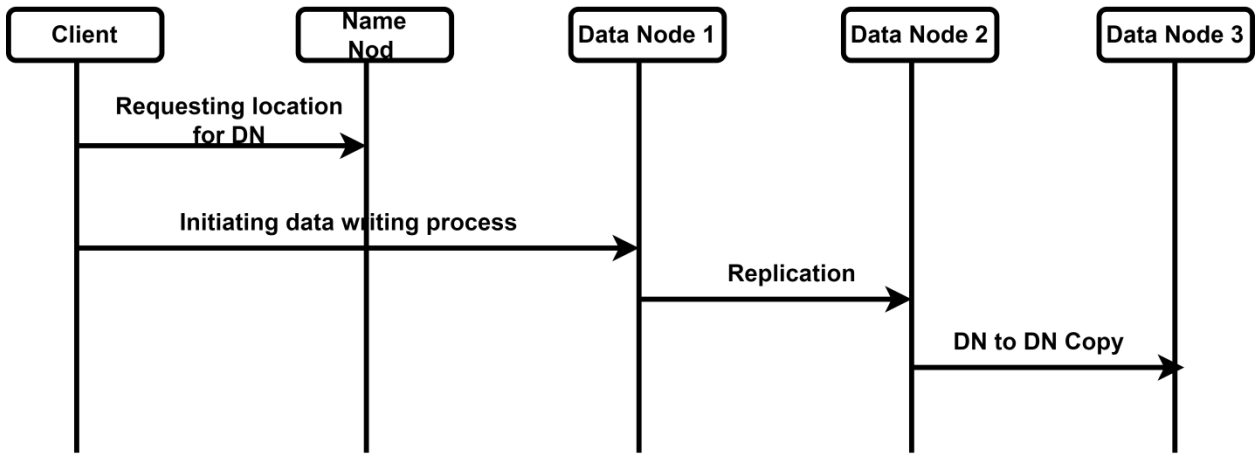

Figure 2. Write operation in HDFS

1. 3. Related Works

C.L. Abad et al. [15] introduce a DARE (dynamic data replication) approach for efficient cluster scheduling. It uses adaptive data replication based on demand patterns to reduce data access latency. DARE improves cluster scheduling efficiency by replicating data close to the nodes where it is frequently accessed. This reduces data transfer time and enhances overall cluster performance. DARE optimizes data replication dynamically, leading to better resource utilization and improved data availability in the cluster. B. Fan et. al. [16] proposes the use of RAID techniques for data-intensive scalable computing. Which combines multiple disks into a single logical unit to improve data reliability and performance. By implementing RAID, DiskReduce achieves better fault tolerance and data availability in data-intensive computing environments. DiskReduce's RAID implementation enhances data resilience and scalability, which is crucial for large-scale computing tasks. Z. Cheng et al. [17] introduces an elastic replication management system for the Hadoop Distributed File System (HDFS). It dynamically adjusts the replication factor of data blocks based on workload characteristics. ERMS effectively balances data replication and resource utilization in HDFS, leading to improved data access performance and reduced storage overhead. ERMS optimizes the replication process in HDFS, resulting in better data availability and more efficient use of storage resources. Q. Feng et. al. [18] proposes a storage architecture for cloud computing that emphasizes high reliability and low redundancy. Magicube achieves high data reliability with minimal redundancy, reducing storage costs and enhancing data integrity. Magicube's design ensures data safety and integrity in cloud computing environments without incurring excessive storage overhead. M. Patel Neha et al. [19] focuses on optimizing the write performance of Hadoop Distributed File System (HDFS) by strategically placing data replicas. By improving replica placement, the write performance in HDFS is enhanced, reducing write latency and improving overall data write throughput. Faster write performance leads to better data ingestion rates and overall HDFS efficiency. M. Patel Neha et al. [20] proposed a model on efficient replica placement, with a focus on data transfer rates and throughput in HDFS. By optimizing replica placement, data transfer rates and overall throughput in HDFS are improved. H. Zhang et al. [21] introduces SMARTH, a method to enable multi-pipeline data transfer in HDFS, allowing for concurrent data transfers. SMARTH's multi-

pipeline approach improves the overall data transfer efficiency in Hadoop clusters by enabling concurrent transfers. Algaradi et al. [22] proposes a mechanism enhances security in Hadoop by implementing a knowledge-based authentication system, improving access control and preventing unauthorized access. By using Kerberos and a knowledge-based approach, the authentication process is more robust and resistant to unauthorized access attempts. Tsu-Yang Wu et al. [23] introduces a lightweight authenticated key agreement protocol that utilizes fog nodes in the Social Internet of Vehicles (SIoV) environment. The proposed protocol establishes secure communication among SIoV entities and fog nodes, enhancing privacy and security in vehicular networks. The lightweight design of the protocol reduces computation and communication overhead in resource-constrained SIoV environments, while providing robust security mechanisms. Hena, M. et al. [24] presents a distributed authentication framework specifically designed for Hadoop-based big data environments. The framework ensures secure authentication and access control in distributed big data systems, safeguarding sensitive information and preventing unauthorized access. The tailored approach for Hadoop-based environments enhances security and scalability, providing efficient authentication services for large-scale big data systems. Honar Pajooh [25] proposes a provenance scheme that utilizes blockchain technology in the Hadoop ecosystem to maintain the integrity and traceability of IoT big data. The scheme ensures data authenticity, provenance tracking, and tamper-resistance, which is crucial for maintaining trustworthiness in IoT big data processing. By leveraging blockchain technology, the scheme enhances data trustworthiness and ensures that data from IoT devices can be verifiably traced and audited within the Hadoop ecosystem. Marco Anisetti et al. [26] introduces an assurance process for assessing and ensuring the trustworthiness of big data. The proposed process provides a comprehensive approach to evaluate the reliability, security, and quality of big data, enhancing user confidence in its usage. The assurance process allows organizations to effectively manage and monitor the trustworthiness of their big data, facilitating informed decision-making. Tall, A.M. et al. [27] presents a framework that enables fine-grained access control based on attributes, ensuring that data is accessed only by authorized users and processes. ABAC offers greater flexibility and security in big data environments with varying data sensitivities, allowing for more precise access control policies.

Several researchers have explored methods to enhance performance, reliability, and scalability in large-scale distributed and learning systems. Beyond the studies focusing on Hadoop and HDFS, additional relevant works provide insights into improving data transmission, compression, optimization, and reliability in distributed environments.

Zhao et al. [31] in *"Performance Analysis of Big Model Transmission under Double Rayleigh Fading"* investigated the transmission performance of large models over wireless channels affected by double Rayleigh fading. Their analytical and simulation-based approach evaluated outage probability and latency in unreliable communication environments. The study's emphasis on reliability and end-to-end transmission efficiency under fading conditions provides a useful analogy for improving data block transfer in HDFS when operating across heterogeneous or partially wireless clusters. By integrating channel reliability concepts, HDFS can better adapt to network variations during multi-pipeline transfers, reducing delays and improving fault tolerance.

Similarly, *"Compression and Transmission of Big AI Model Based on Deep Learning"[32]* proposed a deep-learning-driven compression framework to reduce the size of large AI models during network transfer while preserving accuracy. The adoption of adaptive compression and error-resilient encoding schemes demonstrates that efficient data representation can significantly enhance transfer speed and reliability. In the context of HDFS, such compression strategies can be applied to data blocks prior to replication or transmission through multiple pipelines, thus optimizing bandwidth utilization and further accelerating the write process without compromising data integrity.

In the field of optimization and intelligent resource selection, *"Bi-Directional Feature Fixation-Based Particle Swarm Optimization (BDFF-PSO) for Large-Scale Feature Selection"* introduced a self-adaptive bi-directional mechanism to improve convergence and balance exploration and exploitation in high-dimensional search spaces. This optimization strategy can be mapped to HDFS environments, where the selection of suitable DataNodes (DNs) for replication and pipelined writing can be modeled as a multi-objective optimization problem. Applying BDFF-PSO can enable dynamic and intelligent selection of DNs based on real-time reliability, bandwidth, and workload, leading to improved global efficiency and reduced data transfer latency [33].

Furthermore, in *"Elastic Optimization for Stragglers in Edge Federated Learning,"* [34] researchers addressed the challenge of uneven client performance (stragglers) by developing an elastic optimization mechanism that balances model updates and reduces training delays. The concept of elasticity—adapting resource participation dynamically—has direct relevance for HDFS environments. By adopting elastic replication policies and reliability-based

participation control, HDFS can dynamically exclude or down-weight underperforming DNs, improving system throughput and resilience during high-load or degraded network conditions.

Collectively, these studies demonstrate the significance of reliability modeling, adaptive compression, intelligent optimization, and elasticity in large-scale distributed systems. Integrating these principles into HDFS can lead to enhanced data transmission efficiency, fault tolerance, and throughput. Therefore, this paper's proposed multi-pipeline architecture with dynamic reliability evaluation aligns with and extends these contemporary research directions, establishing a holistic framework for next-generation distributed storage optimization.

Table 3: Comparative analysis of the related works

Citation & Year	Method Used	Advantages	Disadvantages	Future Use
[15] 2011	Adaptive data replication (DARE).	Improves scheduling efficiency in clusters.	May introduce overhead due to replication.	Enhancing data replication algorithms.
[16] 2009	RAID (Redundant Array of Independent Disks).	Enhances data reliability and availability.	Increased storage overhead.	Integration with modern storage technologies.
[17] 2012	Elastic replication management system.	Dynamically adjusts data replication.	Resource consumption during adjustments.	Further optimization for HDFS ecosystem.
[18] 2012	Magicube storage architecture.	High reliability with low redundancy.	Potential complexity in implementation.	Adaptation for different cloud computing platforms.
[19] 2014	Efficient replica placement.	Improved HDFS write performance.	Potential data consistency challenges.	Scalability for larger HDFS clusters.
[20] 2014	Efficient replica placement.	Increased data transfer rate and throughput.	May introduce additional network traffic.	Optimization for various network topologies.
[21] 2014	Multi-pipeline data transfer in HDFS.	Concurrent data transfers in Hadoop clusters.	Resource contention during concurrent transfers.	Integration with modern Hadoop distributions.
[22] 2019	Knowledge-Based Authentication Mechanism.	Enhanced security using Kerberos.	Initial setup and configuration complexity.	Security enhancement in distributed systems.
[23] 2021	Authenticated Key Agreement Protocol.	Enhanced security and privacy in SIoV.	Overhead due to cryptographic operations.	Scalability for larger SIoV environments.
[24] 2022	Distributed authentication framework.	Secure authentication and access control.	Additional processing overhead.	Scalability for larger big data environments.
[25] 2021	IoT Big Data provenance scheme.	Data authenticity and provenance tracking.	Potential blockchain scalability issues.	Integration with diverse IoT devices and platforms.
[26] 2023	Assurance process for Big Data trustworthiness.	Comprehensive evaluation of Big Data trustworthiness.	Resource consumption during evaluation.	Integration with various Big Data frameworks.
[27] 2023	Attribute-Based Access Control Framework	Fine-grained access control for big data	Additional complexity in access policies	Scalability for processing diverse sensitive data

4. Proposed Model

In this section, we present our proposed technique that aims to enhance the reliability and efficiency of data storage in an HDFS (Hadoop Distributed File System) cluster. Our approach focuses on selecting the most reliable DNs while also taking into account the full network bandwidth during the data writing process.

In the proposed technique in figure 3 for HDFS data writing, the process involves the following steps:

1. **Requesting File Construction:** When an HDFS client wants to create a new file, it sends a request to the NN.

2. **Reliability-Based DN Selection**: The NN responds to the client's request by providing

3. an ordered list of DNs to store the file data. The ordering is based on the reliability of each DN. The default value for the number of DNs in the list is set to 3, which means that the file data will be replicated on three DNs.

4. **Data Division and Pipelining:** The HDFS client divides the file data into default-sized blocks, which are subsequently split into packets. These packets are transmitted through a pipeline comprising multiple DNs. In the default replication setup, the pipeline consists of three nodes.

5. **Parallel Data Writing:** Indeed, in the default setup, the packets are sent in parallel to the first and second Data Nodes (DNs) in the pipeline. Each DN receives the packet, stores it locally, and then forwards it to the next DN in the pipeline, which is the third DN. This process ensures fault tolerance and data redundancy in HDFS, as the data is replicated across multiple nodes in the cluster.

6. **Acknowledgment and Pipelining:** After the first and second DNs receive the packets and store them, they send acknowledgment messages to the client.

7. **Reliability Update:** Simultaneously, each DN sends an update message to the NN to update its reliability value. The reliability value is evaluated using an algorithm that starts with initializing the reliability for each DN as 1. The algorithm uses inputs such as F-repli (reliability factor), mnreli, and mxreli. The reliability value for each DN is adapted based on this algorithm, ensuring that it converges towards an optimal reliability level.

8. **Acknowledgment and Reliability Update Continuation:** After storing packet on third DN, it sends an acknowledgment to the second DN and simultaneously updates its reliability value in the NN using the equations specified in step number 6.

This proposed technique aims to improve the overall reliability and efficiency of data storage in the HDFS cluster by strategically selecting reliable DNs and utilizing network bandwidth efficiently through pipelining. The continuous updating of reliability values ensures that the system adapts to changing conditions and maintains optimal data storage reliability.

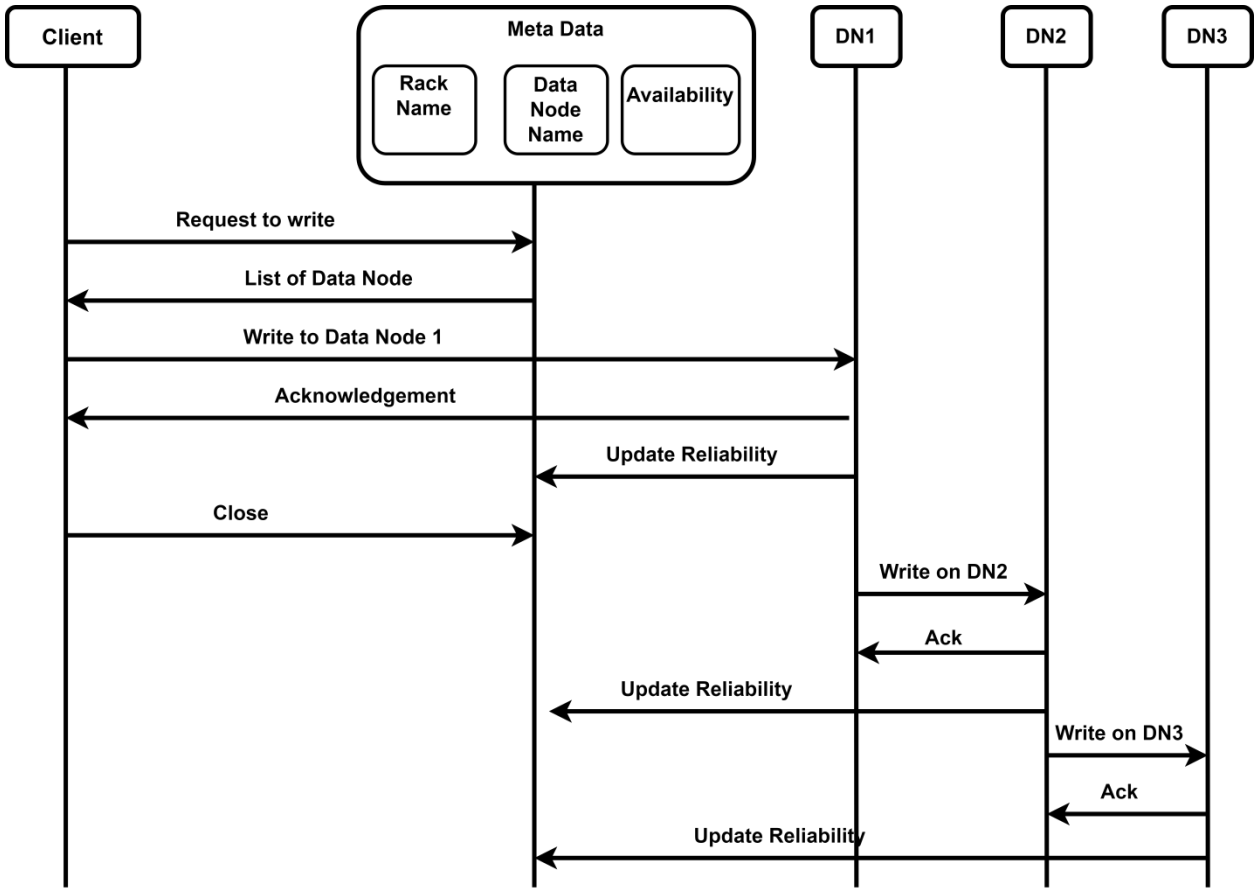

Figure 3. Proposed Model

Algorithm: Reliability Assessment

Initialization
reli=1, Adapt-F:= 0.01
Input
F-repli, mxreli, mnreli, DNS
Output
RDNS

Start
if(DNS= Success)
then
reli= reli+ (reli * F-repli)
if (adpt-F > 1)
then
adpt-F:= adpt-F –1;
else if (DNS = Fail)
then
reli= reli – (reli * F-repli * adpt-F)
adpt-F= adpt-F +1;
if (reli >= mxreli)
then
reli= mxreli
if (reli < mnreli)
then
DNS=Idle
call_proc: RDNS
End

5. Performance Evaluation

An often used assessment of HDFS involves leveraging its DNs across multiple racks. The conventional HDFS strategy works as follows: the NN selects available DNs to store the data block. The second replica is placed in a DN from a different rack, and the third replica is placed in a different DN on the same rack as the second. TestDFSIO benchmark is specifically designed to test various HDFS operations, including both reading and writing. By utilizing TestDFSIO, we were able to assess the performance of the network, operating system, and Hadoop setup on NN and

DNs. Additionally, this benchmark enabled us to calculate the average throughput for HDFS operations.

Our experiment was conducted with three different replication factors, block sizes of 64 MB and 128 MB, and file sizes ranging from 1 GB to 10 GB. Moreover, the initial reliability value was set to 1, with an adaptability factor of 0.01. Additionally, we set the mxreli to 1.4 and the mnreli to 0.06. These parameter settings were used to evaluate and analyze the performance of the proposed writing on HDFS approach under various scenarios. The experiment demonstrates that there is no significant gain when dealing with small file sizes. However, the proposed approach exhibits notable improvements, achieving a 35.9% enhancement compared to traditional HDFS with a 1 GB file and an even more substantial improvement of 61.3% with a 10 GB file.

When compared to the parallel broadcast algorithm, the proposed approach achieves a remarkable 48.9% reduction in execution time. Furthermore, it outperforms the parallel master-slave technique by an impressive 60.3% in the context of large file sizes. Table 4 and Figure 4 illustrates that while the parallel master-slave technique takes 210 seconds to upload a 5 GB file, the proposed approach only takes 90 seconds, making it 57.14% faster.

Table 4: Execution time (Sec) of write operation @ block size 64 MB

File Size	Execution Times (Sec) of different techniques			
	Parallel Broadcast	**Parallel Master-Slave**	**Lazy**	**Proposed Method**
1 GB	40	36	32	30
2 GB	60	65	59	52
3 GB	100	105	97	67
4 GB	135	140	110	85
5 GB	190	210	130	90
6 GB	220	225	170	110
7 GB	300	305	200	120
8 GB	320	330	265	125
9 GB	410	426	265	132
10 GB	450	470	350	190

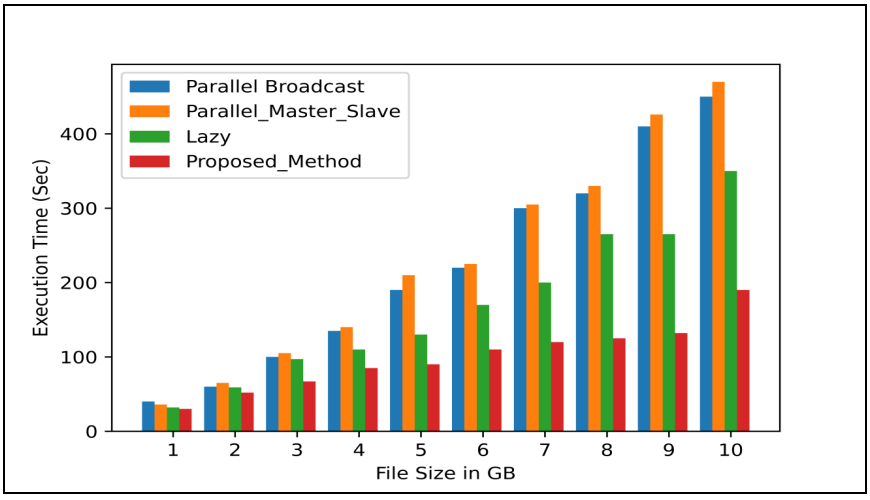

Figure 4. Execution time (Sec) of write operation @ block size 6

Table 5 and Figure 5 illustrate for HDFS writing operations with a block size of 128 MB. When compared to lazy HDFS with a 1 GB file, the proposed approach achieves a commendable improvement. Moreover, with a file size of 10 GB, the improvement reaches 34.03%. When comparing the proposed approach to traditional HDFS, the overall improvement in execution time reaches an impressive 65.06%.

Table 5: Execution time (Sec) of write operation @ block size 128 MB

File Size	Execution Times (Sec) of different techniques			
	Parallel Broadcast	Parallel Master- Slave	Lazy	Proposed Method
1 GB	70	80	60	55
2 GB	110	115	90	70
3 GB	180	200	175	105
4 GB	285	300	190	115
5 GB	380	400	200	180
6 GB	445	460	220	190
7 GB	600	603	300	205
8 GB	635	700	370	230
9 GB	726	770	400	290
10 GB	815	800	485	303

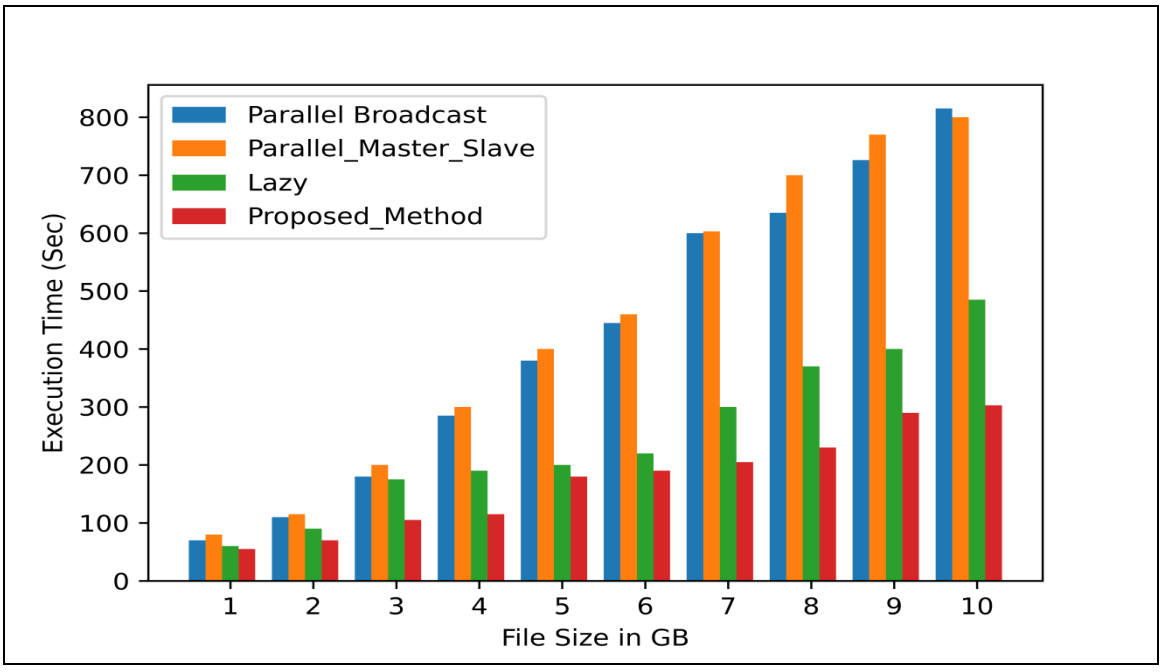

Figure 5. Execution time (Sec) of write operation @ block size 128 MB

Figures 5 and 7 display the enhancements in throughput achieved by different techniques, taking into account block sizes of 64 and 128, respectively. The graphs highlight the comparative performance of these techniques in terms of data transfer rates, showing how each approach fares with different block sizes.

Table 6: Throughput Rate (MB/Sec) of write operation @ block size 64 MB

File Size	Throughput Rate (GB/Sec) of different techniques			
	Parallel Broadcast	Parallel Master- Slave	Lazy	Proposed Method
1 GB	28	27	30	39
2 GB	28	26	31	38
3 GB	27	26	32	37
4 GB	25	24	31	38
5 GB	24	23	30	38
6 GB	24	23	28	37
7 GB	23	22	27	36
8 GB	23	22	27	35
9 GB	22	21	26	35
10 GB	22	21	25	34

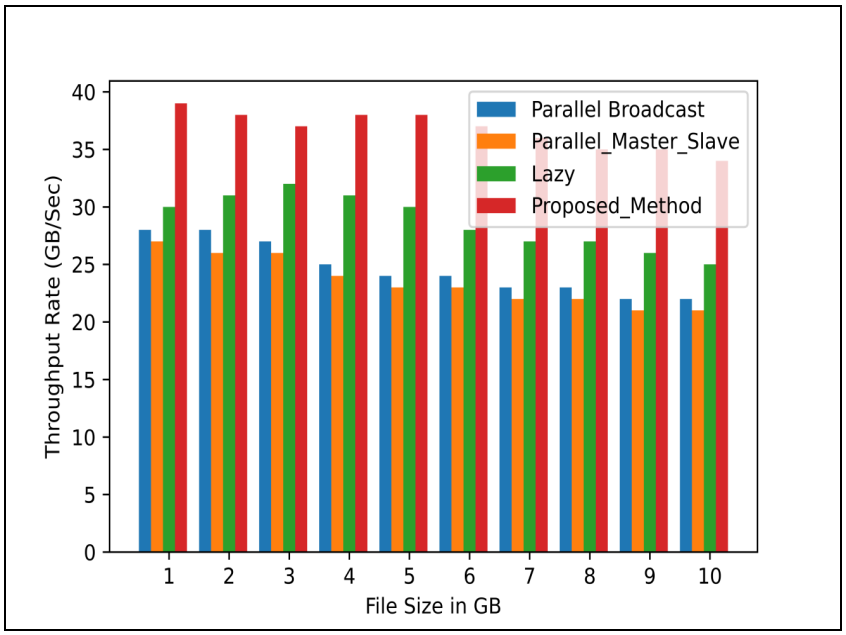

Figure 6. Throughput Rate (MB/Sec) of write operation @ block size 64 MB

Table 7: Throughput Rate (MB/Sec) of write operation @ block size 128 MB

File Size	Throughput Rate (MB/Sec) of different techniques			
	Parallel Broadcast	**Parallel Master- Slave**	**Lazy**	**Proposed Method**
1 GB	32	28	38	48
2 GB	29	28	38	48
3 GB	28	27	37	47
4 GB	27	26	36	47
5 GB	26	25	35	46
6 GB	26	25	35	45
7 GB	26	26	34	44
8 GB	26	25	34	44
9 GB	25.5	24	36	43
10 GB	25.5	24	35	42

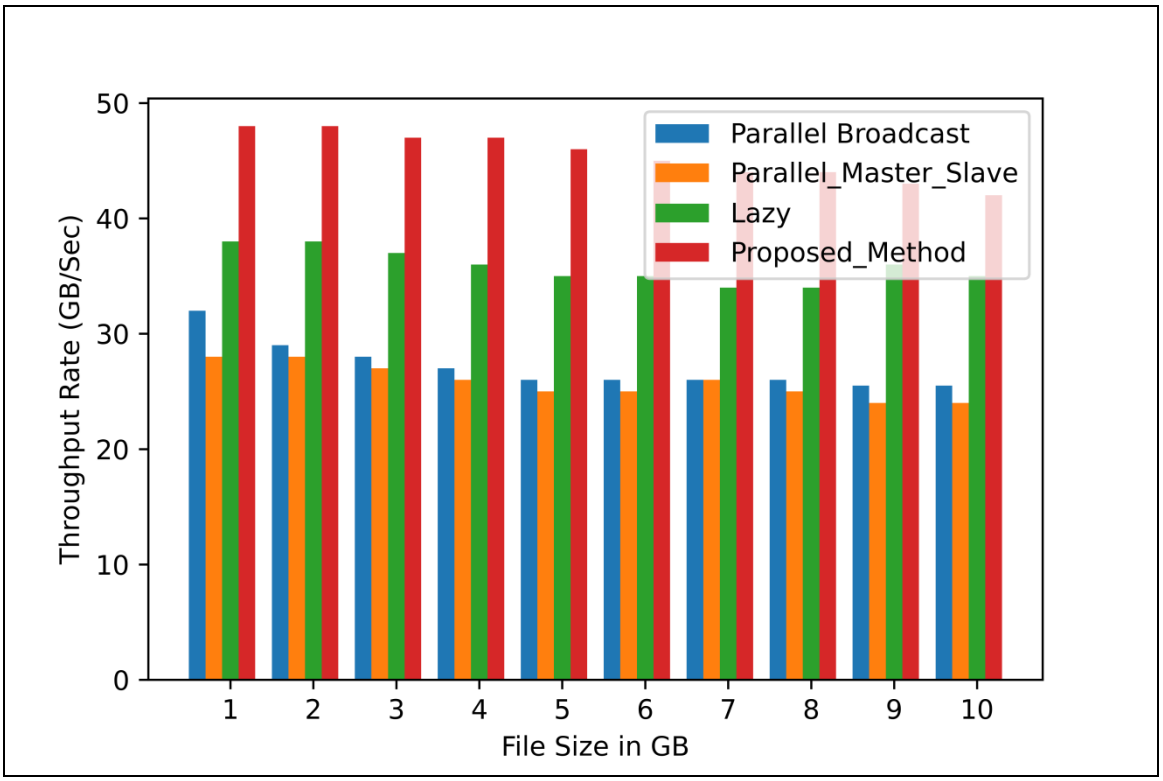

Figure 7. Throughput Rate (MB/Sec) of write operation @ block size 128 MB

With a 64 MB block size, the throughput improvement in the writing operation varies between 28.2% and 32.4% in the parallel broadcast technique, while the traditional HDFS technique shows an improvement ranging from 33.33% to 41.4%. When using a larger 128 MB block size, the overall percentage of throughput improvement reaches 18.7% in the lazy technique. This indicates that the proposed technique performs better in terms of throughput improvement with the 128 MB block size compared to the lazy technique.

5. Conclusion

In traditional HDFS, data reliability and availability are achieved through a data replication technique, wherein data blocks are replicated on different DNs, selected randomly, and transmitted using a single pipeline. This paper presented a novel approach to enhance the replication technique by selecting the most reliable DNs based on their reliability values. Our proposed approach employs a parallel pipelined architecture to replicate the data blocks, resulting in a substantial increase in the data transfer rate and, consequently, the overall system throughput. Specifically, with a block size of 64 MB, our approach has demonstrated an impressive 59.5% increase in system throughput. For a larger block size of 128 MB, the system throughput has been enhanced by 43.6%. This improvement can significantly enhance the performance of HDFS by ensuring data reliability and reducing data transfer latency, making it more efficient and reliable for handling large-scale data storage and processing tasks.

References

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, Design and Implementation of the Sun Network Filesystem, Proceedings of the USENIX Conference & Exibition, Portland, OR, 1985, pp. 119–130.

[2] J.H. Howard, An overview of the Andrew file system, Proceedings of the USENIX Winter Technical Conference, Dallas TX, 1988, pp. 23–26.

[3] S. Ghemawat, H. Gobioff, S.T. Leung, The Google file system, SOSP '03 Proceedings of the 19th ACM Symposium on Operating Systems Review, ACM, New York, USA, 2003, pp. 29–43.

[4] B. Martini, K.K.R. Choo, Distributed filesystem forensics: XtreemFS as a case study, Dig. Invest. 11 (2014), 95–313.

[5] Chen, M., Mao, S., & Liu, Y. (2014). Big Data: A Survey. Mobile Networks and Applications, 19(2), 171-209.

[6] Hassan, M. M., Hossain, M., & Mohammed, N. (2018). A Review on Big Data Analytics: Challenges, Open Research Issues and Tools. International Journal of Data Warehousing and Mining, 14(3), 1-26.

[7] Kumbhare, A., Meshram, B., & Dhoble, S. (2020). Big Data Analytics: Techniques, Challenges, and Opportunities. Journal of Big Data, 7(1), 1-30.

[8] Alharthi, A. A., El-Alfy, E. S. M., & Javaid, A. Q. (2019). Big Data Analytics in Healthcare: A Survey. International Journal of Computer Applications, 182(15), 1-7.

[9] Rajabi, F., Hosseinabady, M., & Mollahasani, A. (2018). A Survey on Big Data: Concepts, Applications, Challenges, and Future Trends. Journal of Information Systems and Telecommunication, 6(4), 258-267.

[10] Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 1-10.

[11] Ghemawat, S., Gobioff, H., & Leung, S. T. (2003). The Google File System. Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), 29(5), 29-43.

[12] Nambiar, V., Sankaralingam, K., & Cavanaugh, R. (2011). Benchmarking Big Data Systems and the BigData Top100 List. IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), 1-11.

[13] Khanafer, A., & Al-Hemyari, A. (2017). Hadoop Distributed File System (HDFS) Challenges and Solutions: A Survey. International Journal of Computer Applications, 171(7), 1-5.

[14] E. Sivaraman and R. Manickachezian, "High Performance and Fault Tolerant Distributed File System for Big Data Storage and Processing Using Hadoop," *2014 International Conference on Intelligent Computing Applications*, Coimbatore, India, 2014, pp. 32-36, doi: 10.1109/ICICA.2014.16.

[15] C.L. Abad, Y. Lu, R.H. Campbell, DARE: adaptive data replication for efficient cluster scheduling, 2011 IEEE International Conference on Cluster Computing, IEEE, Austin, TX, USA, 2011, pp. 159–168.

[16] B. Fan, W. Tantisiriroj, L. Xiao, G. Gibson, DiskReduce: RAID for data-intensive scalable computing, Proceedings of the 4th Annual Workshop on Petascale Data Storage, ACM, Portland, OR, USA, 2009, pp. 6–10.

[17] Z. Cheng, Z. Luan, Y. Meng, Y. Xu, D. Qian, A. Roy, et al., ERMS: an elastic replication management system for HDFS, 2012 IEEE International Conference on Cluster Computing Workshops, IEEE, Beijing, China, 2012, pp. 32–40.

[18] Q. Feng, J. Han, Y. Gao, D. Meng, Magicube: high reliability and low redundancy storage architecture for cloud computing, 2012 IEEE Seventh International Conference on Networking, Architecture and Storage (NAS), IEEE, Xiamen, Fujian, China, 2012, pp. 89–93.

[19] M. Patel Neha, M. Patel Narendra, M.I. Hasan, D. Shah Parth, M. Patel Mayur, Improving HDFS write performance using efficient replica placement, 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), IEEE, Noida, India, 2014, pp. 36–39.

[20] M. Patel Neha, M. Patel Narendra, M.I. Hasan, M. Patel Mayur, Improving data transfer rate and throughput of HDFS using efficient replica placement, Int. J. Comput. Appl. 86 (2014), 254–261.

[21] H. Zhang, L. Wang, H. Huang, SMARTH: enabling multipipeline data transfer in HDFS, 2014 43rd International Conference on Parallel Processing, IEEE, Minneapolis MN, USA, 2014, pp. 30–39.

[22] Algaradi, T. S., B. Rama. Static Knowledge-Based Authentication Mechanism for Hadoop Distributed Platform Using Kerberos. – Int. J. Adv. Sci. Eng. Inf. Technol., Vol. 9, 2019, No 3, pp. 772-780.

[23] Tsu-Yang Wu, Xinglan Guo, Lei Yang, Qian Meng, Chien-Ming Chen, "A Lightweight Authenticated Key Agreement Protocol Using Fog Nodes in Social Internet of Vehicles", Mobile Information Systems, vol. 2021, Article ID 3277113, 14 pages, 2021. https://doi.org/10.1155/2021/3277113

[24] Hena, M., Jeyanthi, N. Distributed authentication framework for Hadoop based bigdata environment. J Ambient Intell Human Comput 13, 4397–4414 (2022). https://doi.org/10.1007/s12652-021-03522-0

[25] Honar Pajooh, H., Rashid, M.A., Alam, F. et al. IoT Big Data provenance scheme using blockchain on Hadoop ecosystem. J Big Data 8, 114 (2021). https://doi.org/10.1186/s40537-021-00505

[26] Marco Anisetti, Claudio A. Ardagna, Filippo Berto, An assurance process for Big Data trust worthiness ,Future Generation Computer Systems,Volume 146,2023,Pages 34-46,ISSN 0167-739X,

[27] Tall, A.M.; Zou, C.C. A Framework for Attribute-Based Access Control in Processing Big Data with Multiple Sensitivities. Appl. Sci. 2023, 13, 1183. https://doi.org/10.3390/app13021183

[28] M.G. Noll, Benchmarking and stress testing an Hadoop cluster with TeraSort, TestDFSIO & Co., 2011, Available from: http://www.michael-noll.com/blog/2011/04/09/benchmarking-and-stress-testing-an-hadoop-cluster-with-terasort-testdfsio-nnbench-mrbench/.

[29] Gupta M, Dwivedi RK. Fortified MapReduce Layer: Elevating Security and Privacy in Big Data . EAI Endorsed Scal Inf Syst [Internet]. 2023 Oct. 2 [cited 2023 Nov. 3];10(6).

[30] M. K. Gupta, A. K. Rai, B. Pandey, A. Gupta and V. K. Verma, "Big Data Privacy: A Survey Paper," *2023 International Conference on IoT, Communication and Automation Technology (ICICAT)*, Gorakhpur, India, 2023, pp. 1-6, doi: 10.1109/ICICAT57735.2023.10263627.