*electronics*

MDPI

*Article*

# Exploiting Data Duplication to Reduce Data Migration in Garbage Collection Inside SSD

**Shiqiang Nie [1], Jie Niu [1], Chaoyun Yang [1], Peng Zhang [2], Qiong Yang [2], Dong Wang [2] and Weiguo Wu [1,*]**

[1] School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China
shiqiang.nie@xjtu.edu.cn (S.N.)
[2] Xi'an Aeronautics Computing Technique Research Institute, Xi'an 710065, China
* Correspondence: wgwu@xjtu.edu.cn

**Abstract:** NAND flash memory has been widely adopted as the primary data storage medium in data centers. However, the inherent characteristic of out-of-place updates in NAND flash necessitates garbage collection (GC) operations on NAND flash-based solid-state drives (SSDs), aimed at reclaiming flash blocks occupied by invalid data. GC processes entail additional read and write operations, which can lead to the blocking of user requests, thereby increasing the tail latency. Moreover, frequent execution of GC operations is prone to induce more pages to be written, further reducing the lifetime of SSDs. In light of these challenges, we introduce an innovative GC scheme, termed SplitGC. This scheme leverages the records of data redundancy gathered during periodic read scrub operations within the SSD. By analyzing these features of data duplication, SplitGC enhances the selection strategy for the victim block. Furthermore, it bifurcates the migration of valid data pages into two phases: non-duplicate pages follow standard relocation procedures, whereas the movement of duplicate pages is scheduled during idle periods of the SSD. The experiment results show that our scheme reduces tail latency induced by GC by 8% to 83% at the 99.99th percentile and significantly decreases the amount of valid page migration by 38% to 67% compared with existing schemes.

**Keywords:** solid-state drive; NAND flash; garbage collection

check for updates

Academic Editor: Elias Stathatos

Received: 15 April 2025
Revised: 2 May 2025
Accepted: 3 May 2025
Published: 4 May 2025

**Citation:** Nie, S.; Niu, J.; Yang, C.; Zhang, P.; Yang, Q.; Wang, D.; Wu, W. Exploiting Data Duplication to Reduce Data Migration in Garbage Collection Inside SSD. *Electronics* **2025**, *14*, 1873. https://doi.org/10.3390/electronics14091873

**Copyright:** © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

NAND flash-based solid-state drives (SSDs) have replaced traditional hard disk drives (HDDs) as an indispensable storage solution in data centers, owing to their superior characteristics such as high bandwidth, high performance, high reliability, and low power consumption. However, SSDs exhibit more pronounced performance fluctuations compared with HDDs. While achieving extremely low average read latency, SSDs suffer from significant long-tail latency, where the 99th percentile latency can be up to 100 times higher than the average latency [1]. The primary cause of long-tail latency in SSDs is garbage collection (GC).

Due to the out-of-place update nature of NAND flash, SSDs cannot directly overwrite data during updates as traditional HDDs do. Instead, updated data must be written to another free page, while the original location is marked as invalid. Consequently, SSDs must perform GC to reclaim flash blocks occupied by invalid pages and maintain sufficient free space for subsequent write operations. The GC process involves relocating scattered valid pages from a victim block to a new location, after which the victim block is erased and returned to the pool of available blocks. However, the valid page migration introduces additional read and write operations, which can potentially lead to normal

I/O requests being temporarily suspended. Under sustained write workloads or when the SSD approaches full capacity, frequent GC invocations prolong the waiting delay of subsequent I/O requests, thereby exacerbating long-tail latency. Moreover, since GC is typically triggered by threshold-based mechanisms, its execution leads to intermittent I/O performance fluctuations. In data center computing scenarios, long-tail latency is not merely an occasional phenomenon affecting a negligible fraction of requests [2]. A single query operation typically requires thousands of disk read operations across the data center. Under such circumstances, even one high-latency disk access can significantly increase the overall query response time.

The latency of GC primarily consists of two components: data migration and block erase. To mitigate the impact of erase operations on long-tail latency, researchers have proposed various techniques [2–4]. In comparison, data migration contributes more significantly to GC latency. This operation competes for the parallel computing resources of flash chips, directly contending with user I/O requests. Furthermore, as flash memory architecture evolves from 2D planner to 3D stack, the capacity of storage blocks continues to increase. Consequently, even with the same valid page ratio, the absolute number of valid pages requiring migration may grow substantially, leading to exacerbated long-tail latency [5]. Therefore, with erase optimization becoming increasingly mature, optimizing data migration emerges as the critical breakthrough for further alleviating long-tail latency. Researchers have improved GC algorithm efficiency and mitigated SSD long-tail latency through various approaches, including optimizing data layout [6,7], refining victim block selection schemes [8,9], adapting to flash memory physical constraints [10], and leveraging advanced command acceleration techniques [11]. However, these schemes have largely overlooked the critical factor of intrinsic data duplication characteristics.

Existing GC optimization methods that utilize data duplication predominantly employ data deduplication techniques [12]. Nevertheless, inline deduplication introduces additional computational overhead and hardware costs in the critical I/O path [13], while offline deduplication requires dedicated data scanning and computational resources. Furthermore, data deduplication causes multiple logical addresses to reference the same physical location, concentrating access pressure on specific flash memory units. The resulting frequent program/erase (P/E) cycles may lead to irreversible data loss, ultimately compromising SSD data integrity and reliability.

Based on the above problems, we propose SplitGC, which capitalizes on the time period during read scrub [14] to optimize the GC procedure. Compared with existing approaches that incur time and hardware overhead from data deduplication and face reliability issues caused by shared pages, SplitGC does not perform deduplication but instead optimizes GC by leveraging data duplication characteristics. SplitGC embeds fingerprint computation and management into the SSD's periodic background read scrub process, utilizing existing SSD resources to avoid additional time and hardware overhead in the critical I/O path. It classifies flash blocks into two categories, those containing duplicate pages and those without, applying differentiated victim block selection schemes. During data migration, SplitGC prioritizes migrating non-duplicate valid pages while deferring the processing of duplicate pages, while maintaining multiple physical copies of duplicate data. SplitGC avoids the concentrated wear risk associated with single-copy storage while preserving data accessibility. SplitGC serves to maximize storage resource utilization efficiency, minimize real-time data migrations within the GC process, and thereby alleviate the issue of long-tail latency. Overall, this paper makes the following contributions:

1. We conduct a preliminary experiment that quantitatively analyzes the impact of different operations within GC on long-tail latency, and it identifies page reads and

page writes (data migration) during GC as the predominant contributors to such issues.
2. We propose a novel GC scheme rooted in data duplication characteristics—SplitGC, which mainly includes a read scrub-assisted fingerprint generation scheme and a latency-bounded GC scheme.
3. We conduct a series of experiments and validate the effectiveness of our scheme. Compared with the state of the art, SplitGC reduces tail latency induced by GC by 8% to 83% at the 99.99th percentile and significantly decreases the amount of valid page migration by 38% to 67% compared with existing schemes.

In the rest of this paper, Section 2 discusses the background and motivates our design. Section 3 presents the detailed scheme. Section 4 describes the experiment methodology and analyzes the results. Section 5 gives the related work, and Section 6 concludes the paper.

## 2. Background and Research Motivation

### 2.1. SSD Architecture

SSDs fully leverage their inherent high parallelism by integrating multiple flash chips to enhance storage capacity. Unlike traditional HDDs, SSDs necessitate erase operations, which are one to two orders of magnitude slower than read-and-write operations. Flash memory is characterized by an erase-before-write feature, mandating that existing data in a flash page must be erased before new data can be written. Unlike read-and-write operations that occur at the page level, erase operations typically take place on a block level. When the SSD requires modification of existing data pages, direct overwriting of the original data is not feasible; instead, the entire block containing the page must be erased before the new data can be written. Moreover, since each flash cell has a limited number of erasable cycles, the lifetime of the SSDs is inherently constrained. The physical attributes of the underlying flash in SSDs significantly distinguish them from conventional HDDs. To accommodate these unique properties and emulate a standard block device, SSDs employ the Flash Translation Layer (FTL). The primary function of the FTL is to map logical addresses from the host onto actual physical addresses within the SSD, thereby masking the complex internal management intricacies of the flash memory. In particular, the FTL implements out-of-place updates, performing necessary data migrations and erase operations in the background. This ensures that users perceive a stable and contiguous storage space, effectively mitigating performance challenges and lifetime constraints imposed by the characteristics of flash memory.

### 2.2. Garbage Collection

To ensure efficient read and write performance of flash memory and to minimize unnecessary block erase operations, SSDs conventionally adopt an out-of-place update strategy. In this context, each page within flash memory is categorized into one of three distinct states: valid, invalid, or free. During data updates, instead of directly overwriting the physical location containing old data, the SSD controller identifies a new free page and writes the newly arrived data into the new page, while updating the FTL such that the logical address now points to the new physical location. Consequently, new data are written to the new physical page, and the original page is marked as invalid. As the SSD operates, invalid pages gradually scatter across used flash blocks, accumulating over time. Therefore, to maintain sufficient available space for future write requests, the SSD must take action to reclaim space occupied by invalid pages. This process of recycling storage space taken up by invalid pages is known as GC, which is a critical background process within SSDs.

When the number of available free blocks drops below a certain threshold, the GC process is triggered, typically consisting of three core steps. First, a suitable flash block is chosen as a victim block based on specific GC algorithms and actual usage conditions of the blocks, such as the ratio of valid pages, erase counts, and usage duration. Subsequently, the valid data within the victim block are migrated to a newly allocated target block; during this phase, it may be essential to consider categorizing and sorting the valid data, as well as devising rational rules for allocating the new block. Finally, after the successful migration of valid data, the selected victim block undergoes an erase operation, freeing up its space.

However, while GC is essential for supporting the out-of-place update in SSDs, the overhead introduced by GC itself should not be overlooked. During the execution of GC, the SSD controller becomes engaged in GC tasks, leading to delayed responses to incoming read and write requests from users, which in turn exacerbate access latency. Moreover, internal data migrations prompted by GC have the potential to introduce supplementary write operations, contributing to write amplification issues. Over the lifespan of an SSD, frequent triggering of GC can lead to performance fluctuations. More seriously, if the GC algorithm is poorly designed, it could result in uneven wear on flash blocks, accelerating device aging, and thereby negatively impacting the lifespan of the SSD.

### *2.3. Motivation*

In this paper, we explore how to utilize the characteristics of data duplication to optimize valid page migration during GC. It is well-known that long-tail latency caused by GC can lead to fluctuations in I/O performance. We investigate the primary contributors to long-tail latency during GC by separately setting the erase latency and data migration latency to 0 under the FIU workload [15]. The results, shown in Figure 1, indicate that page migration is the dominant factor contributing to long-tail latency. Moreover, as the number of pages involved in GC increases, so does the long-tail latency. Erase latency only contributes to a small portion of the tail latency, and some studies have been devoted to optimizing erase [2–4], which could further mitigate the impact of erase operations on long-tail latency.

Data duplication has become a prevalent phenomenon due to frequent copy operations, such as writing data as redo logs to journal files or copying files between different directories in storage systems. One viable approach to leverage data duplication for GC optimization is data deduplication, which identifies and eliminates redundant data either during the write phase or during system idle time to save storage space, thereby indirectly mitigating performance bottlenecks caused by frequent GC triggers due to space saturation. Based on the triggering mechanism, data deduplication can be classified into inline deduplication and offline deduplication. Offline deduplication typically operates during system idle periods, scanning and analyzing stored data to improve storage utilization by removing duplicate content. In contrast, inline deduplication processes data in real time during writes. Since redundant write data are identified and eliminated directly in the write path, fingerprint computation and lookup introduce additional computational overhead, increasing memory consumption in the critical I/O path [16].

Although applying data deduplication in SSDs can effectively improve storage space utilization, the resulting reliability issues cannot be overlooked. When an SSD performs deduplication, multiple logical addresses with identical hash values map to the same physical address. Due to the random distribution of duplication data, certain channels or dies may bear excessive hot data while other parallel units remain underutilized. This asymmetric access pattern not only reduces the parallel processing efficiency of SSDs but also creates localized hotspot regions at the physical level. The concentration of access pressure directly impacts the endurance mechanism of flash memory. The physical

characteristics of flash cells dictate a limited number of P/E cycles. Particularly in high-temperature environments, frequently erased/written cells are more prone to charge leakage, eventually leading to irreversible data loss. Moreover, the actual benefits of deduplication are not immediately apparent to users. Existing file systems preallocate fixed logical capacity upon initialization, meaning that physical space reclaimed by device-level deduplication cannot be directly mapped as available storage resources. Therefore, to ensure data integrity and reliability, we do not employ deduplication but instead use data redundancy characteristics as a guide for GC optimization.

Based on this observation, we propose that optimizing data migration during GC can greatly reduce long-tail latency. The core idea is to skip migrating duplicate pages during GC, record the necessary information, and then write the duplicate pages during the idle time of the SSD. This approach can reduce the number of pages involved in GC. However, several challenges remain, such as generating fingerprints for every valid page within the SSD and postponing the migration of duplicate pages without compromising data reliability. In the next subsection, we outline our design to minimize the overhead of fingerprint generation and implement latency-bounded GC.

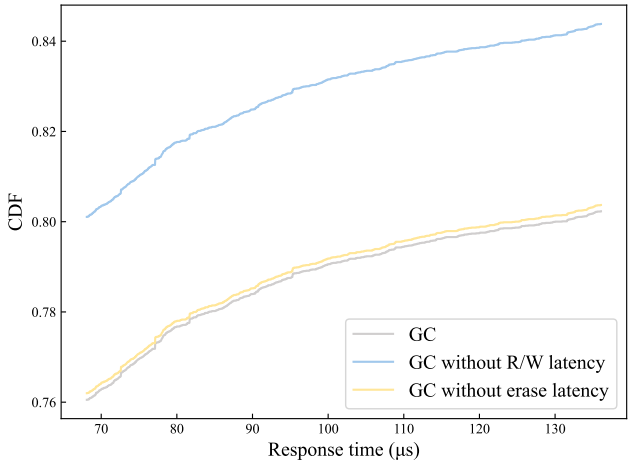



**Figure 1.** The Cumulative Distribution Function (CDF) of read latency in workload home1.

## 3. Design

### *3.1. Design Overview*

The key idea of SplitGC focuses on achieving synchronization between duplication identification and the SSD's periodic tasks, restructuring the timing of valid page migration in conventional GC. Moreover, SplitGC preserves physical copies of duplicate valid pages to avoid the high-risk single-copy storage scenario caused by deduplication. SplitGC embeds the fingerprint generation and management process into the read scrub operation. By leveraging the data reading and verification phase during read scrub, SplitGC computes fingerprints without introducing additional I/O overhead and records the mapping between fingerprint information and physical addresses in a fingerprint table during background operations. SplitGC utilizes the recorded data duplication information as a key criterion for selecting victim blocks during GC. Furthermore, SplitGC differentiates between migrating duplicate and non-duplicate valid pages: during real-time GC, it skips the migration of duplicate pages, only recording essential metadata, and postpones the write operations for duplicate pages to SSD idle periods. SplitGC effectively reduces the volume of real-time data migration during GC.

Figure 2 illustrates the overall system architecture design of SplitGC, which is integrated into the FTL. The design primarily comprises three core modules: the Allocator Module, the Garbage Collection Module, and the Read Scrub-Assisted Fingerprint Gen-

erator Module. In the Read Scrub-Assisted Fingerprint Generator Module, SSD periodically performs read scrub operations to verify data pages and records their duplication information in a fingerprint table. This information is subsequently transmitted to the Allocator Module.

Within the Allocator Module, the block management component categorizes flash blocks into two types based on the duplication information recorded in the fingerprint table: blocks containing duplicate valid pages and blocks without duplicate valid pages. Different victim block selection algorithms are applied depending on whether duplicate-containing blocks exist in the plane. Unlike traditional deduplication approaches that typically employ single-copy storage, SplitGC preserves all physical copies of duplicate pages to prevent data loss caused by concentrated access and wear on shared pages. To accommodate the delayed migration of duplicate valid pages during GC, SplitGC modifies the conventional FTL mapping table, enabling unmigrated valid pages to be read through their copies.

The victim block selection process operates as follows: based on duplication information provided by the allocation module, SplitGC first determines whether the group triggering GC contains blocks with duplicate valid pages. If such blocks exist, it selects the block with the fewest non-duplicate valid pages as the victim; otherwise, it employs the greedy scheme to choose the block with the fewest valid pages overall.

For data migration, SplitGC prioritizes migration of non-duplicate valid pages and places duplicate valid pages into a dynamic priority queue for temporary storage. SplitGC schedules the FTL to complete its migration during SSD idle periods. The blue-highlighted components in Figure 2 represent targeted enhancements introduced by SplitGC over traditional FTL designs, emphasizing the novel management of data page redundancy information, the schemes employed for victim block selection during GC, modifications to the mapping table, and efficient migration of valid pages.

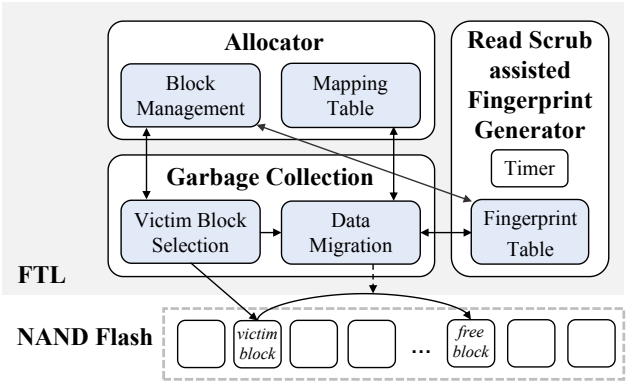



**Figure 2.** Overview of SplitGC.

### *3.2. Read Scrub-Assisted Fingerprint Generation Scheme*

In the SSD routine, the primary objective is to promptly respond to I/O requests from the host. However, putting data deduplication during the processing of I/O operations would introduce additional computational latency. Such operations could significantly prolong the response time for individual I/O requests, thereby directly impacting the overall performance of the storage system. This effect is particularly pronounced in scenarios characterized by these applications with stringent latency requirements. Modern SSDs typically refresh stored data periodically for reliability management [17,18]. To address this challenge, we focus on the internal read scrub [14,19] mechanism of SSD, which is a maintenance technique specifically designed to be triggered during idle or low-load periods for the purpose of detecting and rectifying potential data errors. We propose SplitGC to embed the analysis of data duplication within the read scrub process, thereby

harnessing the idle intervals of the SSD to provide a more conducive environment for computations involving fingerprint calculation and comparison of duplicate data. SplitGC aims to prevent these operations from competing with the user I/O for limited resources. Consequently, the SSD can efficiently serve I/O requests while concurrently conducting background analysis of data duplication, thus striking a balance between resource utilization and performance preservation. Furthermore, as data migration can be incrementally carried out during subsequent periods of system idleness, the performance impact on SSDs is further mitigated.

The read scrub process itself necessitates the reading of each page within the SSD, accompanied by the execution of Error Correction Code (ECC) verification and data integrity checks. This procedure ensures that the system is capable of making decisions based on the most up-to-date, validated state of the data, thereby enabling the acquisition of precise information regarding duplicate data and avoiding misjudgments resulting from data errors or inconsistencies. During this process, the SSD possesses the capability to access all data pages and has provisioned a dedicated time window for data operation, thereby ensuring comprehensive coverage of duplicate data detection. By encapsulating the analysis of data duplication within the established read scrub workflow, we harness the extant hardware access mechanisms and allocated time resources without necessitating the introduction of specialized access paths or scheduling schemes. SplitGC not only simplifies the design but also diminishes the intricacy of implementation.

During the read scrub process, SSDs must traverse all data blocks in a single pass to perform data integrity verification. SplitGC adopts an incremental fingerprint computation scheme, partitioning the physical blocks of the SSD into multiple logical groups, each containing a fixed number of blocks. During read scrub, for the currently active group, in addition to conventional data reading and ECC verification, fingerprints are computed, and the fingerprint table is updated. For other inactive groups, only conventional data verification is performed without fingerprint computation or management. Through this incremental fingerprint computation scheme, SplitGC decomposes the task of generating fingerprints for all data blocks into multiple subtasks, gradually completing fingerprint computation for the entire disk to avoid transient load peaks caused by centralized computation. We employ SHA-1 [16] as the Fingerprint Algorithm in SplitGC. The metadata area of the SSD maintains a processing table that records the read scrub status of each group, using only 1 bit per group to indicate its processing state, thereby reducing metadata management overhead. Each read scrub selects the next group to be processed based on this progress table.

When the periodic read scrub task is initiated, the SSD synchronously computes the fingerprint for each data page in the current group during reading and subsequently performs a comparison. To mitigate the overhead of byte-by-byte fingerprint comparison, SplitGC first employs a Bloom filter to filter out non-existent fingerprints. The Bloom filter, a space-efficient probabilistic data structure, consists of an $m$-bit array and $k$ independent hash functions $h_1$ to $h_k$, whose output ranges cover the entire array space. When processing an input dataset containing $n$ new elements, each element is mapped to multiple array positions via all hash functions, and these positions are set to 1. During element queries, if all mapped positions for an element $n$ are set to 1, it can be inferred with high probability that the element belongs to the set; conversely, if any mapped position is 0, the element is definitively not in the set. In SplitGC, the false positive rate of Bloom filters is primarily determined by the bit array size ($m$), number of hash functions ($k$), and inserted elements ($n$). For the read scrub process in SplitGC, we configure $m = 4.2n$ and employ $k = 3$ hash functions (BKDRHash, APHash, DJBHash).

Leveraging the properties of the Bloom filter, when querying whether a data page's fingerprint exists in the fingerprint table, SplitGC first uses the Bloom filter to quickly exclude definitively non-existent fingerprints, accessing the fingerprint table only for potentially existing fingerprints, thereby reducing the number of fingerprint table accesses. If the Bloom filter query returns a non-existent result, the page is directly marked as unique data, its fingerprint is added, and fingerprint table access is skipped. Since data verification takes precedence over fingerprint comparison, corrupted data will never be misclassified as unique. If the query returns a potentially existing result, the process proceeds to exact fingerprint table comparison. For unique data, its fingerprint is inserted into the fingerprint table, and the Bloom filter is updated; for duplicate data, its fingerprint information is updated, and the containing block is marked as one with duplicate pages. This marking is critical for the selection scheme of the victim block during subsequent GC.

Figure 3 depicts the operational workflow of our proposed latency-bounded GC scheme. When the available space within a specific plane of the SSD dips below a predefined threshold, the GC process is triggered. To begin, we capitalize on the data duplication information amassed during the read scrub operation to investigate the existence of blocks within the target plane that contain duplicate pages. In the absence of such blocks, SplitGC adheres to the conventional scheme by selecting the block with the minimum number of valid pages as the victim block for migration, concurrently updating both the fingerprint table and mapping table. Conversely, upon encountering a block containing duplicate pages, SplitGC opts for the one with the least non-duplicate pages as the victim block. This initiative aims to minimize global data redundancy. Furthermore, SplitGC differentiates between duplicate and non-duplicate pages in its migration scheme. In the case of duplicate pages, immediate migration is not executed; instead, their information, such as the original data content and the target physical address, is recorded in a dedicated waiting queue. This queue is subsequently scheduled by the FTL during idle periods of the SSD, enabling asynchronous and ordered writes of the duplicate data into free pages within other blocks, thereby circumventing the latency overhead incurred by real-time migration. Conversely, valid non-duplicate pages adhere to the standard procedure, undergoing immediate migration.

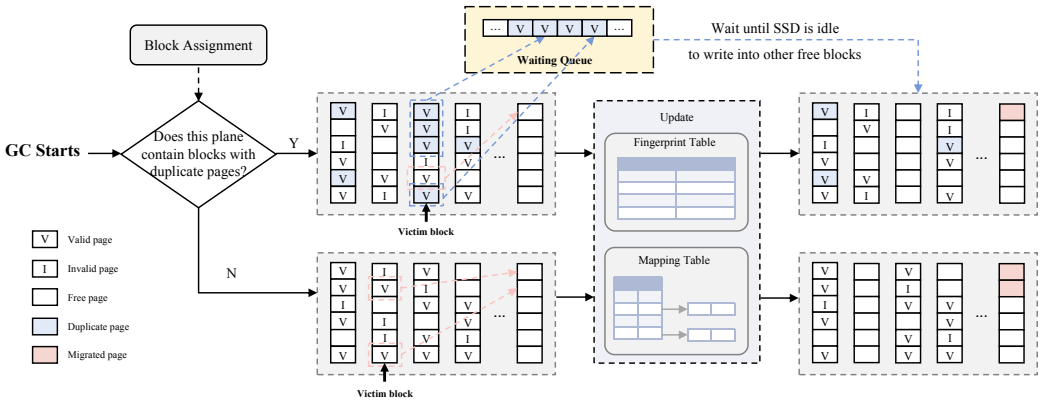



**Figure 3.** The workflow of the latency-bounded GC scheme.

### *3.3. Latency-Bounded GC Scheme*

To ensure that read requests remain unaffected during data migration, SplitGC employs a two-level mapping table, transforming the original one-to-one mapping in SSDs into a one-to-many mapping. The primary mapping table contains the mapping from Logical Page Numbers (LPNs) to Physical Page Numbers (PPNs) or Virtual Page Numbers (VPNs), where the most significant bit (MSB) of the PPN/VPN serves as a flag. If the MSB is 0, it indicates that the data are unique, and the LPN directly maps to a PPN; if the

MSB is 1, it signifies that the data are duplicated, corresponding to a VPN, and requires a lookup in the secondary mapping table (SMT) to retrieve the list of PPNs associated with redundant data. The secondary mapping table records the mapping from VPNs to multiple PPNs storing identical content. When data are written to the SSD, the FTL follows the conventional process of allocating free pages and updating the primary mapping table.

As illustrated in Figure 4, when processing a read request, SplitGC first queries the primary mapping table using the LPN to obtain the corresponding PPN/VPN. If the logical page in the primary mapping table is marked as invalid (due to GC erasing the associated block, where the original duplicate valid pages have entered a waiting queue), SplitGC consults the secondary mapping table to retrieve the list of PPNs linked to that VPN and selects the first valid PPN to read the data. If a PPN becomes invalid due to data updates or block erasure, the corresponding PPN list in the secondary mapping table is updated accordingly. Thus, in SplitGC, duplicate valid pages migrated to the waiting queue maintain multiple copies of their content addresses in the mapping tables. Even if their original physical addresses are marked as invalid, they can still be accessed by querying the secondary mapping table to locate alternative physical copies, ensuring data retrieval. Regardless of the data migration phase, the FTL guarantees accurate data access for read requests by maintaining the state of the mapping tables, thereby preserving system data consistency. Additionally, SplitGC avoids the data reliability issues associated with shared pages in traditional deduplication schemes.

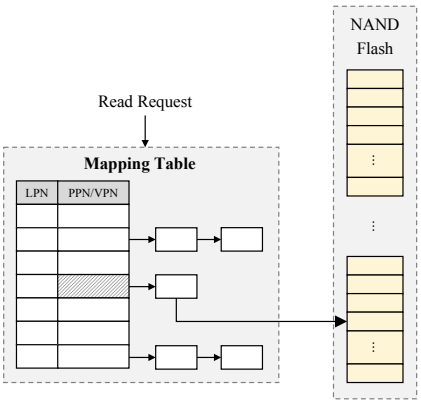



**Figure 4.** SplitGC reads valid pages from the waiting queue.

SplitGC ensures data reliability through the following design features: (1) The SMT in FTL employs an atomic update strategy during GC, guaranteeing that the mapping table remains consistent even in crash scenarios. Physical address information of duplicate pages scheduled for deferred migration is recorded in SMT before being committed to the pending queue. LPNs are mapped to multiple physical copies through VPN mapping, eliminating single-point dependency. (2) Unlike traditional deduplication, SplitGC permanently retains all physical copies. Even if deferred migration is incomplete, duplicate page copies in original blocks remain accessible through the secondary mapping table, ensuring data recoverability after power failures. (3) Modern SSD controllers typically incorporate supercapacitors or battery-backed RAM to flush critical metadata (e.g., mapping tables, pending queue states) to NAND flash during unexpected power loss. SplitGC leverages this mechanism to achieve metadata persistence. In summary, deferred migration cannot cause logical data loss because the dependent physical copies already exist in other blocks, and all metadata updates maintain crash consistency.

Figure 5 shows an example of the comparison between the conventional GC scheme and the SplitGC scheme. Conventional GC straightforwardly opts for the block with the fewest valid pages as the victim block. In contrast, SplitGC, upon encountering a

target plane containing blocks with duplicate valid pages, prioritizes the block housing the minimum number of non-duplicates. This modification fundamentally shifts the focal point of data migration during GC. In practice, SplitGC necessitates the migration of only a relatively small subset of non-duplicate valid pages within the victim block, thereby dramatically curtailing the immediate volume of data to be transferred and correspondingly diminishing the I/O burden and energy consumption during GC. While duplicate pages may not be instantaneously relocated during the GC process, their safety and integrity remain assured. Because duplicates already exist in other blocks within the SSD, ensuring access to these data even if they have not yet undergone physical relocation. Thus, normal reading and usage of user data remain unaffected. In practical storage contexts, data duplication is a highly prevalent occurrence [13,16,20]. To illustrate the implications of this phenomenon, we devised a representative example of GC. As depicted in Figure 5, conventional GC entails the migration of 3 valid pages. By comparison, SplitGC requires the migration of just 1 non-duplicate valid page from the block with the most duplicate pages. Evidently, SplitGC significantly reduces the real-time page migration load relative to conventional GC.

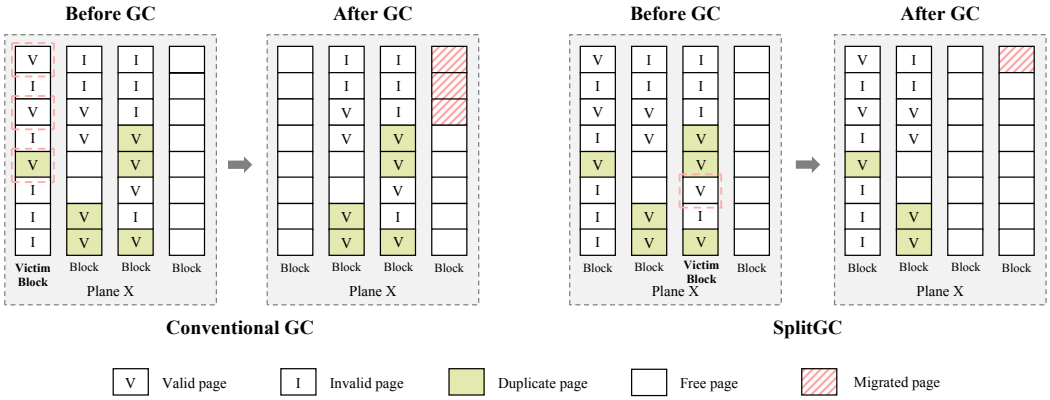



**Figure 5.** An example comparing the number of migrated pages between conventional GC and SplitGC.

### *3.4. Overhead Analysis*

SplitGC leverages the existing periodic background maintenance task of read scrub in SSDs by integrating fingerprint generation and comparison into the data page reading and verification process. This approach eliminates the computational overhead in the critical I/O path introduced by online deduplication and the hardware overhead associated with offline deduplication. The latency of read scrub in SplitGC is calculated as shown in Equation (1).

$$Scrub_{latency} = T_{read} + ECC_{check} + (FP_{generator} + FP_{manage}) * \alpha + T_{recovery} \times Error_{rate} \quad (1)$$

Here, $T_{read}$ represents the time required to read data from flash memory, which depends on the characteristics of the flash chips; $ECC_{check}$ denotes the time for data integrity verification, determined by the complexity of the ECC; $FP_{generator}$ is the fingerprint generation time; and $FP_{manage}$ indicates the fingerprint management time, including operations such as fingerprint lookup and reference count updates. The parameter $\alpha$ represents the triggering probability of fingerprint management. Since SplitGC adopts an incremental fingerprint generation scheme, which generates fingerprints only for data blocks in the currently active group during each process, $\alpha$ is less than 1. $\alpha$ represents the proportion of physical block groups that need to process fingerprints during each read scrub operation (i.e., the granularity of incremental group processing). In SplitGC, we divide the SSD physical blocks into 10 logical groups ($\alpha$ = 0.1), and each read scrub processes

only one group, gradually completing full-disk fingerprint generation. $\alpha$ value represents an empirical trade-off that could be dynamically adjusted based on SSD idle cycles in practical deployments. However, its sensitivity is low and would not significantly alter SplitGC's benefits.

When $\alpha$ = 0.1, fingerprint-related operations ($FP_{generator}$ + $FP_{manage}$) account for a relatively small proportion of the total read scrub latency, with the primary time overhead coming from the read and ECC check processes ($FP_{manage}$ + $ECC_{check}$). In our work, the quantified parameters for each latency component are as follows [16,21]: $T_{read}$: 90 µs; $ECC_{check}$: 20 µs; $FP_{generator}$: 80 µs; $FP_{manage}$ (Bloom filter and table update): 10 µs. In our work, the fingerprint computation overhead is (80 + 10) × 0.1 = 9 µs, accounting for only 7.5% of the total read scrub latency (90 + 20 + 9 = 119 µs), confirming that $\alpha$ has a limited impact on overall overhead. Additionally, although SHA-1 is not the optimal Algorithm, SplitGC further optimizes performance through Bloom filter prescreening [16].

This design aims to distribute the overhead of fingerprint computation and management, avoiding resource contention caused by centralized processing. Read scrub operates as a low-priority background task, triggered only during SSD idle periods, and can be interrupted by high-priority user I/O requests. Even if $\alpha$ = 1 (full-disk processing), its execution is still constrained by the background scheduling mechanism and will not block foreground I/O. $T_{recovery}$ refers to the time required for data error correction or migration, while $Error_{rate}$ represents the probability of data errors, which is typically low.

By employing the SHA-1 Fingerprint Algorithm and an incremental fingerprint computation scheme, SplitGC introduces minimal additional computational overhead compared with conventional read scrub processes. Moreover, since read scrub operates as a background task during SSD idle periods, user sensitivity to latency is low, ensuring that periodic background operations do not impact foreground I/O performance.

Compared with the baseline SSD, SplitGC introduces two additional components: the fingerprint table and the waiting queue. Generally, the size of the fingerprint table is significantly smaller than the actual data storage capacity, though it still occupies a certain amount of internal SSD storage space. In SplitGC, the fingerprint table is incrementally updated on a per-logical-group basis, keeping memory usage controllable. The waiting queue records duplicate valid pages that require subsequent migration during GC, along with their target locations. Its space overhead is minimal, as it only needs to store basic metadata of the pages awaiting migration.

## 4. Experiment

We implemented and evaluated our proposed optimization scheme using SSDsim [22]. The SSDsim configuration is as follows: it consists of 4 channels, each with 2 flash chips, 2 dies per chip, 2 planes per die, 1024 blocks per plane, and 512 flash pages per block. The read, write, and erase latencies are 90 µs, 900 µs, and 3.5 ms, respectively. Furthermore, to more accurately simulate real-world SSD operating conditions, we preprocess the emulated SSD device by filling 90% of its storage capacity. This preconditioning facilitates more frequent triggering of GC, thereby better reflecting practical usage scenarios. According to empirical data from enterprise SSDs [23], the daily write volume of enterprise SSDs often reaches tens of times their rated capacity. For instance, a 480 GB SSD may endure multiple terabytes of daily writes, causing storage space to rapidly fill to high levels (approaching 90%) and frequently triggering GC operations. In such scenarios, GC-induced performance degradation becomes pronounced, making this setting a reasonable benchmark for evaluating SplitGC's effectiveness.

Due to the risk of dataset privacy breaches, both users and researchers face limitations in sharing proprietary data, resulting in a scarcity of workload traces available for study-

ing data deduplication characteristics in storage systems. To address this challenge, we employ the FIU traces [15], a well-established benchmark for studying data duplication properties in SSDs [12,24], to verify and assess SplitGC. FIU traces consist of block I/O data collected from operational systems over several consecutive days, encompassing three distinct user/developer personal directories and three web servers. Table 1 provides the characteristics of these workloads.

**Table 1.** Workload characterization.

| Trace | Write Ratio | Duplication Rate | Average Request Size (KB) |
|---|---|---|---|
| home1 | 98.6% | 45.3% | 8.25 |
| home2 | 86.9% | 33.6% | 10.93 |
| home3 | 99.0% | 36.0% | 8.26 |
| webmail | 69.6% | 70.1% | 8.00 |
| webresearch | 99.9% | 75.2% | 8.00 |
| webusers | 89.3% | 63.5% | 8.76 |

We compared our scheme against three schemes, including a greedy and two GC schemes, both leveraging data deduplication characteristics: Baseline (i.e., selecting the block with the largest number of invalid pages as the victim block); DAGC [25], which postpones the deletion of invalid data in pages and reuses it if subsequent requests with the same content arrive, thus reducing the flash write operations and extending the flash lifespan; and CAGC [12], which checks the fingerprint of the victim block: if the page is not redundant, it writes it to the hot region and updates the index; otherwise, it updates metadata without writing the page.

*Experimental Results Analysis*

**The comparison of average read latency.** Figure 6 presents the normalized average read latency of four GC algorithms under different workloads, with the Greedy-based Baseline as the reference. Across all workloads, SplitGC consistently demonstrates the optimal read latency performance, achieving an average 48% reduction compared with the Baseline. By preferentially selecting blocks containing the fewest non-duplicate valid pages for reclamation, SplitGC preserves flash memory blocks with stronger data locality. These blocks typically exhibit higher spatial proximity, enabling the SSD controller to rapidly locate adjacent physical regions during data access, thereby effectively reducing addressing and data transfer time. CAGC maps multiple logical addresses to the same physical page, creating hotspot regions during concurrent reads that intensify access contention in parallel architectures and prolong read waiting time. The capability of SplitGC to service requests through alternative physical copies significantly reduces the probability of read conflicts.

**The comparison of average write latency.** As illustrated in Figure 7, under a variety of workloads, SplitGC achieves lower write latency compared with others. SplitGC eliminates additional computational overhead by relocating fingerprint computation and management from both the critical I/O path and GC operations to periodic data inspection tasks. While CAGC removes fingerprint computation overhead from the critical I/O path, it still incurs computational and write overhead during valid page migration due to real-time fingerprint calculation and single-copy write processing. In contrast, SplitGC schedules partial data migration operations during SSD idle periods, thereby preventing excessive occupation of front-end resources during write-intensive scenarios and reducing I/O blocking caused by non-interruptible full valid page migration operations.

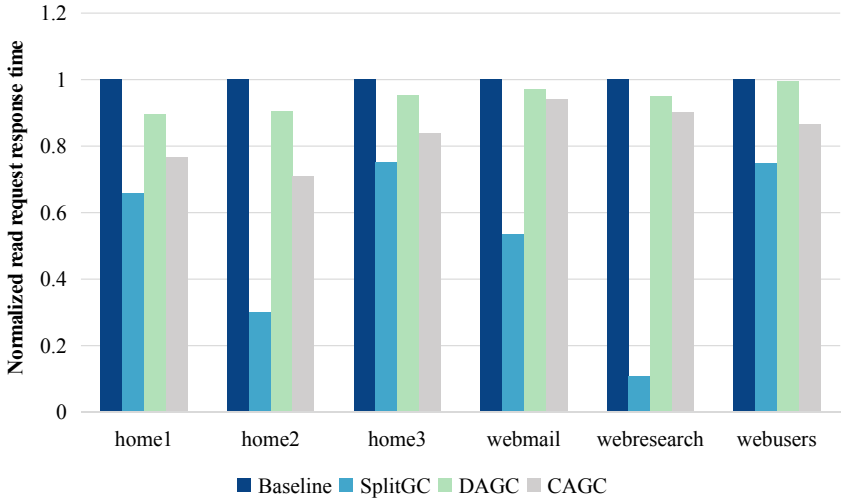



**Figure 6.** Read latency performance comparison.

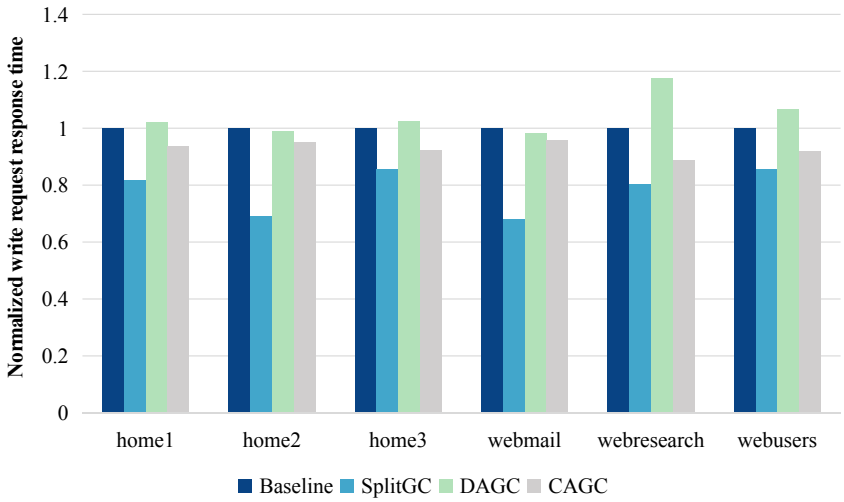



**Figure 7.** Write latency performance comparison.

**The comparison of long-tail latency.** To investigate the impact of various GC schemes on long-tail latency, we analyze the distribution of I/O response times and present the CDF plots for Baseline, SplitGC, DAGC, and CAGC under different workloads. As shown in Figures 8–13, the *x*-axis represents request response time, while the *y*-axis indicates the cumulative distribution of responses up to the corresponding *x*-value. Experimental results demonstrate that SplitGC achieves the lowest long-tail latency (high-percentile response times) across all workloads. Specifically, under the webmail workload, SplitGC reduces the 99.99th percentile tail latency by 50%, 48%, and 46% compared with Baseline, DAGC, and CAGC, respectively. The latency-bounded GC scheme employed by SplitGC effectively removes duplicate valid page migration from real-time GC operations, significantly reducing the volume of valid pages migrated during each GC event. SplitGC yields shorter and more consistent I/O blocking windows during GC, thereby preventing the extreme latency spikes caused by sudden I/O path congestion when migrating large volumes of data in traditional approaches. Furthermore, by backgroundizing fingerprint computation and duplicate identification, SplitGC avoids the long-tail latency amplification that occurs when fingerprint calculations overlap with I/O operations during high-concurrency write scenarios. In contrast, DAGC's reliance on checksum caching mechanisms may introduce metadata synchronization delays due to cache capacity limitations or suboptimal replacement policies. Similarly, CAGC's deduplication-induced single-copy storage tends to create request queue buildup during frequent access to hot data. Both approaches exacerbate tail

latency occurrences. SplitGC produces a more concentrated distribution of I/O response times, resulting in significantly suppressed long-tail latency.

The P99 (99th percentile) latency of each scheme is shown in Figure 14, which directly reflects the performance fluctuations of SSDs in the worst 1% of scenarios. Higher P99 latency indicates greater interference from GC on I/O requests under extreme conditions. Experimental results demonstrate that compared with Baseline, DAGC, and CAGC, SplitGC reduces average P99 latency by 22%, 27%, and 15%, respectively. Traditional GC approaches, which accumulate large amounts of invalid data before performing centralized reclamation, are forced to execute time-consuming block erasures and data migrations during peak user write periods. This directly blocks user I/O responses. In contrast, SplitGC avoids sudden resource contention caused by GC by gradually migrating duplicate valid pages during SSD idle windows, thereby evenly distributing the GC workload over time and preventing concentrated latency spikes. By flattening the timeline of resource scheduling, SplitGC transforms GC's interference with I/O from unpredictable bursts into predictable and smooth operations. This approach effectively compresses the distribution range of long-tail latency.

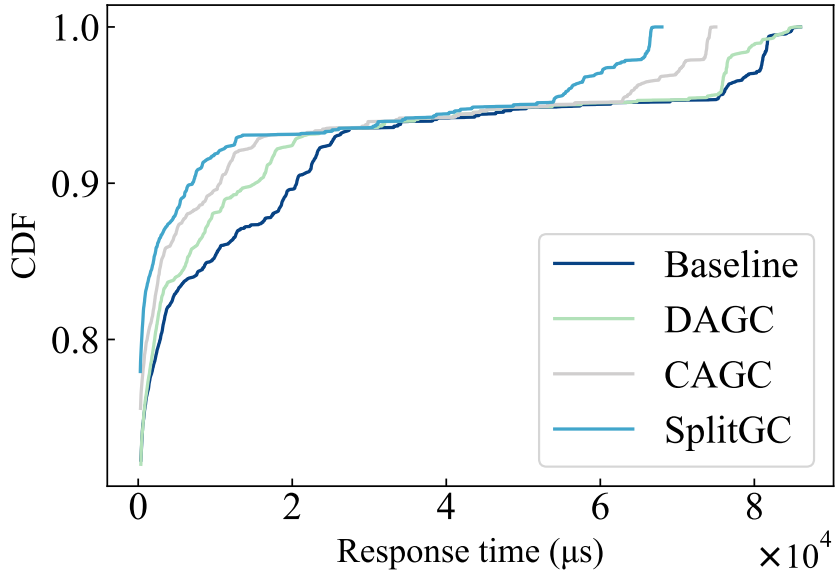



**Figure 8.** The CDF of response time in home1.

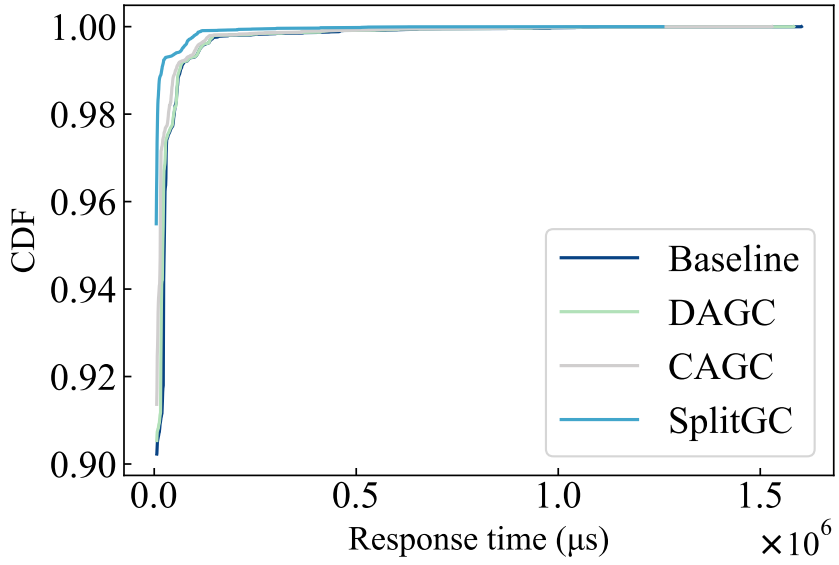



**Figure 9.** The CDF of response time in home2.

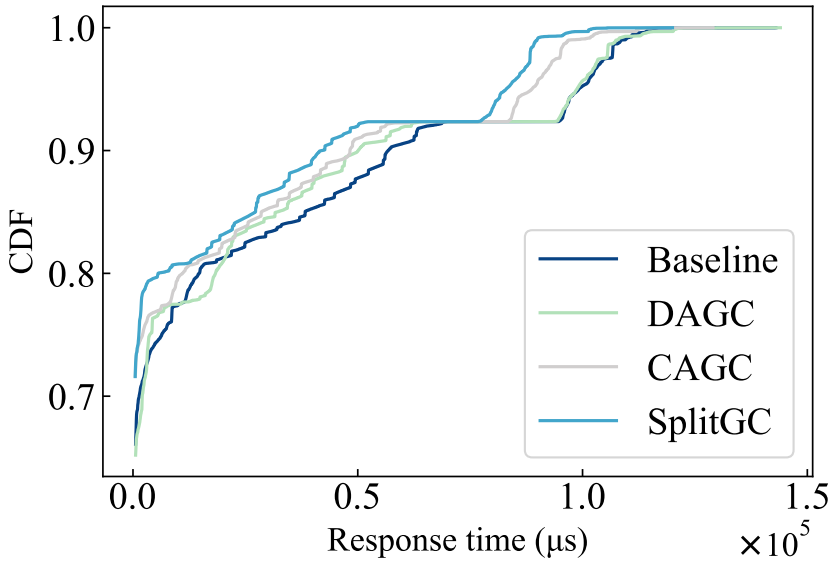



**Figure 10.** The CDF of response time in home3.

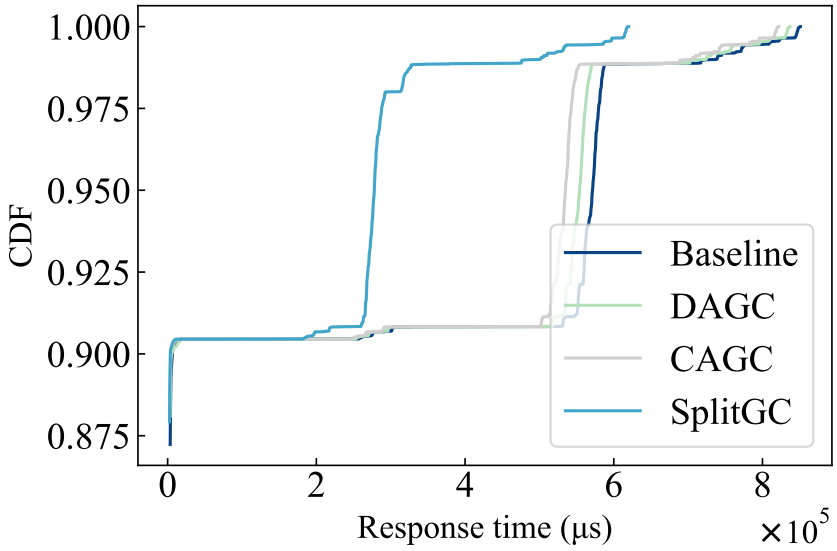



**Figure 11.** The CDF of response time in web mail.

**The average number of migrated pages in GC.** To study the effectiveness of our scheme on the data migration, we conduct the experiment on these schemes to count the number of migrated page during GC. As shown in Figure 15, we can observe that the number of valid page migration is reduced by 38% to 67% compared with other schemes. Specifically, SplitGC reduces about 50% of page migration reduction on average. These gains come from our optimization on GC, as the duplicate pages in victim block are migrated to other position. Conventional GC schemes often result in a higher number of redundant migrations because they do not adequately account for page duplication when selecting pages for migration. Our scheme, by identifying and handling duplicate pages during the GC process, effectively reduces the data migration. This not only reduces the long-tail latency issue but also lowers the overall migration cost of the SSD. Because invalid data with cached checksums could possibly be reused for deduplication, DAGC weighs these invalid data the same way as valid data [25]. DAGC selects the block with the least number of invalid pages with cached checksum plus valid pages as the victim block and migrates the valid pages in it. As a result, DAGC migrates more valid pages than the baseline.

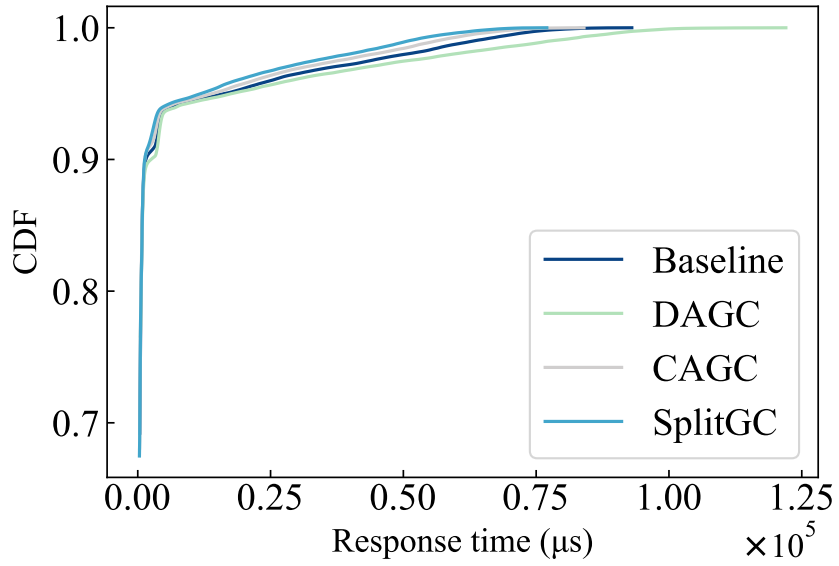



**Figure 12.** The CDF of response time in web research.

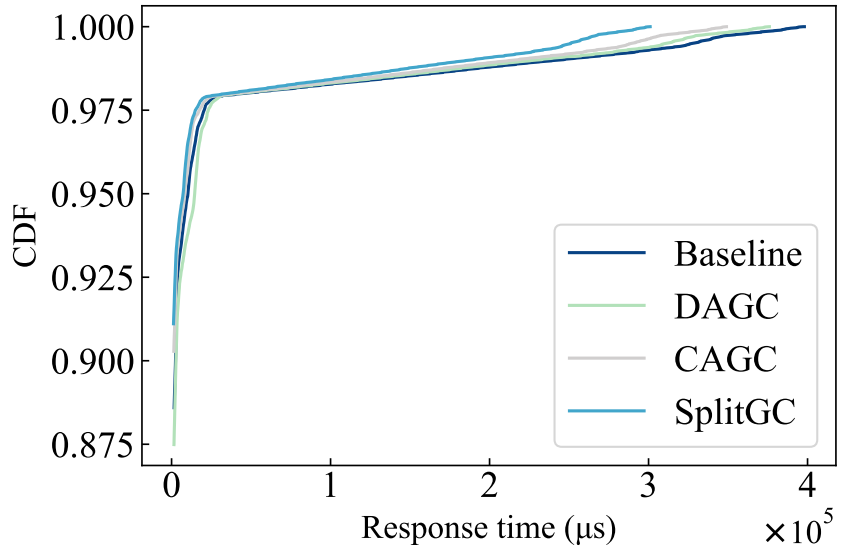



**Figure 13.** The CDF of response time in web users.

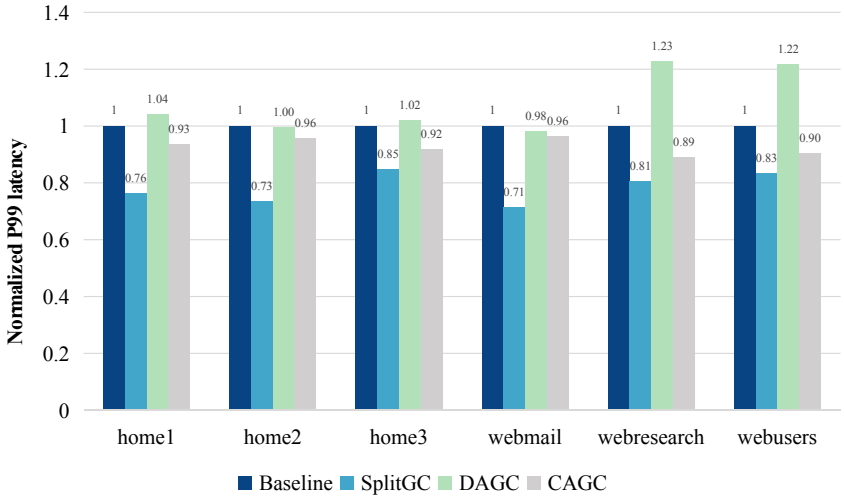



**Figure 14.** The normalized P99 latency.

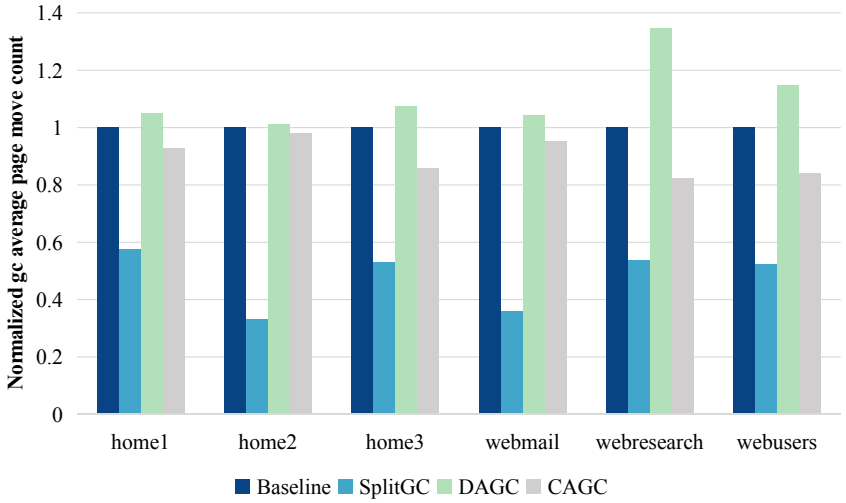



**Figure 15.** The number of valid pages migrated by GC.

**The read latency distribution across a specified time interval.** Figures 16 and 17 present the response time distribution over a specified time interval. Typically, request completion time is influenced by multiple factors, including queue length, operation type, and GC triggering status. During GC operations, shorter request latency indicates lower system blocking and reflects reduced impact of GC on I/O performance. As shown in Figure 16, under infrequent GC conditions, the request completion time exhibits a distinct two-phase characteristic. In the first phase, without GC triggering, all schemes demonstrate comparable completion times. In the second phase, with GC triggering, different valid page migration schemes significantly affect request latency. Figure 17 reveals that during periods with intensive GC operations, SplitGC consistently maintains superior average request latency compared with the other three schemes. Experimental results demonstrate SplitGC's optimal latency control during GC-triggered phases, confirming its effectiveness in reducing I/O latency during GC.

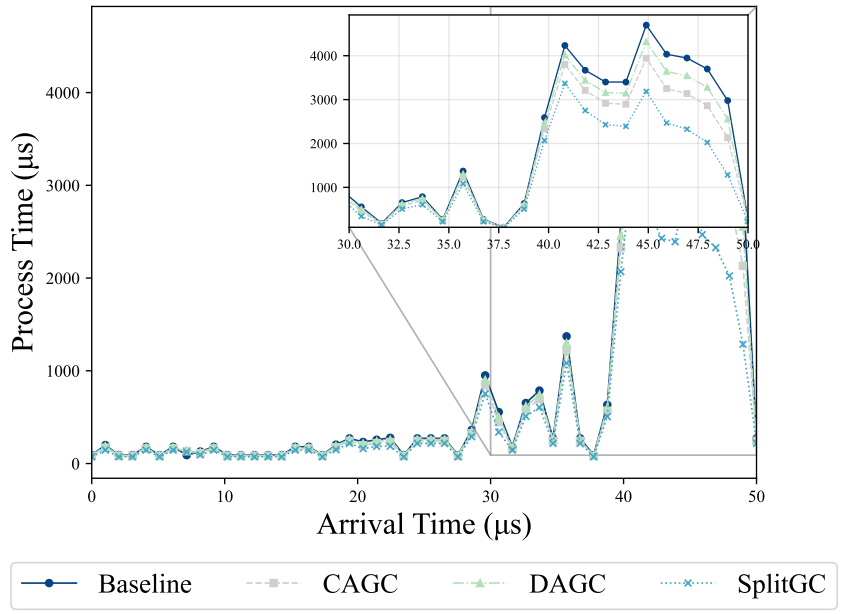



**Figure 16.** The read latency distribution across a specified time interval for workload home2.

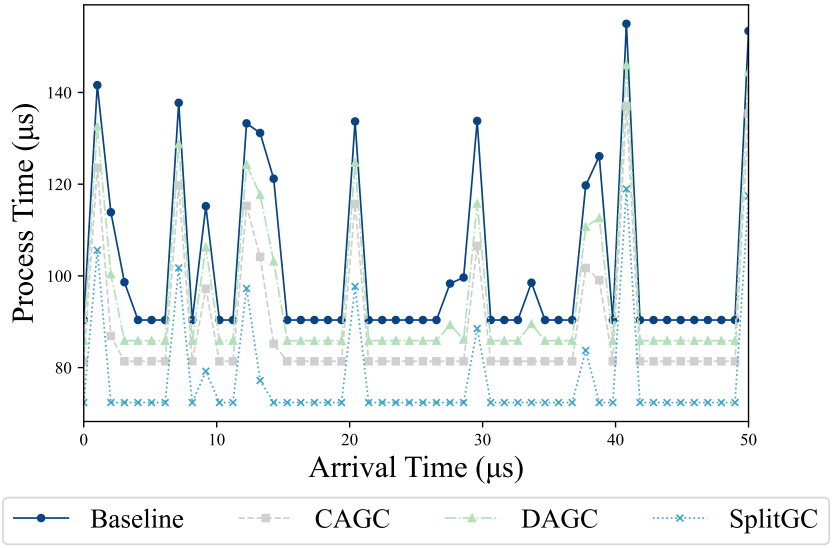



**Figure 17.** The read latency distribution across a specified time interval for workload web users.

**Limitations and applicability analysis.** SplitGC can still reduce migration volume by preferentially reclaiming blocks with higher proportions of non-duplicate pages, even under low duplication rates. For instance, as shown in Figure 15, in the home2 workload (33.6% duplication rate), SplitGC reduces migrated pages by 67%. Even when the duplication rate approaches 0%, its incremental fingerprint management introduces only about 9 µs/page of additional overhead, while traditional GC must perform full migration due to its inability to distinguish duplicates, thus SplitGC maintains an advantage. Note that duplication rates approaching 0% are statistically improbable. Experiments show total fingerprint management latency constitutes only 7.5% of read scrub time, with Bloom filters reducing fingerprint table accesses by 90%. When duplication rates exceed 5% [21], migration reduction benefits significantly outweigh fingerprint costs. Theoretically, if duplication rates approach 0%, SplitGC degenerates to traditional GC, but its design inherently avoids negative impacts. Both fingerprint computation and migration decisions are background processes that never block foreground I/O. No failure threshold was observed in experiments because benefits primarily derive from deferred migration of duplicate pages rather than absolute duplication rates.

## 5. Related Work

### *5.1. Optimizing the GC Algorithm*

In recent years, researchers have devoted attention to minimizing unnecessary data migration during GC by enhancing data page identification methodologies and optimizing data placement strategies. CAGC employs reference count to categorize data pages as hot or cold and subsequently segregates them accordingly [12]. Yang et al. leverage Long Short-Term Memory Networks to predict data activity trends and, leveraging K-Means clustering, allocate data with similar forecasted activity levels into the same flash block [6]. Lifespan-based GC classifies data upon writing to SSDs based on their estimated lifetimes, grouping data with comparable longevity into shared flash blocks [7]. In addition, some studies have focused on dynamically adapting GC algorithms according to SSD system conditions, such as I/O workloads and wear condition [9], intelligently triggering GC requests with varying priorities [26], and enhancing the selection of victim blocks [8]. Overall, these advancements aim to enhance SSD performance and endurance by refining data management and GC processes.

### *5.2. Scheduling GC*

**Priority Adjustment.** To address the potential impact of GC on I/O performance in SSDs, some studies grant higher priority to I/O requests obstructed by GC, ensuring timely processing of critical I/O operations even during GC, thus maintaining Quality of Service (QoS). PGC enables interrupting the GC process when handling I/O requests in the queue, allowing the system to respond more flexibly to I/O demands [27]. Jung et al. propose HIOS, a scheduler both GC-aware and QoS-aware, capable of distinguishing between critical and non-critical I/O requests, prioritizing critical requests by redistributing GC overhead to non-critical requests, thereby ensuring stable QoS during GC execution [28].

**Optimizing the Timing of GC.** Selecting an appropriate time window to conduct GC during system idle periods can reduce competition between GC and regular I/O requests. Kang et al. employ reinforcement learning to predict opportune idle intervals for executing partial GC operations [5], while Sha et al. utilize Fourier transforms to identify periods of sparse I/O requests, informing a model for guiding GC scheduling [29].

**Enhancing Parallelism in SSDs.** Choi et al. proposed I/O-parallelized GC, which exploits idle planes within flash chips during GC operations to concurrently process blocked I/O requests [30]. PaGC mitigates the impact of individual GC operations on overall flash chip performance by conducting GC across multiple planes concurrently [31]. Gao et al. introduce a parallel optimization framework from plane to chip, ensuring that write operations and GC-induced movements of valid pages can be processed in parallel, offering a fresh perspective for boosting system performance [32].

Compared with these existing works, our scheme proposes calculating the fingerprint during the read scrub period, achieving negligible overhead; further, we reduce the number of valid pages to be migrated.

## 6. Conclusions

This paper proposes SplitGC to optimize the tail latency caused by page migration during GC in SSDs. The key idea is to leverage data duplication characteristics to delay the migration of duplicate pages, thereby reducing interference with other normal I/O requests. To achieve this, we designed the read scrub-assisted fingerprint generation scheme and the latency-bounded GC scheme. All experiments were conducted using SSDsim, and the evaluation results show that SplitGC reduces tail latency induced by GC by 8% to 83% at the 99.99th percentile and significantly decreases the amount of valid page migration by 38% to 67% compared with existing schemes. These results validate the effectiveness of our scheme. In future work, we plan to adapt our scheme to incorporate the copy-back command and further optimize valid page migration.

**Author Contributions:** Conceptualization, S.N. and J.N.; methodology, J.N.; software, C.Y.; validation, C.Y., J.N. and P.Z.; data curation, P.Z.; writing—original draft preparation, J.N.; writing—review and editing, S.N. and D.W.; visualization, Q.Y.; supervision, W.W.; project administration, W.W.; funding acquisition, W.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Natural Science Foundation of China under Grant No. 62202368, and in part by Aeronautical Science Foundation of China under Grant No. 20230019070025.

**Institutional Review Board Statement:** Not applicable.

**Data Availability Statement:** Data are contained within the article.

**Acknowledgments:** These authors would like to thank the reviewers for their thoughtful comments and efforts toward improving our manuscript.

**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. Dean, J.; Barroso, L.A. The tail at scale. *Commun. ACM* **2013**, *56*, 74–80. [CrossRef]
2. Kim, S.; Bae, J.; Jang, H.; Jin, W.; Gong, J.; Lee, S.; Ham, T.J.; Lee, J.W. Practical erase suspension for modern low-latency SSDs. In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, 10–12 July 2019; pp. 813–820.
3. Hong, D.; Kim, M.; Cho, G.; Lee, D.; Kim, J. GuardedErase: Extending SSD lifetimes by protecting weak wordlines. In Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST 22), Santa Clara, CA, USA, 22–24 February 2022; pp. 133–146.
4. Cho, S.; Kim, B.; Cho, H.; Seo, G.; Mutlu, O.; Kim, M.; Park, J. AERO: Adaptive Erase Operation for Improving Lifetime and Performance of Modern NAND Flash-Based SSDs. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, La Jolla, CA, USA, 27 April–1 May 2024; Volume 3, pp. 101–118.
5. Kang, W.; Shin, D.; Yoo, S. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Trans. Embed. Comput. Syst. (TECS)* **2017**, *16*, 1–20. [CrossRef]
6. Yang, P.; Xue, N.; Zhang, Y.; Zhou, Y.; Sun, L.; Chen, W.; Chen, Z.; Xia, W.; Li, J.; Kwon, K. Reducing garbage collection overhead in SSD based on workload prediction. In Proceedings of the 11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19), Renton, WA, USA, 8–9 July 2019.
7. Cheng, W.; Luo, M.; Zeng, L.; Wang, Y.; Brinkmann, A. Lifespan-based garbage collection to improve SSD's reliability and performance. *J. Parallel Distrib. Comput.* **2022**, *164*, 28–39. [CrossRef]
8. Matsui, C.; Arakawa, A.; Sun, C.; Takeuchi, K. Write order-based garbage collection scheme for an LBA scrambler integrated SSD. *IEEE Trans. Very Large Scale Integr. (Vlsi) Syst.* **2016**, *25*, 510–519. [CrossRef]
9. Chen, Z.; Zhao, Y. DA-GC: A dynamic adjustment garbage collection method considering wear-leveling for SSD. In Proceedings of the 2020 on Great Lakes Symposium on VLSI, Virtual Event, China, 7–9 September 2020; pp. 475–480.
10. Pang, S.; Deng, Y.; Zhang, G.; Zhou, Y.; Qin, X.; Wu, Z.; Li, J. PcGC: A Parity-Check Garbage Collection for Boosting 3D NAND Flash Performance. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2023**, *42*, 4364–4377. [CrossRef]
11. Wu, F.; Zhou, J.; Wang, S.; Du, Y.; Yang, C.; Xie, C. FastGC: Accelerate garbage collection via an efficient copyback-based data migration in SSDs. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–28 June 2018; pp. 1–6.
12. Wu, S.; Du, C.; Li, H.; Jiang, H.; Shen, Z.; Mao, B. CAGC: A content-aware garbage collection scheme for ultra-low latency flash-based SSDs. In Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Portland, OR, USA, 17–21 May 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 162–171.
13. Chen, F.; Luo, T.; Zhang, X. CAFTL: A Content-Aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST 11), San Jose, CA, USA, 5–17 February 2011.
14. Kim, B.S.; Yang, H.S.; Min, S.L. AutoSSD: An autonomic SSD architecture. In Proceedings of the 2018 USENIX Annual Technical Conference, Boston, MA, USA, 11–13 July 2018; pp. 677–690.
15. FIU IODedup Traces. Available online: http://iotta.snia.org/traces/block-io/391 (accessed on 2 May 2025).
16. Kim, J.; Lee, C.; Lee, S.; Son, I.; Choi, J.; Yoon, S.; Lee, H.u.; Kang, S.; Won, Y.; Cha, J. Deduplication in SSDs: Model and quantitative analysis. In Proceedings of the 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), San Diego, CA, USA, 16–20 April 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1–12.
17. Chun, M.; Lee, J.; Kim, M.; Park, J.; Kim, J. RiF: Improving Read Performance of Modern SSDs Using an On-Die Early-Retry Engine. In Proceedings of the 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), Edinburgh, UK, 2–6 March 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 643–656.
18. Ye, M.; Li, Q.; Gao, C.; Deng, S.; Kuo, T.W.; Xue, C.J. Stop unnecessary refreshing: Extending 3D NAND flash lifetime with ORBER. *CCF Trans. High Perform. Comput.* **2022**, *4*, 281–301. [CrossRef]
19. Kim, B.S.; Choi, J.; Min, S.L. Design tradeoffs for SSD reliability. In Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST 19), Boston, MA, USA, 25–28 February 2019; pp. 281–294.
20. Gupta, A.; Pisolkar, R.; Urgaonkar, B.; Sivasubramaniam, A. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST 11), San Jose, CA, USA, 15–17 February 2011.
21. Wu, S.; Du, C.; Zhu, W.; Zhou, J.; Jiang, H.; Mao, B.; Zeng, L. EaD: ECC-assisted deduplication with high performance and low memory overhead for ultra-low latency flash storage. *IEEE Trans. Comput.* **2022**, *72*, 208–221. [CrossRef]

22. Yang, H.; Hong, J.; Dan, F.; Lei, T.; Shu, P.Z. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In Proceedings of the International Conference on Supercomputing, Tucson, AZ, US, 31 May–4 June 2011.
23. Narayanan, I.; Wang, D.; Jeon, M.; Sharma, B.; Caulfield, L.; Sivasubramaniam, A.; Cutler, B.; Liu, J.; Khessib, B.; Vaid, K. SSD failures in datacenters: What? when? and why? In Proceedings of the 9th ACM International on Systems and Storage Conference, Haifa, Israel, 6–8 June 2016; pp. 1–11.
24. Dong, Y.; Chen, B.; Pan, Y.; Zou, X.; Xia, W. H2C-Dedup: Reducing I/O and GC Amplification for QLC SSDs from the Deduplication Metadata Perspective. In Proceedings of the 2024 ACM Symposium on Cloud Computing, Redmond, WA, USA, 20–22 November 2024; pp. 704–719.
25. Yen, M.C.; Chang, S.Y.; Chang, L.P. Lightweight, integrated data deduplication for write stress reduction of mobile flash storage. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2590–2600. [CrossRef]
26. Qin, Y.; Feng, D.; Liu, J.; Tong, W.; Zhu, Z. DT-GC: Adaptive garbage collection with dynamic thresholds for SSDs. In Proceedings of the 2014 International Conference on Cloud Computing and Big Data, Sydney, Australia, 3–5 December 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 182–188.
27. Lee, J.; Kim, Y.; Shipman, G.M.; Oral, S.; Kim, J. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 247–260. [CrossRef]
28. Jung, M.; Choi, W.; Kwon, M.; Srikantaiah, S.; Yoo, J.; Kandemir, M.T. Design of a host interface logic for GC-free SSDs. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *39*, 1674–1687. [CrossRef]
29. Sha, Z.; Li, J.; Song, L.; Tang, J.; Huang, M.; Cai, Z.; Qian, L.; Liao, J.; Liu, Z. Low I/O intensity-aware partial GC scheduling to reduce long-tail latency in SSDs. *ACM Trans. Archit. Code Optim. (TACO)* **2021**, *18*, 1–25. [CrossRef]
30. Choi, W.; Jung, M.; Kandemir, M.; Das, C. Parallelizing garbage collection with I/O to improve flash resource utilization. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, Tempe, AZ, USA, 11–15 June 2018; pp. 243–254.
31. Shahidi, N.; Kandemir, M.T.; Arjomand, M.; Das, C.R.; Jung, M.; Sivasubramaniam, A. Exploring the potentials of parallel garbage collection in ssds for enterprise storage systems. In Proceedings of the SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 13–16 November 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 561–572.
32. Gao, C.; Shi, L.; Xue, C.J.; Ji, C.; Yang, J.; Zhang, Y. Parallel all the time: Plane level parallelism exploration for high performance SSDs. In Proceedings of the 2019 35th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 20–24 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 172–184.


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.