Pattern-Directed Replication Scheme for Heterogeneous Object-based Storage

Jiang Zhou, Wei Xie, Dong Dai, and Yong Chen
Department of Computer Science
Texas Tech University
Lubbock, USA
{*jiang.zhou, wei.xie, dong.dai, yong.chen*}*@ttu.edu*

***Abstract*—Data replication is a key technique to achieve data availability, reliability, and optimized performance in distributed storage systems and data centers. In recent years, with the emergence of new storage devices, heterogeneous object-based storage system, such as a storage system with the co-existence of hard disk drives and solid state drives, have become increasingly attractive as they combine merits of different storage devices to deliver better promise. However, existing data replication schemes do not place data based on heterogeneous device characteristics as well as considering distinct data access patterns. In this paper, we introduce a novel data replication scheme PRS to achieve efficient data replication for heterogeneous storage systems. Different from traditional schemes, the PRS groups objects according to data access patterns and distributes replicas to heterogeneous devices with their features. It uses a pseudo random algorithm to optimize replica layout by considering storage device performance and capacity. The experimental results confirm that PRS is a highly efficient replication scheme for heterogeneous storage systems.**

***Keywords*-data replication; heterogeneous storage; object-based storage; access pattern; data distribution**

I. 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. It has been widely adopted in production systems like Lustre [1], Ceph [2], and Sheepdog [3]. Recently, *heterogeneous* object storage systems that take advantages of emerged new devices, such as storage class memory (SCM) including solid state drives (SSDs), phase change memory (PCM) [4], in addition to conventional hard disk drives (HDDs), have gained increasing attention[5, 6]. These storage systems leverage different storage devices and combine 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. Its core concept is to automatically replicate data objects and distribute them into multiple devices. With replication, data can be retrieved from other replicas if one copy is not accessible or is corrupted. It improves the data redundancy and reduces data loss in case of failures. Replication can also be used to improve I/O performance, such as by coordinating clients to access a local and closer data object copy or amortizing I/O workload and achieve load balance among multiple copies.

Although numerous studies have been devoted to the design and development of data replication schemes, there has been little work on data replications for heterogeneous object-based storage systems. As the performance and capacity of heterogeneous devices are different, it is desired to place replicas according to their characteristics. Existing replication strategies mainly use frequency or sequentiality to work with heterogeneous storage, in which they place replicas of hot, frequently accessed data on SSDs while the cold, sequentially accessed data on HDDs. However, this does not consider the object access patterns well. For instance, two strongly correlated objects that are normally accessed successively in term of time might be placed into HDD and SSD, respectively, which leads to poor performance when this same access pattern appears again in the future. Even if one replica of the correlated objects are placed together in SSDs, it is up to the policy that selects the replicas from reading data that determine the read performance. Besides, it is difficult to determine the sequentially accessed objects because each object is a separate file on the disk. In this study, in addition to considering object accesses frequency, we further analyze the data access patterns and reorganize temporally related objects into an object group to avoid placing them separately. Furthermore, grouping objects with temporal locality also improves the read performance since it improves data caching and increases 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 considers applications' access pattern and distributes replicas according to heterogeneous device characteristics. It groups objects based on their *temporal locality* or *access frequency* and reorganizes them for replication. A new object will be created as the replica for object group to reduce I/O access times and benefit the spatial locality. To place replicas on heterogeneous devices, a pseudo random distribution algorithm is used to optimize replica layout by considering different device bandwidth while keeping capacity utilization fairness. The storage system can achieve improved I/O performance for future data accesses with the replications while exploiting

the full potentials of heterogeneous devices.

II. Pattern-Directed Replication Scheme

In this section, we introduce the design and implementation of pattern-directed replication scheme. The replication scheme is optimized for object storage in virtual machine (VM), but it can also be used in general object-based storage systems.

A. Architecture Overview

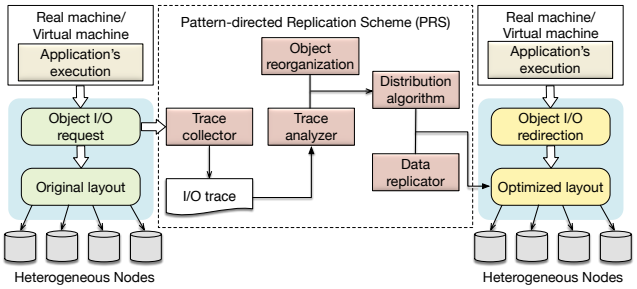


Figure 1: Overview of a pattern-directed replication scheme

The fundamental idea of the PRS is to consider different storage device features and adopt a *pattern-directed* method to reorganize objects based on identified access patterns among objects for replication in a heterogeneous storage system. First, as shown in Figure 1, the application running on physical machines or virtual machines (VMs) send I/O requests to the backend storage system. The data objects are placed with an *original data layout for its first and second replicas*, e.g., via the consistent hashing algorithm to distribute data [7]. Second, *the rest replicas for an object will be created with PRS after pattern analysis*. The data access pattern analysis (can be performed periodically in real time or off-line) is based on the runtime object I/O requests that are traced by a trace collector in each storage node. It makes object replications based on identified access patterns for making the third or more replications, and places them with a replica distribution algorithm in the background. Third, when the same application runs again, the I/O requests will be redirected to the new copy in an optimized layout for better performance if the object has been replicated with PRS.

B. I/O Trace Collection

An I/O trace collector is responsible for collecting the runtime object information of object-based storage. While there are some techniques and tools that can be used for trace profiling, we directly use code library to capture the object I/O in the object-based storage system in this study. We collect five aspects of object request action for further pattern analysis: 1) object ID, 2) start offset in an object, 3) size of access, 4) access time, 5) operation code. The object ID is an unique identifier that is used to find an object. Start offset and size of access are respectively the position and length when reading or writing data in an object. The access time indicates the object access pattern, whereas objects are accessed sequentially, random or iteratively, or the same object is accessed repeatedly. Operation code is classified into read only, write only, and read and write.

During application execution, we can get the object I/O trace. In a distributed storage system, each storage node generates one trace file that contains all its I/O requests in which the node receives or forwards. The trace file is used for data access pattern analysis and object reorganization for replication in the node. The original objects resident on one node may be replicated in other nodes. It does not affect data access as the I/O redirection layer can find the replica in the new optimal layout.

C. Data Access Pattern Analysis

As mentioned above, our PRS scheme tends to provide data replication for VMs storage. When the VMs are stored in backend storage system, they are essentially converted into read/write operations on virtual disk image (VDI) files. Figure 2 shows a typical data placement for VMs using object-based storage. In the figure, the 40GB VDI is divided into fixed objects with 4MB size when storing. For each object, there is an object ID corresponding with it, namely *object0*, *object1*, *object2*, and etc. The objects and their replicas are saved as dedicated files in local file systems of heterogeneous nodes for persistence. Supposing the replica number is 3, it can be seen that *object0* is placed in three nodes which are equipped with HDD and SSD devices, respectively.

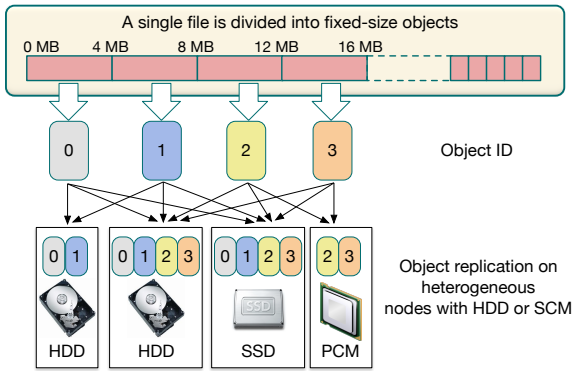


Figure 2: Object division and placement for VMs

Although the objects are distributed in different nodes for replication, there still exist some limitations. First, the division of a large VDI file into objects will result in numerous small files if the object size is small. Modern file systems deliver good performance for large files, but lack efficient support for large numbers of small files. Second, frequent object I/O requests will affect the performance for

VM applications. For example, if there are lots of objects in the HDD devices, the object access latency will be affected for mechanical disks addressing. Third, current replication schemes lack efficient ways to distinguish heterogeneous device characteristics. The frequent accessed objects may be placed in HDD devices with low throughputs while a large amount of cold data occupies valuable space in SSD and other devices.

Different from environments in HPC, there is no fixed data access pattern for applications in cloud computing. We use the correlation among object access pattern to guide data replication. The main idea is to group multiple objects with I/O access correlation into one object and make replication for it.

In an object-based storage system, a file is divided into multiple fixed-size objects. To find the object access behavior, 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 in a period of time. The I/O requests may come from different processes or applications, but they will be eventually converted to object requests on underlying storage. Objects that are accessed together in a given interval and accessed periodically belong to this pattern.

The *global data access pattern* means the objects' behavior is no longer limited to data access in a period of time. Instead, they represent the objects that are most frequently accessed during the application run time. We adopt an *configuration-aware* method to find the hot objects that belong to the global pattern, which means a certain percentage of objects will be considered as hot data according to the architecture configuration of heterogeneous storage systems. For example, in an environment with HDDs and SSDs in which the capacity ratio of HDDs and SSDs is *9:1*, then *10* percent of objects are selected as hot data for the SSDs proportion. The frequently accessed objects in the global pattern have higher priority to be placed in the devices with higher performance, such as SSDs, with our replica distribution algorithm (discussed in detail below). In many situations, local patterns alone cannot reveal the behavior of applications, and a global view is necessary to improve the efficiency of replication.

To explore data access patterns, the method of block distance calculation is widely used in local storage system. In a distributed object-based storage system, we use the object distance calculation between each two objects to identify data access patterns. We further use the object classification algorithm to group objects based on the object distance. To decide which objects belong to the same group, we use a similar strategy as in our previous work used to classify objects into dynamic groups [8]. To avoid the size of one object group is too large, we limit the object number to a certain value in each group.

D. Pattern-Directed Replication Scheme

With the object clustering algorithm, the PRS groups objects based on the local pattern or global one. The objects in the same group are grouped to a new object and replicated. The PRS only replicates the accessed data, which are the objects that have ever been read or written.

The combination granularity is object, which means that the entire object will be replicated even if only part of its data is accessed. For the identified data access pattern, the PRS first replicates objects with a global pattern. That means a proportion of hot objects will first be grouped for replication. The objects that do not belong to any global pattern will be grouped for identified local patterns. The number of replicas can be preset, which reflects the redundancy of objects in the storage system.

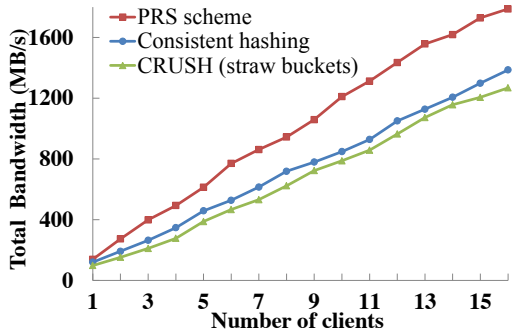
When generating replicas for objects with local or global patterns, how to place replicas according to different device characteristics is an important issue. In order to achieve a very large scale, the existing data placement algorithms like consistent hash [7] and CRUSH [2] are able to distribute data across the storage cluster with non-centralized management. They achieve load-balance by setting the statistical probability and reduce data migration when node adding or moving. However, a heterogeneous storage environment is becoming the norm in large-scale storage systems, where existing data placement algorithms lack full consideration of different device characteristics. To solve this problem, we adopt a pseudo random number algorithm to optimize replica placement in a heterogeneous environment [9].

The PRS first assigns heterogeneous nodes to different segments in a number line. Each node is assigned to one or more segments considering its average bandwidth. After the segment assignment of nodes, the data is then mapped to one segment by pseudo random hash functions. We choose the pseudo random number mapping because it can distribute data proportional to the segment length. A random number sequence is generated according to the data ID until it fits the range of one segment. If the number of replications is m, there is m random numbers generated for mapping m segments in the random number sequqnce. In this way, the data are mapped to heterogeneous nodes with pseudo random hash functions according to device bandwidth. As a node may contain more data for higher performance, the replicas will be placed on other nodes if one or more storage nodes have no sufficient storage space for load balance.

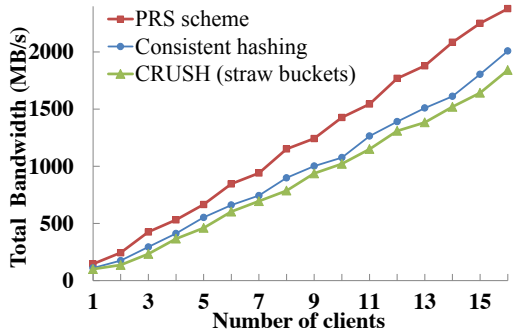
III. Evaluation

In this section, we present the evaluation results of the proposed PRS based the Sheepdog [3] storage system. The experiments were conducted on a local 32-node cluster, including 16 storage nodes (with half HDD nodes and half SSD nodes) and 16 client nodes hosting VMs.

Each storage node in Sheepdog can be used as a *gateway* node to receive I/O requests from clients, where PRS



(a) Random read performance



(b) Sequential read performance

Figure 3: FIO performance comparison with multiple clients

analyzes the trace of the first replica for objects and creates redirection table for replications. In our tests, we first run the application for one time to collect I/O trace. Then the objects with identified data access patterns are replicated asynchronously, thus the application can benefit from the PRS for replica access in the future runs. We use the SIMD-oriented Fast Mersenne Twister (SFMT) [10] algorithm to generate pseudo random numbers to optimize replication placement. We compare the PRS with the data replication scheme based on consistent hashing algorithm [3] and straw buckets(a typical replication algorithm in CRUSH [2]) in Sheepdog, with each one making 3 replicas.

We use multiple clients running on different nodes to perform read operations and get the total result by adding each bandwidth of them. 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 $100MB$ in an asynchronous way with the request size of $256KB$.

From Figure 3, 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 different device characteristics well.

IV. Conclusion

This paper introduces a novel pattern-directed replication scheme (PRS) to achieve highly efficient data replication in heterogeneous object-based storage systems. The promise of the PRS relies on addressing two technical challenges: analyzing object access pattern to guide data replication and optimizing replica layout to take full advantage of heterogeneous device characteristics as we have descried in detail in this paper. To verify the potential, we have built a prototype of the PRS within the Sheepdog distributed storage system and carried out evaluation tests. The experimental results show that the PRS is an effective replication scheme for heterogeneous storage systems. It well considers unique features of storage devices in a heterogeneous environment and significantly improves the performance.

Acknowledgment

We are thankful to the anonymous reviewers for their feedback. This research is supported in part by the National Science Foundation under grant CNS-1162488, CNS-1338078, IIP-1362134, and CCF-1409946.

References

[1] P. J. Braam, "The Lustre storage architecture," White Paper, Cluster File System, Inc., Oct. 2003.

[2] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of OSDI*, 2006, pp. 307–320.

[3] "Sheepdog Project," 2015. [Online]. Available: http://www.sheepdog-project.org/.

[4] H. Kim, S. Seshadri, C. L. Dickey, and L. Chiu, "Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches," in *Proc. of the 12th USENIX Conference on FAST*, 2014, pp. 33–45.

[5] J. Zhou, W. Xie, Q. Gu, and Y. Chen, "Hierarchical consistent hashing for heterogeneous object-based storage," in *Proc. of the 14th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA'16)*, 2016.

[6] 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. of the 11th IEEE International Conference on Networking, Architecture, and Storage (NAS'16)*, 2016.

[7] 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. of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, 1997, pp. 654–663.

[8] D. Dai, Y. Chen, D. Kimpe, and R. Ross, "Provenance-based object storage prediction scheme for scientific big data applications," in *Proceedings of The 2014 IEEE International Conference on Big Data*, 2014, pp. 271–280.

[9] A. Chawla, B. Reed, K. Juhnke, and G. Syed, "Semantics of caching with SPOCA: A stateless, proportional, optimally-consistent addressing algorithm," in *Proc. of the 2011 USENIX ATC Conference*, 2011.

[10] "Simd-oriented fast mersenne twister." [Online]. Available: http://www.math.sci.hiroshima-u.ac.jp/ m-mat/MT/SFMT/index.html.