A Robust Fault-Tolerant and Scalable Cluster-wide Deduplication for Shared-Nothing Storage Systems

Awais Khan[1], Chang-Gyu Lee[1], Prince Hamandawana[2], Sungyong Park[1], Youngjae Kim[1,*]
[1]Sogang University, [2]Ajou University, Republic of Korea
{awais, changgyu, parksy, youkim}@sogang.ac.kr, phamadawana@ajou.ac.kr

***Abstract*—Deduplication has been largely employed in distributed storage systems to improve 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. Further, deduplication introduces transactional changes, which are prone to errors in the event of a system failure, resulting in inconsistencies in data and deduplication metadata. In this paper, we propose a robust, fault-tolerant and scalable cluster-wide deduplication that can eliminate duplicate copies across the cluster. We design a distributed deduplication metadata shard which guarantees performance scalability while preserving the design constraints of shared-nothing storage systems. The placement of chunks and deduplication metadata is made cluster-wide based on the content fingerprint of chunks. To ensure transactional consistency and garbage identification, we employ a flag-based asynchronous consistency mechanism. We implement the proposed deduplication on Ceph. The evaluation shows high disk-space savings with minimal performance degradation as well as high robustness in the event of sudden server failure.**

I. Introduction

The shared-nothing storage systems (SN-SS) accommodate a large number of storage servers for high performance, scalability, availability, and fault-tolerance [10, 22]. SN-SS such as GlusterFS [10] and Ceph Object Storage [22] 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.

Deduplication (dedup) techniques are employed widely in storage systems to improve storage efficiency. There exist several studies on cluster-scale deduplication [4, 6, 8, 8, 9, 12, 14, 17, 24, 25]. However, direct adoption of such dedup techniques on the SN-SS violates the basic design constraints of SN-SS. For example, a centralized deduplication approach adopted in [2, 12, 17, 25] not only violates shared-nothing properties of SN-SS but also limits the scalability. On the other hand, a decentralized approach to distributing deduplication metadata across multiple servers [4, 6, 7, 8, 9, 11, 15, 24] requires specialized high performance dedup appliances for multiple deduplication servers.

Another alternative is to perform deduplication directly on storage nodes without any specialized hardware and rely on external distributed databases or key-value stores for storing deduplication metadata. Whereas, this alternate introduces an external database service dependency, which incurs high implementation complexity to amend I/O path. The system performance is also degraded due to such external database services.

In order to minimize such performance limiting factors, simple database partitioning (DB-Sharding) approach that embeds a single database partition (DB-Shard) of the whole dedup metadata database on each storage server has been proposed [12]. However, this DB-sharding approach to SN-SS suffers from inherited problems, i.e., to identify a duplicate chunk, the fingerprint lookup must be broadcasted to all DB-shards in the cluster, which poses a serious threat to scalability. Another challenging issue is deeply related to storage rebalancing. In SN-SS, the storage rebalancing is triggered whenever a change such as adding or deleting a storage server in cluster occurs. It can also be triggered when there is a significant I/O load or space usage imbalance [20]. This rebalancing shuffles the chunks across the storage servers to evenly balance the space utilization in the cluster. In this case, deduplication metadata must be updated for the new location of the chunk in the cluster. However, this rebalancing incurs high metadata I/Os, because to keep track of chunk position 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 position changes. Figure 1(a)(b) illustrate these problems.

Deduplication also requires transactional level changes, where a complete object-based transaction is split into multiple small fixed or variable chunk-based transactions [12]. These changes, if not implemented carefully, can cause inconsistent data and dedup metadata in the cluster in an event of communication, disk or storage server failures. A recent study to address the consistency of reference counts is to use soft-update style metadata in a single disk-based file system [3]. However, it is not directly applicable to distributed nature of SN-SS, where parallel I/Os are responsible to distribute chunks. Another effect of transaction failures in deduplication storage systems is garbage chunks of failed transactions.

To address the above-mentioned challenges in SN-SS, we propose to build a robust, scalable and consistent cluster-wide deduplication framework for SN-SS. In particular, we have used chunk's content fingerprint to avoid lookup broadcast issue in DB-shard. The content fingerprint based placement determines the exact location of data chunks in the cluster, even if object shuffling occurred. We have employed a tagged

[*]Y. Kim is the corresponding author.

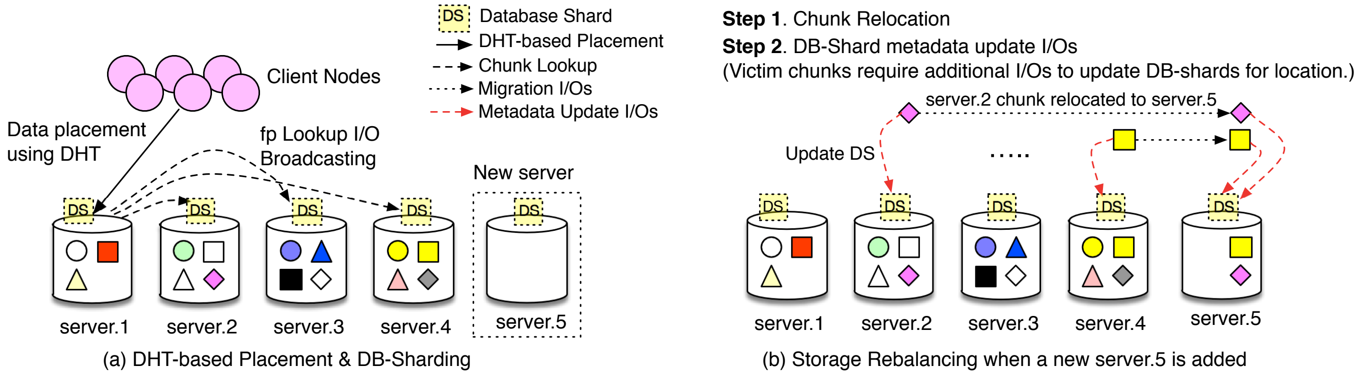

DS Database Shard
DHT-based Placement
Chunk Lookup
Migration I/Os
Metadata Update I/Os
Client Nodes
Data placement using DHT
fp Lookup I/O Broadcasting
New server
server.1
server.2
server.3
server.4
server.5
(a) DHT-based Placement & DB-Sharding
Step 1. Chunk Relocation
Step 2. DB-Shard metadata update I/Os
(Victim chunks require additional I/Os to update DB-shards for location.)
server.2 chunk relocated to server.5
Update DS
(b) Storage Rebalancing when a new server.5 is added

Fig. 1: (a) Traditional distributed DB-sharding approach and (b) storage rebalancing issues in SN-SS such as Ceph [22] and GlusterFS [10]. Specifically, (b) illustrates the chunk relocation when a new server is added to the cluster.

consistency approach to ensure the validity and correctness of deduplication metadata.

This paper has the following specific contributions:

- We use the content-generated fingerprint to distribute and locate the chunks in the cluster atop distributed and partitioned deduplication metadata. We employ database partitioning to handle deduplication metadata in a decentralized manner. The content fingerprint and distributed metadata enable us to preserve the shared-nothing property of SN-SS.
- We design asynchronous tagged consistency which ensures the correct status of the transaction, data and deduplication metadata. Moreover, our partitioned deduplication metadata and tagged consistency aid in identifying garbage chunks without additional monitoring and journaling.
- We have designed and implemented the proposed cluster-wide data deduplication in Ceph [22] and evaluate the proposed ideas in the real testbed.

II. Background and Motivation

The data deduplication increases the available storage capacity by eliminating duplicated data. The existing deduplication studies such as [4, 6, 9, 12, 14, 25] are focused on performance improvement and increasing disk space savings. Our target architecture is shared-nothing storage systems such as Ceph [22]. Ceph is a distributed object storage system that provides high performance, reliability, and scalability [22]. 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 devices (OSDs) [22, 23]. Ceph consists of object storage servers (OSS), monitors and clients. The logical pools are defined on storage servers and each pool is composed of multiple placement groups (PGs) which are configured based on available OSDs. Ceph stores and replicates objects at the granularity of a PG. When Ceph clients store an object, CRUSH computes the PG responsible to store object using logical pool name, object name hash, and modulo, total number of PGs [23].

Currently, Ceph lacks inline cluster-wide data deduplication. In order to design deduplication for Ceph, we need to follow SN-SS design constraints. A simple centralized deduplication in Ceph introduces the central dependency which breaks the no centralized metadata property of Ceph. The decentralized deduplication can address such design constraints. However, it poses several other challenges. For example, how can we accurately and efficiently find the duplicate contents in a Ceph storage cluster spanning over 100s of servers. A simple solution is to use a fix 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 storages like Ceph, the data chunks are relocated across the disk and storage servers to balance the storage utilization [20, 22]. Figure 1 depicts same case, where a new server is added and chunks are relocated to balance the storage utilization to 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).

Apart from self-rebalancing, such dedup approaches fail to scale with increasing cluster size. For example, in order to find duplicates, we need to check all the DB-Shards to find the duplicate fingerprints. Such duplicate fingerprint check latency is greatly impacted by number of nodes in cluster because of fingerprint broadcast overhead. Another challenge is to ensure the data and dedup metadata correctness and consistency along with garbages identification and removal. Overall, the motivation of this study is to design a cluster-wide deduplication which has low fingerprint lookup I/O overhead, and it can adapt to the node addition and removal seamlessly. We also consider it critical to solve dedup metadata inconsistencies in our cluster-wide dedup design. We propose distributed DB-shard approach to manage dedup metadata and adopt fingerprint-based I/O redirection to minimize I/O lookup broadcast overhead.

III. Cluster-wide Data Deduplication

A. Architecture Overview

The proposed cluster-wide deduplication is built on a shared-nothing distributed storage system. Figure 2 shows the architecture design of cluster-wide deduplication. Logically, the SN-SS is composed of clients, storage servers and no additional metadata servers and employs Distributed Hash

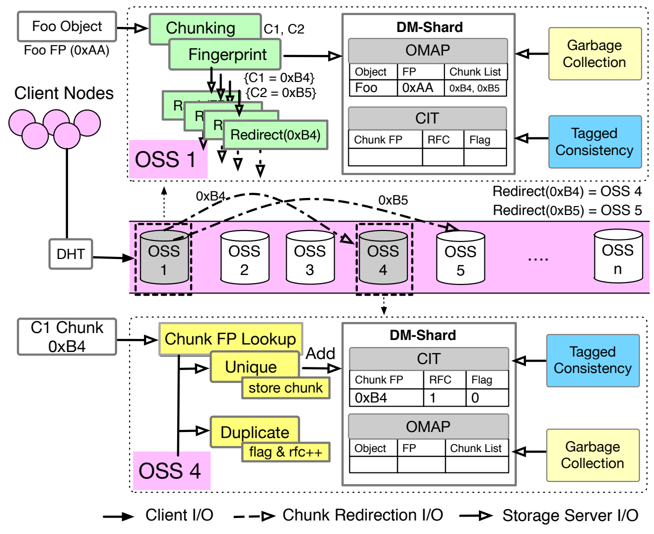


Fig. 2: Cluster-wide deduplication based on DB-sharding and content-fingerprint based placement in SN-SS.

Table (DHT) for data placement [10, 22]. The client performs object name hashing and locates 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 2), it is responsible for splitting the object into small fixed-size data chunks, computes the fingerprint for each chunk's content. Then, it redirects the data chunk to storage server based on the computed fingerprint (OSS 4 in Figure 2).

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 2 (OSS 1). We explain the DM-Shard in Section III-B. The redirected chunks received on other storage servers (OSS 4 in Figure 2) 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 commit 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. The data chunk is stored in the storage server and CIT entry is updated accordingly (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). OMAP 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 tagged 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 flag (Flag in CIT) are interpreted as garbage data chunks and collected periodically.

B. Deduplication Metadata Shard

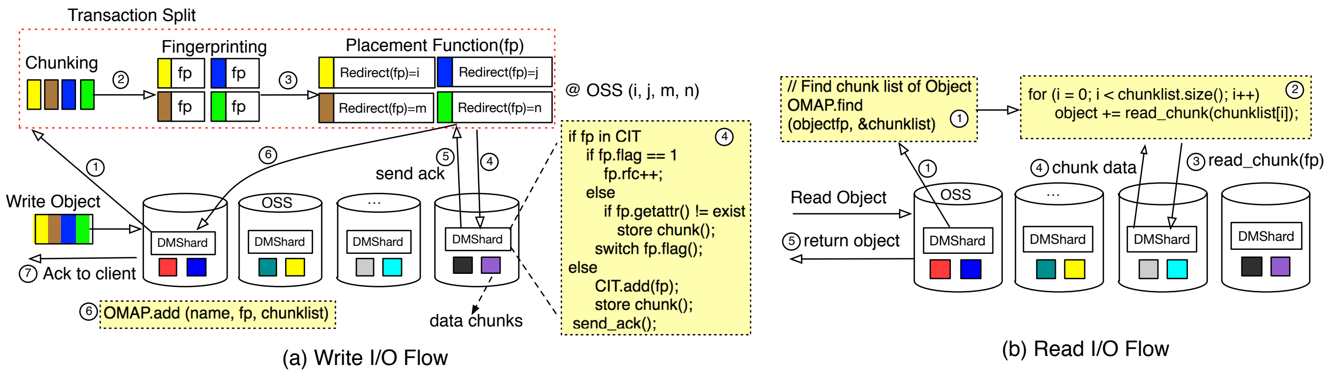
We build DM-Shard as shown in Figure 2 to effectively manage dedup metadata. The design decision to use distributed DM-Shard is to comply scalable and shared-nothing property of SN-SS. Every storage server in the cluster hosts a DM-Shard holding all the persistent data structures such as object layout information and data chunk fingerprint. Each shard keeps the unique information of objects and data chunks in a separate data structure, i.e., Object Map (OMAP) and Chunk Information Table (CIT).

- *Object Map (OMAP):* 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 2. In DHT-based storage systems, an object is identified by hashing the object name, and if we do not maintain the hash of object, we cannot reconstruct the original object because we need all the chunks' fingerprint created from this object. OMAP assists in read operations, where object fingerprint is given to lookup chunks belonging to a specific object.
- *Chunk Information Table (CIT):* CIT maintains the performance-sensitive deduplication metadata. It includes data chunk fingerprint, reference count, and commit flag. All the lookup and reference update operations are possible via this data structure.

The advantage to keep different data structures is manifold: i) to provide effective execution of fingerprint operation, 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 structures are updated 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 transactions with usage of OMAP and CIT in Figure 3. For deduplication metadata replication and fault-tolerance, we rely on SN-SS because we store our DM-Shard in the storage server and is replicated like a normal object.

C. Chunk Relocation and I/O Routing

SN-SS such as Ceph [22] and Gluster [10] distribute objects in a storage-balanced fashion. For instance, Ceph uses CRUSH algorithm [23] to fairly distribute the storage load across the storage servers, when the cluster topology changes, e.g., a new storage server is added or removed. The objects are relocated across the storage servers in order to balance the storage load in the cluster as shown in Figure 1(b). This object and chunk relocation process is neglected in all previous deduplication studies [6, 9, 12, 15]. In previous studies, the location of object and data chunks is stored along with metadata, i.e., *data chunk 1A is stored on server x and data chunk 1B is stored on server y*. This type of dedup metadata management suffers when chunks are relocated in the cluster because object and chunk location is lost. One solution can be; to transform current self-balancing mechanism to update the deduplication metadata



```
if fp in CIT
    if fp.flag == 1
        fp.rfc++;
    else
        if fp.getattr() != exist
            store chunk();
        switch fp.flag();
else
    CIT.add(fp);
    store chunk();
send_ack();
```

```
// Find chunk list of Object
OMAP.find
(objectfp, &chunklist)
```

```
for (i = 0; i < chunklist.size(); i++)
    object += read_chunk(chunklist[i]);
```

Fig. 3: A complete write and read I/O transaction in cluster-wide data deduplication system.

while relocating the objects and chunks, but it entails 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 related DM-Shard across the cluster, we use the data chunk fingerprint (step 3 in Figure 3(a)). 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 using object name or object fingerprint (read request). The computed-fingerprint tells the storage server location responsible for storing the actual data chunk and the metadata shard (CIT) (step 4 in Figure 3(a)). This content-based placement relieves us from i) complicated location management for each data chunk, ii) modifications in 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 for fingerprint lookup to all storage servers, instead we send a single lookup I/O to only a single storage server.

D. Asynchronous Tagged Consistency

The deduplication metadata inconsistencies in distributed storage systems lead to data authenticity and integrity issues. For example, if an object transaction is split into multiple chunk-based transactions, and one of the small transactions fails, the whole object transaction fails. Two problems are likely to happe: first, an invalid reference fingerprint in DB-Shard and second, garbage chunks left of the failed transaction. Worst of all, a new incoming duplicate fingerprint can increment the invalid reference entry, causing serious metadata inconsistency. Due to transactional modifications, a complicated transaction and rollback logic is required to make reference count consistent [11].

To address such consistency concern, we add a commit flag to each data chunk entry which specifies the consistency state of the chunk, i.e., *U* or *C*. The flag with U is invalid chunk (missing from storage or storage in progress) and C is valid chunk (available in storage). A simple approach is to add commit flag with object or chunk data structure and update the commit flag at each transaction completion time. However, this simple approach requires transaction lock and updates the flag synchronously which affects the scalability of the system. To bypass such transaction lock, we propose an asynchronous thread-based consistency manager which runs on every storage server. All the incoming write I/Os register to consistency manager. Once the I/O transaction completes, the consistency manager asynchronously updates the flag managed in CIT (Section III-B). If a crash occurs in the middle of a transaction when data chunk is stored and commit flag is not updated, then, the chunk will be marked as garbage due to invalid commit flag value because transaction partially failed. We explain the tagged consistency using two use cases.

Unique Write: In this case, the object splits into multiple small chunks and stores the chunk on different storage servers based on data chunk fingerprint. Each fingerprint in CIT holds an invalid flag by default, i.e., U. The consistency manager is notified of the received write operation. Once the I/O finishes, the flag is switched from U (invalid) to C (valid) asynchronously.

Duplicate Write: In duplicate write case, whenever a duplicate fingerprint wants to increment the reference count in CIT, it needs to check the flag as shown in Figure 3. The fingerprint entries with a valid or (C) flag allow the reference count increment or decrement operations. If the flag is invalid and reference update is required, the data chunk is required to perform an additional consistency check, to ensure the existence of data chunk in the storage server. We manage consistency check by simply getting data chunk attributes from the storage server just like a stat call in the file system. If the data chunk exists, we switch the flag to valid and conduct the reference operations. Otherwise, we first store the actual data chunk contents and then, switch the flag. This consistency check enables the presence of actual data and can repair the missing data chunks.

To claim free space consumed by garbage data chunks, we design and implement a garbage collection thread. The thread periodically collects the data chunk fingerprints with an invalid commit flag in CIT. It keeps the fingerprints for a pre-defined threshold. Once the threshold expires, the thread cross-matches the collected fingerprints to CIT. This cross-matching is required to assess any change, in particular to invalid fingerprints. If there is no change, then fingerprints along with data chunks are removed from the storage system. We do not use any additional journaling because it requires additional disk space. We claim that the proposed asynchronous consistency manager ensures the data and metadata accuracy even in case of failures and prevent the deduplication storage system from inconsistencies.

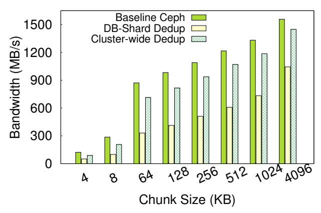
IV. Evaluation

Implementation: We implement the proposed cluster-wide deduplication in Ceph v10.2.3. The DM-Shard, consistency and garbage collecter are embedded in each OSD (Object Storage Daemon). We use the SHA-1 algorithm to generate a data chunk fingerprint and pass the fingerprint to the CRUSH algorithm [23] to distribute the data chunks in the Ceph storage cluster. The dedup operations such as 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 as backend storage for DM-Shard.

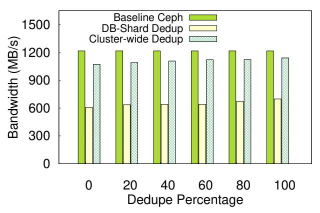
Testbed: We configured Ceph storage cluster on testbed consisting of 7 Object Storage Servers (OSSs), 3 Monitors and 4 Ceph client nodes connected via 10 Gbps network. Each machine is equipped with Intel E5-2670v4@2.40GHz (10 Cores), 32GB DRAM and 2x256GB Samsung SSDs per OSS running Linux CentOS v7.3. We used FIO [1] benchmark for evaluation by varying deduplication ratio and number of client threads with a 500GB synthetic write I/O workload. We compare the proposed cluster-wide deduplication (Cluster-wide Dedup) with Ceph with no deduplication (Baseline Ceph) and Ceph with deduplication implemented via simple DB-sharding (DB-Shard Dedup). The DB-Shard approach relies on storing the location of chunks in OMAP and it does not use content fingerprint-based I/O redirection. To check duplicates, it requires to broadcast fingerprint lookup I/Os to all the OSDs in the cluster.

Performance Analysis: To analyze the performance penalty incurred by the proposed cluster-wide dedup, we use synthetic datasets generated via FIO. To clearly observe the performance overhead, we set the duplicate ratio to 0% and use a total of 8 client threads in FIO benchmark. Figure 4(a) shows the bandwidth of all three approaches. Our proposed cluster-wide dedup scales as much as baseline Ceph with respect to the increased chunk size. Our proposed approach shows an average 18% of performance degradation compared to baseline Ceph whereas, DB-Shard Dedup shows an average of 40% degradation compared to the proposed approach. The reason of performance degradation in DB-Shard Dedup is because of sending out I/Os to all OSDs to verify the presence of duplicates. There is a certain performance overhead which is mainly derived from fingerprint computation and network transfer overhead for small chunk-sizes. The fingerprint overhead can be further minimized by employing hardware-accelerator such as GPU for parallel fingerprint computation.

Next, we discuss the performance of cluster-wide dedup with respect to deduplication ratio as shown in Figure 4(b). We set the chunk size to 512KB. We observe both DB-Shard Dedup and Cluster-wide Dedup show limited performances to certain thresholds regardless of deduplication rate. However,

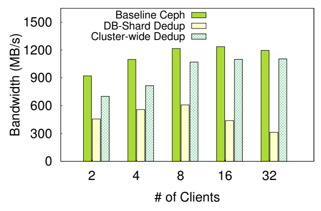


(a) Performance Analysis

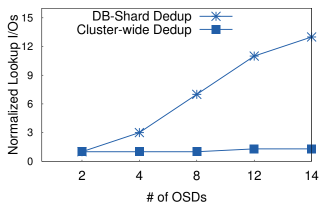


(b) Deduplication Ratio

Fig. 4: Performance analysis.



(a) Scalability with Multiple Clients



(b) Content Fingerprint-based Redirection

Fig. 5: Scalability analysis.

we see the cluster-wide dedup performance is nearly twice of DB-Shard Dedup. This improvement is basically due to scalable and distributed deduplication metadata management, which reduces the metadata I/O contention. We do not observe notable performance improvement with cluster-wide Dedup when dedup ratio varies because the data chunk I/Os are still directed over the network which are too small to show improvement if not stored on the storage server.

Scalability Analysis: To test the scalability, we vary the number of client threads on each Ceph client. In Figure 5(a), we tend to show the impact of I/O contention created by multiple client threads. We set the chunk size 512KB for this experiment. Figure 5(a) shows that, when the number of client threads is less, the cluster-wide dedup performance is not very high compared to DB-Shard Dedup. This is because DB-Shard Dedup server has less I/O broadcasting issue. However, with the increased number of client threads, DB-Shard Dedup further degrades the performance as compared to proposed Cluster-wide Dedup. It becomes worse when the number of client threads is 32, the DB-Shard dedup bandwidth degrades to near 300MB/s. This degradation is mainly derived from two factors: i) high metadata broadcast I/Os and ii) metadata contention on DB shards. Whereas, cluster-wide deduplication shows scalability and improves the bandwidth with increasing number of client threads because CRUSH [23] distributes the data chunks uniformly in a load-aware fashion to object storage servers and DM-Shard is distributed across all the OSSs, which overcomes the possible chances of dedup metadata contention.

Content Fingerprint-based I/O Redirection: 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 5(b) shows the fingerprint lookup I/O performance with respect to

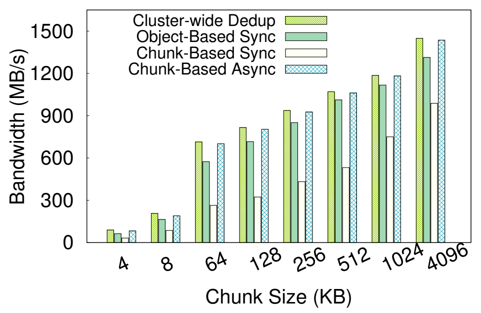


Fig. 6: Asynchronous Tagged Consistency analysis.

increasing number of OSDs. We observe from the results that the Cluster-wide Dedup approach 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.

Asynchronous Tagged Consistency: In chunk-based consistency, the flag is managed for each data chunk fingerprint, i.e., in CIT whereas in object-based consistency, the flag is stored at object granularity, i.e., in OMAP. Figure 6 shows the bandwidth of different variant 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 asynchronous chunk-based tagged consistency. However, increase in chunk size reduces the performance degradation. The chunk-based consistency shows high performance overhead as compared to others. It is due to additional serialized high number of I/Os required to switch flags. Whereas, object-based sync consistency shows fair performance because only a single I/O is required to switch the flag but still degrades the performance more than 15% compared to baseline cluster-wide dedup. On the other hand, the asynchronous tagged consistency incurs negligible overhead compared to chunk and object-based sync consistency. Because both chunk and object sync approaches introduce a transaction lock which increases the I/O latency, whereas our approach switches the commit flag asynchronously without acquiring any transaction lock, hence no overhead is incurred.

V. Related Work

Deduplication is widely adopted in storage systems to improve the space efficiency while effectively reducing the storage cost [12, 14]. There are two state of the art design approaches used for deduplication in shared-nothing 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 [7, 16, 18]. 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 dedup but incurs performance overhead.

Several studies have been conducted on such global dedup [4, 8, 17, 19, 21]. Venti [17] employs a central dedup server which does not fit into shared-nothing architectures. HYDRAstor [6] 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 [6] when the number of files/objects increases because the content-addressable manifest object is stored like a normal object. However, we aim to design a scalable cluster-wide deduplication with no central dependency. Extreme Binning [2], SILO [24], $\sum$-Dedupe [9] and Probabilistic Deduplication [8] 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 [12], DeDe [4] and Boafft [15] 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.

DeDe [4] and Boafft [15] form 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. Also, failure of such superchunk can increase the garbages. Besides, none of the existing studies consider the object relocation problem in cluster-scale dedup which is triggered when storage is imbalanced [20]. The metadata consistency is also a critical factor to ensure dedup system reliability [5, 13, 14]. The inconsistent metadata in deduplication systems can cause data integrity issues such as reference count corruption and garbage data chunks [3]. The syhchronous tagged consistency increases the I/O latency by inline switching of flags per object and chunk for each transaction.

In this study, we propose to build a decentralized data deduplication framework capable of removing duplicates across the cluster. The data chunk and metadata placement is conducted based on content generated fingerprint. We use tagged-consistency to ensure the metadata and data consistency.

VI. Concluding Remarks

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 avoid I/O broadcasting and dynamic object relocation problems. We also propose a tagged consistency approach which can recover reference errors and lost data chunks in case of sudden storage server failures. We implement the proposed ideas in Ceph. Evaluation shows that proposed approaches support high scalability with minimal performance overhead and high robust fault tolerance.

Acknowledgement

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2014-0-00035).

References

[1] Axboe, J. Flexible i/o tester. https://github.com/axboe/fio.

[2] Bhagwat, D., Eshghi, K., Long, D. D. E., and Lillibridge, M. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *Proceedings of the 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (2009), MASCOTS '09.

[3] Chen, Z., and Shen, K. Ordermergededup: Efficient, failure-consistent deduplication on flash. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies* (2016), FAST'16.

[4] Clements, A. T., Ahmad, I., Vilayannur, M., and Li, J. Decentralized deduplication in san cluster file systems. In *Proceedings of the 2009 Conference on USENIX Annual Technical Conference* (2009), ATC'09.

[5] Douglis, F., Duggal, A., Shilane, P., Wong, T., Yan, S., and Botelho, F. The logic of physical garbage collection in deduplicating storage. In *15th USENIX Conference on File and Storage Technologies* (2017), FAST'17.

[6] Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., and Welnicki, M. Hydrastor: A scalable secondary storage. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies* (2009), FAST'09.

[7] EMC. EMC data domain global deduplication array. http://www.datadomain.com/products/global-deduplication-array.html.

[8] Frey, D., Kermarrec, A.-M., and Kloudas, K. Probabilistic deduplication for cluster-based storage systems. In *Proceedings of the Third ACM Symposium on Cloud Computing* (2012), SoCC'12.

[9] Fu, Y., Jiang, H., and Xiao, N. A scalable inline cluster deduplication framework for big data protection. In *Proceedings of the 13th International Middleware Conference* (2012), Middleware'12.

[10] GLUSTER. Storage For Your Cloud. Gluster. http://www.gluster.org.

[11] Guo, F., and Efstathopoulos, P. Building a high-performance deduplication system. In *Proceedings of the USENIX Conference on USENIX Annual Technical Conference* (2011), ATC'11.

[12] Kaiser, J., Meister, D., Brinkmann, A., and Effert, S. Design of an exact data deduplication cluster. In *Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies* (2012), MSST'12.

[13] Lin, X., Douglis, F., Li, J., Li, X., Ricci, R., Smaldone, S., and Wallace, G. Metadata considered harmful…to deduplication. In *7th USENIX Workshop on Hot Topics in Storage and File Systems*, HotStorage'15.

[14] Lu, M., Chambliss, D., Glider, J., and Constantinescu, C. Insights for data reduction in primary storage: A practical analysis. In *Proceedings of the 5th Annual International Systems and Storage Conference* (2012), SYSTOR'12.

[15] Luo, S., Zhang, G., Wu, C., Khan, S., and Li, K. Boafft: Distributed deduplication for big data storage in the cloud. *IEEE Transactions on Cloud Computing* (2015).

[16] PureStorage. The industry best data reduction, hands down. https://www.purestorage.com/products/purity/flash-reduce.html.

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

[18] SolidFire. How solidfire data efficiencies work. http://info.solidfire.com/rs/538-SKP-058/images/SolidFire-Data-Efficiencies-Breif.pdf.

[19] Srinivasan, K., Bisson, T., Goodson, G., and Voruganti, K. iDedup: Latency-aware, inline data deduplication for primary storage. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (2012), FAST'12.

[20] Tang, H., Gulbeden, A., Zhou, J., Strathearn, W., Yang, T., and Chu, L. A self-organizing storage cluster for parallel data-intensive applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2004), SC'04.

[21] Wang, J., Zhao, Z., Xu, Z., Zhang, H., Li, L., and Guo, Y. I-sieve: An inline high performance deduplication system used in cloud storage. *Tsinghua Science and Technology 20* (2015).

[22] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation* (2006), OSDI'06.

[23] Weil, S. A., Brandt, S. A., Miller, E. L., and Maltzahn, C. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2006), SC'06.

[24] Xia, W., Jiang, H., Feng, D., and Hua, Y. Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference* (2011), ATC'11.

[25] Zhu, B., Li, K., and Patterson, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), FAST'08.