Received 8 November 2024, accepted 2 December 2024, date of publication 19 December 2024, date of current version 31 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3520184

RESEARCH ARTICLE

Efficient Data Placement in Deduplication Enabled ZenFS via CRC-Based Prediction

SAFDAR JAMIL[1], JOSEPH RO[1], JOO-YOUNG HWANG[2], AND YOUNGJAE KIM[1], (Member, IEEE)
[1]Department of Computer Science and Engineering, Sogang University, Seoul 04107, South Korea
[2]Samsung Electronics Company, Suwon-si 16707, South Korea

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

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (Ministry of Science and ICT) under Grant NRF-2021R1A2C2014386, Grant RS-2024-00453929, and Grant RS-2024-00416666; and in part by Samsung Electronics Co., Ltd., under Grant IO221014-02908-01.

ABSTRACT The Zoned Namespace (ZNS) interface shifts data management responsibility to upper-level applications, requiring them to reclaim space by issuing the zone-reset command to ZNS SSD devices, a process known as garbage collection (GC). Application-level GC can lead to performance degradation due to the high valid data copy overhead, which is further exacerbated by the larger GC units in ZNS SSDs. However, the impact of larger GC units can be mitigated if GC operations are made interruptible, allowing I/O requests to be served during zone resets or block reclamation. Moreover, the adoption of offline data deduplication as a storage optimization technique in ZNS-based file systems like ZenFS presents additional challenges. Offline deduplication must consider lifetime-based file allocation to avoid deduplicating hot data, and placing unique and duplicate data blocks together can further increase valid data copy overhead during GC. To address these issues, we propose *DeZNS*, an innovative data placement strategy for deduplication-enabled ZenFS. *DeZNS* tackles the increased valid data copy overhead during GC in offline deduplication by employing a lightweight CRC32 checksum-based method to predict potential duplicates with minimal performance impact, segregating unique and duplicate data blocks. This segregation reduces valid data migration overhead during GC, while the interruptible GC mechanism ensures that ongoing I/O requests are not delayed during zone resets, maintaining ZenFS performance. Additionally, *DeZNS* integrates an offline deduplication module that operates on segregated zones. Our extensive evaluation shows that *DeZNS* reduces valid data migration by 28% compared to baseline ZenFS and by up to 2× compared to naive offline deduplication in micro-benchmarks.

INDEX TERMS Zoned namespace SSD, data deduplication, file system, data placement, CRC32-checksum.

I. INTRODUCTION

Key-value stores are popular storage engines known for their simplicity and scalability. For example, RocksDB [1], based on a Log-Structured Merge (LSM) tree, is Facebook's default storage engine and is optimized for write I/Os. There are two main LSM-tree designs: (i) traditional, where keys and values are stored together, and (ii) key-value separation, as proposed by WiscKey [2]. Various studies have optimized the traditional LSM-tree design [3], [4]. While standard RocksDB uses the traditional design, its variant, BlobDB [5], employs the key-value separation design. BlobDB reduces RocksDB's write amplification and write stalls by decoupling values.

RocksDB and BlobDB are optimized for block-based NVMe SSD, however, there are attempts to extend these key-value stores for Zone Namespace SSDs (ZNS SSD). ZNS SSD divides the logical address space into fixed-size zones, allowing only sequential writes, which optimizes NAND flash memory utilization and enhances I/O performance by eliminating Garbage Collection in the Flash Translation Layer (FTL) [6], [7], [8]. These features make ZNS SSDs ideal for BlobDB, which uses a key-value separation design with sequential, append-only write patterns. However, BlobDB requires middleware for effective data management

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

on ZNS SSDs. ZenFS [9], a user-level Log-structured File System optimized for RocksDB and BlobDB, employs an LSM-tree-aware zone allocation strategy to organize files with similar lifetimes into the same zone, reducing garbage collection-related data migration.

Although BlobDB reduces the write amplification, it still has the problem of unnecessary write amplification in workloads with high duplication. To mitigate this, storage optimization technique like deduplication can be adopted. Data deduplication is a storage optimization technique that retains only unique data while eliminating duplicates. Numerous studies have identified high levels of data duplication in user workloads [10], [11], and various research efforts have explored implementing deduplication in block-based storage systems. These investigations have focused on integrating deduplication at the file system level [10], [11], [12], [13] and the block device drive level [14]. Most of the deduplication efforts have targeted conventional SSD/HDD-based systems, however, there has been no exploration of deduplication support within ZNS file system, such as ZenFS.

ZenFS is a user-level file system based on Log-structured File System. Although deduplication has been thoroughly studied for LFS, works like SmartDedup [13] and F2DFS [15]. These works solely focus on optimizing performance for deduplication systems on conventional NVMe SSD and do not consider the impact of deduplication on garbage collection process. This is because in conventional NVMe SSDs the garbage collection process is offloaded to Flash Translation Layer and the erase unit is a few megabytes. Therefore, the garbage collection process does not interfere with the file system operations. However, in ZNS SSDs-based file systems, the garbage collection process is the responsibility of the file system, making it a critical factor in performance and write amplification in ZenFS. Additionally, one of the fundamental differences between NVMe SSD and ZNS SSD is the erase unit which defines the performance of garbage collection process, due to valid data migration overhead. Therefore, naively adopting existing deduplication solutions in ZenFS would lead to following shortcomings.

- *Amplified garbage collection overhead:* Offline deduplication increases garbage collection calls and the overhead of copying valid data. It marks valid duplicate data as invalid, triggering garbage collection when thresholds are crossed, leading to more data copying. To reduce this overhead, unique and duplicate data should be segregated by zone with minimal impact on the application.
- *Agnostic of file organization:* ZenFS organizes files by lifetime. Deduplicating short-lived files yields minimal benefits while wasting resources. Therefore, focusing deduplication on long-lived files is crucial for optimizing resource usage and maintaining system efficiency.

In this work, we propose *DeZNS*, a novel data placement approach for deduplication-enabled ZNS file systems. *DeZNS* uses a CRC32 checksum-based method to segregate unique and duplicate data, predicting potential duplicates with minimal performance overhead. Additionally, *DeZNS* employs an offline deduplication module to handle deduplication in unique and duplicate zones, marking duplicates as invalid. This approach reduces valid data migration overhead during the garbage collection process while maintaining ZenFS performance. The key contributions are as follows:

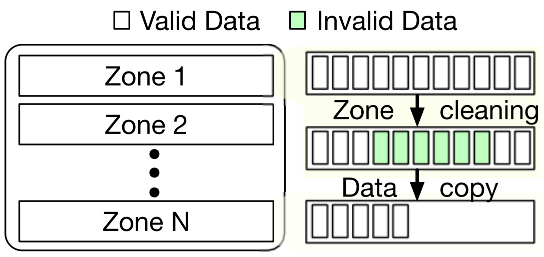


FIGURE 1. Overview of ZNS device and zone reclamation process.

- A systematic investigation of the adoption of deduplication in ZNS file system and identification of potential bottlenecks.
- A CRC32 checksum-based data placement with minimal performance overhead to identify potential duplicates. This approach minimizes valid data migration overhead caused by offline deduplication through effective segregation of unique and duplicate data.
- Implementation of *DeZNS* atop ZenFS and extensive evaluation using macro- and micro-benchmarks. Our evaluation reveals that, *DeZNS* reduces the valid data copy overhead on average by 28% and up-to 2× in comparison to baseline ZenFS and offline dedup-enabled ZenFS, with increasing deduplication ratio using micro-benchmark. *DeZNS* is able to maintain the performance in both macro- and micro-benchmarks.

II. BACKGROUND AND MOTIVATION

A. ZONED NAMESPACE SSD (ZNS SSD)

The NVMe Zoned Namespace (ZNS) [7], [16] utilizes a zone interface designed for flash-based SSDs [6], [17], [18], [19]. Each SSD zone comprises multiple NAND erase blocks, spanning 16 to 128 flash chips, resulting in a few gigabytes of writable capacity per zone. These blocks are directly accessible to the host machine through the zone interface. ZNS ensures only sequential writes to each zone with reset commands, eliminating the necessity for garbage collection in the Flash Translation Layer of the SSD. However, adopting ZNS necessitates software modifications on the host side due to hardware changes in the SSD [6], [17], [19], [20]. For instance, applications utilizing ZNS SSDs must manage data placement by selecting zones during data writes. Furthermore, applications are responsible for free-space reclamation [6], [17], [19], explicitly erasing zones instead of relying on the SSD's FTL. The free-space reclamation process involves executing zone-reset commands, which erase blocks and decrease the Program/Erase (P/E) cycle [21] of cells in the

NAND flash memory. During zone-reset, valid data in the zone to be reset must be copied into a free zone before the reset, leading to increased I/O blocking time. Figure 1 shows the overview of ZNS device and its zone reclamation process.

B. ZONE MANAGEMENT WITH ZenFS

ZenFS [7] is a user-level Log-structured File System tightly integrated with BlobDB to manage persistent components on ZNS devices. ZenFS is responsible for managing zones for Write-Ahead Logs (WAL), SSTables, and blob files created by BlobDB, and reclaiming zones to secure free space. The files are organized in file extents (data blocks) of fixed sizes, 128 KB by default. ZenFS implements a lifetime hint-based algorithm for file allocation, organizing zones to group files with similar lifetimes. The lifetime hint values in ZenFS are *short*, *medium*, *long*, and *extreme*. We categorize *short* and *medium* as hot data, while *long* and *extreme* as cold. This policy minimizes the number of valid files in a single zone, enhancing the efficiency of the garbage collection process. ZenFS allocates a dedicated asynchronous thread to handle the garbage collection process. This thread periodically checks the garbage collection trigger threshold and, once reached, selects a victim zone and proactively performs garbage collection. The garbage collection process is designed to preemptively secure empty zones by choosing a victim zone based on the highest amount of invalid data, thereby minimizing valid data migration overhead.

Moreover, the garbage collection process in ZenFS is categorized based on two cases (i) reclaiming a zone having only invalid files, and (ii) reclaiming a zone having both valid and invalid files. For the case (i), only a zone reset command is issued which resets the write pointer of the zone. Meanwhile, for the case (ii), valid data copy is conducted which results in higher write amplification as in conventional SSDs. ZenFS explicitly calls the case (ii) as 'garbage collection' and a dedicated asynchronous thread is responsible to perform this garbage collection operation. ZenFS wakes up the garbage collection thread periodically to perform the zone reclamation of a victim zone if a predefined condition is satisfied, to proactively secure empty zones.

C. DATA DEDUPLICATION

Deduplication is a technique used to optimize storage utilization. A typical deduplication process involves: *(Step 1)* segmenting data into fixed or variable-sized chunks, *(Step 2)* computing cryptographic fingerprints for each chunk, *(Step 3)* performing duplicate fingerprint lookups, and *(Step 4)* updating deduplication metadata and storing only unique chunks. Deduplication can be classified as inline or offline, based on when the deduplication operation is performed [11]. Inline deduplication occurs during I/O operations, providing immediate storage space savings but potentially degrading I/O throughput. Offline dedup, performed on already stored data, does not impact I/O throughput as it occurs outside the critical I/O path [10], [11], [12].

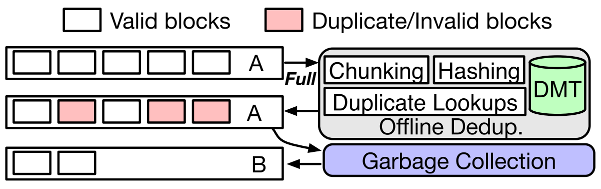


FIGURE 2. An illustration of offline deduplication in ZenFS. The offline deduplication invalidating the duplicate but valid blocks resulting in additional garbage collection overhead.

D. MOTIVATION

The fundamental difference between conventional SSDs and ZNS SSDs lies in their erase unit. This discrepancy raises intriguing questions for deduplication strategies: *How can we adapt existing techniques to handle these larger erase units effectively? And how do we optimize storage requirements while maintaining efficient internal operations?*

To address above-mentioned questions, we conducted a thorough investigation into integrating deduplication into the ZNS-based file system, ZenFS. Although, data deduplication is widely studied for file systems [10], [11], [12] and object storage systems [22], [23], [24], [25], but there is no study that has explored the integration of deduplication in ZNS-based file systems. As explained in Section II-C, data deduplication can be adopted inline and offline manner. We first discuss the opportunities to adopt deduplication in inline manner then provide details of offline deduplication in the rest of this section.

1) INLINE DEDUPLICATION IN ZNS FILE SYSTEM

The inline deduplication is known for providing immediate storage savings. However, the inline deduplication carries out all the deduplication steps within the critical I/O path of write operation. When a data block is written, it goes through the chunking, cryptographic hashing, duplicate detection, and if found unique then writing to the storage device otherwise the data chunk is discarded. The inline deduplication has been extensively exploited in different file systems [10], [26], [27].

Since ZenFS is tightly coupled with BlobDB, our previous work, DenKV [28] explores the integration of inline deduplication system in BlobDB. DenKV performs deduplication steps during the FLUSH operation when Immutable MemTables are written to SSTables and Blob files. During the FLUSH operation, each value goes through the deduplication steps, chunking, cryptographic fingerprinting, duplicate detection, and if found unique then written to the blob file otherwise discarded. Meanwhile the keys and the list of value pointers are written to SSTables. DenKV selectively performs deduplications on values of key-value pairs to achieve better space savings and further reduce the write amplification. Although, DenKV achieves the goal of write and space amplification reduction but it increases the frequency of write stall events due to reduced service rate of the FLUSH operation.

Additionally, ZenFS incorporates various design choices that need to be taken into account when considering the adoption of inline deduplication. For instance, whether to design a system oblivious of the lifetime hinting based data placement and perform deduplication on all the files coming from application? If this design is to be adopted then performing deduplication on short-lived data do not provide any benefit instead exhaust the system resource. Additionally, ZenFS is a user-level file system tightly coupled with write optimized BlobDB, and incorporation of inline deduplication comes with its own negative performance overhead from cryptographic hashing and deduplication metadata management [12], [29].

Adopting deduplication without carefully considering the design choices of ZenFS could directly interfere with BlobDB's performance. For instance, implementing inline deduplication in the critical I/O path of ZenFS would reduce its service rate, impacting BlobDB's FLUSH operation when writing values to blob files. This reduced service rate could eventually lead to write stalls due to memory pressure [3], [28]. DeNOVA [12] demonstrates the overhead of computing cryptographic hash with increasing data block size, which clearly indicates that adopting inline deduplication for write-optimized file system is not a suitable choice.

2) OPPORTUNITIES FOR OFFLINE DEDUPLICATION

ZenFS assigns a dedicated asynchronous thread for the garbage collection process, presenting an opportunity to integrate offline deduplication within this process. During the garbage collection process, deduplication steps can be performed on each valid data block that will be migrated from a victim zone to a target zone. However, integrating offline deduplication within the garbage collection process increases the critical section of the garbage collection process and does not fully exploit the potential of deduplication. The victim selection algorithm in ZenFS's garbage collection process prioritizes zones with the most invalid data to minimize the negative impact of valid data copying, resulting in fewer data blocks being deduplicated and a less efficient deduplication system.

Decoupling offline deduplication from the garbage collection process, as illustrated in Figure 2, yields a more efficient deduplication system. A zone becomes a candidate for offline deduplication when it's states are changed from open to closed and full, meaning the remaining capacity of the zone is zero. The offline deduplication module marks duplicate data as invalid, allowing the garbage collection process to reclaim the space later. However, this approach has two major shortcomings:

a: AMPLIFIED GARBAGE COLLECTION OVERHEAD

Offline deduplication increases the number of garbage collection calls. Offline deduplication marks valid but duplicate data blocks as invalid, leading to a higher ratio of invalid blocks within a zone and, consequently, more garbage collection operations. The garbage collection process involves valid data copying when unique (valid) and duplicate (invalid) data blocks are stored together within a single zone, as shown in Figure 2. To minimize the overhead of valid data copying, it is necessary to segregate cold data further between unique and duplicate zones.

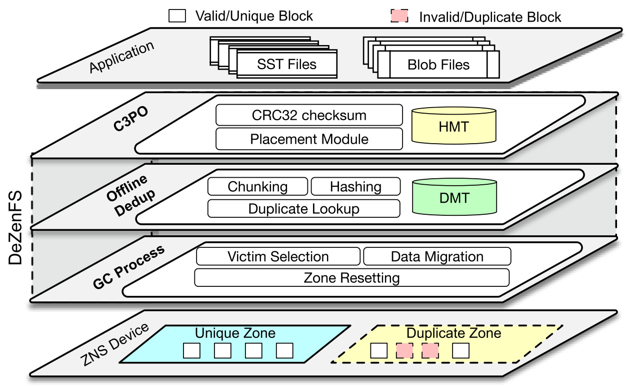


FIGURE 3. Design overview of our proposed CRC32-based data placement for deduplication-enabled ZenFS.

b: AGNOSTIC OF FILE ORGANIZATION

ZenFS's employs a lifetime-based data placement algorithm, as explained in Section II-B, grouping files with similar lifetime into same zones, therefore segregating hot and cold data. Deduplicating files with short lifetime provides minimal benefits while unnecessarily consuming system resources, memory and computational power. Therefore, it is essential to consider the lifetime-based data placement and selectively perform deduplication on long-lived data to provide better long-term space savings.

III. DESIGN OF DEZNS

In this section, we present the design and implementation details of the deduplication-enabled ZNS file system, *DeZNS*.

A. DESIGN GOALS

The design of *DeZNS* revolves around two main goals:

- **Lifetime-aware Selective Deduplication:** ZenFS organizes files within zones based on their lifetime, and deduplication of short-lived files does not yield significant benefits. Additionally, the incorporation of deduplication introduces performance overheads, such as compute-intensive cryptographic fingerprinting and deduplication metadata management. Therefore, it is essential to consider these design choices and perform selective deduplication to maintain ZenFS's performance characteristics.
- **Reducing Data Migration Overhead of the Garbage Collection Process:** Incorporating offline deduplication increases the frequency of garbage collection operations, as it marks duplicate data as invalid, raising the ratio of invalid data. This work focuses on scenarios where the garbage collection process initially does not require

data migration, but the introduction of deduplication necessitates it. Consequently, careful data placement based on uniqueness is crucial to minimize the negative impact of data migration during the garbage collection process.

B. SYSTEM ARCHITECTURE

DeZNS is designed to seamlessly integrate deduplication while achieving the outlined design goals. Figure 3 provides an overview of the proposed design, which consists of three main components.

The first component is a data placement predictor (*C3PO*) that uses a CRC32 checksum to identify incoming data blocks as duplicate or unique. *C3PO* employs a Hinting Metadata Table (HMT) to track written data blocks and uses lifetime hints from file creation to decide on segregation. *C3PO* primarily manages data placement for cold files.

The second component is an *Offline Dedup* module that processes files in the zones marked as duplicate zones, as shown in Figure 3. Since *C3PO* is susceptible to false positives in identification of duplicate data blocks, therefore, to avoid data loss, we also conduct a thorough offline deduplication process on the zones marked as duplicates by the *C3PO*.

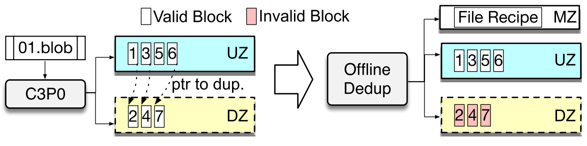
The third component is the GC process, which selects a victim zone based on the duplicate and invalid data ratio. In baseline ZenFS, the garbage collection's victim zone selection is based solely on the invalid ratio of the workload. However, in *DeZNS*, we enhance the selection algorithm by incorporating the duplicate ratio within a zone.

C. C3PO: CRC32-BASED DATA PLACEMENT

Incorporating offline deduplication with baseline data placement algorithm of ZenFS would result in increased valid data migration during garbage collection process. This problem worsens when the zone selected as a deduplication candidate contains unique and duplicate data together, as shown in Figure 2. To mitigate this overhead, identifying duplicate data in an inline manner is essential. However, traditional inline duplicate detection techniques often degrade performance at both the file system and application levels, conflicting with our design goal of maintaining the performance characteristics of ZenFS.

In contrast, several studies have adopted non-cryptographic hashes (e.g.,CRC32 [30] and xxHash [31]) to accelerate the detection of duplicate blocks [26], [27]. However, non-cryptographic hashes have a higher probability of collisions compared to cryptographic hashes, leading to false positives and potential data loss. This data loss occurs when a data block is mistakenly identified as a duplicate due to a hash collision and subsequently reclaimed by the garbage collection process.

To address this limitation, *DeZNS* proposes CRC32-based data placement predictor (*C3PO*) that employs non-cryptographic hashes solely for predicting the placement of data blocks and introduces two new categories of zones, unique and duplicate zones. *C3PO* is only activated for files categorized as cold based on lifetime hint value. *C3PO* omits the hot files that will be invalidated promptly to avoid performing checksum and deduplication operations. *C3PO* consists of three sub-modules: a CRC32-based checksum module, a placement module, and a Hinting Metadata Table (HMT), as shown in Figure 3.



(a) First step of reconstruction (b) Second step of reconstruction

FIGURE 4. An illustration of data placement and two-step reconstruction of files for read operations. MZ, UZ and DZ represent meta, unique, and duplicate zones, respectively.

1) CRC32-BASED CHECKSUM

Cryptographic fingerprints, such as SHA1 [32] and MD5 [33], are widely used for duplicate detection. However, these fingerprints are compute-intensive, requiring tens to hundreds of microseconds to compute, depending on the block size [26], [27]. In contrast, non-cryptographic hashes, such as the CRC32 checksum,[1] can generate a unique identifier for each data block with significantly lower computation time. This allows for predicting the uniqueness of data blocks in a more performance-efficient manner.

When *C3PO* receives a file categorized as cold, it computes the checksum for each data block and queries the Hinting Metadata Table (HMT) to determine if the checksum is already stored. If a checksum is found in the HMT, *C3PO* informs the placement module that the data block is likely a duplicate. Conversely, if a checksum is not found, the data block is considered unique and the placement module assigns it to the unique zone.

Moreover, the HMT of *C3PO* is a hash-based table that keeps track of the already stored checksums. Each entry in HMT is of 8 bytes and can fit in memory to provide fast and efficient tracking. However, HMT can be lost on a sudden power failure, therefore, we periodically flush the HMT contents to the metadata zones.

2) PLACEMENT MODULE

ZenFS organizes files within zones based on their lifetime hint values to reduce data migration overhead. *DeZNS* builds on this concept by allocating two types of zones—unique and duplicate—for files categorized as cold by lifetime hint values. The placement module uses information from the CRC32 checksum module to determine whether data blocks are unique or duplicates, assigning each block to the corresponding zone. Although this segregation might appear to result in random file placement, *DeZNS* maintains the sequential write constraint within each zone.

[1]From hereafter, we use 'checksum' to refer non-cryptographic hashes.

Moreover, the baseline ZenFS writes file sequentially within a single zone, however, the segregation of unique and duplicate zones distribute file's data block into multiple zones. This segregation of data blocks necessitates a robust reconstruction logic for read operations. *DeZNS* implements a two-step reconstruction logic for each file. In the first step, as shown in Figure 4(a), the placement module of *C3PO* tracks data blocks written to the duplicate zone by storing their offsets. It also tracks subsequent data blocks written to the unique zone, and vice versa for files where the first block is a duplicate. For instance, a file '01.blob' contains seven data blocks, with three being duplicates, the offsets of the duplicate blocks are recorded in the unique zones in the same sequence as the original file's data blocks. This allows the file to be reconstructed sequentially by fetching data blocks from the corresponding duplicate zone during read operations. The second step of reconstruction is managed by the *Offline Dedup* module of *DeZNS*.

D. OFFLINE DEDUPLICATION

DeZNS does not solely rely on checksums to identify duplicates but uses them as a hinting mechanism to identify potential duplicates. Due to the susceptibility of checksums to collisions, *DeZNS* employs a cryptographic hash-based *Offline Dedup* in conjunction with the CRC32-based data placement module, as shown in Figure 3. The offline module consists of components responsible for carrying out deduplication operations; chunking, cryptographic fingerprinting, duplicate lookup, and maintaining a deduplication metadata table (DMT).

When a duplicate zone transitions from open to closed and full, it becomes a candidate for offline dedup. There are two approaches to track these zones. The first approach introduces a collaborative data structure that tracks zones requiring deduplication by the o*Offline Dedup*. The second approach is similar to the garbage collection process in ZenFS, where the *Offline Dedup* periodically checks all I/O zones and selects a candidate zone for deduplication. The first approach incurs memory overhead from the collaborative data structure and requires maintaining its consistent state, especially during ungraceful shutdowns.

In contrast, the second approach uses a deduplication flag within each zone's state metadata, allowing the *Offline Dedup* to identify zones for deduplication easily. When a duplicate zone is full, this flag is set, signaling the *Offline Dedup* module. The *Offline Dedup* module then selects a candidate zone, reads the data blocks, and computes their cryptographic fingerprints. If a duplicate is found, the module fetches the unique data block's offset from the DMT and updates the file's reconstruction logic. The duplicate data block is marked as invalid and later reclaimed by the GC process.

Additionally, the *Offline Dedup* is responsible for the second-step of the two-step reconstruction logic for each file. Before marking a duplicate data block as invalid for reclamation by the GC process, the *Offline Dedup* updates the file's reconstruction logic. As the *Offline Dedup* marks duplicate data blocks as invalid, it becomes necessary to update the file's reconstruction logic, as shown in Figure 4(b). The first step of the two-step reconstruction is handled by *C3PO*'s placement module, which stores the offsets of data blocks from the duplicate zone to the unique zone and vice versa.

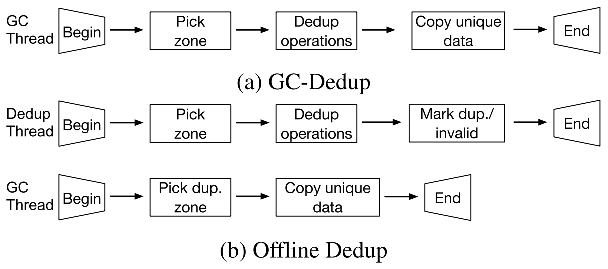


FIGURE 5. Schematics of GC-Dedup and Offline dedup.

During the second step of reconstruction, the *Offline Dedup* constructs reconstruction metadata for files with duplicate data blocks. This metadata includes a list of offsets for all unique data blocks corresponding to a file. Since a file can be distributed across multiple zones, the offsets are constructed by combining the zone identification number with the starting locations of the write pointers where the data blocks are stored within the zones. The reconstruction metadata is written along with other file system metadata in designated metadata zones.

E. GARBAGE COLLECTION

The garbage collection (GC) component of *DeZNS* consists of three sub-modules: victim selection, data migration, and zone resetting, as shown in Figure 3. The victim selection module in *DeZNS* follows two paths; First is for the zones with hot lifetime hint values. For these zones, the victim selection criteria is based on most invalid data within the zone. Second victim selection path is for zones whose lifetime hint values are cold. For these zones, the victim selection module first identifies whether the zone is a unique zone or duplicate zone. *DeZNS* gives higher priority to duplicate zones as it aims to minimize the data migration.

Moreover, the data migration module in *DeZNS*'s GC component considers the segregation of unique and duplicate zones. During the GC process, if a unique data block is encountered, the data migration module identifies the most appropriate zone for migration. It prioritizes the data migration from duplicate zone to unique zone previously allocated alongside the victim zone, unique zone with same lifetime value. If space is available in this corresponding unique zone, the unique data blocks are migrated there. Otherwise, the data migration module moves the data to a different unique zone with a similar lifetime or allocates a new unique zone. Lastly, the zone resetting module simply executes the `zone_reset` command to reset the write pointer of the selected zone.

F. I/O FLOW

The I/O operations in *DeZNS* are categorized into two main types: application I/Os and file system I/Os, as shown in Figure 3. Application I/Os are inline operations performed by the application to read and write data. Within file system I/Os, write operations follow two paths: the default I/O path of baseline ZenFS for data blocks with lifetime hint values categorized as small or medium, and an extended I/O path proposed by *DeZNS* for data blocks categorized as long or extreme. When a data block with a long or extreme lifetime hint is encountered, it is redirected to *C3PO*, where its uniqueness is predicted, and it is placed in the corresponding I/O zone, either unique or duplicate.

The read I/O path for applications remains unchanged, while file system read operations incorporate file reconstruction. Initially, the read operation retrieves the file's metadata, which includes the list of offsets to be read from different zones. Note that a file processed by the *Offline Dedup* can be distributed across several zones. This reconstruction results in random I/O operations, which are typically low-performing I/O operations in SSDs.

IV. EVALUATION

To evaluate the effectiveness of our proposed ideas, we implement *DeZNS* atop ZenFS and compared against three different variants. Our evaluation is based on macro- and micro-benchmarks with low and high deduplication ratio workloads. We first present the evaluation methodology followed by performance analysis.

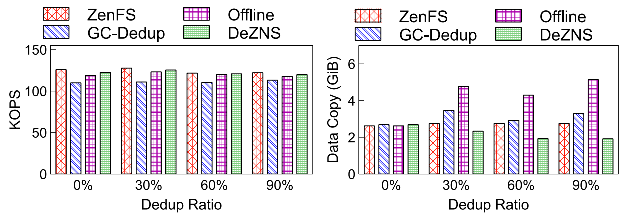
A. EXPERIMENTAL SETUP

1) IMPLEMENTATION

We implemented *DeZNS* on ZenFS v2.1 with Linux v5.10, using BlobDB—a key-value separated variant of RocksDB—as our example application. BlobDB's separate storage of values in blob files provided a prime opportunity for deduplication, as these files are the main space consumers and are assigned a lifetime hint value of *extreme*, since blob files are not frequently updated by the application. We set the garbage collection trigger point at 20% storage utilization, meaning garbage collection is triggered when storage utilization exceeds this threshold. Meanwhile, the decoupled offline deduplication does not follow this threshold and perform deduplication as soon as it finds a zone with concerned lifetime hint value. Moreover, data placement in *DeZNS* is considered at the extent level of ZenFS and offline deduplication is also performed at the extent level.

2) ZNS SSD DEVICES

The ZNS SSDs are not yet commercially available in market, which limits the options for real device evaluation. Therefore, for our initial evaluation of *DeZNS*, we used the state-of-the-art Configurable ZNS (ConfZNS) [34], a ZNS SSD emulator based on FEMU [35]. ConfZNS has been widely adopted in various studies to evaluate different aspects of ZNS SSD. For example, FAR [36], ZACA [37], and BAZA [38] are effectively evaluated on ConfZNS [35].



(a) Performance (b) Valid data migration

FIGURE 6. Performance and garbage collection analysis of large-sized zone device with varying deduplication ratio.

We emulated a 64 GiB ZNS SSD environment with Intel Xeon E5-2640 v4 @ 2.40GHz CPU and 128 GB DDR4 DRAM.

Additionally, different vendors are considering different zone sizes for ZNS SSDs. For example, Western Digital is considering to bring ZNS SSDs with large zone sizes where each zone is about 1 GiB [7], [37], [39], [40], meanwhile Samsung and SK Hynix are considering ZNS SSDs with relatively smaller zone size in between 64 to 256 MiB [18], [41], [42], [43], [44], [45], [46]. Therefore, we conducted experiments with large and small-sized zones using ConfZNS. For large-sized zones, we set the zone size to 1024 MiB while for small-size zones, we set the zone size to 256 MiB. Additionally, we also conducted experiments on real ZNS SSD; however, we only had access to large-sized zone device, therefore our real device evaluation is limited to large-sized zones.

3) BENCHMARKS AND WORKLOADS

Our evaluation comprised micro- and macro-benchmarks. For the micro-benchmark, we developed an in-house synthetic benchmark using a single application thread to perform write I/Os. We used fixed-size key-value pairs (1 KB) and varied the deduplication ratio (0%, 30%, 60%, and 90%) by adjusting the number of duplicate values. For the macro-benchmark, we employed the YCSB benchmark with three different workloads, described in Table 1.

We evaluate the following systems for comparison:

- **ZenFS:** The baseline version of ZenFS.
- **GC-Dedup:** A version of ZenFS where the deduplication operations are integrated within the garbage collection process, performing deduplication during data migration from victim to target zones, as shown in Figure 5(a).
- **Offline:** A version of *DeZNS* utilizing only the offline deduplication module and selecting deduplication candidate based on *long* and *extreme* lifetime hint values, as shown in Figure 5(b).
- ***DeZNS*:** The complete version of *DeZNS*, incorporating *C3PO*, *Offline Dedup*, and garbage collection process.

B. MICRO-BENCHMARK RESULTS

In this section, we evaluate the superiority of *DeZNS* compared to other systems using micro-benchmark. We first present the performance and garbage collection analysis of large-sized zone and small-sized zone devices followed by detailed analysis of garbage collection operation.

1) LARGE-SIZED ZONE

Figure 6 shows the performance and garbage collection analysis results of a micro-benchmark with varying deduplication ratios using large-sized zones. Figure 6(a) presents the user-level performance of the compared systems, while Figure 6(b) shows the amount valid data migrated during garbage collection process. GC-Dedup, which integrates deduplication within the garbage collection process, exhibits the worst performance regardless of the deduplication ratio, with an average degradation of 13% compared to baseline ZenFS. This is due to the increased critical section of the garbage collection process, causing interference between background and foreground operations. Additionally, the amount of data migrated by GC-Dedup increases with the deduplication ratio, as shown in Figure 6(b), because GC-Dedup adopts the default data placement algorithm of baseline ZenFS, where unique and duplicate data reside in the same zone.

Moreover, the offline variant, which uses only the *Offline Dedup* module of *DeZNS*, improves performance compared to GC-Dedup but has a 10% lower performance on average than baseline ZenFS with increasing deduplication ratios, as shown in Figure 6(a). The performance drop in the offline variant is due to the additional data migration overhead during the garbage collection operation with increasing deduplication ratios. Figure 6(b) shows a significant increase in valid data migration during the garbage collection process for the offline variant compared to baseline ZenFS and GC-Dedup. This is because the offline variant marks all duplicate file extents as invalid, which are later reclaimed by the garbage collection process.

In contrast, *DeZNS* with CRC32 checksum-based data placement outperforms the offline variant, with less than a 2% performance drop on average as deduplication increases, as shown in Figure 6(a). This slight performance decline is due to the inline checksum-based data placement by *C3PO*, which segregates unique and duplicate data into separate zones. Additionally, Figure 6(b) shows that with higher deduplication ratios, *DeZNS* reduces valid data migration by up to 28% compared to baseline ZenFS and by 2× compared to the offline variant. This significant reduction is due to the efficient segregation of unique and duplicate data.

Since CRC32 checksum has higher probability of hash collisions than cryptographic fingerprint, we also measured the false positives caused by *C3PO* during data placement. The false positive is considered when a file extent is placed in duplicate zone despite being unique. In *DeZNS*, we measured that less than 0.3% of file extents are misplaced by the *C3PO* which are later migrated by the garbage collection process. Although, the ratio of false positive is significantly less but it would result in data loss if solely relied on CRC32 checksum for duplicate detection. ***DeZNS effectively reduces the valid data migration due its efficient data segregation mechanism with minimal performance impact on user-level.***

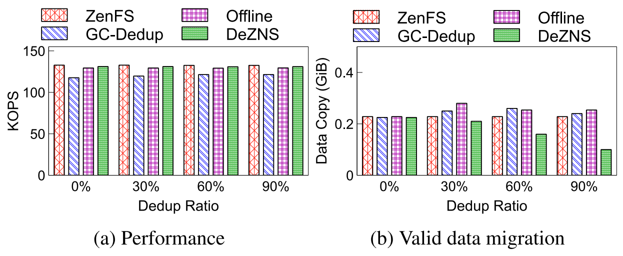


FIGURE 7. Performance and garbage collection analysis of small-sized zone device with varying deduplication ratio.

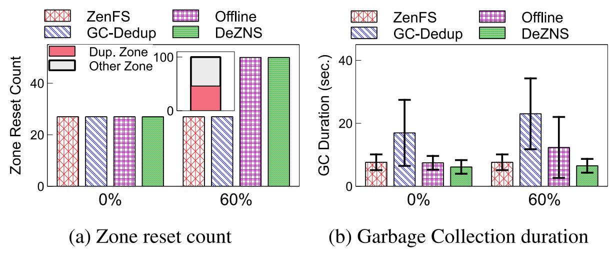


FIGURE 8. Analysis of garbage collection operation. (a) Total number of times `zone_reset` command is executed for workloads. The sub-graph shows the break of `zone_reset` command between different zones of *DeZNS*. (b) The average and standard deviation of garbage collection operation's duration.

2) SMALL-SIZED ZONE

Figure 7 shows the performance and garbage collection analysis results of a micro-benchmark with varying deduplication ratios using small-sized zones. Figure 7(a) presents the user-level performance, while Figure 7(b) shows the amount of valid data migrated during the garbage collection process. Similar to the results with large-sized zones, GC-Dedup suffers from the worst performance due to the reduced service rate of the garbage collection process, and the amount of valid data migration increases with the deduplication ratio, as shown in Figure 7(b). The offline variant improves performance by decoupling offline deduplication from the garbage collection process, but the amount of valid data migrated during the garbage collection process still increases with the deduplication ratio. In contrast, *DeZNS* maintains its performance and valid data migration characteristics even with small-sized zones. ***Although the valid data migration is not as significant as with large-sized zones, DeZNS is able to further reduce valid data migration and achieve comparable performance to baseline ZenFS.***

3) GARBAGE COLLECTION ANALYSIS

In this section, we provide details of garbage collection analysis using micro-benchmark with large-sized zones. Since small-sized zones do not significantly impact garbage collection process, the remaining experiments focus on large-sized zones. We analyze two representative micro-benchmark

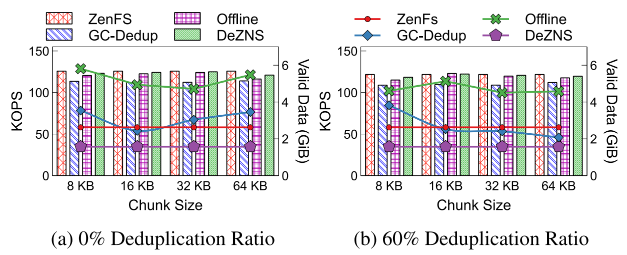


FIGURE 9. Performance and garbage collection analysis with chunking of enabled.

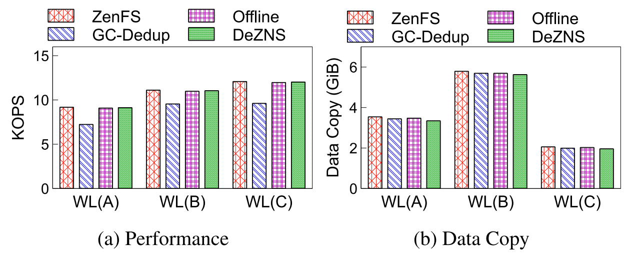


FIGURE 10. Performance and garbage collection analysis of YCSB benchmark.

workloads based on deduplication ratios: 0% (worst case) and 60% (more realistic).

Figure 8(a) shows the total number of zone reset operations during workload execution. With a 0% deduplication ratio, the zone reset count remains the same across all systems. However, with a 60% deduplication ratio, the reset count for the offline and *DeZNS* systems is more than twice that of baseline ZenFS. This increase is due to the invalidation of duplicate data/zones in the offline and *DeZNS* systems, respectively. The sub-graph in Figure 8(a) breaks down *DeZNS*'s total zone reset count, showing that over 45% of zone resets occur in duplicate zones. While the other zones in sub-graph mostly represent zones with lifetime hint value small, medium, and large. The ratio of unique zones in this breakdown is significantly low.

Figure 8(b) presents the average duration of the garbage collection process across compared systems. GC-Dedup exhibits the longest garbage collection duration, as deduplication is part of the garbage collection process's critical section. With a 60% deduplication ratio, the offline system has the second-highest garbage collection duration due to high data migration overhead. In contrast, *DeZNS* achieves a lower garbage collection duration by avoiding unnecessary data migration. Although *DeZNS*'s garbage collection duration is shorter than other systems, its user-level performance is slightly lower than baseline ZenFS due to additional steps during data placement. ***DeZNS increases zone reset calls based on the dataset's duplicate ratio, with most resets occurring in duplicate zones.***

4) IMPACT OF DATA CHUNKING

In previous experiments, we used the extent size (128 KB) of baseline ZenFS as the data chunk size. In this sub-section, we enable fine-grained chunking of extents and present the performance and garbage collection analysis, conducted on large-sized zones.

a: PERFORMANCE ANALYSIS

The bar graphs in Figure 9 show the performance of compared systems with varying chunk sizes. It can be observed the GC-Dedup has consistently demonstrated worst performance. This is because the enabling of chunking further increases the critical section of the garbage collection process as multiple iterations of deduplication operations are performed based on the chunk size. However, this performance does not go beyond 15% as the garbage collection is only triggered after the garbage collection trigger threshold crosses the pre-defined utilization threshold. In contrast, the offline and *DeZNS* has lower performance than ZenFS because of more number of deduplication operations performed within a single zone. However, *DeZNS* outperforms offline due to its effective data segregation.

b: GARBAGE COLLECTION ANALYSIS

The line graphs in Figure 9 display valid data copy overhead with increasing chunk size. The offline version suffers the highest valid data copy overhead regardless of chunk size. In contrast, GC-Dedup's valid data copy decreases with larger chunk sizes due to the significant amount of duplicate data in the workload, Figure 9(b). For *DeZNS*, valid data copy overhead remains constant regardless of chunk size, due to its efficient data placement prediction.

C. MACRO-BENCHMARK RESULTS

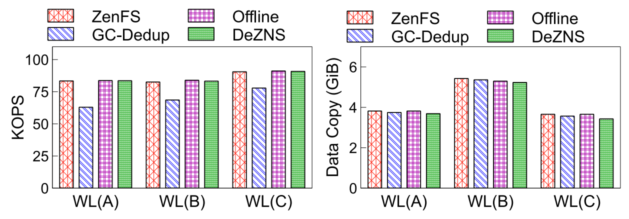
Since ZenFS is tightly coupled with key-value stores, RocksDB and BlobDB, we evaluated the efficiency of *DeZNS* compared to other systems using well-known key-value store benchmark, YCSB. Table 1 presents the details of workloads used for experiments in this section. The *zipfian* distribution is used for queries. We loaded same amount of data before running the workload. We considered the large-sized zones and chunk size as extent size of ZenFS. We first present the performance analysis and then discuss the garbage collection process.

TABLE 1. YCSB benchmark and workload description. U, I, and R represents Update, Insert, and Read operations.

	Workload	Characteristics	KV pair	Dedup Ratio
YCSB	WL(A)	80% U & 20% I	1 KB	0%
	WL(B)	50% U & 50% I		
	WL(C)	50% U & 50% R		

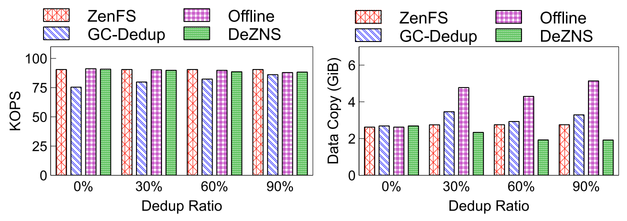
1) PERFORMANCE ANALYSIS

Figure 10(a) presents the performance results of the YCSB benchmark. WL(A) and WL(B) are write-only workloads, while WL(C) is a mixed workload. It is evident that regardless of the workload type, GC-Dedup consistently exhibits lower performance due to the increased critical section of the



(a) Performance (b) Data Copy

FIGURE 11. **Macro-benchmark on Real ZNS device with different access pattern.**



(a) Performance (b) Data Copy

FIGURE 12. **Micro-benchmark on Real ZNS device. We emulated WL(C) of YCSB benchmark with varying duplicate ratio.**

garbage collection process. Since YCSB does not generate duplicate values, and both invalid and valid data are written together in the zone, GC-Dedup suffers a 22% performance drop. Additionally, all workloads contain update operations, increasing the workload for the garbage collection process. In update-intensive scenarios, values are invalidated by the application, triggering garbage collection on more zones.

Conversely, the Offline and *DeZNS* systems reduce the critical section of the garbage collection process by decoupling deduplication operations. Both systems perform offline deduplication on blob files before garbage collection is triggered, allowing them to stay ahead of the garbage collection process and effectively parallelizing tasks. Furthermore, WL(C) includes read operations, and deduplication systems are known for generating random read operations. However, since the workload does not generate any duplicates, it does not suffer from random reads, and the read path remains consistent with baseline ZenFS. The exploration of random reads is left for future work.

2) GARBAGE COLLECTION ANALYSIS

Figure 10(b) presents the valid data copy during the garbage collection operation for compared systems. It can be observed that the amount of data data copy stays the same for all workloads. This is majorly because the benchmark does not generate any duplicate data. However, it can be noticed that the the amount of valid data copy is greatly affected by the workload types.

For instance, the WL(A) is update-intensive workload with 20% new inserts, therefore, it has moderate valid data copy overhead. Moreover, WL(B) being update-heavy workload with 50% new inserts, has the highest valid data copy overhead as the ratio of zones with valid and invalid data is higher. On the other hand, WL(C) is update-heavy but only includes read operations therefore, the ratio of mixed zones with valid and invalid data becomes lower hence less valid data copy. ***The effective parallelization of Offline Dedup and garbage collection process in DeZNS results in reduced interference between background and foreground jobs.***

D. EXPERIMENTS ON REAL DEVICE

To validate our previous experiments, we conducted tests using a real device. We used the ZMe ZNS SSD, which consists of 904 zones, each with a capacity of 1077MB, and a total size of 1TB [40]. For ease of experimentation, we mounted only 64 zones in ZenFS and conducted experiments using Macro- and Micro-benchmarks.

1) MACRO-BENCHMARK RESULTS

Figure 11 shows the performance and garbage collection analysis of three different workloads of YCSB benchmark. The details of workload is presented in Table 1. It can be observed from Figure 11(a) that *DeZNS* maintains its performance characteristics equal to baseline ZenFS while GC-Dedup suffers from performance drop due to inline deduplication at the garbage collection operation. The remaining insights in Figure 11 are similar to what we explained in Figure 10. Meanwhile, Figure 11(b) shows the valid data copy during the zone cleaning operation. Since YCSB does not exhibit any duplicate ratio therefore the amount of valid data copied during zone cleaning operation stays equivalent to the baseline ZenFS, as shown in Figure 11(b).

Additionally, *DeZNS* is designed with two primary goals: maintaining the performance and reducing the write amplification (valid data copy). Figure 11 demonstrate that *DeZNS* achieve these two goals. Moreover, the observed reduction in data migration (valid data copy overhead) directly translates to a reduction in write amplification, as *DeZNS* minimizes the additional writes to zones by segregating unique and duplicate data. This reduction in unnecessary writes helps to extend the longevity of the ZNS SSD, even if it does not immediately reflect as a substantial performance gain in terms of operations per second. The key benefit lies in reducing the wear on the ZNS SSD, which contributes to its overall lifespan.

2) MICRO-BENCHMARK RESULTS

We conducted these experiments in two phases. In first phase, we loaded the 50% of the dataset and then in the second phase, we run the benchmark which simulated the WL(C) of YCSB with 50% writes and 50% read operations. Note that, the read operations are performed on already written data in a uniform distribution.

a: THROUGHPUT AND GARBAGE COLLECTION ANALYSIS

Figure 12 presents the performance and garbage collection analysis from the micro-benchmark conducted on a real ZNS device, where the workload's duplicate ratio varies.

TABLE 2. Put and Get latency analysis of the compared systems with 60% duplicate data. The values presented are in Micro-second unit.

Write	**Average**	**90th**	**99th**	**99.9**	**99.99th**
ZenFS	9.8	13.23	17.15	20.94	70.33
GC-Dedup	12.3	17.98	25.52	30.11	90.23
Offline	9.8	13.23	17.15	20.94	70.33
DeZNS	9.9	14.5	18.7	21.20	72.55

Read	**Average**	**90th**	**99th**	**99.9**	**99.99th**
ZenFS	19.44	26.09	41.12	60.29	69.63
GC-Dedup	19.44	26.09	41.12	60.29	69.63
Offline	21.1	29.7	49.91	64.25	81.0
DeZNS	21.1	29.7	49.91	64.25	81.0

This analysis is crucial to understanding the impact of *DeZNS* on both performance and write amplification in comparison to baseline ZenFS.

In Figure 12(a), we observe that both the Offline and *DeZNS* maintain performance levels comparable to the baseline ZenFS across all deduplication ratio. This result highlights that *DeZNS* and Offline deduplication methods can support high-performance operations without introducing additional overhead from deduplication and data placement. In contrast, GC-Dedup experiences a noticeable performance degradation, particularly at lower deduplication ratios. This performance drop is attributed to the inline deduplication within garbage collection, which becomes more pronounced when the data lacks redundancy. Notable, the GC-Dedup's read performance remains stable since the read operations are served directly from the LSM-tree's components that are yet not processed by the garbage collection. On the other hand, the impact of redirection and random read operations in the Offline and *DeZNS* is negligible as the modern SSD's have similar sequential and random read performance [37], [47].

Figure 12(b) provides a detailed analysis of garbage collection efficiency in terms of data copy reduction, which directly correlates with reduced write amplification. Across all deduplication ratios, *DeZNS* achieves the lowest data copy volume compared to ZenFS, GC-Dedup, and Offline deduplication configurations. This outcome underscores *DeZNS*'s effectiveness in minimizing write amplification by segregating duplicate and unique data, thus significantly lowering the overhead of valid data copy operations. As the deduplication ratio increases, *DeZNS*'s advantage in reducing data copy becomes even more apparent. For instance, at high deduplication ratio (60% and 90%), *DeZNS* demonstrates a substantial reduction in data copy, outperforming all other configurations. This result indicates that *DeZNS* can leverage higher redundancy in workloads to more effectively manage write amplification, a critical advantage for systems based on ZNS SSDs.

b: LATENCY ANALYSIS

Table 2 provides a detailed analysis of the Put (write) and Get (read) latency across different percentile levels for compared systems under 60% duplicate ratio workload from micro-benchmark. The latencies are measured in microseconds (μm), offering insights into the performance consistency of each system, particularly at high percentiles where tail latency becomes a significant concern.

Across all systems, the average write latency remains fairly consistent, with ZenFS and Offline showing the lowest latency at 9.8 μm, and *DeZNS* at a similar level. This slight increase is due to the computation of CRC32 checksum when the data is placed within the zones. GC-Dedup exhibits a slightly higher average write latency, likely due to the overhead introduced by inline deduplication and garbage collection operations. As we move to high percentiles, the variance in performance becomes more apparent. GC-Dedup exhibits a significant increase in write latency at high percentiles, reaching 90.23 μm at the 99.99th percentile. This result highlights the performance penalty associated with GC-Dedup's inline deduplication and garbage collection process under high redundancy workloads, which adds latency during data copy operations. *DeZNS* on the other hand, maintains relatively low and stable write latencies at higher percentiles, with a latency of 72.55 μm at 99.99th percentile. This stability reflects *DeZNS*'s efficient handling of redundant data, reducing the impact on write performance by segregating duplicate data.

For read operations, ZenFS and GC-Dedup have the lowest average read latency at 19.44 μm, while offline and *DeZNS* show a slightly higher latency of 21.1 μm. This minor increase for Offline and *DeZNS* is attributed to the deduplication approach, which introduces redirection and random reads. At high percentiles, especially the 99.99th percentile, both Offline and *DeZNS* show similar read latency results (81.0 μm). Although this is marginally higher than ZenFS and GC-Dedup, the difference remains modest, suggesting that *DeZNS*'s deduplication odes not impose significant penalties on read latency. Additionally, the system's ability to handle 50% of reads directly from the LSM-tree components helps keep latency low across the percentile spectrum.

TABLE 3. Storage requirements after the execution of the macro- and micro-benchmark.

Benchmark	Workloads	ZenFS	GC-Dedup	Offline	DeZNS
Macro-	All	10	10	10	10
Micro-	0%	20	20	20	20
	30%	20	16	14	14
	60%	20	16	8	8
	90%	20	14	2	2

E. SPACE UTILIZATION AND METADATA OVERHEAD

1) SPACE UTILIZATION

We measured the total storage requirements for both macro- and micro-benchmarks. In the macro-benchmark, we controlled the duplicate ratio, whereas the micro-benchmark did not support this feature. Table 3 details the storage requirements of all compared systems. Dedup-integrated systems show lower storage requirements than baseline ZenFS as the deduplication ratio increases. However, GC-Dedup is less effective than Offline and *DeZNS* because GC-Dedup only performs deduplication when the garbage collection process selects a victim zone with duplicate data. In contrast, Offline and *DeZNS* parallelize both deduplication and garbage collection tasks, marking duplicate data as invalid, which is later reclaimed by the garbage collection process. This approach reduces overall storage requirements.

2) METADATA OVERHEAD

DeZNS introduces two additional metadata structures: the Hinting Metadata Table (HMT) and the Deduplication Metadata Table (DMT), as shown in Figure 3. Both require extra memory and storage. The HMT, stored entirely in memory for fast access, uses 8 bytes per entry to reference a 128 KB ZenFS extent, requiring only 64 MB of memory to manage 1 TB of data. For failure consistency, *DeZNS* periodically flushes HMT contents to the metadata zone. The DMT has higher memory and storage demands, with each entry being 32 bytes (20 bytes for the cryptographic fingerprint, 4 bytes for the reference count, and 8 bytes for the offset). The DMT is also stored in memory to ensure fast access for the offline deduplication module, avoiding random lookups and preventing high write amplification. Like the HMT, the DMT is periodically flushed to the metadata zone for failure consistency. Additionally, file reconstruction metadata is directly written to metadata zones. Extensive research on optimizing deduplication metadata can be applied to DeZNS, as metadata optimization solutions complement our proposed solution.

DeZNS achieves significant space savings based on the deduplication ratio of workload while maintaining the minimal performance impact.

V. RELATED WORK

A. ZNS SSD

There have been several studies focused on optimizing systems for ZNS SSDs [18], [34], [41], [48], [49], [50]. Waltz [48] improved tail-latency in LSM-ZNS storage's PUT(k,v) operation by implementing zone-append [49]. ZNSwap [50] addressed the issue of performance isolation failure in multi-tenant environments caused by Garbage Collection, by integrating ZNS SSD into the kernel subsystem's swap memory area, setting it apart from conventional SSDs. ZNS+ [6] enhanced file system performance by offloading copy operations to the device in the F2FS log-structured file system [51]. ConfZNS [34] introduced the concepts of Full-Unit zone (FU-zone) and Single-Unit zone (SU-zone) layouts based on internal parallelism, and provided a real-latency ZNS SSD emulator for researchers. Bae et al. [18] boosted read performance and latency in small-zone layouts by using kernel-level internal parallelism profiling and an interference-aware scheduler. eZNS [41] proposed a method to fully exploit the internal parallelism of ZNS through Zone Ballooning.

B. DATA DEDUPLICATION IN FILE SYSTEM

Deduplication has become essential in both archival and primary storage, including file systems [10], [12], [27], [52], [53], [54], [55], databases [56], [57], and distributed storage systems [22], [23], [24], [25]. Recent works have adopted deduplication in NVM-based storage systems, such as NV-Dedup [27], LO-Dedup [58], DeNOVA [12], and LightDedup [26], which proposed both inline and offline deduplication techniques. Notable file system-level works include iDedup [10], PDFS [11], D^3 [59], and ProSy [60]. For object storage systems, Grate [22], [23] proposed inline deduplication for Ceph-based distributed storage systems.

In recent years, deduplication has become a key focus in file system research, with numerous attempts to integrate it into existing systems [10], [11], [12], [13], [15]. Recent advancements in deduplication for file systems include SmartDedup [13] and F2DFS [15]. Both target F2FS and propose a hybrid approach, combining inline and offline deduplication. SmartDedup is designed for resource-constrained storage devices and adopts adaptive deduplication based on the number of duplicates over time to reduce the access overhead on storage devices. F2DFS also employs a hybrid approach with a file system-coupled design and optimizes deduplication metadata to reduce many-to-one mapping updates during segment cleaning. However, both works focus on conventional SSDs and do not consider the impact of deduplication on the GC process in ZNS-based file systems. Moreover, DenKV [28] is the only work that attempted to incorporate deduplication in BlobDB during the Flush operation when values are written to blob files. However, DenKV suffers from frequent write stalls and reduces the overall performance of BlobDB due to reduced service rate of Flush operation.

However, the existing studies fall short on ZNS SSD due to two reasons. First, the existing solutions do not consider the impact of deduplication on the garbage collection operation which is the major source of write amplification in SSDs. Second, the existing solutions do not provide solutions to segregate the unique and duplicate data blocks as the data placement in traditional SSDs is the responsibility of the internal hardware of the storage device. ZNS SSDs offload this responsibility to the host therefore, providing the flexibility to the deduplication systems to segregate the unique and duplicate data blocks which eventually lead to reduced write amplification and longevity of SSD's lifespan. *DeZNS* leverage this opportunity in ZNS SSD and proposed solutions to maintain performance and reduce write amplification.

VI. CONCLUSION

In conclusion, ZNS interface places the responsibility of data management on upper-level applications, necessitating the use of garbage collection (GC) through the *zone-reset* command. While application-level GC can cause performance degradation due to the significant overhead of copying valid data, this challenge is exacerbated by the larger GC units in ZNS SSDs. However, the negative impact of larger GC units can be alleviated by making GC operations interruptible, allowing I/O requests to be processed during zone resets or block reclamation. Offline deduplication, a commonly used storage optimization technique in ZNS-based file systems like ZenFS, presents additional complexities. It requires careful management of file allocation based on data lifetime to avoid deduplicating hot data, and the co-location of unique and duplicate data blocks further increases the valid data copy overhead during GC.

To address these challenges, we introduced *DeZNS*, an innovative data placement strategy for deduplication-enabled ZenFS. *DeZNS* mitigates the increased valid data migration overhead during GC by using a lightweight CRC32 checksum-based method to predict duplicates with minimal performance cost, while also segregating unique and duplicate data blocks. This segregation reduces the migration overhead during GC, and the interruptible GC mechanism ensures that ongoing I/O requests are not delayed, maintaining ZenFS performance. Furthermore, *DeZNS* integrates an offline deduplication module that operates on these segregated zones. Our evaluation demonstrates that *DeZNS* reduces valid data migration by 28% compared to the baseline ZenFS, and by up to 2× compared to naive offline deduplication in micro-benchmarks, highlighting its effectiveness in improving performance in deduplication-enabled ZenFS environments.

REFERENCES

[1] Meta. (2012). *Rocks DB*. Accessed: Jan. 17, 2024. [Online]. Available: http://rocksdb.org

[2] L. Lu, T. S. Pillai, H. Gopalakrishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "WiscKey: Separating keys from values in SSD-conscious storage," *ACM Trans. Storage*, vol. 13, no. 1, pp. 1–28, Mar. 2017.

[3] O. Balmau, F. Dinu, W. Zwaenepoel, K. Gupta, R. Chandhiramoorthi, and D. Didona, "SILK: Preventing latency spikes in log-structured merge key-value stores," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 19)*, Jan. 2019, pp. 753–766.

[4] J. Yu, S. H. Noh, Y.-R. Choi, and C. J. Xue, "ADOC: Automatically harmonizing dataflow between components in log-structured key-value stores for improved performance," in *Proc. 21st USENIX Conf. File Storage Technol.*, 2023, pp. 65–80.

[5] Meta. (2018). *BlobDB*. Accessed: Jun. 5, 2024. [Online]. Available: http://rocksdb.org

[6] K. Han, H. Gwak, D. Shin, and J.-Y. Hwang, "ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction," in *Proc. 15th USENIX Symp. Operating Syst. Design Implement. (OSDI 21)*, Jan. 2021, pp. 147–162.

[7] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. Le Moal, G. R. Ganger, and G. Amvrosiadis, "ZNS: Avoiding the block interface tax for flash-based SSDs," in *Proc. 2021 USENIX Annu. Tech. Conf. (USENIX ATC 21)*, pp. 689–703, 2021.

[8] H.-R. Lee, C.-G. Lee, S. Lee, and Y. Kim, "Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs," in *Proc. 14th ACM Workshop Hot Topics Storage File Syst.*, New York, NY, USA, Jun. 2022, pp. 93–99.

[9] *Zenfs*, Western Digit. Corp., San Jose, CA, USA, Feb. 2022.

[10] K. Srinivasan, T. Bisson, G. R. Goodson, and K. Voruganti, "IDedup: Latency-aware, inline data deduplication for primary storage," in *Proc. 12th USENIX Conf. File Storage Technol. (FAST 12)*, Feb. 2012, pp. 1–14.

[11] H. Yu, X. Zhang, W. Huang, and W. Zheng, "PDFS: Partially dedupped file system for primary workloads," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 863–876, Mar. 2017.

[12] H. Kwon, Y. Cho, A. Khan, Y. Park, and Y. Kim, "DENOVA: Deduplication extended Nova file system," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2022, pp. 1360–1371.

[13] Q. Yang, R. Jin, and M. Zhao, "SmartDedup: Optimizing deduplication for resource-constrained devices," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, Jan. 2019, pp. 633–646.

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

[15] T. Zhang, R. Chen, Z. Li, C. Gao, C. Wang, and J. Shu, "Design and implementation of deduplication on F2FS," *ACM Trans. Storage*, vol. 20, no. 4, pp. 1–50, Nov. 2024.

[16] *Zoned Storage*, Western Digit. Corp., San Jose, CA, USA, 2022.

[17] M. Bjørling, "From open-channel SSDs to zoned namespaces," in *Proc. Linux Storage Filesyst. Conf. (Vault)*, Jan. 2019, p. 20.

[18] H. Bae, J. Kim, M. Kwon, and M. Jung, "What you can't forget: Exploiting parallelism for zoned namespaces," in *Proc. 14th ACM Workshop Hot Topics Storage File Syst.*, Jun. 2022, pp. 79–85.

[19] T. Stavrinos, D. S. Berger, E. Katz-Bassett, and W. Lloyd, "Don't be a blockhead: Zoned namespaces make work on conventional SSDs obsolete," in *Proc. Workshop Hot Topics Operating Syst.*, New York, NY, USA, Jun. 2021, pp. 144–151.

[20] G. Choi, K. Lee, M.-H. Oh, J. Choi, J. Jhin, and Y. Oh, "A new LSM-style garbage collection scheme for ZNS SSDs," in *Proc. 12th USENIX Workshop Hot Topics Storage File Syst.*, Jan. 2020, pp. 1–13.

[21] P. Desnoyers, "Empirical evaluation of NAND flash memory performance," *ACM SIGOPS Operating Syst. Rev.*, vol. 44, no. 1, pp. 50–54, Mar. 2010.

[22] A. Khan, P. Hamandawana, and Y. Kim, "A content fingerprint-based cluster-wide inline deduplication for shared-nothing storage systems," *IEEE Access*, vol. 8, pp. 209163–209180, 2020.

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

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

[25] P. Bartus and E. Arzuaga, "GDedup: Distributed file system level deduplication for genomic big data," in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, Jul. 2018, pp. 120–127.

[26] J. Qiu, Y. Pan, W. Xia, X. Huang, W. Wu, X. Zou, S. Li, and Y. Hua, "Light-dedup: A light-weight inline deduplication framework for non-volatile memory file systems," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 23)*, 2023, pp. 101–116.

[27] C. Wang, Q. Wei, J. Yang, C. Chen, Y. Yang, and M. Xue, "NV-Dedup: High-performance inline deduplication for non-volatile memory," *IEEE Trans. Comput.*, vol. 67, no. 5, pp. 658–671, May 2018.

[28] S. Jamil, A. Khan, K. Kim, J.-K. Lee, D. An, T. Hong, S. Oral, and Y. Kim, "DenKv: Addressing design trade-offs of key-value stores for scientific applications," in *Proc. IEEE/ACM Int. Parallel Data Syst. Workshop (PDSW)*, Nov. 2022, pp. 20–25.

[29] J. Paulo and J. Pereira, "A survey and classification of storage deduplication systems," *ACM Comput. Surveys*, vol. 47, no. 1, pp. 1–30, Jun. 2014.

[30] G. Castagnoli, S. Brauer, and M. Herrmann, "Optimization of cyclic redundancy-check codes with 24 and 32 parity bits," *IEEE Trans. Commun.*, vol. 41, no. 6, pp. 883–892, Jun. 1993.

[31] Y. Collet. (2016). *XxHash: Extremely Fast Hash Algorithm*. Accessed: Jun. 2, 2024. [Online]. Available: https://github.com/Cyan4973/xxHash

[32] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "The first collision for full SHA-1," in *Proc. 37th Annu. Int. Cryptol. Conf. Adv. Cryptology (CRYPTO)*, Santa Barbara, CA, USA. Cham, Switzerland: Springer, Jan. 2017, pp. 570–596.

[33] M. Ciampa, *CompTIA Security+ 2008 In Depth*. Boston, MA, USA: Course Technology Press, 2008.

[34] I. Song, M. Oh, B. S. J. Kim, S. Yoo, J. Lee, and J. Choi, "ConfZNS: A novel emulator for exploring design space of ZNS SSDs," in *Proc. 16th ACM Int. Conf. Syst. Storage*, New York, NY, USA, Jun. 2023, pp. 71–82.

[35] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, and H. S. Gunawi, "The case of FEMU: Cheap, accurate, scalable and extensible flash emulator," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST 18)*, Feb. 2018, pp. 83–90.

[36] S. Byeon, J. Ro, S. Jamil, J.-U. Kang, and Y. Kim, "A free-space adaptive runtime zone-reset algorithm for enhanced ZNS efficiency," in *Proc. 15th ACM Workshop Hot Topics Storage File Syst.*, Jul. 2023, pp. 109–115.

[37] S. Byeon, J. Ro, J. Y. Han, J.-U. Kang, and Y. Kim, "Ensuring compaction and zone cleaning efficiency through same-zone compaction in zns key-value store," in *Proc. 38th Int. Conf. Massive Storage Syst. Technol. (MSST)*, vol. 24, 2024, pp. 1–23.

[38] H. Hwangbo, J. Ro, S. Byeon, S. Jamil, J. Y. Han, J. Hwang, and Y. Kim, "Towards a unified garbage collection strategy in ZNS key-value store file systems using same-victim GC," in *Proc. 32nd Int. Conf. Model., Anal. Simul. Comput. Telecommun. Syst. (MASCOTS)*, Oct. 2024, pp. 1–8.

[39] K. Doekemeijer, N. Tehrany, B. Chandrasekaran, M. Bjørling, and A. Trivedi, "Performance characterization of NVMe flash devices with zoned namespaces (ZNS)," in *Proc. IEEE Int. Conf. Cluster Comput. (CLUSTER)*, Oct. 2023, pp. 118–131.

[40] W. Digital. (2021). *Western Digital Ultrastar Dc ZN540*. [Online]. Available: https://www.westerndigital.com/products/internal-drives/data-center-drives/ultrastar-dc-zn540-nvme-ssd

[41] J. Min, C. Zhao, M. Liu, and A. Krishnamurthy, "EZNS: An elastic zoned namespace for commodity zns SSDs," in *Proc. 17th USENIX Symp. Operating Syst. Design Implement. (OSDI 23)*, Jul. 2023, pp. 461–477.

[42] M. Im, K. Kang, and H. Yeom, "Accelerating RocksDB for small-zone ZNS SSDs by parallel I/O mechanism," in *Proc. 23rd Int. Middleware Conf. Ind. Track*, Nov. 2022, pp. 15–21.

[43] J. Y. Ha and H. Y. Yeom, "ZCeph: Achieving high performance on storage system using small zoned ZNS SSD," in *Proc. 38th ACM/SIGAPP Symp. Appl. Comput.*, Mar. 2023, pp. 1342–1351.

[44] M. Oh, S. Yoo, J. Choi, J. Park, and C.-E. Choi, "ZenFS+: Nurturing performance and isolation to ZenFS," *IEEE Access*, vol. 11, pp. 26344–26357, 2023.

[45] *Samsung Introduces its First ZNS SSD with Maximized User Capacity and Enhanced Lifespan*, Samsung Electron., Suwon-si, South Korea, 2022.

[46] hynix Inc. (Mar. 2019). *Sk Hynix Demonstrates Industry's First Zns-based Ssd Solution for Data Centers*. Accessed: Oct. 8, 2024. [Online]. Available: https://news.skhynix.com/sk-hynix-demonstrates-industrys-first-zns-based-ssdsolution-for-data-centers-2/

[47] B. Lepers, O. Balmau, K. Gupta, and W. Zwaenepoel, "KVell: The design and implementation of a fast persistent key-value store," in *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 447–461.

[48] J. Lee, D. Kim, and J. W. Lee, "WALTZ: Leveraging zone append to tighten the tail latency of LSM tree on ZNS SSD," *Proc. VLDB Endowment*, vol. 16, no. 11, pp. 2884–2896, Jul. 2023.

[49] M. Bjørling, "Zone append: A new way of writing to zoned storage," in *Proc. Linux Storage Filesystems Conf. (Vault)*, Jan. 2020, pp. 1–16.

[50] S. Bergman, N. Cassel, M. Bjørling, and M. Silberstein, "ZNSwap: Unblock your swap," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC 22)*, vol. 19, Carlsbad, CA, USA, Jul. 2022, pp. 1–25.

[51] C. Lee, D. Sim, J.-Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proc. 13th USENIX Conf. File Storage Technol.*, Feb. 2015, pp. 273–286.

[52] A. Bacs, S. Musaev, K. Razavi, C. Giuffrida, and H. Bos, "DUPEFS: Leaking data over the network with filesystem deduplication side channels," in *Proc. 20th USENIX Conf. File Storage Technol. (FAST 22)*, 2022, pp. 281–296.

[53] M. Lu, D. Chambliss, J. Glider, and C. Constantinescu, "Insights for data reduction in primary storage: A practical analysis," in *Proc. 5th Annu. Int. Syst. Storage Conf.*, Jun. 2012, pp. 1–7.

[54] J.-H. Kim, C. Lee, S. Y. Lee, I. Son, J. Choi, S. Yoon, H.-U. Lee, S. Kang, Y. Won, and J. Cha, "Deduplication in SSDs: Model and quantitative analysis," in *Proc. 28th Symp. Mass Storage Syst. Technol. (MSST)*, Apr. 2012, pp. 1–12.

[55] Z. Cao, H. Wen, X. Ge, J. Ma, J. Diehl, and D. H. C. Du, "TDDFS: A tier-aware data deduplication-based file system," *ACM Trans. Storage*, vol. 15, no. 1, pp. 1–26, Feb. 2019.

[56] J. Park, J. Kim, Y. Kim, S. Lee, and O. Mutlu, "DeepSketch: A new machine learning-based reference search technique for post-deduplication delta compression," in *Proc. 20th USENIX Conf. File Storage Technol. (FAST 22)*, Jan. 2022, pp. 247–264.

[57] L. Xu, A. Pavlo, S. Sengupta, and G. R. Ganger, "Online deduplication for databases," in *Proc. ACM Int. Conf. Manage. Data*, May 2017, pp. 1355–1368.

[58] W. Chen, Z. Chen, D. Li, H. Liu, and Y. Tang, "Low-overhead inline deduplication for persistent memory," *Trans. Emerg. Telecommun. Technol.*, vol. 32, no. 8, p. 4079, Aug. 2021.

[59] J. Yin, Y. Tang, S. Deng, Y. Li, and A. Y. Zomaya, "D^3: A dynamic dual-phase deduplication framework for distributed primary storage," *IEEE Trans. Comput.*, vol. 67, no. 2, pp. 193–207, Feb. 2018.

[60] X. Du, W. Hu, Q. Wang, and F. Wang, "ProSy: A similarity based inline deduplication system for primary storage," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Aug. 2015, pp. 195–204.

SAFDAR JAMIL received the B.E. degree in computer systems engineering from the Mehran University of Engineering and Technology (MUET), Jamshoro, Pakistan, in 2017. He is currently pursuing the integrated M.S. and Ph.D. degree with the Department of Computer Science and Engineering, Sogang University, Seoul, South Korea. He is a member with the Data-Intensive Computing and Systems Laboratory, Department of Computer Science and Engineering, Sogang University. His research interests include scalable indexing data structures and algorithms, key-value stores, zoned namespace storage, and storage optimization techniques.

JOSEPH RO received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 2023, where he is currently pursuing the M.S. degree with the Department of Computer Science and Engineering. Currently, he is a member with the Data-Intensive Computing and Systems Laboratory, Department of Computer Science and Engineering, Sogang University. His research interests include zone namespace SSDs and file systems.

JOO-YOUNG HWANG received the B.S. degree in electrical engineering from Yonsei University, South Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), in 1998 and 2003, respectively. He is currently the Vice President for Research of Samsung Electronics. He is responsible for the research and development of solid-state drives and storage systems. His research interests include file systems, solid-state drive technologies, and high-performance computing.

YOUNGJAE KIM (Member, IEEE) received the B.S. degree in computer science from Sogang University, Seoul, South Korea, in 2001, the M.S. degree in computer science from KAIST, in 2003, and the Ph.D. degree in computer science and engineering from Pennsylvania State University, University Park, PA, USA, in 2009. He is currently a Professor with the Department of Computer Science and Engineering, Sogang University. Before joining Sogang University, he was a Research and Development Staff Scientist with the U.S. Department of Energy's Oak Ridge National Laboratory, from 2009 to 2015, and an Assistant Professor with Ajou University, Suwon-si, South Korea, from 2015 to 2016. His research interests include operating systems, file and storage systems, parallel and distributed systems, computer systems security, and performance evaluation.