electronics

Article

Speed-Dedup: A New Deduplication Framework for Enhanced Performance and Reduced Overhead in Scale-Out Storage

Prince Hamandawana, Da-Jung Cho and Tae-Sun Chung *

Deparment of Software, Ajou University, Suwon 16499 , Republic of Korea; phamandawana@ajou.ac.kr (P.H.); dajungcho@ajou.ac.kr (D-J.C.)
* Correspondence: tschung@ajou.ac.kr

Abstract: Conventional deduplication systems face critical challenges such as excessive write amplification, high read/write latency, and sub-optimal storage utilization. These limitations often undermine the performance benefits of deduplication by slowing down I/O acknowledgements due to amplified deduplication I/Os, excessive data chunk replication, and strict consistency requirements. To address these issues, we present Speed-Dedup, a novel deduplication framework that employs a deduplicated primary–semi-deduplicated replica object approach. This strategy reduces write amplification by restricting deduplication to the primary object while maintaining a semi-deduplicated replica object used for immediate read/write acknowledgements, thus enhancing I/O latency and storage efficiency. Speed-Dedup also replaces traditional strong consistency models with eventual consistency, allowing for non-blocking read operations and improving overall system throughput. Experimental results demonstrate that Speed-Dedup significantly outperforms traditional methods like GRATE and CAO, showing up to 21% improvement in I/O performance under low deduplication ratios and maintaining 14% or more gains under higher ratios. Additionally, write amplification is substantially reduced and latency improves by over 100% with faster recovery times during system failures. These findings highlight the effectiveness of Speed-Dedup as a scalable and efficient solution.

Keywords: data deduplication; distributed storage system; scale-out storage; fault tolerance; write amplification

Citation: Hamandawana, P.; Cho, D-J.; Chung, T-S. Speed-Dedup: A New Deduplication Framework for Enhanced Performance and Reduced Overhead in Scale-Out Storage. *Electronics* **2024**, *13*, 4393. https://doi.org/10.3390/electronics13224393

Academic Editor: Stefanos Kollias

Received: 8 October 2024
Revised: 6 November 2024
Accepted: 7 November 2024
Published: 9 November 2024

1. Introduction

The rapid growth of data in cloud environments has led to increasing storage costs, typically measured in terms of dollars per gigabyte (USD/GB) [1–6]. To address this challenge, data deduplication has become an essential feature of modern cloud and enterprise storage systems. Data deduplication optimizes storage usage by eliminating redundant data, ensuring that only unique data are stored while duplicates are referenced to the original data [7–11]. Deduplication can be implemented through either local deduplication [12], where duplicate elimination occurs within isolated "silos" of individual storage servers, or global deduplication [9,10,13,14], which executes duplicate elimination across all storage servers. The global deduplication approach significantly reduces storage requirements compared to local deduplication, yielding notable cost savings.

It is also imperative to note that most cloud storage solutions follows a shared nothing storage (SNS) architecture, where the failure of one node does not affect the functions of the other nodes. Also due to data replication, the SNS nodes are inherently autonomous, self-healing, and highly redundant. These characteristics make SNS systems particularly well-suited for cloud storage environments. This shared nothing property brings about the scalability in the cloud storage. In this study, we refer to these SNS systems as scale-out storage systems. Consequently, we focus on global deduplication of these scale-out storage systems.

While global data deduplication effectively eliminates redundancy and reduces storage costs, it is associated with significant performance degradation, particularly in terms of read/write (R/W) I/O latency. This performance decline is driven by several key drawbacks:

❶ **R/W I/O Amplification:** As shown in Figure 1, the data deduplication process introduces additional steps, including (i) chunking, where incoming data are split into smaller, fixed- or variable-sized chunks, (ii) hashing, where a hash value (or fingerprint) is generated for each chunk, (iii) redundancy checking, where the computed hash is used to determine whether a chunk is unique or a duplicate, and (iv) storing or referencing, where unique chunks are stored and duplicates are referenced to pre-existing chunks, updating the reference count in the deduplication metadata. These extra steps result in amplified I/O operations.

❷ **Degraded R/W Performance:** The additional overhead of deduplication increases I/O latency for both write and read operations. Write I/O performance suffers due to the need for chunking, hashing, redundancy checks, and disk commits for each chunk. Similarly, read I/O performance is negatively impacted by the reconstruction process, which requires retrieving and reassembling deduplicated data chunks before fulfilling client read requests [10,11,15].

❸ **I/O Redirections Across the Network:** When data are written to a storage server, they are chunked, and the chunks are rehashed to determine their storage location [9,10,13,16]. This rehashing introduces additional I/O redirection, contributing to increased overhead in deduplication-enabled storage environments, particularly in distributed systems.

❹ **Chunk Replication for Fault Tolerance:** Although deduplication reduces the overall storage footprint, the need for chunk replication to ensure fault tolerance can offset these gains. For example, when a client writes a new 4MB data object that is chunked into 1 MB segments, each chunk may be replicated multiple times (e.g., with a replication factor of three) across the cluster to maintain fault tolerance, as seen with replication algorithms like CRUSH [17]. As a result, the original 4 MB object results in 12 MB of stored data. In large-scale environments with millions of chunks, this replication overhead can negate the storage efficiency achieved through deduplication.

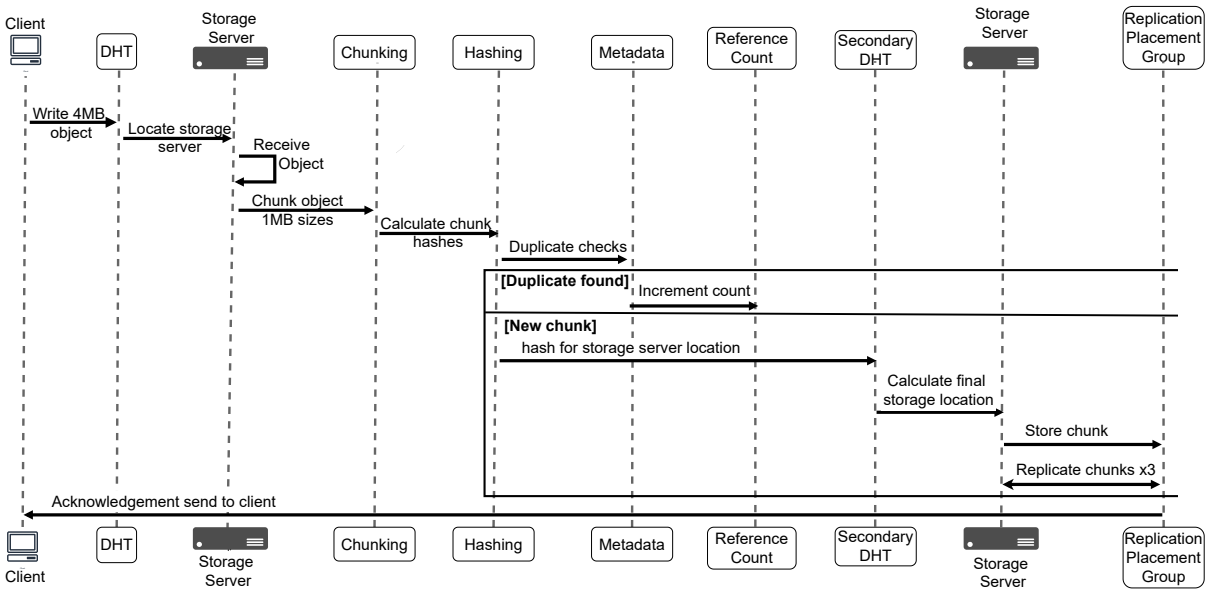


Figure 1. Deduplication timeline process in conventional scale-out storage.

Significant research efforts have been made to address the performance drawbacks associated with data deduplication. Prior works [7,9–11,15,16,18–22] have focused on optimizing various techniques to mitigate these issues. A common strategy in these works

involves implementing some caching mechanisms to reduce the read I/O overhead introduced by deduplication. However, these solutions primarily target the optimization of read performance, leaving the overhead in the write I/O path largely unaddressed. Moreover, none of these works have fully explored the optimization of storage space efficiency in relation to the write amplification caused by chunk replication for fault tolerance purposes.

Another limitation in conventional scale-out deduplication approaches is their reliance on the built-in Write-Ahead Log (WAL) of distributed storage systems to ensure strict data consistency. In these approaches, all write I/Os are first buffered in a WAL before being committed to storage servers. If a write I/O operation fails, it can be retried from the WAL until an acknowledgment is received from the storage server, ensuring data reliability. However, the use of WAL schemes introduces an additional layer of write amplification, as data are written twice—once to the WAL and then to the storage servers. This further exacerbates performance degradation in deduplication-enabled storage systems, adding significant overhead to the I/O path.

In this paper, we aim to further mitigate the various overheads associated with data deduplication, addressing shortcomings observed in prior works. We propose a novel solution called Speed-Dedup which focuses on fully optimizing the deduplication process by enhancing storage space efficiency, reducing latency, and improving fault tolerance. The contributions of this work are outlined as follows:

- **Reducing the write amplification problem:** Speed-Dedup significantly reduces write amplification compared to other deduplication methods like GRATE [10] and CAO [9,16]. It efficiently manages storage by drastically lowering the total data written. For instance, with a 20% deduplication ratio, Speed-Dedup shows a notable reduction in data written to storage, highlighting its capacity for improved storage efficiency.
- **Improving data recovery times:** Speed-Dedup offers superior recovery times for failed data storage locations (OSDs). As validated in our experiments, Speed-Dedup demonstrates faster recovery with a time of 48.23 s for one failed OSD, outperforming GRATE [10] with 53.21 s and CAO [9] with 53.29 s, and this performance improvement increases as the number of OSD failures increase. This underscores Speed-Dedup's enhanced system resilience and ability to restore system health more quickly.
- **Enhancing I/O performance of deduplication systems:** Speed-Dedup improves overall I/O performance by decoupling read and write operations, allowing non-blocking reads. This contributes to reduced latency and increased throughput, optimizing system performance for both read and write tasks.
- **Modified fault tolerance and intelligent self-healing capabilities:** Speed-Dedup maintains critical fault tolerance and self-healing capabilities while optimizing system efficiency. We implement new mechanisms called the object replica check (ORC) and chunk availability check (CAC) for fault tolerance and recovery. ORC and CAC ensure high data availability, enhancing system resilience and robustness, especially in failure scenarios.

Through these innovations, our work seeks to further improve the performance and space efficiency whilst maintaining the consistency of deduplication-enabled storage systems.

2. Background and Motivation

In this section, we provide the essential background and practical challenges involved in implementing effective and high-performance deduplication storage systems. We specifically target the scale-out storage systems due to their inherent intelligent and self-healing nature suitable for cloud storage environments.

2.1. Background

The implementation of a scale-out deduplication-enabled storage system is highly dependent on the architecture of the distributed file system, particularly for ensuring consistency and failure recovery [9,10,13,16]. However, this heavy reliance can significantly impact the overall performance of the deduplication system. Figure 2 illustrates the

components and processes involved in a scale-out deduplication storage system. We begin by outlining the general structure of a conventional scale-out deduplication storage cluster.

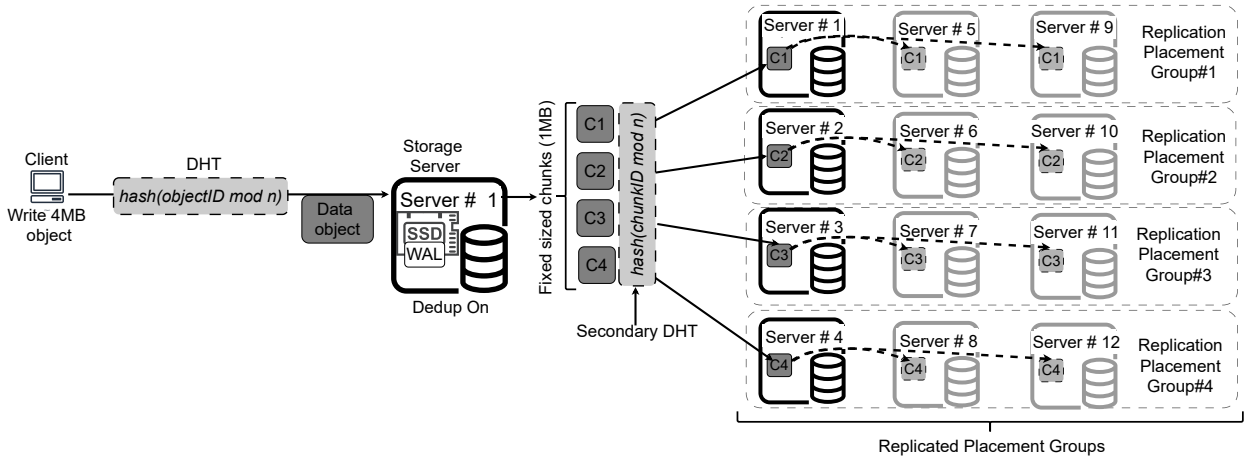


Figure 2. Conventional WAL-based data deduplication with replication in scale-out storage.

❶ **Clients:** Clients interact with the deduplication-enabled storage system through two types of operations:

- Write I/O Operation: This involves writing data objects to the scale-out storage. The write operation can either create a new data object or update an existing one. The object size corresponds to the stripe size of all incoming data. For instance, when writing a file of size 16 MB, the client splits the file into smaller objects of 4 MB sizes, which are then written to the storage system. Different cloud storage systems utilize different stripe sizes; specifically, the default stripe sizes are 4 MB for Ceph, 1 MB for Lustre, and 64 KB for Gluster. This study employs Ceph storage systems, which utilize a default object size of 4 MB.
- Read I/O Operation: This operation allows clients to read data stored in the underlying storage system using some hash based location algorithm.

The size of the read/write (R/W) data object is configurable, and for the purposes of this discussion, the object size is set to 4 MB, as shown in Figure 2.

❷ **Dynamic Hash Table (DHT):** The scale-out storage system employs dynamic hashing to calculate the storage location for read and write operations [10,11,13,17,19,23,24]. The Dynamic hashing algorithm takes two inputs: the object ID and the number of available servers (n) in the cluster. It computes the modulo operation to determine the storage location for the I/O operation [*objectID mod n*]. This hashing process occurs for both the read and write operations. The purpose of this first DHT algorithm is to enable clients to locate the storage server for writing an object.

❸ **Write-Ahead Log (WAL):** To maintain consistency in write operations, conventional storage systems adopt a Write-Ahead Log (WAL), which journals all incoming write requests before they are committed to the storage system [23,25,26]. This journaling mechanism is beneficial in cases where a write operation fails. If a failure occurs, the system can restart the I/O operation from the WAL journal, ensuring strict consistency in the storage system.

❹ **Chunking and hashing on storage servers:** Once a data object is written to its computed storage server, the deduplication process begins. Prior deduplication solutions adopt various chunking methods, either from variable or fixed-size hashing algorithms to chunk the data object [4,27,28]. In this paper, we simplify the explanation by using fixed-size chunking. The incoming data object is divided into fixed-size chunks (1 MB chunks in this example, as shown in Figure 2). The chunk size is a configurable parameter. A hash or fingerprinting process is then applied to compute the hash values of each chunk. A duplicate check is performed against the existing

deduplication metadata (Dedup MD). If a duplicate is found, the reference count in the Dedup MD is incremented. New, unique chunks are directed to a secondary DHT for final storage placement.

❺ **Secondary DHT:** For new, unique chunks (non-duplicates), a secondary dynamic hashing algorithm is applied. This secondary dynamic hashing algorithm takes as input the chunk ID and the number of available storage servers (n) and computes the modulo operation to determine the final storage location of the chunks [*chunkID mod n*]. The chunks are then redirected across the cluster network and stored on the appropriate server. The purpose of this secondary DHT algorithm in hash based deduplication storage systems [9–11,13,16] is to redirect the deduplicated chunks to their final storage locations.

❻ **Replication Placement Group (RPG):** To ensure fault tolerance and enable intelligent self-healing capabilities, the scale-out deduplication system adopts the concept of Replication Placement Groups (RPGs) [9,10,13,16,23]. The RPGs are logical groupings of object storage daemons, which manage the assignment of storage locations to specific placement groups [17,24]. Each RPG is assigned a primary storage location and, by default, two additional replication locations, as depicted in the Figure 2. When data chunks are written to their designated primary storage servers, replicas of these chunks are simultaneously written to the assigned replication locations in the RPGs. Once all replicas are successfully stored, the system sends an acknowledgment to the client, indicating the successful completion of the write operation.

2.1.1. Challenges in Conventional Deduplication Systems

As illustrated in Figures 1 and 2, the deduplication process introduces multiple intermediary steps that contribute to the degradation of the storage system's overall performance. These extra steps, while essential for identifying and eliminating duplicate data, create additional overheads to the storage system. Next, we present the several challenges presented by the implementation of the conventional deduplication systems in scale-out storage systems.

❶ **High Read/Write (R/W) latency:** During read or write operations, additional deduplication-related I/O tasks are invoked in the I/O path. Each R/W I/O operation triggers these extra processes before the data objects can be written to or read from the underlying storage cluster. This results in increased I/O latency, which can severely impact performance, particularly for I/O-sensitive applications. Moreover, as deduplication redirects the chunks to different storage locations, it breaks the locality of the data [29]. Consequently, the overall system throughput is significantly reduced.

❷ **Sub-optimal space efficiency:** While conventional deduplication systems such as [9,10,13,16,19,30] do achieve some degree of space savings, the default replication mechanism hinders optimal storage space efficiency [30,31]. The write amplification caused by chunk replication in the Replication Placement Groups (RPGs) for fault tolerance leads to inefficiencies in regard to space savings [32]. For instance, writing a 4 MB data object to the storage cluster results in storing a total of 12 MB due to the replication effect in the RPGs, assuming all chunks are new. This results in a worst-case write amplification of 3×, which is not ideal for maximizing space efficiency.

❸ **Increased Write Amplification Factor (WAF) in Write Operations:** The use of WALs can slow the performance of a system [25,26,33]. Adopting WAL journalling in conventional deduplication systems leads to double write amplification. Writes are buffered into a temporary storage device, such as an SSD-based WAL, which creates a bottleneck in the I/O path. Data are first written to the WAL journal and then subsequently flushed to process the deduplication steps. While the WAL maintains consistency and ensures failure recovery in write operations, it comes with a significant penalty in terms of overall I/O performance and increased write amplification.

❹ **Strict consistency mechanism:** Conventional deduplication systems adhere to a strict consistency model, in which all write operations are first journaled and only ac-

knowledged after all chunks are written and replicated in the RPGs. In the event of a failure, the write operation is re-initiated from the WAL checkpoints until it succeeds, after which the client receives an acknowledgment. The I/O operations also enforces some strict consistency mechanism by locking the RPGs during simultaneous access [9,16,34]. While this strict consistency model ensures reliable data integrity, it significantly impacts the performance of the storage system, leading to delays and inefficiencies.

2.1.2. Motivation

This paper's motive is to address the aforementioned limitations of conventional deduplication systems in scale-out storage environments. Existing deduplication implementations [9,10,13,16,19,30] inherit the structural inefficiencies of scale-out storage systems, leading to challenges such as increased write amplification, strict consistency overheads, and performance bottlenecks. Our work explores the implementation of new deduplication strategies designed to mitigate these challenges. Specifically, we focus on the following key areas:

- **Reducing write amplification caused by RPGs:** One of the major challenges in conventional deduplication systems is the excessive write amplification caused by chunk replication in RPGs [9,30]. To address this, we propose a solution called deduplicated primary–semi-deduplicated replica. In this approach, the full deduplication is applied only to the primary data object, while the replica object is semi-deduplicated. The proposed system writes the primary object to the storage cluster, replicates a single replica object, and then deduplicates the primary object. The deduplicated chunks are not replicated, reducing the overall write amplification. Fault tolerance is handled by an asynchronous process that manages synchronization between the deduplicated chunks and the semi-deduplicated replica, as discussed in detail in Section 3.5. For example, in a conventional deduplication system, writing a 4 MB data object could result in 12 MB being written to storage due to replication. In contrast, our proposed system reduces this to a worst-case scenario of 8 MB (4 MB for the data chunks and 4 MB for the semi-deduplicated replica).
- **Replacing strong consistency with eventual consistency:** The strong consistency model used in conventional deduplication storage systems introduces significant performance overhead, particularly in terms of latency and throughput. To improve performance, we propose replacing the strong consistency model with an eventual consistency model. This involves eliminating the use of Write-Ahead Log (WAL) journaling. Instead, we adopt a binary flag-assisted consistency mechanism to manage incoming write requests. A binary flag indicates the state of each write operation, changing from "0" to "1" upon successful write to storage. If a write fails, the flag remains with state "0". Future write requests for duplicate data can eventually correct the consistency by flipping the flag to "1", indicating a successful write. If no future duplicates are written, we adopt a periodic garbage collection thread [35] which invalidates all objects and chunks with a flag status of "0". This mechanism improves system performance by avoiding the overhead of WAL journaling. We detail this eventual consistency model in Section 3.4.1.
- **Fault tolerance retention in proposed system:** In our proposed system, we retain the fault tolerance and self-healing capabilities of conventional deduplication systems while introducing modifications meant to improve the deduplication system efficiency. Specifically, we ensure that fault tolerance is achieved between the deduplicated primary object and the semi-deduplicated replica object through new mechanisms. If a replica object becomes unreachable, a periodic Object Replica Check (ORC) thread is initiated to rehash the primary object and store it at a new location in the cluster. Similarly, if any chunk of the deduplicated primary object is unavailable, a periodic Chunk Availability Check (CAC) thread synchronizes with the semi-deduplicated replica and performs background chunking and hashing of the replica object. The background

CAC resultant chunks are compared with the chunks of the deduplicated primary object. Missing chunks are then rehashed and stored in a new location. Both the ORC and CAC threads run periodically to ensure the availability and integrity of the data. The full details of the fault tolerance retention through the ORC and ARC are discussed in Section 3.5.

By focusing on the aforementioned three areas, we are motivated to propose improvements to conventional deduplication storage systems. These enhancements are designed to boost performance without compromising the system's fault tolerance and self-healing capabilities. Next, we present the architecture of our proposed system called Speed-Dedup.

3. Proposed Speed-Dedup Architecture

3.1. Aims and Objectives of Speed-Dedup Architecture

Our proposed deduplication architecture is a novel solution that has the following aims and objectives.

- **Aim #1: Accelerating I/O acknowledgments for improved latency:** Our design aims to amortize the overall deduplication I/O latency by utilizing the semi-deduplicated replica object as the the target acknowledgment point for I/O operations. The rest of the full deduplication processes proceeds in the background for the primary object. In this way, latency is significantly improved. This is achieved by introducing the deduplicated primary–semi-deduplicated replica approach explained in Section 3.2.1.
- **Aim #2: Optimization of deduplication storage efficiency:** Our system aims to reduce the overall storage footprint of data objects by (i) minimizing RPG object replication, (ii) applying full deduplication exclusively to the primary object, (iii) disabling deduplicated chunks of the primary object from replication, and (iv) disabling replication on the semi-deduplicated replica object. This approach further optimizes the storage efficiency of the deduplication system. Fault tolerance is then managed by synchronizing between the deduplicated primary chunks and the semi-deduplicated replica object, as described in Section 3.2.2.
- **Aim #3: Incremental improvement in recovery time with increased storage failures:** To support the modifications in the deduplication system, we aim to implement specialized fault tolerance and data recovery mechanisms that are tightly integrated into our proposed framework. We introduce two new mechanisms, Object Replica Check (ORC) and Chunk Availability Check (CAC), detailed in Section 3.5, that handle fault tolerance and data recovery processes. These techniques are designed to incrementally improve data recovery times as storage node failures increase, offering enhanced performance compared to other state-of-the-art solutions.

Speed-Dedup introduces a unique approach that implements deduplication using a deduplicated primary and a semi-deduplicated replica object, distinguishing it from prior solutions that rely on fully deduplicating primary objects before invoking the replicas of the deduplicated object chunks. This novel deduplication design not only boosts performance but also substantially reduces write amplification, marking a significant advancement in deduplication technology. Additionally, we present new fault tolerance and data recovery mechanisms that speed up fault detection and recovery time, further enhancing system resilience and efficiency. Next, we discuss the details of our proposed solution.

3.2. System Architecture

The design of our proposed Speed-Dedup system addresses the limitations of conventional deduplication approaches in scale-out storage environments. To achieve the desired goals of improved performance and fault tolerance, we modify the conventional deduplication architecture to create a more efficient and resilient system. Below, we outline the key components of this design. Figure 3 presents the architectural diagram of the proposed Speed-Dedup system.

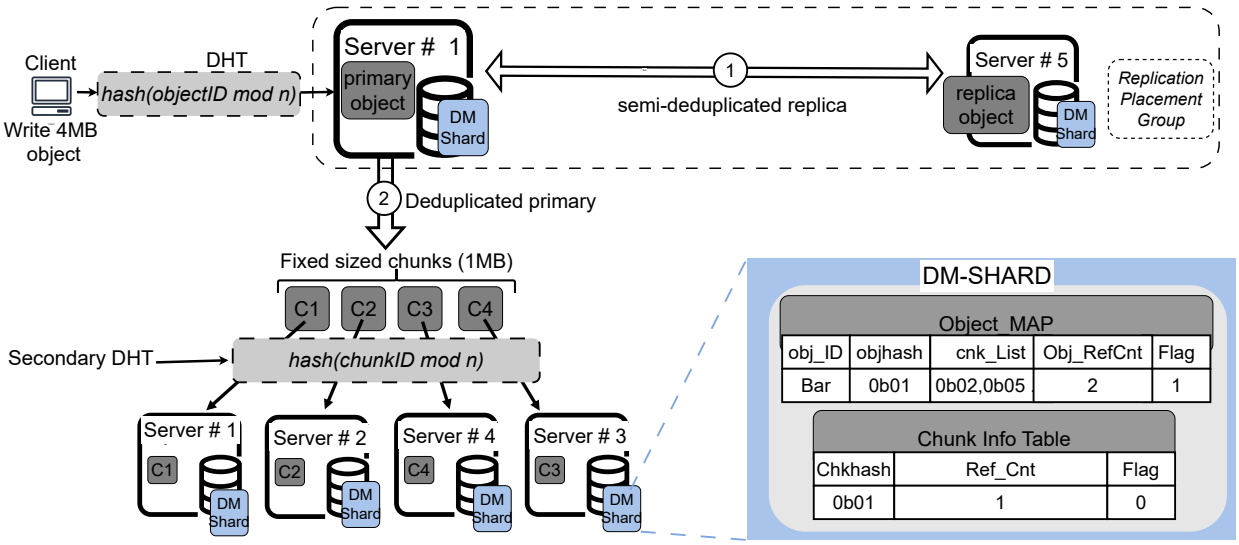


Figure 3. Proposed Speed-Dedup design.

3.2.1. Deduplicated Primary–Semi-Deduplicated Replica Model

The central feature of our design is the deduplicated primary–semi-deduplicated replica model. This model streamlines the deduplication process while maintaining data reliability and fault tolerance. The process is broken down into two main parts:

❶ **Single semi-deduplicated replica of primary object across RPGs:** For fault tolerance, we only replicate the primary object with a replication factor of two, meaning that each data object has a primary copy and a replica. Each RPG consists of a primary storage location and a designated replica location. Once the primary object is written to its assigned primary location, a replica is automatically generated and stored at the secondary location. In the case of duplicate replica object, the duplicate is not replicated, but rather object reference count is incremented in the Object MAP of the DM-Shard, enforcing the concept of semi-deduplicated replica. The semi-deduplication is in the sense that duplicate objects are referenced to already stored replica objects but do not undergo the full process of deduplication (chunking, hashing, and chunk redirection for storage).

❷ **Full deduplication of only the primary object:** In this model, only the primary data object undergoes the deduplication process. This includes chunking the object into smaller data chunks, hashing each chunk, and performing duplicate checks. To manage deduplication metadata, we implement a distributed partitioning method called Deduplication Metadata Sharding (DM-Shard), as inspired by the work in [10]. In the DM-Shards, metadata for both duplicate chunks and new, unique chunks are stored and updated. The new chunks are then passed through a secondary dynamic hashing algorithm to determine their final storage locations. Unlike conventional systems, our design does not replicate the deduplicated chunks of the primary object. While this approach reduces write amplification, it introduces the potential of lack of fault tolerance. To address this, we implement additional fault tolerance and self-healing mechanisms, which are discussed in detail in Section 3.5.

❸ **Structure and processes of the DM-Shard:** The DM-Shard is a distributed metadata management strategy that partitions the deduplication metadata across multiple servers to optimize scalability and performance in a scale-out storage system. The design leverages a hash-based data placement to automate the process of distributing metadata across different storage nodes. Below, we outline the key processes and structure of the DM-Shard.

- **Distributed metadata partitioning:** When a client writes an object to storage, a Dynamic Hash Table (DHT) algorithm is employed to determine the specific server responsible for handling the write request. Once the data reach the assigned storage server, they undergoes two primary operations: (i) Replication within the Replication Placement Group (RPG) for fault tolerance (semi-deduplicated replica); (ii) Deduplication of primary objects to identify and store only unique data chunks.
- **Object mapping table (Object MAP):** During the deduplication process, the metadata of the incoming objects is stored in the Object Mapping Table (Object MAP), which is part of the DM-Shard. The Object MAP records all the essential information required to reconstruct the deduplicated object, including the following: Object ID (obj_ID)—A unique identifier for the object; Object Hash Value (objhash)—Hash value for the entire object; Chunk List (cnk_List)—A list of the deduplicated object's chunk hashes; Object reference count (obj_RefCont), which stores the reference counts of the semi-deduplicated replicas; and a Binary Flag, which indicates whether the object, along with all its chunks, is successfully written to storage. It also reflects the status of the replica. The flag is initially set to "0" and is flipped to "1" once both the deduplicated primary object and its semi-deduplicated replica are successfully written.
- **Chunk Information Table (CIT):** For the deduplicated primary object, each chunk undergoes a secondary DHT algorithm to calculate its final placement location in the storage system. Once the chunk reaches its designated storage location, its metadata are recorded in the Chunk Information Table (CIT) within the DM-Shard. The CIT contains critical chunk-level information, including chunk IDs, chunk hash values, and a consistent flag. Upon successfully writing the chunks to their storage locations, the flag in the CIT is flipped from "0" to "1".

Figure 3, bottom right, illustrates the structure of the DM-Shard, showing how both object-level and chunk-level metadata are managed and distributed across the system. This distributed approach ensures scalability and efficient handling of deduplication metadata in scale-out storage systems.

3.2.2. Modified Replication Placement Group

In our proposed Speed-Dedup system, the Replication Placement Group mechanism is specifically applied between primary objects and their replica objects. This targeted approach optimizes the replication process by focusing solely on these two object types, which reduces overall system overhead and write amplification while still maintaining necessary fault tolerance and data reliability.

❶ **Replication process:** When a client writes a primary object to the storage system, a replication thread is invoked immediately after the primary object is written to its designated primary storage location. This thread replicates the object to a secondary storage location, known as the replica location, which is dynamically determined by the storage system's RPG mechanism [17,24]. This replica location can be any storage node assigned to a primary node by the RPG computation algorithm, such as the CRUSH algorithm. In our design, we enforce a replication factor of two, meaning each data object has exactly one primary copy and one replica. This simplifies the replication process compared to the conventional deduplication systems with higher replication factors, reducing both storage requirements and the potential for excessive write amplification.

❷ **Dynamic RPG calculation and self-healing:** Rather than introducing new algorithms for RPG computation [36,37], Speed-Dedup leverages the built-in dynamic capabilities of the distributed file system's RPG computation algorithm, such as Ceph's CRUSH algorithm [17]. This algorithm automatically calculates and assigns RPGs based on the available storage locations in the cluster, determining the optimal placement of primary and replica objects. If a storage location becomes unavailable due to failure

or other issues, the system automatically recalculates and adjusts the RPGs. This allows for the seamless reallocation of the replica to a new, healthy storage node, ensuring the continued availability and redundancy of data. For example, if a storage node hosting a replica object fails, the system dynamically reassesses the cluster's available storage nodes and identifies new placement locations. This allows for an efficient self-healing process without the need for manual intervention, preserving data integrity and minimizing downtime.

❸ **Optimization of performance and fault tolerance with the Modified RPGs:** By limiting replication to just one primary and one replica, Speed-Dedup achieves a balance between performance and fault tolerance. The replication factor of two ensures that data remain redundant and protected against failures while minimizing the storage overhead and I/O workload associated with higher replication factors. This controlled replication approach, combined with the system's ability to dynamically reallocate RPGs in the event of failures, ensures the fault tolerance features of conventional scale-out storage systems whilst also significantly reducing the write amplification and overhead typically associated with traditional deduplication processes. It achieves these benefits by limiting replication and allowing the system to dynamically adapt to changes in cluster availability, ensuring high performance and data reliability.

3.3. Decoupling of Read and Write (R/W) Operations

As illustrated in Figure 3, our proposed system stores two types of objects for each data write: (i) A semi-deduplicated replica object and (ii) the data chunks of the deduplicated primary object. This deduplicated primary–semi-deduplicated replica model allows us to decouple the read operations from write updates after the initial object write, significantly improving performance.

- **Read operation:** In our design, all read operations are served via the semi-deduplicated replica object, ensuring non-blocking access to data even during write updates. In traditional deduplication systems, read operations issued during an object write update are often blocked. This happens because the system locks the RPG (Replication Placement Group) to prevent simultaneous read and write I/O operations on the same object, maintaining strong consistency but at the cost of I/O performance degradation. To overcome this limitation, Speed-Dedup implements a non-blocking read I/O thread. This thread is invoked through the semi-deduplicated replica object, allowing read operations to proceed independently from ongoing write updates to the primary object. By separating the replica from the deduplication process, we ensure faster and more efficient read operations, improving the overall responsiveness of the system.
- **Write update operation:** The write update operation is handled through the deduplicated primary object. When a write update is triggered, the system retrieves the relevant data chunks using metadata stored in the MD-Shards, such as the object ID (obj_ID), object hash (objhash), and the chunk list (chkList). If the object requiring the update exists, the updated object undergoes the following processes. Chunking: The updated object is divided into chunks. Hashing: Each chunk is hashed via the hashing algorithm. Duplicate checks: The system verifies whether the updated chunks already exist in storage. For any new chunks, the system hashes them to their final storage locations, updating the metadata in the MD-Shards accordingly. The old metadata records of the object are then invalidated to reflect the update.
- **Addressing inconsistencies between primary and replica objects:** A potential challenge of this decoupled R/W model is the risk of inconsistencies between the deduplicated primary object and the semi-deduplicated replica object. Since the write update is applied only to the primary object, the replica may not immediately reflect the changes made to the primary object. To resolve this, we implement a periodic replication thread that runs in the background. This thread periodically reconciles the primary object with its replica by replicating the updated data from the deduplicated primary object to the semi-deduplicated replica and invalidating the old replica copy

to ensure that the replica is always up to date with the primary object. This periodic synchronization ensures that the system maintains consistency between the primary and replica objects without sacrificing the performance benefits of decoupled read and write operations.

By decoupling the read and write processes in this manner, our system significantly improves I/O performance and scalability while still preserving data consistency.

3.4. Write and Read I/O Acknowledgement

In our proposed Speed-Dedup system, when data objects are written to the storage system, the data are first replicated to a semi-deduplicated replica. Upon successfully writing this replica to storage, the client receives an immediate I/O acknowledgment. The deduplication process for the primary object continues in the background. This approach ensures improved I/O latency and overall performance, as the client does not have to wait for the entire deduplication process to complete before receiving confirmation of the write operation. In contrast, conventional deduplication systems like in [9,10] fully deduplicate incoming data objects before issuing an acknowledgment to the client. These systems wait for the object chunks to be redirected to their final storage locations, resulting in a longer delay before the client receives an acknowledgment for the write operation.

For read operations, Speed-Dedup serves all reads from the semi-deduplicated replica object, which operates independently of the write updates that are served by the deduplicated primary object. This decoupling ensures faster read performance, as the client reads directly from the replica without needing to wait for the deduplicated data to be reconstructed. In contrast, conventional deduplication systems serve read operations via the same deduplication I/O path, requiring the system to first reconstruct the deduplicated object before responding to the client's read request. This adds additional latency to read operations. By utilizing this approach, Speed-Dedup achieves enhanced deduplication performance while reducing latency for both write and read I/O operations.

3.4.1. Eventual Consistency Model Implementation

Another objective of our proposed Speed-Dedup system is to enhance performance by replacing the strong consistency model traditionally employed in conventional deduplication storage systems with an eventual consistency model. The strong consistency model enforces blocking I/O operations, and also the client receives an acknowledgment only after the entire data object is successfully written and replicated across the storage system. This approach, while ensuring reliability, results in high latency and reduced throughput, making it suboptimal for large-scale or I/O-sensitive environments. In our proposed system, we implement an eventual consistency model via a flag-ssisted consistency mechanism embedded in the DM-Shard.

❶ **Flag-sssisted consistency mechanism:** To achieve eventual consistency, we adapt a flag-assisted consistency model in [10]. The consistency model relies on a binary consistency flag to monitor the status of an object's commit to storage. Each object and its deduplicated chunks are associated with metadata, including a consistency flag that reflects the success or failure of storage commit operations. When a client write data objects to the proposed deduplication enabled storage, the following metadata management and object storage process for eventual consistency are invoked:

- The object is first hashed to identify its initial storage location;
- In this initial location, the object undergoes replication across its RPG, ensuring redundancy, and the primary object is also chunked and hashed as part of the deduplication process;
- The system then updates the Object MAP table with relevant metadata for the object, including the object ID (obj_ID), object hash (objhash), chunk list (chkList), and a consistency flag. This flag is initially set to "0" and is changed to "1" only after a successful commit of both the replica object and all deduplicated chunks;

- The deduplicated chunks are then processed through a secondary hashing algorithm, which calculates their final storage locations;
- Upon arrival at the final storage location, the Chunk Information Table (CIT) of the DM-Shard is updated with chunk metadata, including chunk hashes, reference counts, and consistency flags. Similar to the Object MAP table, the consistency flag in the CIT is initialized to "0" and flips to "1" upon a successful commit to storage of the data chunk.

❷ **Eventual consistency flag commit:** In an eventual consistency model, even if a chunk fails to write successfully to storage, the consistency flag remains at status "0", and the reference count of the chunk is not incremented, creating a temporary inconsistency in the system.

- **Handling inconsistencies with duplicate data:** Our system resolves inconsistencies by leveraging future data writes. If a duplicate data chunk (matching an earlier failed chunk) is written to the system at any point, the consistency flag in the CIT is updated from "0" to "1", and the reference count for the chunk is incremented. This mechanism allows the system to self-heal by correcting inconsistencies over time as duplicate data naturally enters the system.
- **Garbage collection for inconsistent chunks:** In the worst-case scenario where no future duplicate chunk is written to the system, a periodic garbage collection thread scans through the storage system. It invalidates any objects or chunks with a consistency flag still set to "0", thereby ensuring that inconsistent or incomplete data are removed. This thread prevents long-term accumulation of inconsistent data and maintains the integrity of the storage system.

Adopting this eventual consistency model with flag-assisted consistency mechanism ensures that data reliability is eventually achieved without imposing the high performance costs associated with strict consistency models.

3.5. Fault Tolerance Handling in the Proposed System

To address the fault tolerance changes made to the conventional deduplication system, our Speed-Dedup design includes new fault tolerance mechanisms aimed at preserving the fault tolerance and self-healing capabilities of scale-out storage systems. As described previously, our system adopts a deduplicated primary–semi-deduplicated replica model, where only the replica object is replicated and deduplicated chunks of the primary object are not replicated. This approach reduces the overall write amplification caused by replication in traditional deduplication systems. Despite reducing the replication overhead, maintaining one copy of the deduplicated data chunk and one copy of the replica object introduces challenges for fault tolerance. To mitigate these challenges, we introduce two periodic background threads that handle fault detection and recovery: (i) the Object Replication Check (ORC) thread and (ii) the Chunk Availability Check (CAC) thread.

The fault tolerance mechanisms in Speed-Dedup are designed to operate periodically in the background of each storage server, ensuring minimal disruption to ongoing I/O operations. The ORC and CAC threads run in parallel, which is crucial for reducing interference with foreground I/O processes. This parallel execution allows the system to maintain high performance while continuously monitoring the availability and integrity of both the replica object and primary object chunks. Specifically, the ORC thread is responsible for verifying the availability of replica objects, while the CAC thread monitors the consistency of the deduplicated chunks of the primary object. By implementing these background threads, Speed-Dedup effectively enhances its fault tolerance capabilities without compromising the responsiveness of the system to client requests. Next, we explain how these two threads ensure fault tolerance and data integrity through continuous monitoring and corrective actions.

❶ **Object Replication Check (ORC) thread:** The ORC thread is designed to verify the availability of replica objects and runs periodically in the background. The primary

purpose of the ORC thread is to ensure that the replica object is still accessible and valid. Figure 4 illustrates the fault tolerance mechanism implemented by the ORC thread. The ORC thread periodically pings the replica object location. If the replica location is reachable, it issues an *objhash.getattr()* query to check whether the actual replica object is physically stored at that location. If the query returns a success message, the replica object is confirmed to be intact. However, if the location is unreachable or the *objhash.getattr()* query returns an error, the ORC thread initiates failure recovery operations to reconstruct and restore the replica object. Next, we explain the two failure scenarios for the ORC thread.

Case #1: Replica object location not reachable.

- In this scenario, the distributed storage system automatically recalculates the new Replica Placement Groups (RPGs) (Step 1 in Figure 4).
- Next, the ORC thread retrieves metadata about the failed replica from the Object MAP of the primary object, locates the object chunks using the secondary DHT algorithm (Step 2), and reconstructs the primary object based on these data (Step 3).
- The reconstructed primary object is then replicated to a new replica location in the updated RPG (Step 4).

Case #2: Replica location is reachable, but *objhash.getattr()* returns error.

- This means that the replica object may be corrupted.
- In this case, the ORC thread reconstructs the primary object (Step 2—Figure 4) and replicates the reconstructed primary object onto the current RPG location, that is, Server #5 in Step 4b in Figure 4.

By employing this approach, the ORC thread maintains fault tolerance through continuous synchronization between the primary object chunks and the semi-deduplicated replica.

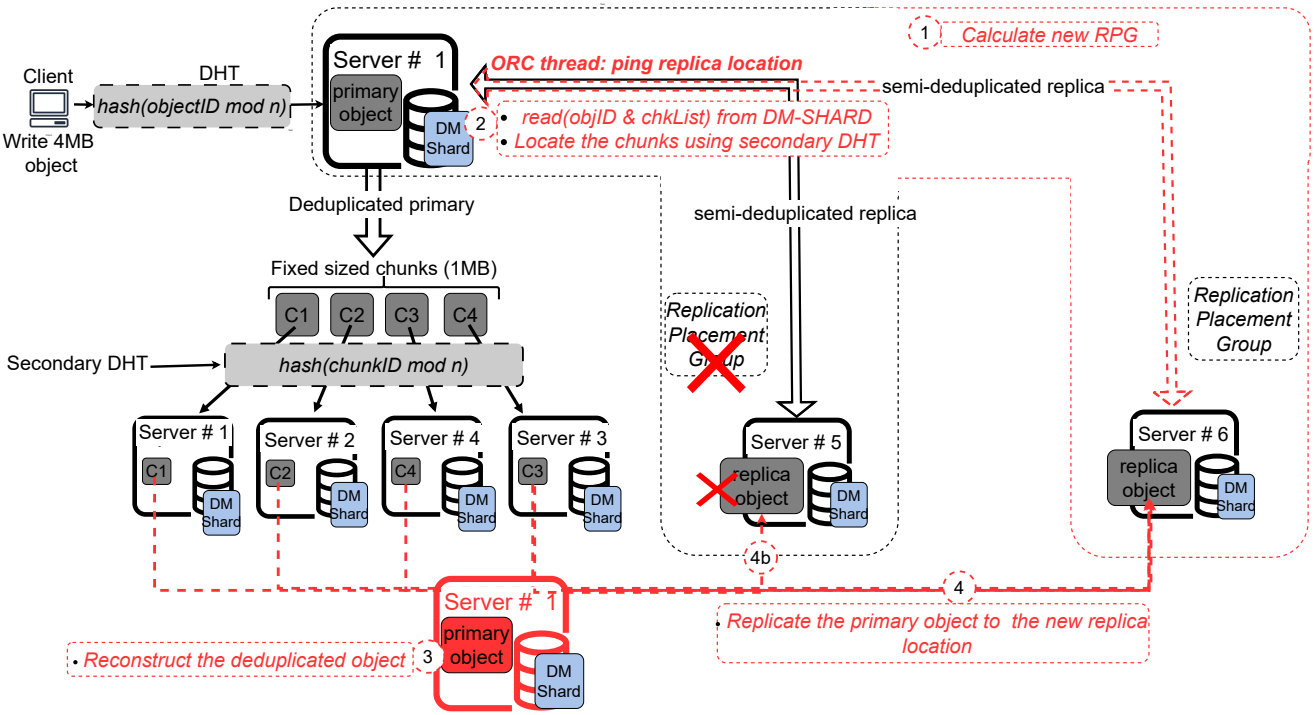


Figure 4. Fault tolerance using Object Replica Check (ORC) thread. Top red X means a faulty replication node and bottom means faulty replica object.

❷ **Chunk Availability Check (CAC) thread:** Figure 5 illustrates the fault tolerance mechanism implemented by the CAC thread. The CAC thread is responsible for monitoring the availability and consistency of the deduplicated chunks of the primary object. Like the ORC thread, the CAC thread runs periodically and ensures the integrity of chunks stored in the system. The CAC thread monitors the fault tolerant levels using two hiarachical operations: (i) Pinging the storage location of the chunk and checking whether it is reachable, and (ii) if the location is reachable, verification of the physical presence of the chunks using the *cnkhash.getattr()* query. If the query returns success, the chunk is confirmed to be available. The CAC thread also manages two failure scenarios: (i) when the chunk location is not reachable and (ii) when the chunk location is reachable but the *cnkhash.getattr()* query returns an error. We proceed to discuss the two failure scenarios for the CAC thread.

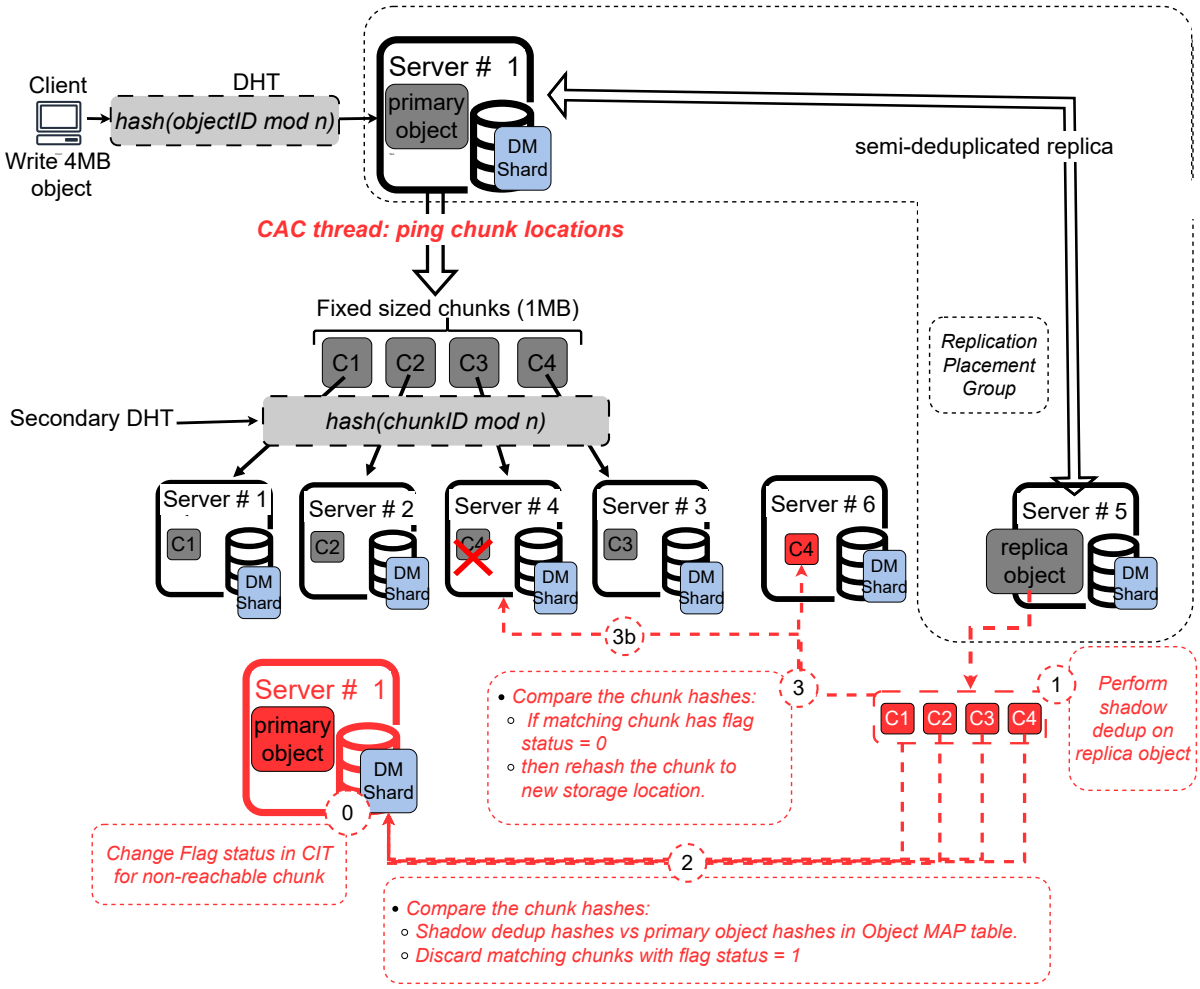


Figure 5. Fault tolerance using Chunk Availability Check (CAC) thread. Red X means an unreachable chunk.

Case #1: Chunk location not reachable.

- The CAC thread first performs a shadow deduplication process on the replica object, Step 1 in Figure 5;
- It then compares the chunk hashes obtained from the shadow deduplication process of the replica with those in the Object MAP of the primary object (Step 2). If any chunk hashes match those stored in reachable locations, the matching chunks are discarded;
- For matching chunks that are not reachable, the CAC thread rehashes the new chunk location and copies the unreachable chunk from the shadow deduplication to the newly identified storage location (Step 3).

Case #2: Chunk location reachable, but *cnkhash.getattr()* returns error.

- This scenario indicates that the chunk may be corrupted. In this case, the CAC thread updates the Chunk Information Table (CIT) by changing the flag for the corrupted chunk to "0" and decrements the reference count, Step 0 in Figure 5;
- The CAC thread then proceeds with shadow deduplication on the replica object (Step 1);
- Next, CAC compares the chunk hashes from the shadow deduplication with the corrupted chunk hash in the CIT table. If a match is found, the shadow deduplicated chunk is copied into the original storage location, the reference count is incremented, and the flag status in the CIT is reset to "1" (Step 3b).

By implementing these two background threads, ORC and CAC, our proposed Speed-Dedup ensures automatic fault tolerance and intelligent self-healing. The periodic checks performed by these threads allow the system to detect and recover from both replica and chunk failures without requiring manual intervention. This design maintains the resilience of the proposed deduplication system while reducing the overhead typically associated with redundant replication in conventional models.

4. Evaluation

We implemented our proposed Speed-Dedup solution on Ceph version 15.2.17, a widely used distributed storage platform. To evaluate the performance of our system, we conducted a comparative analysis against two state-of-the-art deduplication solutions: GRATE [10] and CAO [9]. Both GRATE and CAO were integrated into the same Ceph version by copying their binaries to ensure a consistent evaluation environment. To standardize the comparison across all systems, we adopted a fixed chunking approach for all three solutions: our proposed Speed-Dedup, GRATE, and CAO. For efficient data distribution and rehashing, we adopted the CRUSH algorithm [17] (Ceph's native data placement algorithm) as both the primary DHT algorithm and the secondary DHT algorithm. This approach optimizes the intelligent rehashing of Replica Placement Groups (RPGs) within the storage cluster.

As previously discussed in Section 3.5, our system maintains fault tolerance differently compared to other conventional deduplication systems. Speed-Dedup only stores one copy of each data chunk (from the deduplicated primary object) and one copy of a semi-deduplicated replica object. In this scenario, if either the deduplicated primary object or its semi-deduplicated replica fails, the built-in Ceph recovery mechanisms cannot recover the data. Therefore, fault tolerance in our system is achieved through the Object Replication Check (ORC) and Chunk Availability Check (CAC) mechanisms, which were outlined in Section 3.5. In contrast, for both GRATE and CAO deduplication systems, fault tolerance is managed via the built-in Ceph recovery mechanisms. These mechanisms rely on replicating multiple copies of data across the cluster, ensuring data redundancy and automatic recovery in the event of a failure.

4.1. Testbed and Experimental Setup

For our evaluation, we deployed a Ceph storage cluster using AWS EC2 instances [38] consisting of eight storage servers. Each server is equipped with 2 $\times$ 512 GB SSDs, configured as object storage daemons (OSDs), providing a unified storage capacity of 8 TB across the cluster. Additionally, we deployed a single node that serves a dual role as both the Ceph cluster management node and the client node. Table 1 presents the specifications of each server in the Ceph cluster.

Table 1. Testbed specifications for the deployed Ceph cluster. Storage nodes (left) and cluster management also used as client node (right).

Storage Nodes		**Management/Client Node**	
Atrribute	**Description**	**Atrribute**	**Description**
EC2 VM Type	c5.2xlarge: 8 vCPUs; ×86_64 arch; 16 GB RAM	EC2 VM Type	c5.2xlarge: 8 vCPUs; ×86_64 arch; 16 GB RAM
VM Image	Ubuntu Server 22.04	VM Image	Ubuntu Server 22.04
Storage Type	SSDs = 2 × 512 GB per server	Storage Type	Used for hosting OS and Ceph manager services only
Number of storage servers	Total 8 servers in the storage cluster	Object Size	4 MB

To simulate real-world workloads and generate I/O requests (reads and writes), we used the FIO benchmark tool [39] at the Ceph client node. This tool allows us to test the storage system by generating synthetic deduplication workloads under various I/O parameters and conditions. All incoming write I/O requests were segmented into 4 MB object sizes at the Ceph client. We created and mounted a rados block device (RBD) at the Ceph client node, and the client directs all its I/Os via this RBD. An example of the FIO command that we executed in our experiment is as follows:

```
fio --name speed-dedup.fio --dedupe --dedupe_percentage=20
--filename=/dev/rbd0 --ioengine=rbd --rw=write --size=6291456MB
--iodepth=128 --bs=4096 --numjobs=1
```

where

- **–name speed-dedup.fio:** Sets the name of the job. This name helps to identify the results in the output logs.
- **–dedupe:** Enables deduplication, which allows fio to use deduplication patterns in the data it writes. This is useful for simulating workloads that include deduplicated data.
- **–dedupe_percentage=20:** Sets the percentage of data that are deduplicated. Here, 20 means 20% data is deduplicated (20% duplicate content).
- **–filename=/dev/rbd0:** Specifies the target file or device for the workload. In this case, it is the rbd (RADOS block device) device located at /dev/rbd0.
- **–ioengine=rbd:** Chooses the I/O engine. Here, rbd is specified, meaning it uses the RADOS Block Device (often associated with Ceph) as the I/O engine.
- **–rw=write:** Sets the I/O pattern. Here, write means it performs sequential write operations.
- **–size=6,291,456 MB:** Defines the total data size for the job. Here, it writes a total of 6 TB.
- **–iodepth=128:** Sets the I/O depth, which is the number of I/O requests that can be in flight at the same time. Higher values can increase concurrency, making it useful for testing storage performance under load.
- **–bs=4096:** Sets the block size (in this case, 4 MB) for each I/O operation.
- **–numjobs=1:** Specifies the number of jobs to run in parallel. Here, it is set to one, so only a single job is executed.

The FIO benchmark tool enables us to test the storage performance under various configurations. For example, we can vary the dedupe_percentage, I/O pattern (sequential, reads, sequential writes, random reads, random writes), size of data to be generated and written to storage, etc. We performed a comparative analysis of the following three deduplication approaches using this testbed setup:

❶ **Speed-Dedup:** Our proposed deduplication solution, which deduplicates only the primary object and stores a semi-deduplicated replica for fault tolerance.

❷ **GRATE [10]:** This approach also uses a distributed metadata sharding technique, similar to our proposed solution. However, GRATE deduplicates and replicates all new chunks across the storage cluster, unlike our system which stores a semi-deduplicated replica object and only one copy of each chunk.

❸ **CAO [9]:** In the CAO approach, deduplication is implemented through content-addressable objects. Related deduplication metadata are stored within the extended attributes of each deduplicated object, meaning that all deduplication metadata updates are embedded directly within the object's extended attributes [9,16].

4.2. Performance Evaluation

In this Section, we benchmark the performance of Speed-Dedup with various varying performance configurations.

❶ **Performance with varying chunk sizes:** Figure 6a shows the performance of Speed-Dedup, GRATE, and CAO when the chunk sizes of objects written to storage are varied. In this test, the deduplication ratio was set to 20%. We observed that across all systems, as chunk size increases, performance improves due to the reduced overhead of managing larger chunks and less I/Os. However, Speed-Dedup consistently demonstrates superior performance compared to GRATE and CAO, especially with larger chunk sizes.

The main reason for this performance boost is that Speed-Dedup minimizes write amplification, which means fewer I/O operations are required to write the same amount of data. This increases throughput, as more data can be written in less time. Additionally, Speed-Dedup decouples the read and write I/O processes. This enforces non-blocking read operations, further increasing the system performance. In contrast, GRATE and CAO implements blocking W/R I/O operations, which introduces more overhead in handling reads and writes simultaneously.

❷ **Performance with varying deduplication ratio:** We further evaluated the performance of the proposed Speed-Dedup approach by benchmarking it against varying deduplication ratios of written data to the storage cluster. In this experiment, the chunk size was fixed at 256 KB, while the deduplication ratio was varied between 80% and 20%. The results of this experiment are illustrated in Figure 6b.

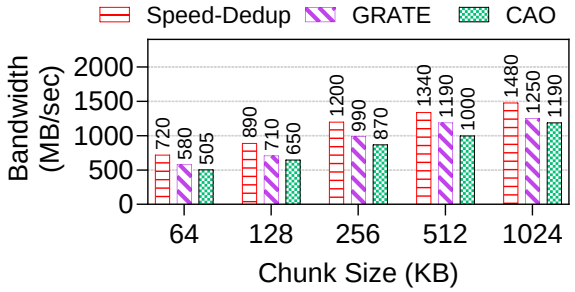
Theoretically, performance is expected to increase as the deduplication ratio increases, since a lower deduplication ratio indicates fewer duplicate data in the incoming workload. Conversely, a higher deduplication ratio implies more duplicates, which reduces the amount of data commits to storage. In the FIO benchmark tool, the deduplication ratio is expressed as a percentage, where a higher percentage reflects a lower deduplication ratio (e.g., 80% deduplication ratio corresponds to 80% unique data and 20% duplicates), while a lower percentage reflects a higher deduplication ratio (e.g., 20% deduplication ratio corresponds to 20% unique data and 80% duplicates).

From the results presented in Figure 6b, we observed that performance improves across all approaches as the deduplication ratio increases. Notably, Speed-Dedup demonstrates the highest performance gains, particularly when the deduplication ratio is towards the lower bound (80% unique data). At this low deduplication ratio, Speed-Dedup outperforms both GRATE and CAO by at least 21%. This performance advantage is due to Speed-Dedup's efficient handling of large volumes of new data, as it writes significantly less data to storage. Speed-Dedup stores only one replica object with fewer deduplicated chunks, whereas GRATE and CAO amplify write operations by writing each chunk of new data to storage and replicating it across multiple nodes within a RPGs for fault tolerance.

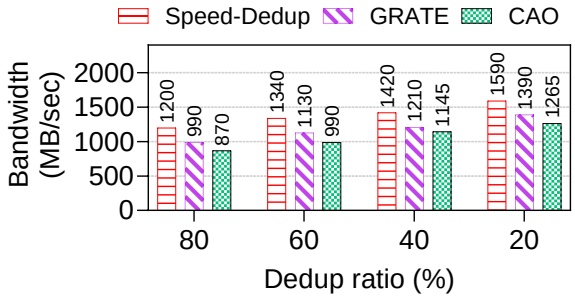
❸ **Latency with varying chunk sizes:** We conducted an evaluation of the latency performance of the three approaches, Speed-Dedup, GRATE, and CAO by fixing the deduplication ratio at 20% and varying the chunk sizes. The results of this experiment are illustrated in Figure 6c. We observed a similar performance trend for all three

approaches, with latency improving as chunk sizes increase. However, Speed-Dedup consistently outperforms both GRATE and CAO in terms of latency.

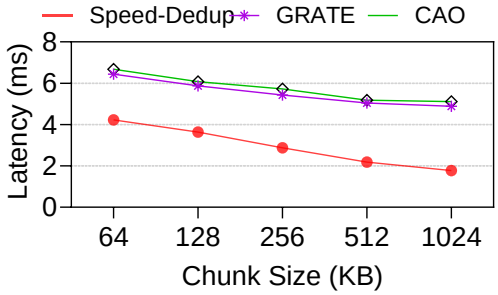
The key advantage of Speed-Dedup lies in its ability to acknowledge the client as soon as all the chunks of the primary object are written to storage, allowing the deduplication of object chunks to continue in the background. This delayed deduplication process ensures eventual consistency while reducing the time it takes for the client to receive an acknowledgment. As a result, Speed-Dedup demonstrates a reduction in latency of more than 100% compared to GRATE and CAO.



(a) Performance with varying chunk size (dedup ratio = 20%)



(b) Performance with varying dedup ratio (chunk size = 256KB)



(c) Latency with varying chunk size (dedup ratio = 20%)

Figure 6. Performance analysis of proposed model.

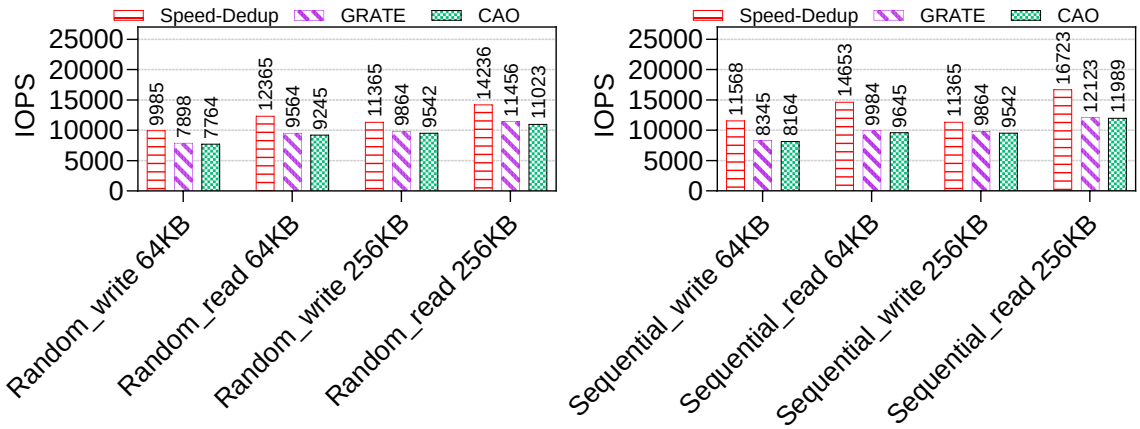
In contrast, CAO waits until the entire deduplication process is completed before sending an acknowledgment to the client, leading to higher latency. However, in GRATE, the client receives an acknowledgement as soon as the chunks are written to the CIT table, meaning GRATE also uses an eventual consistency model. Nevertheless, the object has to undergo the deduplication processes of chunking hashing and I/O redirection to final storage location first before client acknowledgement. This results in a higher number of I/O operations before the client receives an acknowledgment in GRATE and CAO compared to Speed-Dedup, which minimizes these operations and therefore exhibits superior latency performance.

❹ **Evaluating IOPS with Random and Sequential I/Os:** To evaluate the resilience of the proposed Speed-Dedup approach, we analyzed the I/O operations per second (IOPS)

and compared its performance with the GRATE and CAO deduplication methods. The results of this evaluation are presented in Figure 7.

Random I/O Performance: We first examined the IOPS by issuing random I/O requests from the Ceph client while varying the chunk size between 64 KB and 256 KB. The results are shown in Figure 7a. For both random reads and random writes, we observed a general increase in performance across all approaches as the chunk size increases. However, Speed-Dedup demonstrates superior performance in both random write and read operations compared to GRATE and CAO. The key reason for this improvement is that, during write operations, Speed-Dedup sends a write acknowledgment to the client as soon as the semi-deduplicated replica is committed to storage. In contrast, with read operations, Speed-Dedup decouples the read I/O from the write operations, resulting in enhanced overall performance.

Sequential I/O Performance: For sequential I/O operations, as depicted in Figure 7b, Speed-Dedup exhibits a significantly higher performance compared to GRATE and CAO. This improvement is attributed to the enhanced locality characteristics of Speed-Dedup during write operations. In Speed-Dedup, the IOPS for sequential writes are measured based on the commit to storage of the semi-deduplicated replica object. Since this replica object is not deduplicated, the locality of reference is maintained, allowing sequential I/O requests to retain their ordered nature. On the other hand, with GRATE and CAO, IOPS are measured only after the chunks are fully deduplicated and redirected to their respective storage locations. This redirection changes the sequential write and read operations into random I/O patterns, disrupting the locality of reference and leading to reduced performance. As a result, Figure 7b highlights the substantial performance gap in favor of Speed-Dedup for both sequential reads and sequential writes compared to GRATE and CAO, underscoring Speed-Dedup's ability to maintain higher IOPS in sequential I/O scenarios.



(a) Random I/O perfomance varying chunk size (dedup ratio = 20%)

(b) Sequential I/O perfomance varying chunk size (dedup ratio = 20%)

Figure 7. Evaluation of random and sequential client I/O requests with varying chunk sizes.

4.3. Write Amplification Evaluation

In this experiment, we evaluated the I/O write amplification of the three different deduplication methods: the proposed Speed-Dedup, GRATE, and CAO deduplication techniques. The chunk size is fixed at 256 KB throughout the experiment. Table 2 presents the results of this comparative analysis. To benchmark the write amplification, a total of 6 TB of data is written to the storage system by the client, and performance is analyzed under two conditions:

- Lower-bound deduplication ratio (20%): 80% of the data is duplicate;
- Upper-bound deduplication ratio (80%): 20% of the data is duplicate.

Table 2. Comparison of I/O write amplification of the proposed approach vs GRATE and CAO dedup methods. Green means optimal values and red means non-optimal values.

	Total Data Written from Client	Total Duplicate Data	Total Objects Written to Storage	Total Chunks Written to Storage	Total Replica Objects Written to Storage	Final Data Saved in Storage
Speed-Dedup 20% Dedup Ratio	6 TB	1.2 TB	314,573	5,033,168	314,573	2.4 TB
GRATE 20% dedup ratio	6 TB	1.2 TB	314,573	15,099,504	0	3.6 TB
CAO 20% dedup ratio	6 TB	1.2 TB	314,573	15,099,504	0	3.6 TB
Speed-Dedup 80% dedup ratio	6 TB	4.8 TB	1,258,292	20,132,672	1,258,292	9.6 TB
GRATE 80% dedup ratio	6 TB	4.8 TB	1,258,292	60,398,016	0	14.4 TB
CAO 80% dedup ratio	6 TB	4.8 TB	1,258,292	60,398,016	0	14.4 TB

When 6 TB of data was written with the lower deduplication ratio (20%), all approaches wrote 1.2 TB of duplicate objects to storage, equivalent to 314,573 data objects. With the upper bound deduplication ratio (80%), 4.8 TB of duplicate data was written by all approaches, totaling 1,258,298 data objects. However, the total number of chunks written to storage differed significantly between the approaches.

At the lower bound deduplication ratio (20%), Speed-Dedup wrote 5,033,168 chunks to storage, while both GRATE and CAO wrote 15,099,504 chunks. At the upper bound deduplication ratio (80%), Speed-Dedup wrote 20,132,672 chunks compared to 60,398,016 chunks for GRATE and CAO. These results indicate that the proposed Speed-Dedup approach has a lower write amplification. The key difference is that Speed-Dedup writes only one replica object for each primary object; all deduplicated primary object chunks are not replicated. However, GRATE and CAO fully deduplicate the incoming objects and replicate the deduplicated chunks by a factor of three, which amplifies the writes to storage.

The results in Table 2 reveal the following:

- At the lower bound (20% deduplication ratio), Speed-Dedup wrote 2.4 TB of data to storage (314,573 replica objects [4 MB each] and 5,033,168 deduplicated chunks [256 KB each]) compared to 3.6 TB for GRATE and CAO (5,033,168 chunks replicated by a factor of 3 = 15,099,504 chunks of 256 KB each).
- At the upper bound (80% deduplication ratio), Speed-Dedup wrote 9.6 TB (1,258,292 replica objects [4 MB each] and 20,132,672 deduplicated chunks [256 KB each]) to storage, while GRATE and CAO wrote 14.4 TB (20,132,672 replicated by a factor of 3 = 60,398,016 replicated chunks of size 256 KB each).

This write amplification experiment points to the following important insights:

❶ **Storage efficiency:** Speed-Dedup demonstrates significantly better storage efficiency compared to GRATE and CAO. At both low and high deduplication ratios, it writes less data to storage, thereby utilizing storage space more effectively.

❷ **Write amplification:** Speed-Dedup exhibits lower write amplification, as it writes fewer chunks and replica objects. This reduction in write amplification is essential for optimizing I/O operations and improving performance.

❸ **Scalability:** The scalability of Speed-Dedup with growing sizes of data sets is exhibited by its ability to maintain lower storage requirements as the deduplication ratio increases as indicated in Table 2. These results show a much lower write amplification compared to the other state-of-the-art and the consistency of Speed-Dedup across varying deduplication ratios suggests that it scales better with larger datasets. As deduplication ratios increase, it consistently maintains lower storage requirements, making it advantageous for environments with rapidly growing data.

4.4. Fault Tolerance and Failure Recovery

In Ceph, the storage location for data is at the level of the Object Storage Device (OSD). To evaluate recovery performance, we measured the time required to recover all data objects after varying the number of OSD failures. The recovery time was measured from the moment an OSD failed to the point when the system's health status returned to "OK". The results of this experiment are shown in Table 3.

Table 3. Recovery time of failed data storage locations (OSDs) in the cluster. Green means optimal values and red means non-optimal values.

	Recovery Time (s) 1 Failed OSD	**Recovery Time (s) 2 Failed OSDs**	**Recovery Time (s) 3 Failed OSDs**	**Recovery Time (s) 4 Failed OSDs**
Speed-Dedup	48.23	49.56	51.34	54.35
GRATE	53.21	56.23	59.74	62.13
CAO	53.29	56.28	60.02	62.13

The analysis of Table 3, which presents the recovery times for failed OSDs within the storage cluster, reveals several key insights:

❶ **Speed-Dedup's consistent superiority:** Speed-Dedup consistently demonstrates faster recovery times compared to both GRATE and CAO across all test cases. For instance, with a single OSD failure, Speed-Dedup completes recovery in 48.23 s, while GRATE and CAO take 53.21 s and 53.29 s, respectively. This shows that Speed-Dedup has an advantage in recovering from OSD failures.

❷ **Incremental Increase in Recovery Times:** As the number of failed OSDs increases, recovery times increase for all approaches. However, Speed-Dedup consistently outperforms GRATE and CAO, maintaining shorter recovery durations even as the scale of failures grows. This trend underscores the superior efficiency of Speed-Dedup in recovery scenarios.

❸ **Efficiency in Handling Multiple Failures:** The results suggest that Speed-Dedup not only recovers more quickly but also exhibits a smaller incremental increase in recovery times as the number of OSD failures increases. This suggests that Speed-Dedup is more effective at managing multiple OSD failures compared to GRATE and CAO which show a larger increase in recovery time as failures scale up.

Overall, these trends highlight the effectiveness of the ORC and CAC fault tolerance and recovery mechanisms in Speed-Dedup, which contribute to its faster recovery times and superior scalability in handling multiple OSD failures.

4.5. Analysis of Interference During Data Recovery

To further analyze the impact of recovery time during the data recovery process, we traced the interference on foreground I/O operations (IOPS) and the duration of data recovery during OSD failure events. For this evaluation, we set the periodic invocation time for the ORC and CAC threads to 200 s. To simulate OSD failures, we ran a script that deactivates one OSD from three selected storage servers out of the eight servers in our configured cluster every 150 s. The trace was conducted over a total period of 700 s. The results of this experiment are shown in Figure 8. The results indicate that whenever data recovery threads are initiated, foreground I/O operations are temporarily suspended. This period of data recovery represents the time during which the storage cluster redistributes data objects and chunks to new storage locations. During this data migration process, foreground I/Os cannot be served due to interference from the data migration. In Figure 8, we observe that our proposed Speed-Dedup approach demonstrates low interference time, along with superior IOPS performance, compared to the GRATE and CAO methods. This result further demonstrates Speed-Dedup effectiveness in achieving high-performance fault tolerance and data recovery efficiency.

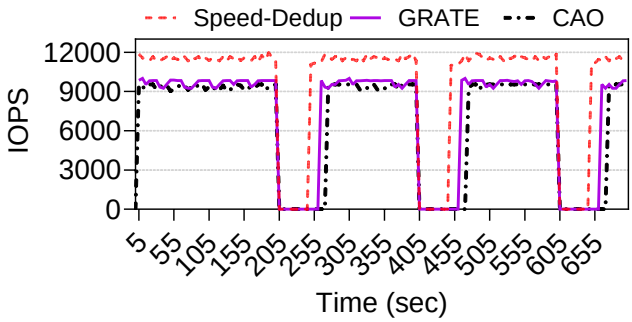


Figure 8. Interference analysis during data recovery.

5. Conclusions

This paper proposes Speed-Dedup, a high-performance deduplication framework aimed at overcoming the inefficiencies of traditional deduplication systems in large-scale storage environments. By introducing a novel approach that maintains a deduplicated primary object alongside a semi-deduplicated replica, Speed-Dedup significantly reduces write amplification, thereby minimizing the overall data written to storage. Unlike conventional systems that fully deduplicate all replicated data, Speed-Dedup optimizes storage space efficiency by only deduplicating the primary object. This strategy not only enhances storage utilization but also mitigates the performance penalties associated with traditional replication methods. Furthermore, Speed-Dedup maintains the critical fault tolerance and self-healing features of traditional systems through mechanisms like ORC and CAC, ensuring reliable data availability and fast recovery without compromising performance. Supported by extensive experimental results, Speed-Dedup demonstrates significant improvements in both write I/O performance and space efficiency when compared to conventional approaches. This makes Speed-Dedup a more efficient and scalable deduplication solution for modern, scale-out storage infrastructures.

Potential future applications for Speed-Dedup could include enhanced cloud storage solutions, where its ability to significantly reduce write amplification and improve I/O performance can be leveraged to optimize storage costs and efficiency in large-scale out-storage environments. Additionally, the framework's fault tolerance mechanisms, such as Object Replica Check (ORC) and Chunk Availability Check (CAC), could be applied in disaster recovery scenarios, ensuring high data availability and faster recovery times during system failures. Also, Speed-Dedup's capability to decouple read and write operations allows for non-blocking reads, which could be particularly beneficial in real-time data processing applications where low latency is critical. This feature could enhance performance in various sectors, including finance, healthcare, and online services, where rapid data access is essential.

6. Future Work

In this study, we implemented our solution using physical disks as object storage daemons (OSDs) to evaluate the effectiveness of the proposed recovery mechanisms. These mechanisms were designed and thoroughly analyzed, with a focus on addressing failures at the level of physical disk locations. Our future work will aim to broaden the scope of Speed-Dedup by incorporating Virtual Machines and the utilizing of snapshots for data storage and recovery. Snapshots are expected to significantly improve recovery times due to their ability to capture the system state more efficiently, reducing the time required to restore data after a failure [40].

Author Contributions: Conceptualization, P.H., D.-J.C. and T.-S.C.; methodology, P.H.; software, P.H.; validation, P.H., D.-J.C. and T.-S.C.; formal analysis, P.H., D.-J.C., and T.-S.C.; investigation, P.H. and D.-J.C.; resources, T.-S.C.; data curation, P.H.; writing—original draft preparation, P.H.; writing—review and editing, P.H., D.-J.C. and T.-S.C.; visualization, P.H. and D.-J.C.; supervision, T.-S.C.; project administration, T.-S.C.; funding acquisition, T.-S.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Institute of Information and Communications Technology Planning and Evaluation (IITP) under the Artificial Intelligence Convergence Innovation Human Resources Development (IITP-2024-RS-2023-00255968) grant and the ITRC (Information Technology Research Center) support program (IITP-2021-0-02051) funded by the Korea government (MSIT). Additionally, this work was supported by the BK21 FOUR program of the National Research Foundation of Korea funded by the Ministry of Education (NRF5199991014091).

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created; we used a synthetic workload generator called FIO that synthetically generates the data. The FIO benchmarking tool is accessible at https://github.com/axboe/fio (accessed on 25 October 2024.). Data sharing is not applicable to this article

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Reinsel, D.; Gantz, J.; Rydning, J. Data Age 2025: The Digitization of the World From Edge to Core—Seagate IDC. Available online: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf (accessed on 25 October 2024).
2. Cisco. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021. Available online: https://virtualization.network/Resources/Whitepapers/0b75cf2e-0c53-4891-918e-b542a5d364c5_white-paper-c11-738085.pdf (accessed on 25 October 2024).
3. Wallis, D.B.M.; Stroman, A. Use Deduplication, Data Compression, and Data Compaction to Increase Storage Efficiency Overview. NetApp ONTAP 9. 2024. Available online: https://docs.netapp.com/us-en/ontap/pdfs/sidebar/Use_deduplication__data_compression__and_data_compaction_to_increase_storage_efficiency.pdf, (accessed on 25 October 2024).
4. Ahmed, S.T.; George, L.E. Lightweight hash-based de-duplication system using the self detection of most repeated patterns as chunks divisors. *J. King Saud Univ.—Comput. Inf. Sci.* **2022**, *34*, 4669–4678. [CrossRef]
5. Fekry, A. Big Data Gets Bigger: What about Data Cleaning Analytics as a Storage Service? In Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 17), Santa Clara, CA, USA, 10–11 July 2017.
6. Kaur, R.; Chana, I.; Bhattacharya, J. Data deduplication techniques for efficient cloud storage management: A systematic review. *J. Supercomput.* **2018**, *74*, 2035–2085. [CrossRef]
7. Zhang, D.; Le, J.; Mu, N.; Wu, J.; Liao, X. Secure and Efficient Data Deduplication in JointCloud Storage. *IEEE Trans. Cloud Comput.* **2023**, *11*, 156–167. [CrossRef]
8. Yu, X.; Bai, H.; Yan, Z.; Zhang, R. VeriDedup: A Verifiable Cloud Data Deduplication Scheme With Integrity and Duplication Proof. *IEEE Trans. Dependable Secur. Comput.* **2023**, *20*, 680–694. [CrossRef]
9. Oh, M.; Park, S.; Yoon, J.; Kim, S.; Lee, K.; Weil, S.; Yeom, H.Y.; Jung, M. Design of Global Data Deduplication for a Scale-Out Distributed Storage System. In Proceedings of the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, 2–6 July 2018; pp. 1063–1073. [CrossRef]
10. Khan, A.; Hamandawana, P.; Kim, Y. A Content Fingerprint-based Cluster-wide Inline Deduplication for Shared-Nothing Storage Systems. *IEEE Access* **2020**, *8*, 209163–209180. [CrossRef]
11. Hamandawana, P.; Khan, A.; Kim, J.; Chung, T.S. Accelerating ML/DL Applications With Hierarchical Caching on Deduplication Storage Clusters. *IEEE Trans. Big Data* **2022**, *8*, 1622–1636. [CrossRef]
12. Zhang, T.; Chen, R.; Li, Z.; Gao, C.; Wang, C.; Shu, J. Design and Implementation of Deduplication on F2FS. *ACM Trans. Storage* **2024**, *20*, 21. [CrossRef]
13. Wang, J.; Wang, Y.; Wang, H.; Ye, K.; Xu, C.; He, S.; Zeng, L. Towards Cluster-wide Deduplication Based on Ceph. In Proceedings of the 2019 IEEE International Conference on Networking, Architecture and Storage (NAS), Enshi, China, 15–17 August 2019; pp. 1–8. [CrossRef]
14. Zhao, N.; Lin, M.; Albahar, H.; Paul, A.K.; Huan, Z.; Abraham, S.; Chen, K.; Tarasov, V.; Skourtis, D.; Anwar, A.; et al. An End-to-end High-performance Deduplication Scheme for Docker Registries and Docker Container Storage Systems. *ACM Trans. Storage* **2024**, *20*, 18. [CrossRef]
15. Goel, A.; Prabha, C. A Detailed Review of Data Deduplication Approaches in the Cloud and Key Challenges. In Proceedings of the 2023 4th International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 20–22 September 2023; pp. 1771–1779. [CrossRef]

16. Oh, M.; Lee, S.; Just, S.; Yu, Y.J.; Bae, D.H.; Weil, S.; Cho, S.; Yeom, H.Y. TiDedup: A New Distributed Deduplication Architecture for Ceph. In Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 23), Boston, MA, USA, 10–12 July 2023; pp. 117–131.
17. Weil, S.A.; Brandt, S.A.; Miller, E.L.; Maltzahn, C. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06, Tampa, FL, USA, 11–17 November 2006; ACM: New York, NY, USA, 2006. [CrossRef]
18. Kwon, H.; Cho, Y.; Khan, A.; Park, Y.; Kim, Y. DENOVA: Deduplication Extended NOVA File System. In Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17–21 May 2022; pp. 1360–1371. [CrossRef]
19. Jackowski, A.; Ślusarczyk, Ł.; Lichota, K.; Wełnicki, M.; Wijata, R.; Kielar, M.; Kopeć, T.; Dubnicki, C.; Iwanicki, K. ObjDedup: High-Throughput Object Storage Layer for Backup Systems With Block-Level Deduplication. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 2180–2197. [CrossRef]
20. Cao, Z.; Wen, H.; Wu, F.; Du, D.H. ALACC: Accelerating Restore Performance of Data Deduplication Systems Using Adaptive Look-Ahead Window Assisted Chunk Caching. In Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST 18), Oakland, CA, USA, 12–15 February 2018; pp. 309–324.
21. Park, D.; Fan, Z.; Nam, Y.J.; Du, D.H. A Lookahead Read Cache: Improving Read Performance for Deduplication Backup Storage. *Comput. Sci. Technol.* **2017**, *32*, 26–40. [CrossRef]
22. Du, C.; Lin, Z.; Wu, S.; Chen, Y.; Wu, J.; Wang, S.; Wang, W.; Wu, Q.; Mao, B. FSDedup: Feature-Aware and Selective Deduplication for Improving Performance of Encrypted Non-Volatile Main Memory. *ACM Trans. Storage* **2024**, *20*, 22. [CrossRef]
23. Weil, S.A.; Brandt, S.A.; Miller, E.L.; Long, D.D.E.; Maltzahn, C. Ceph: A Scalable, High-performance Distributed File System. In Proceedings of the Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06, Seattle, WA, USA, 6–8 November 2006.
24. Li, Z.; Wang, Y. An adaptive read/write optimized algorithm for Ceph heterogeneous systems via performance prediction and multi-attribute decision making. *Clust. Comput.* **2023**, *26*, 1573–7543. [CrossRef]
25. Zhu, L.; Hu, Y.; Wang, C. Asynchronous and Adaptive Checkpoint for WAL-based Data Storage Systems. In Proceedings of the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), Ocean Flower Island, China, 17–21 December 2023; pp. 1238–1245. [CrossRef]
26. Jin, S.; Yan, Q.; Zhang, Y.; Yang, J. A High-Performance Distributed File System for Mass Data. In Proceedings of the 2020 IEEE 6th International Conference on Computer and Communications (ICCC), Chengdu, China, 11–14 December 2020; pp. 1804–1809. [CrossRef]
27. Feng, D. Chunking Algorithms. In *Data Deduplication for High Performance Storage System*; Springer Nature: Singapore, 2022; pp. 25–51. [CrossRef]
28. Xia, W.; Feng, D.; Jiang, H.; Zhang, Y.; Chang, V.; Zou, X. Accelerating content-defined-chunking based data deduplication by exploiting parallelism. *Future Gener. Comput. Syst.* **2019**, *98*, 406–418. [CrossRef]
29. Zou, X.; Yuan, J.; Shilane, P.; Xia, W.; Zhang, H.; Wang, X. The Dilemma between Deduplication and Locality: Can Both be Achieved? In Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21), Virtual, 23–25 February 2021; USENIX Association: Berkeley, CA, USA, 2021; pp. 171–185.
30. Cheng, G.; Luo, L.; Xia, J.; Guo, D.; Sun, Y. When Deduplication Meets Migration: An Efficient and Adaptive Strategy in Distributed Storage Systems. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 2749–2766. [CrossRef]
31. CRUSH Maps Available online: https://docs.ceph.com/en/reef/rados/operations/crush-map/ (accessed on 25 October 2024).
32. Ahmed, S.; Nahiduzzaman, M.; Islam, T.; Bappy, F.H.; Zaman, T.S.; Hasan, R. FASTEN: Towards a FAult-tolerant and STorage EfficieNt Cloud: Balancing Between Replication and Deduplication. *arXiv* **2023**, arXiv:2312.08309.
33. Hwang, J.Y.; Cho, S. Lee,D-Y.; Jeong, K.; Han, S-H.; Kim, J-S.; Hwang, J-Y Understanding write behaviors of storage backends in ceph object store. In Proceedings of the 2017 IEEE International Conference on Massive Storage Systems and Technology, Santa Clara, CA, USA, 12–15 May 2021; 2017; Volume 10. Available online: https://msstconference.org/MSST-history/2017/Papers/CephObjectStore.pdf (accessed on 25 October 2024).
34. Jeong, B.; Khan, A.; Park, S. Async-LCAM: A lock contention aware messenger for Ceph distributed storage system. *Clust. Comput.* **2019**, *22*, 373–384. [CrossRef]
35. Ceph. Ceph Scrubbing. Available online: https://docs.ceph.com/en/latest/rados/configuration/osd-config-ref/#scrubbing (accessed on 25 October 2024).
36. Kisous, R.; Kolikant, A.; Duggal, A.; Sheinvald, S.; Yadgar, G. The what, The from, and The to: The Migration Games in Deduplicated Systems. *ACM Trans. Storage* **2022**, *18*, 31. [CrossRef]
37. Nachman, A.; Sheinvald, S.; Kolikant, A.; Yadgar, G. GoSeed: Optimal Seeding Plan for Deduplicated Storage. *ACM Trans. Storage* **2021**, *17*, 24. [CrossRef]
38. AWS. Amazon EC2. 2024. Available online: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (accessed on 25 October 2024).

39. Axboe, J. Flexible I/O Tester. Available online: https://github.com/axboe/fio (accessed on 25 October 2024).
40. Li, Z.; Yu, X.; Yu, L.; Guo, S.; Chang, V. Energy-efficient and quality-aware VM consolidation method. *Future Gener. Comput. Syst.* **2020**, *102*, 789–809. [CrossRef]