A Data-Correlation and Load-Balancing-Based Optimization Framework for Metadata Migration in Distributed Storage Systems

Jingman He, Zehui Zhang*, Jie Peng, Lin Zhou, Liwei Wang
Inner Mongolia Power Digital Research Institute, Hohhot, Inner Mongolia, 010000, China
E-mail: hejingman0128@163.com, sdgsg454ed234@163.com, pengjie_job@sina.com, zhoulin20250809@163.com, wangliweihk2008@163.com
*Corresponding author

Keywords: data correlation, cluster load balancing, metadata, metadata migration management

Received: August 15, 2025

Against the backdrop of the rapid development of information technology, the in-depth integration of big data, artificial intelligence, and cloud computing has driven explosive growth in data volume. Distributed storage systems have become core infrastructure, and the efficiency and reliability of metadata management play a decisive role in system performance. This study proposes a metadata migration management architecture based on data correlation and cluster load balancing. By constructing a data correlation graph and a multi-objective optimization model, it realizes the intelligent optimization of metadata layout. This method comprehensively considers access coordination, load balancing, and migration costs; meanwhile, it introduces a bandwidth-aware mechanism and a data popularity prediction mechanism to improve dynamic adaptability. Experimental verification shows that this algorithm can effectively improve system performance and resource utilization. Experimental results show that in scenarios of high-frequency synergetic access and periodic large-scale access, the algorithm achieves a hot metadata recognition accuracy of 93.5%; it mitigates the average metadata access delay to 7.4 milliseconds, and shortens the migration trigger response time to 0.15 seconds. All performance indicators outperform traditional methods such as hash mapping and round-robin migration. The innovation of this study lies in the first deep coupling of metadata semantic association with dynamic cluster load balancing, breaking through the single-dimensional limitations of traditional strategies. The study provides a more intelligent and efficient solution for metadata management in distributed storage systems, effectively improving the system's overall service quality and scalability. This has important theoretical and application value for promoting the evolution of distributed storage towards intelligence and adaptability.

Povzetek: Študija predlaga pametnejši pristop za upravljanje in selitev metapodatkov v porazdeljenih shranjevalnih sistemih, ki izboljša zmogljivost, uravnoteženost obremenitve in odzivnost sistema.

1 Introduction

In the context of the continuous growth of information technology, the swift integration of artificial intelligence, cloud computing, and big data has spawned explosive growth in data volume. These technologies drive distributed storage systems to become indispensable infrastructure supporting modern computing platforms [1]. More and more systems adopt distributed file system architectures to meet the needs of high-performance, highly available, and highly scalable management of massive data. Thus, storage resources are horizontally expanded to multiple physical or virtual nodes to improve overall throughput and processing capabilities [2]. In this process, metadata management has gradually become a key factor affecting system performance. Metadata records basic attribute information of files; it also undertakes multiple core tasks such as directory structure organization, access control, and data addressing, with its access frequency much higher than that of actual data files [3].

In existing distributed metadata management systems, common strategies include static partitioning, hash distribution, and dynamic migration based on access load. These methods have alleviated the hot issue of metadata access to a certain extent and improved the utilization efficiency of system resources [4]. However, with the diversification of application scenarios and the complication of access behaviors, strategies that rely solely on access frequency or node load for migration scheduling have gradually exposed limitations. Traditional migration mechanisms often ignore the semantic associations and access correlations between metadata when evaluating migration benefits [5]. Frequent migration operations can also bring additional resource consumption and system instability, especially in environments with large load fluctuations or strong access burstiness. Migration strategies lacking global collaboration and dynamic evaluation mechanisms fail to achieve true load balancing and may induce unbalanced use of node resources [6-7].

This study addresses the fundamental research question of "how to maintain and optimize access relevance between metadata while simultaneously ensuring efficient load balancing in distributed storage systems". This is a topic that demonstrates profound innovative value and theoretical significance. On the one hand, conventional migration strategies predominantly focus on basic node load balancing or simplistic access frequency optimization, while systematically overlooking the inherent logical relationships and access coupling patterns among metadata. This critical oversight frequently results in operational failures during metadata migration processes. Particularly in complex multi-user, multi-task concurrent environments, it manifests as migration interruption, path access delay escalation, and overall system performance degradation [8]. On the other hand, combining semantic understanding capabilities with system-level scheduling strategies represents one of the most promising frontiers in contemporary distributed storage research. Applying this integrated approach to metadata management systems achieves dual academic advancements; it extends the theoretical framework of migration optimization strategies by introducing semantic-aware dimensions while establishing foundational methodologies for developing next-generation intelligent storage systems with self-learning capabilities [9]. This dual contribution positions the study at the forefront of theoretical exploration and practical system development. Therefore, the research topic responds to the practical needs of performance evolution in distributed systems. The goals of this study are to achieve the following quantitative research objectives through multi-objective optimization. First, it minimizes the node load variance to promote the balanced distribution of cluster resources. Second, it reduces the average access delay to optimize system response performance. Finally, it improves the recognition accuracy of hot metadata to ensure the accuracy and efficiency of metadata migration decisions.

2 Related work

In recent years, the swift development and widespread adoption of big data technologies across industries have led to an unprecedented explosion of massive unstructured data. This exhibits exponential growth in various distributed storage systems. As a fundamental supporting module that directly determines system performance and scalability, metadata management has consequently attracted growing attention from an increasing number of researchers in both academic and industrial domains [10].

Researchers have conducted extensive investigations into metadata migration strategies, with most efforts concentrating on two primary aspects: load balancing and system performance optimization. Scholars proposed a metadata dynamic migration algorithm based on access frequency statistics. By periodically analyzing access logs of each node, the algorithm realized the migration of hot metadata to alleviate the problem of single-node overload [11]. Such methods improved the throughput capacity of the system to a certain extent. However, these methods showed a lag in responding to changes in access behavior, which easily caused the failure of migration strategies when the load changed suddenly [12]. Moreover, some scholars designed a multi-factor dynamic migration model based on central processing unit (CPU) occupancy and memory pressure using the Hadoop Distributed File System (HDFS); this further enhanced the ability to perceive the resource status of nodes [13].

Regarding combining metadata migration with system load balancing, load-aware scheduling mechanisms have become one of the key research directions. Scholars showed that scheduling relying solely on a single load indicator (such as CPU, Input/Output (I/O), or network bandwidth) easily fell into a local optimum. It was necessary to comprehensively consider the multi-dimensional resource occupancy status of nodes and introduce dynamic thresholds and feedback adjustment mechanisms to realize real-time optimization of migration strategies [14]. Some researchers pointed out that there was a non-linear correlation between the distribution status of metadata and the overall scheduling performance of the system. Meanwhile, a more accurate performance evaluation model should be established to guide migration behaviors [15]. In addition, some scholars proposed a metadata scheduling framework based on reinforcement learning, which could self-adjust and optimize migration strategies according to system status and historical effects. This reflected an in-depth understanding of metadata management issues in complex dynamic environments [16].

To sum up, the current field of metadata migration management has achieved substantial theoretical and practical advancements through extensive investigations into three key technical dimensions: load perception, access frequency evaluation, and scheduling optimization. These findings have collectively contributed to remarkable progress. However, most studies still do not fully integrate the semantic relationships and access behavior characteristics between metadata, lacking an overall perspective oriented to data organization structure. Consequently, developing a metadata migration strategy requires integrating data correlation analysis to perceive access behavior structures while incorporating real-time cluster load status monitoring. This combined dynamic scheduling optimization method is an important extension of the current research framework of distributed storage systems.

3 The metadata migration management architecture based on data correlation and cluster load balancing strategies

3.1 Definition of data correlation and cluster load balancing

Data correlation and cluster load balancing strategies constitute critical components of metadata migration

management in distributed storage systems. These embody dual considerations for both data organizational structures and system resource distribution [17]. The data correlation strategy forms the intellectual core of intelligent metadata management by systematically identifying, analyzing, and exploiting the inherent relationships that exist among various metadata elements. These relationships manifest through multiple dimensions (access pattern commonality, semantic structure similarity, or directory hierarchy proximity) that reflect different aspects of data usage and organization within the system [18]. The cluster load balancing strategy addresses resource utilization across storage nodes, including CPU, memory, network I/O, and request processing capabilities [19-20].

In metadata migration management, the data correlation strategy describes the access relationships between metadata by constructing a weighted association graph [21-22]. The association strength w_{ij} between two metadata items, m_i and m_j, is usually defined as the weighted sum of access co-occurrence frequency and directory path similarity:

$$w_{ij} = \alpha \cdot \frac{c_{ij}}{c_i - c_j} + \beta \cdot sim(p_i, p_j) \quad (1)$$

Symbol interpretation:

m_i and m_j: Two metadata items;

c_{ij} : The number of times the metadata m_i and m_j are jointly accessed by the same request or transaction;

c_i and c_j: The total number of visits for each of the m_i and m_j;

p_i and p_j: The directory path where the metadata m_i and m_j are located;

$sim(p_i,p_j)$: The directory path similarity function measures the degree of similarity between two paths;

α and β: The weight coefficient, which satisfies $\alpha+\beta=1$, is used to balance the importance of the two indicators;

In Equation (1), the correlation strength between two metadata items (m_i and m_j) is formed by the weighted combination of access co-occurrence frequency and directory path similarity. The directory path similarity function $sim(p_i,p_j)$ can be defined through precise string or structural distance metrics. For example, the Levenshtein edit distance or tree edit distance is used to calculate the difference between two directory paths, and a normalization operation maps this difference to the interval [0,1]. This ensures that the correlation values, after weighting by similarity and access frequency, are comparable. The weight coefficient (α) balances the relative importance of access frequency and directory similarity, and it can be adjusted heuristically or optimized through learning based on system access patterns or historical data.

To measure the resource load of each node in the cluster, the comprehensive load of node s_j is defined as the weighted sum of the utilization of various resources:

$$Ls_j = \omega_1 \cdot CPU(s_j) + \omega_2 \cdot MEM(s_j) + \omega_3 \cdot IO(s_j) + \omega_4 \cdot RQ(s_j) \quad (2)$$

Symbol interpretation:

s_j: The j-th node in the cluster;

$CPU(s_j), MEM(s_j), IO(s_j)$: CPU, Memory (MEM), and IO resource utilization of node s_j (generally values range from 0 to 1);

$RQ(s_j)$: The request quantity (RQ) of metadata processed by node s_j per unit time;

$\omega_1, \omega_2, \omega_3, \omega_4$: The weight coefficient of each resource indicator, which is used to reflect its contribution to the comprehensive load;

In Equation (2), first, the relative importance of each resource in the comprehensive load is initially set according to system performance monitoring and resource usage characteristics. For instance, CPU plays a dominant role, followed by memory and I/O, with the number of requests as a supplementary factor. Second, historical load data is used to iteratively adjust or heuristically optimize these initial weights, enabling the calculated comprehensive load to accurately reflect node bottlenecks. This achieves an effective trade-off between load balancing and performance optimization in migration decisions. The weight coefficients for the node's comprehensive load can be set as w_1=0.35 (CPU), w_2=0.25 (MEM), w_3=0.25 (IO), and w_4=0.15 (RQ). This setting reflects the core role of CPU load in the overall node pressure while considering the non-negligible impact of memory, I/O, and the number of requests on the load.

To evaluate the balance of the overall load of the cluster, calculate the variance of the load on all nodes:

$$Var(L) = \frac{1}{N}\sum_{j=1}^{N}\left(L(s_j) - \bar{L}\right)^2 \quad (3)$$

Symbol interpretation:

N: The total number of nodes in the cluster;

$L(s_j)$: The comprehensive load of node s_j;

$\bar{L} = \frac{1}{N}\sum_{j=1}^{N} L(s_j)$: The average value of the combined load of all nodes;

The two goals of data correlation and load balancing are combined to form a unified optimization function:

$$\min\left[\lambda \cdot Cut(G,P) + (1-\lambda) \cdot Var(L)\right] \quad (4)$$

Equation (4) integrates the two objectives of data correlation and load balancing into a unified optimization function. By balancing the collaborative access relationship between data and the load distribution of each node, it achieves a globally optimal data partitioning that maintains high access locality and sustains system load balancing. The sum of associated edge weights across nodes is defined as:

$$Cut(G,P) = \sum_{\substack{(m_i,m_j)\in E \\ loc(m_i)\neq loc(m_j)}} w_{ij} \quad (5)$$

Symbol interpretation (Overall optimization objective):

$\lambda \in [0,1]$: The weight parameter that regulates the importance of data correlation and load balancing;

$P = \{P_1, P_2, \ldots, P_N\}$: Metadata partitioning scheme; P_j represents the set of metadata assigned to node s_j;

$Cut(G, P)$: The sum of cross-node edge weights, that is, the sum of all metadata association weights distributed across nodes, reflecting the cost of access fragmentation caused by data migration;

E: A collection of edges in a metadata association graph;

$loc(m_i)$: The node where the metadata m_i is located.

3.2 The metadata migration management algorithm

The metadata migration management algorithm aims to optimize the storage layout of metadata in distributed storage systems to improve overall system performance and resource utilization efficiency. With the continuous development of large-scale data systems, the amount of metadata has increased sharply, and access patterns have become complex and variable. How to reasonably distribute metadata across cluster nodes has become a key issue [23]. The metadata migration management algorithm's tool architecture is displayed in Figure 1:

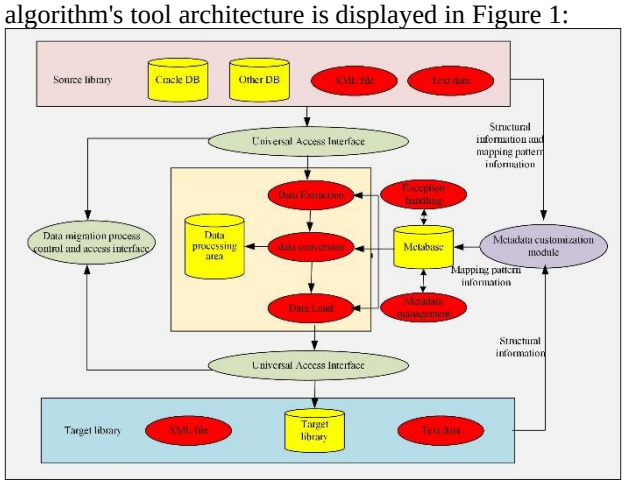

Figure 1: The tool architecture of the metadata migration management algorithm

The metadata support metamodel emphasizes the basic structural support capability in the migration process, focusing on the logical relationships in links such as metadata extraction, mapping, transformation, and loading. Through core modules such as extraction components, transformation rules, loading processes, and log records, it ensures that metadata is migrated accurately and safely between different systems. At the same time, it maintains semantic consistency and traceability [24], as presented in Figure 2:

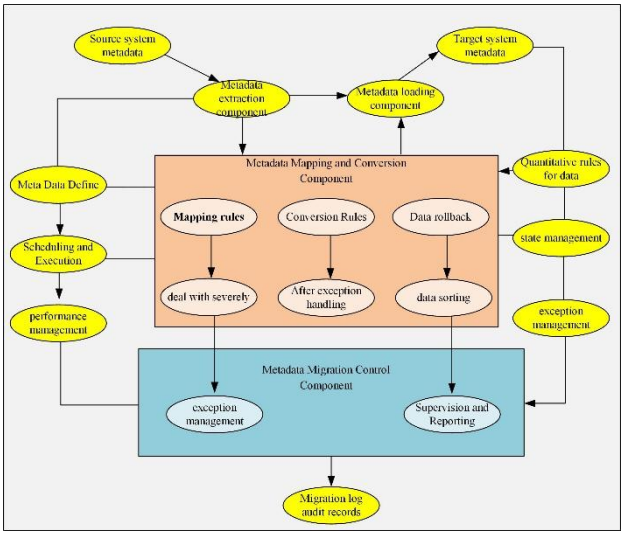

Figure 2: The metadata support metamodel of the metadata migration management

The metadata-driven metamodel takes "configuration rules drive the migration process" as its core idea. This emphasizes the use of logic, such as scheduling strategies, execution plans, and audit rules, to automatically control the entire migration process. Its essence is an architecture centered on control logic, enabling the migration process to have a high degree of flexibility and orchestration [25], as illustrated in Figure 3:

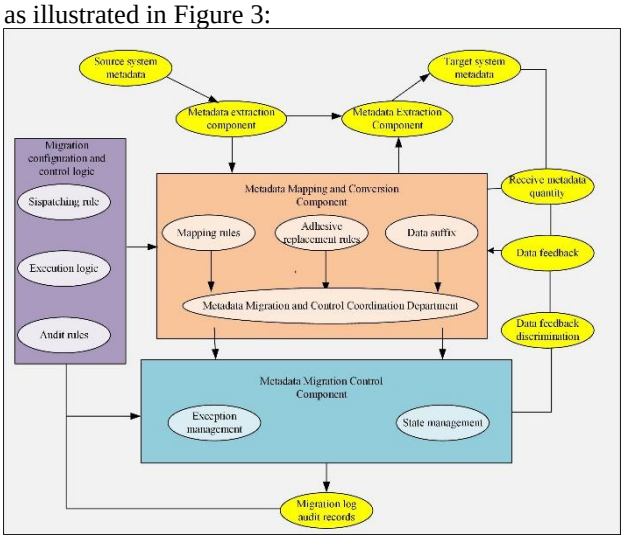

Figure 3: The metadata-driven metamodel of the metadata migration management

(1) Minimization of migration cost (transmission cost)

Each migration operation consumes network bandwidth and time resources when executing metadata migration. Therefore, the system overhead caused by migration must be considered, as follows:

$$C_{total} = \sum_{(m_i, m_j) \in M} \frac{s_m}{b_{i,j}} \quad (6)$$

Symbol interpretation:

C_{total}: Total migration cost;

M: A collection of migration plans containing all metadata m from node i to node j;

S_m: The size of the metadata m;

$b_{i,j}$: Network bandwidth (transmission capacity per unit time) between node i to node j;

$\frac{s_m}{b_{i,j}}$: The time or cost required to migrate metadata m.

(2) Load balancing objective (balancing node load)

To prevent some nodes from becoming performance bottlenecks due to excessive load, the migration algorithm should try to keep the load of each node close to the average value. The expression can be written as follows:

$$L_{imbalance} = \sum_{i=1}^{N} \left| L_i^{new} - \bar{L} \right| \tag{7}$$

Symbol interpretation:

$L_{imbalance}$: Load imbalance (sum of deviations from the average load);

N: The total number of nodes in the system;

L_i^{new}: The total load of node i after migration;

$\bar{L} = \frac{1}{N} \sum_{i=1}^{N} L_i^{new}$: The average load of all nodes after migration;

$\left| L_i^{new} - \bar{L} \right|$: The deviation of node load from the average load.

(3) Minimization of access delay

This objective function comprehensively considers the user's access frequency and the network distance of the access path, aiming to optimize the data location and reduce the overall access delay. The equation is as follows:

$$D_{avg} = \sum_{m \in M} \sum_{u \in U_m} f_{m,u} \cdot d_{u,n(m)} \tag{8}$$

Symbol interpretation:

D_{avg}: System average access delay;

M: Metadata collection;

U_m: A collection of users who access metadata m;

$f_{m,u}$: The frequency of user u access to metadata m;

$d_{u,n(m)}$: The network distance or delay between the user u and the node $n(m)$ where the metadata m is stored;

$n(m)$: The node number of the metadata m is stored.

(4) Maximization of local access hit rate

The algorithm should prioritize migrating metadata to the node where its visitors are located or adjacent nodes to improve the "local hit rate." Equation (9) calculates the total frequency of users accessing metadata on their local nodes.

$$A_{locality} = \sum_{m \in M} \sum_{u \in U_m} 1_{n(m)=n(u)} \cdot f_{m,u} \tag{9}$$

Symbol interpretation:

$A_{locality}$: Total value of local access hit rate;

$1_{n(m)=n(u)}$: The indicator function, which is 1 when user u and metadata m are on the same node, otherwise 0;

$f_{m,u}$: Frequency of user u accessing metadata m;

$n(u)$: Node to which user u belongs;

$n(m)$: Node where metadata m is located.

(5) Storage capacity constraint

Before receiving migrated metadata, the target node must check whether its remaining capacity is sufficient to store the data to avoid overflow. The migration strategy must meet the maximum capacity constraint of each node to prevent failures or data loss due to overload, as shown in Equation (10):

$$\sum_{u \in M_j^{new}} s_m \le C_j \, \forall j \in \{1, \ldots, N\} \tag{10}$$

Symbol interpretation:

M_j^{new}: Set of metadata located on node j after migration;

S_m: Size of metadata m;

C_j: Maximum available capacity of node j;

The inequality indicates that the total metadata size on node j must not exceed its capacity C_j.

(6) Single storage location constraint (exclusivity)

This constraint ensures that metadata is not copied to multiple nodes (non-replica scenarios), thereby avoiding data redundancy or conflicts.

$$\sum_{j=1}^{N} x_{m,j} = 1 \forall m \in M \tag{11}$$

Symbol interpretation:

$x_{m,j} \in \{0,1\}$: Boolean variable indicating whether metadata m is on node j;

N: Total number of nodes;

M: Set of metadata;

Equation (11) indicates that each metadata m can only be located on one node at any time.

(7) Comprehensive optimization objective function (weighted sum)

On the basis of the above objective functions, a comprehensive objective function is introduced. This function weights and sums each sub-objective, and dynamically adjusts the weights according to specific system priorities.

$$\min_{M} \left(\alpha \cdot C_{total} + \beta \cdot L_{imbalance} + \gamma \cdot D_{avg} - \delta \cdot A_{locality} \right) \tag{12}$$

Symbol interpretation:

α, β, γ, an δ: Four non-negative weight coefficients used to adjust the weight of different sub-objectives in the overall optimization;

C_{total}: Migration cost;

$L_{imbalance}$: Load imbalance degree;

D_{avg}: Average access delay;

$A_{locality}$: Local access hit amount (the larger the better, hence the negative sign);

The optimized output of this objective function is the optimal migration plan M.

Based on the learning-based adaptive adjustment method, the initial weights can be set as follows: α (migration cost) = 0.3, β (load imbalance degree) = 0.4, γ (average access delay) = 0.2, and λ (local access hit count) = 0.1. These weights are then dynamically fine-tuned according to the online monitored system indicators. For

example, when large fluctuations in node load are detected, the weight of β can be appropriately increased to strengthen load balancing; if access delay rises, the weight of γ is increased to prioritize optimizing response speed, thereby achieving multi-objective balance.

3.3 The metadata migration management algorithm using data correlation and cluster load balancing strategy

The metadata migration management algorithm, based on a cluster load balancing strategy and data correlation, presents an intelligent migration mechanism. By analyzing the business relevance between metadata, it packages and migrates strongly correlated data units to reduce communication and dependency overhead between systems. At the same time, it dynamically schedules migration tasks to optimal nodes in combination with the current load status of the cluster, achieving resource balance and maximizing migration efficiency [26]. The metadata migration management architecture based on data correlation and cluster load balancing strategies is illustrated in Figure 4:

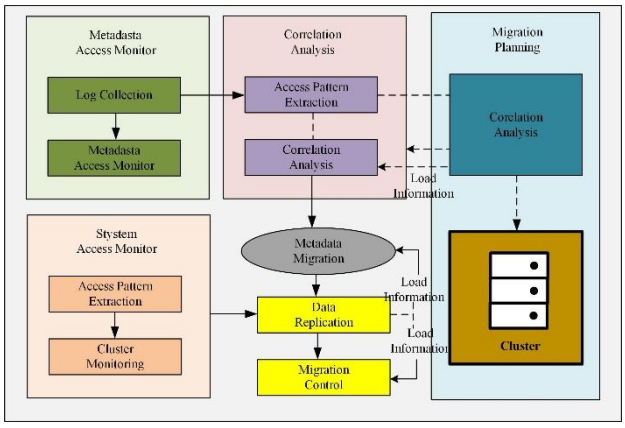

Figure 4: Metadata migration management architecture using data correlation and cluster load balancing

The equation for the data correlation and cluster load balancing strategy-based metadata migration management algorithm reads:

(1) Data correlation weight modeling based on access synergy

$$w_{mi,mj} = \sum_{u \in U} \frac{f_u(m_i, m_j)}{T} \tag{13}$$

$w_{mi,mj}$: Synergetic access weight between metadata m_i and m_j;

U: User set;

$f_{u(mi,mj)}$: Number of times user u accesses m_i and m_j simultaneously within unit time;

T: Length of statistical time window;

The weight is used to construct the edge weight of the access synergy graph for graph clustering or partitioning.

The time window T used in Equation (13) is a sliding window. It dynamically calculates the collaborative access weight between metadata by counting the number of co-accesses by users in the most recent T unit time; this makes it adaptable to workload changes and access pattern fluctuations over time. This sliding mechanism ensures that the collaborative weight reflects the latest access behavior promptly and continuously captures the dynamic characteristics of hotspots and access correlations.

(2) Target function for correlation subgraph partitioning

$$\min_{\{P_k\}} \sum_{k=1}^{K} \sum_{m_i \in P_k, m_j \notin P_k} w_{mi,mj} \tag{14}$$

$\{P_k\}$: The k-th metadata subset after partitioning (corresponding to the migration target node);

$w_{mi,mj}$: Synergetic access weight between metadata;

The inner summation represents the total weight of cut edges between subset P_k and other subsets.

The overall goal is to concentrate data with high-frequency synergetic access in the same partition.

(3) Cluster load normalization evaluation function

$$\sigma_L = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (L_i - \bar{L})^2} \tag{15}$$

σ_L: Standard deviation of node load (measuring balance);

N: Total number of nodes in the cluster;

L_i: Current load of the i-th node (such as access volume, I/O volume, or metadata volume);

$\bar{L}$: Average load of all nodes;

The goal of this function is to minimize to achieve load balancing.

(4) Load-aware data migration cost function

$$C_{m,j} = \lambda \frac{S_m}{b_{i,j}} + (1-\lambda)\left(1 - \frac{R_{m,j}}{R_{\max}}\right) \tag{16}$$

$C_{m,j}$: Cost of migrating metadata m to node j;

S_m: Size of metadata m;

$b_{i,j}$: Bandwidth from source node i to target node j;

$R_{m,j}$: Total correlation between metadata m and other metadata currently on node j;

R_{max}: Maximum possible correlation (for normalization);

$\lambda \in [0,1]$: An adjustable weight parameter, adjusting the weight of bandwidth cost and correlation aggregation;

The goal is to select the node with the smallest $C_{m,j}$ as the migration target.

Equation (16) refers to a function for calculating metadata migration cost considering bandwidth and correlation. Its core idea is to comprehensively balance the transmission overhead caused by metadata size and network bandwidth, as well as the correlation between the metadata and existing data on the target node. It selects the node with the smallest migration cost as the target, thus reducing migration costs while maintaining the correlated aggregation of data.

(5) Overall optimization goal (synergistic integration of load and correlation)

$$\min\left(\alpha \cdot \sigma_L + \beta \cdot \sum_{m \in M} C_{m,n(m)} + \gamma \cdot \sum_{k=1}^{N} \sum_{m_i, m_j \in P_k} w_{mi,mj}\right) \tag{17}$$

σ_L: Standard deviation of node load;

$C_{m,n(m)}$: Cost of migrating metadata m to target node $n(m)$;

P_k: Set of metadata stored on node k;

$w_{mi,mj}$: Synergetic access weight between metadata;

α, β, and γ: Weight parameters, used to adjust the relative importance of the three goals;

The goal is to minimize load imbalance and migration cost while maximizing access synergy.

Equation (17) represents the objective function for comprehensive optimization. It achieves load balancing in the system and reduces migration overhead by minimizing load imbalance between nodes and metadata migration costs, while maximizing the collaborative access degree between metadata. At the same time, it improves access locality and overall performance.

(6) Node capacity and load constraints

$$\sum_{m \in M_j} s_m \le C_j, \forall j \tag{18}$$

$$\sum_{m \in M_j} a_m \le A_j, \forall j \tag{19}$$

M_j: Set of metadata on node j;

S_m: Size of metadata m;

C_j: Maximum capacity of node j;

a_m: Access frequency of metadata m per unit time;

A_j: The maximum access pressure that node j can accept (such as the IOPS upper limit);

All migration plans must satisfy the above two constraints simultaneously.

(7) Data popularity prediction

$$h_m^{t+1} = \eta \cdot h_m^t + (1-\eta) \cdot a_m^t \tag{20}$$

h_m^{t+1}: The popularity of the metadata m for the next period is predicted;

h_m^t: Current popularity;

a_m^t: Current actual access frequency;

$\eta \in [0,1]$: Attenuation factor (the higher the value, the more emphasis is placed on historical trends).

Equation (20) uses exponential smoothing technology to predict metadata popularity. It realizes fast estimation of popularity in the next period by weighted fusion of the current access frequency and historical popularity according to a decay factor. This method can efficiently respond to dynamic changes in access patterns and sudden loads, as it is sensitive to recent access behavior and has low computational overhead. It is suitable for real-time updating of metadata migration strategies in large-scale distributed systems. Compared with complex models such as ARIMA or LSTM, its advantage lies in the ability to quickly adapt to variable access traffic without requiring a large amount of training, while maintaining the smoothness and stability of predictions.

The pseudocode of the proposed metadata migration management algorithm is as follows:

```
# Input: Metadata set M, node set N, user set U
# Output: Optimal migration plan migration_plan
# 1. Calculate collaborative access weights
for mi in M:
    for mj in M:
        if mi != mj:
            co_access_count = sum([1 for u in U if u.accessed(mi) and u.accessed(mj) in window_T])
            sim_score = compute_path_similarity(mi.path, mj.path) # e.g., normalized Levenshtein or tree-edit distance
            weight_alpha = 0.5  # weight, adjustable
            weight_beta = 0.5
            co_access_weight[mi][mj] = weight_alpha * (co_access_count / total_access(mi, mj)) + weight_beta * sim_score
# 2. Metadata subgraph partitioning
partitions = cluster_metadata_graph(M, co_access_weight, num_partitions=len(N))
# 3. Calculate the standard deviation of node load
def compute_load_std(nodes):
    loads = [node.load for node in nodes]
    mean_load = mean(loads)
    std = sqrt(sum((l - mean_load)**2 for l in loads) / len(nodes))
    return std
# 4. Calculate the migration cost
def compute_migration_cost(mi, target_node):
    size_factor = mi.size / network_bandwidth[mi.current_node][target_node]
    assoc_factor = 1 - sum(co_access_weight[mi][mj] for mj in target_node.metadata) / max_assoc
    cost = alpha * size_factor + beta * assoc_factor
    return cost
# 5. Comprehensive optimization
for partition in partitions:
    for node in N:
        cost = compute_migration_cost(partition.metadata, node)
        load_std = compute_load_std(N)
        access_collab_score = sum(co_access_weight[mi][mj] for mi, mj in combinations(node.metadata, 2))
        objective = gamma1 * cost + gamma2 * load_std - gamma3 * access_collab_score
        if satisfies_capacity_and_load_constraints(node, partition):
            migration_plan.assign(partition, node, objective)
# 6. Node capacity and load constraints
def satisfies_capacity_and_load_constraints(node, partition):
    total_size = sum(mi.size for mi in node.metadata) + sum(mi.size for mi in partition.metadata)
    projected_load = node.current_load + sum(mi.access_frequency for mi in partition.metadata)
    return total_size <= node.max_capacity and projected_load <= node.max_IOPS
# 7. Data popularity prediction
for mi in M:
    mi.next_heat = decay_factor * mi.current_heat + (1 - decay_factor) * mi.current_access_frequency
# 8. Output the optimal migration plan
return migration_plan
```

4 The application of the metadata migration management algorithm by the data correlation and cluster load balancing strategy

All data used in this study are derived from access behavior sampling and simulated task loads in the actual operation and maintenance as well as business systems of XX Enterprise. Specifically, the data include three parts. The first is the job execution logs, covering Spark, Hive analysis jobs, and machine learning training tasks running on HDFS. The second is the system operation and access behavior logs, including records of file and directory creation, modification, and deletion, as well as the access frequency and node distribution of metadata by users or applications. The third is resource monitoring logs, encompassing the usage of CPU, memory, network, and disk I/O. The overall data scale reaches the terabyte (TB) level, with hundreds of millions of metadata records, and the number of files and directories ranges from millions to tens of millions. To ensure the reproducibility of experiments, part of the workload is simulated through Filebench and a self-developed testing tool of the enterprise. Meanwhile, the system operation status is obtained through regular sampling by Prometheus. All experiments are repeated five times, with the average value calculated and standardized to ensure the robustness and comparability of the results. The experimental configuration is exhibited in Table 1:

Table 1: Experimental configuration

Experimental elements	Specific settings
Number of Cluster nodes	20–50odes
Metadata object types	HDFS files, directories, Hive tables, Spark job outputs
Metadata object sizes	Small files: 1–100KB; Medium files: 1–500MB; Large files: 1–10GB
Workload composition	Actual access logs 70%, synthetic/simulated load 30%
Simulation tools	Filebench + enterprise self-developed testing tool
Filebench Configuration	Hot file bursts (accounting for about 10–20% of total file access), directory-level collaborative access (local access frequency 3–5 times higher than the global average)
Monitoring sampling tools	Prometheus
Sampling indicators	CPU usage, memory usage, IO utilization, number of metadata requests, node load, etc.
Number of experiment repetitions	5 times
Data processing	Mean calculation and normalization processing

The proposed algorithm selects Hash Map and Polling Migration as controls, as both are representative in distributed metadata management. Hash Map performs hash operations on metadata key-values to map data evenly to cluster nodes, achieving simplicity with no additional scheduling overhead [27]; Polling Migration periodically scans node loads and migrates hot or high-frequency accessed metadata to nodes with lower loads, to alleviate performance bottlenecks and improve access balance [28]. These two methods are representative and easy to reproduce in distributed metadata management, and their performance is similar to that of the algorithm in this study, making them more comparable. More complex optimization algorithms, however, are rarely applied in actual enterprise environments and thus are not included in the comparison.

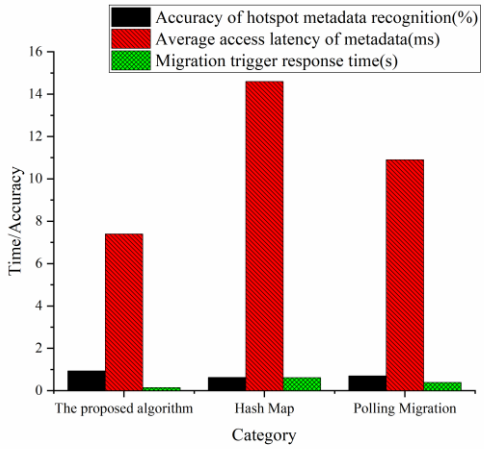

Figure 5: Performance comparison of different algorithms in high-frequency synergetic access scenarios

Figure 5 shows that the proposed algorithm performs advantages in high-frequency synergetic access scenarios. Its hot metadata recognition accuracy reaches 93.5%, far exceeding 62.7% of hash mapping and 70.2% of round-robin migration; this indicates that the proposed algorithm can more accurately capture access rules and optimize resource scheduling. Regarding access efficiency, the average metadata delay of this algorithm is only 7.47.4 milliseconds (ms), which is 49.3% and 32.1% lower than that of hash mapping and round-robin migration B, increasing the response speed. In addition, the migration trigger response time of the proposed algorithm (0.15 seconds (s)) is 75.8% and 61.5% shorter than that of hash mapping (0.62s) and round-robin migration (0.39s), proving its agility in dynamic load adjustment.

Table 2: Performance comparison of different algorithms in high-frequency synergetic access scenarios

Method	Accuracy (%)	Average access delay (ms)	Migration response time(s)	Variance σ^2	95% confidence interval (CI)
The propo	93.50	7.4	0.15	0.82	92.9–94.1

sed algorithm					
Hash Map	62.70	14.6	0.62	1.24	61.4–63.9
Polling Migration	70.20	10.9	0.39	0.97	69.5–71.0

In Table 2, the proposed algorithm achieves a hot metadata recognition accuracy of 93.50%, which is higher than that of Hash Map's 62.70% and Polling Migration's 70.20%; its average access delay and migration response time are also reduced to 7.4 ms and 0.15 s, respectively, demonstrating advantages over other algorithms. The variance and 95% CI reflect the algorithm's robustness under different load conditions, among which the proposed algorithm performs the most stably.

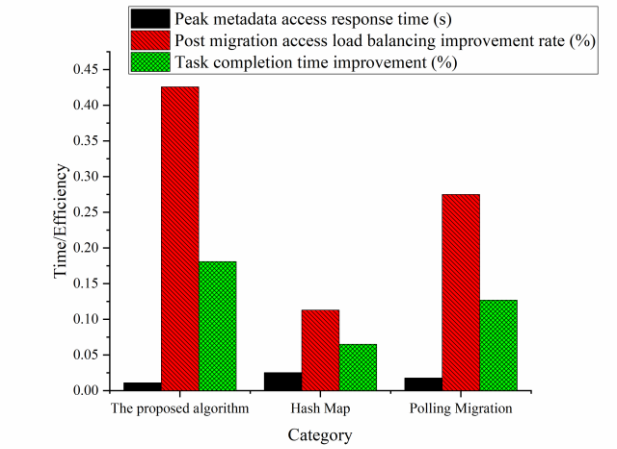

Figure 6: Comparison of diverse algorithms for periodic large-scale access task scheduling scenarios

In Figure 6, the proposed algorithm demonstrates improved performance in periodic large-scale access task scheduling scenarios. The algorithm's peak metadata access response time is only 0.0112 seconds, 55.9% and 37.1% lower than hash mapping and round-robin migration. Regarding load balancing performance, this algorithm increases the access balance rate after migration by 42.6%, which is 3.77 times and 1.55 times that of hash mapping and round-robin, respectively. At the same time, the task completion time is improved by 18.1%, and the improvement range exceeds that of the two baseline algorithms by 48.6% to 178%. Experimental data reveal that through the collaborative optimization of association rules and dynamic load balancing, this algorithm achieves breakthrough improvements in three dimensions: system response speed, resource allocation balance, and task processing efficiency. Thus, these can effectively meet the performance requirements in enterprise-level high-concurrency scenarios.

Table 3: Performance comparison of different algorithms in periodic large-scale access task scheduling scenarios

Method	Peak latency (s)	Load increase (%)	Completion time increase (%)	Variance σ^2	95% CI
The proposed algorithm	0.0112	42.6	18.1	0.000014	40.9–44.3
Hash Map	0.0254	11.3	6.5	0.00009	9.6–13.1
Polling Migration	0.0178	27.5	12.7	0.00004	25.3–29.6

In Table 3, the proposed algorithm achieves 0.0112s, 42.6%, and 18.1% in peak access latency, load balance improvement rate, and task completion time improvement rate, respectively, all of which are better than those of Hash Map and Polling Migration. The variance and 95% CI indicate that the proposed algorithm has higher stability and reliability in dynamic task scheduling scenarios.

5 Discussion

The proposed metadata migration management algorithm integrates data correlation with cluster load balancing principles. It outperforms traditional Hash Map and Polling Migration algorithms in scenarios involving high-frequency synergetic access and periodic large-scale access. Experiments show that the algorithm achieves a hot metadata recognition accuracy of 93.5%, which is 49.1% and 33.2% higher than the latter two methods; the average metadata access dealy is 7.4 ms, with a reduction range of 32.1%–49.3%; the migration trigger response time is 0.15 s, shortened by 61.5%–75.8%; in periodic scenarios, the peak access latency is 0.0112 s, reduced by 37.1%–55.9%, and the load balance improvement rate is 42.6%, which is 1.55–3.77 times that of traditional methods. The advantages stem from the algorithm breaking through the limitations of traditional single-dimensional approaches. The proposed algorithm realizes the aggregation of strongly correlated metadata through a weighted correlation graph to avoid access fragmentation; it combines node comprehensive load calculation and dynamic variance evaluation to solve the problems of insufficient adaptation of static mapping and delayed response of periodic scanning; thus, it achieves the coordination of access optimization and load balancing. This algorithm provides an adaptive solution for distributed storage metadata management and is especially suitable for enterprise-level complex concurrent scenarios. Future work should optimize the weight adaptation of heterogeneous clusters and the

efficiency of large-scale metadata correlation subgraph partitioning to improve its universality and engineering value.

6 Conclusion

This study focuses on the issue of metadata management in distributed storage systems and proposes a migration management optimization algorithm based on data correlation and cluster load balancing strategies. In high-frequency synergetic access scenarios, the proposed algorithm can accurately identify strongly correlated metadata under the same task or directory. Concurrently, it reduces cross-node access delay and improves the efficiency of multi-file joint access in big data analysis platforms by aggregating storage. Its hot spot recognition accuracy (93.5%) and average delay (7.4ms) can support the real-time response requirements of high-concurrency services. In periodic large-scale access tasks, the algorithm effectively alleviates node overload through dynamic load sensing and associated data aggregation. The 42.6% improvement rate in load balancing and shorter task completion time make it suitable for periodically fluctuating scenarios such as machine learning training tasks and log batch processing, ensuring system stability during load peaks. In enterprise-level distributed storage systems, the proposed algorithm can optimize the dynamic distribution of metadata, reduce scheduling delays caused by data dispersion, and improve storage resource utilization. It is particularly suitable for complex environments with multi-user and multi-task concurrency, providing technical support for efficient operation and maintenance of infrastructure such as cloud storage and data centers.

The limitations of this study are mainly reflected in the fact that most experimental data are based on the actual operation and maintenance environment and simulated loads of a single enterprise. The number of cluster nodes, hardware configuration, and network topology are fixed in the experiment, so the universality of the results may be limited. Its adaptability in heterogeneous clusters, extreme load fluctuations, or multi-cloud environments has not been fully verified; the algorithm may need further adjustment in edge scenarios or under extreme real-time constraints. The dynamic adjustment mechanism of weight coefficients in data correlation modeling still needs to be improved; the mining of implicit correlations across tasks and directories is not in-depth enough, and the potential of "semantic correlation development" has not been fully exploited. To enhance the generalization ability and practicality of the study, future work can conduct multi-load mode tests under different hardware specifications and cluster topologies; deep learning is introduced to optimize the dynamic adaptation ability of correlation weights to improve the accuracy of complex semantic correlation perception; the algorithm's application in heterogeneous cloud environments is expanded to enhance its universality; real-time stream processing technology is combined to reduce the response latency of migration decisions, realizing more refined dynamic scheduling, thus improving overall performance and system stability.

References

[1] Zhu Y, Xia T, Zhu T, et al. RAPO: An Automated Performance Optimization Tool for Redis Clusters in Distributed Storage Metadata Management[J]. IEEE Access, 2025, (13): 58060- 58074. DOI:10.1109/access.2025.3556240

[2] Huang X, Gao Y, Zhou X, et al. An adaptive metadata management scheme based on deep reinforcement learning for large-scale distributed file systems[J]. IEEE/ACM Transactions on Networking, 2023, 31(6): 2840-2853. DOI:10.1109/tnet.2023.3266400

[3] Singh H J, Bawa S. Scalable metadata management techniques for ultra-large distributed storage systems--A systematic review[J]. ACM Computing Surveys (CSUR), 2018, 51(4): 1-37. DOI:10.1145/3212686

[4] Eickhoff T, Eiden A, Göbel J C, et al. A metadata repository for semantic product lifecycle management[J]. Procedia CIRP, 2020, 91: 249-254. DOI:10.1016/j.procir.2019.11.006

[5] Chen Y, Ke Q, Li H, et al. xMeta: SSD-HDD-hybrid Optimization for Metadata Maintenance of Cloud-scale Object Storage[J]. ACM Transactions on Architecture and Code Optimization, 2024, 21(2): 1-20. DOI:10.1145/3652606

[6] Qi Y. Optimisation of distributed storage technology for large-scale data based on Hadoop technology[J]. International Journal of Reasoning-based Intelligent Systems, 2025, 17(7): 11-20. DOI:10.1504/ijris.2025.10071387

[7] He Z, Fang W. Research data management in institutional repositories: an architectural approach using data lakehouses[J]. Digital Library Perspectives, 2025, 41(1): 145-178. DOI:10.1108/dlp-02-2024-0022

[8] Cheng Y, Zhou K, Wang J, et al. A Framework for Big Earth Observation Data Using Horizontal Scaling Strategy[J]. IEEE Geoscience and Remote Sensing Letters, 2022, 19: 1-5. DOI:10.1109/lgrs.2022.3192640

[9] Punithavathi R, Kowsigan M, Shanthakumari R, et al. Protecting Data Mobility in Cloud Networks Using Metadata Security[J]. Computer Systems Science & Engineering, 2022,42(7):105-120. DOI:10.32604/csse.2022.020486

[10] Furner J. Definitions of “metadata”: A brief survey of international standards[J]. Journal of the Association for Information Science and Technology, 2020, 71(6): E33-E42. DOI:10.1002/asi.24295

[11] Vijaykumar N, Olgun A, Kanellopoulos K, et al. MetaSys: A practical open-source metadata management system to implement and evaluate cross-layer optimizations[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2022, 19(2): 1-29. DOI:10.1145/3505250

[12] Huang X, Gao Y, Zhou X, et al. An adaptive metadata management scheme based on deep reinforcement learning for large-scale distributed file systems[J]. IEEE/ACM Transactions on Networking, 2023, 31(6): 2840-2853. DOI:10.1109/tnet.2023.3266400

[13] Dai H, Wang Y, Kent K B, et al. The state of the art of metadata managements in large-scale distributed file systems—scalability, performance and availability[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(12): 3850-3869. DOI:10.1109/tpds.2022.3170574

[14] Dou W, Liu B, Lin C, et al. Architecture of virtual edge data center with intelligent metadata service of a geo-distributed file system[J]. Journal of systems architecture, 2022, 128: 102545. DOI:10.1016/j.sysarc.2022.102545

[15] Gao Y, Gao X, Zhang R, et al. An end-to-end learning-based metadata management approach for distributed file systems[J]. IEEE Transactions on Computers, 2021, 71(5): 1021-1034. DOI:10.1145/3337821.3337924

[16] Singh H J, Bawa S. LaMeta: An efficient locality aware metadata management technique for an ultra-large distributed storage system[J]. Journal of King Saud University-Computer and Information Sciences, 2022, 34(10): 8323-8335. DOI:10.1016/j.jksuci.2022.08.012

[17] Altahat M A, Daradkeh T, Agarwal A. Virtual machine scheduling and migration management across multi-cloud data centers: blockchain-based versus centralized frameworks[J]. Journal of Cloud Computing, 2025, 14(1): 1. DOI:10.1186/s13677-024-00724-7

[18] Song S, Park S, Kwon D. metaSafer: A Technique to detect heap metadata corruption in WebAssembly[J]. IEEE Access, 2023, 11: 124887-124898. DOI:10.1109/access.2023.3327817

[19] Peng Z, Feng D, Chen J, et al. RHPM: Using relative hotness to guide page migration for hybrid memory systems[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2022, 42(8): 2514-2526. DOI:10.1109/tcad.2022.3231836

[20] Gao Y, Gao X, Yang X, et al. An efficient ring-based metadata management policy for large-scale distributed file systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(9): 1962-1974. DOI:10.1109/tpds.2019.2901883

[21] He X, Zhang R, Tian P, et al. WOPE: a write-optimized and parallel-efficient B+-tree for persistent memory[J]. Journal of Systems Architecture, 2024, 153: 103187. DOI:10.1016/j.sysarc.2024.103187

[22] Cha M H, Lee S M, Kim H Y, et al. Effective metadata management in exascale file system[J]. Journal of Supercomputing, 2019, 75(11): 7665–7689. DOI:10.1007/s11227-019-02974-8

[23] Reddy N R S, Adapa M. AI-Driven Data Integration: Transforming Enterprise Data Pipelines through Machine Learning[J]. Journal of Computer Science and Technology Studies, 2025, 7(12): 110-119. DOI:10.30574/wjaets.2025.15.1.0245

[24] Wang D, Liu L, Liu Y. Normalized storage model construction and query optimization of book multi-source heterogeneous massive data[J]. IEEE Access, 2023, 11: 96543-96553. DOI:10.1109/access.2023.3301134

[25] Kumar A, Sivakumar P. Cat-squirrel optimization algorithm for VM migration in a cloud computing platform[J]. International Journal on Semantic Web and Information Systems (IJSWIS), 2022, 18(1): 1-23. DOI:10.4018/ijswis.297142

[26] Rao J, Ao T, Dai K, et al. ARCE: towards code pointer integrity on embedded processors using architecture-assisted run-time metadata management[J]. IEEE Computer Architecture Letters, 2019, 18(2): 115-118. DOI:10.1109/lca.2019.2935445

[27] Castro A, Areias M, Rocha R. Performance Evaluation of Separate Chaining for Concurrent Hash Maps[J]. Mathematics, 2025,13(17):2820-2820. DOI:10.3390/math13172820

[28] Smith J. Metadata remediation through migration, post-migration or necessary clean-up: A roadmap for success[J]. Journal of digital media management, 2024(3):12. DOI:10.69554/idto8966