Jingwei: An Efficient and Adaptable Data Migration Strategy for Deduplicated Storage Systems

Geyao Cheng, Deke Guo, *Senior Member, IEEE*, Lailong Luo, Junxu Xia, Yuchen Sun
Science and Technology on Information Systems Engineering Laboratory, National University of Defense Technology
{chenggeyao13, luolialaong09, junxuxia14, sunyuchen18}@nudt.edu.cn, guodeke@gmail.com

***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 would attach extra space overhead. Secondly, the redundancy elimination heightens the risk of data unavailability during server crashes. The existing methods fail to tackle them at one shot. To this end, we propose Jingwei, an efficient and adaptable 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 data 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. Trace-driven experiments show that Jingwei fortifies the file replicas by 25%, while only 5.7% of the extra storage space is occupied compared with the latest "Goseed" method.**

I. Introduction

The data volume surges exponentially in the "big data" era. To handle the "big data" challenge, current storage systems mainly adopt the data deduplication technologies [1] 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 [2]. A common practice for data deduplication is to split files into multiple blocks with either fixed size [1], [3] or varied sizes [4]. By doing so, a data-sharing dependency among the files is established, and only one copy of each block is maintained in the storage system.

When a server is overloaded, part of its files must be migrated out to another server [5]. 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 in both the source server and the migration target server. Second, redundancy elimination ensures the space efficiency but makes the storage system not failure-tolerant during server crashes, rendering data unreliable and unavailable. This situation may further deteriorate when part of the files become hot. The frequent requests of such hot files may overwhelm the stored server.

Therefore, in this paper, we envision the following two rationales for data migration in deduplicated storage systems: 1) *Space Efficiency* – the introduced extra space overhead is minimized; 2) *Service Adaptability* – files are allowed to have multiple replicas for fault-tolerance, like Ceph [6] and Google [7] file systems. These two rationales, if both realized, will bring unprecedented benefits for the storage systems. To be specific, the scarce storage resources can be economized, and in the meanwhile, the concentrated data requirements of hot data 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 [8], the space reduction [5], [9], etc. However, they are oblivious of the impact of data replicas. The storage system without replicas, especially for hot files, may impact the service adaptability significantly in practice [1]. On the other hand, the popularity-aware replication managements [10]–[13] or file assignments [14] improve the service performance undoubtedly. However, they are currently not incorporated with the space reduction technologies in the deduplicated storage systems.

Inspired by these observations, in this paper, we propose Jingwei [1], an efficient and adaptable 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 data 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 data from the overloaded Server 1 to the under-utilized Server 2. Scheme (a) [5] minimizes the amount of replicated data through allocating

[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.

files with more common blocks to one server. Nevertheless, there are no data 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) [13], 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).

The major contributions are summarized as follows.

- We report Jingwei, an effective and adaptable data migration strategy in the deduplicated storage systems. As far as we know, this is the first work to jointly consider the space efficiency and the service adaptability simultaneously and realize an elegant trade-off between them.
- We design two heterogeneous migration scenarios for Jingwei to improve the strategy applicability. When an empty migration target server is allowed, we model such a problem as an integer linear programming (ILP) and solve the NP-hard problem with the ILP solver.
- We also extend this problem to a more general 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. To be specific, we leverage the space-saving migration algorithm and heat-aware data replication algorithm separately to determine the proper files to migrate or replicate. We also attach mathematical analyses on the algorithm complexity.
- 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" migration scheme.

The rest of this paper is organized as follows. Section II introduces the related works. Section III states the Jingwei overview. Section IV presents the problem formulation for the first migration scenario. Section V exhibits the heuristic algorithms for the more general migration scenario with complexity analyses. Section VI reports our experimental results, and finally, Section VII concludes this paper.

II. Related Work

Data migration in deduplicated storage systems has attracted more attention in recent years with different concerns. Harnik et al. [8] 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. [9] deploy cloud tier systems to decrease the cost of copy forward in deduplicated data migration. Nachman et al. [5] propose "Goseed" to generate an optimal plan by minimizing the extra space occupation for data migration, based on the data-sharing dependencies. However, these works

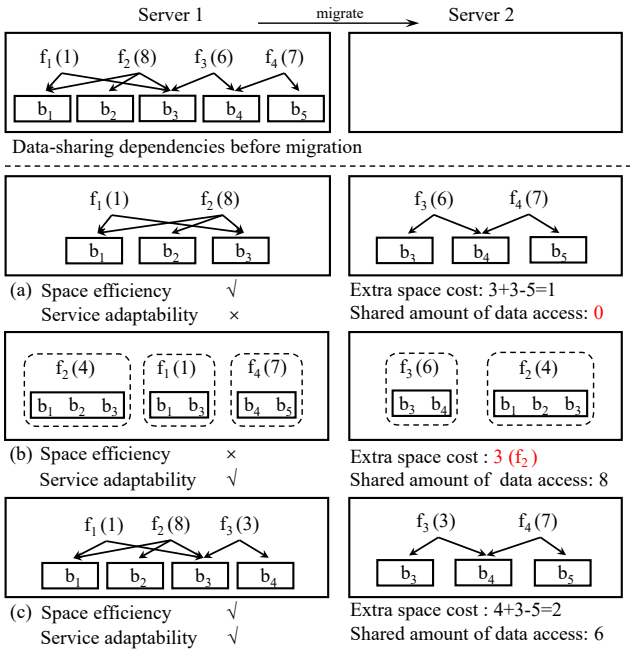


Fig. 1. The illustrative examples of the Jingwei strategy and some existing methods [5], [14]. Four files ($f_1 \sim f_4$), which are attached with heat degrees (1, 8, 6, 7), separately, are partitioned into five blocks ($b_1 \sim b_5$). The ambition is migrating a part of the four files from server 1 to server 2.

mainly focus on the space occupation, while not taking 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.

Besides, the data popularity plays a vital role for optimizing the file assignment [14], replication management [10]–[12], and load balancing [15] 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 [16], [17]. A highlighted solution to avoid service-overload is replication management [10]–[13]. Hamdeni et al. [10] provide a comprehensive survey on the data popularity and emphasize the importance of data replicas. Literature [11] increases the replicas for hot data and allocates them evenly across the storage system for a convenient retrieve. Wei et al. [12] deploy a minimal number of replicas and place them separately to those servers with the most available service capacities. Shen et al. [13] utilize file replication technology to reduce hot spots and improve file query efficiency. However, these solutions are not incorporated with deduplication technologies, which is crucial to realize the space efficiency rationale for the data storage, especially for the real-time big data with a high deduplication ratio [2].

The previous work has investigated the data migration for the space efficiency rationale through data deduplication, 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 [1] caches hot data at edge with data deduplication, which considers the data popularity as well as the space occupation in deduplicated storage sys-

TABLE I
COMPARISON OF RELATED WORKS.

Literature	*Space efficiency*	*Service adaptability*
[1], [5], [8], [9]	✓	×
[11]–[13], [18]	×	✓
this paper	✓	✓

tems. However, this work only deals with the file distribution 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 [18], 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 tarball replicas, and the Deduplication cluster stores the unique deduplicated blocks from the file tarballs. This strategy implements the prefetch/pre-construct cache algorithm based on user's access patterns, but is still not space-efficient for storing replicas of all involved files.

Unlike the existing strategies, our proposed Jingwei scheme is path-breaking 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.

III. OVERVIEW OF THE 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 through 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.
- *Service adaptability:* the data migration strategy should maintain some replicas of hot files for fault-tolerance and better user experience.

Migration Mode: Some deduplicated systems split the incoming data into blocks, and store the blocks dispersedly without the constraint of file unit [19], [20]. 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]. We track the latter mode in this paper, where a files' partitioned blocks are stored at one server, and then the data deduplication is executed at the server level. In this mode, the migration scheme should be conducted at the file layer.

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 generated at the source server. To conduct the efficient and adaptive

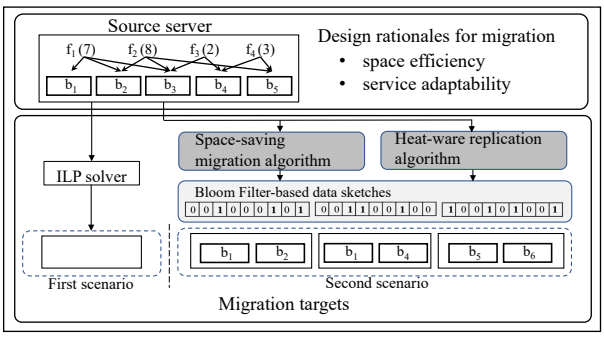


Fig. 2. The overview of the Jingwei strategy.

data migration strategy more comprehensively, we design two scenarios when migrating out a part of files from the overloaded source server. Specifically, when only one empty migration target server is allowed, we model such a 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 IV. To adapt to more migration situations and requirements, we extend the problem into a more general scenario, where any server, whether empty or non-empty, 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 a low computational overhead. The specific algorithms are exploited in Section V.

IV. MIGRATING FILES TO A SINGLE EMPTY SERVER

We first model the migration problem when only one empty target server is allowed. Specifically, we present the problem definitions in Section IV-A. With the problem being analyzed, we formulate the data migration with one single empty target server in section IV-B.

A. Problem Definition

In the overloaded source server S_s, there is a set of files $F_s = \{f_1, f_2, ...\}$ with heat degrees $H_s = \{h_1, h_2, ...\}$. Let $B_s = \{b_1, b_2, ...\}$ be the set of unique blocks that partitioned from files in F_s. 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 [1], and varies for the variable-sized block chunking algorithms [4]. 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. We do not consider the case that a file is replicated several times at one server, because it has no effect on the access shunt but only aggravates data redundancies.

The initial state of the source server S_s before data migration can be defined as a quintuple $< F_s, H_s, B_s, C_s, T_s >$, where C_s and T_s represent the space and service constraints of S_s, respectively. With this initial state, each candidate migration

file f_i in the source server S_s would be in one of the following three states after data migration:

- **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. To be specific, a block can also be migrated, replicated or unaltered, with their states being expressed by Boolean variable definitions. 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

$$m_j = \begin{cases} 1 & if\ block\ b_j\ is\ migrated. \\ 0 & otherwise. \end{cases} \tag{3}$$

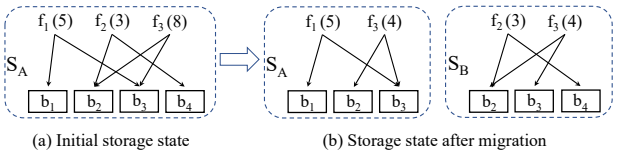
When $m_j = 1$, the block b_j should be migrated out from the source server S_s to the target server S_t. This state can only be caused by the migration of its subordinated file f_i, where $x_i = 1\ \&\ (f_i, b_j) \in\ I_s$. We further define a Boolean variable $r_j \in \{0,1\},\ \forall b_j \in B_s$, such that:

$$r_j = \begin{cases} 1 & if\ block\ b_j\ is\ replicated. \\ 0 & otherwise. \end{cases} \tag{4}$$

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 movements, 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, 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, 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 such that both the source and the target server have one copy of it, such as file f_3 in Fig. 3. In this case, all involved blocks of file f_3, i.e., $\forall b_j \in B_s$ for $(f_3, b_j) \in I_s$, would be replicated. Besides, the access requirements of the file would be spread by its replications with a distribution parameter $\gamma \in [0,1]$ (γ = 0.5 in the example of Fig. 3). This parameter is adjusted by the widely utilized load balancer in the network [21], which is responsible for load balancing in the storage systems according to the actual server load.



(a) Initial storage state (b) Storage state after migration

Fig. 3. An illustrative example of data migration from S_A to S_B.

B. 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 contains the block would be migrated to the target server:

$$m_j \le x_i,\ \forall f_i \in F_s,\ b_j \in B_s\ \&\ (f_i, b_j) \in I_s. \tag{5}$$

- 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:

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

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

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

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

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

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

- 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., realize 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., $\sum_{f_i \in F_s} y_i \times h_i$. These two rationales are normalized as follows.

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

where the parameter λ can be adjusted to adapt to different optimization tendencies for these two rationales.

With Equ. (14) as the migration objective and Equ. (5)$\sim$(13) as the constraints, the problem can be formulated as an Integer Linear Programming (ILP) problem. The ILP problem is known to be NP-hard [22], 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 long known to be NP-Complete [23]. Fortunately, commercial optimizers, like CPLEX [24], lp_solve [25], and Gurobi optimizer [26], 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.

However, the scenario with only one empty target migration server may not be applicable for the 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 the performance improvement.

V. Migrating files to Multiple Non-empty Servers

In a more general scenario, 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.

A. 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 [27] or SHA-1 [28] 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) [29], a hashing mapping method that has been widely utilized in networking and distributed systems, to represent the blocks on each candidate server. This captures the data characteristics and facilitates the 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.

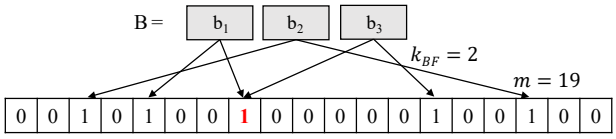


Fig. 4. An illustrative example of the BF-based data sketch. Note that the 8^{th} bit of the sketch suffers from the hash collisions.

Fig. 4 provides an illustrative example for the BF-based data sketches. 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 $m = 19$. All m 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.

The false positive is that, for any block $b \notin B$, all of its k_{BF} hash positions in the bit vector may be set as 1 when representing other blocks in set B. This is caused by the unavoidable hash conflicts, as the 8^{th} bit in Fig. 4. The false-positive rate, denoted as p, can be derived by $p=(1-(1-1/m)^{n \cdot k_{BF}})^{k_{BF}}$ [29], where n represents the number of represented blocks in set B.

B. The Effective Heuristic Algorithms based on Bloom filters

The BF-based data sketch elaborates a feasible and effective method to detect the data similarity through membership queries. According to the data sketches, we propose effective heuristic algorithms for migrating files to multiple non-empty servers. Note that, a rational migration strategy in the general scenario can potentially decrease the total space cost. This is achieved by making maximum use of the shared blocks on the target servers to rebuild the migrated files. The heuristic

Algorithm 1: Space-saving Data Migration

Input: Data sketch (Ψ) and file set (F) for S_s and $\tilde{S}_t$; the target migrtaion 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 $I(i,k),\ \forall f_i \in F_s',\ \forall S_k \in \tilde{S}_t$.
5 determine the file to migrate and its target $[f_{\hat{i}}, S_{\hat{k}}]$ in $\max_{i=1}^{|F_s|} \max_{k=1}^{|\tilde{S}_t|} I(i,k)$.
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'-$\{f_{\hat{i}}\}$;
8 updated $\Psi_{\hat{k}}$ with file set $F_{\hat{k}} = F_{\hat{k}} \cup \{f_{\hat{i}}\}$;
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 Ψ_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** $I(i,k),\ \forall\ f_i \in S_s$

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 κ.
Output: The replica locations $repeat\_set(i)$ and the heat allocation γ_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 the descending order;
5 **for** $k = 1 \rightarrow |Q_i|$ **do**
6 calculate $SDA(i) = h_i/(|repeat\_set(i)|+1)$;
7 **if** $SDA(i)/(size(f_i) - \varphi(i,k)) >= \kappa$ **then**
8 $repeat\_set(i)$=$repeat\_set(i) \cup \{S_k\}$;
9 updated the sketch Ψ_k with $F_k = F_k + \{f_i\}$;
10 **else**
11 break;
12 Adjust γ_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** γ_i, ASC

algorithms are composed of the space-saving data migration in Section V-B1 and the heat-aware data replication in Section V-B2, with complexity analyses attached in Section V-C.

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 the 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 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 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-efficient 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 through detecting the maximum $I(i,k)$ iteratively, until M percentage of the data amount in S_s has been migrated (Lines 3-9). Note that, each migration would change the storage state of both the source server and the target server. Thus, the data sketches should be locally updated on the server that files are added or released. Furthermore, the ranking index should also be updated on the related servers accordingly (Lines 7-9).

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 fault tolerance and better user experience. We present a unit-heat value (κ) to exploit the necessity of file replication. We also calculate the quotient between the split data access requirements (SDA) and the extra space the replica requires. The $SDA(i)$ is defined as the evenly split data access frequency that each replica of file f_i undertakes. Any replication is executed if the quotient value is greater than 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 a descending order to construct the server queue Q_i (Lines 1-4). The server ranking at the front of Q_i contains more similar content with f_i. For each server in Q_i, we calculate the split data access $SDA(i)$ and the extra space cost $size(f_i) - \varphi(i,k)$. If the division 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, with the data

TABLE II
TIME AND SPACE COMPLEXITY ANALYSIS.

Algorithm	Time complexity	Space complexity
BF-based Data Sketch	$O(\|B_t\|_{max} \cdot k_{BF})$	$O(m)$
Space-saving Data Migration	$O(\|F_s\|^2 \cdot \|\tilde{S}_t\|^2 \cdot n_{max} k_{BF}$	$O(m\|\tilde{S}_t\| + \|F_s\|\|\tilde{S}_t\|$
Heat-aware Data Replication	$O(\|F_s\| \cdot \|\tilde{S}_t\|^2 \cdot \log_2 \|\tilde{S}_t\|)$	$O(\|F_s\|\|\tilde{S}_t\|)$

sketch Ψ_k being updated (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 adjusts 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 of the involved servers.

C. Time and Space Complexity

We analyze the time and space complexity of the above three algorithms in this subsection as shown in Table II. 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 would reduce the time complexity by a factor of the sample ratio. The space complexity of the BF-based data sketch is $O(m)$, where m 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. Note that, after each file migration, the storage states of both the source server and the targets are updated partially. This would not augment the overall time complexity of the algorithm. In addition, the space complexity is $O(m|\tilde{S}_t| + |F_s||\tilde{S}_t|)$, where $m|\tilde{S}_t|$ 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 result from ordering the target servers based on the shared data volume. The complexity $|F_s|$ and $|\tilde{S}_t|$ is caused by the maximum migration times and the maximum replication times for each file in F_s. The space complexity is $O(|F_s||\tilde{S}_t|)$.

VI. 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.

A. Experimental Settings

Our experiments use an HP OMEN Desktop PC, equipped with an Intel(R) Core(TM) i7 CPU with 3.80GHz 8-core CPU and 64GB of RAM. The machine runs Ubuntu Linux 16.04 x64 with 4.15.0 kernel.

Datasets. We use a real-world GitHub dataset for the evaluation to demonstrate the universality of our Jingwei strategy. The dataset is downloaded on GitHub websites, which consist of the zip compressed source codes of 117 randomly selected projects on some hot topics, such as Altair [30] and Azure [31]. There are in total 20,000 files in this dataset, with a maximum size of 8.59M and a minimum size of 1B. We partition the files using the variable-sized chunking approaches [4]. They declare block boundaries based on the byte contents, which has been demonstrated to be more effective for similarity detection. The average block size is 3.14KB, and the global deduplication ratio is 45.94% for this dataset.

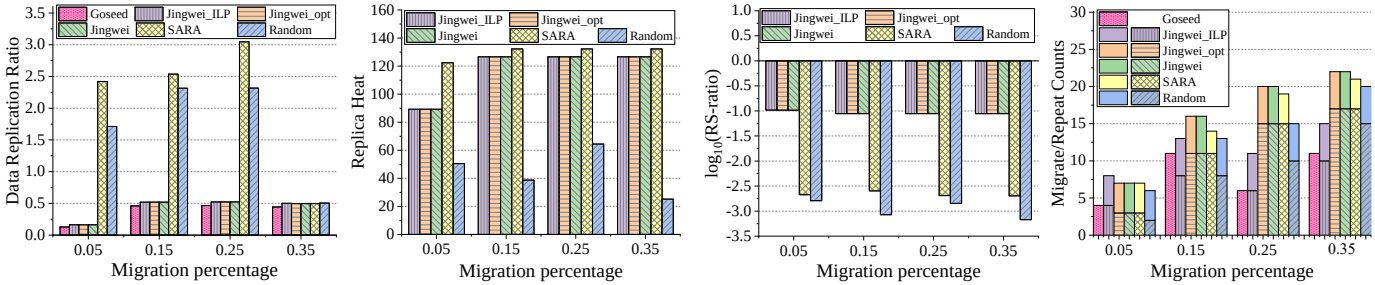
Comparison methods. To illustrate the performance of Jingwei more comprehensively, we consider three other comparison methods in this paper.

- **Goseed** [5], 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 [11] and are allocated to the servers with more available service capabilities [12]. We assign the migration status of Jingwei to SARA directly to compare the 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 checks, 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 migrations 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)/Replication Count (RC)*** is also considered, which is defined as the counts of file migration/replication when a certain amount of data has been migrated from the overload source server.

Parameter setting. We first unzip and partition the files in the dataset into variable-sized blocks. Each block is represented by its fingerprint using MD5 [27]. We sketch the data blocks at each involved server using Bloom filters with $k_{BF} = 2$ and $m = 15000$ by default. We employ the widely utilized Zipf distribution to govern the file popularity in heat degree generation [32], where the concentration degree of data access is set as 1. We set $\lambda = 0.4$ and $\kappa = 0.02$ to unify the value of RC as 4 in the first scenario with one single empty target server. For the second scenario with multiple available



(a) The Data Replication Ratio. (b) The Replica Heat. (c) The RS-ratio. (d) The MC and RC.

Fig. 5. The performance with different migration percentages in the first scenario.

target servers, we set $\kappa = 5$, and the RC follows that of Jingwei for other comparison methods. The unity of RC facilitates the performance comparisons in the file replication stage.

B. Numerical Results

We conduct large-scale experiments to test the respective performance of Jingwei and its competitors in two migration scenarios, separately.

1) Performance in the first scenario: For the first scenario, we only utilize one project in the data set. The reason is that the performance of Jingwei and its competitors is more significant for data set 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. In the "altair" project [30], 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. 5 depicts the performance of Jingwei and its competitors in the first scenario. The performance of the data replication ratio (DRR) is exhibited in Fig. 5(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 DDR is triggered compared with the Goseed method. This verifies the space efficiency of Jingwei, which does not cause much extra space overhead during data replications. By contrast, SARA and Random, which construct file replicas without considerations of data deduplication, lead to $2\times$ and even $3\times$ space occupation in the worst cases.

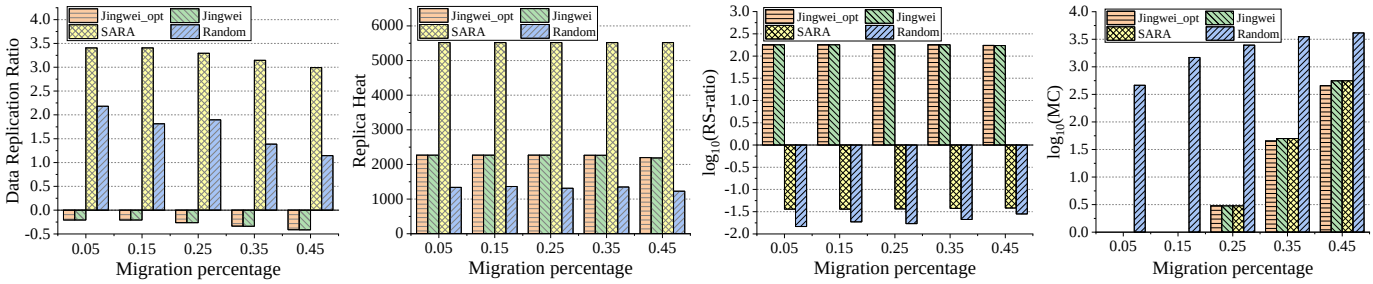
Fig. 5(b) reflects the replica heat (RH) and Fig. 5(c) exhibits the RS-ratio. The Jingweis perform well in both of these two metrics. The reason is that, Jingweis prefer to replicate files with a relatively high heat degree, and allocate the replicas to the server with high similarity. The SARA method, 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 that the replica allocation of SARA 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. 5(d), where the migration counts (MC) are represented by histograms that are filled with patterns. 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. 5(a)). This exhibits that Jingweis conduct space-saving file replications. The MC of our heuristic methods (Jingwei and Jingwei_opt) is 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 add the migration counts, but fortunately, it has little impact on other metrics. It is because that the extra migrated files may have numerous shared blocks with their targets.

2) Performance in the second scenario: For the general migration scenario, all files in the dataset are initially allocated to ten servers randomly so as 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.

Fig. 6 illustrates the evaluation performance for the general scenario. Specifically, in Fig. 6(a), Jingwei_opt and Jingwei achieve the DRR with a negative $\log_{10}$ value. This means that the total space occupation is dramatically decreased after the 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 41% of occupied space can be freed from the source server. However, the methods without deduplication incorporated, i.e., SARA and Random, lead to double or even multiple storage occupation. Fig. 6(b) illustrates that SARA outperforms others in terms of replica heat (RH). The reason is that it chooses the hottest files to replicate, which facilitates the average RH for each replica. Jingwei's RH is about half of SARA because it considers not only the heat degree but also the extra space cost that the replica requires.

Jingweis achieve absolute advantages in RS-ratio, as shown in Fig. 6(c). Specifically, the RS-ratio of Jingweis is about 5×10^3 times higher than that of SARA and around 10^4 times higher than that of Random. The reason is that the replica allocation of the latter two methods fails to realize the space reduction through similarity detection between the migrated



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

Fig. 6. The performance with different migration percentages in the general scenario.

TABLE III
THE PERFORMANCE OF JINGWEI WITH DIFFERENT SAMPLE RATIOS.

	methods	sample ratio				
		1	1/2	1/4	1/8	1/16
DDR	Jingwei_opt	-0.335	-0.328	-0.339	-0.335	-0.333
	Jingwei	-0.338	-0.326	-0.338	-0.335	-0.329
RH	Jingwei_opt	2266.4	2259.0	2249.5	2186.6	2146.8
	Jingwei	2266.2	2248.1	2246.3	2184.6	2104.7
RS-ratio	Jingwei_opt	179.1	178.5	177.7	172.8	160.7
	Jingwei	179.1	177.6	177.5	172.6	160.4
MC	Jingwei_opt	45	106	183	503	621
	Jingwei	50	186	219	531	819
RC	Jingwei_opt	7761	7698	7624	7318	7209
	Jingwei	7755	7624	7591	7290	7026

files and data in the candidate targets. Otherwise, Jingweis jointly consider the replica heat and the extra space utilized.

Fig. 6(d) further illustrates the MC performance in the second scenario. When the migration percentage is less than 15%, one single migration can accomplish the migration task ($\log_{10} 1 = 0$) for Jingweis, while the Random method requires more than 400 times. This illustrates the high efficiency of Jingweis in data migration. They 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. Note that, the MC of SARA follows that of Jingweis. The reason is that the SARA method does not involve the migration stage. We assign the migration states of Jingwei to SARA directly to compare the method performance in the file replication stage. In addition, 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.

Table III shows the performance of Jingwei with different sample ratios when the migration percentage is fixed as 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 more blocks are sampled. The leading cause is that the sampling technologies would weaken the accuracy and efficiency during file rankings. Note that, the sampled blocks in files and servers are chosen randomly, and the chosen blocks in files may not be sampled at their optimal targets. Otherwise, fortunately, the metrics except for MC closely track the non-sample situation. Note that, the false positives only retard the migration process because less space is freed from the source for each migration. The DRR would not be impacted too much when each migrated file finds its optimal target.

TABLE IV
THE PERFORMANCE OF JINGWEI WITH DIFFERENT BF LENGTHS.

	BF lengths				
	5000	10000	15000	20000	25000
DDR	-0.338	-0.338	-0.335	-0.335	-0.335
RH	2266.1	2266.0	2266.2	2266.4	2266.4
RS-ratio	179.1	179.0	179.1	179.1	179.1
MC	56	51	50	45	45
RC	7748	7753	7755	7761	7761

Thereafter, the performance of Jingwei with different BF lengths is exhibited in Table IV, with the migration percentage being fixed as 0.35. As the BF length grows from 5,000 to 25,000, the MC is abbreviated from 56 to 45. Note that, the MC of the Jingwei_opt is 45 under the same conditions. This declares that the impact of false positives can be alleviated with a longer bit array. The other metrics change slightly as the BF length increases. Table III and Table IV together prove the robustness of our Jingwei strategy. It still provides relatively satisfactory performance with lower computational consumption (sampling technologies) and less space occupation (BF-length reduction).

In summary, Jingwei realizes an efficient and adaptable migration strategy, which constructs a large number of file replicas with only limited extra storage space. To be specific, Jingwei generates 25% replicas, with only 5.7% of the extra space utilization compared with Goseed.

VII. CONCLUSION

In this paper, we report Jingwei, an efficient and adaptable 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. The trace-driven experiments under different scenarios show that our solution can significantly lessen the extra space cost in migration while increasing the replicas for hot files.

REFERENCES

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

[2] Y. Zhang, Y. Wu, and G. Yang, "Droplet: A distributed solution of data deduplication," in *Proc.of 13th ACM/IEEE International Conference on Grid Computing, GRID, Beijing, China*. IEEE Computer Society, 2012, pp. 114–121.

[3] B. Balasubramanian, T. Lan, and M. Chiang, "SAP: similarity-aware partitioning for efficient cloud storage," in *Proc.of IEEE Conference on Computer Communications, INFOCOM, Toronto, Canada*. IEEE, 2014, pp. 592–600.

[4] 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 Annual Technical Conference, USENIX ATC, CO, USA*. USENIX Association, 2016, pp. 101–114.

[5] A. Nachman, G. Yadgar, and S. Sheinvald, "Goseed: Generating an optimal seeding plan for deduplicated storage," in *Proc.of 18th USENIX Conference on File and Storage Technologies, FAST, CA, USA*. USENIX Association, 2020, pp. 193–207.

[6] 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 7th Symposium on Operating Systems Design and Implementation, OSDI, WA, USA*. USENIX Association, 2006, pp. 307–320.

[7] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *Proc.of the 19th ACM Symposium on Operating Systems Principles, SOSP, NY, USA*. ACM, 2003, pp. 29–43.

[8] D. Harnik, M. Hershcovitch, Y. Shatsky, A. Epstein, and R. I. Kat, "Sketching volume capacities in deduplicated storage," in *Proc.of 17th USENIX Conference on File and Storage Technologies, FAST, Boston, MA*. USENIX Association, 2019, pp. 107–119.

[9] 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 2019 USENIX Annual Technical Conference, USENIX ATC,WA, USA*. USENIX Association, 2019, pp. 647–660.

[10] 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.

[11] X. Wei and Y. Wang, "Popularity-based data placement with load balancing in edge computing," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021.

[12] 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 2010 IEEE International Conference on Cluster Computing, Crete, Greece*. IEEE Computer Society, 2010, pp. 188–196.

[13] 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.

[14] 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.

[15] 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.

[16] 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 International Conference on Computer Design, ICCD, FL, USA*. IEEE Computer Society, 2018, pp. 569–572.

[17] 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 International Conference on Communications, ICC, Shanghai, China*. IEEE, 2019, pp. 1–6.

[18] 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 Annual Technical Conference, USENIX ATC*. USENIX Association, 2020, pp. 769–783.

[19] 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 Conference on File and Storage Technologies, FAST, Boston, MA*. USENIX Association, 2019, pp. 129–142.

[20] 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 Conference on File and Storage Technologies, FAST, CA, USA*. USENIX, 2009, pp. 111–123.

[21] 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.

[22] 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.

[23] 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*. Springer, 2010, pp. 219–241.

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

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

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

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

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

[29] 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.

[30] "Topics on github." https://github.com/topics/chrome-extension.

[31] "Topics on github." https://github.com/topics/azure.

[32] 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 Symposium on Architecture for Networking and Communications Systems, ANCS, TX, USA*. ACM, 2012, pp. 15–26.