Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 3561887, 16 pages
https://doi.org/10.1155/2022/3561887

Research Article

Optimizing Object Storage System by the Object Multi-Tiered Balanced Organization

Lei Zhang,[1,2] **Dongyu Feng**,[3] **and Chengxi Lei**[4]

[1]*State Key Laboratory of Media Convergence and Communication (Communication University of China), Beijing 100024, China*
[2]*Institute of Computer Science and Cyberecurity, Communication University of China, Beijing 100024, China*
[3]*CETC Cloud Technology Company Limited, Beijing 100041, China*
[4]*School of Food and Advanced Technology, Massey University, Palmerston North 4442, New Zealand*

Correspondence should be addressed to Lei Zhang; leizhang@cuc.edu.cn

Received 4 March 2022; Revised 30 March 2022; Accepted 2 April 2022; Published 2 June 2022

Academic Editor: Naeem Jan

The era of big data has been constantly producing a huge amount of small files that will cause problems such as uneven distribution of copies and massive random readings and writings, resulting in delays in data accessions and degrading performances. By aiming at resolving the problems of an object storage system concerning the organization of small file objects such as layout imbalance, uneven loading, and adaptability of poor data migration, this article introduces a new strategy for object organization called object multi-tiered balanced organization strategy (OMBOS) with improved performance on massive small files in a real object storage system. Apart from the conventional strategy, the OMBOS achieves a multi-level balanced object by establishing weight indicators that reflect the comprehensive performance of nodes, including the balanced distribution of objects between replica sets and the balanced distribution of I/O requests in the replica sets as well as adapting to the equilibrium of the scale changes of the system dynamics. The OMBOS establishes a comprehensive performance evaluation model based on the fuzzy analytic hierarchy process and calculates the performance weights of the nodes of the object storages and replica sets. Then, the consistent hash-based object placement between the replica sets algorithm (BROP) and the I/O-loading balance in the replica set algorithm (IRIB) are utilized to implement performance-weighted objects in the object storage system through a tiered organization when the performance weights are taken into account. When the system scale changes flexibly, data migration is the only performed process in the nodes by adding or removing nodes, so data will not be migrated between existing nodes. Hence, the amount of data migration reaches the theoretical setting. Experiments show that the OMBOS achieves a balanced layout of objects based on performance weights, which meets the balancing and adaptability requirements of data organization. When compared to the original layout strategy through the balanced layout of objects, it performs nearly double the random access method.

1. Introduction

With the expanding development of the big data era, a huge number of small files that are needed to be used flexibly have been generated. Due to the needs of both planeness and flexibility of the data organization of the object storage system, the efforts of storing those files in an orderly manner grow significantly [1–5]. Just a few massive sets of examples dealing with those issues are Amazon Simple Storage Service (Amazon S3) [6], Lustre [7, 8], Ceph [9], Sheepdog [10], and OpenStack Object Storage (Swift) [11].

Data organization methods need to meet such goals of redundancy, balancing, and adaptability. (1) Redundancy: only a single copy of the data is stored. Once the storage device fails, the data on it will be permanently invalid. Therefore, from the fault tolerance perspective of the storage, it is necessary to ensure that data can still be accessed under the premise that certain storage devices would fail. Usually, multiple copies of data are generated or an erasure coding is adopted. So, the data layout method considers multiple copies or uses erasure codes when data are stored, which is called redundancy. This article only studies the data

organization of the object storage system using the copying method. (2) Balance: usage of storage device capacity can be balanced concerning I/O loading [12–14]. To maximize the capabilities of each storage device and optimize the overall performance of the distributed storage system, data need to be distributed evenly according to the capacity and bandwidth of the storage device to ensure the amount of data obtained by the weighted storage device. If the proportion of the data on the storage device is equal to its relative weight, then the data are called to be evenly distributed on each storage device. Hence, the data organization method should be as balanced as possible when data are distributed. Thus, the proportion and weight of the data volume in a storage device would not deviate largely. (3) Adaptability: to meet scalability requirements, large-scale distributed storage systems will add new nodes and remove old ones at regular intervals, or failure could occur regarding storage devices. To balance data distribution to maximize system performance, data need to be rebalanced. Migrating a large amount of data in a distributed storage system will inevitably bring a lot of performance overhead, causing large performance jitter and finally reducing the quality of the storage service. In order not to affect services of storage systems, the amount of data that should be migrated would be as small as possible. The optimal amount of data to be migrated is equal to the amount of the data mapped by the added or removed storage devices.

There have been currently no effective distributed file systems (DFSs) that can be applied well to large-scale small files [15, 16]. On the other hand, existing replication strategies mainly improve the balance of the data distribution according to different characteristics of a device [17–19], e.g., CPU, storage, capacity, and bandwidth. However, they did not consider the balance of the I/O request in the replica set. Besides, the existing work does not consider the jitter problem regarding the poor performance caused by the data migration after the node changes in the object storage system. Our research study presented in this paper addresses the challenges to achieve an efficient, multi-tiered balanced data replication strategy by considering device characteristics.

We proposed a new object organization strategy called object multi-tiered balanced organization strategy (OMBOS) that realizes the multi-level balanced objects, including the balanced distribution of objects among replica sets and the balanced distribution of I/O requests in the replica sets as well as a dynamic balance that adapts to alterations in a system scale. The OMBOS establishes a comprehensive performance evaluation modeling through a fuzzy analytic hierarchy process and calculates the performance weights of the nodes for the object storage and replica sets. Then, according to the performance weights, the consistent hash-based object balanced placement between replica (OBPBR) algorithm and the I/O balance placement in replica set (BPRS) algorithm realize the balanced layout of objects within the object storage system. When the system scale changes flexibly, data migration is only performed on nodes that are added or removed, so data will not be migrated between existing nodes. Hence, the amount of data migration reaches the theoretical settings. Thus, the storage system can benefit from performance improvement for future data accessions with the distribution of the balanced replication and I/O load balancing distribution by exploiting the full potentials of fast devices. Besides, it can reduce performance jitter when the node changes.

To resolve the problems mentioned above, this paper presents four main contributions as follows:

(1) We designed a comprehensive evaluation model based on cluster node performance employing the fuzzy analytic hierarchy process for the theoretical basis of the organization for small file objects utilizing performance weight.

(2) We designed object placement between replica sets algorithm (BROP), I/O-load balance in replica set algorithm (IRIB), and adaptive dynamic balancing strategy (ADBS) based on a consistent hash.

(3) We analyzed the distribution of objects among replica sets, the balance of I/O in replica sets, and adaptive dynamic balance.

(4) We implemented a prototype of the object multi-tiered balanced organization strategy based on the Ceph software storage system.

Experimental results suggest that the OMBOS can significantly improve the I/O performance of a huge amount of small files and effectively distribute data among devices.

The article is structured as follows. Section 2 provides an overview of related work. Section 3 presents the problem and establishes the model. Section 4 describes the proposed method in detail. Section 5 expresses the evaluation of the proposed method. Section 6 discusses the results of the proposed method, and Section 7 finally provides the conclusion of the research and future work.

2. Related Work

2.1. Combining Small Files into Larger Files. Although a lot of research is devoted to the design and development of optimization schemes for small file scenarios, there have been a few related works to object-based storage systems [19]. The existing methods mainly rely on the archiving system that combines small files into larger files to improve the performance of data readings and writings [15]. Hence, they use index files to save relevant information to retrieve the contents of small files from larger archived files [12, 20]. Nevertheless, the distribution of data in this type of resolution in a cluster does not consider the overall performance difference of each node, so the existing hardware resources cannot be fully utilized. Apart from these methods, our proposed method called the OMBOS fully considers the comprehensive performance difference of each node and distributes the data according to the performance weight.

2.2. The Scheme of Both Replication and Object Organization. The uniform distribution of data in the cluster is a key factor in the small file problems for distributed storage systems. From a cluster perspective, the remaining capacity of each node should be gauged that the data volume of each node is

roughly the same. On the other hand, highly correlated data should be stored in different nodes to ensure that the requested data will not cause a partition of the data, which is called a data-based perspective. When node I/O is busy and some nodes are idle, the object organization of the object storage system based on the CRUSH (controlled replication under scalable hashing) data distribution algorithm was proposed in [9, 13]. This algorithm was used to obtain a more uniform storage location, but there is a small file object layout between replica sets under the storage of the small file scenarios. Problems of imbalance, uneven distribution of reading and writing loads in the replica set, and high data migration appear as a cost.

2.2.1. The Layout of Small File Objects among Replica Sets Is Out of Balance. The layout algorithm of the object storage system takes the remaining capacity of the node as weight and then takes this weight as a constraint condition to select the storage node. Then, the uniformity of the capacity is satisfied, and it cannot effectively resolve the problem of a correlation between data objects regarding the performance of the cluster, which is called the cluster perspective. The calculations are given by

$$\frac{\text{crush}_{lh(b,x)}}{W_{osd.i}},$$
$$\text{crush}_{lh(b,x)} = \exp\left(-x \sum_{i=1}^{n} b_i\right), \tag{1}$$

where $\text{crush}_{lh(b,x)}$ is the smallest distribution of exponential random variables calculated according to the parameters b (bucket type) and parameters x (object name), and the result is distributed in the interval $[-1, 0)$ and then divided by the largest weight value of the node. The id of the target bucket or device id of the object storage is the constraint condition.

2.2.2. In the Implementation Scenario of Small File Storage. The selection of storage nodes only depends on the weight of the node, and the capacity of each node is not necessarily equal to each other. Due to the small volume of objects encapsulated by small files, the impact on the weight of each node is minimum, and the data loading cannot immediately affect the weight of each node in the cluster. According to the principle of the spatial position of data, the data relevance of consecutive requests may be very high. When writing a group of small file objects constantly continues, it is easy to store a group of objects based on a higher relevance due to the small size of the data. On the other hand, the layout of small file objects in the object storage system could be unbalanced in the nodes, resulting in the problem of excessive local I/O loading in the cluster, and the advantages of parallel processing in the distributed system cannot be fully utilized.

2.2.3. When the I/O Loading in the Replica Set Is Uneven. Existing object storage systems mostly use replicas to achieve storage availability. A replica set is composed of multiple object storage nodes and multiple replicas that ensure strong consistency through synchronous replication. Then, the selection of the master node in the replica set still uses the recursive method to select the node with the highest weight as the master node. Afterward, the requests of all the users regarding readings and writings are concentrated on the master node. On the other hand, other replica nodes only copy the data from the master node when data are written at the time of the reading request. As a result, massive loads of both readings and writings are concentrated on several nodes with larger weights, while many other node resources are idle, which results in lower resource utilization for the overall system and poor performance.

2.2.4. When Data Migration Lacks Adaptability. The current layout algorithm of the object storage system has been designed based on both tiered and recursive object placement rules and has strict requirements on the configuration of storage devices [14, 18, 21, 22]. Thus, it takes a long time to locate data. When the collection of object storage devices changes, a large amount of data needs to be migrated. Hence, the system performance jitter is hefty.

Besides, considering node weights with spare capacity as a single indicator cannot fully reflect the performance of the nodes, and it is more likely to cause loading of the storage node and performance imbalance.

The distribution of data on the physical structure is determined by the algorithm of the data organization. Thus, the balancing and adaptability of the data organization method are also achieved through the data layout algorithm. The standard hash method is the simplest data organization algorithm, which can ensure balancing, but when the system scale changes, the location of all data will change. Therefore, the existing improvement method is utilized to add additional rules to the hash method [23, 24], so that data are only migrated to new storage devices and are not migrated between old devices.

The conventional consistent hash mechanism was proposed by Karger et al. [25]. It constructed a hash function $h_1(x)$: $D \longrightarrow [0, 1]$ to uniformly map the space of the storage device $D\{d_1, \ldots, d_N\}$ to the unit ring. Hence, each mapped node represents a device. Suppose that the storage device collection of the storage system is denoted by $D_0\{d_1, \ldots, d_N\}$ and D_0 belongs to D. For each element d_i in D_0, $h_1(d_i)$ is used to map to a node the ring. Then, the hash function $h_2(x)$ is employed to evenly map the data to the ring and each node of the map represents a piece of data. Allocating the data to the storage device is represented by the closest node on the ring. Since the elements of D_0 are not evenly distributed on the ring, this mechanism virtualizes $k \log|N|$ nodes for each node where k is a constant and N is the total number of nodes in the node space to ensure the balance of data distribution. Besides, the introduction of virtual nodes enables the consistent hash mechanism to satisfy the balance of data layout. When the size of the storage system changes, the use of the closest distance principle only needs to consider the data of the neighboring nodes, that is, the data are transferred between the added or removed node or its left neighbor node and right neighbor

node. Thus, the amount of data to be migrated once is the optimal amount of data, and it has good dynamical adaptive characteristics. Therefore, the consistent hash mechanism has been widely used in the Sorrento system (a self-organizing storage system upon the cluster architecture) and the Chord protocol (a scalable peer-to-peer lookup protocol for Internet applications) [26].

However, in the actual large-scale object storage environment, the capacity and bandwidth of different nodes are usually so different. The conventional consistent hash method cannot distribute data evenly according to the weights of nodes. Hence, the weight of the node can be expressed concerning its capacity, bandwidth, or a combination of both. To adapt to the heterogeneous storage environment, a simple method is used to improve the conventional consistent hash algorithm. The virtual nodes are allocated according to the weight of the node, and the node with more weights contains more virtual nodes. However, the simple improvement of the consistent hash requires a lot of storage overhead. Suppose that the minimum weight of a node in the system is denoted by $w_{\min}$, and the number of virtual nodes allocated to this node is denoted by $k \log|N|$. Then, for the node with the weight w_i, $w_i/w_{\min} k \log|N|$ virtual nodes need to be allocated. The nodes in the node set denoted by C_0 need $\sum_{i=1}^{n} w_i/w_{\min} k \log|N|$ virtual nodes in total where n is the total number of nodes in C_0. Therefore, $\log(\sum_{i=1}^{n} w_i/w_{\min} k \log|N|)$ bits are required to distinguish all virtual nodes. In heterogeneous storage systems with large weight differences, this method requires the introduction of many virtual nodes, occupying a large amount of storage space, which is intolerable by the storage system.

Brinkmann et al. [27] transformed the data layout problem in a heterogeneous environment into a homogeneous one. Hence, the tiers are based on the remaining capacity of the nodes, and the capacity of all nodes in each tier is the same. Starting from layer 0, a consistent hash layout strategy is used in each layer to distribute data to nodes in a balanced manner. When the storage scale changes, the process of migrating data utilizing this method is very complicated, and it takes a longer time to locate the data, and thus the adaptation would be poor. Brinkmann et al. [28] proposed two data layout algorithms called SHARE and SIEVE. While the SHARE method has poor balance results and still needs to introduce virtual nodes, the SIEVE method has better balance outcomes but uses many intervals. Hence, a large number of intervals need to be reallocated when the storage scale changes greatly. Thus, the amount of data transferred by the two methods is twice the optimal amount of data. However, the adaptability is relatively poor. Current commercial storage systems such as Lustre (distributed file system software) [7, 8, 29], Panasas (distributed file system software) [30], GPFS (distributed file system software) [31], and so on use pseudo-random methods or heuristic methods based on the available capacity of nodes to allocate data. These systems generally do not migrate data to ensure the balancing of data redistribution when the node set changes, so their adaptability is relatively poor too. On the other hand, Panasas [7] uses an active balancing mechanism to redistribute data by migrating data and modifying the matching table. These systems store the matching information between the data and the node in a table form, so when locating the data, it is necessary to query the matching table, which consumes a lot of time and space. Apart from these, the OMBOS achieves multi-tiered balancing of objects between and within replica sets based on comprehensive performance weights. Besides, it can adaptively migrate data to achieve dynamic balancing when the node changes. Therefore, the amount of adaptive data migration can reach the theoretical settings.

3. The Model

An effective data organization method needs to meet the conditions of redundancy, balancing, and adaptability of data organization. The object storage system composes of several storage nodes and a replica set to store multiple copies of files to achieve data fault tolerance, which satisfies the redundancy feature and thus focuses on the research of objects between replica sets and within replica sets that satisfy balancing and adaptability. This section presents the mathematical model of the tiered organization of objects as well as the definition of the balancing and adaptability of the organization method.

3.1. Definition of Object Tiered Organization

3.1.1. Object Organization between Replica Sets. When the object storage system is initialized, the node set $C_0 = \{c_1, \ldots, c_n\}$ is divided into k replica sets denoted by $R = \{C_1, \ldots, C_k\}$ according to the uniformly constructed number of replicated set nodes. Suppose that the replica set C_j is denoted by $\{c_i^j; i = 1, 2 \ldots, n_j\}$ where $1 \le j \le k$ and the summation is represented by $\sum_{j=1}^{k} n_j = 1$. A function $f: X \longrightarrow R$ maps the collection of the small file objects X to the collection of the replica set. The weight of the replica set $C_j \in R$ is denoted by W_j, and the weights of the k replica sets are denoted by $\{W_1, \ldots, W_k\}$ and $\sum_{i=1}^{k} W_i = 1$, respectively. The data layout algorithm D can be represented as a binary function denoted by $D(x, f)$, where $x \in X$, f represents a matching function, and $D(x, f) = f(x)$. Data x are placed in the replica set identified by a $D(x, f)$ value.

3.1.2. The Distribution of I/O Loading in the Replica Set. The replica set C_j consists of $\{c_i^j; i = 1, 2, \ldots, n_j\}$ nodes, and the weights and summation of n_j nodes are denoted by $\{w_1^j, \ldots, w_{n_j}^j\}$ and $\sum_{i=1}^{n_j} w_i^j = 1$, respectively. The load distribution algorithm L can be expressed as a binary function represented by $L(x, g)$, where $x \in X$ and g indicates a matching function, namely, $L(x, g) = g(x)$. The I/O loading of data x is distributed to C_j nodes in the replica set identified by $L(x, g)$ value.

3.1.3. Definition of the Characteristics of the Organization Method

Definition 1 (balance). Suppose that the replica set $R = \{C_1, \ldots, C_k\}$ of the object storage system is given, the

weight of the replica set $C_j \in R$ is denoted by W_j, the dataset is denoted by $X = \{x_1, \ldots, x_m\}$, the function $f: X \longrightarrow R$ maps the object set X to the replica set sets R for $\forall x \in X$, and the data organization algorithm A distributes the load of the data x to $C_j \in R$.The probability is denoted by $p(A(x, f) = C_j)$; for $\forall \varepsilon > 0$, if $|p(A(x, f) = C_j) - W_j| < \varepsilon$, then, the algorithm A is called balanced.

The balanced algorithm implies that the data $x \in X$ are written into the replica set. So, $C_j \in R$ would be infinitely close to the relative weight C_j, ensuring that the data are written to each replica set in a balanced manner.

Definition 2 (adaptability). Suppose that the set of replica sets $R = \{C_1, \ldots, C_k\}$ of the object storage system is given, the weight of the replica set $C_j \in R$ is W_j, the dataset is denoted by $X = \{x_1, \ldots, x_m\}$, and the function $f: X \longrightarrow R$ maps the object set to the replica set R. If the data organization algorithm A meets the following two conditions, the data organization algorithm A is called adaptive.

The current set of a replica set R becomes $R^+ = \{C_1, \ldots, C_{k+1}\}$. Let the function $f^+: X \longrightarrow R^-$ map the object set X to the current set of the replica set R^+; $\forall x \in X$, if $A(x, f_+) = f(x_+) \in R$, then $A(x, f_+) = A(x, f_0)$.

The current set of a replica set R becomes $R^- = \{C_1, \cdots, C_{j-1}, C_{j+1}, \ldots, C_k\}$, and the function $f^-: X \longrightarrow R^-$ maps the object set X to the current set of the replica set R^-. $\forall x \in X$, if $A(x, f) = f(x) \in R^-$ for $\forall x \in X$, then $A(x, f_0) = A(x, f^-)$.

When the scale of the object storage system changes, the adaptability of the organization algorithm makes the migrated data move only from the unchanged replica set to the newly added replica set or from the removed replica set to the unchanged replica set. Thus, there is no data migration between the unchanged replica set to ensure that the amount of data to be migrated is minimum. For example, given that a replica set C_{k+1} is added, when the set of a replica set R becomes $R^+ = \{C_1, \ldots, C_{k+1}\}$, then the data are only migrated from the set of the replica set R to $R^+ = \{C_1, \ldots, C_{k+1}\}$ and there is no data migration between replica sets of R. This makes the amount of data migration equal to the amount of data on the added or removed replica set without adding new rules. Hence, the system can still use the algorithm A to calculate the location of the data after the data are migrated. Therefore, the algorithm A can be adapted to changes in the scale of the object storage system.

4. The Proposed Method

4.1. The Overview of the Architecture. By aiming at achieving a balanced distribution of large amounts of small files in the object storage system, this paper proposes an object tiered organization method based on consistent hashing. An object storage system is generally composed of several replica sets. The replica set especially is the smallest unit of the logical storage of an object, and a many-to-one mapping relationship is established between the object and the replica set. Besides, the replica set is composed of several storage nodes to meet the fault tolerance of object storage, and data synchronization is adopted between storage nodes to achieve redundant storage of object data. Thus, object placement and accessing of stored objects need to go through a two-level organization of the replica set and the nodes of the object stored in the replica set.

The flow diagram of the method used for the placement of the stored object is shown in Figure 1. The spare capacity of the storage node is only used as the constraint. Then, objects are mapped to the replica set. The selection of the master node in the replica set is based on choosing the storage unit with the largest free capacity. However, the performance factor of the storage unit is not considered, and the replica set is provided by a single master node. Thus, a single balancing condition would lead to an imbalance between the performance of the storage unit and the loading of the data. Moreover, a single read-write master node leads to lower utilization of reading and writing resources in the replica set, and multi-level complex data migration jointly restricts the overall performance of the system.

To achieve a balanced distribution of object data in the performance of the system accession, this paper proposes an object multi-tiered balanced organization strategy (OMBOS) based on consistent hashing. The object placement process is shown in Figure 2. The method of the tiered organization first needs to measure the weight indicators that can fully reflect the performance of the nodes in the system and then finish the organization of small file objects at the replica set level. Then, the I/O loading distribution of the storage nodes in the replica set utilizes the consistent hash method. When the system scale changes, the amount of migration data is reduced to lower the system performance jitter.

The OMBOS consists of four parts.

(1) The comprehensive performance evaluation modeling and weight measurement: comprehensive performance evaluation modeling for each storage device in the object storage system is established, and the weight indicators of the object storage nodes and their constituent replica sets are measured.

(2) Balanced distribution algorithm among object replica sets: according to the weight of the replica set, the layout and organization of objects at the replica set level are completed, and small file objects are mapped to specific replica sets.

(3) I/O load balancing distribution algorithm in the replica set: according to the weight of the nodes in the replica set, the I/O loading distribution of the object at the node level in the replica set is completed, and the object I/O loading borne by the replica set is mapped to the specific the object storage node.

(4) Adaptive dynamic balancing strategy: when the scale of the object storage system changes, the system will automatically migrate data within the system to minimize the amount of data migration, reduce performance jitter, and meet the scalability of the system.

4.2. The Comprehensive Performance Evaluation Modeling and Weight Measurement. To achieve a balanced organization of data objects in the system, comprehensive and

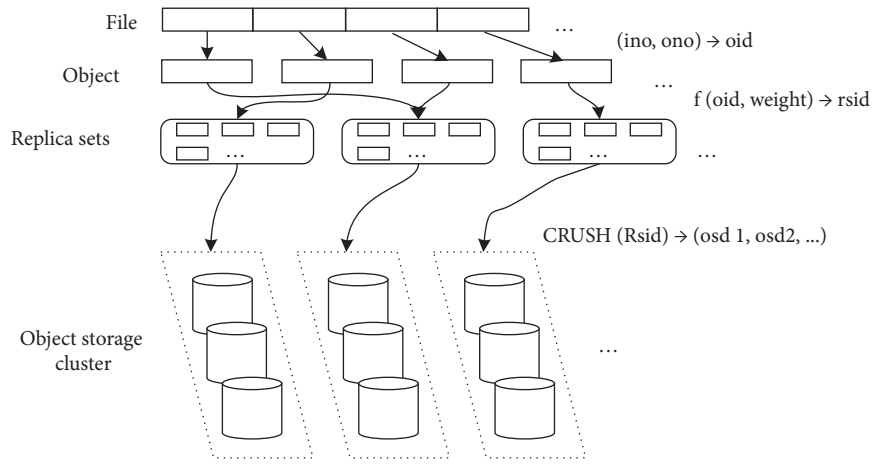

Figure 1: The object storage process of small files.

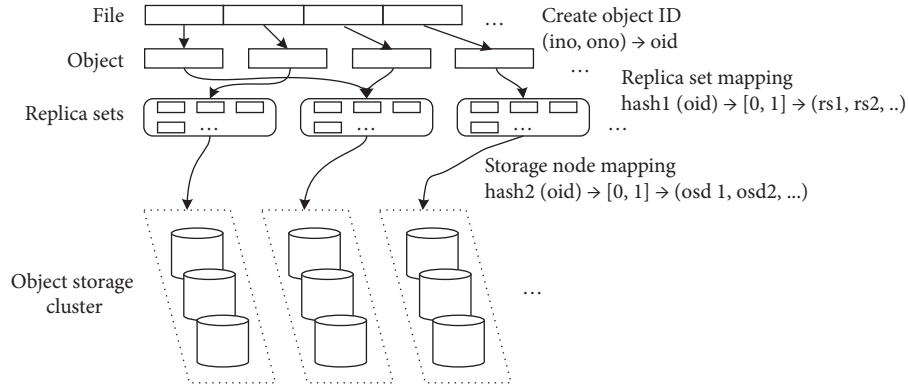

Figure 2: The object storage process of the tiered organization method.

fine-grained performance indicators are needed to be picked to comprehensively evaluate the performance of storage nodes and replica sets. This section begins with the cost of resource characteristics of storage nodes in small file storage scenarios [32, 33], then establishes a comprehensive performance evaluation modeling for storage devices in the object storage system based on the fuzzy analytic hierarchy process, and uses the least square method to complete the process. According to the comprehensive performance evaluation modeling, the weight measurement of the replica set and the nodes in the replica set is completed.

4.2.1. Definition of Comprehensive Performance Evaluation Modeling. To fully reflect the performance of the node, the definition of the node performance index is synthesized by the hardware status of the server itself such as CPU (C), memory (M), hard disk (D), and available bandwidth (B) to comprehensively evaluate the performance of physical nodes, which is denoted by

$$P_i = \alpha C_i + \beta M_i + \gamma D_i + \eta B_i. \tag{2}$$

The node performance index is a weighted index of the hardware status containing CPU, memory, disk, bandwidth, etc. Each parameter is represented by a variable to describe its relative weight in the index. The method to determine the coefficients is given in detail below.

4.2.2. Determination of the Coefficients of the Comprehensive Performance Evaluation Modeling. To objectively and reasonably measure the parameter coefficients in the comprehensive performance evaluation modeling, the fuzzy analytic hierarchy process [26] is introduced, which combines the coefficient tilt based on the function of the node and the fuzzy matrix operation to accurately calculate the coefficient of each variable, so that the performance is comprehensively evaluated. It has the following steps.

(1) Construction of a fuzzy matrix.

Let $P = \{M_1, M_2, M_3, \ldots, M_m\}$ and $Q = \{N_1, N_2, N_3, \ldots, N_n\}$ be bounded sets. R is a binary relationship from P to Q; then, $R = (r_{ij})_{m\times n}$= is the relationship matrix denoted by

$$R = \begin{bmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & \ddots & \vdots \\ r_{m1} & \cdots & r_{mn} \end{bmatrix}. \tag{3}$$

In classical mathematics, the elements of the relationship matrix can only take 1 or 0 to indicate

whether the set elements satisfy this relationship. When two elements conform to this relationship, number 1 is used, and when the elements do not conform to this relationship, it takes 0. But in the fuzzy relationship, this choice of 1 or 0 does not fully reflect the actual subordination relationship. The fuzzy analytic hierarchy process uses the concept of membership degree to quantify the relative importance of elements. The commonly used membership degrees are shown in Table 1.

In the circumstance of a small file object storage, the I/O characteristics of the object storage node are analyzed. The small file of the I/O has the characteristics of large access volume and a relatively small amount of data, so the memory of the object storage node is the I/O processing. Thus, the disk space is a prerequisite to providing services. The amount of small file I/O data is also relatively small, and the current Gigabit network card can meet the demand, and the amount of CPU computing in the multicopy mode is small too. Therefore, it can be concluded that the relative importance of each element of the object storage node is represented by disk > memory > network > CPU, and then the fuzzy relationship matrix for the parameter coefficients of the comprehensive performance evaluation model is constructed by

$$R = \{C, M, D, B\} \times \{C, M, D, B\} = \begin{bmatrix} 0.5 & 0.3 & 0.1 & 0.4 \\ 0.7 & 0.5 & 0.4 & 0.6 \\ 0.9 & 0.6 & 0.5 & 0.8 \\ 0.6 & 0.4 & 0.2 & 0.5 \end{bmatrix}. \tag{4}$$

(2) The least square method to determine the coefficients.

Although the membership degree of the element is determined by the application characteristics, there are still subjective judgment factors in the construction of the fuzzy matrix, which often cannot truly reflect the exact membership degrees. In this fuzzy setting, the mapping between membership degree and weight also has a certain deviation. To minimize this deviation, a method to resolve this issue is needed. The least square method [34] is one of the common methods used to solve it. The optimization problem is expressed by

$$\begin{cases} \min f = \sum_{i=1}^{n} \sum_{j=1}^{n} \left[r_{ij} - 0.5 - \alpha\left(w_i - w_j\right) \right]^2 \\ \text{s.t} \sum_{k=1}^{n} w_k = 1 \end{cases}. \tag{5}$$

The structure of the problem is transformed into a problem with no linear constraints utilizing the Lagrangian multiplier method and the partial reciprocal method so that the value of each partial derivative is equated to 0. The linear constraint is denoted by $\sum_{k=1}^{n} w_k = 1$. Then, $n+1$ equations containing $n+1$ variables are obtained, and the value of each coefficient w_i can be obtained by solving the equations. The calculated results are represented by

$$w_i = \frac{1}{n} - \frac{1}{2\alpha} + \frac{1}{n\alpha} \sum_{k=1}^{n} r_{ik}, \quad i = 1, 2, \ldots, n. \tag{6}$$

So, when $n = 4$,

$$w_i = \frac{1}{4} - \frac{1}{2\alpha} + \frac{1}{4\alpha} \sum_{k=1}^{4} r_{ik}, \quad i = 1, 2, \ldots, 4. \tag{7}$$

The linear coefficient α satisfies $\alpha \geq n - 1/2$. Therefore, the linear sequence of the corresponding coefficients is obtained through the fuzzy judgment matrix. By the examination of equation (7), we observe that the order of the linear arrangement of the coefficients is only related to the sum of the membership values of the row and not related to the parameter α. The choice of its value often depends on the experience of the decision maker. The smaller the value is, the more distinct the level of the coefficient would be. Therefore, $\alpha = n - 1/2$ is selected in this design, that is, 3/2 is the parameter coefficient.

$$w_i = \frac{1}{6} \sum_{k=1}^{4} r_{ik} - \frac{1}{12}, \quad i = 1, 2, 3, 4. \tag{8}$$

Thus, calculations give $\alpha = 0.133, \beta = 0.283$, $\gamma = 0.383$, and $\eta = 0.2$ when (2) is employed. Then, (2) is represented by

$$P_i = 0.133C_i + 0.283M_i + 0.383D_i + 0.2B_i. \tag{9}$$

(3) Measurement method of performance weight.

The node performance index is calculated according to the comprehensive performance evaluation modeling, and the performance index of the replica set can be further defined as the sum of the performance indexes of all nodes in the replica set, that is, $\sum_{i=1}^{n_j} P_i$.

According to the performance index of the replica set and the performance index of the node, the weight of the replica set and the relative weight of the nodes in the replica set can be obtained.

Definition 3 (the weight of the replica set). The proportion of the performance index of the replica set in the total performance index of all nodes for the system is represented by

$$W_j = \frac{\sum_{i=1}^{n_j} P_i}{\sum_{j=1}^{k} \sum_{i=1}^{n_j} P_i}. \tag{10}$$

Definition 4 (the relative weight of nodes in the replica set). The proportion of the node performance index in the

Table 1: The commonly used membership scale.

Membership values	Definition	Illustration
0.5	Equally important	The two elements are equally important
0.6	Slightly important	Comparing two elements, one is slightly more important than the other
0.7	Important	Comparing two elements, one is more important than the other
0.8	Much more important	Comparing two elements, one is much more important than the other
0.9	Extremely important	Comparing two elements, one is more important than the other
0.1 0.2 0.3 0.4	Inverse comparison	Inverse comparison of relative importance between elements

sum of the node performance index of the replica set is represented by

$$W_i^j = \frac{P_i}{\sum_{i=1}^{n_j} P_i}. \tag{11}$$

The measurement of the weight index involves the parameters of the hardware status such as CPU, memory, disk, and network. This article deals with the performance monitoring interval and calls system tools to realize the collection of data monitoring. To avoid a sudden increase or decrease in resource utilization, the weight is calculated based on the average utilization rate. The parameters of the hardware are regularly collected to complete the measurement of the weight index.

4.2.3. Algorithm for a Balanced Distribution of Objects among Replica Sets. To satisfy the balancing of the object organization of the storage system [18, 19, 33], this section proposes an object placement between replica sets algorithm (BROP). According to the weight index of the replica set proposed in the previous section, the small files are mapped to the specific replica set. Then, the objects are placed in the specified replica set, so the organization of the data layout of the object at the replica set level is conducted. Thus, the mass small files are saved in the object storage system with the balanced distribution.

According to the definition of the weight of the replica set and the definition of the small file object organization, the weight of the replica set C_j is denoted by W_j where $W_j = \sum_{i=1}^{n_j} P_i / \sum_{j=1}^{k} \sum_{i=1}^{n_j} P_i$. The problem is transformed to design the distribution of the solution objects based on the OBPBRS algorithm when the set of replica sets is utilized.

To achieve a balanced distribution of objects in the replica set $R = \{C_1, \ldots, C_k\}$, the [0, 1] interval is divided into multiple subintervals according to the weight of each replica set in R, and the subinterval at which the replica set C_j is located is denoted by $I_j = [\sum_{i=1}^{j-1} W_i, \sum_{i=1}^{j} W_i)$. Then, the hash function h_1: $X \longrightarrow [0, 1]$ is constructed to map the small file object $x \in X_0$ to the [0, 1] interval. As shown in Figure 3, the mapping from the object to the replica set is conducted according to the consistent hash rule. If $h_1(x) \in I_j$ is given, then x is assigned to the replica set C_j corresponding to the subinterval I_j.

Suppose that a function f_r: $X_0 \longrightarrow R$ represents the mapping of the object collection X_0 to the replica set R; then, OBPBRS can be represented as the function $BROP(x, f_r)$- where, $BROP(x, f_r) = f_r(x)$. The pseudocode of fr is given in Pseudocode 1.

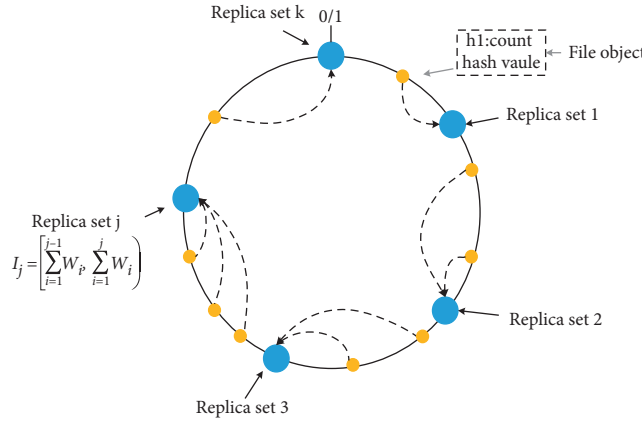

Figure 3: The diagram of the BROP algorithm.

According to probability theory, the amount of data falling in each interval is proportional to the interval size that is positively correlated with the performance of the replica set. Therefore, the loading of small file object storage can be evenly distributed in the replica set $R = \{C_1, \ldots, C_k\}$ according to the performance.

4.2.4. I/O Balanced Distribution Algorithm in the Replica Set. The OBPBRS algorithm evenly distributes data objects to each replica set. A balancing mechanism is also needed within the replica set to evenly distribute the I/O requests allocated to each replica set to each node at the node level in the replica set.

I/O load balancing in replica set algorithm (IRIB) is also based on a consistent hash mechanism, which distributes I/O loading among nodes according to the weight of nodes in the replica set to achieve node-level I/O load balancing layout.

Suppose that the replica set C_j is composed of $\{c_i^j; i = 1, 2, \ldots, n_j\}$ nodes, and the weight of C_j is denoted by W_j; then, the weight of the ith node in C_j is the replica set denoted by $w_i^j = P_i / \sum_{i=1}^{n_j} P_i$, and the total weight of the n_j nodes is equal to $\sum_{i=1}^{n_j} w_i^j = 1$.The problem is transformed into designing the BPRS algorithm to solve the I/O loading distribution among the nodes in the replica set C_j.

To achieve a balanced distribution of I/O loading in the replica set $C_j = \{c_i^j, i = 1, 2, \ldots, n_j\}$, the [0, 1] interval is divided into multiple subintervals according to the weight of each node in the replica set in C_j, and the subinterval at which c_i^j node is located is denoted by $S_i^j = [\sum_{i=1}^{j-1} w_i^j, \sum_{i=1}^{j} w_i^j)$. Since the dataset falling into the replica set C_j is denoted by X_j according to the BROP

```
Input: x, R = {C_1, ..., C_k}, {X_1, ..., X_k}
Process
for (j = 1, j ≤ k, j++)
  //Determine the subinterval I_j to which x belongs.
  //If there are multiple subintervals related with I_j,
  //then traverse all of these subintervals.
  If (h_1(x) ∈ I_j)
  then add x to X_j related with I_j;
  End if
End for
Output: replica set C_j related with I_j
```

Pseudocode 1: Pseudocode of BROP.

algorithm, the data objects X_j are in a local aggregation state due to the hash function h_1: $X \longrightarrow [0, 1]$ that is gathered in the interval I_j on [0, 1), instead of [0, 1]. To ensure the balance of I/O loading distribution, the hash function h_2: $X_j \longrightarrow [0, 1]$ is constructed to recalculate the hash value of the data objects X_j, so that the data objects X_j are evenly distributed in the [0, 1] interval. As shown in Figure 4, the mapping from the object to the replica node is completed according to the consistent hash rule. If $h_2(x) \in S_i^j$ is defined, the I/O loading of the object x is allocated to the node corresponding to the subinterval in the replica set C_j.

Suppose that a function g_c: $x \longrightarrow C_j$ representing the I/O loading of the small file object x is mapped to the nodes in the replica set C_j; then, the BPRS can be represented by a function $IRIB(x, g_c)$, where $IRIB(x, g_c) = g_c(x)$ and g_c is expressed in Pseudocode 2.

Similarly, the I/O load falling into each interval is proportional to the interval size that is positively correlated with the node performance in the replica set. Therefore, the I/O load of small file objects can be evenly distributed within the replica set $C_j = \{c_i^j; i = 1, 2, \ldots, n_j\}$ according to performance.

4.3. The Strategy of Adaptive Dynamic Equalization. The object storage system has elastic expansion characteristics, and data migration can be triggered by adding or removing nodes. The object organization method needs to adaptively migrate objects according to scale changes to reduce the system performance jitter caused by data migration. So, the amount should be as small as possible. For the scenario of flexible changes in the system scale, the object storage system organizes small file objects in units of replica sets. The addition and removal of individual nodes are not generally conducted, so we will not consider them here. But we only discuss the addition and removal of replica sets as the unit.

The adaptive dynamic balancing strategy (ADBS) mainly includes the following two state transitions [30, 35, 36].

4.3.1. Adding Data Migration to a Replica Set. In the initial situation, the node set in the system is divided into k replica sets, and the [0, 1] interval is divided into k subintervals. By adding a new replica set C_{k+1}, the weight of all replica sets in the system has changed due to the introduction of the replica set, so the interval is reallocated for the replica set according to the new weight. The data migration process of adding a replica set is shown in Figure 5. Assume that the net weight of the replica set C_j is denoted by W_j', $1 \le j \le k$. To meet the balanced object distribution, the replica set C_j needs to migrate $(W_j - W_j')m$ objects to a replica set C_{k+1}, in which m represents the total number of objects. The interval I_j of the replica set C_j is divided into $[\sum_{i=1}^{j-1} W_i, \sum_{i=1}^{j-1} W_i + W_j')$ and $[\sum_{i=1}^{j-1} W_i + W_j', \sum_{i=1}^{j} W_i)$, and the interval is mapped to the replica set C_{k+1}; then, the subinterval I_{k+1} corresponding to the replica set C_{k+1} is represented by $\cup_{j=1}^{k} [\sum_{i=1}^{j-1} W_i + W_j', \sum_{i=1}^{j} W_i)$, and the objects falling in these intervals are migrated to the replica set C_{k+1}.

When adding new replica sets one at a time, the intervals of the existing replica sets are divided according to the weight. Then, the redundant intervals of each replica set are allocated to the new replica set. Hence, the objects falling in these intervals are migrated to the new replica set. Thus, the objects in the old replica set are only migrated to the new replica set, and there remain no objects regarding migration between the old replica sets. Therefore, the amount of migrated data is equal to the amount of data that should be obtained from the new replica set.

4.3.2. Removing the Data Migration for the Replica Set. Similar to adding a replica set, the node set in the system is divided into k replica sets, and the [0, 1] interval is divided into k subintervals. A replica set C_g is removed, so the weights of all replica sets in the system have changed. Hence, the interval is reallocated for the replica set according to the new weight. To meet the balanced object distribution, the replica set C_g needs to migrate $w_g \cdot m$ objects to a replica set $\{C_1, \ldots, C_{g-1}, C_{g+1}, \ldots C_k\}$, where m represents the total number of objects. Assume that the new weight of replica set C_j is denoted by w_j', where $1 \le j \le k$. The interval I_j of the replica set C_j is expanded into $[\sum_{i=1}^{j-1} w_i, \sum_{i=1}^{j} w_i)$ and $[\sum_{i=1}^{j} w_i, \sum_{i=1}^{j-1} w_i + w_j')$. Hence, the $[\sum_{i=1}^{j} w_i, \sum_{i=1}^{j-1} w_i + w_j')$ interval is mapped to the replica set C_g, and the replica set C_g selects $(w_j' - w_j)m$ objects to migrate to the replica set C_j. In this way, the objects are distributed evenly on the $g - 1$ replica sets.

When sequentially removing replica sets, the interval of the existing replica set is divided according to the weight, and the small file objects of the removed replica set are allocated to the extended interval part of other replica sets. Hence, the objects in the removed replica set are migrated to other replica sets, and there is no object migration between other replica sets except for the removed replica set so the amount of migrated data is equal to the removed replica set.

5. The Analysis of Both Balancing and Adaptability

5.1. Balancing Analysis

5.1.1. Analysis of Distribution Balance among Replicas. Suppose that the function f: $X \longrightarrow R$ represents the mapping of the object set X to the replica set $R = \{C_1, \ldots, C_k\}$ for the object placement mechanism

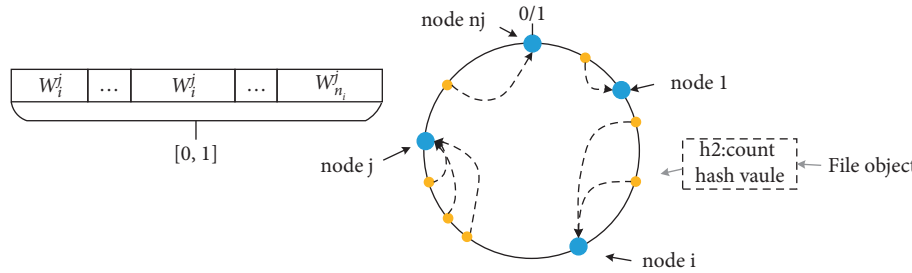

Figure 4: The schematic diagram of the IRIB algorithm.

Input: $x \in X_j$, $C_j = \{c_i^j; i = 1, 2, \ldots, n_j\}$
Process:
for $(i = 1; i \leq n_j; i++)$
  //Determine the subinterval S_i^j to which x belongs.
  //If there are multiple subintervals related with S_i^j,
  //then traverse all of these subintervals.
  If $h_2(x) \in S_i^j$
  then distribute I/O of x to c_i^j related with S_i^j;
  End if
End for
Output: replica node c_i^j related with S_i^j

Pseudocode 2: Pseudocode of IRIB algorithm.

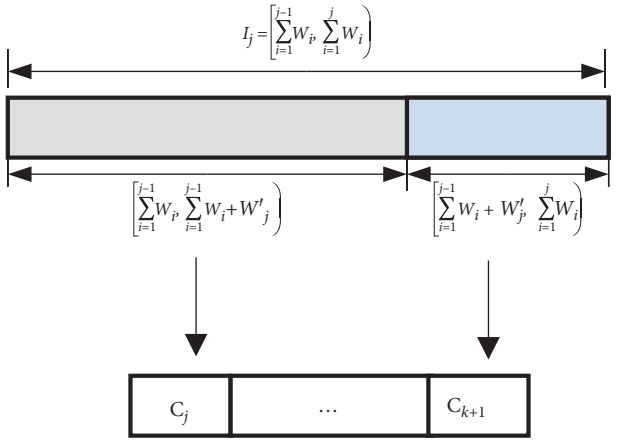

Figure 5: Addition of a replica set for the data migration process.

between replica sets; then, the object organization algorithm (BROP) between replica sets can be expressed as a function $BROP(x, f)$ where $BROP(x, f) = f(x)$. The interval allocated to the replica set $C_j \in R$ is denoted by I_j. Suppose that the length of the interval I_j is denoted by $|I_j|$, and the relative weight of the replica set C_j is denoted by W_j. The BROP divides the interval according to the weight of the replica set. For $x \in \forall X$, x falls into the interval I_j with the length of $|I_j|$. The probability that x mapped to the replica set $|I_j|$ is represented by $P(BROP(x, f)) = f(x) = C_i = I_j = W_j$, where $|P(BROP(x, f) = C_j) - W_j| = 0$. So, $\forall \varepsilon > 0$, $|P(BROP(x, f) = C_j) - W_j| < \varepsilon$. According to Definition 1, the BROP is called balanced, and the elements of the object X are evenly distributed in R of the replica sets.

5.1.2. Analysis of the I/O Balance in the Replica Set. Suppose that a function g_c: $x \longrightarrow C_j$ represents the small file object x that is mapped to the node set $C_j = \{c_i^j; i = 1, 2, \ldots, n_j\}$ of the replica set in the I/O loading distribution mechanism; then, the I/O load balancing algorithm in the replica set can be expressed as the function $IRIB(x, g)$, where $IRIB(x, g) = g(x)$. The interval allocated to the node $c_i^j \in C_j$ is denoted by S_i^j. Suppose that the length of the interval S_i^j is denoted by $|S_i^j|$, and the relative weight of the node C_i^j in the replica set C_j is denoted by w_i^j. The IRIB allocates intervals according to the weight of nodes in the replica set, so $|S_i^j| = w_i^j$. For $x \in \forall X$, x falls into the interval S_i^j with the length of $|S_i^j|$. Then I/O load distribution node falling into the replica set C_j is denoted by $|S_i^j|$, so $P(IRIB(x, g) = g(x) = c_i^j) = |S_i^j| = w_i^j$ and $|P(IRIB(x, g) = c_i^j) - w_i^j| = 0$. Therefore, $\forall \varepsilon > 0$, $|P(IRIB(x, g) = c_i^j) - w_i^j| < \varepsilon$. According to Definition 1, the IRIB algorithm is balanced, and the I/O loading $x \in X_j$ is evenly distributed among the nodes of the replica set C_j.

5.2. Adaptive Analysis

5.2.1. Adding the Adaptability of the Replica Set. When adding a new replica set, assume that the set $R = \{C_1, \cdots, C_k\}$ of the replica set is adjusted to $R^+ = \{C_1, \ldots, C_k, C_{k+1}\}$. For each replica set $C_j \in R$, the weight of the replica set denoted by C_{k+1} is recalculated according to the weight of C_j. So, the weight of the replica set C_j will become smaller, and the interval of the replica set C_j will be divided according to the weight difference. Afterward, the extra interval will be assigned to C_{j-1}, and the objects falling in the redundant interval will be transferred to C_{j+1}. After the objects are migrated, the function f^+: $X \longrightarrow R^+$ represents the mapping of the object set X to the new replica set R^+. While objects are moved to C_{j+1}, no objects are moved to C_j. Therefore, if an object is placed on C_j after the migration, the object is also placed on C_j before the migration. That is, if $BROP(x, f^+) = C_j$ and $C_j \subset R$, then $BROP(x, f) = C_j = BROP(x, f^+)$. Finally, the data migration strategy of the object organization method between replica sets becomes adaptive when adding replica sets.

5.2.2. Removing the Adaptability of the Replica Set. When removing an existing arbitrary replica set C_j, the set of replica sets $R = \{C_1, \ldots, C_k\}$ is set to $R^- = \{C_1, \ldots, C_{j-1}, C_{j+1}, \ldots, C_k\}$. For each replica set $C_i \in R\,(i \neq j)$, the weight of the replica set C_i is recalculated according to the weight of C_j. The weight of the replica set C_i will become larger, and

the sum of the weight changes of other replica sets is equal to the weight of C_j, which is equal to $\sum_{j\neq i}^{C_i} \Delta W_i = W_j$. According to the weight difference, the interval of the replica set C_j is divided, so the divided subintervals have sequentially corresponded to the replica set $C_i \in R(i \neq j)$. Thus, the objects falling in the subintervals are sequentially migrated to the unremoved $C_j \in R(j \neq i)$. After the objects are migrated, the function $f\colon X \longrightarrow R$ represents the mapping from the object set X to the new node set R. While C_j only moved objects out to $C_i \in R(i \neq j)$, no objects moved between $C_i \in R(i \neq j)$ that were not removed. If an object is placed on $C_i \in R(i \neq j)$ before the migration, the object will also be placed on $C_i \in R(i \neq j)$ after the migration. That is, if $BROP(x, f) = C_i$, and $i \neq j$, then $BROP(x, f^-) = C_i = BROP(x, f)$. Therefore, the data migration strategy of the object organization method between replica sets is adaptive when the replica set is removed.

6. Evaluation

6.1. Experimental Environment. The experiment uses three configuration servers *A*, *B*, and *C* to build the object storage system and test environment. The hardware configuration of the experimental environment is shown in Table 2. The object storage system uses 9 A nodes, 6 B nodes, and 3 C nodes to construct an object storage cluster. The replica set is composed of servers with the same configuration using 3 replicas, and 18 nodes form a total of 6 replica sets (A1–A3, B1-B2, C1). Hence, 3 physical nodes are utilized to mix the deployment of monitor nodes and test nodes, and all nodes are interconnected through Huawei S-5700 Gigabit network switches.

The experimental environment topology is shown in Figure 6.

The configuration of the experimental environment software is shown in Table 3. All physical nodes deploy CentOS 7.2 operating system and configure Linux 3.10.0 kernel. The object storage system software uses the Ceph v0.94 release. The test tool uses COSBench, an object storage system benchmark tool developed by Intel. A 1–64 kB distribution with an average size of 16 kB and a total of 1 TB test dataset is used.

6.2. Data Distribution. The strategy of the object organization for the object storage system is conducted as the original strategy and improvement strategy. Using the COSBench tool, a test set of small files into the object storage clusters with a total size of 1 TB, an average size of 16 kB, and a randomly distributed size range of 1-64 kB can be written. And run testing was conducted or the data distribution among the replica set. The weight, initial capacity, and proportion of data for each replica set were recorded. The test results are shown in Table 4.

Figure 7 shows the data distribution under the original strategy. The horizontal axis represents the weight, capacity, and total data proportion of each replica set. Besides, the original organization strategy focuses on writing objects into the replica set with the largest capacity, resulting in data gathering in the replica set composed of nodes A and B with the same capacity, and only a very small amount of data is written to the set composed of node C. From the perspective of replica set performance, the C replica set with the best performance only bears a very small amount of data load, and the replica sets of A and B have performance differences, but carry the same load.

Figure 8 shows the data distribution under the proposed improvement organization strategy. Besides, the improved strategy completes the placement of objects according to the weight of the replica set when distributing objects. The replica sets of the three configurations of A, B, and C bear data load ratios roughly close to their weights, achieving a statistically balanced data load based on weights. Therefore, the improvement strategy meets the design requirements for the balance organization of the data.

6.3. Adaptive Data Migration Test. To test the adaptability of the data migration of the proposed small file object tiered organization method when the system scale changes, the elastic expansion of the object storage based on the system scale is simulated, and adding and removing replica set composed of 4 groups of A configuration nodes (A4–A7) are sequentially conducted to test the actual migration of data in the system and compare it with the theoretical migration.

According to the adaptive dynamic balance strategy, the theoretical migration amount is the amount of data corresponding to the weight of the changed replica set, which is calculated by multiplying the weight of the changed replica set and the total amount of experimental data (1 TB). The method for calculating the actual migration amount employs files that the system uses to monitor the file system of module Inotify in Linux kernel mode, in order to track the path of the object file stored in the file system. If a new file is generated, it can only be the object migrated from other replica sets, and the actual migration amount is completed through self-written script statistics.

6.3.1. Adding Replica Set Test in Turn. Table 5 shows the test results for the weight set of the new replica, theoretical migration amount, and actual migration amount after adding replica sets (A4–A7) in sequence.

As shown in Figure 9, the amount of data migration and theoretical migration is computed after adding replica sets in sequence. When adding replica sets A4 and A7, respectively, the experimental value of the migration data volume is slightly higher than the theoretical one. When adding replica set A5, the experimental value is very close to the theoretical one. For the replica set A6, the experimental value is slightly lower than the theoretical one. Besides, when a new replica set is added, intervals are allocated to the new replica set, and data falling in these intervals are migrated to the new replica set. Since the amount of data falling into this interval during the experiment is smaller than the theoretical amount of data, the data transferred are smaller than the theoretical value, and thus the experimental value is lower than the theoretical value. Figure 9 depicts that when adding a replica

Table 2: The hardware configuration of the experimental environment.

Type	Quantity	Role	CPU	RAM (GB)	Storage
A	12	Replica sets A1–A3 Monitor/test node	Intel Xeon E3-1270	64	2 TB HDD
B	6	Replica sets B1-B2	Intel Xeon E3-1270	16	2 TB HDD
C	3	Replica set C1	Intel Xeon E5-2620	256	500 GB SSD
Switch	1	Huawei S-5700	—	—	—

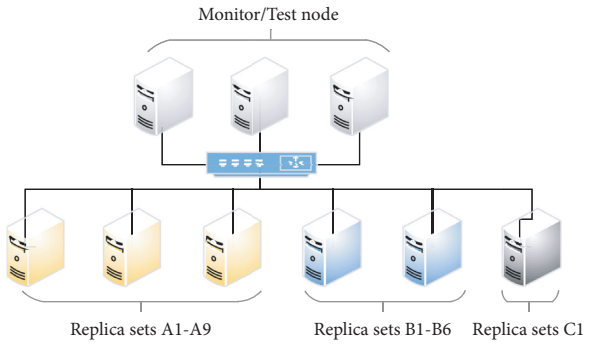

Figure 6: Experimental environment topology.

Table 3: Experimental environment for the software configuration.

Type	Configuration
OS	CentOS 7.2 Linux 3.10.0
Object storage	Ceph v0.94
Test tools	Intel COSBench
Test dataset	Simulate to generate small files of 1–64 kB, with an average size of 16 kB, and a total of 1 TB

Table 4: The test results of the data distribution.

Organizational strategy	Replica set	Replica set weight (%)	Initial capacity (TB)	Percentage of data volume
Original strategy	A1	17.1	1.84	20.0
	A2	17.9	1.84	21.2
	A3	17.1	1.84	19.0
	B1	12.0	1.84	19.1
	B2	13.6	1.84	20.0
	C1	29.9	0.46	1.63
Improvement strategy	A1	17.1	1.84	16.46
	A2	17.9	1.84	15.32
	A3	17.1	1.84	17.36
	B1	12.0	1.84	12.42
	B2	13.6	1.84	14.15
	C1	29.9	0.46	25.56

set, the experimental value of the migration data volume is very close to the theoretical one.

6.3.2. Removing the Replica Set Test in Turn. Table 6 shows the test results of the removed replica set weight, theoretical migration amount, and actual migration amount after the replica set (A4–A7) is sequentially removed.

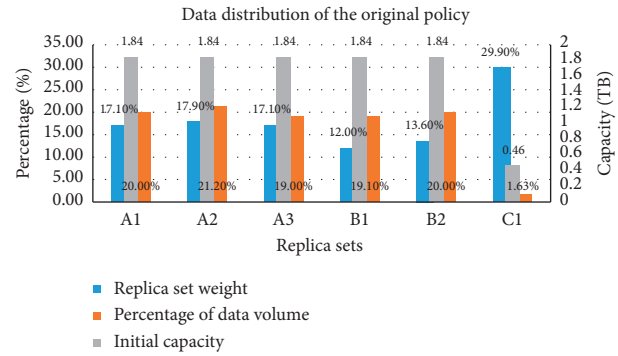

Figure 7: The data distribution of the original strategy.

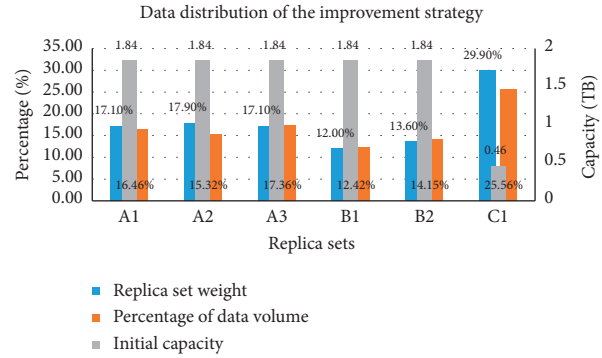

Figure 8: The optimization strategy of the data distribution.

Table 5: The data migration results of adding a replica set.

	A4	A5	A6	A7
The weight of the added replica set	15.10%	14.75%	14.36%	13.63%
Theoretical migration (GB)	142.70	151.04	147.05	139.57
Actual migration (GB)	156.77	150.12	141.31	142.75

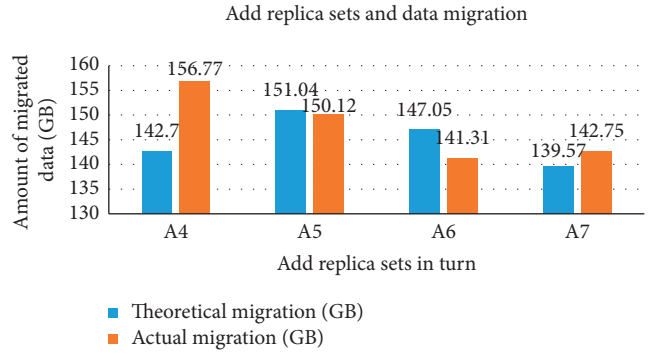

Figure 9: Adding the data migration volume of the replica set.

Table 6: The results of the data migration amount of the removed replica set.

	A4	A5	A6	A7
The weight of the removed replica set	12.25%	13.13%	14.22%	14.94%
Theoretical migration (GB)	125.44	134.45	145.61	139.57
Actual migration (GB)	127.80	135.27	141.31	154.73

As shown in Figure 10, the data migration volume and the theoretical migration volume after removing the replica set in turn are observed. While the horizontal axis represents the removed copy set, the vertical axis represents the amount of data migrated when a certain copy set is removed. Similar to the amount of migration data to add a replica set, the experimental value of the migration data amount is slightly higher than the theoretical value when the replica set A4 and replica set A7 are removed. When the replica set A6 is removed, the experimental value is slightly lower than the theoretical value. For the replica set A5 that is removed, the experimental value is very close to the theoretical one. Besides, when the replica set is removed, the interval of the replica set is expanded, and the data of the removed replica set are migrated to these intervals in turn. Similar to the experiment of adding a replica set, because the amount of data falling into these intervals during the experiment is less than the theoretical amount of data, the data transferred are less than the theoretical one, so the experimental value is lower than the theoretical value.

6.3.3. The Test for Data Redistribution. To test the redistribution of the data after the adaptive migration of the data, the case of adding replica set A7 is used as an example, and the data distribution after adding the replica set is shown in Figure 11. The proportion of data in each copied set is close to its weight, indicating that the tiered organization of small file objects can meet the balanced object organization when the system scale is changed.

Therefore, the data migration strategy of the small file object tiered organization method has a better adaptability and can ensure the balance of the data distribution after migration.

6.4. The Performance of Reading and Writing. To test the object's tiered organization method to achieve the improvement of the performance accession by the balanced layout of the objects, the performance accession was tested under the two organization strategies. The test scenario selects the common random access request for small file accession and uses the same dataset as the data distribution test. The test tool called COSBench picks 8, 16, 32, 64, and 128 threads with a total of 5 I/O modes to simulate different concurrent threads. Writing and reading randomly is assumed. In the experiment, the warm-up time of COSBench is set to 100 s, and the running time is 600 s. The reading and writing performance test results are shown in Table 7.

As shown in Figure 12, the average response delay curve implies that the random writing performance of the original

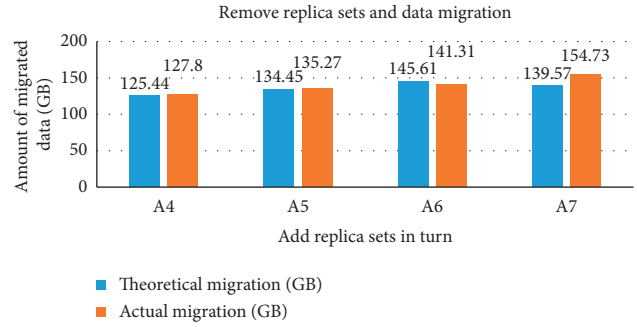

Figure 10: The amount of data migration to remove the replica set.

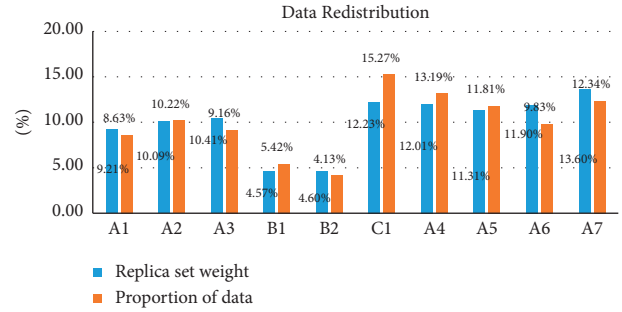

Figure 11: The redistribution of the data.

Table 7: The test results of both reading and writing performances.

Scenes	Layout strategy	Threads	Average response delay (ms)
Random writing	Original strategy	8	11.56
		16	19.32
		32	42.08
		64	73.71
		128	85.2
		256	107.11
	Improved strategy	8	23.87
		16	33.03
		32	57.49
		64	43.25
		128	41.95
		256	47.12
Random reading	Original strategy	8	4.32
		16	7.09
		32	15.21
		64	22.47
		128	26.38
		256	38.2
	Improved strategy	8	5.79
		16	8.23
		32	14.9
		64	12.7
		128	10.3
		256	14.22

strategy is better than the improved strategy when the number of the concurrent threads is small because the temporal locality of writing operations under the original strategy is transformed into the spatial locality of I/O within individual nodes. The local cache of the single-machine file

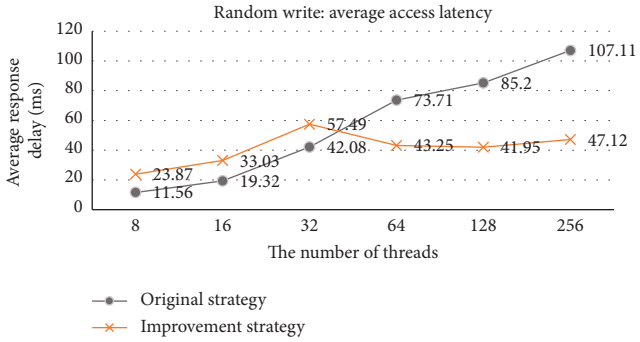

Figure 12: The test results of the random writing performance.

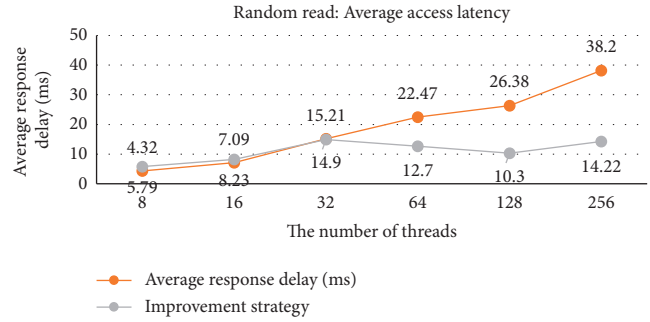

Figure 13: The test results of the random reading performance.

system reduces the access latency of writing while the improved strategy performs write load balancing globally and small-scale concurrency. The network overhead of distributing the load is dominated, and the overall performance cannot be fully utilized. With the increase in the number of the concurrencies, the single-machine I/O bottleneck of the original strategy has caused the writing performance to decrease and the delay increases. Hence, the improved strategy can balance the writing load according to the performance of the replica set, giving full play to the advantages of the distributed system in processing concurrent requests. Therefore, the writing performance of the later improved strategy is better than the original strategy, and as the number of concurrency increases, the advantage becomes more obvious. The improved random writing performance is doubled when compared to the original strategy.

6.4.1. Analysis of Random Reading Performance Results. As shown in Figure 13, comparing the test result curves in Figures 12 and 13, it can be found that the average response delay of the reading operation is approximately 3 times higher than that of the writing operation. The replica set takes the form of 3 replicas. By observing the average response delay curve in Figure 13, there is the same situation as the random writing performance test is conducted. So, the data location path of the reading operation and the writing operation are the same. Thus, the original strategy is found better than the improved strategy when the number of concurrencies is small. As the number of concurrencies gradually increases, the local hot issues of the original strategy appear, and the improved strategy just takes advantage of the parallelism of the distributed loading, finally making the reading performance better than the original strategy. Also, as the number of concurrency increases, the advantage becomes more and more obvious. When compared with the original strategy, the random reading performance excels.

Therefore, the tiered organization of small file objects can achieve a balanced layout based on weights. Moreover, the balanced layout of the system achieves a balanced layout of reading and writing loads in the system, which is well adapted to the large-scale random reading and writing access scenarios that are common in small file applications. Therefore, it gives full play to the advantages of parallelism of the object storage system and effectively improves the randomness of small files in the object storage system.

7. Conclusion and Future Work

By aiming at resolving the problem of the object layout in the small file object storage, this article first analyzes the organization of the small file objects and the definition of the problem. Comprehensive performance evaluation modeling is established based on the fuzzy analytic hierarchy process, which serves as the theoretical basis for the organization of the small file objects based on performance weights. Hence, a data organization method based on consistent hashing and an adaptive dynamical balance strategy is proposed, and the characteristics and optimization effects of the proposed method are verified through both theoretical analysis and experimental tests.

Tables and figures are used to present our results. Tables 1–3 present the membership degrees in the fuzzy analytical hierarchical method, hardware, and software configurations. While Table 4 presents the test results of data distribution, Tables 5 and 6 summarize data migration results when adding and removing replica nodes, respectively. Finally, Table 7 presents the performance results of both writing and reading. On the other hand, Figures 1 and 2 present a storage system for small files and a storage system based on the tiered organization. Figures 3 and 4 present the two algorithms. While Figure 5 summarizes the results of the data migration when a replica is added, Figures 6–8 present experimental environments, the data distribution for the optimal strategy, and the optimization strategy for the data distribution. While Figures 9 and 10 deal with either addition or removal of data migration volume to a replica set, Figures 11–13 present the results of the redistribution of the data and performances of reading and writing.

Therefore, the writing performance of the later improved strategy is better than the original strategy, and as the number of concurrency increases, the advantage becomes more obvious. The improved random writing performance is doubled when compared to the original strategy. Besides, as the number of concurrencies gradually increases, the local hot issues of the original strategy appear, and the improved strategy just takes advantage of the parallelism of the distributed loading, finally making the reading performance better than the original strategy. Also, as the number of the

concurrency increases, the advantage becomes more and more obvious. When compared with the original strategy, the random reading performance excels.

Although the tiered replica placement strategy has advantages over the performance enhancement, further studies are still needed. Besides, future research is planned to investigate the role of the parameters such as reliability, resource utilization, and security enhancement.

Data Availability

The numerical dataset used to support the findings of this study is available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This study was supported by the Engineering Planning Project of Communication University of China (no. 313XNG1527).

References

[1] Y. Li, M. Tian, Y. Wang, Q. Zhang, D. K. Saxena, and L. Jiao, "A new replica placement strategy based on multi-objective optimisation for HDFS," *International Journal of Bio-Inspired Computation*, vol. 16, no. 1, pp. 13–22, 2020.

[2] M. Mesnier, G. R. Ganger, and E. Riedel, "Storage area networking - object-based storage," *IEEE Communications Magazine*, vol. 41, no. 8, pp. 84–90, 2003.

[3] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 8, pp. 2377–2393, 2016.

[4] S. Ma, H. Chen, Y. Shen, H. Lu, B. Wei, and P. He, "Providing hybrid block storage for virtual machines using object-based storage," in *Proceedings of the 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, Hsinchu, Taiwan, December 2014.

[5] J. E, Y. Cui, Z. Li, M. Ruan, and E. Zhai, "Hycloud: tweaking hybrid cloud storage services for cost-efficient filesystem hosting," *IEEE/ACM Transactions on Networking*, vol. 28, no. 6, pp. 2629–2642, 2020.

[6] A W S, "Amazon Simple Storage Service (Amazon S3)," https://aws.amazon.com/cn/s3/.

[7] R. S. Studham and R. Subramaniyan, "Lustre: a future standard for parallel file systems," in *Proceedings of the Invited presentation at International Supercomputer Conference*, Heidelberg, Germany, 2005.

[8] W. Yu, R. Noronha, S. Liang, and D. K. Panda, "Benefits of high speed interconnects to cluster file systems: a case study with Lustre," in *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium*, IEEE, Rhodes, Greece, April 2006.

[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 *Proceedings of the 7th symposium on Operating systems design and implementation*, Seattle Washington, D.C, USA, November 2006.

[10] Sheepdog, "Sheepdog Project," https://sheepdog.github.io/sheepdog/.

[11] Openstack, "OpenStack Object Storage(Swift)," https://docs.openstack.org/swift/latest/.

[12] W. Tao, Y. Zhai, and J. Tchaye-Kondi, "LHF: a new archive-based approach to accelerate massive small file access performance in HDFS," in *Proceedings of the 2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*, IEEE, Newark, CA, USA, April 2019.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: controlled, scalable, decentralized placement of replicated data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, IEEE, Tampa, FL, USA, November 2006.

[14] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant, and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM symposium on Cloud computing*, Indianapolis IN USA, June 2010.

[15] Y. Zhai, J. Tchaye-Kondi, K.-J. Lin et al., "Hadoop Perfect File: a fast and memory-efficient metadata access archive file to face small files problem in HDFS," *Journal of Parallel and Distributed Computing*, vol. 156, pp. 119–130, 2021.

[16] J. Zhou, D. Dai, Y. Mao, X. Chen, Y. Zhuang, and Y. Chen, "I/O characteristics discovery in cloud storage systems," in *Proceedings of the 2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, IEEE, Francisco, CA, USA, July 2018.

[17] S. He, X.-H. Sun, Y. Wang, A. Kougkas, and A. Haider, "A heterogeneity-aware region-level data layout for hybrid parallel file systems," in *Proceedings of the 2015 44th International Conference on Parallel Processing*, IEEE, Beijing, China, September 2015.

[18] C. L. Abad, Y. Lu, and R. H. Campbell, "DARE Adaptive data replication for efficient cluster scheduling," in *Proceedings of the 2011 IEEE international conference on cluster computing*, IEEE, Austin, TX, USA, September 2011.

[19] J. Zhou, Y. Chen, W. Xie, D. Dai, S. He, and W. Wang, "Prs: a pattern-directed replication scheme for heterogeneous object-based storage," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 591–605, 2020.

[20] S. Sheoran, D. Sethia, and H. Saran, "Optimized map file-based storage of small files in Hadoop," in *Proceedings of the 2017 17th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGRID)*, IEEE, Madrid, Spain, May 2017.

[21] Z. Yang, J. Wang, D. Evans, and N. Mi, "AutoReplica: automatic data replica manager in distributed caching and data processing systems," in *Proceedings of the 2016 IEEE 35th International performance computing and communications conference (IPCCC)*, IEEE, Las Vegas, NV, USA, December 2016.

[22] C. Huang, Q. Huang, and D. Wang, "Stochastic configuration networks based adaptive storage replica management for power big data processing," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 373–383, 2020.

[23] R. Honicky and E. L. Miller, "Replication under scalable hashing: a family of algorithms for scalable decentralized data distribution," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, IEEE, Santa Fe, NM, USA, April 2004.

[24] J. Zhou, W. Xie, Q. Gu, and Y. Chen, "Hierarchical consistent hashing for heterogeneous object-based storage," in

Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, Tianjin, China, August 2016.

[25] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, El Paso Texas USA, May 1997.

[26] H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, "A self-organizing storage cluster for parallel data-intensive applications," in *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, IEEE, Pittsburgh, PA, USA, November 2004.

[27] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Compact, adaptive placement schemes for non-uniform requirements," in *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, Winnipeg Manitoba Canada, August 2002.

[28] A. Brinkmann, S. Effert, F. M. auf der Heide, and C. Scheideler, "Dynamic and redundant data placement," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS'07)*, IEEE, Toronto, ON, Canada, June 2007.

[29] P. J. Braam, "The Lustre storage architecture," *White Paper, Cluster File Systems. Inc.*vol. 23, no. 2, 2003.

[30] R.-S. Chang and H.-P. Chang, "A dynamic data replication strategy using access-weights in data grids," *The Journal of Supercomputing*, vol. 45, no. 3, pp. 277–295, 2008.

[31] M. Barrios, J. Barkes, F. Cougard et al., *GPFS: A Parallel File System*, IBM Red Book, New York, NY, USA, 1998.

[32] S. He, Y. Wang, X.-H. Sun, C. Huang, and C. Xu, "Heterogeneity-aware collective I/O for parallel I/O systems with hybrid HDD/SSD servers," *IEEE Transactions on Computers*, vol. 66, no. 6, pp. 1091–1098, 2017.

[33] S. He, X.-H. Sun, and A. Haider, "HAS Heterogeneity-aware selective data layout scheme for parallel file systems on hybrid servers," in *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*, IEEE, Hyderabad, India, May 2015.

[34] H.-H Lin, "Application of a fuzzy decision model to the design of a pillbox for medical treatment of chronic diseases," *Applied Sciences*, vol. 9, no. 22, p. 4909, 2019.

[35] M.-C. Lee, F.-Y. Leu, and Y.-P. Chen, "PFRF: an adaptive data replication algorithm based on star-topology data grids," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1045–1057, 2012.

[36] Y. Yin, J. Li, J. He, X. Sun, and R. Thakur, "Pattern-direct and layout-aware replication scheme for parallel I/O systems," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, IEEE, Cambridge, MA, USA, May 2013.