When Deduplication Meets Migration: An Efficient and Adaptive Strategy in Distributed Storage Systems

Geyao Cheng, Lailong Luo*, Junxu Xia, Deke Guo, *Senior Member, IEEE*, Yuchen Sun

Abstract—The traditional migration methods are confronted with formidable challenges when data deduplication technologies are incorporated. Firstly, the deduplication creates data-sharing dependencies in the stored files; breaking such dependencies in migration may attach extra space overhead. Secondly, the redundancy elimination makes the storage system reserves only one copy for each storage file, and heightens the risk of data unavailability. The existing methods fail to tackle them in one shot. To this end, we propose Jingwei, an efficient and adaptive data migration strategy for deduplicated storage systems. To be specific, Jingwei tries to minimize the extra space cost in migration for space efficiency. Meanwhile, Jingwei realizes the service adaptability by encouraging replicas of hot files to spread out their data access requirements. We first model such a problem as an integer linear programming (ILP) and solve it with a commercial solver when only one empty migration target server is allowed. We then extend this problem to a scenario wherein multiple non-empty target servers are available for migration. We solve it by effective heuristic algorithms based on the Bloom Filter-based data sketches. The Jingwei strategy can suffer from performance degradation when the heat degree varies significantly. Therefore, we further present incremental adjustment strategies for the two scenarios, which adjust the number of block replicas and their locations in an incremental manner. The mathematical analyses and trace-driven experiments show the effectiveness of our Jingwei strategy. To be specific, Jingwei fortifies the file replicas by 25% with only 5.7% of the extra storage space, compared with the latest "Goseed" method. With the small extra space cost, the file retrieval throughput of Jingwei can reach up to 333.5Mbps, which is 12.3% higher than that of the Random method.

Index Terms—data migration, data deduplication, replica storage, heat variation

1 Introduction

The data volume surges exponentially in the "big data" era. This brings a serious impact on the network system and has become a nonnegligible hot issue. To handle the "big data" challenge, current storage systems mainly adopt the data deduplication technologies [2], [3] to save space. It has been reported that, for some multimedia and IoT storing data, up to 70% storage space can be released when deduplication technologies are assembled [4]. A common practice for data deduplication is to split files into multiple fixed- [2], [5] or variable-size [6] blocks. By doing so, the data-sharing dependencies among the files are established, and only one copy of each block is maintained in a storage server.

When a server is overloaded, part of its files must be migrated out to another server [7], [8]. However, the traditional migration methods are confronted with formidable challenges when data deduplication is incorporated to economize the scarce storage resource. First, the deduplication creates data-sharing dependencies between the stored files; breaking such dependencies may attach additional space overhead to the system. The reason is that, the shared blocks must be copied at both the source server and the migration target server to maintain the file integrity. Second, redundancy elimination makes the storage system reserves only one copy for each storage file. Then, the file retrieval performance may deteriorate when files become hot, because the frequent requests of such hot files may overwhelm their stored servers, heightening the risk of data unavailability.

Therefore, in this paper, we envision the following two rationales for data migration in the deduplicated storage systems: 1) *Space Efficiency* – the introduced extra space overhead is minimized; 2) *Service Adaptability* – files are allowed to have multiple replicas to avoid data unavailability, like Ceph [9] and Google [10] file systems. These two rationales, if both are realized, will bring unprecedented benefits to the storage systems. To be specific, the scarce storage resources can be economized, and in the meanwhile, the concentrated data requirements of hot files can be spread to alleviate the potential request congestion. Furthermore, the replica generation may attach only a tiny or even non-amount of extra space overhead when the two rationales are integrated.

The existing data migration strategies, however, fail to consider these two rationales jointly. The intrinsic reason is that, these two rationales are mutually exclusive. Eliminating all redundancies would impact the service adaptability, but too many replicas would bring unnecessary space spending. The current data migration strategies coupled with data deduplication mainly focus on the capacity measurement [11], the space reduction [7], [12], etc. However, they are oblivious to the impact of data replicas. The storage system without replicas, especially for hot files, may impact the service adaptability significantly in practice [2]. On the other hand, the popularity-aware replication managements [13]–[16] or file assignments [17] improve the service performance undoubtedly. However, they are currently not incorporated with the

- *A preliminary version of this paper was accepted in INFOCOM 2022 [1]. This version further considered the heat variation, and innovatively proposed adjustment strategies to alleviate performance degradation caused by heat changes. More contents were fortified to the mathematical analysis, evaluation, discussion, etc.*
- *The authors are with the Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology, Changsha Hunan* 410073*, China. E-mail: {chenggeyao13, luolailong09, sunyuchen18}@nudt.edu.cn; {junxuxia, guodeke}@gmail.com*
- *Lailong Luo is also with the National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha Hunan* 410073*, China. Lailong Luo is the corresponding author.*

space reduction technologies in the deduplicated storage systems.

Inspired by these observations, in this paper, we propose Jingwei[1], an efficient and adaptive data migration strategy for deduplicated storage systems. Jingwei realizes a proper trade-off between the space efficiency (minimizing data replication in the migration process) and the service adaptability (building replicas of hot files to spread the frequent data access requirements). These two optimization aspects are traditionally carried out separately, yet it is pathbreaking to realize and couple these two rationales jointly so as to yield rational data migration strategies.

An example of the Jingwei strategy is illustrated in Fig. 1. The ambition is to migrate a portion of files from the overloaded Server 1 to the under-utilized Server 2. Scheme (a) [7] minimizes the amount of replicated data by allocating files with more common blocks to one server. Nevertheless, there are no file copies to guarantee the service adaptability. Furthermore, the aggregation of hot files f_3 and f_4 may overwhelm Server 2 with the accumulated data access requirements. Scheme (b) [16], by contrast, satisfies the service adaptability through replicating f_2 at the two servers. However, the data deduplication is not incorporated, leading to much more space occupation for redundancies. Scheme (c) (i.e., Jingwei), fortunately, is a relatively optimal solution. It detects the file similarity to realize the space efficiency. Meanwhile, it replicates the hot file f_3 to permit the service adaptability rationale with little extra space, i.e., one more block than that of Scheme (a).

Furthermore, Jingwei should be able to handle dynamic changes in file heat. Otherwise, the original file allocation strategies can experience performance degradation. Taking Fig. 1 as an example, Server 1 in Scheme (c) might suffer from potential request congestion when files' heat degrees vary, i.e., the heat degree of file f_1 increases from 1 to 10, while that of file f_3 decreases from 6 to 2. Then, we can allocate all requests for file f_3 to Server 2 and release its occupied space (b_4) from Server 1. Furthermore, file f_1 can be replicated at Server 2 with one more block b_1, as shown in Fig. 1(d). After this incremental adjustment, the concentrated file requests can be apportioned to the two servers evenly, thereby avoiding the appearance of access hotspots.

The major contributions are summarized as follows.

- We design two heterogeneous migration scenarios for Jingwei to improve the strategy applicability. The first scenario is migrating files to an empty server. We model such a problem as an integer linear programming (ILP) and solve the NP-hard problem with the ILP solver. The problem is also addressed in a more general scenario wherein multiple non-empty target servers are available. We solve it by effective heuristic algorithms, i.e., space-saving data migration and heat-aware data replication, based on the Bloom Filter (BF)-based data sketches.
- To alleviate the performance degradation caused by files' heat variation, we further provide incremental adjustment techniques that perform additional block deletion/migration/replication operations at a reasonable cost for both of the two heterogeneous scenarios.
- We also attach the mathematical analyses to guarantee the theoretical effectiveness of the BF-based data migration strategies. The analyses theoretically highlight the effectiveness of our Jingwei strategy.

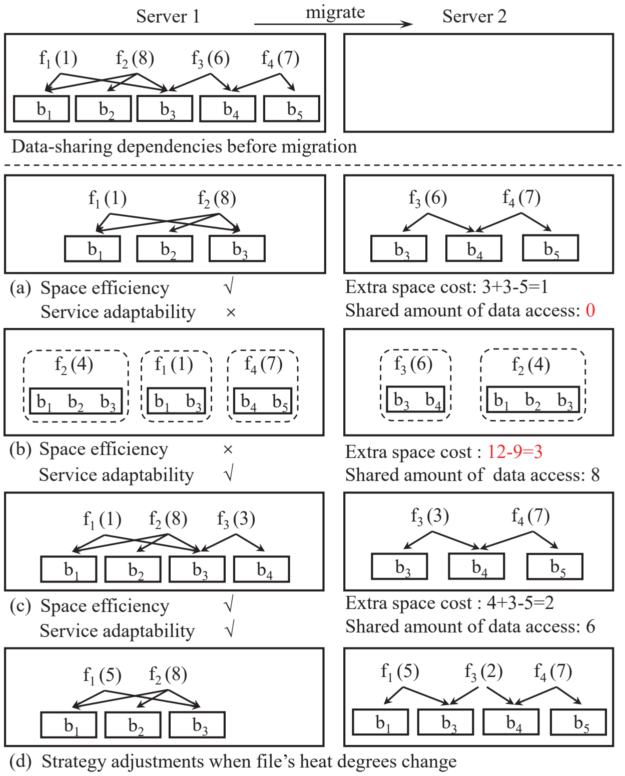


Fig. 1. The illustrative examples of the Jingwei strategy and some existing methods [7], [17]. Four files ($f_1 \sim f_4$), which are attached with heat degrees (1, 8, 6, 7), are partitioned into five blocks ($b_1 \sim b_5$). The ambition is to migrate a part of the four files from server 1 to server 2.

- Trace-driven experiments show that our Jingwei strategy fortifies the file replicas by 25%, while only 5.7% extra storage space is occupied compared with the latest "Goseed" scheme. With the small extra space cost, the file retrieval throughput of Jingwei can reach up to 333.5Mbps, which is 12.3% higher than that of the Random method.

The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 states the Jingwei overview. Section 4 presents the problem formulation for the first migration scenario. Section 5 exhibits the heuristic algorithms for general migration scenario. Section 6 discusses the incremental adjustment techniques to deal with files' heat variation. Section 7 demonstrates the performance analyses. Section 8 reports our experimental results. Section 9 provides some discussions. Finally, Section 10 concludes this paper.

2 Related Work

Data migration in deduplicated storage systems has attracted more attention in recent years with different concerns. Harnik et al. [11] provide sketch-based estimations of the reclaimable/attributed capacity when a group of volumes is removed from/added into the deduplicated storage system. Duggal et al. [12] deploy cloud-tier systems to decrease the cost of copy forward in deduplicated data migration. Goseed [7] minimizes the extra space occupation for data migration based on the data-sharing dependencies. However, these works mainly focus on the space occupation, but do not take the data popularity into account. In the worst cases, the frequent data access for hot data would exhaust the limited service capability of the server, resulting in significant degradation of user experience.

[1] Jingwei is a famous fictional character in Chinese folklore, who carries pebbles and branches from the land to the sea for her revenge, meaning migrating files from one server to others in this paper.

TABLE 1
Comparison of Related Works.

Literature	***Space efficiency***	***Service adaptability***
[2], [7], [11], [12]	✓	×
[14]–[16], [21]	×	✓
This paper	✓	✓

Besides, the data popularity plays a vital role in optimizing the file assignment [17], replication management [13]–[15], and load balancing [18] in the intelligent data management systems. It measures the frequency of data access and correlates closely with service-related objectives, such as the hit ratio and the request throughput [19], [20]. A highlighted solution to avoid service overload is replication management [13]–[16]. Hamdeni et al. [13] provide a comprehensive survey on the data popularity and emphasize the importance of data replicas. Literature [14] increases the replicas for hot data and allocates them evenly across the storage system for convenient data retrieval. Wei et al. [15] deploy a minimal number of replicas and place them separately on those servers with the most service abilities. Shen et al. [16] utilize file replication technologies to reduce hot spots and improve file query efficiency. However, these solutions are not incorporated with deduplication technologies, which is crucial to enhance the space efficiency for storage systems, especially for the successive backups with a high deduplication ratio [24].

The previous works have investigated the data migration strategies integrated with data deduplication technologies, or how to place the replicas of hot data rationally, but not both. Note that, optimizing on any one dimension alone is too restrictive. Literature [2] caches hot data at edge with data deduplication, which considers the data popularity as well as the space occupation in deduplicated storage systems. However, it only works in the granularity of storing files at data centers or edge coarsely. In addition, the method does not elevate the system's service adaptability by adding data replicas in response to the network's unstable situations. Literature [21], by contrast, permits data redundancies in the deduplicated storage system. It builds a two-tier storage hierarchy, where the Primary cluster is responsible for storing full file replicas, and the Deduplication cluster stores the unique blocks. This strategy implements the prefetch/pre-construct cache algorithm based on the user's access patterns, but is still not space-efficient for storing replicas of all involved files.

Unlike the existing strategies, It is path-breaking for our Jingwei to highlight the importance of data replicas in the space-efficient deduplicated storage systems. In addition, Jingwei achieves an elegant trade-off between the proposed space efficiency and service adaptability rationales.

3 The Overview of Jingwei

The "big data" era has put forward a tough challenge for the server's storage and service capacity. When a server is overloaded, data migration provides an effective way to alleviate the load burden in the storage systems. Data deduplication further economizes the scarce space resources by splitting files as blocks and removing duplicated ones. Two design rationales are required when the data deduplication is incorporated into the migration strategy:

- *Space efficiency:* the data migration strategy should decrease the extra space cost caused by breaking the data-sharing dependencies in the migration process.

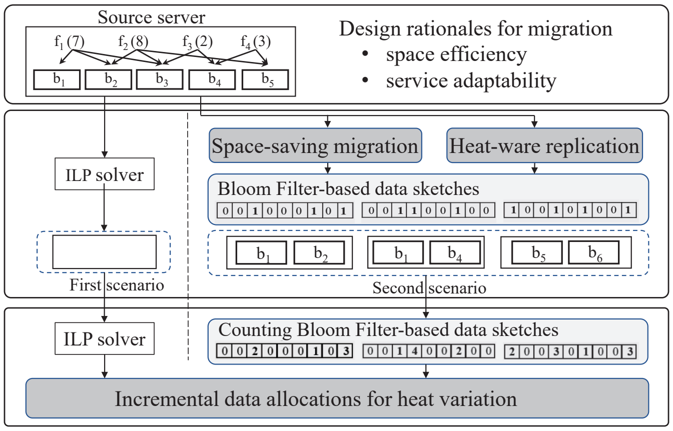


Fig. 2. The overview of Jingwei.

- *Service adaptability:* the data migration strategy should maintain some replicas of hot files to amortize the frequent file requests and improve the service quality.

Migration Mode: Some deduplicated systems split the incoming files into blocks, and store the blocks dispersedly without the constraint of file integrity [22], [23]. Another emerging deduplication model supports storing all blocks of the original file at one server, so that accessing a file will not require excessive rounds of communications to multiple servers [3], [5], [7]. We track the latter mode in this paper, where a file's partitioned blocks are stored on one server, and the migration scheme is conducted at the file layer. Each storage server manages its own local fingerprint index and identifies deduplicate blocks by looking up this index table.

Jingwei overview: The overview of our Jingwei strategy is exhibited in Fig. 2. The data deduplication is conducted at each involved server, wherein only one copy of each block can be maintained, and the duplicated blocks are replaced with pointers. Thus, the data-sharing dependencies are established. To comprehensively conduct the efficient and adaptive data migration strategy, we design two scenarios when migrating out a part of files from the overloaded source server. Specifically, when only one empty server acts as the migration target, we model the problem as an integer linear programming (ILP) and solve this NP-hard problem with the ILP solver. The specific problem formulation is exhibited in Section 4.

To adapt to more migration situations and requirements, we extend the problem into a more general scenario, where any empty or non-empty server can act as the candidate for the migration targets. We leverage the space-saving migration algorithm to determine the migrated files and their migration targets in priority of low extra space cost. Thereafter, we present the heat-aware data replication algorithm to replicate hot files with only limited extra space overhead, which achieves the service adaptability. The BF-based data sketches assist the above two algorithms by detecting the content similarity with low computational overhead. The specific algorithms are exploited in Section 5.

To ascertain the system performance with heat variation, we further design the incremental adjustment techniques for the two scenarios in Section 6. Based on the current storage state, these techniques perform additional block deletion/migration/replication operations according to the varied file's heat degree, but do not resort to entire reallocating. Thus, the adjustment cost, such as the extra space cost and bandwidth consumption, can be reduced. Specifically, the ILP solver is still adopted for the first scenario. As for the second scenario, we upgrade the BF-based data sketches with the Counting Bloom Filter, which facilitates the item deletion operations when files become cold. The ambition for the incremental adjustment is to mitigate the performance degradation caused by heat variation, while reducing the adjustment cost.

4 Migrating Files to A Single Empty Server

In this section, we model the migration problem when only one empty target server is allowed. Specifically, we first provide the problem definition in Section 4.1. Thereafter, we formulate this migration problem in section 4.2.

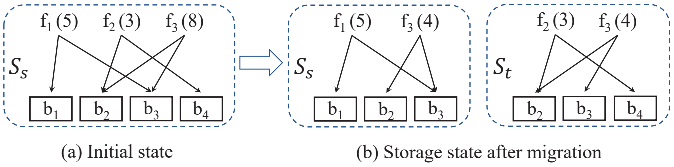
4.1 Problem Definition

In the initial storage state, the source server S_s stores a set of files $F_s = \{f_1, f_2, ...\}$ with heat degrees $H_s = \{h_1, h_2, ...\}$. These files are partitioned into blocks, and only the unique blocks (after data deduplication) can be contained, which are represented by $B_s = \{b_1, b_2, ...\}$. Let $size(b)$ denote the size of block b, then the storage cost of server S_s is the total size of the blocks stored on it, i.e., $size(S_s) = \sum_{b_j \in B_s} size(b_j)$. Note that, this size function generates a constant value for fixed-size block chunking [2], and varies for the variable-size block chunking algorithms [6]. Let $I_s = F_s \times B_s$ indicate an inclusion relation, where $(f_i, b_j) \in I_s$ means that block b_j is included in file f_i.

Currently, the source server S_s is overloaded, i.e., the storage capacity C_s cannot afford the stored data size $size(S_s)$. Thus, a part of files in F_s should be migrated from the source server S_s to the empty target server S_t. The ambition is to realize the two proposed rationales simultaneously, i.e., space efficiency and service adaptability. The critical point lies in how should we choose the files to migrate in F_s? To do this, we first categorize the possible states of each candidate file $f_i \in F_s$. Then, we deeply analyze the introduced extra space overhead in breaking the data-sharing dependencies, and the apportioned access requests for the file replicas. To be specific, the file states would be in one of the following three states as follow when enough amount of data is migrated out from the source server S_s:

- **migrated**, i.e., f_i is migrated to the target server S_t, while the space occupied at the source server is released. We introduce the following Boolean state variable $x_i \in \{0,1\}$, $\forall f_i \in F_s$, to represent this state, such that:
$$x_i = \begin{cases} 1 & if\ f_i\ is\ migrated\ to\ the\ target\ server; \\ 0 & otherwise. \end{cases} \tag{1}$$
- **replicated**, i.e., the source server sends a copy of f_i to the target server. This usually appears for hot files, where the data access requirements may overwhelm the capacity of the source server. We introduce the following binary Boolean state variable $y_i \in \{0,1\}, \forall f_i \in F_s$, to express this state, such that:
$$y_i = \begin{cases} 1 & if\ f_i\ is\ replicated\ to\ the\ target\ server; \\ 0 & otherwise. \end{cases} \tag{2}$$
- **unaltered**, i.e., f_i remains at the source server without being migrated or replicated, meaning $x_i = 0$ and $y_i = 0$.

Based on the file states indicated by the above Boolean variables, the deeply-associated state of their partitioned blocks can also be mathematically expressed by Boolean variable definitions.



(a) Initial state

(b) Storage state after migration

Fig. 3. An illustrative example of data migration from the source server S_s to the empty target server S_t.

To denote the *migrated* state of a block, we define a Boolean variable ($m_j \in \{0,1\}, \forall b_j \in B_s$), where
$$m_j = \begin{cases} 1 & if\ block\ b_j\ is\ migrated; \\ 0 & otherwise. \end{cases} \tag{3}$$
When $m_j = 1$, the block b_j should be migrated out from the source server S_s to the target server S_t. This state can only be caused by the migration of its subordinated file f_i, where $x_i = 1$ & $(f_i, b_j) \in I_s$. We further define a Boolean variable $r_j \in \{0,1\}, \forall b_j \in B_s$, as:
$$r_j = \begin{cases} 1 & if\ block\ b_j\ is\ replicated; \\ 0 & otherwise. \end{cases} \tag{4}$$
When $r_j = 1$, the block b_j would appear at both the source and the target server. If any of its affiliated files (the files that contain b_j) is replicated during the migration process, the state of b_j would be labeled as *replicated*. Furthermore, breaking the data-sharing dependencies of two files (one is migrated, and the other is unaltered) would also attach block replications in the shared part. Note that, r_j also relates to the extra space cost caused by file migration, which can be represented as $\sum_{b_j \in B_s} size(b_j) \times r_j$. If both $m_j = 0$ and $r_j = 0$, it means that b_j remains *unaltered*.

To conclude, the state interrelations between files and their partitioned blocks in I_s can be dissected and denoted as one of the following three cases. Fig. 3 takes intelligible examples to illustrate these unique instances.

- Case 1: One file remains at the source server S_s, i.e., x_i=0, y_i= 0, like file f_1 in Fig. 3. In this case, all blocks included in the file are either unaltered (like block b_1) or replicated (like block b_3), hinging on the block sharing dependencies between the unaltered file and the migrated/replicated files.
- Case 2: One file is migrated to the target server S_t, i.e., x_i=1, like file f_2 in Fig. 3. In this case, all blocks included in the file would be either migrated (like block b_4) or replicated (like block b_2).
- Case 3: One file is replicated at both the source and the target server, such as file f_3 in Fig. 3. In this case, all involved blocks of file f_3, i.e., $\forall b_j \in B_s$ & $(f_3, b_j) \in I_s$, would be replicated. Besides, the access requirements of this file would be spread by its replications with a distribution parameter $\gamma \in [0,1]$ ($\gamma = 0.5$ in the example of Fig. 3). This parameter can be adjusted by the widely utilized load balancer in the network [25], which is responsible for load balancing across servers in the storage systems.

4.2 Problem Formulation

With the aforementioned Boolean variables about files and blocks, we can formulate the migration problem with an empty target server as follows.

- When block b_j is migrated, i.e., $m_j = 1$, then all files that contain the block would be migrated to the target server:
$$m_j \leq x_i,\ \forall f_i \in F_s,\ b_j \in B_s\ \&\ (f_i, b_j) \in I_s. \tag{5}$$

- When file f_i is migrated, i.e., $x_i = 1$, then all of its contained blocks would be either migrated or replicated:

$$x_i \le m_j + r_j,\ \forall f_i \in F_s,\ b_j \in B_s\ \&\ (f_i, b_j) \in I_s. \tag{6}$$

- When file f_i is replicated at both server S_s and S_t, i.e., $y_i = 1$, then all of its involved blocks should also be replicated:

$$y_i \le r_j,\ \forall f_i \in F_s,\ b_j \in B_s\ \&\ (f_i, b_j) \in I_s. \tag{7}$$

- The states of files, i.e., x_i and y_i, and the states of blocks, i.e., m_j and r_j, are mutually exclusive:

$$x_i + y_i \le 1,\ m_j + r_j \le 1,\ \forall f_i \in F_s,\ b_j \in B_s. \tag{8}$$

- The migrated block volume, i.e., $\sum_{b_j \in B_s} size(b_j) \cdot m_j$, $\forall b_j \in B_s$ should meet the pre-defined migration percentage (M) of the server load in S_s (C_s). This percentage can be determined by the joint considerations of the storage burden of server S_s and the actual storage situations in the deduplicated storage systems.

$$\sum_{b_j \in B_s} size(b_j) \times m_j \ge M \cdot C_s. \tag{9}$$

- The final space (C)/service (T) overhead of both the source and target server should not exceed the corresponding capacities for $\forall f_i \in F_s, b_j \in B_s\ \&\ (f_i, b_j) \in I_s$.

$$\sum_{b_j \in B_s} size(b_j) \times (1 - m_j) \le C_s. \tag{10}$$

$$\sum_{b_j \in B_s} size(b_j) \times (m_j + r_j) \le C_t. \tag{11}$$

$$\sum_{f_i \in F_s} h_i \times (1 - x_i - \gamma_i \cdot y_i)) \le T_s. \tag{12}$$

$$\sum_{f_i \in F_s} h_i \times (x_i + \gamma_i \cdot y_i) \le T_t. \tag{13}$$

- The state variables are all Boolean: $x_i, y_i, m_j, r_j \in \{0, 1\},\ \forall f_i \in F_s,\ b_j \in B_s$.

We develop the objectives of our Jingwei scheme, i.e., realizing an elegant trade-off between the space efficiency and the service adaptability. The space efficiency is described by minimizing the extra space cost caused by block replication, i.e., $\sum_{b_j \in B_s} size(b_j) \times r_j$. The service adaptability can be represented by maximizing the amount of share data requirements, i.e., $\sum_{f_i \in F_s} h_i \times y_i$. These two rationales are normalized as follows.

$$\min \frac{\sum_{b_j \in B_s} size(b_j) \times r_j}{\sum_{b_j \in B_s} size(b_j)} - \lambda \frac{\sum_{f_i \in F_s} h_i \times y_i}{\sum_{f_i \in F_s} h_i}, \tag{14}$$

where the parameter λ can be adjusted to adapt to different optimization tendencies for these two rationales. To be specific, with the increase of parameter λ, the derived block migration/replication schemes would tend to enhance the service adaptability rationale.

With Equ. (14) as the migration objective and Equ. (5)∼(13) as the constraints, the problem can be formulated as an Integer Linear Programming (ILP). The ILP problem is known to be NP-hard [26], and there is currently no known efficient solving algorithm in polynomial time complexity. In particular, when the variables are restricted to Boolean assignments (0 or 1), then merely deciding whether the problem has an optimal solution has been known to be NP-Complete [27]. Fortunately, commercial optimizers, like CPLEX [28], lp_solve [29], and Gurobi optimizer [30], can solve this kind of problems efficiently for instances with hundreds of thousands of variables. Therefore, we exploit these highly-optimized solvers to search out the optimal migration plan directly.

5 Migrating files to Multiple Non-empty Servers

The scenario with only one empty target migration server may not be applicable for large-scale storage systems, where it is not common for a server to join with an empty state. In addition, the constraint of migrating all files to one server may limit performance improvement. Therefore, we design a more general scenario, where multiple non-empty servers rather than one empty server can accept the migrated files from an overloaded server. As a consequence, the above formulation will not be applicable. Therefore, in this section, we further propose efficient heuristic algorithms for the general migration scenario based on the Bloom filter-based data sketches.

5.1 Bloom Filter-based Data Sketch

To find the optimal target server for each file to migrate, an intuitive method is to compare the fingerprints (using MD5 [31] or SHA-1 [32] coding) of blocks contained by the file and that stored by the candidate servers. The files prefer to be migrated to the server with more common blocks. However, the information comparisons would consume non-trivial computation resources and lead to unbearable processing latency. For example, for a file with n blocks, it takes $O(n \times |B_t|)$ time-complexity to determine whether the server contains such blocks or not, where $|B_t|$ is the total number of blocks in a candidate server. In order to decrease the computation complexity, we adopt Bloom Filter (BF) [33], [34], a hashing mapping method that has been widely utilized in networking and distributed systems, to represent the blocks on each candidate server. This facilitates data similarity detection from pair-wise fingerprint checking to the membership queries on the data sketches. Then the time-complexity of determining whether a server contains the n blocks in a file can be decreased as $O(n \cdot k_{BF})$, where k_{BF} indicates the number of utilized hash functions in BF.

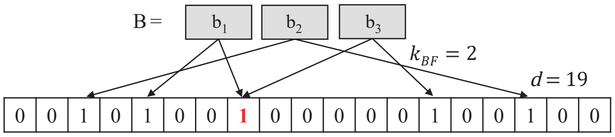


Fig. 4. An illustrative example of the BF-based data sketch. Note that the 8^{th} bit of the sketch suffers from the hash collisions.

Fig. 4 provides an illustrative example for a BF-based data sketch. Given the block set B with three partitioned blocks b_1, b_2, and b_3, the BF represents B with a bit vector of length $d = 19$. All d bits in the vector are initially set as 0. The $k_{BF} = 2$ independent hash functions are employed to map each block into k_{BF} positions in the bit vector. Those hit positions would be all set to 1. The binary string derived from the hash functions is exactly the BF-based data sketch.

Each server would maintain a bit vector, with the same k_{BF} functions and vector length, to record the membership information at the block level. According to the bit vector and the k_{BF} used hash functions, we can realize the membership queries against any data block. To be specific, when a file f_i in the source server tries to select its optimal target server from all available candidates, it would first require the BF vector of each candidate. For any block b_j in file f_i, the BF judges that this block does not belong to the candidate server, if any bit at the k_{BF} hashed positions in the BF vector is 0. Otherwise, the BF believes that the queried block b_j belongs to the candidate target server with a rate of false positives.

5.2 The Effective Heuristic Migration Algorithms

The BF-based data sketch elaborates a feasible and effective method to detect data similarity through membership queries. According to the data sketches, we propose effective heuristic algorithms for migrating files to multiple non-empty servers. The heuristic algorithms are composed of two parts, i.e., the space-saving data migration in Section 5.2.1 and the heat-aware data replication in Section 5.2.2.

Note that, the prerequisite of this general data migration is that the candidate targets are all underloaded and have the potential to accommodate more data blocks. However, as data accumulate, some underloaded servers would gradually approach their storage capacities, and have no extra space to receive new files. In this case, these high-loaded servers would be kicked out of our migration/replication candidates. Other candidates would be selected as the migration/replication targets, even though they may contain fewer common blocks and occupy more extra space to accommodate the selected file. One extreme situation, although unlikely to appear, is that all servers in the storage systems are overloaded. To counter this special case, we would suggest adding some empty servers to our storage systems. Then, this general migration scheme can be transferred to the first scenario (expressed in Section 4), where the new-added empty servers act as the migration targets.

5.2.1 Space-saving Data Migration

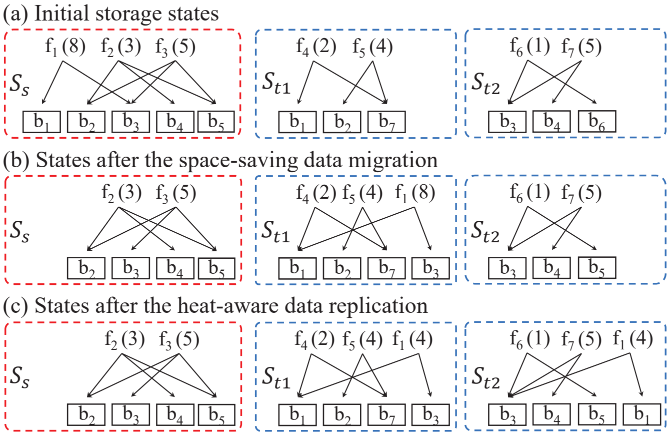
The space-saving data migration determines which files to migrate and where they should be directed to, with the ambition of less extra space cost. To achieve this, we rank the migration sequence of files according to a space-saving index. We define this index as the amount of saved storage resource when a file migrates to a candidate target server. The index can be represented by the deviation between the data amount that is freed from the source server and the increased space on the migration target. We prefer the data migration in priority of the high space-saving index. This plays a vital role in improving the systems' space efficiency.

The specific steps are detailed in Algorithm 1. The input includes the BF-based data sketches and file sets for the source server S_s and all target candidates $\tilde{S}_t$, where $\tilde{S}_t = \{S_1, S_2, \ldots, S_n\}$. The file set of the candidate server S_k is denoted by F_k. To derive the migration variable x_i and the corresponding migration target $S_t(i)$, we elaborate a space-saving index to inspire the migration sequence. The function is shown in Lines 10-14. Let $\varphi(i,k)$ represent the data volume of shared blocks between f_i and $S_k \in \tilde{S}_t$, which can be derived from the BF-based membership queries of blocks in f_i on the S_k' data sketch (Ψ_k). Then, the function returns the index $I(i,k)$ according to the deviation between $\varphi(i,k)$ and $\varphi(i,s)$, which reflects the saved space resources through migrating file f_i from S_s to the S_k. Note that, the value of $\varphi(i,s)$ is calculated based on the sketch without f_i, which can actually reflect the space overlapping between f_i and others in S_s.

With the space-saving index for each file-server matching, we can determine the files to migrate and their target servers by calculating the maximum $I(i,k)$, if the storage capacity of the candidate target server $S_{\hat{k}}$ is available. This step would be processed iteratively until M percentage of the storage volume in S_s has been migrated (Lines 3-9). Note that, each migration would release or add blocks on the source server and the target server. Thus, the data sketches should be locally updated. Furthermore, the ranking index should also be updated on the related servers accordingly (Lines 7-9).

Algorithm 1: Space-saving Data Migration

```
   Input: Data sketch ($\Psi$) and file set ($F$) for $S_s$ and $\tilde{S}_t$; the
          target migration percentage $M$.
   Output: The migration variable $x_i$ and the target server
          $S_t(i)$ for each file $f_i$ in $F_s$.
1  $F_s' = F_s$; $x_i = 0$, $S_t(i) = S_s$, $\forall f_i \in F_s'$.
2  Generate the global space-saving indexes through
     INDEX_CALCU($F_s', S_k$), $\forall S_k \in \tilde{S}_t$.
3  while $M$ is not reached do
4      Get $I(i,k)$, $\forall f_i \in F_s'$, $S_k \in \tilde{S}_t$;
5      Determine the file to migrate and its target $[f_{\hat{i}}, S_{\hat{k}}]$ in
         $\max_{i=1}^{|F_s|} \max_{k=1}^{|\tilde{S}_t|} I(i,k)$, if the capacities are available;
6      Migrate file $f_{\hat{i}}$ to $S_{\hat{k}}$, where $x_{\hat{i}} = 1$ and $S_t(\hat{i}) = S_{\hat{k}}$;
7      Updated the file set $F_s'$: $F_s' = F_s' \setminus \{f_{\hat{i}}\}$;
8      Updated $\Psi_{\hat{k}}$ with the new-added file $f_{\hat{i}}$;
9      Update $I(i,\hat{k})$ with INDEX_CALCU($F_s', S_{\hat{k}}$).
10 function INDEX_CALCU($F_s, S_k$)
11 for $i = 1 \rightarrow |F_s|$ do
12     Calculate $\varphi(i,k)$ based on data sketch $\Psi_k$;
13     Define the ranking index of file $f_i$ and server $S_k$ by
         $I(i,k) = \varphi(i,k) - \varphi(i,s)$.
14 return $I(i,k)$, $\forall f_i \in S_s$
```


(a) Initial storage states

(b) States after the space-saving data migration

(c) States after the heat-aware data replication

Fig. 5. An illustrative example of the space-saving data migration and the heat-aware data replication. There are one source server (S_s) and two target servers (S_{t1} and S_{t2}) in this example. All servers can accommodate at most four blocks and serve 10 requests per unit of time.

Fig. 5 gives an illustrative example of our space-saving data migration process. When the source server S_s becomes overloaded in the initial storage states, we first calculate the space-saving index (I) for all its contained files (f_1, f_2, and f_3) to inspire the migration sequence. To be specific, $\varphi(i,k) = 1, \forall f_i \in \{f_1, f_2, f_3\}$, $S_k \in \{S_{t1}, S_{t2}\}$, while $\varphi(1,s) = 1$, $\varphi(2,s) = 2$, $\varphi(3,s) = 3$. Therefore, we can get one minimum space-saving index, i.e., $I(1,1) = \varphi(1,1) - \varphi(1,s) = 0$, which reflects that no more extra space would be occupied for migrating file f_1 from S_s to S_{t1}. We select this migration, with block b_1 being released from S_s and block b_3 being attached to S_{t1}. Then, the load burden of S_s is alleviated, and the migration algorithm would stop.

5.2.2 Heat-aware Data Replication

After determining the migrated files and their destinations, the next step is to adjust this migration plan considering the files' heat degree. Overheated files should have multiple replicas in the system for apportioning the frequent file requests. We present a

Algorithm 2: Heat-aware Data Replication

```
Input: Heat degree $H_s$ of file set $F_s$; available service
       capacities (ASC) of $\tilde{S}_t$; the unit-heat value $\kappa$.
Output: The replica locations $repeat\_set(i)$ and the heat
        allocation $\gamma_i$ for each file $f_i$ in $F_s$.
1  for $i = 1 \rightarrow |F_s|$ do
2      $repeat\_set(i) = \{S_t(i)\}$;
3      Get $\varphi(i,k),\ \forall S_k \in \tilde{S}_t\ \&\ S_k \neq S_t(i)$;
4      Build $Q_i$ by sorting $\varphi(i,k)$ in descending order, if the
         capacities are available;
5      for $k = 1 \rightarrow |Q_i|$ do
6          Calculate $AFR(i) = h_i/(|repeat\_set(i)|+1)$;
7          if $AFR(i)/(size(f_i) - \varphi(i,k)) >= \kappa$ then
8              $repeat\_set(i)$=$repeat\_set(i) \cup \{S_k\}$;
9              Updated the sketch $\Psi_k$ with file $f_i$;
10         else
11             break;
12     Adjust $\gamma_i$ by HEAT_ALLOCATION($f_i$,$repeat\_set(i)$).
13 function HEAT_ALLOCATION($f_i$, $repeat\_set(i)$)
14   Derive $ASC$ for all servers in $repeat\_set(i)$;
15   Compute $\gamma_i(j) = ASC(j)/\sum_{j=1}^{|repeat\_set(i)|} ASC(j)$;
16   Update $ASC(j)$=$ASC(j)-\gamma_i(j)h_i,\ \forall S_j \in repeat\_set(i)$.
17 return $\gamma_i$, $ASC$
```

unit-heat value (κ) to exploit the necessity of file replication. The value of κ can adjust the number of generated file replicas. With a smaller κ, there can be more file replicas to amortize the frequent data accesses, thereby enhancing the service quality in the storage system. We also calculate the quotient between the apportioned file requests (AFR) and the extra space the replica requires. The $AFR(i)$ is defined as the evenly apportioned file requests that each replica of file f_i undertakes. Any replication is executed if the quotient value exceeds the unit-heat value. This ensures the replica generation of hot files with little extra space cost.

The specific algorithm is expressed in Algorithm 2. For any file f_i with its current storage server $S_t(i)$, we get $\varphi(i,k), \forall S_k \in \tilde{S}_t\ \&\ S_k \neq S_t(i)$. The value of $\varphi(i,k)$ is thereafter sorted in descending order to construct the server queue Q_i, when the capacities are under the constraints (Lines 1-4). The server ranking at the front of Q_i contains more similar content with file f_i. For each server in queue Q_i, we calculate the apportioned file requests $AFR(i)$ and the extra space cost $size(f_i)-\varphi(i,k)$. If the quotient between these two parameters is larger than κ, then the file f_i would be replicated to S_k, i.e., the k^{th} server in Q_i. Then, the data sketch Ψ_k would be updated with this new-added replica of file f_i (Lines 5-9). After determining the replica locations $repeat\_set(i)$ for file f_i, we further leverage function HEAT_ALLOCATION$(f_i, repeat_set(i))$ (Lines 13-17) to adjust the allocated amount of data access for each server in $repeat\_set(i)$. This function takes over the role of the load scheduler, which balances the service load according to the available service capabilities (ASC) of the involved servers. To be specific, the server with more available service capability would undertake more potential data access requirements.

Fig. 5 further illustrates the example for our heat-aware data replication process. For all files, we calculate the extra space cost with one more file replica ($size(f_i)-\varphi(i,k)$) as well as their split data access $AFR(i)$. We find the hot file f_1 can be replicated to S_{t2} with one more block, where $AFR(1)/(size(f_1) - \varphi(1,2)) = 4/1 = 4$. When setting $\kappa \leq 4$, file f_1 would be replicated to S_{t2}, with its data access requirements being distributed to the two affiliated servers (S_{t1} and S_{t2}). The HEAT_ALLOCATION function further balances the service load according to the available service capabilities of the involved servers.

6 Incremental Data Allocation Adjustment Strategies with Heat Variance

Data popularity (heat degree) measures the frequency of data access requirements. One non-negligible feature of data popularity is its variation, which indicates file' heat degree varies over time. This feature is more pronounced for time-sensitive data, such as hot news and real-time weather.

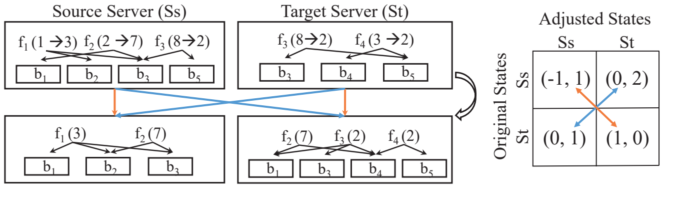
When files' heat degrees change significantly, the original data allocation strategy can suffer from performance degradation. This can break the service balancing maintained by the original allocation strategies and lead to the emergence of new access hotspots. To be specific, for the first scenario wherein only one server is allowed for migration, the one-side optimization objective of maximizing the shared amount of data requirements $\sum_{f_i \in F_s} h_i \times y_i$ may be inconsistent with the actual requirements when any heat value $h_i \in H$ varies. In the same manner, for the second scenario wherein multiple non-empty target servers are allowed for migration, the updated heat degree of any file may inconsistent with the original heat-aware data replications. The magnitude of performance degradation grows with the increase of heat variance.

To this end, we present an adjustment strategy that performs block deletion/migration/replication according to the varied files' heat degrees. The core idea is to adjust a part of files in an incremental manner when the heat variation reaches a threshold, instead of recomputing the entire data allocation scheme. Such an incremental adjustment technique is more practical and efficient, since it can respond to frequent changes in file heat at a reasonable cost. To be specific, our proposed incremental adjustment technique monitors the heat changes at regular intervals. The extra block deletion/migration/replication is triggered whenever the file's heat change ($\Delta = \sum |\Delta h_i| / \sum h_i$) crosses a certain threshold. This threshold (T_Δ) is a system parameter, and can be set according to the sensitivity requirements for heat variance. When the heat change exceeds this threshold, our adjustment policy would conduct based on the current storage states.

6.1 Incremental Adjustment for the First Scenario

The incremental adjustment for the first scenario is detailed in Algorithm 3. When the heat variance (Δ) exceeds a given threshold (T_Δ), the allocation scheme would be recomputed with the optimization objective expressed in Equ. 14. The updated decision variables are denoted by $x'_i,\ y'_i,\ m'_j,\ r'_j$. To alleviate the performance degradation caused by heat changes, we should adjust the storage state from $x_i,\ y_i,\ m_j,\ r_j$ to $x'_i,\ y'_i,\ m'_j,\ r'_j$.

The puzzle is that there may be two adjusted states based on our problem formulation in Section 4.2, when the migration percentage is satisfied. Different mappings from the original states to the adjusted states would lead to diverse adjustment costs, such as the extra space cost and the transmission bandwidth. The reason is that the number of block migration/replication/deletion operations is different in the two mappings. To reduce these resource-consuming operations, we compare the different mappings from the original states to the adjusted states. We refer the space minimization as our main optimization objective, and try to decrease



(a) Different mappings (b) Adjustment cost

Fig. 6. Different mappings from the original states to the adjusted states, when heat degrees are changed from (1, 2, 8, 3) to (3, 7, 2, 4). The adjustment cost is divided into two categories: the extra space cost and the bandwidth consumption.

the extra space cost caused by the replication/deletion operations. When the two mappings lead to the same extra space, we consider bandwidth minimization as our secondary optimization objective, where the number of migration operations should be reduced.

Take Fig. 6 as an example, in the original file states, there are f_1, f_2, f_3 in the source server, and f_3, f_4 in the target server. The adjusted states separate the files into two parts: f_1, f_2, and f_2, f_3, f_4. When choosing the mapping denoted by the orange arrows, the original state of S_s would be mapped to the adjusted state of S_s exactly. In this case, block b_5 should be deleted directly from server S_s, while b_1 should be replicated to S_t with its copy migrating between the two servers. We assume that the size of all partitioned blocks ($b_1 \sim b_5$) is 1. Then, server S_s reduces its space cost by one block deletion (b_5), and Server S_t increases its space cost with one block replication (b_1). The conducted migration between these two servers occupies one unit of bandwidth resources (b_1). Otherwise, for the mapping represented by the blue arrows, there are in total three block migrations, i.e., b_1, b_2, and b_4, while the space cost remains unchanged for the two servers. Therefore, we would select the state mapping denoted by the orange arrows, because the two mappings lead to the same extra space, but the former causes less bandwidth consumption.

When the heat variance (Δ) exceeds the threshold (T_Δ), our adjustment algorithm would first recompute the files' and blocks' states, and derive the adjustment cost in different mappings. Our algorithm would map the states with less adjustment cost, where the recomputed Boolean variables are denoted by x'_i, y'_i, m'_j, r'_j. The adjustment process after the state mapping is listed as follows.

- For each block $b_i \in B_s$, if $m'_j = 1$, block b_j should be migrated from S_s to S_t when $u_j = 1$, and be deleted from S_s when it has been replicated at both S_s and S_t ($r_j = 1$).
- When $r'_j = 1$, block b_j would be first replicated, and then its copy should be migrated to the other server when $u_j = 1$ or $m_j = 1$.
- When $u'_i = 1$, the unaltered block b_j should be migrated back to the source server S_s when the original state is migrated ($m_j = 1$). Otherwise, if $r_j = 1$, the replicated block b_j should be deleted from the target server S_t.
- In the remaining cases, the block states remain unchanged.

6.2 Incremental Adjustment for the Second Scenario

6.2.1 Data Sketches based on Counting Bloom filters

When a file gets cold, fewer replicas are sufficient to meet the potential access requirements. Thus, part of its replicas should be deleted from the storage system for the space-saving purpose. However, the traditional Bloom filter used in our previous algorithms cannot support the item deletion function. The reason is that resetting the corresponding 1s to 0s may lead to false-negative results for other elements. Therefore, we employ the Counting Bloom Filter (CBF) [35] instead in this section to support deletion operations.

Algorithm 3: Incremental Adjustment Strategy for the first Scenario

```
Input: The heat variance $\Delta$; the variance threshold $T_\Delta$, the
       current variables for files and blocks $x_i$, $y_i$, $m_j$, $r_j$.
Output: The update storage state for all files in $F_s$.
while $\Delta > T_\Delta$ do
    Recompute the Boolean variables $x'_i$, $y'_i$, $m'_j$, $r'_j$ based
      on the state mapping;
    for $j = 1$ to $|B_s|$ do
        Compute the unultered state of block $b_j$ with
          $u_j = 1 - m_j - r_j$ and $u'_j = 1 - m'_j - r'_j$;
        if $m'_j = 1$ then
            if $u_j = 1$ then
                Migrate block $b_j$ from $S_s$ to $S_t$;
            if $r_j = 1$ then
                Delete block $b_j$ from $S_s$;
        if $r'_j = 1$ then
            if $u_j = 1$ then
                Replicate block $b_j$ from $S_s$ to $S_t$;
            if $m_j = 1$ then
                Replicate block $b_j$ from $S_t$ to $S_s$;
        if $u'_j = 1$ then
            if $m_j = 1$ then
                Migrate block $b_j$ from $S_t$ to $S_s$;
            if $r_j = 1$ then
                Delete block $b_j$ from $S_t$;
```

The CBF extends the BF by replacing each bit as a counter. In the framework of CBF, when inserting an element, the corresponding k_{CBF} counters in the vector increment by 1. In contrast, the deletion of an element will be supported via decreasing the corresponding counters by 1. In this way, the deletion of an element will not affect the membership queries of other elements. It has been proved that 4 bits for a counter are enough to achieve eligible overflow probability [33]. CBF also supports constant-time membership queries. To answer a membership query, the CBF checks the k_{CBF} corresponding counters. If all of them are non-zero, CBF judges that the queried element is a member; otherwise, negative.

We employ CBF to construct the data sketches for each candidate server in our incremental adjustments for the second scenario. We use $C\Psi$ to denote the CBF-based data sketches. According to the CBF-based server sketches, we can realize the membership queries against any data block. In addition, the update operations, especially element deletions, can be realized.

6.2.2 CBF-based Incremental Data Allocations

The incremental adjustment strategy for the second scenario is expressed in Algorithm 4. We first build an adjustment sequence HQ by sorting $|\Delta h_i|$ in a descending order (Line 1). We adjust the data allocation for files in HQ, until the variance percentage Δ is below the threshold T_Δ. For the file with the maximum heat variance (f_q), we judge whether the variance is positive

Algorithm 4: Incremental Adjustment Strategy for the Second Scenario

```
Input: The heat variance $\Delta$; the variance threshold $T_\Delta$;
       CBF-based data sketches $C\Psi$.
Output: The updated $repeat\_set(i)$ and the heat
        allocation $\gamma_i$ for file $f_i \in F_s$.
1  Build $HQ$ by sorting $|\Delta h_i|$ in descending order;
2  while $\Delta > T_\Delta$ do
3      Denote the first file in $HQ$ as $f_q$;
4      if $\Delta h_q > 0$ then
5          Get $\varphi(q,k)$, $\forall S_k \in \tilde{S}_t$ & $S_k \neq repeat\_set(q)$;
6          Build $Q_q$ by sorting $\varphi(q,k)$ descendingly;
7          for $k = 1 \to |Q_q|$ do
8              $k'$=$repeat\_set(q)[k]$, if the capacities are
                 under the constraints;
9              Calculate $AFR(q) = h_q/(|repeat\_set(q)|+1)$;
10             if $AFR(q)/(size(f_q) - \varphi(q,k')) >= \kappa$ then
11                 $repeat\_set(q)$=$repeat\_set(q) \cup \{S_{k'}\}$;
12                 Updated sketch $C\Psi_{k'}$ with added file $f_q$;
13             else
14                 break;
15     if $\Delta h_q < 0$ then
16         Get $\varphi(q,k)$, $\forall S_k \in repeat\_set(q)$;
17         for $k = 1 \to |repeat\_set(q)|$ do
18             $k' = repeat\_set(q)[k]$;
19             if $h_q/|repeat\_set(q)|(size(f_q) - \varphi(q,k')) < \kappa$
                 then
20                 $repeat\_set(q)$=$repeat\_set(q) \backslash \{S_{k'}\}$;
21                 Updated sketch $C\Psi_{k'}$ with deleted file $f_q$;
22     Adjust $\gamma_q$ by $\textsc{Heat\_Allocation}(f_q, repeat\_set(q))$.
23     $\Delta = \Delta - |\Delta h_q| / \sum_{i=1}^{|H|} h_i$;
24     $HQ = HQ \backslash \{f_q\}$;
```

or negative. If the file is getting hot, i.e., positive variance, we would find several additional candidate target servers for file replicas to alleviate the request congestion. Specifically, we get $\varphi(q,k), \forall S_k \in \tilde{S}_t$ & $S_k \neq repeat\_set(q)$, and sort the values descendingly to construct the server queue Q_q (Lines 5-6). Thereafter, we calculate the division between the split data access $AFR(q)$ and the extra space cost $size(f_q) - \varphi(q,k')$. If the division between these two parameters is larger than κ, then the file f_q would be replicated to the underloaded $S_{k'}$, i.e., the k^{th} server in Q_q, with the data sketch $C\Psi_{k'}$ being updated (Lines 7-14). The new replicas support data access requests for the hotter file f_q. This process is the supplementary operation for Algorithm 2.

In contrast, when the file f_q is getting cold, some replicas would be removed from the systems for the space-saving purpose. Specifically, for all servers that store the file replicas of f_q, i.e., $repeat\_set(q)$, we would recalculate whether or not they meet the criteria for storing replicas (Lines 18-19). If not, the occupied space would be released from the server, with sketch $C\Psi_{k'}$ being updated (Lines 20-21). After updating the replica locations $repeat\_set(q)$ for file f_q, we further leverage function $\textsc{Heat\_Allocation}(f_q, repeat\_set(q))$ (Line 22) to adjust the allocated percentage (γ_q) for each server in $repeat\_set(q)$. The heat variance Δ and the adjustment sequence HQ would also be updated (Line 23-24).

7 Performance Analysis

In this section, we perform mathematical analyses to guarantee the theoretical effectiveness of our BF-based data migration strategies. We first prove that the BF-based data sketch would negligibly impact the similarity detection through Theorem 1. After that, we demonstrate the rationality of utilizing the space-saving index through Theorem 2. At last, we analyze the time and space complexities of the provided algorithms.

7.1 Impact on Data Similarity due to Data Sketching

Theorem 1. *Given a block b that does not exist in server S_A, and the data sketch Ψ_A of S_A with length d, the probability that judging the block b belongs to server S_A through indexing the server sketch Ψ_A is negligible.*

Proof. We assume there are ε blocks stored at server S_A, and these blocks are sketched into a d-bit string (Ψ_A) by k_{BF} BF-based hash functions. When inquiring whether server S_A contains the block b, the corresponding k_{BF} positions of sketch Ψ_A would be checked. The sketch judges that this block does not belong to server S_A, if any bit at the k_{BF} hashed positions is 0. Otherwise, the BF believes that the queried block b belongs to S_A with a rate of false positives. The false positives are caused by hash conflicts, as the 8^{th} bit in Fig. 4. For block b, all of its k_{BF} hash positions in the bit vector may be set as 1 when representing other blocks in server S_A. The probability that a bit in Ψ_A is not set to 1 by the k_{BF} independent hash function is $(1-1/d)^{k_{BF}}$. When inserting the ε blocks into the sketch, the probability becomes $(1-1/d)^{\varepsilon \cdot k_{BF}}$. Then, for block b, all of its k_{BF} hash positions are projected by other blocks with a probability $p = [1-(1-1/d)^{\varepsilon \cdot k_{BF}}]^{k_{BF}}$. This probability is also called the false-positive rate [33].

When the sketch length d is set with a large value, for example, d=100000 in experiments in Section 8, the false-positive rate tends to zero. Therefore, the impact of sketch-based data similarity mining is negligible, which validates the rationality of our heuristic algorithms using Bloom filters or Counting Bloom filters. □

7.2 Accuracy of File Ranking on Space Efficiency

Theorem 2. *Given two files (f_1 and f_2) in the source server with their maximum ranking index $I(1,k_1)$ and $I_s(2,k_2)$, where $I(1,k_1) > I(2,k_2)$, then the migration of file f_1 is more space-efficient than that of file f_2 in the current step.*

Proof. We assume that the data sketches for all involved servers are set with a fixed length d. The chosen migration target for file f_1 is server S_{k_1}, and that for file f_2 is server S_{k_2}. $I(1,k_1) = \varphi(1,k_1) - \varphi(1,s)$, where $\varphi(1,k_1)$ represents the data volume of shared blocks between f_1 and F_{k_1}, and $\varphi(1,s)$ expresses the data volume of shared blocks between f_1 and $F_s \backslash \{f_1\}$.

Let $\varphi(i,k)$ denote the data volume of shared blocks between f_i and $S_k \in \tilde{S}_t$. This can be derived from the BF-based membership queries of blocks in f_i on data sketch Ψ_k. Then, the function returns the index $I(i,k)$ according to the deviation between $\varphi(i,k)$ and $\varphi(i,s)$, which reflects the saved space resources through migrating file f_i from S_s to the S_k. However, these values contain false positive results caused by hashing conflicts. We let $\widehat{\varphi}(i,k)$ and $\widehat{\varphi}(i,s)$ represent the real values of the estimated $\varphi(i,k)$ and $\varphi(i,s)$. Then $\widehat{\varphi}(i,k) = (\varphi(i,k) - p_k V_i)/(1-p_k)$, where V_i represents the data volume of file f_i, and p_k indicates the false positive ratio of membership queries on the sketch of server S_k. The aim is to prove

$\widehat{\varphi}(1,k_1) - \widehat{\varphi}(1,s) > \widehat{\varphi}(2,k_2) - \widehat{\varphi}(2,s)$ when $I(1,k_1) > I(2,k_2)$. We rewrite the space-saving ranking indexes as follows:

$$\begin{aligned}&I(1,k_1) - I(2,k_2)\\&= (1-p_{k_1})\widehat{\varphi}(1,k_1) - (1-p_{k_2})\widehat{\varphi}(2,k_2) + (1-p_s)[\widehat{\varphi}(2,s) - \widehat{\varphi}(1,s)]\\&+ (p_{k_1} - p_s)V_1 - (p_{k_2} - p_s)V_2 > 0.\end{aligned} \tag{15}$$

Based on this inequation, the real saved storage space between file f_1 and f_2 can be represented as follows:

$$\begin{aligned}&[\widehat{\varphi}(1,k_1) - \widehat{\varphi}(1,s)] - [\widehat{\varphi}(2,k_2) - \widehat{\varphi}(2,s)]\\&> p_{k_1}\widehat{\varphi}(1,k_1) - p_{k_2}\widehat{\varphi}(2,k_2) + p_s[\widehat{\varphi}(1,s) - \widehat{\varphi}(2,s)]\\&+ p_s(V_2 - V_1) + p_{k_1}V_1 - p_{k_2}V_2.\end{aligned} \tag{16}$$

When the data sketches for all involved servers (S_s, S_{k_1}, and S_{k_2}) are set with a fixed and large length d, the false positive ratios (p_s, p_{k_1}, and p_{k_2}) are all tend to zero. With this premise, $\widehat{\varphi}(1,k_1) - \widehat{\varphi}(1,s) > \widehat{\varphi}(2,k_2) - \widehat{\varphi}(2,s)$ with a high probability, and then Theorem 2 can be verified.

□

Theorem 2 can also be applied for the heat-aware file replication algorithm, where the server queue Q_i is sequenced by index $\varphi(i,k)$ for file f_i. The server with a higher $\varphi(i,k)$ is more space-efficient to store a replica of file f_i under the same conditions.

To conclude, Section 7.1 and Section 7.2 together indicate that our Jingwei method picks up a rational sketching method to reduce the computation overhead, and ranks the file migration/replication sequence rationally based on data sketching. These guarantee the theoretical effectiveness of our Jingwei in the general scenario.

7.3 Time and Space Complexity Analysis

We analyze the time and space complexity of our proposed data allocation strategies in this subsection, including the BF-based data sketch, space-saving data migration, and heat-aware data replication. The specific comparisons are shown in Table 2.

The time complexity of the BF-based data sketch is $O(|B_t|_{max} \cdot k_{BF})$, where k_{BF} indicates the number of utilized hash functions and $|B_t|_{max}$ represents the maximum number of blocks at any candidate server. Note that, the sampling technologies can be assembled to reduce the time complexity by a factor of the sample ratio. The space complexity of the BF-based data sketch is $O(d)$, where d expresses the BF length.

The time complexity of the space-saving data migration is $O(|F_s|^2 \cdot |\tilde{S_t}|^2 \cdot n_{max}k_{BF})$, where n_{max} indicates the maximum number of blocks in any file. In addition, the space complexity is $O(d|\tilde{S_t}| + |F_s||\tilde{S_t}|)$, where $d|\tilde{S_t}|$ records the server sketches and $|F_s||\tilde{S_t}|$ records the space-saving indexes.

As for the heat-aware data replication algorithm, the time complexity is $O(|F_s| \cdot |\tilde{S_t}|^2 \cdot \log_2|\tilde{S_t}|)$. Here, $O(|\tilde{S_t}| \cdot \log_2|\tilde{S_t}|)$ is caused by ordering the target servers based on the shared block volume. The complexities $|F_s|$ and $|\tilde{S_t}|$ are caused by the maximum migration times and the maximum replication times for each file in F_s. The space complexity is $O(|F_s||\tilde{S_t}|)$. Note that, our incremental adjustment strategies incrementally conduct the update operations on only a part of the original blocks. Thus, their complexities would be lower than these two algorithms.

In the case of having a great number of servers in a large storage system, both the time and space complexities would increase linearly or even quadratically. It is unavoidable that our Jingwei would require more time to conduct a feasible migration strategy. However, it may be unnecessary to incorporate all servers as our migration candidates. The reason is that the data migration between the two remote servers would involve significant network transferring cost. The common solution [36], [37] is partitioning the storage servers into smaller clusters to confine the transmission distance. Then, we can perform our data migration schemes within each cluster, which reduces the network cost and decreases the algorithm complexities significantly.

TABLE 2
Time and space complexity analysis.

Algorithm	Time complexity	Space complexity
BF-based Data Sketch	$O(\lvert B_t\rvert_{max} \cdot k_{BF})$	$O(d)$
Space-saving Data Migration	$O(\lvert F_s\rvert^2 \cdot \lvert\tilde{S_t}\rvert^2 \cdot n_{max}k_{BF}$	$O(d\lvert\tilde{S_t}\rvert + \lvert F_s\rvert\lvert\tilde{S_t}\rvert$
Heat-aware Data Replication	$O(\lvert F_s\rvert \cdot \lvert\tilde{S_t}\rvert^2 \cdot \log_2\lvert\tilde{S_t}\rvert)$	$O(\lvert F_s\rvert\lvert\tilde{S_t}\rvert)$

8 Performance Evaluation

In this section, we empirically evaluate the performance of our Jingwei strategy using a real-world dataset. We describe our experimental settings and then present the experimental results, which show the efficiency of our proposed data migration strategy over other comparison methods.

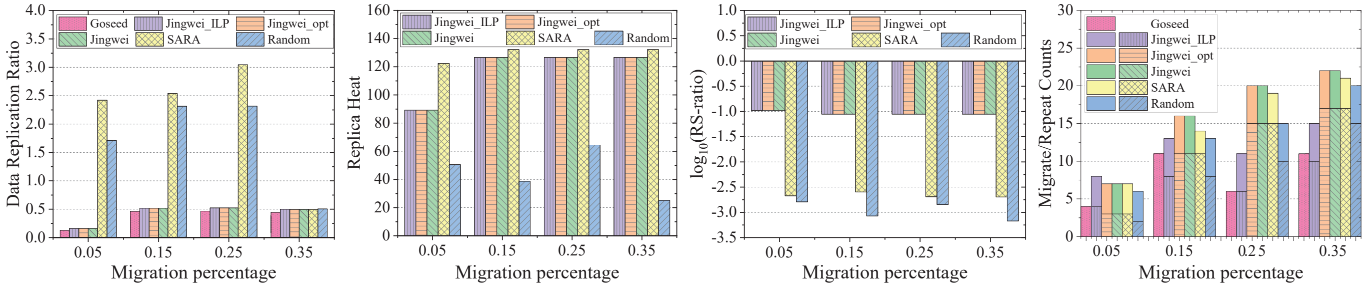
8.1 Experimental Settings

We implement a prototype system of Jingwei, which includes 11 virtual machines (VMs) to evaluate the system performance. Ten VMs represent the storage servers, which store the allocated data blocks in schemes derived from different comparison methods. One VM acts as a file requester, which sends file retrieval requests to the storage servers according to the files' heat degrees. This VM is also a management node, which maintains the affiliation information between files and blocks, as well as the mappings from each block and to their storage servers. In our prototype system, the VMs are deployed on a Desktop PC, equipped with a 3.50GHz Intel(R) Core(TM) i9-11900K CPU with 8 cores and 64GB RAM using 500GB SSD. Each VM is allocated 4GB of RAM and 30GB virtual disk drive, running Ubuntu Linux 20.04 x64. The CPU cores are shared by all VMs.

Datasets. We use the real-world dataset [38] for the evaluation to demonstrate the universality of our Jingwei strategy. The dataset is downloaded from the GitHub website, which consists of the zip-compressed source codes of 403 randomly selected projects on some hot topics, such as `Atom` [39] and `Azure` [40]. Each project contains several historical versions of the source code file, with a great number of duplicate chunks between them. There are in total 55,797 files in this dataset, with a maximum size of 30.65MB and a minimum size of 1B. We partition the files using the variable-size chunking approaches [6]. The average block size is 3.87KB, and the global deduplication ratio (the ratio of saved space after data deduplication to the original space) is 55.79%.

Comparison methods. To illustrate the performance of Jingwei more comprehensively, we consider three other comparison methods in this paper.

- **Goseed** [7], which provides an optimal solution with the commercial optimizer to minimize the extra space occupation in the migration process. However, it can only be applied to migrate files to a single empty server.
- **SARA** (Service-Aware Replication Allocation scheme), where replicas are generated for hotter files [14] and are allocated to the servers with more available service



(a) The Data Replication Ratio.

(b) The Replica Heat.

(c) The RS-ratio.

(d) The MC and RC.

Fig. 7. The performance with different migration percentages in the first scenario.

capabilities [15]. We assign the migration status of Jingwei to SARA directly to compare its performance in the file replication stage.

- **Random**, which is the baseline of all these comparisons. In the Random method, files are ranked randomly and then migrated/replicated to a randomly chosen server.

We also compare **Jingwei_ILP** for the first scenario, which exhibits the optimal migration strategy derived from the ILP solver. In addition, the optimal result of our heuristic algorithms, **Jingwei_opt**, is also compared. It detects content similarity through pair-wise fingerprint checking, but not membership queries on the bit arrays. Thus, Jingwei_opt avoids the false positives caused by the BF-based hash mappings.

Metrics. Firstly, we verify the performance of space efficiency rationale with the ***Data Replication Ratio (DRR)***, which is defined as the ratio between the extra space cost attached by file moves and the initial space occupation at the source server. The second comparison metric is the ***Replica Heat (RH)***, which indicates the total heat degree of the file replicas. A high value of RH indicates that more replicas are generated for hot files, which is vital for the service adaptability. Furthermore, ***RS-ratio*** is a comprehensive index of the DRR and RH, which reflects the amount of RH per extra storage unit in the file replication stage. This quantifies the performance balance between the space efficiency and the service adaptability. The ***Migration Count (MC)*** and ***Replication Count (RC)*** are also considered for counting the file migration and replication when a certain amount of data has been migrated from the overload source server.

To evaluate the performance of our incremental adjustment strategies, we consider the ***Data Unavailable Ratio (DUR)***, which is triggered by the heat variance, so that the total available service capacities cannot meet the frequent file requests. We also emphasize the adjustment cost, which is mainly composed of the extra space cost and bandwidth consumption. We continue to utilize the ***DRR*** metric to represent the extra space cost, and we newly define ***Extra Migration Volume (EMV)*** to represent the bandwidth consumption.

For our prototype system, we evaluate the file retrieval behaviors. The file requests are sent with different frequencies based on their heat degrees. Using the `iPerf` and `ping` tools, we test the bandwidth between these VMs is 543Mbps, and the network latency is 0.275ms on average over ten rounds. Then, we evaluate the file ***Retrieval Throughput*** and ***Retrieval Delay*** for different comparison methods.

Parameter setting. We first unzip and partition the files in the dataset into variable-size blocks. Each block is represented by its fingerprint using MD5 [31]. We sketch the data blocks at each involved server using Bloom filters with $k_{BF} = 2$ and $d = 100000$ by default. We employ the widely utilized Zipf distribution to govern the file popularity in heat degree generation [41], where the concentration degree of data access is set as 1. We randomly allocate the files to 10 servers. The storage and service capacities of the migration targets are set to twice the initial usage. We set $\lambda = 0.4$ and $\kappa = 0.02$ to unify the RC value as 4 for all comparison methods in the first scenario. For the second scenario, we set $\kappa = 5$ by default, and the replication count (RC) follows that of Jingwei for other comparison methods. The unity of RC facilitates the performance comparisons at the file replication stage.

8.2 Numerical Results

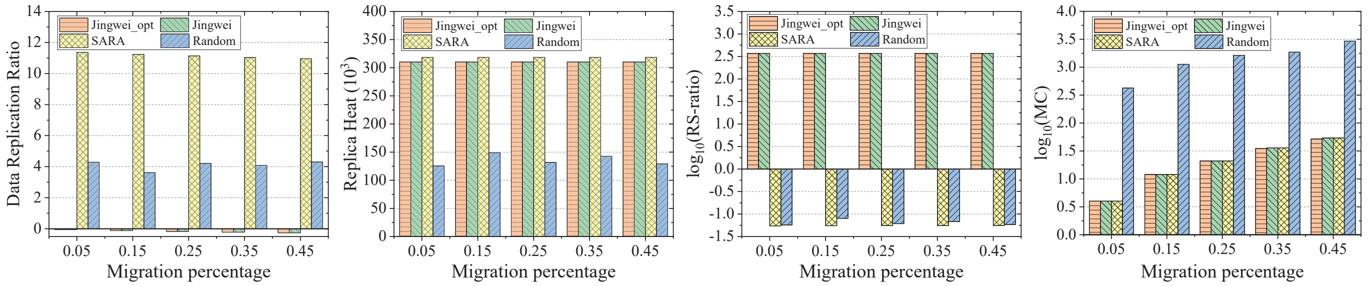
We conduct large-scale experiments to test the respective performance of Jingwei and its competitors in two migration scenarios, separately. The performance with heat variation is also exhibited.

8.2.1 Performance in the first scenario

For the first scenario, we only utilize one project in the dataset. The reason is that the performance advantages of Jingwei and its competitors are more significant for a dataset wherein the files are pretty similar with numerous shared blocks. Otherwise, the migration can be viewed as separating the two irrelevant sub-datasets without data-sharing dependencies. We utilize the `Azure` project [40], there are 20 files with 16,328 unique blocks, where each file contains a maximum of 3,635 blocks and a minimum of 18 blocks.

Fig. 7 depicts the performance of Jingwei and its competitors in the first scenario. The performance of data replication ratio (DRR) is exhibited in Fig. 7(a). The data volume when DRR=1 represents the original file volume in the source server without data deduplication. Jingwei consistently achieves a similar DRR compared with Jingwei_ILP and Jingwei_opt, while only about 6% extra DRR is triggered compared with the Goseed method. This verifies the space efficiency of Jingwei, which does not cause much additional space overhead during data replications. By contrast, SARA and Random, which construct file replicas without consideration of data deduplication, lead to 2× and even 3× space occupation in the worst cases.

Fig. 7(b) reflects the replica heat (RH) and Fig. 7(c) exhibits the RS-ratio. The Jingweis perform well in both of these two metrics. The reason is that, Jingweis prefer to replicate files with a relatively high heat degree, and allocate the replicas to the server with high similarity. The SARA method, although achieving higher RH through replicating the hottest files, performs unsatisfactorily in terms of the RS-ratio (around $10^{1.7}$ times lower than that of Jingwei). It is because the replica allocation of SARA



(a) The Data Replication Ratio.
(b) The Replica Heat.
(c) The RS-ratio.
(d) The Migration Count.

Fig. 8. The performance with different migration percentages in the general scenario.

TABLE 3
The performance of Jingwei with different λ (the adjusting parameter in the optimization objective).

	λ						
	0.1	0.4	0.7	1	1.3	1.6	1.9
DRR	0.471	0.500	0.689	0.689	0.730	0.730	0.747
RH	162.64	126.60	328.97	328.97	343.50	343.50	347.30
RS-ratio	0.1245	0.0882	0.1724	0.1724	0.1697	0.1697	0.1678
MC	12	10	10	9	9	9	8
RC	3	5	10	10	11	11	12

TABLE 4
The performance of Jingwei with different κ (the unit-heat value that controls the necessity of file replication).

	κ				
	7	6	5	4	3
DRR	-0.2235	-0.2233	-0.2229	-0.2228	-0.2223
RH(10^3)	303.04	306.64	310.25	315.23	321.17
RS-ratio	463.31	416.52	370.82	316.26	256.89
MC	36	36	36	36	36
RC	16,474	16,648	16,899	17,243	17,822

considers just the available service capacities, but ignores the potential space reduction with deduplication technologies.

The migration and replication times are finally counted in Fig. 7(d). The migration count (MC) is represented by histograms with patterns, while the replication count (RC) is not. The Goseed method only considers the migration stage, thus with the RH always being zero. We adjust parameters λ and κ to align the RC of the comparison methods as 4, which avoids the performance impact caused by the RC variance. When the migration percentage is 25%, about 25% (5/20) file replicas are extra generated in Jingwei_ILP, with only 5.7% of the extra space cost compared with Goseed (as shown in Fig. 7(a)). This exhibits that Jingweis conduct space-saving file replications. The MCs of our heuristic methods (Jingwei and Jingwei_opt) are sometimes higher than the optimal Jingwei_ILP. The reason is that the space-efficient data migration of the heuristic methods may not be globally optimal. Some extra migration of similar files may increment the migration count, but fortunately, it has little benefit on other metrics. It is because the extra migrated files may have numerous shared blocks with their targets.

Table 3 shows the performance of Jingwei with different λ when the migration percentage is fixed as 0.35. Note that, λ is an adjusting parameter to adapt to different optimization tendencies for the space efficiency and the service adaptability rationales. With the increase of parameter λ, the derived data migration schemes would tend to enhance the service adaptability rationale. To be specific, the RC and DRR increase gradually, which indicates that more file replicas are generated to apportion the file requests.

8.2.2 Performance in the second scenario

For the general migration scenario, all files in the dataset are initially randomly allocated to ten servers to construct the original storage states. Goseed and Jingwei_ILP are not compared in this subsection, because they are only applicable to the first migration scenario. The migration count (MC) of SARA follows that of Jingwei. The reason is that the SARA method does not involve the migration stage. We assign the migration states of Jingwei to SARA directly, so as to test its performance in replica generation.

Fig. 8 illustrates the evaluation performance for the general scenario. Specifically, in Fig. 8(a), Jingwei_opt and Jingwei achieve the DRR with a negative value, which means that the total space occupation dramatically decreases after data migration and replication. This benefits from the similarity-aware file allocation and verifies the space efficiency of our Jingweis. Furthermore, the saved space progressively increases as the migration percentage grows up. When 45% of data migrates, about 26.3% of the occupied space can be freed from the source server. However, the methods without deduplication incorporated, i.e., SARA and Random, lead to several times storage occupation. Fig. 8(b) illustrates that SARA outperforms others in terms of the replica heat (RH). The reason is that it chooses the hottest files to replicate, which rises the average RH for each replica. Jingwei's RH follows that of SARA because it considers both the heat degree and the extra space cost that each replica requires.

Jingwei achieves absolute advantages in RS-ratio, as shown in Fig. 8(c). Specifically, the RS-ratio of Jingwei is around 10^4 times higher than that of SARA and Random. The reason is that the replica allocations of the latter two methods fail to realize the space reduction through similarity detection between the migrated files and data in the candidate targets. Fig. 8(d) further illustrates the MC performance in the general scenario. When M=5%, the migration task of Jingweis can be accomplished by migrating four files ($\log_{10} 4$=0.6), while the Random method requires more than 400 times. The reason is that Jingweis tend to migrate files that release more space from the source server, which accelerates the migration process. The MC of Jingwei is slightly higher than that of Jingwei_opt. This phenomenon is caused by the potential false positives of Bloom filters. Such false positives may disorder the file ranking in Algorithm 1. We do not compare the RC performance in the general scenario. The reason is that the RCs are all kept as a constant value for these comparison methods.

The performance of Jingwei with different κ values is exhibited in Table 4. The value of κ exploits the necessity of file replication. With a smaller κ, more file replicas would be generated to amortize the frequent file requests. As κ decreases

TABLE 5
The performance of Jingwei with different sample ratios.

	Method	Sample ratio				
		1	1/4	1/16	1/64	1/256
DRR	Jingwei_opt	-0.2239	-0.2198	-0.1837	-0.1553	-0.1184
	Jingwei	-0.2229	-0.2196	-0.1813	-0.1544	-0.1098
RH(10^3)	Jingwei_opt	310.25	310.24	309.94	309.62	309.24
	Jingwei	310.25	310.24	309.94	309.62	309.24
RS-ratio	Jingwei_opt	370.82	368.81	358.68	351.86	349.80
	Jingwei	370.82	368.81	360.44	350.44	346.84
MC	Jingwei_opt	35	52	88	103	183
	Jingwei	36	52	91	104	219
RC	Jingwei_opt	168,96	168,93	168,53	168,38	166,84
	Jingwei	168,99	168,93	168,48	168,33	165,91

TABLE 6
The performance of Jingwei with different BF lengths.

	BF length					
	12500	25000	50000	100000	200000	400000
DRR	-0.1323	-0.1355	-0.2228	-0.2229	-0.2237	-0.2239
RH(10^3)	298.04	308.19	310.25	310.25	310.25	310.25
RS-ratio	354.14	368.49	370.82	370.82	370.82	370.82
MC	1232	599	38	36	36	35
RC	15,834	16,410	16,899	16,899	16,899	16,896

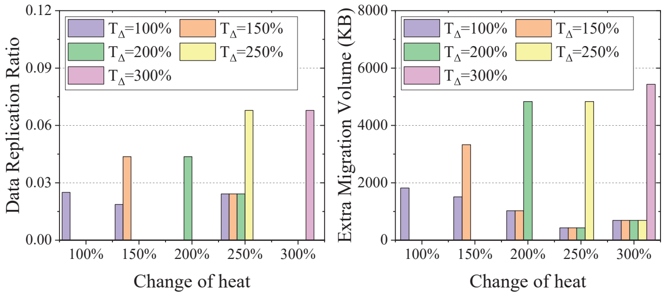
from 7 to 3, the RC grows up from 16,474 to 17,822, and the RH increases from 303.04×10^3 to 321.17×10^3. These indicate that more replicas are generated with a small κ, especially for hot files. The smallest DRR and largest RS-ratio are achieved when $\kappa = 7$. The reason is that files are replicated in a priority of high heat degree and less extra space cost. The first few generated replicas provide the highest efficiency in space saving and request apportionment.

Table 5 shows the performance of Jingwei with different sample ratios when the migration percentage is fixed at 0.35. Note that, the sampling technologies can be further assembled to alleviate the computational overhead, especially for real-world storage systems with a large number of variables and constraints. One notable change is that the MC increases gradually with fewer blocks being sampled for similarity detection. The leading cause is that the sampling technologies would weaken the accuracy in file rankings. To be specific, the sampled blocks in files and servers are chosen randomly, and the chosen blocks in files may not be sampled at their optimal targets. This retards the migration process with less space being freed from the source in each migration.

Thereafter, the performance of Jingwei with different BF lengths is exhibited in Table 6, with the migration percentage being fixed as 0.35. As the BF length grows from 12,500 to 400,000, the MC is abbreviated from 1232 to 35. Note that, the MC of Jingwei_opt is exactly 35 under the same conditions. This declares that the impact of false positives can be alleviated with a larger bit array. The metrics are impacted seriously when the BF length is set as 12,500. The reason is that a large number of false positives appear in similarity detection, when data blocks are mapped to a Bloom filter with this short length.

8.2.3 Performance with heat varies

To test the performance with heat varies, we fix the migration percentage as 0.35 and change the heat degree up or down by [1.2, 1.5, 2, 2.5, 3, 3.5, 4, 5] times randomly. We evaluate the data unavailable ratio (DUR) against heat variation to illustrate its impact on file access. Note that, we do not exhibit the DUR value for the first scenario, because the ILP formulation constraints the occupation of the service capacities. Then, all file requests can be responded to within the service capabilities. We also present the adjustment cost (DRR and EMV) to ascertain the effectiveness of our adjustment strategies. The comparison methods include: 1) the Jingwei method without incremental adjustments, which is denoted by No_incre; 2) the Jingwei method integrated with the incremental adjustments, and the variance threshold T_Δ varies from 100% to 300%. For the incremental adjustments, no action will be taken until the heat change (Δ) reaches its threshold T_Δ, and the adjusting stops when the heat variance falls below the threshold.



(a) The data replication ratio. (b) The extra migration volume.

Fig. 9. The adjustment cost in the first scenario.

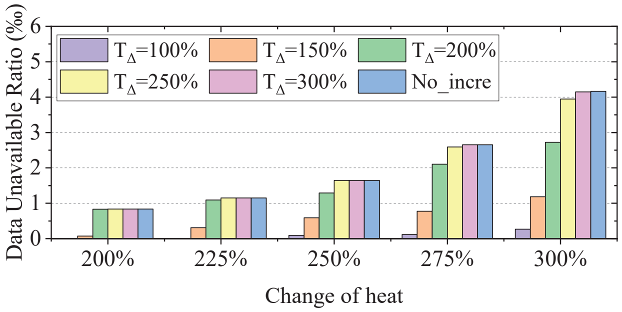
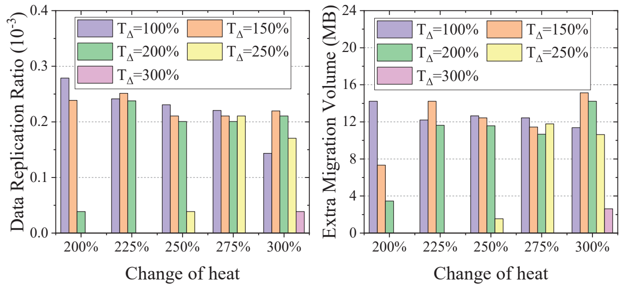


Fig. 10. The data unavailable ratio in the second scenario.

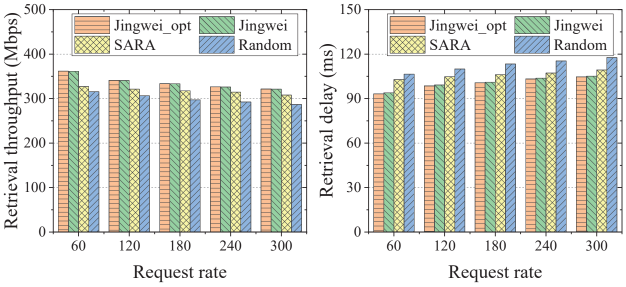
Fig. 9 illustrates the data replication ratio (DRR) and extra migration volume (EMV) of our incremental adjustments for the first scenario. We can observe that when the storage system suffers from a drastic heat change, for example, with 250% or 300%, the incremental method with a larger T_Δ would incur more block replication and migration per time. The reason is that the significant heat variance would widen the gap between the original allocation scheme and the recomputed scheme. One noticeable point lies in that the system storage volume maintains unchanged when hundreds of data blocks are migrated. It may be because the extra occupied volume taken up at the migration target is the same as the released volume from the migration source.

The data unavailable ratio (DUR) in the second scenario is exhibited in Fig. 10. All methods experience gradual increments in DUR when the heat change grows up. This reflects the severe impact of heat variance on the system performance. The DUR of the methods with incremental adjustments (T_Δ) follows that of No_incre until the heat change reaches their corresponding variance threshold. Then our incremental adjustments are employed to add or delete file replicas adaptively. The method with $T_\Delta = 100\%$ generates the lowest DUR (always less than 0.3‰) compared to other methods. The reason is that this method adjusts the file allocations more promptly and thoroughly with a small variance threshold. On the contrary, when the threshold is large, e.g., $T_\Delta = 300\%$, the DUR value is close to that of No_incre. The reason is that this incremental method works when the heat



(a) The data replication ratio. (b) The extra migration volume.

Fig. 11. The adjustment cost in the second scenario.



(a) The file retrieval throughput. (b) The file retrieval delay.

Fig. 12. The file retrieval throughput and delay.

variance reaches $\Delta = 300\%$, and stops adjusting when the heat change falls below the threshold. Therefore, the storage system still undertakes significant heat variance, which is not conducive to data availability.

Fig. 11 thereafter highlights the adjustment cost for the second scenario. The DRR for all methods is below 0.3×10^{-3}, which is relatively smaller than that of the first scenario. The reason is that more similar files and candidate servers are involved in the second scenario, where the adjustment on a small part of data blocks can change the state of a large number of files. The values of DRR and EMV are relatively similar in different methods, and the advantage in methods with a small threshold (T_Δ) is less obvious than that in the first scenario. The intrinsic reason is that the adjusted schemes are recomputed based on the ILP technologies for the first scenario, where all methods share the same recomputed states. A large heat variance would widen the gap between the original scheme and the recomputed scheme. However, the adjustments in the second scenario only deal with the newly appeared heat variance incrementally, leading to indistinctive adjustment cost for methods with different thresholds.

8.2.4 Performance of the prototype system

Finally, Fig. 12 reports the average file retrieval throughput and delay for the comparison methods. With the request rate increasing from 60 to 300 per minute, the retrieval throughput of all methods decreases, while the retrieval delay grows up gradually. The reason is that the frequent file requests may congest the transmission links to servers with limited service capacities. The rational file replication schemes can apportion the requests to different servers, so that the congestion can be alleviated. To be specific, the retrieval throughput of our Jingwei strategy can reach up to 333.5Mbps when the arrival rate is 180 per minute. It is 12.3% higher than that of the Random method, and 5.2% higher than that of the SARA method. Furthermore, the rational file replication of Jingwei decreases the retrieval delay by 5.1% compared with SARA, and 11.2% compared with Random. Note that, the performance improvement is realized when the occupied storage space of our Jingwei is far below that of others, as shown in Fig. 8(a). This strongly demonstrates that our Jingwei can retrieve files effectively, even though its storage space is limited.

In summary, Jingwei realizes an effective combination of space efficiency and service adaptability, enabling it to generate replicas for hot files with less extra space cost. To be specific, Jingwei generates 25% replicas with only 5.7% of the extra space utilization compared with Goseed. The incremental adjustments can handle up to a 250% of heat variance by replicating around 7% of the original data volume. With the small extra space cost, the file retrieval throughput of Jingwei can reach up to 333.5Mbps, which is 12.3% higher than that of the Random method.

9 Discussion

Several uninvolved aspects of our Jingwei strategy warrant further discussion. We introduce them from two design standpoints, which also suggest avenues for future work.

Diverse migration policies. we carry out in-depth literature collection and policy classification on the migration policies in deduplicated storage systems, including *replication cost* [7], [12], *fault tolerance* [42], [43], *migration traffic* [44], *load balance* [7], [44], [45], and *energy consumption* [43], [46].

The replication cost is the total size of duplicated blocks that are created as a result of migrating or replicating files [7], [12]. The energy consumption is considered to be saved by reducing data volumes for running large storage systems [43], [46]. These two policies are in reality the same as our emphasized *space efficiency* rationale. As for migration traffic [44], i.e., the amount of data that is moved across servers, we think it is implicitly consistent with our *space efficiency* rationale. The intrinsic reason is that replicating a block means transmitting this replica across the network, leading to more migration traffic.

The fault tolerance policy [42], [43] can be enhanced by generating data replicas, which has some parallels with our *service adaptability* rationale. It is because the generated data replicas in our work provide the opportunity for block copies to work when a block suffers from hardware failure or software crash. Load balance [7], [44], [45] is a major concentration in distributed storage systems, which often conflicts with our *space efficiency* rationale. To be specific, the system's space cost can be minimized by mapping all files to a single server, which enables detection and deletion of all duplicate blocks. However, this approach results in poor load balancing as only one server is utilized while others remain under-utilized. We only confine the server load below the capacity constraints in this paper. The reason is that the strictly balanced server load may miss the opportunity of generating file replicas with less extra space cost.

These guideline policies more or less provide insight to improve the system performance for the deduplicated storage systems. We believe the incorporation or trade-off of these policies provides an exciting avenue for future work.

Extended Scenarios. In this paper, we focus on a class of deduplication storage systems that holds all blocks of a file on one server [3], [5], [7], [24]. In such a distributed setting with multiple storage servers, every incoming file is allocated to a single storage server only. This enables a single disk access for block lookup per file instead of per block. Jingwei can also be extended to handle the scenarios where blocks of a file are distributed across servers. In such a scenario, the shared blocks are not necessary to be copied at both the source server and the target server during data migration. The reason is that these shared blocks can be retrieved from any server in the system, without the constraint of the server-level file integrity. This is essentially a special case of the problem presented in our work, which can be solved by releasing some constraints of Jingwei.

Specifically, we can modify the Jingwei strategy by ignoring the affiliation relationships between files and blocks, and each block is set as the unit of migration and replication. Specifically, in the first-phase data migration process, when the source server is overloaded, the blocks in the source server can be migrated out to the underweighted servers until the pre-defined migration percentage is reached. For the second-phase data replication, we first calculate the heat degree of each block, which is the heat sum of all its affiliated files. Then, we can directly replicate the popular blocks to the underweighted servers. Which blocks to replicate and how many replicas for each chosen block can refer to Algorithm 2, i.e., the heat-aware data replication.

10 Conclusion

In this paper, we report Jingwei, an efficient and adaptive data migration strategy to migrate and replicate files to the proper servers. This contributes to the space efficiency and service adaptability rationales simultaneously in the deduplicated storage systems. We first design the migration strategy based on the ILP technologies when only one empty migration target is allowed. We further extend the problem into the general scenario, wherein multiple non-empty servers are available for migration. We solve the general migration using effective heuristics based on Bloom filters. To alleviate performance degradation caused by heat variance, we further propose incremental adjustment strategies to adjust the number of file replicas and their locations in an incremental manner. The trace-driven experiments show that our solution can significantly lessen the extra space cost in migration while increasing the replicas for hot files. When heat varies, our adjustment strategies can guarantee data availability with a small adjusting cost.

Acknowledgment

This work is partially supported by National Natural Science Foundation of China under Grant No. U19B2024 and No. 62002378.

References

[1] G. Cheng, D. Guo, L. Luo, J. Xia, and Y. Sun, “Jingwei: An efficient and adaptable data migration strategy for deduplicated storage systems,” in *Proc.of IEEE INFOCOM 2022, London, United Kingdom*. IEEE, pp. 1659–1668.

[2] S. Li and T. Lan, “Hotdedup: Managing hot data storage at network edge through optimal distributed deduplication,” in *Proc.of 39th IEEE INFOCOM 2020, ON, Canada*. IEEE, pp. 247–256.

[3] G. Cheng, D. Guo, L. Luo, J. Xia, and S. Gu, “Lofs: A lightweight online file storage strategy for effective data deduplication at network edge,” *IEEE Transactions on Parallel & Distributed Systems*, vol. 33, no. 10, pp. 2263–2276, 2022.

[4] Y. Zhang, Y. Wu, and G. Yang, “Droplet: A distributed solution of data deduplication,” in *Proc.of 13th ACM/IEEE GRID 2012, Beijing, China*. IEEE Computer Society, pp. 114–121.

[5] B. Balasubramanian, T. Lan, and M. Chiang, “SAP: similarity-aware partitioning for efficient cloud storage,” in *Proc.of IEEE INFOCOM 2014, Toronto, Canada*. IEEE, pp. 592–600.

[6] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Q. Liu, and Y. Zhang, “Fastcdc: a fast and efficient content-defined chunking approach for data deduplication,” in *Proc.of USENIX ATC 2016, CO, USA*. USENIX Association, pp. 101–114.

[7] A. Nachman, G. Yadgar, and S. Sheinvald, “Goseed: Generating an optimal seeding plan for deduplicated storage,” in *Proc.of 18th USENIX FAST 2020, CA, USA*. USENIX Association, pp. 193–207.

[8] T. Qu, D. Guo, Y. Shen, X. Zhu, L. Luo, and Z. Liu, “Minimizing traffic migration during network update in iaas datacenters,” *IEEE Transactions on Services Computing*, vol. 12, no. 4, pp. 577–589, 2019.

[9] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, “Ceph: A scalable, high-performance distributed file system,” in *Proc.of OSDI 2006, WA, USA*. USENIX Association, pp. 307–320.

[10] S. Ghemawat, H. Gobioff, and S. Leung, “The google file system,” in *Proc.of the 19th ACM SOSP 2003, NY, USA*. ACM, pp. 29–43.

[11] D. Harnik, M. Hershcovitch, Y. Shatsky, A. Epstein, and R. I. Kat, “Sketching volume capacities in deduplicated storage,” in *Proc.of 17th USENIX FAST 2019, Boston, MA*. USENIX Association, pp. 107–119.

[12] A. Duggal, F. Jenkins, P. Shilane, R. Chinthekindi, R. Shah, and M. Kamat, “Data domain cloud tier: Backup here, backup there, deduplicated everywhere!” in *Proc.of USENIX ATC 2019, WA, USA*. USENIX Association, pp. 647–660.

[13] C. Hamdeni, T. Hamrouni, and F. B. Charrada, “Data popularity measurements in distributed systems: Survey and design directions,” *Journal of Network and Computer Applications*, vol. 72, pp. 150–161, 2016.

[14] X. Wei and Y. Wang, “Popularity-based data placement with load balancing in edge computing,” *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 397–411, 2023.

[15] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, “CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster,” in *Proc.of the IEEE International Conference on Cluster Computing, Crete, Greece*. IEEE Computer Society, 2010, pp. 188–196.

[16] H. Shen, “An efficient and adaptive decentralized file replication algorithm in P2P file sharing systems,” *IEEE Transactions on Parallel Distributed Systems*, vol. 21, no. 6, pp. 827–840, 2010.

[17] L. Lee, P. Scheuermann, and R. Vingralek, “File assignment in parallel I/O systems with minimal variance of service time,” *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 127–140, 2000.

[18] T. Janaszka, D. Bursztynowski, and M. Dzida, “On popularity-based load balancing in content networks,” in *Proc.of 24th International Teletraffic Congress, ITC, Kraków, Poland*. IEEE, 2012, pp. 1–8.

[19] K. Zhou, Y. Zhang, P. Huang, H. Wang, Y. Ji, B. Cheng, and Y. Liu, “LEA: A lazy eviction algorithm for SSD cache in cloud block storage,” in *Proc.of 36th IEEE ICCD 2018, FL, USA*. IEEE Computer Society, 2018, pp. 569–572.

[20] M. Ma and V. W. S. Wong, “An optimal peak hour content server cache update scheduling algorithm for 5g hetnets,” in *Proc.of 2019 IEEE ICC 2019, Shanghai, China*. IEEE, pp. 1–6.

[21] N. Zhao, H. Albahar, S. Abraham, K. Chen, V. Tarasov, D. Skourtis, L. Rupprecht, A. Anwar, and A. R. Butt, “Duphunter: Flexible high-performance deduplication for docker registries,” in *Proc.of USENIX ATC 2020*. USENIX Association, pp. 769–783.

[22] Z. Cao, S. Liu, F. Wu, G. Wang, B. Li, and D. H. C. Du, “Sliding look-back window assisted data chunk rewriting for improving deduplication restore performance,” in *Proc.of 17th USENIX FAST 2019, Boston, MA*. USENIX Association, pp. 129–142.

[23] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezis, and P. Camble, “Sparse indexing: Large scale, inline deduplication using sampling and locality,” in *Proc.of 7th USENIX FAST 2009, CA, USA*. USENIX, pp. 111–123.

[24] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge, “Extreme binning: Scalable, parallel deduplication for chunk-based file backup,” in *Proc.of 17th IEEE/ACM MASCOTS 2009, London, UK*. IEEE Computer Society, pp. 1–9.

[25] D. Huang, D. Han, J. Wang, J. Yin, X. Chen, X. Zhang, J. Zhou, and M. Ye, “Achieving load balance for parallel data access on distributed file systems,” *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 388–402, 2018.

[26] W. Zhong, S. Xie, K. Xie, Q. Yang, and L. Xie, “Cooperative P2P energy trading in active distribution networks: An milp-based nash bargaining solution,” *IEEE Transactions on Smart Grid*, vol. 12, no. 2, pp. 1264–1276, 2021.

[27] R. M. Karp, “Reducibility among combinatorial problems,” in *Proc.of 50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art, 2010*. Springer, pp. 219–241.

[28] Cplex optimizer. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[29] Introduction to lp_solve 5.5.2.11. [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[30] The fastest mathematical programming solver. [Online]. Available: http://www.gurobi.com/

[31] R. L. Rivest, “The MD5 message-digest algorithm,” *RFC*, vol. 1321, pp. 1–21, 1992.

[32] D. E. E. III and P. E. Jones, “US secure hash algorithm 1 (SHA1),” *RFC*, vol. 3174, pp. 1–22, 2001.

[33] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, “Optimizing bloom filter: Challenges, solutions, and comparisons,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[34] L. Luo, D. Guo, J. Wu, O. Rottenstreich, Q. He, Y. Qin, and X. Luo, “Efficient multiset synchronization,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1190–1205, 2017.

[35] L. Fan, P. Cao, J. M. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[36] S. Li, T. Lan, B. Balasubramanian, M. Ra, H. W. Lee, and R. K. Panta, “Ef-dedup: Enabling collaborative data deduplication at the network edge,” in *Proc.of 39th IEEE ICDCS 2019, TX, USA*. IEEE, pp. 986–996.

[37] S. Li, T. Lan, B. Balasubramanian, H. W. Lee, M. Ra, and R. K. Panta, “Pushing collaborative data deduplication to the network edge: An optimization framework and system design,” *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 2110–2122, 2022.

[38] “Popular topics on github.” https://github.com/topics.

[39] “Public repositories on atom.” https://github.com/topics/atom.

[40] “Public repositories on azure.” https://github.com/topics/azure.

[41] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, and L. Dong, “Popularity-driven coordinated caching in named data networking,” in *Proc.of ANCS 2012, TX, USA*. ACM, pp. 15–26.

[42] W. Leesakul, P. Townend, P. Garraghan, and J. Xu, “Fault-tolerant dynamic deduplication for utility computing,” in *Proc.of 17th IEEE ISORC 2014, NV, USA*. IEEE Computer Society, pp. 397–404.

[43] W. Leesakul, P. Townend, and J. Xu, “Dynamic data deduplication in cloud storage,” in *Proc.of 8th IEEE SOSE 2014, Oxford, United Kingdom*. IEEE Computer Society, pp. 320–325.

[44] R. Kisous, A. Kolikant, A. Duggal, S. Sheinvald, and G. Yadgar, “The *what*, the *from*, and the *to*: The migration games in deduplicated systems,” *ACM Transactions on Storage*, vol. 18, no. 4, pp. 31:1–31:29, 2022.

[45] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, “Grid resource management - CRUSH: controlled, scalable, decentralized placement of replicated data,” in *Proc.of the ACM/IEEE Conference on High Performance Networking and Computing, Tampa, FL, USA*. ACM Press, 2006, p. 122.

[46] H. Li, M. Dong, X. Liao, and H. Jin, “Deduplication-based energy efficient storage system in cloud environment,” *Computer Journal*, vol. 58, no. 6, pp. 1373–1383, 2015.

Geyao Cheng received her BS and MS degrees in management science and engineering in 2017 and 2019, respectively, from the National University of Defense Technology, Changsha, China, where she is currently working toward the Ph.D. degree with the College of Systems Engineering. Her research interests include edge computing, deduplicated storage, and distributed system.

Lailong Luo received his Ph.D. degree, Bachelor’s and Master’s degrees from the National University of Defense Technology, China, in 2019, 2013, and 2015, respectively. He was also a visiting research scholar at the National University of Singapore, Singapore, in 2018. His research interests include probabilistic data structures and data analysis.

Junxu Xia received the BS and MS degrees in management science and engineering in 2018 and 2020, respectively, from the National University of Defense Technology, Changsha, where he is currently working toward the Ph.D. degree with the College of Systems Engineering. His main research interests include data centers, cloud computing, and distributed storage systems.

Deke Guo received the B.S. degree in industrial engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from the National University of Defense Technology, Changsha, China, in 2008. He is currently a Professor with the College of System Engineering, National University of Defense Technology. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks. He is a senior member of the IEEE and a member of the ACM.

Yuchen Sun received the B.S. degree in Telecommunication Engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2018. He has been with the School of System Engineering, National University of Defense Technology, Changsha, since 2018, where he is currently a Ph.D. candidate. His research interests include Mobile Edge Computing, Dynamic Neural Network, and Wireless Indoor Localization.