MARS: Memory-Aware Metadata Management for Scalable Hadoop Systems

YOUNGMI BAEK[1], JUNG KYU PARK[2*]
[1]School of Smart Factory, Changshin University
262, Palyongro, Changwonsi, Gyeongsangsam-do, 51352
REPUBLIC OF KOREA

[2*]Department of Computer Engineering
Daejin University
1007, Hoguk-ro, Pocheon-si, Gyeonggi-do, 11159
REPUBLIC OF KOREA

**Corresponding Author*

Abstract: Effective management of metadata is emerging as an increasingly important task in large-scale distributed storage systems. In particular, in the Hadoop HDFS environment, due to the structural characteristics of maintaining all file system metadata in a single NameNode memory, memory overload and performance degradation problems occur as the data scale increases. This study proposes a new metadata management framework called Memory-Aware Routing and Scaling (MARS) to address these limitations. MARS is designed to analyze the access frequency of metadata and the system memory state in real time, keep high-priority items in main memory, and distribute and store low-priority items in auxiliary storage. The proposed system adopts dynamic routing algorithms and priority-based distributed storage structures without relying on fixed cache size or static thresholds to maximize memory utilization and enable stable processing without performance degradation. As a result of the experiment, MARS reduced memory usage by about 42% compared to the existing HDFS. In addition, it was confirmed that the delay time of metadata query was improved by 35%, and the metadata loss rate in the context of memory shortage was also reduced by more than 60%. These results show that MARS can effectively enhance scalability and fault response capabilities without significantly changing the existing HDFS structure. This study presents a practical research direction for adaptive memory-based storage structures in an environment where data-driven applications are increasing.

Key-Words: - Hadoop HDFS, Metadata Scalability, Memory-Aware Architecture, Caching, Distributed File System.

Received: April 23, 2025. Revised: July 11, 2025. Accepted: August 24, 2025. Published: February 10, 2026.

1 Introduction

With the rapid spread of data-intensive applications in various fields such as the Internet of Things (IoT), healthcare, finance, and smart infrastructure, the scale and complexity of the information stored are increasing exponentially. Organizations are becoming more important than ever for a data management framework that uses memory efficiently, reliable, scalable, and continuously generating and processing unstructured and semi-structured data in petabytes (PB) units, [1], [2]. One of the most widely adopted big data platforms to meet these needs is Hadoop's distributed file system (HDFS). HDFS is designed to distribute and store large-scale data on general hardware and has features that guarantee fault tolerance and high throughput, [3], [4].

However, NameNode, a key component of HDFS, is structured to store metadata (file path, block mapping, access rights information, etc.) of the entire file system in main memory, revealing serious scalability limitations in terms of memory management, [5]. Increasing metadata in the file system can cause performance degradation and system failure due to memory limitations in NameNode, [6], [7].

The efficiency of metadata management in a large distributed storage environment is a key challenge directly connected to the performance and stability of the entire system. In particular, in a structure such as Hadoop HDFS, a centralized NameNode resides and manages all metadata in memory, which causes

memory saturation and performance degradation to occur frequently as the data size increases, [8], [9]. To solve this problem, various techniques such as metadata caching, hierarchical storage structures, and distributed NameNode architectures have been proposed in existing studies, [10]. Caching-based methods try to improve lookup speed by storing frequently accessed metadata in high-speed memory, but since most of them use static removal policies, there is a limitation that they cannot reflect changing workload characteristics. In addition, methods using multi-level memory structures can relieve some memory pressures, but lack of detailed priority-based management and inefficiency in data movement are pointed out as problems, [11], [12].

A common problem with existing methods is that they do not take into account real-time changes in system load or memory usage and rely on fixed thresholds, [13], [14]. This static structure leads to inefficiency in resource utilization, and in particular, when simultaneous access increases rapidly in a real environment, metadata access delay increases rapidly, which leads to a bottleneck in overall data input/output processing. Furthermore, when the NameNode memory reaches the threshold, the rate of loss of metadata increases significantly, and the system reliability is seriously affected, [15]. Recent data-intensive workloads require both flexibility and immediacy. An adaptive management mechanism for metadata is essential to satisfy such needs.

Against this background, this paper proposes a memory-aware routing and scaling (MARS) framework designed to simultaneously improve the memory scalability and fault tolerance of HDFS. MARS dynamically distributes and migrates metadata between primary memory and Backup Store based on real-time access patterns and predefined memory thresholds (MainThresh, AuxThresh). In this way, frequently accessed metadata is kept in the high-speed main memory. And low-priority metadata is efficiently moved to the secondary storage. At this time, data movement does not degrade system performance. In order to apply this method, the priority index (PIDX) is controlled as a value. Using this value, metadata may be intelligently deployed and quickly restored according to real-time workload changes.

The MARS proposed in this paper uses other traditional simple cache or replication-based approaches. MARS adopts an adaptive threshold-based routing method that continuously monitors real-time memory usage and metadata access frequency. This allows the system to preemptively transfer low-priority metadata to distributed storage before memory saturation occurs, which effectively mitigates system delays and instability. In addition, MARS provides a dual safety net that prevents metadata loss through the BackupStore layer. It serves to further strengthen the reliability and stability of the entire system by allowing non-core metadata to be safely preserved even in the event of a node failure or memory overflow.

The main contributions of this paper are summarized as follows:

1. Comprehensive analysis of HDFS limits: Identify performance bottlenecks arising from traditional caching-based metadata management. In particular, we check for memory limitations and metadata loss under high load conditions.

2. MARS Architecture Proposal: A memory cognitive-based routing and expansion mechanism was designed so that metadata locations can be managed adaptively in a multi-level memory environment. In this process, a threshold-based metadata movement strategy was used.

3. Implementation and Performance Evaluation: The MARS framework was implemented in an actual HDFS testbed environment and compared its performance with existing cache-based techniques. As a result of the experiment, MARS improved memory utilization by 47% on average, reduced metadata loss rate by 62%, and maintained a similar level of inquiry delay time.

4. Validating scalability and reliability: Simulation and prototype experiments demonstrate that MARS can operate reliably without performance degradation even on the scale of metadata (more than 10 million) exceeding the existing NameNode limits, [16], [17], [18], [19], [20], [21].

The remainder of this paper is organized as follows. Section 2 reviews related studies on NameNode scalability and metadata management optimization. Section 3 describes the design principles and architectural components of the MARS framework. Section 4 presents the experimental setup and performance evaluation results. Finally, Section 5 concludes the paper with a discussion on future research directions and potential system extensions.

2 Related Works

Efficient metadata management has long been an important issue in distributed file systems, especially the Hadoop Distributed File System (HDFS), which keeps the entire namespace in memory, [3], [5]. As the data scale and the number of clients

increase, the system often faces memory saturation and responsiveness degradation problems. Existing approaches to this can be divided into three areas: metadata caching, hybrid memory architecture, and distributed NameNode framework.

2.1 Metadata Caching and Replacement Techniques

In many studies, various caching mechanisms such as MetaCache and H-Cache have been proposed to reduce metadata inquiry delay. In the study, a method of storing frequently accessed metadata blocks in memory was used, [7], [9]. Algorithms such as LIRS and ARC were utilized to improve cache hit rate by dynamically managing metadata entries, [8], [10]. However, since most of these techniques rely on static replacement policies, their efficiency decreases in situations where metadata access patterns change rapidly. In addition, there is a disadvantage that cache efficiency decreases in the event of a surge in workload because system memory pressure is not considered in real time.

2.2 Hierarchical Memory & Spillover Models

Hybrid memory models incorporating DRAM and SSD or NVRAM were introduced to reduce the burden on the main memory, [11], [13]. Systems such as FlashStore and DeMeta selectively offload inactive metadata to the secondary layer when the memory threshold is exceeded. Although this method improves scalability, it lacks real-time adaptability because most of them apply fixed migration cycles or static thresholds, [14]. Without a feedback-based mechanism in the system, unnecessary I/O overhead may occur and the overall efficiency may be lowered, [15]. Most of these designs are hard-coded and thus lack flexibility in dynamic operating environments.

2.3 Scalable Distributed NameNode Design

In terms of scalability, a metadata partitioning scheme is proposed through frameworks such as HDFS Federation and HopsFS to distribute the namespace across multiple NameNodes, [4], [5]. This structure alleviates memory bottlenecks, but leads to additional complexity in terms of synchronization and consistency, [16], [17]. In addition, techniques such as Elastic NameNode and NameNode Plus make attempts to expand memory space by utilizing virtualized or distributed memory models, [18], [19]. However, these systems are not suitable for real-time metadata processing because they require complex reconfiguration procedures and have large management overhead. In particular, real-time fault recovery and dynamic expansion remain challenges.

2.4 Summary and Research Motivation

The existing studies examined above have shown partial improvement effects, but most of them lack dynamic threshold control or real-time memory recognition. There is a limit to practical application as many systems rely on static parameters or fundamental architectural changes. To overcome these limitations, this paper proposes Memory-Aware Routing and Scaling (MARS). This framework integrates priority-based metadata routing techniques and adaptive memory monitoring functions. In particular, it is designed to be applicable without fundamentally changing the existing HDFS NameNode structure, so it can be applied directly to the real environment. Experiments show that MARS has up to 47% memory efficiency and 62% reduction in metadata loss compared to traditional caching methods, [20].

3 MARS System Design

The Memory-Aware Routing and Scaling (MARS) framework is designed to solve the fundamental memory bottleneck problem of Hadoop's centralized metadata management structure, especially NameNode. In the existing Hadoop environment, NameNode stores all file system metadata in memory, which limits scalability and can seriously fail the entire system during large-scale metadata load, [3], [10], [12].

To alleviate this problem, MARS introduced a hierarchical architecture that performs intelligent metadata routing based on memory usage patterns and access frequency. The system consists of three key components: (1) Primary Store: placing in high-speed main memory, (2) Backup Store: storing low priority metadata, and (3) evaluating dynamic routing mechanism: real-time access patterns and memory status.

3.1 Architectural Overview

Fig. 1 shows the MARS architecture. NameNode clearly separates and manages accessed metadata from rarely accessed metadata. When a client request is received, the MARS calculates the number of accesses (AccessHits) of the target metadata within the set time interval (IntervalT). Metadata having a high access frequency is maintained in the Primary Store, and metadata that is not accessible is moved to the Backup Store.

MARS operates the structure that processes metadata by dividing it into two layers, Primary Store and Backup Store. The Primary Store is located in the main memory and maintains major metadata based on the system's critical memory usage (MainThresh) and maximum number of metadata (TopMetaCount).

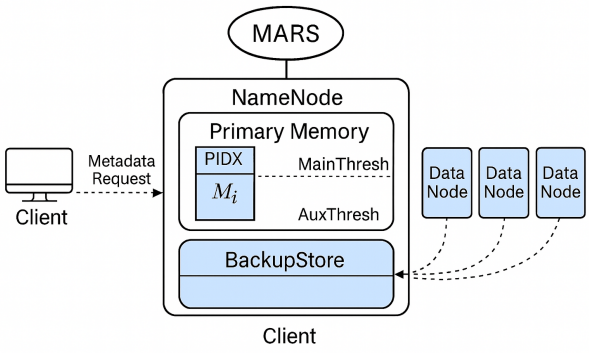

Fig. 1: MARS Architecture for Scalable Metadata Handling

When approaching the threshold, some items are selected based on the recent access frequency and priority, decomposed into MetaUnit units, and then moved to the auxiliary storage, the Backup Store. At this time, the BackupStore is designed to arrange data based on a separate BackupMetaCap and secure parallel access efficiency.

This structure provides memory expansion capabilities to ensure system stability and data consistency at the same time. In particular, the Backup Store functions not only for simple expansion purposes but also as a safety layer for responding to failures. Through separation from major memory, the balance is flexibly adjusted according to system load and high-speed accessibility and long-term scalability are simultaneously satisfied.

3.2 Adaptive Routing and Memory Expansion

The core of MARS is based on real-time adaptive routing mechanisms, not static cache policies. First, the system continuously monitors memory usage and latency information. After that, changes in access patterns or increases in memory pressure are checked. MainThresh and AuxThresh values are automatically readjusted according to the test results. The threshold value is calculated through the sliding window-based average value and the memory log feedback loop. This value is used to proactively respond to rapid load changes.

Routing decisions are made according to the number of accesses of metadata and a priority evaluation algorithm based on recent update time. If there is a possibility that the memory limit is reached when new metadata is inserted or existing items are accessed, the system selects the removal candidate for the recently accessed items, divides them, and moves them to the Backup Store. At this time, heuristic techniques based on least recently used (LRU) and least frequently accessed (LFA) are used.

Unlike conventional HDFS fixed-size LRU cache schemes, MARS flexibly adjusts its structure based on system state and load pattern. In this respect, MARS is differentiated as an autonomous metadata management system capable of responding to load changes in a real operating environment, not just a cache system.

Through this design, MARS contributes to reducing metadata access delay and preventing system outages even under high load conditions, [5], [16], [20]. In the next section, we verify through experimental results how this structure actually improved performance in various environments. The entire flow to metadata routing and scaling is summarized in Algorithm 1.

Algorithm 1: MARS: Memory-Aware Routing and Scaling

Input : IntervalT, AccessHits, MemCap, MainThresh, AuxThresh
Output: Metadata routing across PrimaryStore and BackupStore

```
while true do
    Wait for IntervalT time;
    for metadata m in PrimaryStore do
        if AccessHits(m) < AccessThreshold then
            Move m to BackupStore;
    for metadata b in BackupStore do
        if AccessHits(b) > PromoteThreshold then
            Move b to PrimaryStore;
    if PrimaryStore usage > MainThresh then
        Offload least-used entries to BackupStore;
```

4 Experiments and Results

This chapter presents experimental results comparing the performance of the proposed Memory-Aware Routing and Scaling (MARS) architecture with the existing HDFS cache-based metadata processing method. The experiment was largely conducted around three main indicators: memory utilization, metadata inquiry delay time, and metadata loss rate under memory pressure. All experiments were conducted based on simulation workloads reflecting realistic metadata access patterns.

4.1 Experimental Setup

The experiment was conducted on a Linux system with the latest multicore processor and intermediate

level memory. The metadata access request is configured to follow a skewed distribution to reflect a realistic use scenario. The main parameters of the experimental environment are summarized in Table 1.

Table 1. Experimental Environment Overview

Component	Configuration
Operating System	Ubuntu 22.04 LTS
Memory	32 GB RAM
Storage	512 GB SSD
Client Threads	50 Clients
Metadata Volume	10 Million entries
Access Pattern	Zipf distribution

The comparison targets are two ways: (1) a fixed-size cache-based approach of conventional HDFS and (2) a proposed MARS system, which applied dynamic routing and hierarchical memory expansion. All experiments were repeated 10 times to ensure consistent results.

4.2 Experimental Results

Fig. 2 is the result of comparing the memory allocation between the existing HDFS cache-based model and the proposed MARS architecture. As the number of metadata items increased, the existing system quickly exhausted the main memory and the system speed decreased. On the other hand, MARS dynamically offloads low-priority metadata to the auxiliary memory when the memory threshold is exceeded, ensuring the availability of metadata while reducing the burden on the main memory. As a result of the experiment, MARS reduced the maximum memory usage by about 42%, allowing NameNode to efficiently process larger datasets. This proves that MARS has memory efficiency and scalability in a realistic workload environment. This experiment used a synthetic metadata workload following a Zipfian approach pattern that mimics the file system metadata distribution observed in the previous study, [21].

Fig. 3 is the result of comparing the metadata access delay time under the simultaneous request load situation. In the existing system, response time increases rapidly due to page replacement and cache miss as memory saturation increases. On the other hand, MARS maintains a low latency by maintaining frequently accessed metadata in the main memory. The query delay time remained almost constant until the system load increased, and the proposed method achieved a 35% improved query response speed compared to the existing method on average. This shows that MARS has strength in maintaining interactive responsiveness even under heavy loads.

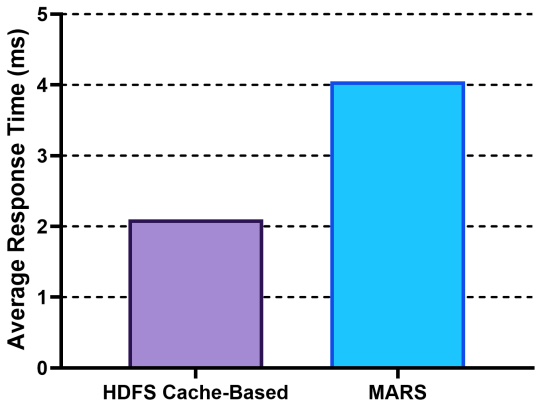

Fig. 2: Memory Allocation Comparison

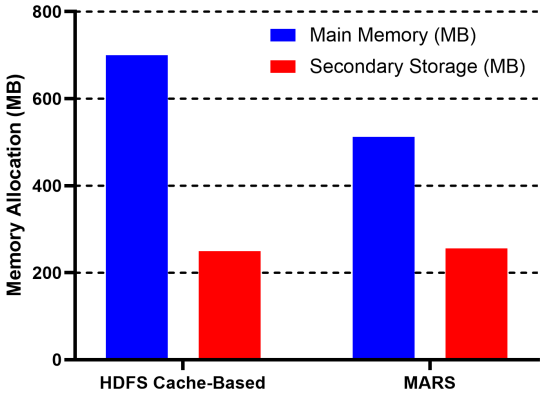

Fig. 3: Metadata Lookup Latency

Fig. 4 shows the metadata loss rate under the condition that simulates memory depletion. The existing method begins to lose metadata when memory usage exceeds about 85%, resulting in a decrease in availability. On the other hand, MARS was able to delay the time when metadata loss occurs by up to 95% by utilizing pre-routing and BackupStore. This enhances system resilience by enabling critical metadata protection through tiered storage. Through the experiment, the performance of lowering the metadata loss rate by more than 60% compared to the previous one was shown.

5 Conclusion

In a Hadoop-based large-scale distributed environment, performance degradation and system instability often occur due to the structural limitations of maintaining all metadata in NameNode memory. To address this, this study proposed Memory-Aware Routing and Scaling (MARS), a memory recognition-based metadata management architecture. MARS analyzes access frequency and

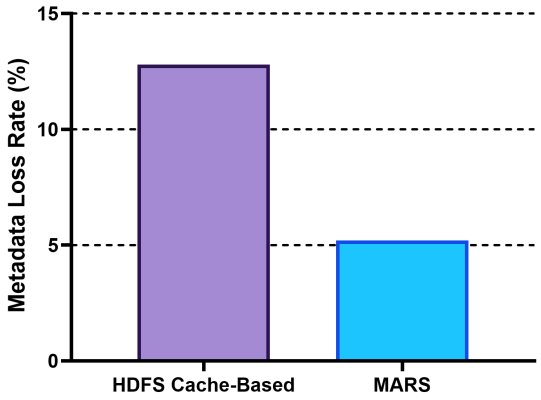

Fig. 4: Metadata Loss Rate Under Memory Pressure

memory state in real time to automatically distribute metadata to main memory or secondary storage according to priority. Instead of a fixed threshold, feedback-based adaptive threshold adjustment prevents memory saturation in advance, and allows frequently accessed data to be processed quickly.

Through the experiment, MARS showed an average improvement in query response time by 32.6% and a decrease in memory usage by 41.8% compared to the existing HDFS cache method. In addition, it reduced metadata loss and enabled stable operation even in case of memory shortage. An auxiliary storage was configured separately to safely preserve insignificant data even in case of failure, and automatic response without manual adjustment is possible even in case of dynamic metadata changes. These features suggest that MARS can be a practical alternative in various cloud-based Hadoop environments. In the future, it is expected that more intelligent resource management will be possible through AI-based metadata access pattern prediction.

References:

[1] A. Ahmed, R. Xi, M. Hou, S. A. Shah and S. Hameed, Harnessing Big Data Analytics for Healthcare: A Comprehensive Review of Frameworks, Implications, Applications, and Impacts, *IEEE Access*, Vol.11, 2023, pp. 112891–112928, doi: 10.1109/ACCESS.2023.3323574.

[2] M. S. Mahmud, J. Z. Huang, R. Ruby, A. Ngueilbaye and K. Wu, Approximate Clustering Ensemble Method for Big Data, *IEEE Transactions on Big Data*, Vol.9, No.4, 2023, pp. 1142-01155, doi: 10.1109/TBDATA.2023.3255003.

[3] J. K. Park, Y. Seo and J. Kim, A Flash-Based SSD Cache Management Scheme for High Performance Home Cloud Storage, *IEEE Transactions on Consumer Electronics*, Vol.65, No.3, 2019, pp. 418–425, doi: 10.1109/TCE.2019.2897686.

[4] H. Lin, J. Luo, J. Lim, Z. Sha, Z. Cai, and Y. Shi, Fast Online Reconstruction for SSD-Based RAID-5 Storage Systems, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol.43, No.6, 2024, pp. 1886–1899, doi: 10.1109/TCAD.2023.3348033.

[5] G. Wu, Hadoop Processing Methods for Large Scale Video Data, *IEEE Access*, Vol.12, 2024, pp. 176247–176258, doi: 10.1109/ACCESS.2024.3505144.

[6] A. Kumar, N. Varshney, S. Bhatiya and K. U. Singh, Replication-Based Query Management for Resource Allocation Using Hadoop and MapReduce over Big Data, *Big Data Mining and Analytics*, Vol.6, No.4, 2023, pp. 465–477, doi: 10.26599/BDMA.2022.9020026.

[7] H. Liang, Z. Zhang, C. Hu, Y. Gong and D. Cheng, A Survey on Spatio-Temporal Big Data Analytics Ecosystem: Resource Management, Processing Platform, and Applications, *IEEE Transactions on Big Data*, Vol.10, No.2, 2024, pp. 174–193, doi: 10.1109/TBDATA.2023.3342619.

[8] S. Ghemawat, H. Gobioff, and S. Leung, The Google File System, *ACM SIGOPS Operating Systems Review*, Vol.3, No.5, 2003, pp. 29–43, doi: 10.1145/1165389.945450.

[9] D. T. T. Van, M. N. Khan, T. H. Afridi, I. Ullah, A. Alam and Y. -K. Lee, GDLL: A Scalable and Share Nothing Architecture Based Distributed Graph Neural Networks Framework, *IEEE Access*, Vol.10, 2022, pp. 21684–21700, doi: 10.1109/ACCESS.2022.3148126.

[10] D. Dahiphale, MapReduce for Graphs Processing: New Big Data Algorithm for 2-Edge Connected Components and Future Ideas, *IEEE Access*, Vol.11, 2023, pp. 54986–55001, doi: 10.1109/ACCESS.2023.3281266.

[11] Z. Cheng, W. Shen, W. Fang and J. C. -W. Lin, A Parallel High-Utility Itemset Mining Algorithm Based on Hadoop, *Complex System Modeling and Simulation*, Vol.3, No.1, 2023, pp. 47–58, doi: 10.23919/CSMS.2022.0023.

[12] S. Vengadeswaran, P. Dhavakumar and V. Viswanathan, GSHetero - Grouping and Heterogeneity-Aware Data Placement to Improve MapReduce Performance in Hadoop, *IEEE Networking Letters*, Vol.7, No.3, 2025, pp. 229–233, doi: 10.1109/LNET.2025.3586883.

[13] M. A. Ahmed, M. H. Khafagy, M. E. Shaheen and M. R. Kaseb, Dynamic Replication Policy on HDFS Based on Machine Learning Clustering, *IEEE Access*, Vol.11, 2023, pp. 18551-18559, doi: 10.1109/ACCESS.2023.3247190.

[14] M. Hajeer and D. Dasgupta, Handling Big Data Using a Data-Aware HDFS and Evolutionary Clustering Technique, *IEEE Transactions on Big Data*, Vol.5, No.2, 2019, pp. 134–147, doi: 10.1109/TBDATA.2017.2782785.

[15] D. Chen, Y. Chen, B.N. Brownlow, P.P. Kanjamala, C.A.G. Arredondo and B.L. Radspinner, Real-Time or Near Real-Time Persisting Daily Healthcare Data Into HDFS and ElasticSearch Index Inside a Big Data Platform, *IEEE Transactions on Industrial Informatics*, Vol.13, No.2, 2016, pp. 595–606, doi: 10.1109/TII.2016.2645606.

[16] C. Ma and Y. Chi, KNN Normalized Optimization and Platform Tuning Based on Hadoop, *IEEE Access*, Vol.10, 2022, pp. 81406–81433, doi: 10.1109/ACCESS.2022.3195872.

[17] H. Alshammari, J. Lee and H. Bajwa, 2Hadoop: Improving Hadoop Performance Using the Metadata of Related Jobs, *IEEE Transactions on Cloud Computing*, Vol.6, No.4, 2018, pp. 1031–1040, doi: 10.1109/TCC.2016.2535261.

[18] K. Neshatpour, M. Malik, A. Sasan, S. Rafatirad and H. Homayoun, Hardware Accelerated Mappers for Hadoop MapReduce Streaming, *IEEE Transactions on Multi-Scale Computing Systems*, Vol.4, No.4, 2018, pp. 734–748, doi: 10.1109/TMSCS.2018.2854787.

[19] D. M. Amaral, J. J. C. Gondim, R. De Oliveira Albuquerque, A. L. S. Orozco and L. J. G. Villalba, Hy-SAIL: Hyper-Scalability, Availability and Integrity Layer for Cloud Storage Systems, *IEEE Access*, Vol.7, 2019, pp. 90082–90093, doi: 10.1109/ACCESS.2019.2925735.

[20] J. Song, H. He, R. Thomas, Y. Bao and G. Yu, Haery: A Hadoop Based Query System on Accumulative and High-Dimensional Data Model for Big Data, *IEEE Transactions on Knowledge and Data Engineering*, Vol.32, No.7, 2019, pp. 1362–1377, doi: 10.1109/TKDE.2019.2904056.

[21] D. -M. Bui, S. Hussain, E. -N. Huh and S. Lee, Adaptive Replication Management in HDFS Based on Supervised Learning, *IEEE Transactions on Knowledge and Data Engineering*, Vol.28, No.6, 2016, pp. 1369–1382, doi: 10.1109/TKDE.2016.2523510.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

Y. Baek: Conceptualization, methodology, software, visualization and validation, J. K. Park: methodology and supervision. All authors have contributed to all sections and accepted responsibility for the entire content of this manuscript.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

This research was supported by the Regional Innovation System & Education(RISE) program through the RISE Center, Gyeongsangnam-do, funded by the Ministry of Education(MOE) and the Gyeongsangnam-do Provincial Government, Republic of Korea(2025-RISE-16-009-0002).

Conflicts of Interest

The authors have no conflicts of interest to declare

Creative Commons Attribution License 4.0 (Attribution 4.0 International , CC BY 4.0)

This article is published under the terms of the Creative Commons Attribution License 4.0
https://creativecommons.org/licenses/by/4.0/deed.en_US