PRS: A Pattern-Directed Replication Scheme for Heterogeneous Object-Based Storage

Jiang Zhou, Yong Chen, Wei Xie, Dong Dai, Shuibing He, *Member, IEEE*, and Weiping Wang

Abstract—Data replication is a key technique to achieve high data availability, reliability, and optimized performance in distributed storage systems. In recent years, with emerged new storage devices, heterogeneous object-based storage systems, such as a storage system with a mix of hard disk drives, solid state drives, and other non-volatile memory devices have become increasingly attractive since they combine the merits of different storage devices to deliver better promises. However, existing data replication schemes do not well consider distinct characteristics of heterogeneous storage devices yet, which could lead to suboptimal performance. This article introduces a new data replication scheme called Pattern-directed Replication Scheme (PRS) to achieve efficient data replication for heterogeneous storage systems. Different from traditional schemes, the PRS selectively replicates data objects and distributes replicas to various storage devices based on their characteristics. It aggregates objects that have I/O correlation into object groups by calculating object distance and makes replication for grouped objects according to application's data access pattern identified. In addition, the PRS uses a pseudo random algorithm to optimize replica placement by considering the storage device performance and capacity features. We have evaluated the pattern-directed replication scheme with extensive tests in Sheepdog, a typical object-based storage system. The experimental results confirm that it is a highly efficient replication scheme for heterogeneous storage systems. For instance, the read performance was improved by 105 percent to nearly 10x compared with existing replication schemes.

Index Terms—Data replication, heterogeneous storage, object-based storage, access pattern, data distribution

1 Introduction

The object-based storage model, which abstracts files as multiple data objects stored in object-based devices, becomes increasingly important for distributed storage systems in data centers [1]. It has been widely adopted in production systems like Lustre [2], Ceph [3], and Sheepdog [4]. Recently, heterogeneous object storage systems that take advantages of emerged new devices, such as non-volatile memory (NVM) including solid state drives (SSDs), phase change memory (PCM) [5], resistive RAM (ReRAM) [6], in addition to conventional hard disk drives (HDDs), have gained increasing attention. These storage systems leverage different storage devices and combine the merits of them to deliver an efficient storage solution. On the other hand, *replication* remains a key technique to achieve desired data availability and reliability in many storage systems [2], [3], [4], [7], [8]. Its core concept is to automatically replicate data objects and distribute them to multiple devices. With replication, data can be retrieved from other replicas if one copy is not accessible or corrupted. Replication can also be used to improve I/O performance by coordinating clients to access a local or near copy of the data object [9], [10], [11], or by amortizing I/O workload to achieve load balance among multiple copies [12], [13].

Although numerous studies have been devoted to the design and development of data replication schemes, there has been little work on data replication for heterogeneous, object-based storage systems. As the performance and capacity of heterogeneous devices are different, it is critical to place replicas concerning their characteristics. Existing replication strategies mainly distribute data on heterogeneous devices according to device types/tiers, e.g., placing replicas on HDD storage and NVM storage separately [14], [15], [16], [17], [18], [19]. However, these strategies focus only on one characteristic of devices (e.g., the capacity), while ignoring other features, such as bandwidth. They do not distinguish different characteristics of heterogeneous devices and do not work well for heterogeneous storage systems. Our research study presented in this paper addresses this challenge to achieve an efficient, heterogeneous data replication strategy by considering both device characteristics (e.g., capacity and bandwidth) and data reliability.

When making data replication on storage systems, understanding data-access patterns is an efficient way to improve storage system performance [20], [21], [22], [23], [24]. Current replication strategies primarily use frequency or sequentiality to work with heterogeneous storage, in which they place replicas of the hot, frequently accessed objects on fast devices (e.g., NVMs), while the cold, sequentially accessed objects on slower devices (e.g., HDDs) [16]. Several distribution methods have also been used to model data-access patterns, e.g., the distribution of access counts on files of a system or file popularity [25], [26]. Although numerous prior

- *J. Zhou and W. Wang are with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China. E-mail: {zhoujiang, wangweiping}@iie.ac.cn.*
- *Y. Chen and W. Xie are with the Department of Computer Science, Texas Tech University, Lubbock, TX 79401. E-mail: {yong.chen, wei.xie}@ttu.edu.*
- *D. Dai is with the Department of Computer Science, College of Computing and Informatics, University of North Carolina at Charlotte, Charlotte, NC 28223. E-mail: dong.dai@uncc.edu.*
- *S. He is with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China. E-mail: heshuibing@zju.edu.cn.*

Manuscript received 22 Apr. 2019; revised 29 Oct. 2019; accepted 12 Nov. 2019. Date of publication 19 Nov. 2019; date of current version 10 Mar. 2020. (Corresponding author: Yong Chen.)
Recommended for acceptance by A. Sorniotti.
Digital Object Identifier no. 10.1109/TC.2019.2954089

work explored sequential locality or access frequency for performance improvement, they place replicas in a way ignorant of other patterns, such as data-access correlation. For instance, two strongly correlated objects (e.g., two objects that are often successively accessed together but in a varying order) might be placed into NVM (e.g., SSD) and HDD, respectively. If one object is all replicated on NVMs and the other is all replicated on HDDs, it will lead to poor performance when this same access pattern appears again in the future. Even if the replica of the successive objects is placed together on NVMs, it is up to the read policy that selects replicas for reading data that decides the read performance. Besides, it is difficult to determine data-access sequence (sequential or random) in object-based storage for node/server-side tracing because one file is divided into multiple objects, which are distributed on different nodes. When a sequential access occurs on a file, the I/O requests can disperse on different nodes. Even merging all node/server-side traces cannot get the accurate patterns because the time when an I/O request reaches a target node is affected by the network traffic and the local time of the target node. Our study addresses this challenge by finding object correlation via analyzing data-access behaviors. We further aggregate the objects that have I/O correlation into one object group for fast access. Specifically, grouping objects with temporal correlation also improves the read performance since it improves data location and ensures sequential accesses.

In this paper, we propose a new replication scheme called *pattern-directed replication scheme (PRS)* for heterogeneous object-based storage systems. The PRS analyzes applications' access patterns and distributes replicas by considering data-access patterns and heterogeneous device characteristics. It groups objects based on their local pattern, namely *temporal correlation*, or global pattern, namely *frequency correlation*, and merges them for replication. A new object will be created as the replica for object group to reduce I/O access times and benefit spatial locality. Data access sequence within object groups is also considered. Specifically, a pseudo random distribution algorithm is designed to optimize replica layout according to device bandwidth while keeping data balance for capacity utilization. The storage system can benefit from performance improvement for future data accesses with the replication by exploiting the full potentials of fast devices. Compared with the preliminary, conceptual overview of pattern-directed replication scheme presented in our earlier study [27], we introduce the complete design and methodology of PRS in this paper including data-access pattern analysis, object replication for access patterns, and pattern-directed replication strategy. The proof-of-concept prototype we have built and the evaluation we have conducted confirm that the PRS scheme can significantly improve the I/O performance while achieving fair utilization and data redundancy for heterogeneous object-based storage. As far as we know, there are no existing data replication schemes that leverage data-access correlation to place replicas based on heterogeneous device characteristics.

The contributions of this study are four-fold:

- We introduce a pattern-directed replication scheme that can achieve efficient data distribution for heterogeneous storage systems by understanding data-access patterns.
- We introduce a pattern analysis method to identify local or global data-access pattens for objects and cluster objects into groups based on their I/O correlations.
- We design a data replication algorithm to distribute object groups with identified patterns and optimize replica placement with a pseudo random algorithm that fully considers different node characteristics.
- We implement a prototype of the pattern-directed replication scheme based on the Sheepdog storage system. Experimental results confirm that the PRS can significantly improve the I/O performance and effectively distribute replicas among heterogeneous devices.

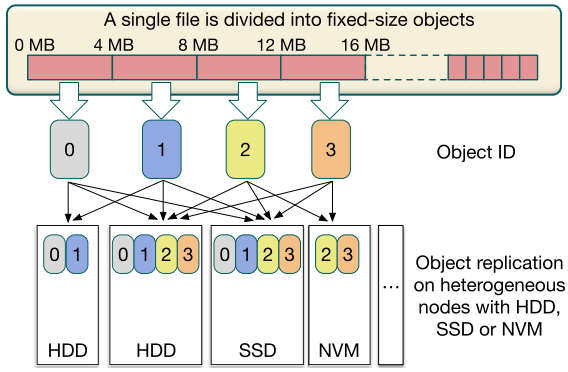


Fig. 1. File division with replication in a typical object-based storage system.

The rest of this paper is organized as follows. Section 2 presents the background and motivation of this research. Section 3 describes the key designs of the PRS scheme, including architecture overview, object access pattern analysis, pattern-directed replication, and optimized data distribution. Section 4 presents the evaluation results. Section 5 reviews related work in data replication scheme and heterogeneous storage systems. Finally, Section 6 concludes this research study.

2 Background and Motivation

2.1 File Striping and Data Objects

To meet the I/O demands of distributed applications, large-scale clusters rely on object-based storage to manage data. A storage object is a logical collection of bytes on a storage device, with proper access methods, attributes describing characteristics of the data, and other considerations like security policy [1]. Unlike block storage, it can be used to store entire data structures, such as files, database tables, etc. In an object-based storage system, a file is divided into multiple fixed-size objects, in which each object is stored separately. Fig. 1 shows an example of data placement in an object-based storage system. In this figure, a single file is divided into multiple 4 MB objects. For each object, there is an object ID corresponding to it, namely *object0*, *object1*, *object2*, etc. The object ID uniquely identifies an object for locating it. All objects and their replicas are distributed across different nodes/devices for storage.

2.2 Replication Management on Heterogeneous Storage

Replication is a key technique for data reliability and fault tolerance. It is a mechanism that automatically copies data and

distributes them with multiple replicas. By replication, the data can be recovered or retrieved from other replicas if it is not available or corrupted. In parallel/distributed file systems, replication is used to improve parallelism by amortizing I/O workload on different nodes. When placing replicas, the storage systems deliver different levels of consistency guarantees to applications for accessing data. To satisfy more acquirements, replication can also help to achieve additional functionalities, such as snapshot and data migration.

Traditional replication strategies can improve data availability but face challenges in heterogeneous environments. A heterogeneous storage consists of different devices with each having distinct characteristics. Compared with slow, albeit cheap, hard disk drives, persistent NVM devices provide impressive performance advantages, but, at the same time, suffer lower density, higher cost, and shorter endurance. If we cannot take advantages of their unique features, it will lead to a suboptimal design for data replication. For instance, in Fig. 1, the I/O performance will be affected if placing most replicas on large-capacity HDDs, while ignoring NVMs. On the other hand, the NMVs will be quickly exhausted if simply concerning on the performance. Therefore, if the replicas are not distributed well, it will result in imbalance on nodes and reduce system efficiency. To address this issue, we propose a novel, heterogeneity-aware data replication strategy that selectively makes replication and distributes replicas in an optimized way by considering device capacity and performance features.

2.3 Data-Access Pattern Matters

In large-scale storage systems, different applications often generate distinct I/O workloads due to nonuniform data accesses. For instance, some applications frequently access certain hot data over a period of time while other applications are not. It has a direct impact on replication efficiency if the hot data are not properly placed on faster devices. On the other hand, heterogeneous device performance can change with different data-access patterns, depending on access sequence, request size, etc. Lacking awareness of this variability will lead to resource waste and potential performance degradation. Thus it is highly desired to explore application behaviors for making data replication. In this study, we leverage object correlation to discover local or global data-access patterns, consider access sequence in each pattern, and then use identified patterns to accommodate data replication on heterogeneous storage systems.

3 Pattern-Directed Replication Scheme

In this section, we introduce the detailed design of the pattern-directed replication scheme for heterogeneous object-based storage systems. The success of the PRS scheme relies on solving two technical issues: analyzing object access pattern to guide data replication and optimizing replication layout for the heterogeneous storage environment, which will be discussed in detail below.

3.1 Architecture Overview

The fundamental idea of the PRS is to consider different storage device features and adopt a *pattern-directed* method to aggregate objects with respect to identified access

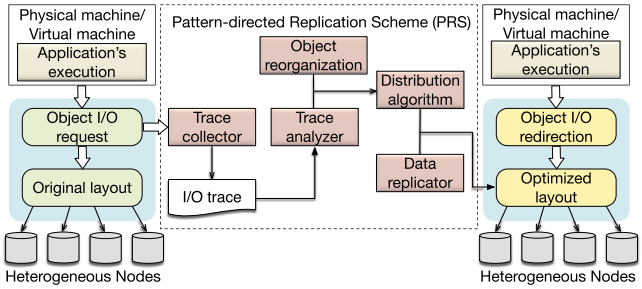


Fig. 2. Overview of the pattern-directed replication scheme.

patterns among objects for replication in a heterogeneous storage system. It can improve data reliability and performance efficiency based on existing replication strategies of storage systems. The principle of this proposed scheme is described below.

First, as shown in Fig. 2, the application running on physical machines or virtual machines (VMs) sends I/O requests to the backend object-based storage system. The objects are placed following the *default data layout for its first and second replicas*, e.g., via the hashing-based algorithm to distribute data which depends on the system design [4]. The data is initially replicated with two copies for the redundancy goal so that it is accessible if one copy is corrupted.

Second, *the rest replicas for an object will be created with PRS with pattern analysis*. These replicas are determined for the performance benefit since they are generated by considering access patterns and heterogeneous device characteristics. The data-access pattern analysis is based on the historical object I/O requests that are traced by a tracing collector in each storage node (server-side tracing). The analysis focuses on five parameters: 1) object ID, 2) start offset in an object, 3) size of the access, 4) access time, and 5) operation code (e.g., read or write). The object I/O trace can be collected online during application execution. In a distributed storage system, each storage node generates one trace file that contains all its I/O requests for which the node receives or forwards. The trace file is used for data-access pattern analysis and object reorganization for replication in the node.

With this information, the PRS analyzes the trace to identify data-access patterns. It groups objects based on identified local pattern or global pattern for determining the third or more replicas. The analysis phase can be performed online or offline, and we conduct it in the offline manner in the current study. The reason is two-fold. First, the trace analysis is carried out for each application run, which depends on the entire application behaviors instead of requiring frequent real-time analysis. Second, the analytic results can be obtained in the background and used for replication as the PRS generates new grouped object replicas in an asynchronous way. Its influence on the system performance can be negligible.

Finding data-access patterns and grouping objects can also be used in a homogeneous environment. For example, we can merge correlated objects and place them on a single HDD instead of multiple HDDs for an optimization. However, in a heterogeneous storage system, the devices have different characteristics, which can be inefficient if making replication on them equally. To distinguish heterogeneous devices, a further optimized replication algorithm is proposed for data

distribution. With PRS replication, the original objects resident on one node may be replicated in other nodes. It does not affect data access as PRS find the replica via an I/O redirection to the new optimized data layout.

Third, with identified patterns and PRS-generated replicas, future accesses will benefit from the scheme. When an object is being accessed and its PRS-generated replica is found, a read policy may read the PRS-generated replica via the object I/O redirection layer and the subsequent reads are pre-loaded. Compared to the non-PRS replication schemes that may separate correlated object replicas in different devices, the PRS can improve the performance significantly.

3.2 Data-Access Pattern Analysis

The goal of PRS scheme is to provide an efficient data replication strategy for heterogeneous storage systems. To achieve that, we explore application behaviors from I/O trace and make data replication based on analysis results.

3.2.1 Access Pattern Definition

In this study, we use the object accesses for pattern analysis and leverage the identified patterns to direct data replication. We define two types of data-access patterns: a local data-access pattern and a global data-access pattern. The *local data-access pattern* represents the temporal-based correlation, which reflects the object access order from a temporal view. The I/O requests may come from different processes or applications, but they will be eventually converted to object requests on underlying storage at the storage node side. Objects that are accessed together in a given interval and accessed periodically belong to this pattern. Here we do not consider spatial correlation for local patterns because the neighboring objects with related object IDs (e.g,. object 1 and object 2) may belong to different files and are distributed to different nodes.

The *global data-access pattern* is defined to represent the objects that are most frequently accessed during the application run time. This type of pattern can be analyzed by synthesizing the traces on all storage nodes. To find hot objects in the global pattern, we consider a certain percentage of objects as hot data in storage, which is decided according to the architecture configuration of heterogeneous systems. The rationale of this design choice is rather straightforward, as the capacity of fast devices (e.g., NVMs) is limited in a heterogeneous storage system, where the hot data should be placed on these fast devices with higher priority. For example, in an environment with HDDs and NVMs where the capacity ratio of HDDs and NVMs is *9:1*, then *10* percent of objects are selected as hot data for the NVMs proportion. These frequently accessed objects identified from the global pattern have a higher priority to be placed in the NVM devices with higher bandwidth, such as SSDs, with the PRS. In many situations, local patterns alone cannot reveal the behavior of an application, and a global view is necessary for guiding replication.

To better demonstrate the difference between local and global patterns, we show a real example from tracing the Sheepdog [4], a distributed object storage system, on a cluster while performing the FIO benchmark [28]. Fig. 3 shows

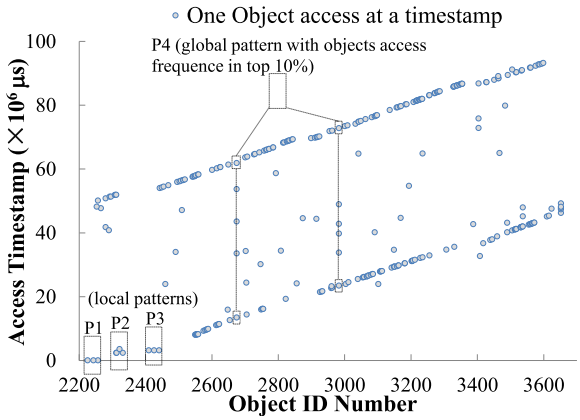


Fig. 3. Local and global data access patterns in real trace.

the result, which is representative and observed in nearly all our tests. Each point in the plot indicates that an object is accessed at one timestamp. These points can be divided into various groups with local patterns, where each group represents temporal correlation depending on the distance selection. For example, the objects in *pattern1*, *pattern2*, and *pattern3* (*P1, P2, P3*) belong to *local patterns* as these objects in each pattern are accessed within 10 milliseconds. On the other hand, *P4* denotes a global pattern, in which two objects (object ID is 2675 and 2983) are frequently accessed with the access frequency in top 10 percent.

3.2.2 Object Distance Calculation

We define two types of data-access patterns as discussed above and the PRS analyzes the trace to identify those patterns. The objects with identified access patterns are grouped for replication. There are two reasons to group objects that are temporally related or frequently accessed.

First, grouping based on temporal correlation with local pattern can help reduce I/O access times and benefit from the locality. The grouping transforms multiple object requests (random or sequential operations) on different nodes into one single access on one node, which avoids additional network communication and disk I/O overhead.

The second reason is that these frequently accessed objects with global pattern are preferred to be placed on fast devices, such as NVM. It is not always possible to identify the correlation among data accesses. However, considering the cache layer in storage systems, grouping hot data can help improve the read performance as these data are frequently accessed in NVMs with higher bandwidth. Grouping hot data with global pattern may lead to hotspotting in fast devices. We address this issue from two aspects. First, our data replication strategy distributes object groups among various devices according to device bandwidth via a pseudo-number algorithm, instead of centralizing them on a few devices. Second, the read performance can be alleviated by accessing a data replica on other devices if one device is overloaded.

To identify data-access patterns, the method of file/block distance calculation is widely used in storage systems [29], [30]. In a distributed object-based storage system, we use the object distance calculation between two objects to identify data-access patterns. We further use an object classification algorithm to group objects based on the object distance. To

avoid the size of one object group is too large, we limit the object number to a predefined value in each group.

For the local access pattern, both the data-access interval and order are important. The object trace collected for applications is a list of object requests in chronological order. If the interval of two object accesses is very long from each other in terms of the access time, their distance should be far, which is not beneficial for replication. Also, if two object accesses are close in terms of the time, but they only co-occur occasionally (e.g., once only), then their correlation should not be considered stable. The PRS restricts the time interval to compute the object distance. Specifically, it examines how far apart two object accesses are. A time window is defined by the measured time elapsed between two continuous object accesses. A maximal threshold is specified, denoted as max_{window}. Once an object access happens later than the max window after the current one, the PRS considers it out of the temporal order and does not count them as continuous accesses. For any two objects that are ever accessed in a period of time (i.e., falling into the same max_{window}), the PRS further calculates their access times in the same max_{window} as an indicator of their distance, as defined below:

$$dist(o_1, o_2) = 1 - Min\left\{\frac{count(o_1, o_2)}{count(o_1)}, \frac{count(o_1, o_2)}{count(o_2)}\right\}, \tag{1}$$

where $count(o_1, o_2)$ is the access counts of both objects (o_1, o_2) accessed in the max_{window}, and $count(o_1)$ and $count(o_2)$ are the counts of these separate objects. The max_{window} is a fixed value that can be preset for the trace. As we can see, the distance function reflects the access correlation of two objects. The larger counts two objects are accessed temporally in the max_{window}, the less distance between them.

For instance, for two objects, say object 1 and object 2, the $count(o_1, o_2)$ is 3 if they are accessed together for three times within a time window. Assume these two objects are accessed for one and two times at other timestamps (not in the same time window), then the $count(o_1)$ and $count(o_2)$ are 4 and 5, respectively. With the Equation (1), the distance between object 1 and object 2 is $dist(o_1, o_2) = 1 - Min\{0.75, 0.6\} = 0.4$. Given the constant value $count(o_1, o_2)$, the distance between object 1 and object 2 will increase if $count(o_1)$ or $count(o_2)$ is larger. The increase of distance means that object 1 and object 2 have lower correlation because they are not always accessed together.

Different from local patterns, we consider the data-access frequency to find objects for global access patterns, which means only the most frequently accessed data will belong to the global data-access pattern. As mentioned above, the PRS selects a certain percentage of objects as hot data according to heterogeneous device configuration. For the global data-access pattern, the distance between two objects is calculated with their hotness measure, as described below. To better support the read performance, we calculate the $count(o_1)$ and $count(o_2)$ as the read counts of each object. The more frequently two objects are accessed, the less distance between them.

$$dist(o_1, o_2) = \left\{\frac{|count(o_1) - count(o_2)|}{count(o_1) + count(o_2)}\right\}. \tag{2}$$

For instance, if object 1 and object 2 have the values $count(o_1) = 55$ and $count(o_2) = 45$, then their distance $dist(o_1, o_2) = 0.1$. These two objects are in close correlation for similar access frequency. However, the distance between two hot objects can be high if there is a gap between their access counter. Given the $count(o_1) = 150$ and $count(o_2) = 50$, the object distance is $dist(o_1, o_2) = 0.5$, which means object 1 and object 2 have low correlation.

3.2.3 Object Clustering Algorithm

With the distance formulas, we can calculate the distance between any two objects and further classify them into different groups through clustering algorithms. Specifically, we first classify hot objects into groups according to frequency correlation, then classify the objects that have temporal correlation. In other words, objects are grouped based on both the global and local patterns. Two issues remain to be resolved. One is that the group number is unknown, and we may need to add or remove groups accordingly. The other is that the number of objects in each group should be limited to avoid the size of an object group to be too large.

We introduce the clustering algorithm by describing the assignment of each new object as shown in Algorithm 1. We define dis_thr as the maximal access distance in a group. If two objects are classified in the same group, their distance should be less than dis_thr. We further define obj_thr as the maximal number of objects that belong to the same group. If the number of objects classified into the same group exceeds this threshold, we will decrease the value of dis_thr to split the objects apart to avoid a large, single object group. We define a factor α as the minimal adjustment parameter on dis_thr each time.

Algorithm 1. Object Clustering Algorithm

```
1: procedure Clustering obj
2:   (dis_min, near_g) = min_io(groups, obj)
3:   if dis_min ≤ dis_thr then
4:     near_g = assign(obj, near_g)
5:     if size(near_g) > obj_thr then
6:       dis_thr− = α
7:       Re − run the clustering algorithm
8:     end if
9:   else
10:    new_g = create_new_group(groups, obj)
11:  end if
12: end procedure
```

During object grouping process, the *clustering* function will first call *min_io(groups, obj)* to calculate the nearest group ($near\_g$) and also the minimal distance (dis_min) for a new object obj. If the minimal distance is less than the distance threshold dis_thr, this new object should be put into that group $near\_g$, which increases the number of objects in that group too. If the number of objects in this group $size(near\_g)$ is larger than the threshold obj_thr, we will adjust (i.e., decrease) the dis_thr by α to constrain the number of objects in one group. Once the distance threshold dis_thr is changed, we need to re-run the clustering algorithm on these objects again to adjust other groups. If the minimal distance dis_min between objects to any group is

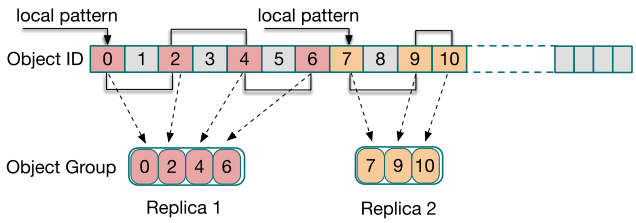


Fig. 4. Grouping objects with local patterns. Objects 0, 2, 4, 6 and objects 7, 9, 10 belong to two different local patterns.

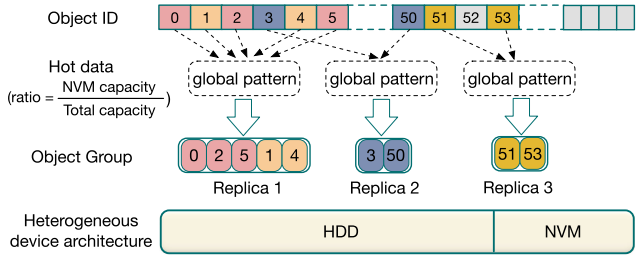


Fig. 5. Grouping of objects with global patterns. Objects 1, 4 also belong to one local pattern.

larger than the distance threshold dis_thr, then the new object obj should be placed into a new group created by calling *create_new_group*.

3.3 Object Replication for Access Patterns

With the object clustering algorithm, the PRS groups objects based on the local pattern and global pattern by analyzing the I/O accesses of applications. The objects in the same group are aggregated to a new object and replicated. The PRS only replicates the data with identified access pattern, which are generated as the third or the rest replicas for objects. The granularity of grouping is object-based, which means that the entire object will be grouped for replication even if part of its data is accessed.

Fig. 4 illustrates the grouping for objects with local patterns. It can be seen that multiple objects with identified local patterns are aggregated to one object. For example, the object group *replica 1* consists of 4 objects, in which they belong to one local data access pattern according to the clustering algorithm. Each of the object group is a new object with a new *object ID*. It will be replicated as separate files in the dedicated directory on the local file system where the original objects are stored.

For an application, the data are often accessed with different frequency. The PRS replicates the hot objects identified in the global pattern, as shown in Fig 5. According to the ratio of NVM devices' capacity and the total capacity in a heterogeneous environment, a certain percentage of objects are selected as hot data by sorting read counts. With the object distance between each two objects, the hot data are clustered into different groups in global patterns. The objects in the same group are reorganized to create a new object for replication. From Fig. 5, it can be seen that there are three replicas, each of which has multiple objects for identified global patterns. The replication of the global pattern can also include objects with the local pattern. For example, *replica 1* has two objects (e.g., object 1 and object 4) that belong to one local pattern.

When grouping high frequency objects to merged replicas, the high frequency objects are not necessarily accessed together. It would cause reading amplification which reads too much for what actually needs. However, this issue can be relieved by turning down the ratio of hot data to merge only a small part of hot data. Moreover, we can also limit the hot data amount of a group by considering object access time. For instance, the hot data in a group need to be accessed in a certain period, such as one minute.

For the objects with identified access pattern, the PRS first replicates them with a global pattern. This means the hot objects will first be grouped for replication. The remaining objects that do not belong to any global pattern will be grouped for identified local patterns. The objects which are not in global or local patterns will not be replicated with PRS. The number of replicas can be pre-defined, which reflects the redundancy of objects in the storage system. When objects are grouped into a new one, they are reorganized in a random order, where the index of each object in the group will be recorded in a redirect table for search. Then all replicas will be placed in an optimized layout on the heterogeneous environment to improve the I/O performance and keep data balanced across nodes, as described in next subsection.

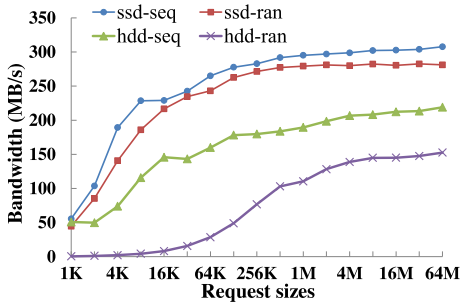
3.4 Pattern-Directed Replication Strategy

When generating replicas for objects with local or global patterns, how to place replicas according to different device characteristics is an important issue. To address this problem, we adopt a heterogeneous, device-aware data replication strategy. The novelty is that we consider both device capacity and bandwidth for data distribution and place replicas for objects with identified access patterns on different devices. More specifically, we adopt a pseudo random number algorithm to optimize replica layout in a heterogeneous environment. It is inspired by the SPOCA [31], [32] and SUORA [18] algorithms, but extends them by distributing data based on data access patterns and device characteristics.

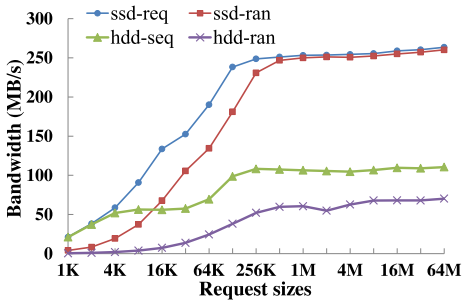
3.4.1 Access Sequence in Local or Global Patterns

We have described how to group objects with local or global patterns for their I/O correlations. To decide which device an object group should be best placed, it is also critical to consider data access sequence within the object group because device performance can change with different I/O workloads. In order to achieve an optimized data distribution, we first analyze device performance (specially we compare HDDs and SSDs) under different I/O workloads, and then introduce how to distribute object groups on heterogeneous devices.

Fig. 6 shows the I/O performance comparison of HDD and SSD, where the tests were conducted with FIO benchmark [28] on one node with Ext4 file system. From the figure, two observations can be made. First, it can be seen that SSD favors workloads with random and small I/O size. For instance, the bandwidth of SSD is from 84.2 to 6.5 times than that of HDD for random read when the request size varies from 1 KB to 128 KB. The similar results are observed for random write operation. However, the SSD only achieves 1.45 times to 3.7 times of the HDD bandwidth with the



(a) Read workloads



(b) Write workloads

Fig. 6. I/O bandwidth comparison of the HDD (Seagate ST9500620NS) and the SSD (Intel SSDSC2BA200G3T) with random and sequential workloads.

random request size larger than 128 KB. Second, the performance gap between SSD and HHD is small for sequential access. The improvement on bandwidth is from 1 to 3 times for SSD compared with HDD under sequential read and write operations. Although the results were collected and tested with specified devices, they represent the typical difference of HDDs and SSDs.

The test results show that access sequence has an impact on the performance of data access. In order to explore access sequence, we further analyze *access sizes* of objects in one object group. For objects with local pattern, if each access size is less than a threshold, such as 128 KB, the object group can be identified as "random and small access" because all data accesses in it are dispersive with small size. This object group should be placed on SSDs because SSDs achieve much better performance than HDDs for this pattern. On contrary, objects with "large size access" can be placed on HDDs because there is no significant performance gap between HDDs and SSDs in this case. Particularly, an object group can be regarded as "sequential access" when access length of all objects in the group reach the object size (e.g, 4 MB). Similarly, for global pattern, objects with "random and small" access should be placed on SSDs. Objects that are accessed with large sizes or sequentially are placed on HDDs. These objects can also be placed on SSDs because they are hot data. Fig. 7 illustrates data access sequence in local or global patterns.

3.4.2 Optimized Data Distribution in Heterogeneous Storage

In this section, we introduce the design of optimized replication placement algorithm on heterogeneous storage systems by considering both object access patterns and device

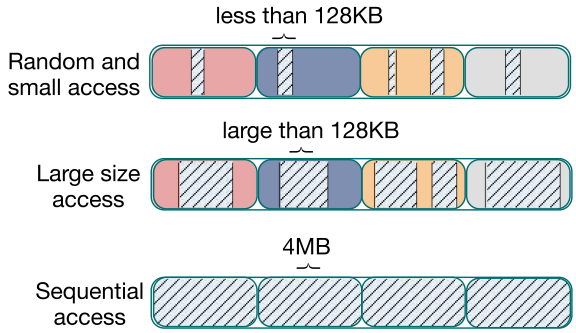


Fig. 7. Three patterns of data access sequence in the object group with local or global patterns. In each pattern, one object group is used as an example with a few objects in it. Assuming the object size is 4 MB and the threshold between small and large-size access is 128 KB, the figure shows different access sequence: (1) in the random and small access, data are accessed with small sizes in different objects; (2) in the large size access, data are accessed with large sizes in different objects; (3) in the sequential access, data are accessed with full object size in consecutive objects. If there are multiple accesses on the same object (e.g., an object is accessed more than one time in the same period), the maximize access size is used.

characteristics. Specifically, we adopt a pseudo random number algorithm to distribute object groups in a heterogeneous environment. This approach extends SPOCA [31], [32] and SUORA [18] algorithms by considering both node performance and node capacity.

With this idea, PRS first organizes heterogeneous nodes (or devices) by assigning them to different segments in a logic number line. Each node is assigned to one or more segments according to its average bandwidth via dividing the bandwidth by a performance parameter p, which can be predefined by users or according to device configuration (e.g., the minimum value of all average bandwidth).

$$Segment\ length = \frac{Node\ bandwidth}{p}.$$

The segment begins with the point of an integer number with the maximal length set to 1. When the segment length of a node exceeds one segment, it is assigned to a new consecutive one with the smallest segment number in the number line. Fig. 8 shows an example of segment assignment for storage nodes with different specifications in Table 1. Given $p = 360$, *Node 1* and *Node 2* are assigned to segments *seg_0* and *seg_1* with the segment length 0.41 and 0.73, respectively. Note that *Node 3* is assigned to two segments (seg_2

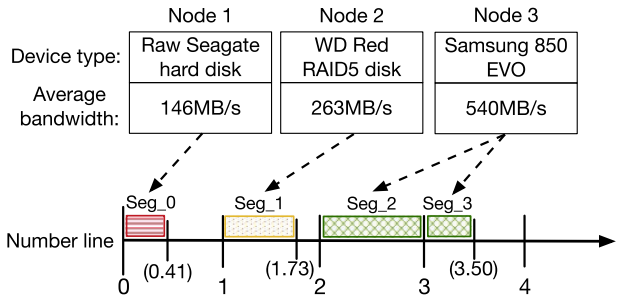


Fig. 8. Mapping of nodes and segments. Nodes are divided into various segments in a logic number line, in which a specific device is assigned to one or more segments for *performance feature* (e.g., four segments for node1 to node3 with their segment ranges being [0, 0.41), [1, 1.73), [2,3), and [3, 3.5)).

TABLE 1
The Specification of Data Nodes in an Assumed Cluster

Node number	Segment number	Device name	Capacity (GB)	Max throughput (MB/s)
1	0	Raw Seagate hard disk	1000	146
2	1	WD Red RAID5 with 4 disks	500	263
3	2, 3	Samsung 850 EVO	256	540

and seg_3) because its segment length is larger than 1. As the node is assigned to segments proportional to the bandwidth, the value of p does not affect the replica placement.

After the segment assignment of nodes is determined, the data is then mapped to one segment by pseudo random hash functions (e.g., the SIMD-oriented Fast Mersenne Twister (SFMT) algorithm [33]). We choose the pseudo random number mapping because it can distribute data proportional to the segment length (also is the node bandwidth). As there may have gaps between node segments (due to different segment length), a random number sequence $\vec{R}$ is generated according to data ID until it fits the range of one segment. The random numbers in the sequence $\vec{R}$ are generated in a given range $[e,\ w)$ by a hash function $f(x,\ s)$, where x is data ID, s is the seed, and e and w are lower and upper range limits of distribution, respectively. In most cases, the value of e is 0, and the value of w is the position of segment with maximum number. If the seed for a data ID is the same, the same random number sequence will be generated. For example, in Fig. 8, the range $[e,\ w)$ can be set to $[0,\ 4)$. Supposing the number sequence of one data is $\vec{R} = \{0.8,\ 2.3\}$, the data will be placed on *seg_2* because the number 2.3 first fits *seg_2* in the number line. If the number of replicas is m, there are at least m random numbers generated for mapping m segments in the $\vec{R}$.

To further optimize the replication placement algorithm, data access sequence is considered to distribute data on heterogeneous nodes. For the object group with random and small access, if it is assigned to HDDs by the pseudo algorithm, new random numbers will continue to generate until the object group maps to NVMs (e.g., SSDs). For the object group with large size and local patterns, it will be finally mapped to HDDs even if random numbers match NVMs at first. Otherwise, the object group will be distributed in a general way as discussed before via the pseudo random number algorithm.

With our algorithm, one node may contain large amounts of data for high bandwidth (e.g., an NVM node compared with an HDD node). To address this issue, data replicas will be placed on other nodes if the node has no enough storage space. Algorithm 2 details the optimized replication distribution algorithm in PRS. The *Match* function means the process of mapping an object group to nodes using the pseudo random number algorithm. It ensures efficient usage of heterogeneous device performance and to balance the consumption of the remaining capacity of each node.

3.5 Object I/O Redirection

The function of the object I/O redirection layer is simple and straightforward. To find the original object in an object group, the I/O redirection layer re-calculates the offset and maps the object to its replica (object group) with the optimized data placement algorithm. A typical data access request usually includes three parameters *object id*, *data offset* and *request size*. With these parameters, as well as access patterns, an object redirect table is built to transform the data access from an object to its replica.

Object ID	Replica object ID	Object offset	Pattern	Sequence
0	100	0	local	random
1	101	4MB	global	large
2	102	8MB	local	sequential
⋮	⋮	⋮	⋮	⋮

Fig. 9. The data structure of redirect table in PRS.

Algorithm 2. Replication Placement Algorithm

INPUT data ID x, replica number m, segment number n, threshold of remaining capacity t, seed s;

```
segment[n] = segment array
for i = 0; i < m; i++ do
  val ⇐ hash(x, s)
  while x ∉ segment do
    for j = 0; j < n; j++ do
      if Match(val, segment[j]) and
       remaining capacity of segment[j] > t then
          segment[j] ⇐ x
          node assigned to segment[j] ⇐ x
        end if
      end for
      val ⇐ hash(x, s)
  end while
end for
```

Fig. 9 shows the data structure of the redirect table. It is a mapping table that maintains the relationship between *object ID* and *replica object ID*. The *object offset* indicates the start address of the original object in the replica object (which contains an object group). The *pattern* field marks whether the replica object is in local or global pattern. The *sequence* represents access sequence in the replica object, such as random and small access, large size access and sequential access.

The mapping in the redirect table will not be modified once the objects are replicated. This means the replica position and object group will not change for an object with local or global patterns. It can keep replica consistency and reduce data migration. The redirect table can be constructed when making replication and only records the mapping for objects that are replicated. As PRS analyzes I/O trace for each storage node, each storage node has an independent redirect table. Different nodes have a different view of the mapping and there is no need to be globally updated (we do not consider node changes in this study).

For object *read*/*write* operations, the PRS will first search the object in the redirect table. If an object is replicated, it is located in the replica by *object offset* and accessed with *data offset* and *request size* when reading. To reduce the overhead of search in the redirect table, we use a binary search tree for data structure management (the evaluation in Section 4.6

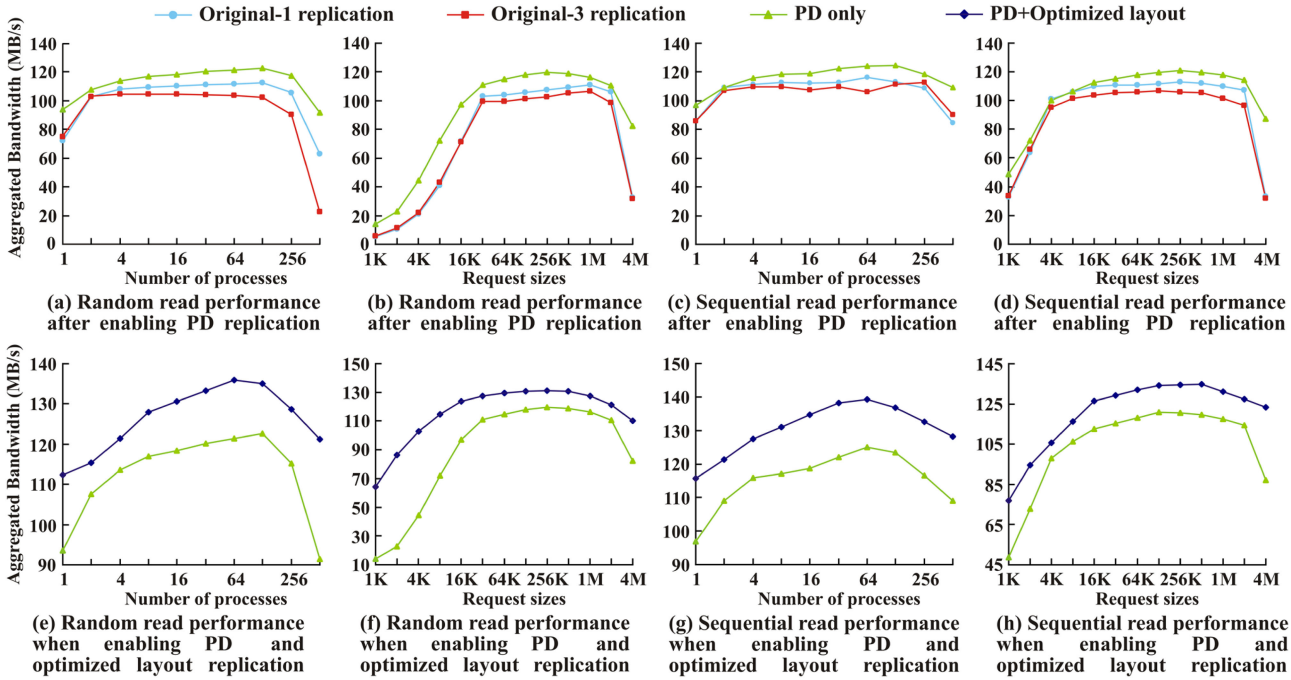


Fig. 10. FIO performance improvements with pattern-directed replication and optimized data distribution.

shows that there is a negligible influence on performance). As the random number sequence is determinate for an object ID, an object can be redirected correctly to its replicas with our optimized data placement algorithm.

4 Evaluation

In this section, we present the evaluation results of the proposed PRS. We implemented the PRS in Sheepdog [4], a distributed object-based storage for virtual machine storage, which is a typical storage scenario in data centers.

The experiments were conducted on a local 32-node cluster, including 16 storage nodes and 16 client nodes hosting VMs. In the storage nodes, three of them are equipped with dual 2.6 GHz Xeon 8-core processors, 16 GB memory, and a 200 GB Intel SSD (average bandwidth 380 MB/s, SSDSC2 BA200G3T); others have dual 2.5 GHz Xeon 8-core processors, 16 GB memory and a 500 GB Seagate SATA HDD (average bandwidth 160 MB/s, ST9500620NS). The VM running in the client node was emulated by KVM-QEMU and configured with 2 vCPUs and 8 GB RAM. The VDI was created with 128 GB capacity. All nodes run Linux kernel 2.6.32 version with 1Gbit/s network connection.

Each storage node in Sheepdog can be used as a *gateway* node to receive or forward I/O requests from clients. The PRS analyzes the trace of the first replica for objects and creates *redirect table* for replication. The trace collector is implemented based on Sheepdog to collect data accesses of each node in real-time with little runtime overhead (below 1 percent). In our tests, we first ran the application for one time to collect the I/O trace. Then the objects with identified data access patterns were replicated asynchronously, thus the application can benefit from the PRS for accessing replicas in the future. With the trace data in our experiments, the max_{window} was set as 10 milliseconds, and the maximum access distance dis_thr and the maximum number of objects in a group obj_thr were 0.1 and 10, respectively. The thresholds of remaining capacity (in Algorithm 2) and of identifying random and small access were 10 percent and 128 KB. We use the SIMD-oriented Fast Mersenne Twister (SFMT) [33] algorithm to generate pseudo random numbers to optimize replication placement, with the parameter $p = 160$.

4.1 Evaluation With FIO Benchmark

The purpose of the evaluation presented in this subsection is to study the effectiveness of the proposed PRS in a heterogeneous environment. The tests were conducted with 8 HDD nodes and 2 SSD nodes, in which the capacity ratio of SSDs is near 10 percent. One client VM was launched to perform the FIO benchmark. We tested PRS on two aspects: evaluation on pattern-directed replication and optimized data distribution.

First, to ensure we observe the improvement only from applying the pattern-directed replication, we disabled the optimized data distribution in these tests. Therefore, the replication and original objects use the same data layout. We use the formulas in Section 3.2.2 to calculate object distance and adopt Algorithm 1 to group objects. The grouping process only takes several seconds (e.g., $0.95s$ for distance calculation and $1.57s$ for grouping objects for 100,000 data accesses), hence has little impact on our asynchronous data replication scheme.

Fig. 10a, 10b, 10c, 10d show the read performance of enabling pattern-directed replication (*PD only*) compared with the original data access in Sheepdog configured with one original (*original-1 rep*) and three copies (*original-3 rep*). The PD replication makes 3 copies of objects with identified access pattern and places them using a consistent hashing algorithm, which is also the default distribution algorithm in Sheepdog [4], [34]. For the object that is replicated with PD replication, it will be read from the replica (object group) via the I/O redirection layer. In Fig. 10a and 10c, we run FIO

benchmark with 1 to 512 jobs in the VM, which uses Sheepdog as the backend storage. Each job accessed an independent file with 100 MB in an asynchronous way to make use of the concurrent access of storage nodes, and the request size was 256 KB. For *PD only* test, it reads the same files but accesses data from copies generated based on the pattern-directed replication. The objects with identified access pattern are replicated and grouped for access location and performance improvement. It can be observed the average bandwidth was improved by 2 to 303 percent.

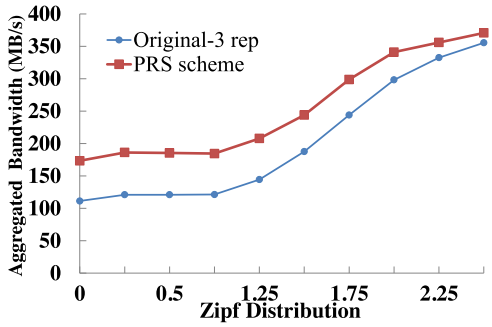
With more jobs, the performance gradually increased till it reached a stable value and decreased after the number of jobs reached 256. The cause of the decrease was that each storage node needs to serve more jobs, which makes the competition among requests in the I/O queue more severe. Compared with the original data access in Sheepdog, the rate of bandwidth degradation was much lower for PD. These tests confirmed the effectiveness and scalability of our replication scheme, which aggregates and replicates objects with identified access patterns to improve I/O performance. Fig. 10b and 10d show the similar results when we run FIO with request size from 1 KB to 4 MB. The performance of *PD only* was 101 to 273 percent higher than that of original data access. When the request was smaller, average bandwidth became lower as the storage node needs to serve more requests, which resulted in more time for disk seeking.

Second, we conducted experiments to study the impact of the optimized data distribution on data access performance. The replication is stored in two different ways: one is the default consistent hashing algorithm (*PD only*), and the other is by the optimized data layout (*PD + optimized-layout*). In both data distributions, the number of replicas was set to 3.

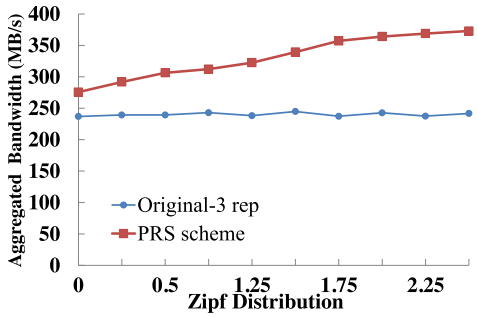
Fig. 10e, 10f, 10g, 10h show the results of reading performance with FIO benchmark by adjusting job numbers or request sizes. It can be seen that the optimized replication distribution contributed an extra 109 to 460 percent performance improvement based on the performance that was already achieved by PD replication scheme. This is because that the replicas with identified access patterns have higher priority to be placed on the SSD nodes, which improves the I/O performance. The overall performance was improved by 1.05 times to 11.56 times when using both *PD* and *optimized-layout* replication in the PRS. Note that all the average bandwidth reached a similar peak value during tests. It is because the FIO performance in the VM is limited by the physical network bandwidth in the node where the client is running.

4.2 Performance Improvement on Skew Data Distribution

To measure the performance improvement of the PRS on unbalanced data access, we run FIO benchmark with workloads of random and sequential I/O with increasingly skew access distribution (Zipf distribution) [35]. The tests were conducted on Sheepdog running on the same node setting as aforementioned, where a VM client is in the SSD node. It generates more skew data distribution with the increase of Zipf distribution value, ensuring that some part of the data is more frequently accessed than others.



(a) Random I/Os with increasing skewness



(b) Sequential I/Os with increasing skewness

Fig. 11. Read performance improvements with different data skewness.

Fig. 11 shows the read performance by comparing the original system with three replicas to the PRS enabled system. It can be observed that both the original system (*original-3 rep*) and PRS had better performance with the increase of Zipf distribution value. The performance increase is because more data become hot and can be accessed from the objects/replicas in the SSD nodes. Moreover, the PRS scheme outperformed original data access in both random and sequential workloads. The performance improvement is mainly because PRS can distinguish hot data and aggregate hot data with global patterns for replication. These replicas of hot data are placed in SSDs with higher priority via the Algorithm 2, which contributes to the read bandwidth in PRS. On contrary, the original data access treats heterogeneous devices uniformly with consistent hashing distribution. Although it may place some object replicas in SSDs, the benefit on performance is limited.

The bandwidth observed was also higher than previous results in Section 4.1 because the client can read data directly from the local SSD device. In a nutshell, our replication scheme achieved higher performance by identifying data access pattern and taking advantage of heterogeneous device characteristics.

4.3 Evaluation With File System Workloads

In this subsection, we present the results of experiments conducted using the benchmark Filebench [36] that emulates file system level workloads of real applications. The specification of workload is described in Table 2, where *R/W* means the ratio of reads and writes, and *WF* means reading or writing a whole file. The configuration is 8 HDD nodes, 2 SSD nodes and 1 client node. We compare the PRS with consistent hashing algorithm [34] and straw buckets (a

TABLE 2
The Specification of Emulated File System Workloads of Different Applications

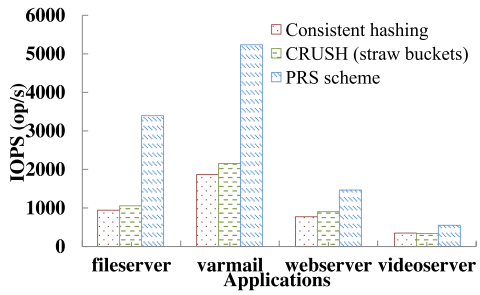
Application	Dataset	R/W	Ave File Size	I/O size
fileserver	36 GB	1:2	256 KB	WF
varmail	15 GB	1:1	32 KB	WF
webserver	25 GB	10:1	256 KB	WF
videoserver	42 GB	1:0	1 GB	1 MB

typical replication algorithm in CRUSH) [14] based on Sheepdog, with each one making 3 replicas.

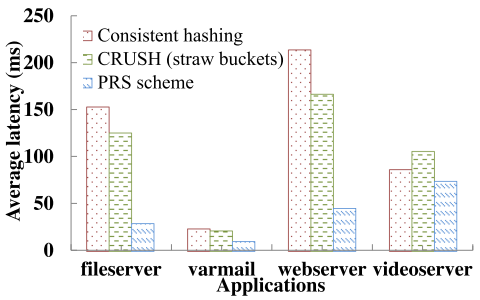
Fig. 12 shows the performance comparison of different replication schemes under various file system workloads. It can be observed that applications with the PRS scheme achieved the best performance. This performance advantage is because that the PRS identifies data access patterns and selectively makes replication in an optimized data placement. It considers both the I/O request patterns and heterogeneous device characteristics. In contrast, the consistent hashing and straw buckets algorithms place data among devices with a uniform distribution, without distinguishing different device merits. The results further confirm the efficiency of the PRS.

4.4 Evaluation with Multiple Clients

To further study the performance benefit of the PRS, we use multiple clients running on different nodes to perform read operations and get the total result by adding each bandwidth of them. The tests were conducted on 15 storage nodes and 16 client nodes with replica number 3. We run 1 to 16 clients to launch FIO on different virtual machines. Each FIO instance has 16 jobs, in which each job accessed an independent file with 100 MB in an asynchronous way with the request size of 256 KB.

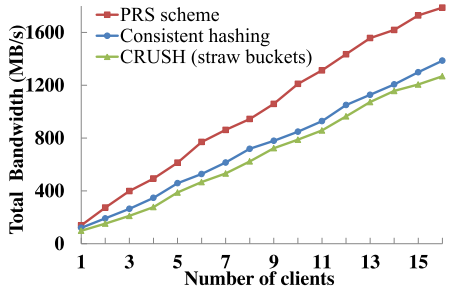


(a) IOPS under workloads of different applications

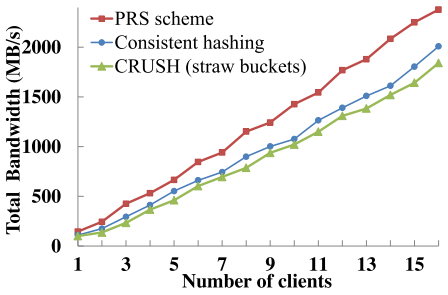


(b) Average latency under workloads of different applications

Fig. 12. Performance under workloads of different applications.



(a) Random read performance



(b) Sequential read performance

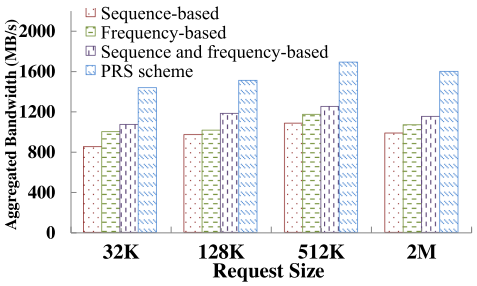
Fig. 13. FIO performance comparison with multiple clients.

From Fig. 13, it can be observed that the performance of all algorithms improves with the increase of client number. This is because Sheepdog distributes objects on various storage nodes and provides concurrent data access for clients. Compared with consistent hashing and CRUSH algorithms, the PRS scheme shows an improved bandwidth. It achieves a more rapid growth for performance with the increase of clients. As the PRS groups objects with identified access patterns, it can reduce the I/O access times and network communication cost. Simultaneously, the PRS scheme places replications in an optimized layout, which efficiently use the performance advantage of SSDs. For consistent hashing and CRUSH algorithms, they place replications on HDD and SSD nodes evenly, which cannot distinguish the different device characteristics.

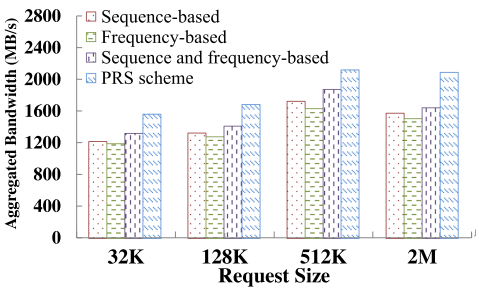
4.5 Evaluation and Comparison of Different Schemes

We have also conducted a series of tests to compare PRS with other replication schemes. For comparison, we also implemented three replication schemes in Sheepdog based on data access sequence or frequency patterns, in which the methods are widely used in existing works. In the *sequence-based scheme*, two objects' replicas are placed in the same node if they are sequentially accessed in one file. In the *frequency-based scheme*, hot objects (e.g., 10 percent of all objects) are replicated in SSD nodes. The *sequence and frequency-based scheme* replicates objects by taking both access sequence and frequency into account.

Fig. 14 reports the FIO read results of different schemes. The evaluation was conducted on all storage nodes with 16 clients. In sequential read benchmark, the sequence-based scheme achieved better performance than the frequency-based scheme as it can achieve more data locality for object access in this pattern. On contrary, the frequency-based scheme has higher bandwidth than the sequence-based



(a) Random read performance for different schemes



(b) Sequential read performance for different schemes

Fig. 14. FIO performance comparison of different schemes.

scheme for placing hot data on SSDs. In both tests, the bandwidth can be further improved by the sequence and frequency-based scheme. Compared with other schemes, the PRS achieved the best performance for a wide spectrum of request sizes. The advantages of PRS are primarily attributed to the design that it considers both I/O correlations and access sequence to replicate objects on heterogeneous nodes and by distinguishing their different characteristics.

4.6 System Overhead and Write Optimization

In our evaluations, the object requests are traced one time, and replications are made in an asynchronous way. With the PRS, the I/O redirection layer needs to check whether the opened object is replicated for object *read*/*write* operations, thus to decide whether to do the offset calculation. To test the overhead caused by the mapping in the redirect table, we run FIO with original data access in Sheepdog, and just create an object redirection table to store each object and look up it before accessing an object.

Fig. 15 shows the performance comparison by FIO between Sheepdog and data access on Sheepdog through an object redirection table (the PRS scheme). As expected, the overhead of looking up an object in the redirection table is negligible. It is because we use a binary search tree for

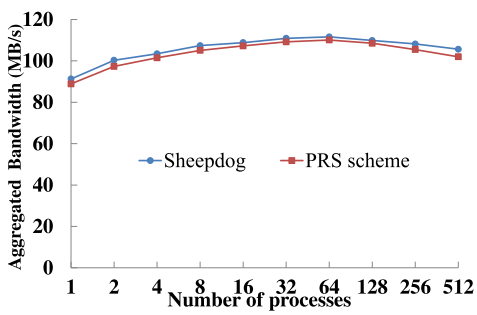


Fig. 15. System overhead for sequential read.

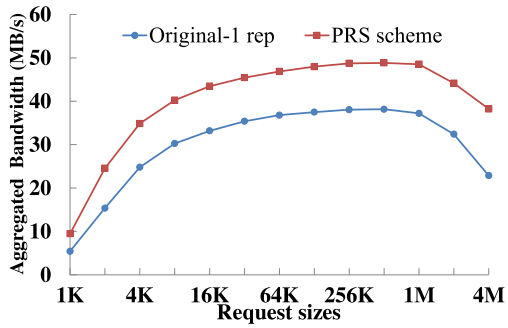


Fig. 16. Write performance with different request sizes.

data structure management in the redirection table, which generally follows the $O(logn)$ asymptotic trend. In some cases, some applications do not have regular data access patterns thus will not benefit from the access to the PRS replication objects. At this time, the overhead may exist, which may affect the performance for applications.

The write optimization is often more complicated in storage systems due to I/O scheduling policy, data consistency, and other factors. The effectiveness is also not evident sometimes due to the existence of write-buffer and asynchronous writes. However, in our tests and evaluations, it was observed that the PRS achieved better aggregated bandwidth over the existing replication schemes due to the fact that it well considers distinct characteristics of heterogeneous devices. We report the results of one set of representative tests, as shown in Fig. 16, for the asynchronous sequential write performance with a total 10 GB file and *sync* flag. The Sheepdog runs on 8 HDD nodes and 2 SSD nodes with only one replica, which is notated as *original-1 rep*. For PRS scheme, the file was created first with one replica and then rewritten, thus updating an additional new replica if it is replicated. It can be seen that the performance improvement was up to 1.75 times, not as significant as the benefits observed in the read tests, partially due to the fact that the write operation needs to be propagated to all replicated copies.

5 Related Work

Numerous studies have been conducted in recent years on data replication schemes and heterogeneous storage systems. We discuss existing work in this section and compare them with this study.

5.1 Data Replication Scheme

Data replication is widely used in many storage systems. The popular object-based storage systems, such as Ceph [3] and Sheepdog [4], provide data replication with deterministic hash mapping functions. The distributed file systems, such as GFS [37] and HDFS [38], divide data sets into blocks of a fixed size and replicate them with rack-aware data placement. To avoid the uneven load distribution across nodes, the dynamic block replication schemes have been proposed with the goal of balancing node load while ensuring node and rack-level reliability requirement [39], [40]. Different from these strategies that depend on replication algorithms to improve data redundancy, our proposed PRS scheme makes data replication according to data-access

patterns, thus can further improve I/O performance while having little impact on data reliability [10].

There are also many I/O optimizations based on data replication in parallel/distributed file systems. Zhang et. al. [12] proposed a data replication scheme to amortize I/O workload to multiple replicas to eliminate I/O interference so that each I/O node only serves requests from one or a limited number of processes. Song et. al. [13] proposed a hybrid replication scheme for complex applications, which consist of different data-access patterns, and choose one replica with the lowest access cost for each data access. Yin et. al. [41] described an I/O data replication scheme to reorganize data according to the data-access pattern for parallel file systems and saves these reorganized replicas with appropriate data layout based on a cost analysis model. Jenkins et. al. [42] proposed a partial data replication system which captures I/O access traces and uses a performance model to evaluate and select optimized data distribution. PFRF [25] selects the most popular files and replicates them to the clusters. All these studies, however, are designed for homogeneous storage systems. In contrast, our proposed PRS scheme provides data replication and placement for heterogeneous storage systems, which can distinguish heterogeneous node performance while achieving data balance for capacity utilization.

Some other replication schemes are designed to dynamically replicate data and reorganize data layout on distributed storage systems [9], [43], [44], [45], [46]. Several approaches are designed to reduce the effect on data integrity and availability when storage node members change [47], [48], [49] or correlated failures occur [50], [51]. Skute [44] determines the most cost-efficient locations of data replicas with respect to their popularity and their client locations to achieve a scalable and highly available key-value store. DARE [9] proposed a distributed adaptive data replication algorithm that aids the scheduler to achieve better data locality. Additionally, recent studies have also applied hybrid methods [52] or coding theory [51], [53] to achieve a trade-off among data consistency, availability, and performance. These studies provide a flexible and adaptive way for data replication. Different from them, our proposed PRS scheme groups objects based on data-access patterns and makes replication for object groups with an optimized layout, which can further alleviate the performance gap between compute and the I/O subsystem in object-based storage systems.

5.2 Heterogeneous Storage Systems

Heterogeneous storage architecture has become increasingly popular for massive data storage in data centers and distributed computing platforms. Most existing data placement algorithms can fairly distribute data in homogeneous storage, but do not meet the requirements well in a heterogeneous environment [34]. Some algorithms address data placement in a heterogeneous environment with considering limited device characteristics. For instance, CRUSH [14] is a pseudo-random algorithm that efficiently and robustly places data across a heterogeneous and tiered storage cluster. However, for each bucket, it essentially distributes data with hashing functions among homogeneous devices. The SPOCA [31], [32] and SUORA [18] algorithms assign segments of the hash space in a number line proportional to nodes' capacity and store data among segments with pseudo-random numbers. Although these algorithms are designed to distribute data in heterogeneous environments, they focus on one device feature (e.g., capacity) while oversighting other characteristics, such as bandwidth or latency of different devices.

There is a rich body of related works on hybrid storage systems [54], [55], [56]. Welch et. al. [57] studied the file size distribution on high performance computing storage systems and proposed to allocate metadata and small files onto SSDs whereas using much cheaper HDDs for large files. Janus [58] is a multi-tiered distributed file system that stores newly created files in the flash tier and moves them to the disk tier using the First-In-First-Out (FIFO) or Least-Recently-Used (LRU) policy. MOBBS [16] provides a hybrid block storage for VMs (virtual machines) by dynamically optimizing data placement between the HDD pool and SSD pool based on real-time workload. Hystor [59] manages both HDDs and SSDs as a single block device that stores data incurring large I/O latencies on SSDs. HAS [60] and HARL [61] present heterogeneity-aware data layout schemes that distribute data across HDD and SSD servers with the consideration of applications' data-access patterns and I/O performance. Our previous work, HiCH [19] and two-mode distribution [17] introduce new variations of the consistent hashing algorithm to manage data distribution in a heterogeneous storage system. However, most of these studies explore heterogeneous node potentials by considering the overall performance difference between them (e.g., partitioning nodes into HDD and SSD storage, respectively) and distribute data on different storage or tier in a separate way.

Different from these efforts, our proposed PRS scheme addresses the challenge by distinguishing heterogeneous device characteristics. The PRS achieves replication with considering both the bandwidth and capacity of heterogeneous devices, and also data-access patterns, to efficiently distribute data with a pseudo-random algorithm in a hybrid or multi-tier storage hierarchy including HDD, SSD, and NVM devices in general.

6 Conclusion

Along with the emergence and rapid adoption of new storage devices, such as solid state drives and non-volatile memory, a heterogeneous storage system that leverages merits of different storage devices has become highly attractive. On the other hand, these heterogeneous storage systems still largely rely on data replication technique to achieve high data availability, reliability, and optimized performance. Existing replication schemes, however, are primarily designed for a homogeneous environment and focus on considering data locations and rack/power/switch redundancy. Yet, they do not well consider distinct storage device features in a heterogeneous environment. Existing schemes do not well consider the impact of applications' access patterns in a heterogeneous environment either.

Motivated by addressing these limitations of existing replication schemes, this paper introduces a new pattern-directed replication scheme to achieve highly efficient data replication in heterogeneous object-based storage systems. The PRS addresses two technical challenges: analyzing object access pattern to guide data replication and optimizing replica layout to take advantages of heterogeneous device

characteristics as we have described in detail in this paper. To verify the potential, we have built a prototype of the PRS in the Sheepdog distributed storage system and carried out extensive tests with various workloads. The experimental results show that the PRS is an effective replication scheme for heterogeneous storage systems. It considers unique features of storage devices in a heterogeneous environment and significantly improves the performance. For instance, it improved the read performance by 105 percent to nearly 10x compared with typical data replication schemes.

While this study is only one step toward making a highly efficient replication scheme for growingly critical heterogeneous storage systems, the pattern-directed replication scheme has shown its promise. Although the evaluations were performed on virtual machines, they are a reasonable representation of what could be expected in a physical system because the PRS provides a general method for data replication. In the future, we plan to continue exploring efficient ways of utilizing devices and applications' characteristics for a better storage solution. We also believe that such a pattern-directed replication scheme has value in a deep storage hierarchy that extends the traditional permanent data store to multiple hierarchies with a variety of non-volatile devices. We plan to continue research investigation along this direction too, and this study also calls for community's collective efforts to address these challenges.

Acknowledgments

Authors would like to thank to the anonymous reviewers for their valuable feedback. This research is supported in part by the National Science Foundation under Grants CNS-1338078, CNS-1362134, CCF-1409946, CCF-1718336, OAC-1835892, and CNS-1817094. This research is also supported in part by the Beijing Municipal Science and Technology Project under Grant Z191100007119002, by the National Science Foundation of China No. 61572377, the Natural Science Foundation of Hubei Province of China No.2017CFC889, and the Fundamental Research Funds for the Central Universities No. 2018QNA5015.

References

[1] M. Mesnier, G. R. Ganger, and E. Riedel, "Object-based storage," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 84–90, Aug. 2003.
[2] P. J. Braam, "The Lustre storage architecture," White Paper, Cluster File System, Inc., Oct. 2003.
[3] 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. USENIX Oper. Syst. Des. Implementation*, 2006, pp. 307–320.
[4] "Sheepdog Project," 2018. [Online]. Available: https://sheepdog.github.io/sheepdog/
[5] H. Wong *et al.*, "Phase change memory," in *Proc. IEEE Conf.*, 2010, pp. 2201–2227.
[6] I. Baek *et al.*, "Highly scalable nonvolatile resistive memory using simple binary oxide driven by asymmetric unipolar voltage pulses," in *Proc. IEEE Int. Electron Devices Meet.*, 2004, pp. 587–590.
[7] "GlusterFS," 2018. [Online]. Available: https://www.gluster.org/.
[8] J. Wang, H. Wu, and R. Wang, "A new reliability model in replication-based big data storage systems," *J. Parallel Distrib. Comput.*, vol. 108, pp. 14–27, 2017.
[9] C. L. Abad, Y. Lu, and R. H. Campbell, "Dare: Adaptive data replication for efficient cluster scheduling," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2011, pp. 159–168.
[10] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2377–2393, Aug. 2016.
[11] S. He and X. Sun, "A cost-effective distribution-aware data replication scheme for parallel I/O systems," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1374–1387, Oct. 2018.
[12] X. Zhang and S. Jiang, "InterferenceRemoval: Removing interference of disk access for MPI programs through data replication," in *Proc. SC Conf.*, 2010, pp. 223–232.
[13] H. Song, Y. Yin, Y. Chen, and X. Sun, "A cost-intelligent application-specific data layout scheme for parallel file systems," in *Proc. Int. Symp. High-Perform. Distrib. Comput.*, 2011, pp. 37–48.
[14] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. SC Conf.*, 2006, pp. 654–663.
[15] R. J. Honicky and E. L. Miller, "Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, 2004, pp. 96.
[16] 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 *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2014, pp. 150–157.
[17] W. Xie, J. Zhou, M. Reyes, J. Nobel, and Y. Chen, "Two-mode data distribution scheme for heterogeneous storage in data centers (short paper)," in *Proc. IEEE Int. Conf. Big Data*, 2015, pp. 327–332.
[18] J. Zhou, W. Xie, J. Noble, K. Echo, and Y. Chen, "SUORA: A scalable and uniform data distribution algorithm for heterogeneous storage systems," in *Proc. IEEE Int. Conf. Netw., Architecture Storage*, 2016, pp. 1–10.
[19] J. Zhou, W. Xie, Q. Gu, and Y. Chen, "Hierarchical consistent hashing for heterogeneous object-based storage," in *Proc. ISPA Conf.*, 2016, pp. 1597–1604.
[20] Y. Yin, S. Byna, H. Song, X. Sun, and R. Thakur, "Boosting application-specific parallel I/O optimization using IOSIG," in *Proc. 12th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2012, pp. 196–203.
[21] J. He *et al.*, "I/O acceleration with pattern detection," in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2013, pp. 25–36.
[22] J. Liu and H. Shen, "A popularity-aware cost-effective replication scheme for high data durability in cloud storage," in *Proc. IEEE Int. Conf. Big Data*, 2016, pp. 384–389.
[23] Z. Li *et al.*, "Time and space-efficient write parallelism in PCM by exploiting data patterns," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1629–1644, Sep. 2017.
[24] J. Zhou, D. Dai, Y. Mao, X. Chen, Y. Zhuang, and Y. Chen, "I/O characteristics discovery in cloud storage systems," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 170–177.
[25] M. Lee, F. Leu, and Y. Chen, "PFRF: An adaptive data replication algorithm based on star-topology data grids," *Future Gener. Comput. Syst.*, vol. 28, no. 7, 2012.
[26] S. Wu, W. Zhu, B. Mao, and K. Li, "PP: Popularity-based proactive data recovery for HDFS RAID systems," *Future Gener. Comput. Syst.*, vol. 86, pp. 1146–1153, 2017.
[27] J. Zhou, W. Xie, D. Dai, and Y. Chen, "Pattern-directed replication scheme for heterogeneous object-based storage," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2017, pp. 645–648.
[28] "FIO Tool," 2015. [Online]. Available: http://freecode.com/projects/fio
[29] Z. Li, Z. Chen, S. M. Srinivasan, and Y. Zhou, "C-Miner: Mining block correlations in storage systems," in *Proc. FAST Conf.*, 2004, pp. 173–186.
[30] P. Xia, D. Feng, H. Jiang, L. Tian, and F. Wang, "FARMER: A novel approach to file access correlation mining and evaluation reference model for optimizing peta-scale file system performance," in *Proc. 17th Int. Symp. High Perform. Distrib. Comput.*, 2008, pp. 185–196.
[31] A. Chawla, B. Reed, K. Juhnke, and G. Syed, "Semantics of caching with SPOCA: A stateless, proportional, optimally-consistent addressing algorithm," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2011, p. 33.
[32] K. I. Ishikawa, "ASURA: Scalable and uniform data distribution algorithm for storage clusters," 2013, *arXiv:1309.7720*.
[33] "Simd-oriented fast mersenne twister," 2017. [Online]. Available: http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/SFMT/index.html
[34] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *Proc. Annu. ACM Symp. Theory Comput.*, 1997, pp. 654–663.
[35] "Zipf distribution," 2017. [Online]. Available: https://en.wikipedia.org/wiki/Zipf's_law
[36] "The file system benchmark," 2018. [Online]. Available: http://sourceforge.net/projects/filebench

[37] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. ACM Symp. Oper. Syst. Princ.*, 2003, pp. 29–43.
[38] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, 2010, pp. 1–10.
[39] G. Ananthanarayanan *et al.*, "Scarlett: Coping with skewed content popularity in MapReduce clusters," in *Proc. Eur. Conf. Comput. Syst.*, 2011, pp. 287–300.
[40] Q. Zhang, S. Zhang, A. Leon-Garcia, and R. Boutaba, "Aurora: Adaptive block replication in distributed file systems," in *Proc. Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 442–451.
[41] Y. Yin, J. Li, J. He, X. Sun, and R. Thakur, "Pattern-direct and layout-aware replication scheme for parallel I/O systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2013, pp. 345–356.
[42] J. Jenkins, X. Zou, H. Tang, D. Kimpe, R. Ross, and N. F. Samatova, "RADAR: Runtime asymmetric data-access driven scientific data replication," in *Proc. Int. Supercomputing Conf.*, 2014, pp. 296–313.
[43] R. S. Chang and H. P. Chang, "A dynamic data replication strategy using access-weights in data grids," *Future Gener. Comput. Syst.*, vol. 45, no. 3, pp. 277–295, 2008.
[44] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 205–216.
[45] Z. Yang, J. Wang, D. Evans, and N. Mi, "AutoReplica: Automatic data replica manager in distributed caching and data processing systems," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf.*, 2016, pp. 1–6.
[46] V. Nagarajan and M. Mohamed, "A prediction-based dynamic replication strategy for data-intensive applications," *Comput. Elect. Eng.*, vol. 57, pp. 281–293, 2017.
[47] A. Brinkmann, S. Effert, F. Meyer, and C. Scheideler, "Dynamic and redundant data placement," in *Proc. 27th Int. Conf. Distrib. Comput. Syst.*, 2007, p. 29.
[48] A. Brinkmann and S. Effert, "Redundant data placement strategies for cluster storage environments," in *Proc. Int. Conf. Principles Distrib. Syst.*, pp. 551–554, 2008.
[49] A. Higai, A. Takefusa, H. Nakada, and M. Oguchi, "A study of effective replica reconstruction schemes at node deletion for HDFS," in *Proc. 14th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2014, pp. 512–521.
[50] A. Cidon, R. Escriva, S. Katti, M. Rosenblum, and E. G. Sirer, "Tiered replication: A cost-effective alternative to full cluster geo-replication," in *Proc. USENIX Annu. Tech. Conf.*, 2015, pp. 31–43.
[51] H. Zhang, M. Dong, and H. Chen, "Efficient and available in-memory KV-Store with hybrid erasure coding and replication," in *Proc. FAST Conf.*, 2016, pp. 167–180.
[52] B. Mao, S. Wu, and H. Jiang, "Improving storage availability in cloud-of-clouds with hybrid redundant data distribution," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 633–642.
[53] R. Li, Y. Hu, and P. Lee, "Enabling efficient and reliable transition from replication to erasure coding for clustered file systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2500–2513, Sep. 2017.
[54] Y. Son, H. Yeom, and H. Han, "Optimizing I/O operations in file systems for fast storage devices," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 1071–1084, Jun. 2017.
[55] S. He, Y. Wang, X. Sun, C. Huang, and C. Xu, "Heterogeneity-aware collective I/O for parallel I/O systems with hybrid HDD/SSD servers," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 1091–1098, Jun. 2017.
[56] E. Kakoulli and H. Herodotou, "OctopusFS: A distributed file system with tiered storage management," in *Proc. ACM Int. Conf. Manage. Data*, 2017, pp. 65–78.
[57] B. Welch and G. Noer, "Optimizing a hybrid SSD/HDD HPC storage system based on file size distributions," in *Proc. IEEE Symp. Mass Storage Syst. Tech.*, 2013, pp. 1–12.
[58] C. Albrecht *et al.*, "Janus: Optimal flash provisioning for cloud storage workloads," in *Proc. USENIX ATC Conf.*, 2013, pp. 91–102.
[59] F. Chen, D. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proc. Int. Conf. Supercomputing*, 2011, pp. 22–32.
[60] S. He, X. Sun, and A. Haider, "HAS: Heterogeneity-aware selective layout scheme for parallel file systems on hybrid servers," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2015, pp. 613–622.
[61] S. He, X. Sun, Y. Wang, A. Kougkas, and A. Haider, "A heterogeneity-aware region-level data layout for hybrid parallel file systems," in *Proc. 44th Int. Conf. Parallel Process.*, 2015, pp. 340–349.

Jiang Zhou received the PhD degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, China in 2014. He is currently an assistant professor with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include file and storage systems, parallel and distributed computing, metadata management, I/O optimization, and cloud computing.

Yong Chen is an associate professor and director of the Data-Intensive Scalable Computing Laboratory, Computer Science Department, Texas Tech University, USA. He is also the site director of the NSF Cloud and Autonomic Computing Center, Texas Tech University. His research interests include data-intensive computing, parallel and distributed computing, high-performance computing, and cloud computing. For more information about him can be found at http://www.myweb.ttu.edu/yonchen/.

Wei Xie received the PhD degree in computer science from Texas Tech University, USA in 2018. He is a technical staff with VMware Inc. His research interests include distributed storage systems, cloud computing, non-volatile memory systems, and checkpointing systems.

Dong Dai received the BS and PhD degrees in computer science from the University of Science and Technology of China, China. He is currently an assistant professor with the Computer Science Department, University of North Carolina at Charlotte. His major research interests include building high-performance storage systems, such as parallel file systems, metadata management systems, and graph storage to facilitate data-intensive applications.

Shuibing He received the PhD degree in computer science and technology from Huazhong University of Science and Technology, in 2009. He is now a ZJU100 young professor with the College of Computer Science and Technology, Zhejiang University. His research interests include parallel I/O systems, file and storage systems, high-performance and distributed computing. He is a member of the IEEE and ACM.

Weiping Wang received the PhD degree in computer science from Harbin Institute of Technology, China, in 2008. He is currently a professor with the Institute of Information Engineering, Chinese Academy of Sciences. His research interests include database and storage systems.