IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 24, 2020, accepted November 10, 2020, date of publication November 18, 2020,
date of current version December 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3039056

A Content Fingerprint-Based Cluster-Wide Inline Deduplication for Shared-Nothing Storage Systems

AWAIS KHAN[1], (Member, IEEE), PRINCE HAMANDAWANA[2], AND YOUNGJAE KIM[1]
[1]Department of Computer Science and Engineering, Sogang University, Seoul 04107, South Korea
[2]Department of Computer Science and Engineering, Ajou University, Suwon 16499, South Korea

Corresponding author: Youngjae Kim (youkim@sogang.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF), Korea government (MSIT), under Grant NRF-2018R1A1A1A05079398, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP), Korea government (MSIT) (Development of low-latency storage module for I/O intensive edge data processing) under Grant 2020-0-00104.

ABSTRACT Deduplication has been principally employed in distributed storage systems to improve storage space efficiency. Traditional deduplication research ignores the design specifications of shared-nothing distributed storage systems such as no central metadata bottleneck, scalability, and storage rebalancing. Likewise, inline deduplication integration poses serious threats to storage system read/write performance, consistency, and scalability. Mainly, this is due to ineffective and error-prone deduplication metadata, duplicate lookup I/O redirection, and placement of content fingerprints and data chunks. Further, transaction failures after deduplication integration often render inconsistencies in data chunks, deduplication metadata, and garbage data chunks. results in rendering inconsistencies in data chunks, deduplication metadata, and garbage data chunks. In this paper, we propose GRATE, a high-performance inline cluster-wide data deduplication, complying with the design constraints of shared-nothing storage systems. In particular, GRATE eliminates duplicate copies across the cluster for high storage space efficiency without jeopardizing performance. We employ a *distributed deduplication metadata shard*, which promises high-performance deduplication metadata and duplicate fingerprint lookup I/Os without introducing a single point of failure. The placement of data and deduplication metadata is made cluster-wide based on the content fingerprint of chunks. We decouple the deduplication metadata shard from read I/O path and replace it with a *read manifestation object* to further speedup read performance. To guarantee deduplication-enabled transaction consistency and efficient garbage identification, we design a *flag-based asynchronous consistency scheme*, capable of repairing the missing data chunks on duplicate arrival. We design and implement GRATE in Ceph. The evaluation shows an average of 18% performance bandwidth improvement over the content addressable deduplication approach at smaller chunk sizes, i.e., less than 128KB while maintaining high storage space savings.

INDEX TERMS Parallel and distributed storage systems, shared-nothing architecture, data deduplication.

I. INTRODUCTION

The shared-nothing storage systems (SN-SS) accommodate a large number of storage servers for high performance, scalability, availability, and fault-tolerance [1], [2]. SN-SS such as GlusterFS [2], Sorento [3] and Ceph Object Storage [1] is widely employed in cloud storage due to multiple properties: (i) it contains no central metadata bottleneck, therefore it is highly scalable, (ii) storage servers are independent where a single storage server failure cannot crash the whole cluster, and (iii) it allows dynamic changes in the cluster, such as addition or removal of storage servers and can relocate objects in the cluster to balance storage utilization across the storage servers.

Nowadays, the ever-growing volume of digital information has raised a critical and increasing concern for storage

The associate editor coordinating the review of this manuscript and approving it for publication was Li Wang.

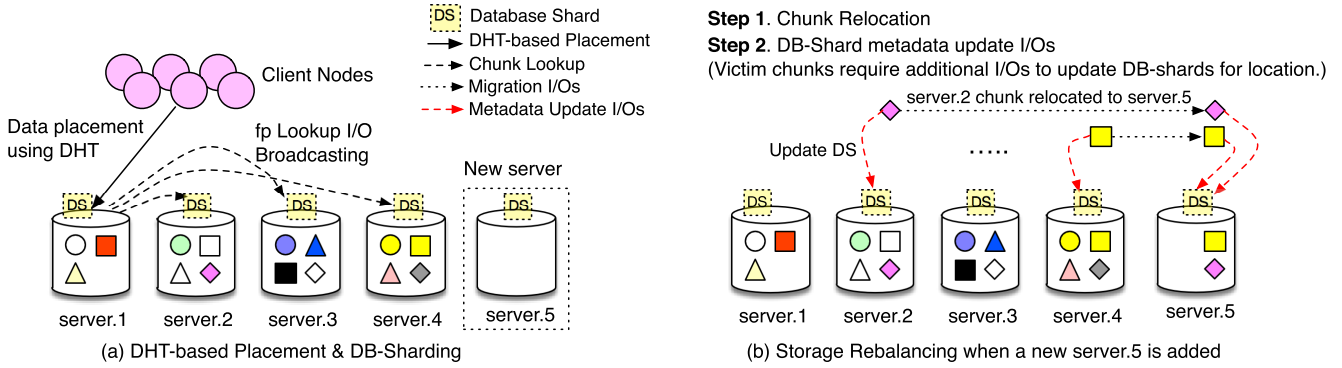


FIGURE 1. (a) Traditional distributed DB-sharding approach and (b) storage rebalancing issues in shared-nothing storage system such as Ceph [1] and GlusterFS [2]. Specifically, (b) illustrates the chunk relocation when a new server is added to the cluster.

capacity optimization [4]–[11]. Deduplication (dedup) techniques are employed widely in storage systems to improve storage efficiency. There exist several studies on cluster-scale dedup [7], [12]–[19]. However, direct adoption of such dedup techniques on the shared-nothing storage system violates the basic design constraints of SN-SS. For example, a centralized dedup approach, where a single centralized dedup server handles all the dedup requests adopted in [12], [14], [20], not only violates shared-nothing properties of SN-SS but also limits the scalability and introduces a single point of failure. Thus, it is critical to consider a decentralized dedup approach for SN-SS. On the other hand, a simple decentralized approach to distributing dedup metadata across multiple external dedup servers [6], [13], [15]–[17], [21]–[26] requires specialized high performance dedup appliances, increasing the management and hardware cost of large-scale clusters [26]. Furthermore, using distributed key-value stores or external database services also violate the node autonomosity of SN-SS.

An alternative approach is to embed dedup on each storage server in the cluster while ensuring the design properties such as shared-nothing and node independence [1], [3]. For instance, a simple database partitioning (DB-sharding) approach [27], [28] that embeds a single database partition (DB-Shard) of the whole dedup metadata database on each storage server has been proposed [18]. However, this database sharding approach to SN-SS suffers from inherited problems, i.e., to identify a duplicate chunk, the fingerprint lookup is broadcasted to all DB-Shards in the cluster, which poses a severe threat to scalability. Figure 1 depicts global dedup implemented via the DB-sharding approach in SN-SS. When a duplicate fingerprint lookup is required, the lookup I/Os are redirected to all the DB-shard instances alive in the cluster to find out the existence of fingerprint, as shown in Figure 1(a). This random broadcasting incurs high-performance degradation, particularly for small chunk sizes employed to improve storage efficiency.

Another challenging issue is deeply related to storage rebalancing. In SN-SS, the storage rebalancing happens on the addition, removal, or failure of a storage server in the cluster. It can also be triggered when there is a significant I/O load or space usage imbalance among servers [3], [29]. This rebalancing shuffles the data chunks across the storage servers to evenly balance the space utilization in the cluster, as shown in Figure 1(b). In such a case, dedup metadata must be updated for the new location of the chunk in the cluster. Therefore, rebalancing incurs high metadata update I/Os. So, to keep track of chunk location across the cluster, the respective DB-shards are updated on each storage server. Additionally, it also requires to modify the existing rebalancing mechanism to monitor chunk location changes. Figure 1(a) and (b) illustrate these problems.

Figure 1(a) depicts the addition of a new server, i.e., server.5 in the storage cluster, whereas Figure 1(b) shows the process of chunk shuffling to balance storage utilization across all the servers in the cluster. The black dotted arrows depict the chunk relocation from one server to the other server, whereas the red dotted arrows depict chunk location updates in database shards. Similarly, the read performance also degrades when dedup is integrated into storage systems due to inherent change in read I/O path, i.e., the object is reconstructed by contacting DB-shards for object layout/metadata which often acts as a bottleneck [30]. Another approach is to employ object recipe or content-addressable object approach [21], [31]–[33]. In this approach, each object has its separate metadata object containing recipe or layout information, whereas reference count related metadata is stored in an extended attribute of each data chunk [31], [32]. This approach is viable for SN-SS, but it incurs high dedup metadata overhead by having a content/recipe file for each object. Moreover, it degrades performance due to slower fingerprint lookup and reference update operations.

Further, dedup also requires transactional changes, where a complete object transaction splits into multiple small fixed or variable chunk-based transactions [18], [34]–[36]. These changes, if not implemented carefully, can cause inconsistent data and dedup metadata in an event of communication, disk or storage server failures. To address such inconsistencies, a soft-update metadata approach in a single disk-based file system was proposed [30], [37]. However, it is not directly applicable to the distributed nature of SN-SS, where parallel

I/Os distribute data chunks. Additionally, transaction ordering and delay operations require extra check-pointing and journaling, which is contrary to dedup, i.e., storage space savings. Another outcome of transaction failures is the presence of garbage chunks in the cluster, which are the remains of failed transactions. The effective removal of garbage chunks is essential in reclaiming space and also to ensure the correctness of reference count against each fingerprint. There exist studies on identifying and removing garbage chunks [24], [38]. However, again both incur additional monitoring and journaling overhead.

To address the above-mentioned challenges, we propose to build GRATE, a fingerprint-based scalable and consistent cluster-wide inline dedup for SN-SS. We employ distributed dedup metadata shard (DM-Shard) hosted on each storage server and use content-generated fingerprints for duplicate lookup I/O and data chunk placement. To ensure transaction consistency and garbage identification, we design a flag-based asynchronous consistency scheme. We decouple the dedup metadata shard from read I/O path with a read manifestation object to further speedup read performance for hot objects.

This paper has the following specific contributions:

- We use the content-generated fingerprint to distribute and locate the chunks in the cluster to overcome I/O broadcasting overhead. We employ database partitioning to handle deduplication metadata in a decentralized manner. The content fingerprint and distributed metadata together enable to preserve the design attributes of SN-SS.
- We design flag-based asynchronous consistency to ensure the correct status of the transaction, data, and deduplication metadata. Our consistency scheme is capable of repairing the corrupt deduplication data and metadata in case of duplicate arrivals.s
- We propose a contention-free read I/O by decoupling DM-Shard from read I/O path to minimize the deduplication metadata bottleneck for hot objects, a significant performance degradation factor in the read I/O path.
- We design and implement an effective garbage collection for distributed deduplication enabled storage systems without additional monitoring and journaling overhead.
- We implement the proposed cluster-wide deduplication in Ceph and evaluate the proposed ideas in a real testbed. We compared GRATE with distributed deduplication and content-addressable approach. The evaluation shows an average performance improvement of 38% and 18% at smaller chunk sizes, i.e., 4KB and 64KB without depreciating disk space savings.

The rest of the paper is organized as follows: Section II describes the background, motivation and challenges. In Section III we describe how we address the challenges through our key components followed by design and implementation in Section IV. We discuss the contention-free read I/O design in Section V. Section VI presents our evaluation results followed by read performance analysis in Section VII and related work in Section VIII. We conclude in Section IX.

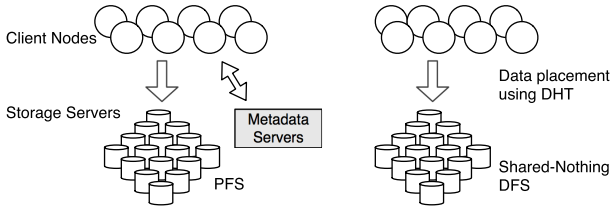


FIGURE 2. An overview of distributed storage architectures.

II. BACKGROUND AND MOTIVATION

In this section, we present the necessary background and practical problems to apply cluster-wide inline deduplication on the distributed shared-nothing storage systems.

A. SHARED-NOTHING STORAGE SYSTEM

SN-SS has emerged as an essential storage architecture in recent years [1]–[3], [39]–[41]. The key characteristics of such systems include, i) high performance and scalability, ii) no centralized metadata bottleneck, iii) no single point of failure, and iv) addition and removal of storage servers on the go. On the contrary, shared storage architectures e.g., Lustre [42] has the issues of a single point of contention and failure, i.e., centralized metadata server. Figure 2 depicts an architectural overview of two systems, i.e., Parallel File system (PFS) with a centralized metadata server shown on (left) and Distributed File System (DFS) following the shared-nothing architecture shown on (right). In particular, Lustre implements a centralized metadata server where file layout information is stored, and when multiple clients access to the metadata server, it becomes a bottleneck. Further, such centralized metadata server also acts as a single point of failure, resulting in loss of data. In contrast, distributed storage systems following shared-nothing architecture such as Ceph [1], [43] and GlusterFS [2], do not have such a single point performance bottleneck issue. Both do not employ a centralized metadata server and instead, use a Distributed Hash Table (DHT) for data placement. Since they know where data is placed in advance before I/O requests are issued, SN-SS can, therefore, scale-out without metadata servers. In particular, Ceph is also widely known for its dynamic and highly efficient placement algorithm CRUSH [44]. CRUSH algorithm empowers Ceph to invalidate the need for metadata servers. Thus, making it a single point of failure-free.

1) CEPH STORAGE MODEL

Ceph is a distributed object storage system that provides excellent performance, reliability, and scalability [1], [45]. Ceph maximizes the separation between data and metadata management by replacing allocation tables with a uniform and balanced data distribution algorithm named CRUSH,

designed for unreliable object storage daemons (OSDs) [1], [44]. Ceph consists of object storage daemon (OSD), monitors, and clients. The logical pools are defined on storage servers, and each pool comprises of several placement groups (pgs), which is configured based on available OSDs. Ceph stores and replicate an object at the granularity of a placement group. When Ceph clients store an object, CRUSH computes the placement group for storing objects using logical pool name, object name hash, and modulo total number of placement groups [44]. The additional details of the CRUSH algorithm can be found in [44].

B. EXISTING DEDUPLICATION TECHNIQUES

Next, we describe the existing data deduplication studies along with their scope and limitations.

1) INLINE VS. OFFLINE

Deduplication in primary storage can be classified as inline or offline, depending on the time when deduplication operation is carried [46]. Inline deduplication is carried out when the I/O is in progress [16], whereas offline deduplication is performed on already stored data [47]. Inline deduplication is highly effective in terms of instant storage space savings [5], [19], [37], [48]–[50]. However, it degrades I/O throughput. It is highly sensitive to I/O latency because it includes chunking, computing fingerprints of data chunks, and duplicate lookups in the critical I/O path.

2) LOCAL DISK-BASED DEDUPLICATION

Data deduplication is an essential component of cloud storage environments [18], [19]. The storage systems with deduplication decrease storage consumption by identifying distinct data chunks with identical content. A single instance of unique data chunk is stored with metadata to reconstruct original data [50]. To gain such storage efficiency, local or disk-based deduplication is adopted in different systems [14], [30], [37], [51]–[56]. A few commercial products such as Pure storage [52], EMC [51], and HPE 3Pa [57] perform inline deduplication at the storage device level, however, requires a massive amount of storage space for deduplication metadata. In disk-based deduplication, each disk is responsible for removing duplicates locally. The benefit of disk-based deduplication is high performance and compliance with storage system characteristics. However, storage space efficiency decreases with the increasing number of disks in the cluster. For instance, as shown in Figure 3, foo and bar are two objects with different names but identical contents. The disk-based deduplication fails to capture such duplicates due to object name-based placement and non-awareness of neighbor disk contents. As reported in [32], local deduplication gains a space savings of only 15% even in the case of a 50% deduplication ratio in the workload. Such local disk-based deduplication solutions fail to improve storage space efficiency in cluster storage environments sufficiently.

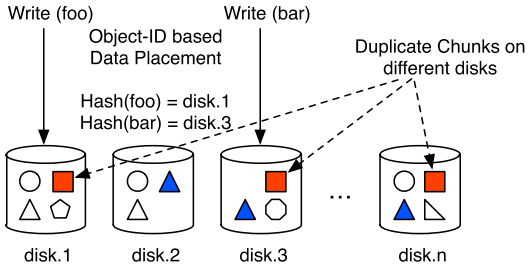


FIGURE 3. An overview of disk-based local data deduplication.

3) GLOBAL DATA DEDUPLICATION

Over the past years, there have been many efforts to develop global or cluster-wide deduplication solutions [12], [13], [15], [39], [49], [58]. In global deduplication, each data chunk can have only a single instance across the whole cluster, unlike local or disk-based deduplication. Venti [12] employs a central deduplication server which does not fit into shared-nothing architectures. HYDRAstor [21] can scale because it uses the distributed content-addressable manifest object to maintain the reference list of each chunk. However, the latency can increase in HYDRAstor [21] when the number of objects increases because the content-addressable manifest object is stored and managed like a data storage object. Extreme Binning [20], SILO [17], $\sum$-Dedupe [16] and Probabilistic Deduplication [13] can remove duplicates from the cluster. However, storage space efficiency is highly dependent on workload because they use different similarity and locality-based algorithms to detect duplicates. Exact Deduplication [18], DeDe [15] and Boafft [23] share high similarity to our proposed design. But these studies require a two-level fingerprint check, i.e., first check fingerprint in local index partition, and then remote node index partition. Moreover, these studies target backup and archival storage rather than primary storage. Further, SolidFire [59], EMC VNX [22], and Clustered Data on Tap [60], all offer global deduplication but limited to block-level interface.

C. FAILURE CASES IN DEDUPLICATION

As per our observation and analysis, deduplication implementation introduces transactional complexity along with failure possibility. The deduplication write order, i.e., storing fingerprint first followed by data chunk storage is essential, to identify duplicates in advance before writing to storage. Whereas, reversing the write order, i.e., storing data chunk followed by storing fingerprint, cannot ensure the presence of duplicate, as every data chunk is stored being unaware that a particular data chunk is already stored in the storage. Further, such reversed order cannot prevent duplicate write I/Os causing additional disk bandwidth consumption. The two likely failures can occur as a result of deduplication integration in SN-SS.

1) PARTIAL TRANSACTION FAILURE

The deduplication integration in storage systems enables storage space efficiency. However, it requires a complete

transaction roll-back mechanism for failure scenarios to ensure data and metadata consistency, which brings more complex transaction handling. The partial transaction failure is defined as when a transaction fails in the middle where all the chunk fingerprint entries are populated in fingerprint index tables, but some of the unique data chunks are not stored entirely due to temporary network, software, or any hardware issue [30], [61]. The effects of this failure include, i) false or invalid entries populated in the fingerprint index table, i.e., the actual data chunk corresponding to fingerprint does not exist in the storage. And ii) subset of data chunks stored correctly during this transaction with no parent object linked, i.e., garbage data chunks. An additional fault-tolerance and recovery patch is required to prevent such failures.

2) REFERENCE TO AN INVALID FINGERPRINT

An invalid fingerprint means that the associated data chunk to this fingerprint does not exist in the storage system. In this case, when a duplicate fingerprint lookup I/O arrives, scans the fingerprint index table. If the fingerprint is found, it simply increments the fingerprint reference count value. For example, let's assume that DM-Shard contains false or invalid fingerprint entry. If a duplicate arrives, it increments the reference count value of that invalid fingerprint, meaning that there are two duplicates. However, there exists no actual stored data chunk associated with those two fingerprints. In the conventional deduplication approach, the presence of a fingerprint in DM-Shard denotes that the corresponding data chunk is also present in the storage, which might not be accurate in the failure case, thus making duplicate fingerprints to increment the reference value. Such reference to an invalid fingerprint problem is attributed to deduplication transaction ordering. The deduplication implementations using reference count logic are highly susceptible to such a problem. Currently, there is no mechanism to ensure that the fingerprint is valid or invalid. In other words, whether the associated data chunk is present in the storage or not. This failure introduces the data and metadata inconsistency.

D. MOTIVATION

Our target architecture is shared-nothing storage systems such as Ceph [1].

Currently, Ceph lacks such inline cluster-wide data deduplication and in order to design deduplication for Ceph, we need to follow shared-nothing storage system design constraints. A simple and centralized deduplication in Ceph introduces the central dependency which breaks the no centralized metadata property of Ceph. The distributed and decentralized deduplication can address such design constraints. However, it poses several other challenges. For example, how accurately and efficiently we can find the duplicate contents in a Ceph storage cluster spanning over 100s of servers. A simple solution is to use a fixed location of chunks in storage cluster. However, we cannot rely on fixed or confined location of data chunks across the cluster because in self-balanced storage systems like Ceph, the data chunks are relocated across the disk and storage servers to balance the storage utilization [1], [3]. Figure 1 depicts this scenario, where a new server is added and chunks are relocated to balance the storage utilization to the newly added server. The fixed or confined location adopted in existing studies can cause serious issues such as additional heavy metadata update I/Os depicted by red dotted arrows in Figure 1(b). Thus, it is very challenging to embed a deduplication approach that complies with the design properties of shared-nothing distributed storage systems such as, shared-nothing and self-rebalancing.

Apart from self-rebalancing, these approaches merely scale as the number of storage servers increases in the cluster. For example, to find duplicates, we need to check all the DB-Shards to find the duplicate fingerprints. Such duplicate fingerprint lookup latency is greatly impacted by the number of nodes in the cluster. The similarity or locality based algorithms such as [13], [17], [25], [29], [62], [63] cannot remove the duplicates from cluster entirely. Similarly, the direct integration of deduplication also influences the read I/O performance because an additional redirection is needed to satisfy the read request. In such scenarios, the deduplication metadata often becomes a hotspot and bottleneck if not carefully designed [30]. Another challenge is to ensure the deduplication data and metadata transactional consistency and correctness. All referenced data chunks and fingerprint entries must be preserved, so deduplicated data can be retrieved for future reads.

A partially failed deduplication transactions can leave garbage data chunks, which are not pointing to any parent object or being referenced by any other chunk in the cluster. The motivation behind the removal of garbage data chunks is to claim the space occupied by the stored data chunks which are not referenced by any object. A simple logging approach to track and remove garbages of failed transactions is viable in the disk-based deduplication approach. But it is challenging in clusters spanning over 100s of OSD servers. As, logging based methods require additional space, which is contrary to deduplication, i.e., less space savings.

Overall, the motivation of this study is to design an inline server-side cluster-wide deduplication, which has low fingerprint lookup I/O overhead, and it can adapt to a node joining and removal seamlessly. We also consider it critical to solve the deduplication data and metadata inconsistencies from the failed transactions in our cluster-wide deduplication design. Further, to minimize read performance degradation introduced by the inclusion of deduplication, we consider decoupling the read I/O path from write I/O.

III. GRATE: CLUSTER-WIDE DATA DEDUPLICATION

In this section, we discuss the critical design decisions for cluster-wide deduplication in shared-nothing storage systems. We firstly introduce the overview of each component followed by the workflow, as shown in Figure 4.

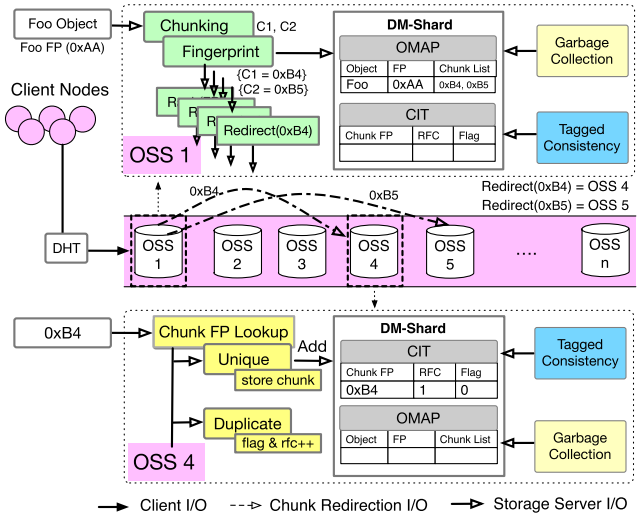


FIGURE 4. GRATE: cluster-wide deduplication based on DB-sharding and content-fingerprint based placement in SN-SS.

A. SYSTEM OVERVIEW

The proposed deduplication design comprises five components, as shown in Figure 4.

1) CONTENT FINGERPRINT-BASED I/O REDIRECTION

This module ensures the distribution and lookup of deduplication metadata and data chunks. One of the challenges in our study is not to fix the location of data chunks due to storage rebalancing property of storage systems such as Ceph [1] and Sorrento [3]. We accomplish this by employing a second CRUSH [44] algorithm to distribute and locate the data chunks and chunk fingerprints.

2) DISTRIBUTED DEDUPLICATION METADATA SHARD

Each storage server stores data chunks and deduplication metadata shard, i.e., DM-Shard in Figure 4. Specifically, we manage deduplication metadata in a scalable, distributed manner. Each deduplication metadata partition (DM-Shard) keeps the unique information of objects and data chunks in a separate data structure, i.e., Object Map (OMAP) and Chunk Information Table (CIT).

3) FLAG-ASSISTED CONSISTENCY

This module provides the consistency of data chunks and deduplication metadata. We integrate a consistency flag against each chunk fingerprint in the deduplication metadata shard. The flag validates the consistency status of any chunk fingerprint entry in DM-Shard.

4) CONTENTION-FREE READ I/O

To improve the read performance due to the integration of deduplication, the contention-free read I/O module selectively generates the manifest object to speed-up and for the scalable reconstruction of hot objects for the read I/Os.

5) PARALLEL GARBAGE COLLECTION

This module periodically executes on each object storage server. It identifies, collects, and removes the garbage data chunks and invalid fingerprint entries in deduplication metadata shards.

B. WORKFLOW

GRATE write workflow is shown in Figure 4. The client performs object name hashing and locate the storage server to write or read objects in the cluster. Each storage server performs deduplication and stores data and metadata. When storage server receives a write request (OSS 1 in Figure 4), it is responsible for splitting the object into small fixed-size data chunks and computing the fingerprint for each chunk's content. Then, it redirects the data chunk to the storage server based on the computed fingerprint (OSS 4 in Figure 4). This fingerprint-based redirection frees from keeping the location of each data chunk in the storage system. At this point, the storage server builds a mapping of the object and its data chunks' fingerprints in Deduplication Metadata Shard (DM-Shard) as shown in Figure 4 (OSS 1). We explain the DM-Shard in Section IV-A.

The redirected chunks received on other storage servers (OSS 4 in Figure 4) are treated in the following manner; The chunk fingerprint lookup is made in Chunk Information Table (CIT) of DM-Shard. If chunk fingerprint exists and consistency flag is valid, then the reference count (RFC in CIT) increment is granted. Whereas, the non-existence of fingerprint is treated as a unique chunk. CIT entry is populated with an invalid flag and data chunk is stored in the storage server (OSS 4). This process is iterated for all the data chunks in parallel. When all the chunks are stored, then Object Map (OMAP) entry is created (OSS 1 in Figure 4) which defines the object layout such as name, fingerprint and chunk list of the object. The write operation finishes, when all the data chunks, OMAP and CIT data structures are created. The flag-assisted consistency guarantees the validity and correctness of all the CIT entries and data chunks in storage without additional logging and journaling. The DM-Shard and tagged consistency together assist in identifying the garbages and orphan data chunks, i.e., remains of partially failed transactions. The chunk fingerprints with an invalid consistency flag (Flag in CIT) are interpreted as garbage data chunks and collected periodically.

Note that, each read I/O first landing to any OSS needs to contact DM-Shard's OMAP to fetch the object reconstruction layout, i.e., list of data chunk fingerprints belonging to the object. Then, each data chunk against the fingerprint is read in parallel, and the full object is assembled and returned to the client. Besides, we also offer a different read I/O path for objects with high sharing and access frequency, i.e., unique data chunk, which can become hotspot or overlaps in many objects. We generate a recipe or manifest object for such objects and avoid aggressive contact points to DM-Shard, thus improving read I/O latency in deduplication storage systems. We detail this contention-free read I/O in Section V.

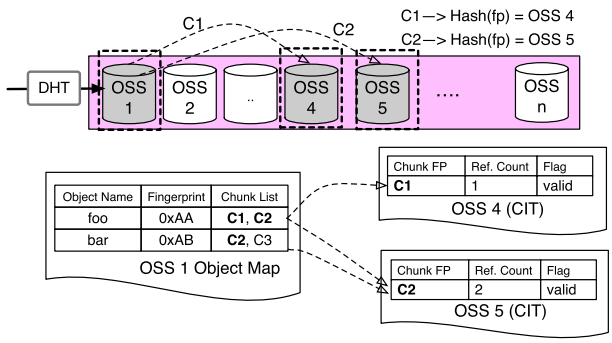


FIGURE 5. Object map and chunk information table layout.

IV. DESIGN AND IMPLEMENTATION

In this section, we discuss in depth the design and implementation of GRATE.

A. DEDUPLICATION METADATA SHARD

We build Deduplication Metadata Shard (DM-Shard) as shown in Figure 4 to effectively manage deduplication metadata. The design decision to use distributed DM-Shard is to comply with the scalable and shared-nothing property of shared-nothing storage systems. The centralized deduplication metadata design limits the scalability and violates the shared-nothing property of SN-SS. Every storage server in the cluster hosts a DM-Shard holding all the persistent data structures such as object layout information and data chunk fingerprint. Each shard keeps unique information of objects and data chunks in a separate data structure, i.e., Object Map (OMAP) and Chunk Information Table (CIT), as illustrated in Figure 5.

- *Object Map:* OMAP maintains the complete layout and reconstruction logic of an object, i.e., object name, object fingerprint, and list of data chunks. The OMAP data structure is shown in Figure 5. In DHT-based storage systems, an object is identified by hashing the object name. If we do not maintain the hash of the object, we cannot reconstruct the original object because we need all the chunks' fingerprints created from this object. OMAP assists in read operations, where object fingerprint is given to lookup chunks belonging to a specific object. Each row of OMAP denotes an original object layout.
- *Chunk Information Table:* CIT maintains the performance-sensitive deduplication metadata. It includes data chunk fingerprint, reference count, and commit flag. All the chunk lookup and reference update operations are possible via this data structure. Each row of CIT denotes information about a specific chunk fingerprint.

The benefit of keeping different data structures is manifold: i) To provide an effective execution of fingerprint operations, i.e., lookup, increment/decrement, ii) Reduced congestion on a single data structure when multiple I/Os access the data structure, and iii) To avoid data chunk fingerprint lookup in case of the read request.

Both OMAP and CIT data structure entries are created synchronously during a write operation to avoid concurrent lookups of identical fingerprints, which can result in storage inefficiency. We describe complete read and write I/O transaction with the usage of OMAP and CIT in Subsection IV-B. To ensure deduplication metadata replication and fault-tolerance, we rely on the underlying shared-nothing storage system because we store our DM-Shard in the storage server, and it is replicated like a regular object.

B. CHUNK RELOCATION AND I/O ROUTING

SN-SS such as Ceph [1] and Gluster [2] distribute objects in a storage-balanced fashion. For instance, Ceph uses the CRUSH algorithm [44] to fairly distribute the storage load across the storage servers, when the cluster topology changes, e.g., a new storage server is added, removed or disk failure occurred. The objects are relocated across the storage servers to balance the storage load in the cluster, as shown in Figure 1(b). This object and chunk relocation process is neglected in previous deduplication studies such as [16], [18], [21], [23]. In previous studies, the location of the object and data chunks is stored along with metadata, i.e., data chunk 1A is stored on server.a, and data chunk 1B is stored on server.b. This type of deduplication metadata management suffers when chunks are relocated in the cluster because the object and chunk location is lost. One solution can be to transform current self-balancing mechanism to update the deduplication metadata while relocating the objects and chunks, but it entails the complex implementation and a high number of I/Os for every object and chunk relocation to update the deduplication metadata.

To determine the exact location of the data chunk and respective DM-Shard across the cluster, we use the data chunk fingerprint. The fingerprint can be obtained in two ways: i) to generate the fingerprint directly from the data chunk contents (write request approach), and ii) to obtain the data chunk fingerprint from OMAP fingerprint tells the storage server location responsible for storing the actual data chunk and the deduplication metadata shard (CIT). This content-based placement relieves us from i) complicated location management for each data chunk, ii) modifications in the existing self-balancing mechanism, and iii) frequent deduplication metadata updates. Another gain of this content-based placement is that we do not require to broadcast I/Os to all storage servers for fingerprint lookup. Instead, we send a single lookup I/O to only a single storage server.

C. WRITE I/O FLOW

Next, we discuss the write I/O transaction flow in our proposed cluster-wide dedup approach, as shown in Figure 6(a) & (b). The encircled number depicts the sequence of steps/operations in Figure 6.

1) WRITE I/O

The client performs object name hashing and locates the storage server to write the object. At first, the object is divided

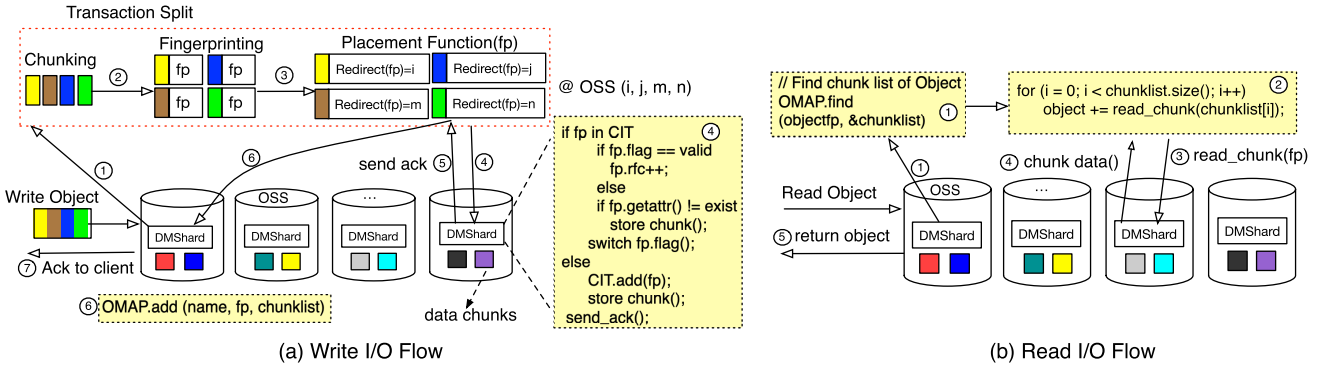


FIGURE 6. A complete write I/O transaction in cluster-wide data deduplication system.

into small data chunks, as shown in Figure 6(a) (step 1). The different color represents unique data chunk content, and the fingerprint is computed against each data chunk's content (fp in step 2). Then, the fingerprint is supplied to the placement function for redirection to OSS in the cluster (step 3). This placement function is exactly same with client placement function, i.e., CRUSH in Ceph [1] and DHT in GlusterFS [2]. The DM-Shard hosted on redirected OSS validates the fingerprint existence in CIT. If fingerprint exists and the consistency flag is valid, then the reference count increment is granted (step 4). If fingerprint exists and the consistency flag is invalid, then reference count increment is not granted. Rather, a stat like system call is issued to chunk storage to validate the existence of data chunk contents (getattr() in step 4). If the stat call fails, then we store the data chunk and switch the consistency flag to valid. If stat returns true then, we switch flag to valid and send the acknowledgment to OSS responsible for performing chunking, fingerprinting, and redirection (step 5 & 6). Finally, respective OSS sends the acknowledgment message to the client (step 7).

2) READ I/O

The read I/O is simpler than the aforementioned write I/O, as shown in Figure 6(b). The client issues a read object request, and the request is redirected to OSS bound to store the object using the CRUSH algorithm. At this moment, we retrieve the object layout information (list of chunk fingerprints comprising the object) from the OMAP stored in the DM-Shard given object fingerprint (step 1 & step 2). The OMAP structure is shown in Figure 5. Next, we issue parallel I/Os to read chunk data (step 3 & 4) and then reconstruct the object and return to the client (step 5).

D. FLAG-ASSISTED ASYNCHRONOUS CONSISTENCY

The deduplication metadata inconsistencies in distributed storage systems lead to data authenticity and integrity issues [61], [64]. For example, if an object transaction is split into multiple chunk-based transactions, and one of the small transactions fails. Then, in such a case, the whole object transaction fails, and two problems are likely to happen. First, it results in an invalid or misleading reference fingerprint in DM-Shard, and second, the existence of garbage data chunks left from the failed transaction. Worst of all, a new incoming duplicate fingerprint increments the invalid reference fingerprint entry, causing metadata inconsistency. Due to transaction-level modifications, a complicated transaction and rollback logic are required to cater to failure cases and ensure reference count consistency [24], [38]. A simple solution is to populate fingerprint entries in DM-Shard once the transaction completes successfully, and if the transaction fails, there are no invalid fingerprint entries. However, this simple approach has two significant problems, i.e., First, it compromises the deduplication efficiency because when duplicates arrive in parallel, we cannot check fingerprints in DM-Shard. Second, if any failure happens during the transaction, the data chunks become unreachable, as there are no corresponding fingerprints in DM-Shard.

To address such consistency concerns, we add a consistency flag to each data chunk entry in CIT, which not only specifies the consistency state of the chunk but is capable of restoring the missing data chunk as well. The CIT structure is shown in Figure 5. The consistency flag can only be valid or invalid. By default, the consistency flag is invalid for every chunk entry in CIT. The valid consistency flag ensures that the data chunk is stored correctly, and deduplication metadata is consistent. Whereas, an invalid consistency flag shows that data chunk corresponding to the fingerprint entry is missing from storage or transaction is currently in progress.

An alternative is to add a consistency flag with an object or chunk entry and update the consistency flag at transaction completion synchronously. However, such a choice requires transaction lock and updating the flag synchronously, which affects the scalability of the storage system. To avoid such unnecessary transaction locking, we propose an asynchronous thread-based consistency manager that runs on every storage server. All the incoming write I/Os registers to consistency manager. Once the I/O transaction completes, the consistency manager asynchronously updates the consistency flag managed in CIT (Section IV-A). If there is a crash during reference count update I/O, we do not roll

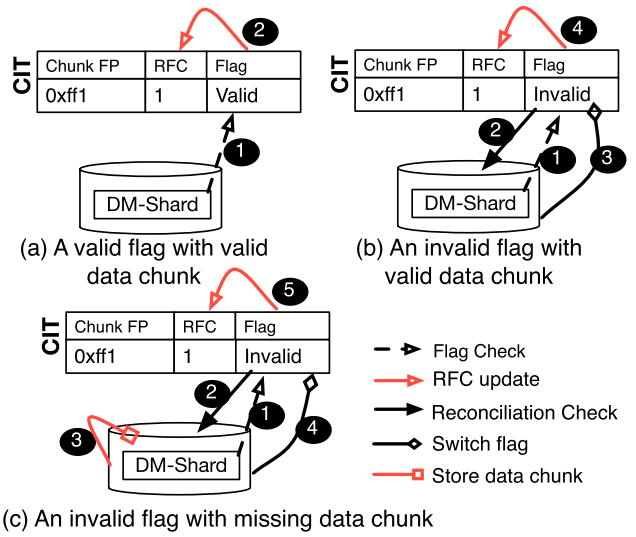


FIGURE 7. Reference count increment operation with valid and invalid consistency flag.

back the whole transaction. Instead, such fingerprint entries with an invalid consistency flag are collected by the garbage collector, and later, both data chunk and fingerprints are removed.

Note that, when storage rebalancing is triggered, the system relocates the data chunks across the cluster. In such cases, the CIT entry is populated with an invalid flag on DM-Shard for the relocated data chunk, and once the data chunk is rearranged successfully, then the flag entry is updated. Note that the flag update is carried out in the same fashion as for the regular data chunk write operation. In the current scope of the work, there might exist some reference count values which can be higher than the real duplicates referring to that fingerprint. However, we believe that such can be tolerable compared to a fingerprint entry having no data chunk associated in the storage.

1) UNIQUE AND DUPLICATE WRITE USE-CASE

We describe the proposed consistency scheme with simple use-cases, as shown in Figure 7.

a: UNIQUE WRITE

In this case, the object splits into fixed-size small chunks and stores the chunk on different object storage servers based on data chunk fingerprint. Whenever a CIT entry is created for any data chunk with an invalid consistency flag, the consistency manager is notified. It validates the status of the chunk in storage and switches the consistency flag from invalid to valid asynchronously. This approach does not introduce additional notable latency to write I/O. Note that, the consistency manager is not notified for reference count operations such as increment or decrement of chunk fingerprint with a valid consistency flag. Moreover, it is not possible to have an entry with a valid consistency flag and no associated data chunk stored in the storage system.

b: DUPLICATE WRITE

In duplicate write case, when a duplicate fingerprint arrives and requires to increment the reference count in CIT, it needs to check the flag before increment as shown in step 1 of Figure 7(a). The fingerprint entries with a valid consistency flag allow the reference count increment. However, if the consistency flag is invalid, additional measures are needed to ensure the consistency of deduplication data and metadata. Figure 7(b) shows a case of partial transaction failure where a chunk is stored successfully, but due to crash, the flag is not switched from invalid to valid. Then, there is a need to perform an additional consistency check, which we call *Reconciliation Check* (RCC), to ensure the existence of data chunk in the storage server (step 2 in Figure 7(b)). The RCC operates, in the same manner, like get attribute system call in the file system. If RCC returns the data chunk attributes, we switch the consistency flag from invalid to valid (step 3) and proceed with reference count operation, i.e., increment (step 4). Another use-case is for duplicate arrival similar to Figure 7(b), i.e., the flag is invalid, but the actual data chunk linked to the corresponding fingerprint is missing from storage as depicted in Figure 7(c). Such a scenario can happen when fingerprints are stored in DM-Shard, and failure occurred before the storage of data chunks. This use-case highlights how the proposed consistency flag and reconciliation check can repair the data chunk inconsistency. When the reconciliation check finds that data chunk is missing from the storage (Figure 7(c) step 2), it stores the actual data chunk (step 3), and the flag is changed from invalid to valid (step 4). Then, only the reference increment operation is granted (step 5).

We claim that the proposed asynchronous consistency scheme ensures the data and deduplication metadata accuracy even in case of failures and prevents the storage system from inconsistencies.

E. PARALLEL GARBAGE COLLECTION

The garbage identification and removal are studied in previous studies as well [24], [37], [38], [61]. However, these studies use different kinds of logging and monitoring, which can overload the storage servers and consume the storage server resources, thus reducing the overall system performance. To claim free space consumed by garbage data chunks, we design an effective parallel garbage identification and removal mechanism. As our deduplication service is embedded on each object storage server, so the garbage collection is also dedicated to each object storage server. The garbage collection thread runs in the background and periodically collects the data chunk fingerprints with an invalid consistency flag from CIT. It keeps the fingerprints for a pre-defined threshold. Once the threshold expires, the thread cross-validates the consistency flag of collected fingerprints to CIT entries. This matching is a compute-intensive operation and is required to assess any change, in particular to the collected fingerprints. If there is no change in consistency flag status for collected fingerprint entries, then the garbage

data chunks are removed from the storage system, and their corresponding fingerprints are also removed from DM-Shard. Furthermore, the fingerprints with an invalid flag but no corresponding data chunks in storage are also removed. We do not use any additional journaling because it requires extra disk space.

Our garbage collection operates in three phases mainly as shown in Figure 8;

Select: The select phase collects and buffers the fingerprints with an invalid consistency flag from local DM-Shard, as shown in Figure 8 (step 1).

Filter: The filter phase cross-validates the collected fingerprint flags with respective fingerprint flags in local DM-Shard and removes the fingerprints from the list with an updated consistency flag. For example, if a fingerprint in the select phase has an invalid flag, but while cross-validating, the fingerprint consistency flag is valid. Then, we do not remove this reference fingerprint from DM-Shard. Neither we remove the associated data chunk. It is triggered after the select phase based on a configured time threshold. Note that, a smaller time threshold can compromise the storage server throughput and can elevate CPU usage.

Remove: This is the final phase, where we remove the stored chunks (step 3.1) and buffered entries from CIT (step 3.2) in Figure 8. We pick fingerprint entry from the buffered list, check the stored data chunk against it. If the data chunk exists, remove the data chunk first and then fingerprint entry from the DM-Shard. Otherwise, remove the fingerprint entry from the DM-Shard.

Note that the ordering of fingerprint and data chunk removal is important in deduplication environments. Figure 8 (step 3.1 and 3.2) reflects the order when data chunks are removed before fingerprint removal. It is because if there is any failure or interruption in the garbage collection operation after data chunks are removed, we can collect and remove the fingerprint entries from the DM-Shard in the next garbage collection cycle. On the other hand, if the fingerprints are removed earlier than the data chunks and failure happens, the data chunks become unreachable. It is because the fingerprints referencing to those data chunks are already removed, Thus, leaving garbage and freely floating data chunks inaccessible.

V. CONTENTION-FREE READ I/O

In this section, we discuss the motivation behind proposed contention-free read I/O design.

One of the dominant limiting factors in degraded read performance in deduplication enabled storage systems is essentially derived from high access latency to deduplication metadata [30]. There are several factors elevating latency access to deduplication metadata. Table 1 shows the deduplication metadata I/Os making contact with DM-Shard aggressively in our previous work [36]. So, when the workload is a mix of read and write I/Os, each deduplication metadata I/O hitting DM-Shard contributes to introducing additional latency for the read I/Os. It is because underlying

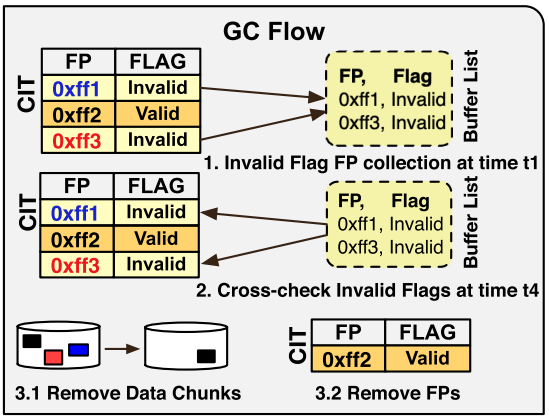


FIGURE 8. Overview of garbage collection flow. Note that, RFC column in CIT is not shown for simplicity.

TABLE 1. Deduplication metadata I/O categorization with respect to DM-Shard.

Deduplication I/Os	I/O Description
Lookup I/O	Scans DM-Shard to find duplicate fingerprint.
Reference I/O	Increment or decrement I/Os to manage reference counts.
Consistency I/O	Updates the flag to ensure consistency status of fingerprints in DM-Shard.
Garbage I/O	Periodically checks the fingerprint entries with an invalid flag status.
Read I/O	Gets the layout of object to reconstruct the contents (Section IV-A).

DM-Shard is a single thread, which fails to facilitate multiple requests simultaneously. Even if it is multi-thread implementation, then various writers will acquire the lock, incurring additional latency. Similarly, when a read I/O comes, it has to wait until the availability of DM-Shard, in particular, OMAP, as shown in Figure 5. Moreover, the problem gets worse when the workload has high deduplication ratio, which denotes a particular server hosted DM-Shard is going to be a hotspot for deduplication metadata I/Os shown in Table 1 due to content-centric deduplication. If the deduplication core architecture is changed, i.e., similarity or locality-based algorithms to identify duplicates as proposed in [17], [25], [26], it reduces the deduplication efficiency. The goal of our research is to design high performance and scalable cluster-scale deduplication without compromising space savings.

The existing deduplication approaches intended for global deduplication in Ceph [31], [32], [36] are not suitable for both read and write at the same time. For instance, [31], [32] heavily relies on managing reference count values in extended attributes of chunks, which degrades write I/O performance. Whereas, [36] maintains reference count and consistency flags in DM-Shard, which is a magnitude faster when it comes to fingerprint scan/lookup and reference update operations.

To reduce the conflict between read and write I/Os in DM-Shard, we propose to decouple read I/O path from write I/Os by designing a hybrid approach, where the write I/Os are served via DM-Shard and read I/Os are

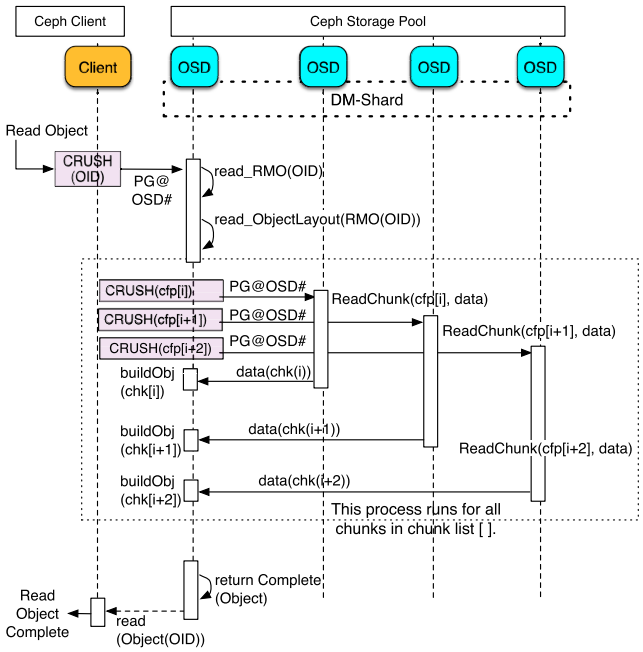


FIGURE 9. **Contention-free read I/O path. The cfp[] denotes the chunk fingerprint list similar to OMAP's chunk list (see Figure 5).**

served from a content-addressable manifest object. In the rest of the paper, we refer to it as Read Manifest Object (RMO). The RMO is nothing but just a Ceph object containing the object's layout information, similar to OMAP in DM-Shard. The reason to redirect read I/Os to RMO objects is manifolds, i.e., i) RMO is treated like a Ceph object and reads are served in a scalable fashion, ii) it reduces stress on DM-Shards, and iii) it is easy to cache frequently accessed RMO objects. There are several studies solving read contention problems with respect to data contention [29], [32], [61], [65]. However, most of these deduplication works neglect deduplication metadata contentions in read I/O path.

A. READ MANIFEST OBJECT GENERATION

By simply generating RMO against each object incurs high space overhead and will result in a higher number of additional I/Os as well in the cluster. Thus, we propose to dynamically generate RMO against hot objects with high access frequency. The hotness of the object is defined by the chunk with high reference count value (CIT in Figure 5). Thus, we can improve write I/O latency by minimizing stress on DM-Shard and also improve read I/O latency by decoupling it from DM-Shard. However, it is not true for every read I/O, and some read I/Os for objects with low hotness measure will be still entertained by DM-Shard. We specifically generate RMOs by scanning CIT of DM-Shards, and backtracking the CIT chunk entry with higher reference count in CIT and generate a RMO object against particular object in OMAP. Further, different cache eviction policies can be implemented on such RMO object generation and removal to minimize storage space utilization.

B. READ I/O REDIRECTION

Figure 9 shows the read I/O path in Grate. As we have discussed earlier the read I/O flow in Section IV-C2, and is valid for objects with low hotness value or access frequency. However, for objects with higher access frequency, the I/O request will not hit DM-Shard and instead CRUSH [44] will directly retrieve the generated RMO against read I/O for object layout information and read data chunks in parallel as shown in Figure 9. Note that, the proposed contention-free design will reduce the stress on DM-Shard and decouples the read I/O path from write I/O. However, the chunk fragmentation problem still exists and we consider to solve this challenge as our future work.

VI. EVALUATION

This section provides the evaluation of proposed cluster-wide data deduplication framework.

Implementation: We implement Grate in Ceph v10.2.3. The DM-Shard, consistency manager, and garbage collection thread are embedded in each OSD. We use fixed-size chunking and SHA-1 algorithm to generate a data chunk fingerprint. We pass the fingerprint to the CRUSH algorithm [44] to distribute the data chunks in the Ceph storage cluster. The deduplication operations such as fingerprint lookup I/Os, tagged consistency, and garbage collection are achieved via the Ceph standard messenger framework. We slightly modified the self-balancing and recovery mechanism to update deduplication metadata when data chunks relocate across the storage cluster. We use SQLite [66] as back-end storage for DM-Shard.

A. EXPERIMENTAL SETUP

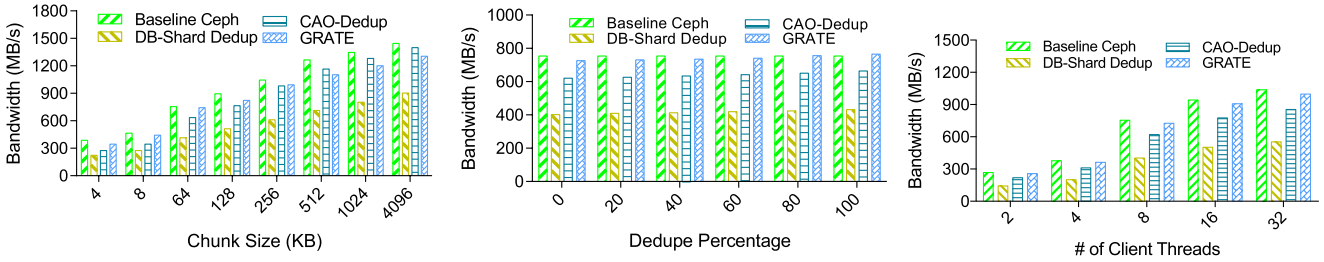
We configured Ceph storage cluster on a testbed consisting of 7 OSSs, 3 Monitors, and 4 Ceph client nodes. Each machine is equipped with Intel E5-2670v4@2.40GHz (10 Cores), 32GB DRAM and 2 × 256GB Samsung SSDs per OSS running Linux CentOS v7.3.

We used the FIO [67] benchmark for evaluation by varying deduplication ratio and number of client threads with a 4 TB synthetic write I/O workload. We compare the proposed deduplication approach (Grate) with three variants.

Baseline Ceph: Ceph without deduplication integration.

DB-Shard Dedup: Ceph with deduplication implemented via simple DB-sharding. It stores the location of chunks in OMAP, i.e., fixed or pinned location and does not use content fingerprint-based I/O redirection. To check duplicates, it requires to broadcast fingerprint I/Os to all the DM-Shards on each OSD server across the cluster. Note that, DM-Shard uses SQLite DB with a single thread implementation as in Grate.

CAO-Dedup: Ceph with deduplication implemented via content-addressable object. It is also referred to as hash object or file recipe approach in the previous studies [21], [31], [32]. In this approach, each object transaction has its own separate content addressable object which lists the



(a) Performance analysis {0% DR, 8 Clients}

(b) Deduplication ratio {128 KB Chunk size, 8 Clients}

(c) Scalability with multiple clients {128 KB Chunk size, 20% DR}

FIGURE 10. Comparative performance analysis with distributed deduplication and content addressable approach. The DR denotes deduplication ratio.

chunk fingerprints, i.e., similar to our OMAP structure (see Section IV-A). However, the reference count is maintained in the extended attribute of each stored data chunk. As discussed in Section VIII, due to high design similarity in both [31] and [32], we only compare the proposed approach with [31]. We claim that, the comparison with one of the two studies in the current manuscript reflect the characteristics of both studies and does not require comparison to each of them individually and CAO approach [31] in the current manuscript reflects the characteristics of the work in [32] as well.

B. WRITE PERFORMANCE ANALYSIS

1) VARYING CHUNK SIZE

Figure 10(a) shows the write bandwidth of all four approaches. We set the deduplication percentage to zero, i.e., no duplication and use 8 client threads in FIO benchmark. We observe that, DB-Shard Dedup shows poor performance out of all approaches due to duplicate lookup I/O broadcast problem, i.e., to find a duplicate fingerprint, it requires to send I/Os to all DB-Shards in the cluster. Whereas, GRATE and CAO-Dedup both show scalable performance with increasing chunk size. It is because both of the approaches employ content-generated fingerprint and no blind lookup I/Os are broadcasted. An important thing to note here is that GRATE and CAO-Dedup show a little performance difference but when the chunk size gets bigger than 512KB, CAO-Dedup outperforms our approach. It is mainly attributed to OSD caching. CAO-Dedup performance improves via OSD caching which makes duplicate lookup operations faster. On the contrary, the smaller chunk size ends in more OSD cache misses, thus leading to degraded performance compared to our approach. GRATE has high performance and space efficiency for small chunk sizes. CAO-Dedup, on the other hand, has poor performance in this case, i.e., GRATE is better than CAO-Dedup when it requires storage space efficiency.

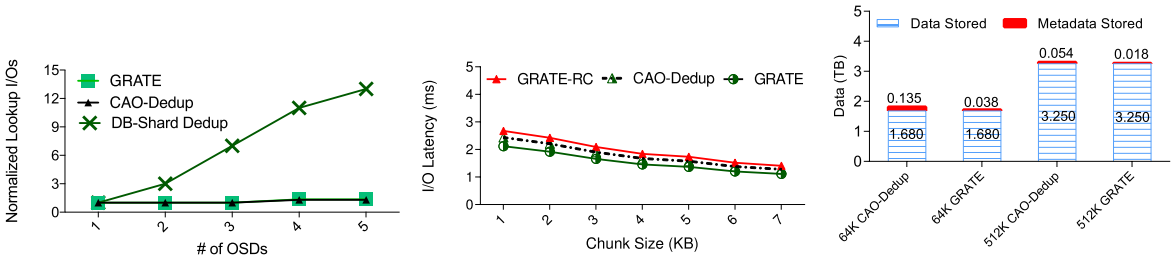
2) VARYING DEDUPLICATION RATIO

Next, we discuss the performance of GRATE with respect to deduplication ratio as shown in Figure 10 (b). We set the chunk size to 128 KB and use 8 client threads to compare GRATE with other approaches. We observed that all the three approaches, i.e., DB-Shard, CAO-Dedup and GRATE shows a limited performance improvement up to a certain threshold regardless of deduplication percentage in the workload. A simple fact is that the high deduplication ratio incurs reduced number of writes in the cluster, hence increasing the performance. Whereas, at high deduplication ratio, the expected performance gain is masked by chunking, fingerprint and deduplication metadata operations. The higher deduplication ratio, the high number of deduplication metadata I/Os, i.e., specifically reference increment I/Os. Surprisingly, all of the approaches show an average of only up to 6% of performance improvement compared to 0% deduplication ratio. One of the reason for less performance improvement is derived from redirection of small data chunk I/Os over the network, which are too small to show further improvement even if not stored. An important thing to note here is that, if the network is slower, the performance improvement will be higher with increasing deduplication ratio in the workload.

3) VARYING NUMBER OF CLIENTS

To test the scalability, we vary the number of client threads in FIO [67]. We set the chunk size 128KB and fixed deduplication ratio to 20%.

Figure 10(c) shows that all of the approaches scale with increasing number of client threads. However, as the client thread varies from 8 onwards in DB-Shard Dedup, the performance increase becomes less compared to performance gain between 4 and 8 threads. A simple reason is that; DB-Shard Dedup uses a single thread version of SQLite which cannot scale with massively broadcasted duplicate chunk I/Os causing contention on DB-Shards. Similarly, GRATE also has same DB-Shard implementation but I/O broadcast problem is resolved by using content-fingerprint based I/O redirection, i.e., CRUSH [44], which minimizes the stress on each DB-Shard. GRATE show scalability and improves the bandwidth with increasing number of client threads because CRUSH [44] distributes the data chunks uniformly in a load-aware fashion to object storage servers and DM-Shard is distributed across all the object storage servers which overcome the possible chances of deduplication metadata



(a) Normalized fingerprint lookup I/Os (b) Reference update I/O latency (c) Deduplication metadata storage

FIGURE 11. Deduplication metadata lookup I/O frequency and latency analysis.

contention. Even though, CAO-Dedup also uses CRUSH [44] for data distribution, the duplicate chunk lookup and reference management in extended attribute limits the performance increase in CAO-Dedup.

Note that, we observed a similar performance trend for Figure 10(b) and (c), for all the chunk sizes as in Figure 10(a), thus we show only results with 128 KB chunk size. In overall, the proposed approach outperforms the CAO-Dedup and DB-Shard Dedup in smaller chunk sizes and show a substantial performance gain when deduplication ratio increases in workload. Further, scalability with increasing number of clients also proves that the proposed design is highly robust.

C. I/O REDIRECTION AND DEDUPLICATION METADATA

In this subsection, we discuss the content fingerprint based I/O redirection scalability and deduplication metadata storage overhead.

1) LOOKUP IO SCALABILITY AND REFERENCE OPERATIONS

The efficient and scalable fingerprint lookup directly impacts the deduplication enabled storage system performance. The distributed storage systems comprising of hundreds of OSDs require a scalable lookup I/O to ensure high performance. Figure 11 (a) shows the fingerprint lookup I/O performance with respect to increasing number of OSDs. We observe from the results that the GRATE shows a consistent lookup I/O latency as compared to DB-Shard Dedup. However, DB-Shard Dedup broadcasts fingerprint lookup I/O to all the OSDs across the cluster to validate the duplication of chunk. This I/O broadcast limits the scalability of deduplication storage systems.

For deduplication metadata operations, in particular, for reference count increment, we compare the proposed with approach with two implementations, i) CAO-Dedup, and ii) GRATE-RC, a variant of the proposed approach, in which the for every reference increment I/O, an additional *Reconciliation Check* I/O is initiated to validate the status of data chunk in the storage. Note that, unlike GRATE-RC the proposed approach only issues the *Reconciliation Check* I/O when the consistency flag is invalid and not for every I/O.

Figure 11 (b) depicts the reference count increment operation for each of the aforementioned approach. As observed from the previous analysis in Figure 10, when the chunk size is much smaller, CAO-Dedup performance is lower compared to GRATE. It is because of placement group locking structure of Ceph, i.e., a higher number of reference increment I/O requests land to the same placement group. However, Ceph uses lock to ensure consistency at placement group level, introducing additional latency to dedup metadata operations. More importantly, we observed GRATE-RC shows poor performance than CAO-Dedup, it is because an additional lookup I/O is redirected to storage to confirm the presence of actual data chunk. In reality, GRATE only redirects the increment I/O if the consistency flag is invalid for a particular fingerprint entry, which is an outcome of transaction failure scenarios.

2) DEDUPLICATION METADATA STORAGE OVERHEAD

To analyze the deduplication metadata storage overhead, we compared the proposed approach with CAO-Dedup approach. The CAO-Dedup creates an additional content addressable object to facilitate future read operations and similarly, this content addressable object is also replicated. We use two different chunk sizes to clearly observe the metadata storage overhead,.i.e., 64K and 512K as shown in Figure 11(c). For both chunk sizes, we observe that CAO-Dedup incurs more than the twice metadata storage overhead compared to GRATE's metadata storage overhead. It is because we used database object to manage all deduplication metadata. Whereas, CAO-Dedup manages the reference counter in extended attribute and each content addressable object, i.e., layout/CAO object has its own Ceph object metadata. Irrespective of the chunk size, the Ceph object metadata cannot be avoided which is approximately at least 512 bytes for each content addressable object [32].

Overall, the number of duplicate lookup I/Os of proposed approach and CAO-Dedup are same due to usage of same methodology, i.e., CRUSH algorithm to place and locate the fingerprint and data chunks. However, the reference update operation and deduplication metadata storage overhead is lower in the proposed approach than the counter parts. Moreover, the deduplication metadata storage overhead of GRATE is magnitude less than CAO-Dedup, thus contributing to disk storage space savings.

D. ASYNCHRONOUS TAGGED CONSISTENCY

To analyze the performance penalty incurred by the proposed consistency scheme, we compare the proposed approach

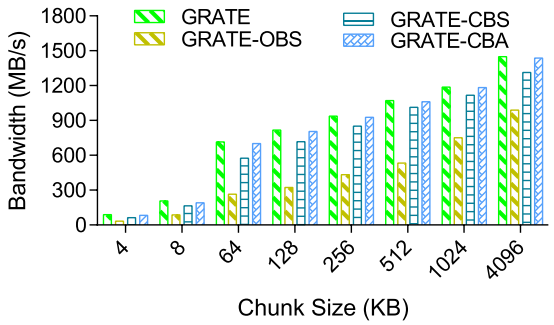


FIGURE 12. GRATE flag-assisted consistency performance.

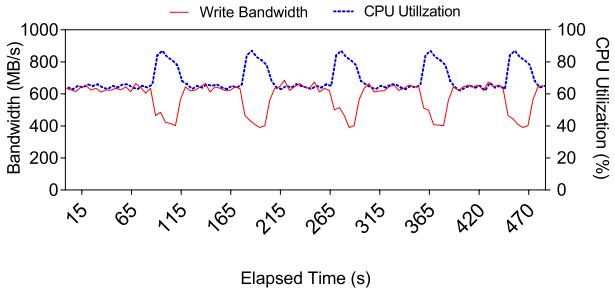


FIGURE 13. Garbage collection impact on performance and CPU utilization in CAO-Dedup.

(GRATE-CBA) with three other implementation variants. First, GRATE without any consistency flag mechanism, i.e., system becomes inconsistent in case of failures. Second, unlike the proposed approach, where we manage the consistency flag in CIT table. This approach manages the consistency flag in OMAP table and update the flag synchronously at transaction completion and refer to this approach as (GRATE-OBS). The benefit of keeping the flag in OMAP is magnitude lower number of I/Os to update the consistency flag. Third, we store and manage consistency flag against in CIT table and update the flag synchronously and refer to this approach as (GRATE-CBS). This approach requires an I/O to update the consistency flag synchronously causing higher number of I/Os compared to GRATE-OBS.

Figure 12 shows the bandwidth of different consistency variants when employed. We see that, when chunk size is small, the performance is poor in both chunk and object-based synchronous consistency compared to proposed GRATE-CBA. However, when we increase the chunk size, the performance improves. The chunk-based consistency shows high performance overhead as compared to others. It is due to additional higher number of I/Os required to switch the flags. Whereas, GRATE-OBS shows fair performance because only a single I/O is required to switch the flag but it still degrades the performance more than 15% compared to baseline GRATE. On the other hand, the GRATE-CBA incurs negligible overhead compared to chunk and object-based synchronous consistency schemes. Because both of the synchronous approaches introduce a transaction lock which increases the I/O latency, whereas the proposed approach switches the consistency flag

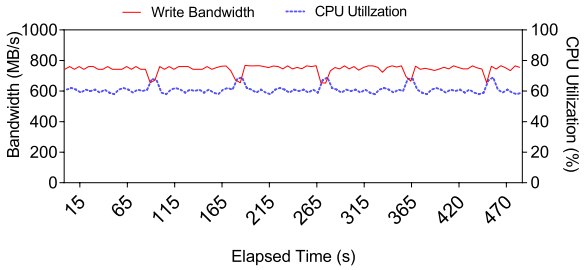


FIGURE 14. Garbage collection impact on performance and CPU utilization in GRATE.

asynchronously without acquiring any transaction lock, thus no performance overhead is incurred.

E. GARBAGE COLLECTION OVERHEAD

To show the effectiveness of the proposed garbage identification and removal, we compare our approach against Ceph internal scrubbing mechanism [1]. As, the current content addressable approach, i.e., CAO-Dedup approach relies on Ceph scrubbing mechanism to remove garbage chunks, analogous to Fsck [1], [32]. Ceph scrubbing can be carried out in two ways, light scrubbing, and deep scrubbing [68]. Deep scrubbing reads the data and uses checksums to ensure data consistency and integrity, Whereas, the light scrubbing checks the object size and attributes stored within each placement group [68]. Therefore, to make a fair comparison, we consider only light scrubbing and invoke garbage collection after every 80 seconds. Figure 13 depicts the write performance of CAO-Dedup with scrubbing in progress. From the results, we observe that the when scrubbing in invoked, the CAO-Dedup approach reduces the performance and elevates the CPU utilization. It is because the scrubbing scheme has to analyze all the placement group catalogs and check all the attributes of both content addressable object and data chunks stored.

Whereas, Figure 14 depicts the write performance of our approach along with garbage collection in progress. Our approach outperforms the CAO-Dedup with integrated scrubbing approach. Because our approach only analyze the fingerprint entries with an invalid consistency flag. We also analyze the CPU utilization of our approach compared to CAO-Dedup when garbage collection is triggered. Figure 13 and 14 show the CPU utilization of each approach. We observe a similar pattern as in write performance, that CAO-Dedup has higher CPU consumption because of compute-intensive scanning of all placement groups catalog and metadata attribute matching for each object. However, there is another limitation in CAO-Dedup integrated with scrubbing approach, that it cannot remove the invalid, erroneous and corrupt reference count entries.

F. STORAGE EFFICIENCY

We conduct this experiment to show the storage space efficiency of proposed GRATE compared to local disk-based deduplication. To enable disk-based dedup, we configure Ceph cluster with BtrFS [53] as backend disk file system with

TABLE 2. Deduplication space savings in percentage.

	# of OSDs			
	1	2	4	8
GRATE: Cluster-wide Dedup	85	85	85	85
Disk-based Deduplication	85	77	65	61

TABLE 3. Recovery time (in seconds) with varying failed OSDs.

	# of Failed OSDs		
	1	2	4
Baseline Ceph	22.44	29.27	64.38
DB-Shard Dedup	19.43	24.17	56.38
GRATE	17.51	20.36	39.21

deduplication enabled. We use 100% deduplication ratio and report the results in Table 2. We observe that disk-based dedup storage efficiency decreases with increasing number of disks. It is because disks are not aware of each other and cannot identify the duplicates stored on other disks. Whereas, GRATE storage efficiency remains high irrespective of number of disks. Note that, Grate achieves up to 85% of space savings even with a high duplication ratio in the workload. The rest of the space saving is populated by an additional amount of deduplication metadata being generated while deduping data. Further, the size of this additional metadata grows proportionally as the chunk size becomes smaller.

G. STORAGE REBALANCING AND RECOVERY

In this experiment, we show the robustness of proposed GRATE, cluster-wide deduplication and its impact on existing rebalancing and recovery mechanism. We compare our approach with baseline Ceph and DB-Shard Dedup. For evaluation, we store 100GB of data with 80% deduplication ratio in the cluster and mark an OSD as failed OSD. Then, Ceph triggers the self-balancing and recovery mechanism. The experimental results are shown in Table 3. The results clearly depict the improved recovery time as taken by baseline Ceph and DB-Shard approach. To further strengthen our observation, we increase number of failed OSDs in the cluster by marking OSD down and out. Whenever an OSD is marked down and out in Ceph, the OSD is considered as failed OSD and cannot participate in data placement. This improved recovery time is due to less number of write operations in cluster because both DB-Shard and GRATE approaches write the unique chunks, whereas, baseline Ceph writes complete set of objects irrespective of deduplication ratio. However, difference between GRATE and DB-Shard Dedup is mainly derived from additional location metadata update I/Os in DB-Shard Dedup. Whereas, the proposed approach employs fingerprint-based location determination and requires no additional location update I/Os. The evaluation results conclude that, the proposed GRATE is highly robust and adopts to existing recovery mechanism. It also reduces the recovery time as compared to baseline.

VII. READ PERFORMANCE ANALYSIS

In this section, we discuss the read I/O performance of proposed cluster-wide deduplication and compare it with CAO-Dedup by varying deduplication ratio, and mixed read-write workloads. We use synthetic datasets generated via FIO [67]. Because FIO provides suitable controls for generation of read-write mixed workload and deduplication ratio. The motivation behind to use a mixed read-write workload is to clearly portray the performance penalty in different contention scenarios by the proposed deduplication architecture.

FIGURE 15. Read I/O performance analysis.

Figure 15 depicts the comparative analysis of GRATE with no read optimization, GRATE-RMO with conflict-free read I/O and CAO-Dedup approach. We set the chunk size to 4KB, deduplication ratio to 30%, and use 8 client threads. Then, we vary the read-write ratio to analyze the performance overhead caused by contention of deduplication metadata. Note that, we only generate 30% RMO objects for total workload. We see that CAO-Dedup shows higher performance than the proposed approaches in read I/Os. It is because CAO-Dedup approach uses the content addressable objects for all read requests, which are treated like normal Ceph objects [32] and offers Ceph inherited scalability for read I/Os. Whereas, GRATE degrades performance in mixed read-write workloads due to potential contention at DM-Shard. We clearly see that GRATE-RMO performs better than GRATE due to less contact points with DM-Shard but performs lower than CAO-Dedup. This lower performance is attributed to selective RMO objects, which gives us more space efficiency compared to CAO-Dedup approach but at the cost of read performance degradation.

VIII. RELATED WORK

In this section, we focus on well-known dedup approaches and compare against the proposed approach.

There are two state of the art design approaches used for data dedup in distributed storage systems. First, disk-based data dedup where each disk or storage server in the system is responsible for removing duplicates locally such as [37], [51]–[53], [59], [69]. The benefit of disk-based dedup is high performance. However, storage space efficiency is limited to local disk only and degrades with increasing number of disks/nodes in the cluster. The second design approach is global data dedup which can give maximum space savings as compared disk-based local deduplication but incurs certain performance overhead.

Several studies have been conducted on such global dedup [12], [13], [15], [49], [58]. Venti [12] employs central deduplication server which does not fit into shared-nothing architectures. HYDRAstor [21] can scale because it uses the

distributed content-addressable manifest object to maintain the reference list of each chunk. However, the latency can increase in HYDRAstor [21] when the number of objects increases because the content-addressable manifest object is stored and treated like a general object. Extreme Binning [20], SILO [17], ∑-Dedupe [16] and Probabilistic Deduplication [13] can remove duplicates from the cluster. However, the storage space efficiency is highly dependent on workload because they use different similarity and locality based algorithms to detect duplicates. Exact Deduplication [18], DeDe [15] and Boafft [23] share high similarity to our proposed design. But these studies require two level fingerprint check, i.e., first check fingerprint in local index partition, and then remote node index partition. Moreover, these studies target the backup and archival storage rather than primary storage.

Additionally, DeDe and Boafft forms a superchunk by aggregating multiple small chunks based on similarity prediction algorithm and reroute the request to respective storage server. Whereas, superchunk similarity cannot always make good decision. Other Deduplication approaches for primary storage systems, such as iDedup [49], DBLK [70], and I/O Deduplication [71], exploit different workload characteristics to attain a fair throughput and latency.

Besides, none of the existing studies consider the object relocation problem in cluster-scale deduplication which is triggered when storage is imbalanced [3]. The metadata consistency is also a critical factor to ensure deduplication system reliability [19], [30], [64], [72]. The inconsistent metadata in deduplication systems can cause data integrity issues such as reference count corruption and garbage data chunks [37], [73]. The proposed method in [73] increases the I/O latency by inline switching of flags per object and chunk for each transaction. One of the recent studies [24] proposed a container-based dedup framework equipped with group mark and sweep based garbage collection approach. This approach divides the underlying storage into small containers in a similar fashion to Ceph placement group concept [1], [43] and attach logs to each container to monitor all the writes and updates. After a certain time threshold, the garbage collection is triggered. Each log is scanned by threads for modified and updated chunks. This approach requires high monitoring and logging overhead. Moreover, every log entry cross-checking with storage contents degrades performance during garbage collection.

A recent study [32] shows the design trade-offs of global deduplication via content addressable approach in Ceph [1]. However, firstly [32] is not purely inline, rather, it is offline or lazy deduplication more suitable for hybrid storage. Secondly, it requires high metadata storage space compared to our approach. Thirdly, it relies on internal scrubbing of Ceph for garbage collection, which is unable to repair the reference count errors. The most recent work [31] targets cluster-wide deduplication in Ceph and uses the hash and deduplicated object approach, which is similar to the content addressable object adopted in [32]. Both [32] and [31] implements cluster-scale deduplication in Ceph and share high resemblance in their design from various aspects, i.e., i) both employ double hashing algorithm, i.e., CRUSH [44] to reach actual data object, ii) both keep hash of content in an additional Ceph object called Hash object, iii) both require every Ceph object to have an additional hash object, and iv) both use extended attributes for reference count and employ write lock to modify flag and reference count. On the contrary, we rely on DB partitioning to manage lookups and reference count. The deduplication metadata storage is magnitude smaller compared to [31], [32]. Crocus [39] a recent study proposed GPU-aware chunking and fingerprinting to mitigate the performance overhead of cluster-wide inline deduplication in Ceph hybrid storage. In this study, we propose to build cluster-wide data deduplication capable to remove duplicates across the cluster. The data chunk and deduplication metadata placement are conducted based on content generated fingerprint. We employ Asynchronous tagged consistency scheme to ensure the metadata and data consistency. Our distributed metadata design and consistency scheme enables us to efficiently identify and remove garbage data chunks which are the result of the partially failed transactions. Further, we eliminate the deduplication metadata bottleneck from read I/O path.

IX. CONCLUSION

This paper presents a robust fault-tolerant, cluster-wide deduplication framework for shared-nothing storage systems. We design and implement a distributed deduplication metadata shard approach that uses the content hash of chunks to minimize I/O broadcasting and object relocation problems. We propose a tagged consistency approach to recover reference errors and lost data chunks in case of failures. The distributed deduplication metadata and consistency approach enables effective garbage identification and removal of garbage data chunks. Further, the proposed contention-free read I/O eliminates the deduplication metadata bottleneck from read I/O path. We implement the proposed ideas in Ceph, a scale-out distributed shared-nothing storage system. The evaluation shows that the proposed approach supports high scalability with minimal performance overhead, high robustness, and fault tolerance.

REFERENCES

[1] 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. 7th Symp. Operating Syst. Design Implement.*, 2006, pp. 307–320.
[2] GLUSTER. *Storage For Your Cloud. Gluster*. Accessed: Oct. 23, 2020. [Online]. Available: http://www.gluster.org
[3] H. Tang, A. Gulbeden, J. Zhou, W. Strathearn, T. Yang, and L. Chu, "A self-organizing storage cluster for parallel data-intensive applications," in *Proc. ACM/IEEE SC Conf.*, Nov. 2004, p. 52.
[4] A. Fekry, "Big data gets bigger: What about data cleaning analytics as a storage service?" in *Proc. 9th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*. Santa Clara, CA, USA: USENIX Association, 2017.
[5] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta, "Primary data deduplication-large scale study and system design," in *Proc. USENIX Conf. Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2012, p. 26. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342821.2342847

[6] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, and L. Xu, "Application-aware local-global source deduplication for cloud backup services of personal storage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1155–1165, May 2014.

[7] X. Zhao, Y. Zhang, Y. Wu, K. Chen, J. Jiang, and K. Li, "Liquid: A scalable deduplication file system for virtual machine images," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 5, pp. 1257–1266, May 2014.

[8] J. Wu, Y. Hua, P. Zuo, and Y. Sun, "Improving restore performance in deduplication systems via a cost-efficient rewriting scheme," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 119–132, Jan. 2019.

[9] J. Kaiser, T. Suss, L. Nagel, and A. Brinkmann, "Sorted deduplication: How to process thousands of backup streams," in *Proc. 32nd Symp. Mass Storage Syst. Technol. (MSST)*, 2016, pp. 1–14.

[10] W. Xia, Y. Zhou, H. Jiang, D. Feng, Y. Hua, Y. Hu, Y. Zhang, and Q. Liu, "FastCDC: A fast and efficient content-defined chunking approach for data deduplication," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2016, pp. 101–114. [Online]. Available: http://dl.acm.org/citation.cfm?id=3026959.3026969

[11] K. Kim, J. Kim, C. Min, and Y. I. Eom, "Content-based chunk placement scheme for decentralized deduplication on distributed file systems," in *Proc. 13th Int. Conf. Comput. Sci. Appl.*, vol. 1. Berlin, Germany: Springer-Verlag, 2013, pp. 173–183.

[12] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in *Proc. Conf. File Storage Technol.* Berkeley, CA, USA: USENIX Association, 2002, pp. 89–101. [Online]. Available: http://dl.acm.org/citation.cfm?id=645371.651321

[13] D. Frey, A.-M. Kermarrec, and K. Kloudas, "Probabilistic deduplication for cluster-based storage systems," in *Proc. 3rd ACM Symp. Cloud Comput. (SoCC)*. New York, NY, USA: ACM, 2012, pp. 17:1–17:4, doi: 10.1145/2391229.2391246.

[14] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proc. 6th USENIX Conf. File Storage Technol.* Berkeley, CA, USA: USENIX Association, 2008, pp. 18:1–18:14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1364813.1364831

[15] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized deduplication in san cluster file systems," in *Proc. Conf. USENIX Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2009, pp. 101–114. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855807.1855815

[16] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in *Proc. 13th Int. Middleware Conf.* New York, NY, USA: Springer-Verlag, 2012, pp. 354–373. [Online]. Available: http://dl.acm.org/citation.cfm?id=2442626.2442649

[17] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2011, pp. 26–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002207

[18] J. Kaiser, D. Meister, A. Brinkmann, and S. Effert, "Design of an exact data deduplication cluster," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol. (MSST)*, Apr. 2012, pp. 1–12.

[19] M. Lu, D. Chambliss, J. Glider, and C. Constantinescu, "Insights for data reduction in primary storage: A practical analysis," in *Proc. 5th Annu. Int. Syst. Storage Conf. (SYSTOR)*, 2012, pp. 17:1–17:7, doi: 10.1145/2367589.2367606.

[20] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Proc. IEEE Int. Symp. Modeling, Anal. Simulation Comput. Telecommun. Syst.* Washington, DC, USA: IEEE Computer Society, Sep. 2009, pp. 1–9.

[21] 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 *Proc. 7th USENIX Conf. File Storage Technol.* San Francisco, CA, USA: USENIX Association, 2009, pp. 197–210.

[22] *EMC Data Domain Global Deduplication Array*. Accessed: Mar. 1, 2020. [Online]. Available: https://www.dellemc.com/en-us/collaterals/unauth/white-papers/products/storage/h12209-vnx-deduplication-compression-wp.pdf

[23] S. Luo, G. Zhang, C. Wu, S. Khan, and K. Li, "Boafft: Distributed deduplication for big data storage in the cloud," *IEEE Trans. Cloud Comput.*, early access, Dec. 23, 2015, doi: 10.1109/TCC.2015.2511752.

[24] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2011, p. 25. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002181.2002206

[25] K. Eshghi, M. Lillibridge, D. Bhagwat, and M. Watkins, "Improving multi-node deduplication performance for interleaved data via sticky-auction routing," HP Lab., Tech. Rep. HPL-2015-77, 2015.

[26] M. Ajdari, P. Park, D. Kwon, J. Kim, and J. Kim, "A scalable HW-based inline deduplication for SSD arrays," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 47–50, Jan. 2018.

[27] H. Sim, Y. Kim, S. S. Vazhkudai, G. R. Vallée, S.-H. Lim, and A. R. Butt, "Tagit: An integrated indexing and search service for file systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.* New York, NY, USA: ACM, Nov. 2017, pp. 1–12, doi: 10.1145/3126908.3126929.

[28] A. Khan, T. Kim, H. Byun, and Y. Kim, "SciSpace: A scientific collaboration workspace for geo-distributed HPC data centers," *Future Gener. Comput. Syst.*, vol. 101, pp. 398–409, Dec. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18326025

[29] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *Proc. 9th USENIX Conf. File Stroage Technol.* Berkeley, CA, USA: USENIX Association, 2011, p. 2. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960477

[30] X. Lin, F. Douglis, J. Li, X. Li, R. Ricci, S. Smaldone, and G. Wallace, "Metadata considered harmful. . .to deduplication," in *Proc. 7th USENIX Workshop Hot Topics Storage File Systems (HotStorage)*. Santa Clara, CA, USA: USENIX Association, 2015, pp. 1–5.

[31] J. Wang, Y. Wang, H. Wang, K. Ye, C. Xu, S. He, and L. Zeng, "Towards cluster-wide deduplication based on Ceph," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Aug. 2019, pp. 1–8.

[32] M. Oh, S. Park, J. Yoon, S. Kim, K.-W. Lee, S. Weil, H. Y. Yeom, and M. Jung, "Design of global data deduplication for a scale-out distributed storage system," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 1063–1073.

[33] D. Meister, A. Brinkmann, and T. Süß, "File recipe compression in data deduplication systems," in *Proc. 11th USENIX Conf. File Storage Technol.* San Jose, CA, USA: USENIX, 2013, pp. 175–182.

[34] D. Harnik, E. Khaitzin, and D. Sotnikov, "Estimating unseen deduplication—From theory to practice," in *Proc. 14th USENIX Conf. File Storage Technol.* Santa Clara, CA, USA: USENIX Association, 2016, pp. 277–290.

[35] H. Wu, C. Wang, K. Lu, Y. Fu, and L. Zhu, "One size does not fit all: The case for chunking configuration in backup deduplication," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 213–222.

[36] A. Khan, C.-G. Lee, P. Hamandawana, S. Park, and Y. Kim, "A robust fault-tolerant and scalable cluster-wide deduplication for shared-nothing storage systems," in *Proc. IEEE 26th Int. Symp. Modeling, Anal., Simulation Comput. Telecommun. Syst. (MASCOTS)*, Sep. 2018, pp. 87–93.

[37] Z. Chen and K. Shen, "OrderMergeDedup: Efficient, failure-consistent deduplication on flash," in *Proc. 14th Usenix Conf. File Storage Technol.*, 2016, pp. 291–299.

[38] F. Douglis, A. Duggal, P. Shilane, T. Wong, S. Yan, and F. Botelho, "The logic of physical garbage collection in deduplicating storage," in *Proc. 15th USENIX Conf. File Storage Technol.* Santa Clara, CA, USA: USENIX Association, 2017, pp. 29–44.

[39] P. Hamandawana, A. Khan, C.-G. Lee, S. Park, and Y. Kim, "Crocus: Enabling computing resource orchestration for inline cluster-wide deduplication on scalable storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1740–1753, Aug. 2020.

[40] H. Song, X.-H. Sun, and Y. Chen, "A hybrid shared-nothing/shared-data storage architecture for large scale databases," in *Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.* Washington, DC, USA: IEEE Computer Society, May 2011, pp. 616–617, doi: 10.1109/CCGrid.2011.78.

[41] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proc. 21st ACM SIGOPS Symp. Operating Syst. Princ.* New York, NY, USA: ACM, 2007, pp. 205–220, doi: 10.1145/1294261.1294281.

[42] F. Wang, S. Oral, G. Shipman, O. Drokin, T. Wang, and I. Huang, "Understanding lustre filesystem internals," Oak Ridge Nat. Lab., Oak Ridge, TN, USA, Tech. Rep. ORNL/TM-2009/117, 2009.

[43] F. Wang, M. Nelson, S. Oral, S. Atchley, S. Weil, B. W. Settlemyer, B. Caldwell, and J. Hill, "Performance and scalability evaluation of the Ceph parallel file system," in *Proc. 8th Parallel Data Storage Workshop (PDSW)*, 2013, pp. 14–19.

[44] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Grid resource management—CRUSH: Controlled, scalable, decentralized placement of replicated data," in *Proc. ACM/IEEE Conf. Supercomputing*. New York, NY, USA: ACM, 2006, p. 122-es, doi: 10.1145/1188455.1188582.
[45] A. Aghayev, S. Weil, M. Kuchnik, M. Nelson, G. R. Ganger, and G. Amvrosiadis, "File systems unfit as distributed storage backends: Lessons from 10 years of Ceph evolution," in *Proc. SOSP*, 2019, pp. 353–369.
[46] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proc. IEEE*, vol. 104, no. 9, pp. 1681–1710, Sep. 2016.
[47] J. Ma, R. J. Stones, Y. Ma, J. Wang, J. Ren, G. Wang, and X. Liu, "Lazy exact deduplication," in *Proc. 32nd Symp. Mass Storage Syst. Technol. (MSST)*, 2016, pp. 1–10.
[48] C. Li, P. Shilane, F. Douglis, H. Shim, S. Smaldone, and G. Wallace, "Nitro: A capacity-optimized SSD cache for primary storage," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2014, pp. 501–512. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643686
[49] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, pp. 1–4.
[50] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in *Proc. 9th USENIX Conf. File Stroage Technol.* Berkeley, CA, USA: USENIX Association, 2011, p. 1. [Online]. Available: http://dl.acm.org/citation.cfm?id=1960475.1960476
[51] EMC. *Introduction to the EMC XtremIO Storage Array (Ver. 4.0)*. Accessed: Aug. 3, 2018. [Online]. Available: https://www.dellemc.com/el-gr/collaterals/unauth/white-papers/products/storage-2/h16444-introduction-xtremio-x2-storage-array-wp.pdf
[52] PureStorage. *The Industry Best Data Reduction, Hands Down*. Accessed: Mar. 1, 2020. [Online]. Available: https://www.purestorage.com/uk/products/purity/purity-reduce.html
[53] *Btrfs Wiki*. Accessed: Sep. 3, 2018. [Online]. Available: https://btrfs.wiki.kernel.org/index.php/Main_Page
[54] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy elimination within large collections of files," in *Proc. Annu. Conf. USENIX Annu. Tech. Conf.* Berkeley, CA, USA: USENIX Association, 2004, p. 5. [Online]. Available: http://dl.acm.org/citation.cfm?id=1247415.1247420
[55] P. T. Corporation. *PERMABIT*. Accessed: Oct. 23, 2020. [Online]. Available: http://permabit.com
[56] Y. Lu, J. Shu, and W. Zheng, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *Proc. 11th USENIX Conf. File Storage Technol.* Berkeley, CA, USA: USENIX Association, 2013, pp. 257–270.
[57] H. P. Enterprise. *HPE 3PAR StoreServ Storage*. Accessed: Mar. 1, 2020. [Online]. Available: https://www.hpe.com/us/en/storage/3par.html
[58] J. Wang, Z. Zhao, Z. Xu, H. Zhang, L. Li, and Y. Guo, "I-sieve: An inline high performance deduplication system used in cloud storage," *Tsinghua Sci. Technol.*, vol. 20, no. 1, pp. 17–27, Feb. 2015.
[59] SolidFire. *How Solidfire Data Efficiencies Work*. Accessed: Mar. 1, 2020. [Online]. Available: https://www.netapp.com/us/media/ds-solidfire-data-efficiencies-breif.pdf
[60] *Clustered Data on Tap*. Accessed: Mar. 1, 2020. [Online]. Available: https://www.netapp.com/us/media/tr-4476.pdf
[61] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, and Q. Liu, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *Proc. USENIX Annu. Tech. Conf.* Philadelphia, PA, USA: USENIX Association, 2014, pp. 181–192.
[62] X. Du, W. Hu, Q. Wang, and F. Wang, "ProSy: A similarity based inline deduplication system for primary storage," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*. Boston, MA, USA: IEEE, Aug. 2015, pp. 195–204.
[63] Z. Sun, N. Xiao, F. Liu, and Y. Fu, "DS-dedupe: A scalable, low network overhead data routing algorithm for inline cluster deduplication system," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*. Honolulu, HI, USA: IEEE, Feb. 2014, pp. 895–899.
[64] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," *ACM Trans. Storage*, vol. 3, no. 3, p. 9, Oct. 2007, doi: 10.1145/1288783.1288788.
[65] A. El-Shimi, R. Kalach, A. Kumar, A. Oltean, J. Li, and S. Sengupta, "Primary data deduplication—Large scale study and system design," in *Proc. USENIX Conf. Annu. Tech. Conf.* Berkeley, CA, USA, 2012, pp. 285–296.
[66] *SQLite: SQLite Home Page*. Accessed: Oct. 23, 2020. [Online]. Available: https://www.sqlite.org/
[67] J. Axboe. *Flexible I/O Tester*. Accessed: Oct. 23, 2020. [Online]. Available: https://github.com/axboe/fio
[68] *Ceph Architecture: Scrubbing Documentation*. Accessed: Oct. 23, 2020. [Online]. Available: http://docs.ceph.com/docs/master/architecture/?highlight=scrub
[69] B. K. Debnath, S. Sengupta, and J. Li, "Chunkstash: Speeding up inline storage deduplication using flash memory," in *Proc. USENIX Annu. Tech. Conf.*, 2010, pp. 1–16.
[70] Y. Tsuchiya and T. Watanabe, "Dblk: Deduplication for primary block storage," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, May 2011, pp. 1–5.
[71] R. Koller and R. Rangaswami, "I/O deduplication: Utilizing content similarity to improve I/O performance," *ACM Trans. Storage*, vol. 6, no. 3, pp. 13:1–13:26, 2010.
[72] Z. Sun, G. Kuenning, S. Mandal, P. Shilane, V. Tarasov, N. Xiao, and K. E. Zadok, "A long-term user-centric analysis of deduplication patterns," in *Proc. 32nd Symp. Mass Storage Syst. Technol. (MSST)*, 2016, pp. 1–7.
[73] A. Khan, C. Lee, S. Park, and Y. Kim, "Tagged consistency and garbage identification in deduplication-enabled storage systems," in *Proc. 16th (WiP) USENIX FAST*. Oakland, CA, USA: USENIX Association, 2018, pp. 1–2.

AWAIS KHAN (Member, IEEE) received the B.S. degree in bioinformatics from Mohammad Ali Jinnah University, Islamabad, Pakistan. He is currently pursuing the M.S. and Ph.D. degrees (integrated program) with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. He was with Digital Research Laboratories as a Software Engineer from 2012 to 2015. He is currently a member with the Laboratory for Advanced System Software, Department of Computer Science and Engineering, Sogang University. His research interests include cloud computing, cluster-scale deduplication, and parallel and distributed file systems.

PRINCE HAMANDAWANA received the B.Sc. degree (Hons.) in computer science with the National University of Science and Technology, Bulawayo, Zimbabwe. He is currently pursuing the M.S. and Ph.D. degrees (integrated program) with Ajou University, Suwon, South Korea. He had the privilege to work on some of the Zimbabwean leading service providers, including Econet Wireless from 2008 to 2011 and Liquid Telecom from 2011 to 2016. He is currently a member of the Database and Dependable Systems Laboratory, Department of Computer Engineering, Ajou University. His research interests include distributed and parallel storage systems and GPU-assisted cluster-wide data deduplication.

YOUNGJAE KIM received the B.S. degree in computer science from Sogang University, Seoul, Republic of Korea, in 2001, the M.S. degree from KAIST in 2003, and the Ph.D. degree in computer science and engineering from Penn State University, University Park, PA, USA, in 2009. He was a Staff Scientist with the Oak Ridge National Laboratory, U.S. Department of Energy, from 2009 to 2015 and an Assistant Professor with Ajou University, Suwon, Republic of Korea, from 2015 to 2016. He is currently an Associate Professor with the Department of Computer Science and Engineering, Sogang University. His research interests include distributed file and storage, parallel I/O, operating systems, emerging storage technologies, and performance evaluation.