SmartDedup: Efficient and Reliable Deduplication on Object Storage System

Yichen Jia
Louisiana State University
yjia@csc.lsu.edu

Abstract

Deduplicated storage is widely used to reduce storage cost by eliminating redundant data. However, block-based or file-based storage system suffers from semantic gap between applications and storage devices, which make quality of service requirement, reliability and overhead control issues hard to handle. In order to overcome all these three challenges while maintaining high deduplication ratio, we propose a deduplication mechanism at host side on object-based storage system, called SmartDedup, to achieve high performance and reliability. By getting the performance demand, reliability requirement, data type of each object from the applications, SmartDedup can schedule and manage each object in an more efficient and safe way. Specifically, applications assign priorities to each object according to the performance demand and SmartDedup will process the I/O accordingly. Meanwhile, duplicate chunks will be kept in storage without conducting the deduplication operation if they need a higher reliability guarantee. Moreover, in order to reduce the indexing overhead and improve the duplication efficiency, we proposed to check the duplication only if the objects share the same data type since most of the data duplication comes from objects with same data type. By combining these three techniques, the reliability of the system can be significantly improved with little deduplication ratio loss. Finally, we design a thorough experiment plan to measure the effectiveness and analyze the overhead of our proposed methods.

1. INTRODUCTION

Data deduplicated is widely used to reduce storage cost by eliminating redundant data [19, 13, 33] at block or file level storage systems. Since there are numerous sources of duplicate, such as application-level versioning of records, inclusion relationship between records, frequent backups and VM migration, deduplication technique is quite effective to save storage cost. However, deduplication storage systems also raise performance and reliability concerns of the users, namely *quality of services*, *reliability* and *overhead control*, because of lack of semantic information from the application. We will discuss these three critical issues in details respectively.

Quality of Services While deduplication can reduce the cost of storage, a storage system must also meet the quality of service (QoS) requirements for a client. There is an additional levels of indirection to map from a fingerprint to data chunk location in the deduplication storage system and deduplication tends to turn sequentially written content into references to chunks scattered across the HDDs. It is difficult to predict and guarantee the the time needed to finish one I/O request because of the complicated data layout in the physical devices. Besides client initiated I/O, d the resource-intensive background tasks also need QoS control in the deduplicated storage systems, such as garbage collection process that determines which data chunks are referenced from live files versus which chunks are to be freed. There may be opportunities to reorder the processing of data to serve both background tasks and client I/Os.

Reliability An area that needs to pay more attention to is the complex relationship between deduplication and data reliability. By reliability, we mean the possibility that corresponding data chunks will be available in the storage system when requested. One of straightforward methods to guarantee the data reliability is to create multiple copies to protect against data loss, while deduplication removes redundancies. In particular, backup and archival systems, designed to protect data, were among the earliest adopters of deduplication because of the inherent redundancy. However, users often question the risk of data loss because of the inherent nature of deduplication.

Overhead The overhead of deduplication storage system mainly comes from indexing and on-disk layout. Identifying duplicated data chunks is highly related to the duplication quality and space saving. Moreover, since physical data chunks of the same file may be scattered in the disk after deduplication, the performance may degrade on the hard disk based storage system. Eliminating the indexing and data layout overhead introduced by the deduplication mechanism is very important for modern storage system.

All these issues on block or file storage in practices motivate us design an alternative deduplication system to provide high performance and high reliable services. Though popular, the block-based and file-based storage interfaces are difficult to optimize for the increasingly complex deduplication storage system without the semantic information. On the contrary, object-based storage, which essentially bundles the data itself along with metadata tags and a unique identi-

fier, provides opportunities to remove redundant data while meeting QoS, reliability and overhead requirements. Since the metadata is highly customizable in object-based storage system, we can easily input more identifying information for each piece of data for further management in deduplicated storage system.

In this paper, we present a deduplicated object-based storage system, called SmartDedup, to fundamentally address these issues by taking advantage of the semantic information of each object. By classifying objects into different classes at the application level according to their performance and reliability requirements, object-based storage device can schedule the I/O requests and manage the data layout accordingly. Specifically, I/O requests are grouped into two categories: latency intensive and throughput intensive ones. Latency intensive I/O requests will be served at higher priority to meet the SLA. On the contrary, throughput intensive I/O requests are batched together to achieve the maximum throughput. In order to guarantee the reliability of the storage system, objects are classified as critical and normal classes. Critical objects will maintain duplicated data chunks and normal objects will remove duplicated copies in the device. To reduce the matching overhead incurred by the large volume of objects, we propose to search duplicated data that share the same data type. Combining all these three techniques, SmartDedup may meet both performance and reliability requirements with minimal overhead.

Our contributions will cover the following aspects:

(1) To our best knowledge, SmartDedup is the first work to implement deduplication on object-based storage system, achieving high performance and reliability at the same time.

(2) SmartDedup identifies the critical objects with the help of applications and achieves high reliability by eliminating deduplication operation for the critical objects.

(3) SmartDedup can meet the QoS requirements of clients by scheduling latency-intensive I/O requests at high priority and throughput-intensive ones at low priority.

(4) SmartDedup can effectively reduce the matching overhead by constraining deduplication operation within objects those share the same data type.

The rest of paper is organized as follows. Section 2 gives the background. Section 3 introduces the design of SlimCache. Section 4 shows our experimental plan. Section 5 gives a brief discussion about our proposed mechanism. Related work is presented in Section 6. The final section concludes this paper.

2. BACKGROUND

In this section, we briefly introduce data deduplication technique and object-based storage system. The difference between block, file and object interfaces has motivated us to conduct a comprehensive deduplication system on object-based storage system, to achieve high performance and high reliability at the same time.

2.1 Data Deduplication

Data deduplication is a popular data reduction technique [39, 35, 24, 31, 13, 5] for eliminating duplicate copies of redundant data, as shown in Figure 1. This technique is used in both storage system and network data transfers to improve the storage utilization and reduce the network traffic. In the deduplication process, chunks of data, or byte patterns, are first identified as unique or not. Unique data chunk will

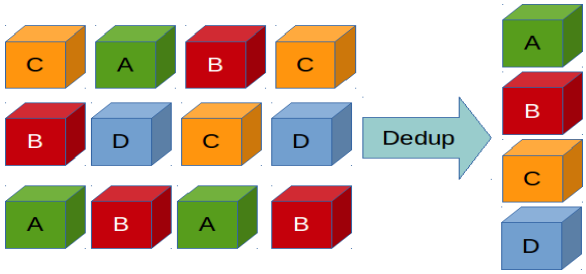

Figure 1: Data Deduplication

be stored while redundant chunk is replaced with a small reference that points to the stored chunk. Given that the same data chunk may occur dozens, hundreds, or even thousands of times, the amount of data that must be stored in storage system or transferred through the network can be significantly reduced.

Data deduplication systems can be classified into two groups, namely, primary and backup deduplication systems. Primary data deduplication is implemented in a native file system on a host or a cluster supporting deduplication operations. Data stored in primary deduplication systems, such as of emails, multimedia, and databases, are frequently accessed by users in a random fashion. On the contrary, data in the deduplicated backup or archive system, whose capacity is typically large ,is rarely reaccessed. Thus, it requires high I/O throughput, rather than low latency, when dealing with the incoming traffic or restoring the entire dataset. Whenever data is deduplicated, concerns arise about potential loss of data. By definition, in data deduplication systems, multiple files in the logical layer may share one physical data chunk in the psychical layer. . As a result, users are concerned with the integrity of their data. Scaling is also challenging for deduplication systems because the scope of deduplication needs to be shared across storage devices. If there are multiple disk backup devices in an infrastructure with discrete deduplication, then space efficiency is adversely affected. A deduplication shared across devices preserves space efficiency, but is technically challenging from a reliability and performance perspective.

2.2 Storage Interfaces

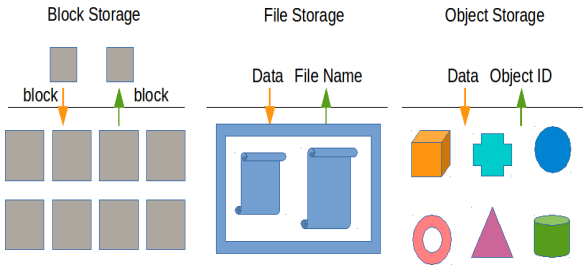

Figure 2: Storage Interfaces

Block storage is a popular and simple form of data storage. As shown in Figure 2, traditional block storage interface uses a series of fixed size blocks which are numbered starting at zero. Data is stored in fixed-sized chunks called blocks, which is identified by its logical block number (LBN). Later, users can retrieve that block of data by specifying its unique

LBN. Although block level storage is extremely flexible, a highly available and accessible file system still remains necessary in many applications. In typical files systems [7, 6, 30], files are organized into a hierarchy, with directories and sub-directories. Each file also has a limited set of metadata associated with it, such as the file name, creation date, and last modification date, etc. Differently, object-based storage is a data storage architecture that manages data as objects. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier. Even though block storage and file system are popular and well supported, they have limited metadata about the data pieces they're storing. On the contrary, the metadata tags are a key advantage with object storage, thus they allow for much better identification and classification of data.

2.3 Object-based Storage System

Object storage explicitly separates file metadata from data to support additional capabilities. As opposed to fixed metadata in file systems (filename, creation date, type, etc.), object storage provides for full function, custom, object-level metadata in order to capture application-specific or user-specific information for better indexing purposes and support data-management policies (e.g. a policy to drive object movement from one storage tier to another). The command interface includes commands to create and delete objects, write bytes and read bytes to and from individual objects, and to set and get attributes on objects. Object storage can be implemented at multiple levels, including the device level (object-storage device), the system level, and the driver level. Object-based storage is widely used as archive storage. EMC Centera [5] and Hitachi HCP [8] are two popular object storage products for archiving. Meanwhile, the vast majority of cloud storage services available in the market,such as Amazon Web Services S3 [2] and IBM Cloud [10], leverage an object-storage architecture.

3. DESIGN

In this section, we present a comprehensive object-based deduplication storage system, which provides high deduplication ratio and high reliability guarantee at the same time, while minimizing the deduplication overhead.

3.1 Overview

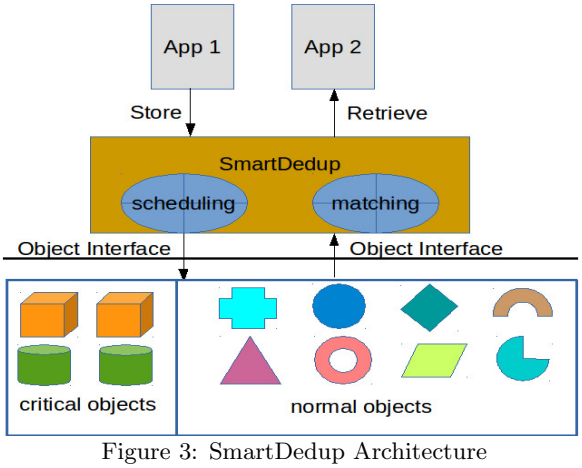

Figure 3: SmartDedup Architecture

In order to provide an efficient and reliable deduplication storage system, we propose a thorough design, called SmartDedup, which makes use of the semantic information provided by the applications to response to the requests based on their performance demand and selectively remove the redundant data chunks according to their reliability requirements. Meanwhile, SmartDedup proposes to reduce the matching overhead by conducting deduplication among the objects with the same data type. As shown in Figure 3, application is responsible to label the objects according to their semantic meanings. SmartDedup, which is running at the driver level, will identify and schedule the requests based on the our proposed policies.

3.2 Selective Deduplication

Deduplication storage system removes the redundant objects to save space. In other words, multiple objects may share the same physical data chunks on the device. Thus the physical level data loss may cause multiple objects unreadable. However, objects stored in the storage system have different attributes, and their impact to the data reliability differs a lot. As Figure 4 shows, objects are grouped into two classes: critical objects and normal objects. Specifically, on one hand, critical objects include all the metadata that are related to the system management, frequently accessed data that are essential to the system performance, and other critical data defined by the application. On the other hand, temporary data, regenerated data, frequently updated data and rarely accessed data are categorized as normal objects. They can be either regenerated or rewritten in a short time. Data loss happens on them has minimal impact compared with critical objects. We propose to conduct deduplication operation only on normal objects, while skipping the critical objects to improve the system reliability. By selectively applying deduplication on the objects, we guarantee that the system maintains a high reliability since all the objects that are essential to the system correctness or performance are kept in the original format, without removing the redundant copy in the physical layer. We only need to add a 1-bit attribute which is called *reliability* to the metadata of the each object. In this way, we can make sure that all the essential objects which have high reliability requirements are much more reliable than the traditional deduplication systems.

3.3 QoS based Scheduling

Requests of the application may have different QoS requirements. SmartDedup proposes to meet the QoS requirements by scheduling requests with different policies according to the service level agreement(SLA). As Figure 5 shows, application is involved to label the requests or objects as latency-sensitive or throughput-sensitive ones. SmartDedup assigns high priority to the latency sensitive requests and low priority to the throughput sensitive requests. The waiting queue is divided into two queues. The first one is to store the latency-sensitive requests while the second one is to store the throughput-sensitive requests. The latency-sensitive requests are sent to the device immediately after they are issued to guarantee these requests finished following SLAs. On the contrary, the throughput-sensitive requests are sent the the device after a period of time to make sure the scheduler can accumulate enough requests issued to the device. We study the effect of time threshold in Section 4. In this way, we can meet QoS requirements for the latency-sensitive

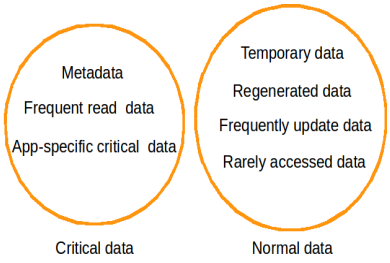

Figure 4: Data Classification

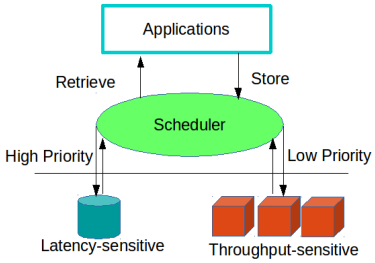

Figure 5: Scheduler

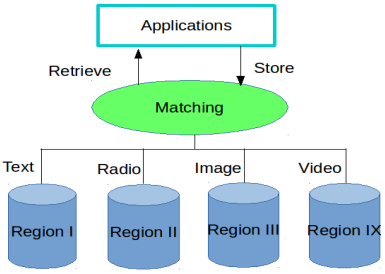

Figure 6: Matching

requests and throughput-sensitive requests at the same time. We only need to add a 1-bit attribute which is called *sensitivity* to the metadata of the each object. It is the responsibility of application to label it as latency-sensitive or throughput-sensitive according to its semantic information.

3.4 Type-based Matching

One of the overhead of deduplication storage system is the overhead of finding the matching fingerprint as the storage system becomes increasingly large. Since the objects that have the same data type are more likely to be the same with each other, SmartDedup introduces a type-based matching policy to reduce the matching overhead. As Figure 6 shows, all the objects are grouped into four categories based on their data types: text, radio, video and images. The fingerprints of the objects with the same data type are stored separately. When the new object comes, SmartDedup only check the hash table that indexes the objects which have the same data type. If the same fingerprint is found, that means the same object is found in the devices. If the newly incoming object is not critical one, SmartDedup will conduct deduplication operation on it. Otherwise, SmartDedup will skip the deduplication operation and store the new objects as normal objects. This type-based matching is highly efficient for two reasons. First, the searching range can be reduced to the objects that share the same type, which reduces the searching time. Second, since most of the duplicated objects share the same data type, the deduplication ratio will remain similar rather than the searching the global hash table. We only need to add a 2-bit attribute which is called *type* to the metadata of the each object. In this way, we can reduce the matching overhead significantly while maintaining high deduplication efficiency.

4. EXPERIMENTAL PLAN

In this section, we propose our experimental plan to evaluate the effect of SmartDedup.

(1) **Data classification.** we test the deduplication ratio changes with and without data classification. Since we skip the deduplication operation for the critical data, the deduplication efficiency is expected to decrease. However, as the critical data is only a small portion of the dataset, the space saving is expected to be close to that without data classification.

(2) **Scheduler.** We measure the average latency and average throughput of the requests with and without priority. Since latency-sensitive requests are assigned high priority and are scheduled to finish immediately after issued, the average latency of them is expected to decrease. On the contrary, since throughput-sensitive requests waits for a period of time to accumulate enough data for better scheduling, the throughput is expected to increase. In this way, we may meet the QoS requirements of the clients by labeling requests according to their requirements.

(3) **Matching.** We measure the lookup time and deduplication ratio changes of the each request with and without our proposed type-based matching. We expect the lookup time will decrease dramatically, however, the deduplication ratio will keep almost the same with the global hash table, since most of the deduplication comes from objects those sharing the same data type.

(5) **Overhead analysis.** We measure the memory or storage overhead for the extra semantic information in the metadata of the objects. Also, we may compare the workload of programmers, when passing information through a file system interface and object interface.

(6) **Multiple applications.** We test the performance when running multiple applications, which have different data reliability, performance requirements and data types, on top of SmartDedup. Since the storage device is becoming huge, sharing the underlying device among multiple applications will be a typical practice in real world.

(6) **Multiple interfaces.** We tend to make a thorough comparison between the deduplication on block-based, file-based and object-based storage system, which includes the deduplication ratio, the latency, the throughput, and reliability guarantee, etc.

5. DISCUSSIONS

In this section, we discuss some major concerns of users when deploying deduplication on the object-based storage systems.

5.1 Application involvement

SmartDedup is a comprehensive deduplication storage system that may need the involvement of applications. Compared with block or file based storage system, the additional information about the objects is much easier to manage, taking advantage of the object API. Together with the data itself, objects always keep an expandable amount of metadata, which contains contextual information about what the data is, what it should be used for, its confidentiality, or anything else that is relevant to the way in which the data is used. So the critical identification, priority assignment and data type classification on the object storage system are all natural choices that involve little overhead.

5.2 Performance concerns

Typically, block interface provides a relatively low overhead than the object interface. However, for the primary

storage system, our proposed scheduling mechanism will make sure the latency-sensitive requests finished with high priority, which will make sure the latency is acceptable by the application. Moreover, for the backup storage system, throughput, instead of latency, is the main concern. The throughput-sensitive requests are grouped together to expect more optimization opportunities in the OSD device. So SmartDedup can provide both low-latency and high-throughput, taking advantage of semantic information from applications and object API.

5.3 Combination with other techniques

Deduplication works well with other data reduction techniques such as compression and delta encoding [4]. In the object based storage systems, compression can be easily combined with deduplication to reduce storage cost further. However, delta encoding may not be suitable in this case, that is because all the data are stored as objects and indexed by the object ID. The original data chunk and delta (different part) are both treated as object, which make the indexing much more complicated and the overhead more pronounced. By indexing the newly updated object, user has to load the original object and each delta (stored as object) in a sequence, which makes the loading overhead too large. This combination may be acceptable in the backup or archive storage systems, where loading operation is quite rare. We will look into the optimization opportunities of combining deduplication and delta compression in the future work.

6. RELATED WORK

The data deduplication and object-based storage system are both extensively researched in academia [35, 31, 20, 27, 32, 26] and widely used in industry [9, 22, 5, 13, 12, 14]. We summarize the most related work in these two fields in this section.

6.1 Data Deduplication

Data deduplication has received recent interest in academia and industry. The related work in the area of data deduplication was mostly about data chunking, primary storage and backup storage.

Data chunking: Deduplication systems differ in the granularity at which they detect duplicate data. Different from Microsoft Storage Server [18] and EMC Centera [5]. those use file level deduplication, LBFS [32] uses variable sized data chunks obtained by Rabin fingerprinting [34] and Venti [33] uses individual fixed size disk blocks. Two-Threshold Two-Divisor (TTTD) [25] and bi-modal chunking algorithm [29] also produce variable-sized chunks.

Primary data deduplication: VMware ESX Server deploys a decentralized host-driven block-level deduplication system designed for SAN clustered file systems [21]. Inline block-level deduplication is widely deployed in primary storage products, such as Sun ZFS [19], Linux SDFS [12], NetApp [13], Ocarina [14] and Permabit [15] iDedup [35] is a primary data deduplication system that uses inline deduplication that tries to improve the performance by exploiting spatial locality to duplicate data sequence that are contiguously stored on a disk to reduce disk fragmentation. El-Shimi et al. conduct a large scale study for the deduplication system [24].

Backup data deduplication: A secondary storage system, facilitating data backup and restore operations, requires high I/O throughput. Zhu et al. describe an inline backup data deduplication system that improves the performance by a bloom filter [20] and index prefetching [39]. Sparse Indexing [31] efficiently reduce in-memory index size with sparse indexing by sacrificing deduplication quality. HYDRAstor [23] is content addressable and a global data deduplication system which serves as a distributed backup storage system for NEC primary data storage solutions.

6.2 Object Storage System

Numerous studies have been done in object-based storage systems. Network-Attached Secure Disk (NASD) [26] is the primary work that defines the specifications of standards [17] for object-based storage. NASD introduces variable-length objects with attributes and moves data management to the storage disks by alleviating information from operating system. The Panasas [38] and Lustre [22] and Parallel Virtual File System [28] are built on object-based storage devices. Goodell et al. extended the POSIX API by organizing the storage around data objects in order to map complex data structures to these data objects and have direct access between the data objects and applications [27]. Datamods [36] is a framework that exploits existing large-scale storage system services to support complex data models and interfaces. Datamods avoids duplicating services already provided in distributed storage systems in middleware and improves scalability since it is not limited to a single dimension at the file level. Rados Gateway [11] is an object storage service forming the foundation of Ceph [37]. It provides the clients a single logical object store and offloads object replication, failure detection, and data management tasks to the underlying object store daemons. Object storage is widely used in cloud cloud services, such as Amazon S3 [2] and IBM cloud [10] and popular data stores, such as Amazon SimpleDB [3], Amazon DynamoDB [1], Redis [16], and Hyperdex [9].

7. CONCLUSIONS

Data duplication is an effective approach to reduce the storage cost by eliminating redundant data chunks. Although significant research has gone into improving the deduplication efficiency at block and file level, the impact of deduplication on object storage (and associated APIs) has not received much attention.

In this paper, we present an efficient and reliable deduplication mechanism on the object-based storage system, called SmartDedup, to guarantee the quality of service and reliability while minimizing the overhead. Compared with block or file based storage system, object-based storage system provides the opportunities to distinguish the objects by simply changing their attributes defined in the metadata. By labeling the objects as critical or normal, latency-sensitive or throughput-sensitive objects with different data types, we can achieve high performance and high reliability at the same time. We design a thorough experimental plan to measure the effect of our proposed mechanisms. We expect the deduplication ratio keeps high while maintaining the high reliability of the system and meeting the QoS requirements of the clients. We prove that SmartDedup can support multiple applications with different performance and reliability requirements.

8. REFERENCES

[1] Amazon dynamodb. https://aws.amazon.com/dynamodb/.

[2] Amazon s3. https://aws.amazon.com/s3/.

[3] Amazon simpledb. https://aws.amazon.com/simpledb/.

[4] Delta encoding. https://en.wikipedia.org/wiki/Delta_encoding.

[5] Emc corporation. https://germany.emc.com/collateral/hardware/data-sheet/c931-emc-centera-cas-ds.pdf.

[6] Ext3. https://en.wikipedia.org/wiki/Ext3.

[7] Ext4. https://kernelnewbies.org/Ext4.

[8] Hitachi cp. https://www.hitachivantara.com/en-us/products/cloud-object-platform/content-platform.html.

[9] Hyperdex. http://hyperdex.org/.

[10] Ibm cloud. https://www.ibm.com/cloud/.

[11] Librados api documentation. http://ceph.com/docs/master/api/librados/.

[12] Linux sdfs. www.opendedup.org.

[13] Netapp deduplication and compression. www.netapp.com/us/products/platform-os/dedupe.html.

[14] Ocarina networks. www.ocarinanetworks.com.

[15] Permabit data optimization. www.permabit.com.

[16] Redis. https://redis.io/.

[17] T10 technical committee of the international committee on information technology standards, âĂIJobject-based storage devices - 3 (osd-3). http://www.t10.org/.

[18] Windows storage server. technet.microsoft.com/en-us/library/gg232683(WS.10).aspx.

[19] Zfs deduplication. blogs.oracle.com/bonwick/entry/zfs_dedup.

[20] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Network applications of bloom filters: A survey. In *Internet Mathematics*, volume 1, pages 485–509, 2002.

[21] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li. Decentralized deduplication in san cluster file systems. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, USENIX'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[22] O. Corporation. Lustre file system. http://wiki.lustre.org/.

[23] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki. Hydrastor: A scalable secondary storage. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 197–210, Berkeley, CA, USA, 2009. USENIX Association.

[24] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta. Primary data deduplication-large scale study and system design. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 26–26, Berkeley, CA, USA, 2012. USENIX Association.

[25] K. Eshghi and H. K. Tang. A framework for analyzing and improving content-based chunking algorithms. HP Labs Technical Report HPL-2005-30 (R.1).

[26] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. *SIGPLAN Not.*, 33(11):92–103, Oct. 1998.

[27] D. Goodell, S. J. Kim, R. Latham, M. Kandemir, and R. Ross. An evolutionary path to object storage access. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pages 36–41, Washington, DC, USA, 2012. IEEE Computer Society.

[28] I. F. Haddad. Pvfs: A parallel virtual file system for linux clusters. *Linux J.*, 2000(80es), Nov. 2000.

[29] E. Kruus, C. Ungureanu, and C. Dubnicki. Bimodal content defined chunking for backup streams. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[30] C. Lee, D. Sim, J. Hwang, and S. Cho. F2fs: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286, Santa Clara, CA, 2015. USENIX Association.

[31] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, pages 111–123, Berkeley, CA, USA, 2009. USENIX Association.

[32] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 174–187, New York, NY, USA, 2001. ACM.

[33] S. Quinlan and S. Dorward. Venti: A new approach to archival data storage. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[34] M. O. Rabin. Fingerprinting by random polynomials. Harvard University Technical Report TR-15-81 (1981).

[35] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. idedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 24–24, Berkeley, CA, USA, 2012. USENIX Association.

[36] N. Watkins, C. Maltzahn, S. Brandt, and A. Manzanares. Datamods: Programmable file system services. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 42–47, Nov 2012.

[37] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.

[38] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the panasas parallel file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 2:1–2:17, Berkeley, CA, USA, 2008. USENIX Association.

[39] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.