# Simplicity as the Ultimate Principle: The Art of Garbage Collection Management in SSDs Inspired by Natural Data Behavior

KEYU WANG, College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
HUAILIANG TAN, College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
KEQIN LI, Department of Computer, State University of New York, New Paltz, United States

As solid-state drives (SSDs) are increasingly used in various computing environments, effective garbage collection (GC) management is crucial for enhancing performance and extending lifespan. Existing GC strategies rely on complex data categorization techniques to distinguish between hot and cold data. This process is not only computationally expensive but also inefficient. This article introduces a revolutionary simplified GC management method, which we call SUP-GC, based on the core design principle that simplicity is the ultimate principle. SUP-GC requires almost no computation and naturally redefines the data storage pattern. All newly written data is defaulted as hot data and stored directly in hot data blocks; data that needs to be moved during the GC process is considered cold data and uniformly migrated to cold data blocks. Our strategy eliminates the need for precise but resource-intensive real-time analysis of data states and instead adopts a storage strategy that is highly consistent with the natural properties of data. This intuitive partitioning significantly reduces the need for complex judgments about data states, thereby optimizing the storage management process. Experiments and designs on real SSDs have shown that our SUP-GC strategy significantly outperforms existing mainstream GC methods.

CCS Concepts: • **Computer systems organization** → **Firmware**; *Embedded software*; Embedded hardware;

Additional Key Words and Phrases: Computational efficiency, flash translation layer (FTL), flash storage, garbage collection (GC), solid state drive (SSD)

**ACM Reference Format:**
Keyu Wang, Huailiang Tan, and Keqin Li. 2025. Simplicity as the Ultimate Principle: The Art of Garbage Collection Management in SSDs Inspired by Natural Data Behavior. *ACM Trans. Storage* 21, 4, Article 36 (November 2025), 34 pages. https://doi.org/10.1145/3725219

## 1 Introduction

NAND flash memory, with its compact size, lack of mechanical noise, shock resistance, light weight, and low energy consumption, has garnered extensive attention in both academia and industry

Authors' Contact Information: Keyu Wang, College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China; e-mail: 274100667@qq.com; Huailiang Tan (Corresponding author), College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan, China; e-mail: tanhuailiang@hnu.edu.cn; Keqin Li, Department of Computer, State University of New York, New Paltz, New York, United States; e-mail: lik@newpaltz.edu.

ACM 1553-3077/2025/11-ART36
https://doi.org/10.1145/3725219

[1–3]. As semiconductor technology continues to advance, **solid-state drives** (**SSDs**) based on flash memory have gradually replaced traditional **hard disk drives** (**HDDs**) as the mainstream storage solution. These SSDs, known for their excellent performance and reliability, are widely used not only in consumer electronics such as desktops and laptops but also play a crucial role in **high-performance computing** (**HPC**) and enterprise data centers [4–9].

An SSD consists of multiple flash memory chips, each divided into numerous blocks and pages. In flash memory, data operations are performed on a per-page basis, while erasure occurs at the block level. Due to the necessity to erase before new data can be written, and the much slower speed of erasure compared to reading and writing, this becomes a performance bottleneck. To mitigate this, SSDs employ an out-of-place updating strategy, where new data is written to new locations and old data is marked as invalid, thus optimizing the overall write efficiency [10–14]. To manage the invalid pages that accumulate during updates, the **Flash Translation Layer** (**FTL**) implements a **garbage collection** (**GC**) mechanism responsible for reclaiming these pages to free up block space. Although this process is crucial for maintaining storage efficiency, the execution of GC is resource-intensive and time-consuming, especially under high system loads, significantly impacting system performance. Therefore, optimizing the GC process to minimize its impact on performance is a key challenge in enhancing SSD performance [15–19].

In the literature, the enhancement of GC performance has been extensively researched, either directly or indirectly, across various flash translation layer schemes. These studies encompass the use of different FTL address mapping strategies [20–22], the selection of appropriate victim blocks [23–25], and leveraging workload characteristics along with internal parallelism to boost performance [26–28]. Among these, the pure page mapping mechanism is one of the most advanced mapping methods adopted by modern high-performance SSDs. The greedy algorithm, as a highly rational choice for selecting victim blocks, has been adopted by various SSD architectures. Internal parallelism is commonly implemented through channel-level parallelism, which is both the most common and effective method.

Despite significant advancements in GC for SSDs, severe performance challenges remain. Hot and cold data separation strategies have been extensively researched and are considered among the most promising optimization approaches for enhancing GC efficiency. By categorizing data into hot and cold based on access frequency, recency, or other characteristics, and combining internal parallelism with optimizations tailored to specific workloads, these strategies effectively improve system performance and reduce write amplification effects [26, 29, 30].

The specific methods for hot and cold data separation primarily include strategies based on Bloom filters, linked lists, and machine learning. Bloom filter–based methods identify hot data by tracking data access frequency, providing high space efficiency and fast query speed. However, they suffer from a certain false positive rate, necessitating design tradeoffs [29, 30]. Linked list–based methods (e.g., two-level LRU lists) identify hot data by managing the access order, but they have several drawbacks, including high computational overhead, large memory consumption, and high implementation complexity. Furthermore, these methods exhibit poor performance under dynamic access patterns, especially in high-concurrency environments, where these limitations significantly impact system performance [27, 31]. Machine learning–based methods predict data heat using historical access patterns, offering higher prediction accuracy and adaptability, but the high implementation complexity and computational overhead limit their applicability in resource-constrained environments [42, 43].

The advantages of hot and cold data separation strategies lie in their ability to reduce the volume of data migration during GC, thereby enhancing GC efficiency. Additionally, by leveraging the parallel processing capabilities of SSDs, these strategies further optimize overall system performance. However, there are two major limitations associated with these strategies. First, under

dynamic and unpredictable access patterns, the accuracy of data classification poses a significant challenge, potentially resulting in performance degradation or increased write amplification. Second, maintaining complex data structures and executing predictive algorithms entail high overheads, which can offset the benefits gained from improved GC efficiency. Consequently, despite the significant advantages of hot and cold data separation strategies, their complexity and associated system overheads limit their effectiveness in practical applications. Future work should focus on developing accurate, effective, and simplified solutions to fully leverage the benefits of hot and cold data separation while minimizing additional system overhead.

This article presents a novel perspective based on the inherent natural behavioral attributes of data, categorizing data into hot and cold states. By combining minimalist design principles with these natural behavioral characteristics, we propose a new GC management strategy named SUP-GC, aimed at enhancing performance during GC operations and improving the durability of SSDs. The core idea of this approach lies in leveraging the natural temporal locality of data along with its aging characteristics in storage systems. All newly written data in SSDs is classified as hot data and stored in hot blocks, while, over time, the valid data retained during GC is classified as cold data and migrated to designated cold blocks. Unlike existing research, which focuses on determining the classification of written data as hot or cold, our strategy directly classifies data based on its natural behavior. The advantage of this approach is that it effectively addresses the tradeoff between precise data classification and the associated computational and spatial overheads observed in previous studies. We have embedded the SUP-GC strategy into actual SSD devices utilizing mainstream and advanced FTL strategies, and extensive experiments and comparisons have validated the correctness and effectiveness of the proposed strategy.

To the best of our knowledge, SUP-GC represents a novel and disruptive GC architecture. It utilizes a minimalist approach, managing data based on its natural behavioral characteristics. This design not only simplifies the data processing workflow but also enhances the overall efficiency and responsiveness of the system. The main contributions of this article are as follows:

(1) This study conducted an in-depth analysis of mainstream high-performance SSD architectures, identifying key performance bottlenecks during the GC process. We also extensively reviewed existing hot and cold data partitioning schemes, highlighting their complexity and inefficiencies in data identification. These findings underscore the urgent need for the development of more effective data classification technologies to enhance GC efficiency.

(2) We propose a novel GC management strategy, named SUP-GC. This strategy is based on the inherent behavioral characteristics of data and employs minimalist design principles for hot and cold data state classification, aimed at optimizing system performance during GC operations and enhancing the durability of SSDs. By precisely managing data storage and migration within flash blocks, this strategy significantly improves data handling efficiency.

(3) We integrated the SUP-GC strategy into actual SSDs and conducted extensive experimental validation using current mainstream and advanced FTL strategies. The experimental results not only confirmed the effectiveness of the SUP-GC strategy but also demonstrated its significant enhancement of system responsiveness during the GC process and its effective reduction in the number of flash erasures, thereby validating the performance advantages and technical feasibility of the strategy in practical applications.

The remainder of this article is structured as follows: Section 2 provides the background and motivation underlying our study. Section 3 elaborates on the details of the SUP-GC scheme. Implementation and comparative schemes are discussed in Section 4. Section 5 details the results of our performance evaluation. Section 6 reviews work related to our study. Finally, Section 7 concludes the article.

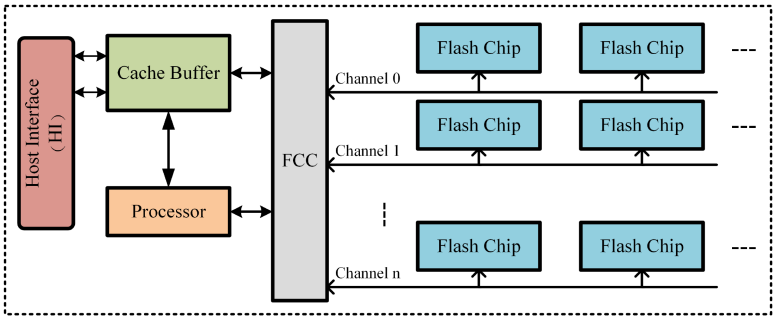



Fig. 1. A typical architecture diagram of SSD.

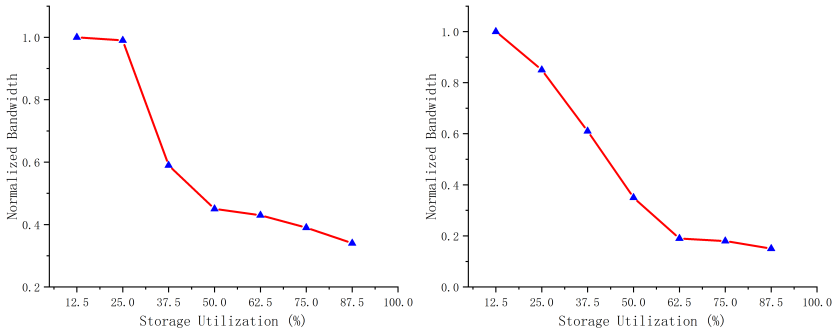
## 2 Background and Motivation

In this section, we first introduce the device organizational architecture of NVMe SSDs relevant to this research. We then explore the primary limiting factors of SSD performance, focusing on GC and the impact of existing hot and cold data classification strategies on the GC process. Building on this foundation, we clearly articulate the motivation for our research and provide a detailed description of the design rationale behind our proposed SUP-GC scheme.

### 2.1 SSD Architecture

Figure 1 shows a typical SSD architecture diagram. SSDs consist of key components: the host interface, microprocessor, on-board RAM, and flash memory. Traditionally, SSDs used the SATA interface, but its limited throughput has been outpaced by the NVMe protocol over PCIe interfaces, offering higher **bandwidth** (**BW**) and better support for internal operations. The microprocessor handles the firmware, including the FTL which manages tasks like address mapping and GC. These processors are generally low-power, limiting their capacity for complex algorithms. On-board RAM supports firmware processes and acts as a buffer to smooth data transfers and minimize delays, particularly useful during operations like GC. Lastly, flash memory stores data, operating in various capacities from SLC to QLC, which store one to four bits per cell respectively, enhancing the storage density and performance of SSDs.

SSDs exhibit substantial advantages in parallel processing compared to traditional HDDs, with their architecture enabling channel-level, plane-level, die-level, and block-level parallelism. This extensive parallelism, particularly at the channel-level, is utilized in high-performance SSDs to maximize flash memory BW, essential for accelerated data processing. In managing storage, these devices employ a GC mechanism due to the inability of flash memory to overwrite existing data. To reclaim space, old blocks must be erased before new data can be written, a process managed through strategies such as sequential, random, and primarily, greedy collection methods, which are favored for their efficiency. Additionally, the FTL of SSDs involves complex address translation mechanisms to map logical addresses from the host to physical addresses on the flash memory. These translations occur at block-level, hybrid-level, and page-level, each offering different balances of performance and memory space overhead. Previously, the high cost of RAM limited SSD design options, but recent advancements in semiconductor technology have mitigated these cost issues, allowing the integration of large-capacity RAM. This development supports advanced page-level address translation, which is critical for achieving optimal performance in today's high-performance SSDs by optimizing access and storage efficiency.




(a) Intel SSD DC P3520 (b) Western Digital Black SN750

Fig. 2. Impact of GC on commercial SSDs.

## 2.2 Performance Bottlenecks and Challenges of Related GC Strategies

This section begins by examining the primary performance bottlenecks affecting SSDs, with a particular focus on the challenges encountered during GC processes. Subsequently, we analyze existing GC strategies, evaluating their effectiveness and limitations in mitigating these performance bottlenecks.

In SSDs, the primary bottlenecks to performance are rooted in the physical limitations of the storage media and the design of the storage system architecture. One of the most critical processes affecting SSD performance is GC, which is necessitated by the non-overwrite characteristics of NAND flash memory. This process involves identifying blocks that contain invalid pages and migrating the remaining valid data to new blocks, a procedure that not only consumes substantial system resources but also exacerbates the write amplification effect, significantly reducing the lifespan of the SSD. Moreover, as the SSD's storage space nears capacity, the reduced availability of free blocks necessitates more frequent GC operations, particularly impacting performance in environments requiring high-speed data processing, such as HPC and enterprise data centers.

We conducted experiments using the FIO tool on two commercial SSD models (Intel SSD DC P3520 and Western Digital Black SN750) to evaluate the impact of GC on SSD performance. Figure 2(a) and (b) presents the performance results for the Intel and Western Digital SSDs, respectively. During the experiments, we observed that as storage space gradually became occupied, the write pressure on the SSD increased, leading to a significant rise in the frequency of garbage collection operations. These frequent GC processes had a noticeable negative impact on SSD throughput, further demonstrating the limiting effect of GC on overall SSD performance.

In recent years, hot-cold data classification strategies have become an important research focus in the field of SSD data management, due to their significant advantages in optimizing the GC process. Hot-cold data classification is based on data access frequency, dividing data into frequently accessed "hot data" and less frequently accessed "cold data" to reduce the frequent data migrations, as well as the associated write and erase cycles, during GC. Device-side FTL often use traditional methods such as linked lists [27, 31] and Bloom filters [29, 30] to support dynamic data classification and rapid identification. Linked lists are used to dynamically reorder data pages based on access frequency, while Bloom filters are used to record the access status of data pages to assist in quickly identifying their hot or cold attributes.

However, these major research directions face numerous challenges in practical applications. For example, traversing linked lists can significantly impact performance when handling large-scale data, and the memory overhead for node management can be burdensome in resource-constrained environments. Additionally, the false positive tendency of Bloom filters may lead to cold data

being misclassified as hot data, thereby increasing the frequency of GC operations and negatively impacting overall performance. Furthermore, both linked lists and Bloom filters can only classify data based on short-term access frequency, which presents limitations when applied over broader time frames, meaning they fail to accurately identify long-term hot or cold data changes. As data access frequency may vary over time, this short-term classification approach can lead to severe hot-cold data misclassification, ultimately reducing the efficiency of GC and storage management. Therefore, although linked lists and Bloom filters have significant theoretical value for device-side FTLs, their widespread application in real-world environments is limited by tradeoffs in complexity and performance, as well as difficulties in maintaining accurate data classification on a global scale.

With the development of host-side FTLs, researchers have begun exploring new device architectures, such as **Zone Namespace** (**ZNS**) SSDs and Open-Channel SSDs [7, 44–47], while extensively incorporating machine learning techniques to improve data classification accuracy in hot-cold data segregation. ZNS SSDs divide storage into multiple zones, enabling the host to write data sequentially, which reduces the need for internal data management and GC. Open-Channel SSDs, on the other hand, provide greater flexibility to the host, allowing optimization of data placement and GC strategies based on workload characteristics. However, this flexibility also introduces new complexities, requiring the host to have precise flash management capabilities. Although machine learning has significant advantages in improving the accuracy of hot-cold data identification, it also presents limitations in practice. First, machine learning methods require specific software and hardware support, making it difficult to popularize in most current device-side FTL SSD scenarios. Second, machine learning involves substantial training and runtime requirements, especially in large-scale, write-intensive scenarios, where the computational overhead becomes a critical bottleneck, significantly affecting its feasibility and widespread applicability in real-world applications [19, 42, 43].

In summary, research on hot-cold data classification has evolved from traditional device-side FTL methods to host-side FTLs with machine learning-driven new architectures, aiming to enhance the efficiency of SSD data management and GC. However, these methods face various challenges in practical applications. Balancing classification accuracy, hardware resources, and system overhead remains a crucial direction for future research.

Apart from hot and cold data classification, other advanced GC strategies have also been evolving. Techniques such as suspend GC and idle-time GC aim at optimizing the timing of GC operations, thereby minimizing their impact on foreground I/O performance [11, 48, 49]. These methods attempt to reduce GC interference with I/O operations by executing GC tasks during system idle periods. However, these approaches do not fundamentally improve GC efficiency; rather, they optimize resource scheduling in terms of timing. Furthermore, in scenarios characterized by large-scale intensive write operations, the system often lacks sufficient idle time, making these GC strategies less effective, which may further affect overall system performance.

Furthermore, cache technologies play a crucial role in improving SSD performance. Through a series of cache replacement strategies, the data access patterns can be effectively optimized, thereby enhancing SSD read and write performance and improving the overall system responsiveness [2, 50, 51]. However, these strategies are not specifically designed for GC optimization, and their impact on the extensive data migration and erase operations during GC is limited, showing constraints in specific application scenarios. Cache strategies specifically optimized for GC, such as GCaR [17], aim at reducing the impact of the data flow during GC processes by temporarily caching GC-related data. However, GCaR faces practical challenges: it does not adequately consider the effects of algorithmic complexity on SSD performance, which may lead to performance degradation during high-load conditions, and its effectiveness is constrained by

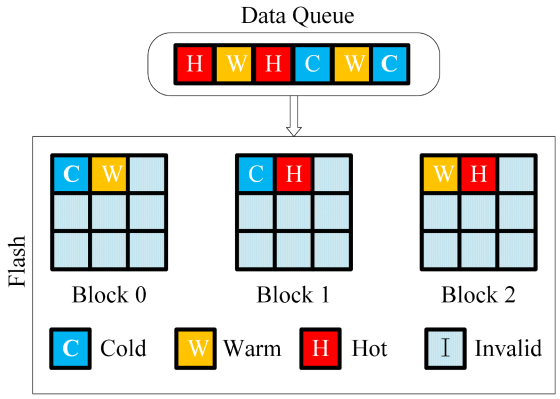



Fig. 3. A typical data dispatch process diagram of existing SSD.

limited cache capacity. Overall, although cache strategies theoretically contribute to improving GC efficiency, their practical effectiveness is often limited due to limited cache capacity and the computational overhead of the algorithms.

### 2.3 Motivation

Despite significant progress in GC research, SSDs still face severe performance challenges in write-intensive environments, such as HPC and enterprise data centers. Current GC strategies, particularly those involving dynamic hot and cold data classification, though theoretically effective, suffer from various limitations. The classification of hot and cold data is particularly challenging in mainstream device-side FTL SSDs, where computational and memory overheads are difficult to control effectively. Moreover, improper data classification can lead to complex internal I/O blocking issues, which makes these methods less effective under high-load conditions. Therefore, while existing GC methods show some potential for improving SSD performance, their complexity and resource demands limit their effectiveness in practical applications. Developing a more efficient, scalable GC scheme that can adapt to diverse SSD usage scenarios, thereby better balancing the tradeoff between performance improvement and system resource overhead, remains a critical research direction.

In the existing SSD data dispatch process, as illustrated in Figure 3, data is written to flash memory without specific differentiation between hot and cold data. Data in the queue, whether it is cold data (C), warm data (W), or hot data (H), is written to different blocks in the flash memory in a first-in, first-out manner. This lack of specialized handling results in inefficient GC because all types of data are treated equally during the GC process. Consequently, frequent data migrations and write-erase operations occur, inevitably accelerating flash memory wear and reducing overall system performance.

To address the aforementioned issues, researchers have extensively explored hot and cold data separation strategies, which have been applied to various storage systems, including SSDs [26–30, 42, 52]. These strategies typically improve efficiency by classifying data at the time of writing. As shown in Figure 4, data in the queue is categorized based on access frequency into cold data, warm data, and hot data, and then stored in different blocks accordingly. For instance, cold data is written to Block 0, warm data to Block 1, and hot data to Block 2. This approach aims at reducing unnecessary data migrations and enhance GC efficiency.

However, existing hot and cold data separation strategies face significant challenges in practical applications. These strategies often rely on real-time data access pattern analysis and complex

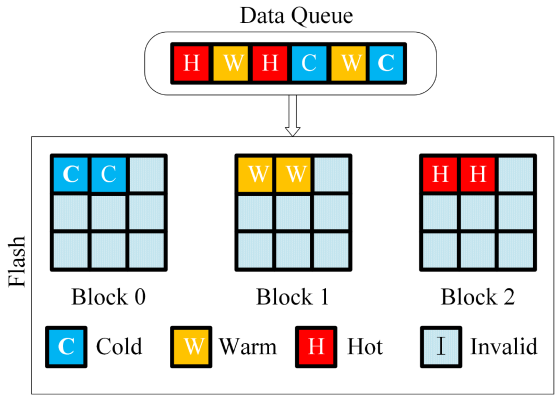



Fig. 4. A typical data dispatch process diagram of an SSD with hot and cold data separation.

classification algorithms. Bloom filter–based methods identify hot data by tracking data access frequency, offering high efficiency and low memory usage, but they may suffer from false positives, necessitating a tradeoff between accuracy and overhead [29, 30]. Linked list–based methods (e.g., two-level LRU lists) determine data heat by managing the order of data access, but due to frequent operations on list nodes, they incur high computational overhead and memory usage, and their performance is unstable under high concurrency and dynamic workloads [27, 31]. Machine learning–based methods predict data heat by analyzing historical access patterns, offering strong adaptability and high prediction accuracy, but the complexity of model training and inference requires considerable computational resources and memory, limiting their applicability in resource-constrained environments [42, 43]. These methods often require the continuous maintenance of numerous counters or complex data structures, which not only increases the computational and memory burden on SSD controllers but may also lead to increased system latency. Moreover, due to the diversity and dynamic nature of workloads, accurately distinguishing between hot and cold data is challenging, and incorrect classification may reduce the effectiveness of these strategies.

To quantify the overhead and impact on system performance of existing hot and cold data separation strategies, we conducted experiments on OpenSSD. We implemented a typical frequency-based hot and cold data classification algorithm in the OpenSSD firmware targeting cache data. We measured the relationship between the execution time of data classification during GC and the volume of data, as well as its impact on system performance. Figure 5 presents the relationship between data classification workload and SSD performance, illustrating that as the volume of data requiring classification increases, SSD throughput and efficiency significantly decline. This outcome demonstrates that complex algorithms indeed affect SSD I/O performance and suggests that complex machine learning algorithms, due to their high computational complexity, are more suitable for execution on the host side rather than on the device side. Machine learning–based hot and cold data classification strategies have complexities far exceeding the typical classification strategy used in this experiment. Another scheme based on Bloom filters, although having lower computational overhead, still introduces a certain degree of computational burden (this will be further explained in the experiments). These results further confirm the significant impact of the GC process on SSD performance, indicating that complex hot and cold data classification strategies introduce substantial system overhead in practical applications, potentially offsetting their theoretical advantages in optimizing GC.

Given these challenges, we recognize the urgent need for a novel GC management strategy that can achieve the benefits of hot and cold data separation without introducing significant

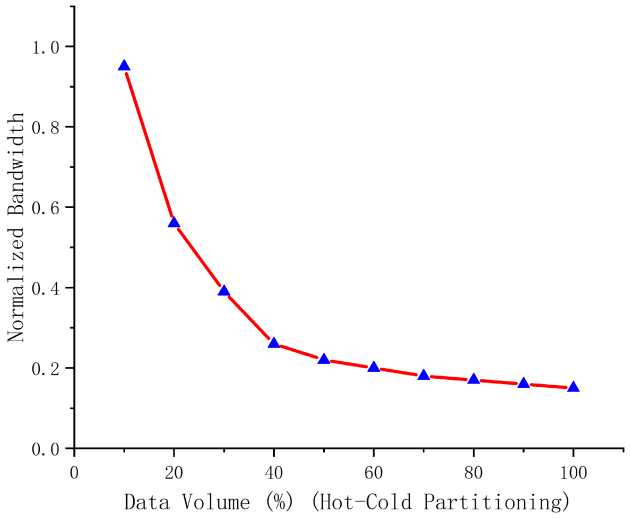



Fig. 5. Impact of data classification volume on SSD performance.

system overhead, thereby improving GC efficiency and the overall performance of SSDs. However, achieving both minimal computational overhead and avoiding complex I/O blocking while perfectly separating hot and cold data seems to be an unattainable goal. Therefore, we approached this problem from an entirely new perspective. Previous studies have often artificially pursued an ideal separation of hot and cold data, neglecting the natural distribution properties inherent within SSDs. Is it possible that hot and cold data within SSDs could inherently exist in a state of natural separation? Fortunately, we found such a state. Based on this insight, we proposed a minimalist SUP-GC strategy, which leverages natural data behavior.

This strategy adheres to the most common natural distribution of hot and cold data and follows a minimalist design principle without requiring real-time data analysis. The rules of SUP-GC are as follows: according to the natural temporal properties of data, all newly written data is automatically treated as hot data and is directly written into hot data blocks. During the GC process, valid data in flash blocks selected for reclamation is treated as cold data and migrated to dedicated cold data blocks. By exploiting the natural hot and cold properties of data, this approach avoids complex predictive algorithms, thereby eliminating computational overhead and mitigating I/O blocking issues.

## 3 SUP-GC

In this section, we thoroughly explore the SUP-GC scheme, an innovative GC management strategy designed to optimize the performance and extend the lifespan of SSDs. By simplifying the data processing workflow, SUP-GC significantly reduces system overhead and enhances data processing efficiency. This section first provides an overview of the system architecture, and then delves into a detailed discussion of its two core strategies: the data allocation strategy during the data writing process (Data Allocation) and the data migration strategy during GC (Garbage Collection Migration).

### 3.1 System Overview

Figure 6 shows a system overview of an SSD storage system based on the SUP-GC scheme. The host side consists of an application layer, a file system layer, and a block device layer. The SSD connects

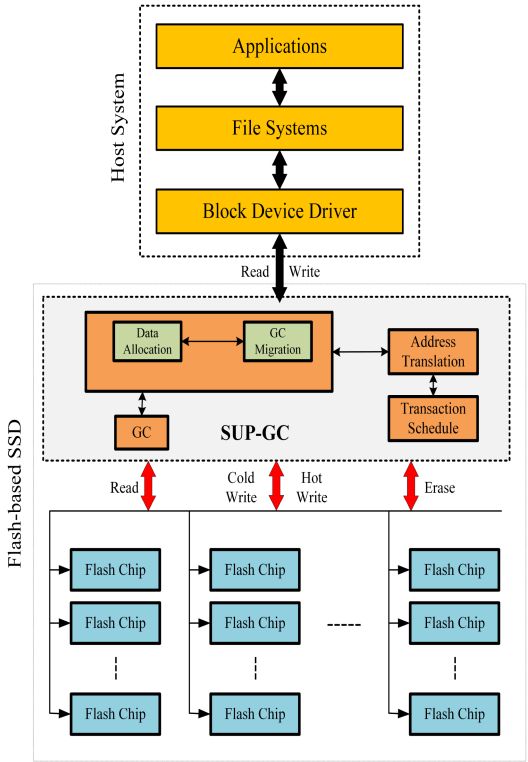



Fig. 6. System overview of an SSD storage system based on the SUP-GC scheme.

to the host via a PCIe interface using the NVMe protocol, through which the host's I/O requests are transmitted to the SSD. These requests are processed by SUP-GC before being sent to the flash memory. SUP-GC comprises two main components—Data Allocation and GC Migration—both of which operate within the FTL.

Figure 7 illustrates the overall execution process of the SUP-GC mechanism. The process begins when the host issues a write request, prompting the NVMe manager to generate a corresponding I/O write request. The system then checks for sufficient free blocks within the memory. If adequate free blocks are available, the system proceeds with address mapping, followed by Data Allocation, where new data is written into hot data blocks. In contrast, if the system detects insufficient free blocks, the GC mechanism is initiated. During this process, the GC Migration module transfers valid data to cold data blocks and releases invalid blocks, thereby reclaiming storage space. Once these steps are completed, the system returns to the original I/O write request and continues the write operation until the request is fully processed.

The Data Allocation module handles write requests from the host, where all newly written data is by default considered hot data and is directly stored in hot data blocks, thereby avoiding complex and resource-intensive real-time data state analysis. As data usage frequency naturally changes, previously hot data that becomes inactive is recognized as cold data during the GC process and is uniformly migrated to cold data blocks by the GC Migration module. This intuitive partitioning significantly reduces the need for complex judgments about data states, optimizing the storage management process. By effectively distinguishing and managing data states, SUP-GC significantly enhances the efficiency of data migration during GC, thereby improving the

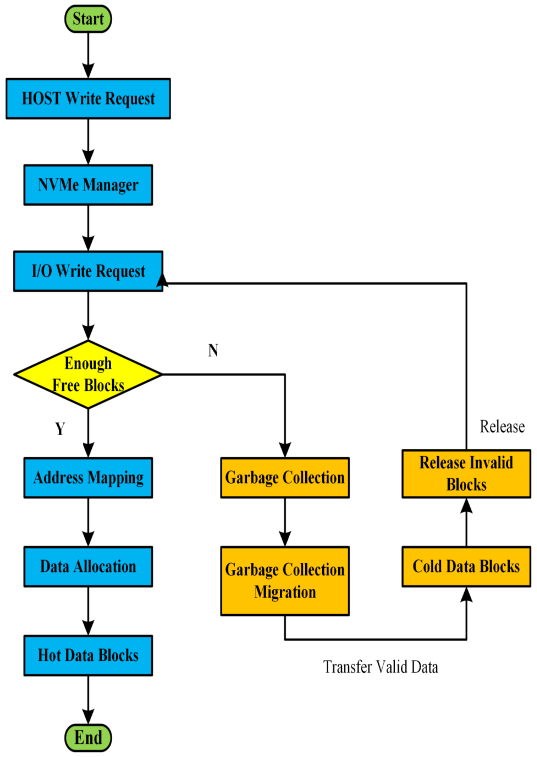



Fig. 7. System design flowchart of SUP-GC mechanism.

performance and durability of SSDs. In the following sections, we will explore in detail the specific design of Data Allocation and GC Migration.

### 3.2 Data Allocation

In mainstream SSDs, data allocation strategies typically adopt a minimalist approach by directly distributing data to flash memory blocks through channel-level parallelism to optimize write efficiency. This method significantly reduces computational and spatial overhead. However, during the GC process, the lack of effective data temperature segmentation often leads to inefficient reclamation. Although academic research has proposed methods to enhance GC efficiency by categorizing data into hot and cold data, these methods often overlook the computational and spatial costs involved in implementing these strategies in actual SSD systems.

The SUP-GC strategy addresses this issue by leveraging the principle of temporal locality. Temporal locality is a fundamental concept in computer architecture and memory hierarchy design, stating that data recently accessed or modified is more likely to be accessed again in the near future. In the context of SSDs, this means that newly written data is expected to have a higher frequency of subsequent access. Therefore, SUP-GC automatically classifies all newly written data as hot data and stores it in designated hot data blocks. By doing so, we improve data access efficiency and reduce unnecessary erase operations and write amplification because frequently updated data remains in hot data blocks, minimizing interference with less frequently accessed cold data.

To illustrate this concept, Figure 8 presents a conceptual diagram of the theoretical foundation of our data allocation strategy. As shown in the figure, hot data is directly written into hot data blocks,

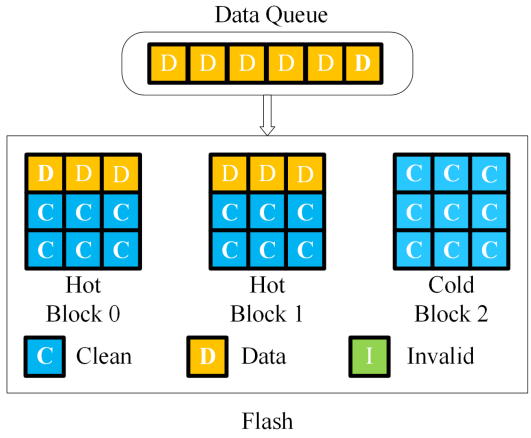



Fig. 8. SUP-GC data allocation concept.

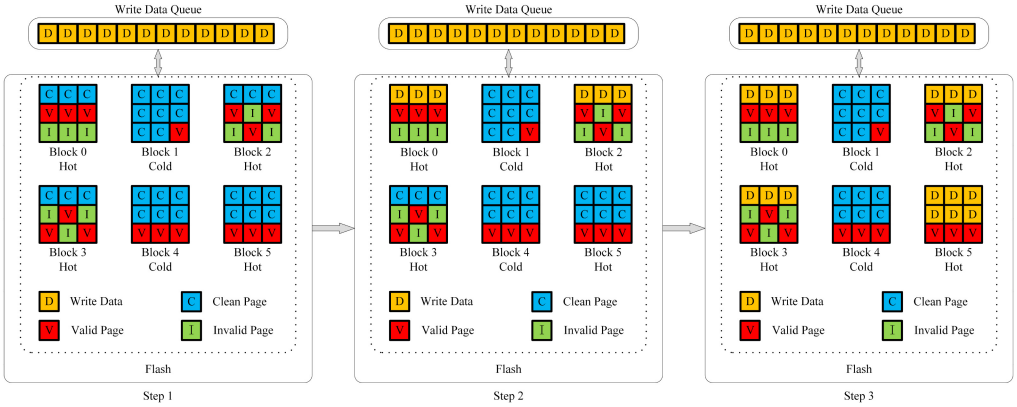



Fig. 9. A typical data allocation scenario.

while cold data blocks do not receive any new data writes. This visual representation emphasizes how SUP-GC leverages temporal locality by isolating hot data into specific blocks, thus optimizing performance and reducing overhead.

This approach not only maintains the high write efficiency of high-performance SSDs but also avoids unnecessary computational and storage overhead during the data allocation phase, thereby optimizing the entire data management process. Through this method, the SUP-GC strategy significantly enhances the overall system performance and responsiveness while reducing the impact of GC on system performance. Additionally, it simplifies the decision-making process in data handling by eliminating the immediate need for complex analysis of data usage patterns, which typically consumes substantial computational resources.

In Figure 9, we illustrate a typical data allocation scenario under the SUP-GC strategy. In this scenario, D represents new data waiting to be written, located in the write queue; V denotes valid data pages; I indicates invalid data pages; and C represents clean pages, that is, pages that have not yet been written to. Initially, Figure 9 shows the D data in the write queue ready to be written. At this point, the system includes four hot data blocks (Block 0, Block 2, Block 3, and Block 5), which contain valid and invalid data pages as well as clean pages, while the two cold data blocks (Block 1 and Block 4) mainly consist of clean pages and valid data. In the second step illustrated in

**ALGORITHM 1:** Data Allocation Algorithm in SUP-GC

**Require:** A sequence of write requests from the host $\{(LSA_i, data_i)\}$, where $i = 1$ to $n$
**Ensure:** Allocate and write data to flash memory blocks

```
for each write request (LSA_i, data_i) do
    /* Check if there are sufficient free pages in hot data blocks */
    if there are available free pages in hot data blocks then
        Map the logical address LSA_i to a physical address in a hot block
        Write data_i to the mapped physical address
    else
        Trigger Garbage Collection (see Algorithm 2)
        After garbage collection, map LSA_i to a physical address in a hot block
        Write data_i to the mapped physical address
    end if
end for
```

Figure 9, the new data marked as D in the write queue begins to be allocated across various data blocks. When writing data to the flash blocks, the system first checks for clean pages in Block 0. If clean pages are available, the data D is written to these pages. Once Block 0 is filled, the system proceeds to check Block 1 for clean pages, according to the logical sequence of blocks. However, as Block 1 is a cold block, the system skips it and moves to the next logical block, Block 2, where clean pages are available and thus data D is written into Block 2. As new data fills Blocks 0 and 2, the pages previously marked as C for clean are now transformed to D, indicating that the new data has been written. In the third step, more data from the write queue is written into the data blocks. The system first checks whether Block 3 contains clean pages. If clean pages are found, the new data D is written into Block 3. The system then checks the next logical block for clean pages. However, since Block 4 is a cold block, the system also skips it and proceeds to the next logical block, Block 5, which meets the requirements for writing. Consequently, new data D is written into Block 5. Ultimately, the clean pages in Blocks 3 and 5 are filled with new data D. Throughout this process, the SUP-GC strategy allocates new data to hot blocks, aiming to avoid mixing new data with cold data in cold blocks, thereby minimizing unnecessary erasure operations and optimizing the use of clean pages to maintain the performance of the flash memory.

Pseudocode 1 illustrates the data allocation process within the SUP-GC framework, optimizing the data writing procedure for SSDs. The algorithm begins by processing incoming write requests from the host, each associated with a **logical slice address** (**LSA**) and the corresponding data. The LSA is a logical address used to abstract the physical storage location of data. The LSA establishes a mapping between the host system and physical storage, making data management within SSDs more flexible and efficient. Each LSA corresponds to a set of data, and through this logical-to-physical mapping relationship, the SSD can efficiently locate and access the data stored internally. For each write request, the system first checks whether there are sufficient free pages available in the hot data blocks. If free pages are available, the algorithm maps the logical address to a physical address in a hot block and writes the data to this location. This data allocation strategy ensures that new data is preferentially written to hot blocks, effectively preventing mixing with cold block data, thereby significantly reducing unnecessary erase operations and enhancing data storage efficiency. However, if there are insufficient free pages in the hot data blocks, the algorithm triggers the GC process, as detailed in Algorithm 2. After GC frees up space, the algorithm proceeds to map the logical address to a physical address in a hot block and writes the data accordingly. By integrating GC seamlessly into the data allocation routine, the algorithm maintains optimal write performance without introducing significant delays or computational overhead.

From the preceding analysis, it is evident that the data allocation strategy adopted by SUP-GC enables the preservation of the existing advantages associated with high-performance SSD writing, while circumventing the complexities of computational overhead and RAM space consumption. This efficiency is achieved by foregoing the categorization of data into hot and cold types; instead, it automatically classifies incoming write data as hot based on its inherent characteristics and segregates this from the existing cold data within the flash memory. However, this mechanism of hot data allocation alone does not fully leverage the capabilities of SUP-GC. Subsequently, we will explore another fundamental strategy within SUP-GC that dictates the distribution and migration of cold data during the GC process.

### 3.3 GC Migration

In both industrial and academic contexts, the greedy algorithm is commonly employed as a strategy for selecting victim blocks for GC, due to its effectiveness in identifying and reclaiming blocks with the greatest number of invalid pages, thereby rapidly freeing up storage space. However, this method typically does not involve further processing of the remaining valid data in the reclaimed blocks. Over time, these valid data, due to their low access frequency, often evolve into naturally cold data.

The GC migration strategy in SUP-GC is grounded in two fundamental concepts: temporal locality and data aging. Temporal locality is a well-established principle in computer architecture, stating that data that has been accessed recently is more likely to be accessed again in the near future. In contrast, data that is accessed less frequently over a longer period can be broadly categorized under data aging. Data aging is a phenomenon in storage systems that refers to the gradual decline in the frequency of data access as time progresses. In the context of SSDs, frequently updated hot data are often invalidated over a short period, whereas the data that remain valid during GC are generally those that have not been modified for an extended duration, inherently qualifying them as naturally cold data.

By leveraging this understanding, SUP-GC identifies these valid data as cold during the GC process and migrates them to designated cold data blocks, effectively achieving the isolation of hot and cold data and minimizing their intermixing. This approach simplifies the data management process, reduces interference with hot data, and consequently enhances the system's response time and data access efficiency.

To illustrate this concept, Figure 10 presents a conceptual diagram of the theoretical foundation of our GC migration strategy. As shown in the figure, the GC process involves identifying and migrating valid data. Specifically, the victim block is selected for GC because it contains a large number of invalid pages, but it still has some valid data. Since these valid data pages have not been accessed for a long time, they are classified as cold data and are migrated to cold data blocks.

Figure 11 illustrates a typical GC data migration scenario under the SUP-GC strategy, with symbol definitions consistent with those in Figure 11: D represents new data waiting in the write queue, V denotes valid data pages, I indicates invalid data pages, and C stands for clean pages that have not yet been written to. In the initial state (Step 1), new data D is ready for writing, with blocks 0, 2, 3, and 5 containing a mix of valid and invalid data, while blocks 1 and 4 mainly contain a large number of clean pages and some valid data. As the system progresses to Step 2, the new data D begins to be written to flash memory. Due to the lack of clean pages in blocks 0, 2, 3, and 5, the system needs to perform GC operations. Following a greedy algorithm, the system first recycles block 0, which has the most invalid pages, transferring its valid pages to the cold block 1, after which the new data D occupies block 0. When block 0 is filled, the system searches for the next available block, finds no free blocks, and thus performs GC on block 2, transferring its valid pages to block 1. Subsequently, block 1 is erased to become a new clean block, and new data D is written

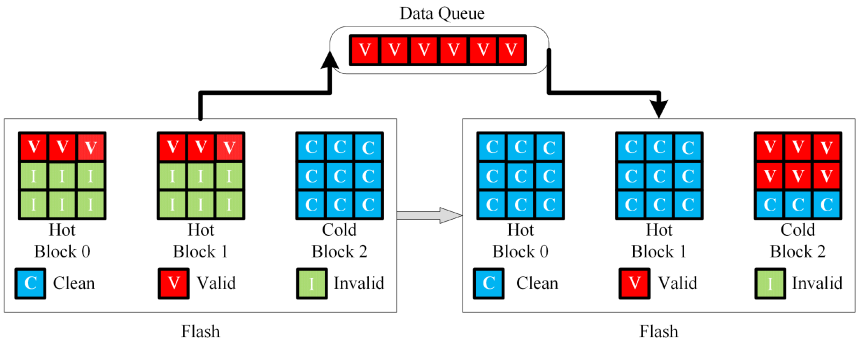



Fig. 10. SUP-GC garbage collection migration concept.

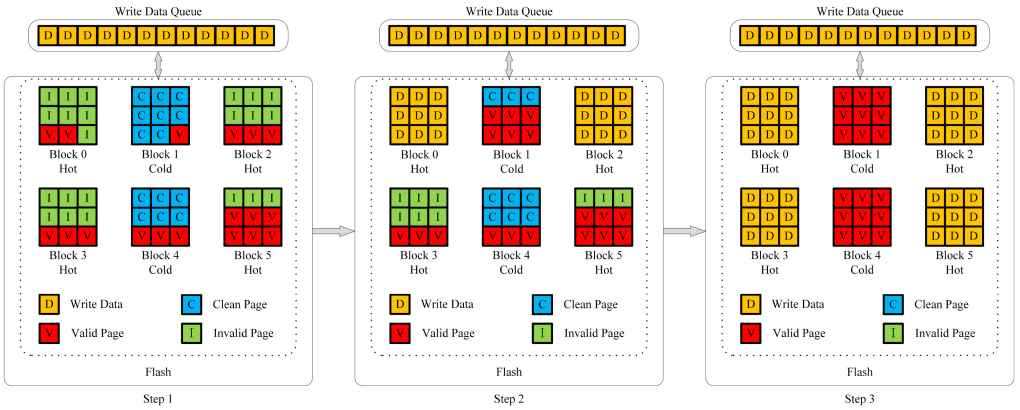



Fig. 11. A typical GC data migration scenario.

into block 2. In Step 3, the valid pages from block 3 are transferred to block 1 by the same principle, followed by the erasure of block 3 and the writing of new data D. As operations continue, block 5 also enters GC status, requiring the migration of valid pages. Since cold block 1 is now full, the valid data from block 5 is transferred to the next available cold block, block 4, after which block 5 is erased, and new data D is written into block 5. This SUP-GC strategy effectively stores valid pages in cold blocks, efficiently preventing the mixing of cold and hot data, and this data migration strategy does not impose additional computational or space costs on the system.

Pseudocode 2 illustrates the GC migration process within the SUP-GC framework, optimizing the management of valid data during GC in SSDs. The algorithm is triggered when the system identifies insufficient free pages in the hot data blocks during the data allocation process. Upon activation, the algorithm employs a greedy strategy to select victim blocks for GC. These victim blocks are chosen because they contain the highest number of invalid pages, thereby maximizing the efficiency of space reclamation. For each selected victim block, the algorithm iterates over all valid pages contained within it. The valid data from these pages is read and migrated to cold data blocks that have available free pages. By transferring valid data to cold blocks, the algorithm effectively segregates cold data from hot data, reducing interference between frequently and infrequently accessed data. This segregation minimizes write amplification and unnecessary erase operations on hot blocks, enhancing both the performance and the lifespan of the SSD. After all valid data has been migrated, the victim block is erased, reclaiming space that can be used for future write operations. The mapping table is updated accordingly to reflect the

**ALGORITHM 2:** Garbage Collection Migration Algorithm in SUP-GC

**Require:** Garbage collection is triggered when there are insufficient free pages in hot data blocks
**Ensure:** Reclaim space and migrate valid data to cold data blocks

```
Select victim blocks for garbage collection using the greedy algorithm
for each victim block do
    for each valid page in the victim block do
        Read the valid data
        Select a cold data block with free pages
        Write the valid data to the cold data block
        Update the mapping table with the new physical address
    end for
    Erase the victim block to reclaim space
end for
Proceed with the data allocation process (see Algorithm 1)
```

new physical locations of the migrated data. Once the GC process is complete, the algorithm returns control to the data allocation process, allowing it to proceed with handling pending write requests.

This GC strategy leverages the natural aging of data; data that remains valid during GC is likely to be less frequently accessed in the future. By migrating such data to cold blocks, the SUP-GC framework avoids the computational overhead of real-time data heat analysis while still achieving effective separation of hot and cold data. Overall, this method enhances storage efficiency, reduces write amplification, and contributes to the improved durability and performance of SSDs.

The SUP-GC strategy fundamentally eliminates the computational overhead associated with traditional hot-cold data segregation processes. In this strategy, all newly written data is automatically considered hot, and data migration during GC relies solely on simple activity-level judgments, avoiding complex real-time data analysis or heat calculations. This direct and effective method significantly reduces the processor's burden, enhancing the system's overall efficiency and responsiveness. Additionally, the SUP-GC strategy simplifies the management of data blocks by merely marking them as hot or cold, greatly reducing the need to maintain state information in RAM. This minimalist management strategy not only optimizes memory usage but also simplifies the operations during the GC process, adding minimal extra RAM overhead. Theoretically, this simple design based on natural data behavior has clear advantages, and its practicality and efficiency will be validated in the subsequent experimental section. Detailed performance evaluations will demonstrate how the SUP-GC strategy significantly enhances SSD responsiveness and data processing capabilities while reducing computational and space overhead, proving its practical value in high-load environments. The upcoming experimental section will further showcase the performance of the SUP-GC strategy in actual applications, providing solid evidence of its innovative contributions to SSD GC optimization.

## 4 Implementation

This section describes the implementation of the SUP-GC architecture and the implementation of other mainstream comparative schemes. All experiments were conducted on the Cosmos+ FPGA OpenSSD [32]. We modified the firmware layer of the Cosmos+ FPGA OpenSSD and compiled the source code written in C language into the firmware of the Cosmos+ FPGA OpenSSD development board using cross-compilation techniques.

Cosmos+ FPGA OpenSSD is an open-source SSD development platform designed to support research and education in flash-based SSD technology. The platform features two advanced ARM

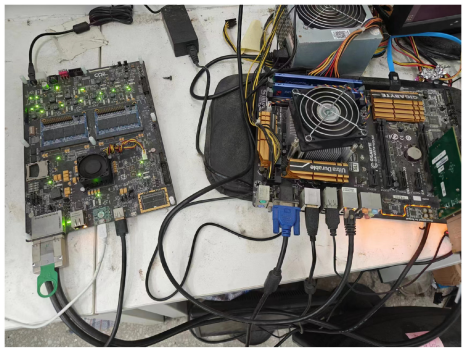


Fig. 12. Cosmos+ FPGA OpenSSD connected to the host system.

Cortex-A9 cores as its microprocessor, with other hardware implemented in FPGA. For address mapping, Cosmos+ OpenSSD uses a pure page mapping method, while its cache replacement algorithm follows the LRU strategy. GC is handled using a greedy algorithm. Additionally, Cosmos+ FPGA OpenSSD utilizes the NVMe protocol to connect with the host system through the PCIe interface. Figure 12 illustrates the connection of Cosmos+ FPGA OpenSSD to the host system via the PCIe interface.

**SUP-GC scheme implemented in Cosmos+ FPGA OpenSSD.** In implementing SUP-GC within the Cosmos+ FPGA OpenSSD firmware, we modified the metadata within the blocks to include two new data types: hot block and cold block. New data is written to hot blocks within the write function, and during the GC process, valid data from the collected blocks is written into cold blocks. This entire process does not introduce additional computational overhead or space overhead beyond the metadata.

**Cache-related GC scheme implemented in Cosmos+ FPGA OpenSSD.** Cache, as a key component in SSDs, significantly enhances data processing speed by reducing read/write latency and optimizing the data storage process. Among the many academic studies aimed at improving SSD performance, GCaR [17] is particularly notable in the cache domain. Initially implemented in the Disksim simulator, it was later adapted and applied to the OpenSSD platform. In this process, we adjusted the data buffer area of Cosmos+ FPGA OpenSSD to temporarily store cache entries related to GC during cache replacement, thus enhancing the system's response speed during GC. Additionally, while maintaining the core concept of GCaR, we made several technical improvements and lightweight design modifications, enabling it to effectively operate on real SSD devices. It's worth mentioning that the GCaR referred to here is not specifically the original GCaR plan but represents a typical example of this class of cache-related GC technologies. We refer to this enhanced implementation as GCaR-Optimized.

**Bloom filter-related GC scheme implemented in Cosmos+ FPGA OpenSSD.** Bloom filter-based strategies [29, 30] are used to differentiate cold and hot data in SSDs by identifying cold and hot data associated with logical addresses. This method can efficiently identify obsolete data blocks without exhaustive checks, thus reducing the overhead of traditional GC methods. Although different implementations may vary, they share a common core principle: using Bloom filters to identify cold and hot data and achieve data isolation within flash memory blocks. We optimized these technical details and adopted a lightweight design to ensure these strategies can operate efficiently on real SSD devices. Additionally, given the effectiveness of Bloom filters in SSDs, we consider these strategies as a representative of advanced hot and cold data classification schemes and evaluate their performance, referring to them as BF-Optimized.

Table 1. Cosmos+ OpenSSD Platform Parameters

| Related Parameters | Parameter Values |
|---|---|
| Page Size | 16KB |
| Pages per Block | 128 |
| Blocks per Die | 8,192+(88*2) |
| Total Dies | 64 |
| Flash Type | MLC |
| FPGA Chip Model | Xilinx Zynq-7000 AP SoC (XC7Z045-FFG900-3) |
| DRAM Capacity | 1 GB |
| CPU Model | Dual-Core ARM CortexTM-A9 |
| CPU Clock Frequency | Up to 1,000 MHZ |

Since the display information of Cosmos+ FPGA OpenSSD is relatively simple, in order to display the internal parameters of all the above experiments when running on Cosmos+ FPGA OpenSSD, we have written and added related functions to display Cosmos+ FPGA OpenSSD related running time, internal physical block status, and block Information such as page status. The relevant information is displayed on the interface of the compiling host connected to the Cosmos+ FPGA OpenSSD through the serial port.

## 5 Performance Evaluation

In this section, we first describe the experimental setup and test benchmarks. Then, we will conduct comprehensive testing and evaluation of our proposed SUP-GC scheme at the block device layer, file system layer, and application layer.

### 5.1 Experimental Setup

The Cosmos+ FPGA OpenSSD development platform is equipped with HYNIX H27Q1T8YEB9R flash memory chips. These flash memory chips are of MLC NAND type with 16 KB pages and 128 pages per block. The number of effective blocks in each die is 8,192, and there are a total of 64 dies. The flash memory module adopts 8 channels and 8 channels. The microcontroller of Cosmos+ OpenSSD uses Xilinx's ZYNQ-7000 series chip which contains two ARM Cortex-A9 embedded CPUs, and the controller has 1 GB DRAM to store the metadata like FTL mapping table and the buffer cache data.Cosmos+ FPGA OpenSSD uses the PCIe interface with NVMe protocol to connect to the host. The NVMe protocol is version 1.2, and the PCIe interface uses Xilinx 7 series IP cores (PCIe 2.0 version). The detailed parameters of the Cosmos+ OpenSSD platform are shown in Table 1.

The host machine is an Intel Core i7-4790K 4.4 GHz processor with 16 GB DRAM and 256 GB SSD. The operating system is Ubuntu 16.04 based on Linux kernel 4.15 with ext4 file system.

### 5.2 Benchmark

To comprehensively test system performance, we utilized three types of benchmark tests. The Sysbench benchmark is used to test the I/O performance of the SSD at the block device layer, the FIO benchmark is employed to assess performance at the file system layer, and the MySQL database test demonstrates performance at the application layer.

Block Device Layer: Sysbench [33] is an open-source, multi-functional benchmarking tool that provides various testing modules including CPU, memory, disk I/O, threads, and databases. In our system evaluation, we utilized Sysbench to generate an equivalent amount of data to test the disk I/O performance in a non-GC state. This setup allowed us to examine the raw performance of the

SSD without any interference from background GC processes, providing a clear baseline for our system's I/O performance.

File System Layer: FIO (Flexible I/O Tester) [34] is a robust tool capable of simulating a wide range of I/O operations. It is designed to generate multiple threads or processes to perform user-specified I/O tasks, making it ideal for testing both files and storage systems. In our tests, FIO was specifically used to evaluate the impact of each configuration scheme on the random read and write performance of the file system. This benchmark is critical for assessing how well the file system handles high-concurrency I/O operations, which are crucial for real-world application performance.

Application Layer: MySQL [35], a widely used relational database management system, was selected to demonstrate the performance at the application layer. By conducting OLTP tests using Sysbench in conjunction with MySQL, we analyzed the system's capability to handle complex database operations, focusing on transaction speed and query response times. This testing phase is essential for understanding how our system performs under typical operational loads at the upper layers.

### 5.3 Run-time Performance

In this section, we will conduct a comparative analysis of the performance of each strategy based on empirical tests. This includes the Sysbench test on the block device layer, the FIO test on the file system, and the Sysbench combined with MySQL test on the application layer.

At the block device layer, data is typically written sequentially to maximize I/O efficiency and throughput while minimizing seek time. Performance at this layer is primarily measured by throughput (MB/s), which effectively reflects the I/O subsystem's ability to handle sequential accesses. Optimizing throughput at this layer is crucial for enhancing the inherent performance of the underlying storage media. To assess throughput, we used the Sysbench benchmarking tool, which is specifically designed to measure the performance of various storage media under different loads, particularly focusing on I/O throughput and latency. In our tests, Sysbench was used to evaluate the SSD's I/O performance, specifically avoiding the influence of GC to present a clearer picture of the SSD's raw performance.

At the file system level, data layout and structure have a significant impact on performance. Access patterns typically involve a mix of sequential and random access. Therefore, performance evaluation should focus on several key metrics: **Input/Output Operations Per Second** (**IOPS**), which measures the system's ability to handle large-scale read/write requests in a concurrent environment; latency, which reflects the system's response time to operation requests, with lower latency indicating faster responses; and BW, which represents the system's efficiency in handling large volumes of data. In this study, we employed the FIO tool for performance testing. FIO generates a variety of access patterns and provides detailed performance data, including IOPS, latency, and throughput. By analyzing these performance metrics and comparing them with the performance characteristics of SSDs and other storage media, this article evaluates the file system's performance from multiple perspectives across different access patterns and effectively identifies potential performance bottlenecks.

At the application layer, data access patterns become even more complex. Applications may generate highly random workloads, such as those encountered in database transaction processing and query operations, which place considerable pressure on the storage devices. Performance evaluation at this layer primarily focuses on response time and transaction throughput, reflecting the impact of I/O latency on application performance. To more comprehensively evaluate performance at the application layer, we used Sysbench in conjunction with MySQL database for benchmarking, examining the database system's performance under real-world workloads.

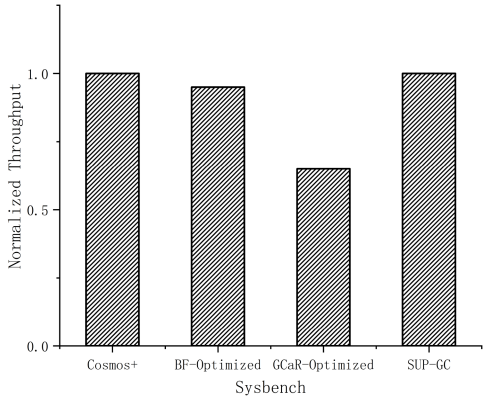
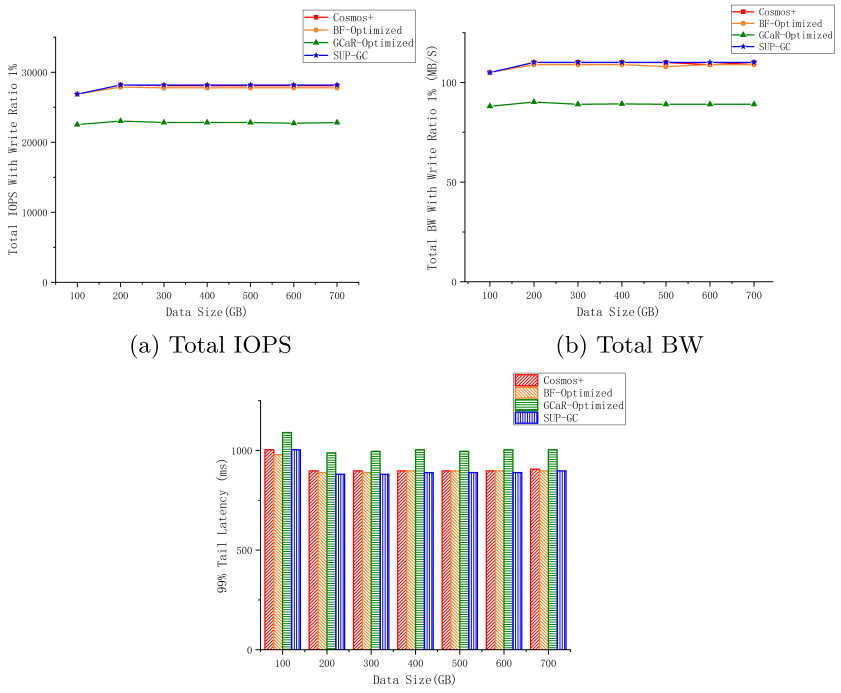



Fig. 13. Normalized throughput.

*5.3.1 Block Device Layer.* In our tests conducted at the block device layer, the primary goal was to assess the actual computational and blocking overheads imposed by various strategies on SSD performance. Utilizing Sysbench's data preparation feature within the Sysbench workload allowed us to bypass the application and file system layers, enabling direct data writing to SSDs through the block device layer. While it is challenging to directly quantify the impact of each strategy's complexity on real SSDs, this methodology effectively demonstrates how the complexity of strategies influences SSD system performance.

Figure 13 depicts the relationship between SSD throughput and the various strategies tested. We used Cosmos+ as the baseline to evaluate the actual performance of each strategy, with results normalized for comparison. As demonstrated in the figure, Cosmos+ employs a notably simple and efficient architecture that does not introduce any additional computational or resource blocking overhead, thereby setting its performance metric at 1. To illustrate the resource overhead associated with the GCaR-Optimized strategy during cache replacement, this strategy was also activated in a non-GC state. Experimental findings reveal that GCaR-Optimized's performance was only 65% of the baseline due to its ongoing requirement to search and traverse the cache during the replacement process, resulting in substantial resource blocking and computational overheads. Remarkably, even minimal cache traversal by GCaR, at only 1%, leads to significant performance degradation, underscoring the critical limitations of such cache replacement strategies in real-world scenarios. In contrast, the BF-Optimized strategy benefits from an inherently lightweight design, resulting in a much lesser impact on performance than GCaR. However, BF-Optimized still exhibits a performance reduction of about 5% compared to the baseline, indicating that even highly optimized, lightweight strategies can negatively affect system performance in actual SSDs. Our newly proposed SUP-GC strategy, which incurs no computational or blocking overheads, matches the performance of the Cosmos+ strategy, achieving a performance index of 1. These results confirm that our strategies do not impose any computational burdens and can fully harness the native performance potential of SSDs in block device layer testing.

*5.3.2 File System Layer.* This section compares the performance of different strategies in real-world file system tests. We use the FIO tool to evaluate the performance of SUP-GC and other strategies. The evaluation includes a 700 GB data volume test to assess SSD performance from an empty to nearly full state. To comprehensively evaluate SSD performance under different read/write ratios, we set five test ratios: 1%, 25%, 50%, 75%, and 99%. The evaluation metrics include IOPS and BW, which directly reflect the throughput performance of SSDs; block erase counts and GC page




(a) Total IOPS

(b) Total BW

(c) Tail Latency

Fig. 14. FIO 1% write ratio test.

move counts, which indicate block utilization efficiency and write amplification performance. The 99% tail latency demonstrates the tail latency performance during the tests. The FIO test configuration is as follows: the IO engine is set to libaio, the access pattern is randrw, the block size is 4 KB, and the number of concurrent jobs is set to 10.

In Figure 14, with the FIO write ratio set to 1%, the entire testing process primarily remains in a read state. Throughout the 700 GB data testing process, none of the schemes enter the GC state, indicating that both the block erase count and the GC page move count remain at zero (experimental data for these values is not shown due to the data volume being zero). Consequently, we observe no performance degradation as the data volume increases from 100 GB to 700 GB. Although this scenario does not reflect the performance of each scheme during GC, it indirectly indicates the impact of algorithmic complexity on the performance of each scheme in a real SSD. Using Cosmos+ as the baseline, we observe that BF-Optimized's average performance is approximately 5% lower than the baseline. This is attributed to the hash computations performed by the Bloom filter during each write operation. Despite the Bloom filter being designed to be lightweight, it still incurs around a 5% performance impact in a real SSD. In this test, GCaR experiences a certain degree of performance decline relative to the baseline. This is because GCaR causes I/O blocking issues during cache traversal, even though we set the GCaR cache traversal value to occupy only a very small portion of the entire cache. This results in a significant performance drop. Our proposed SUP-GC, due to its lack of additional computational overhead, exhibits performance identical to the baseline in this test. The experimental results in this scenario are similar to the conclusions drawn from the Sysbench test in Figure 13.

The next set of tests examines the performance at a higher write ratio of 25%.

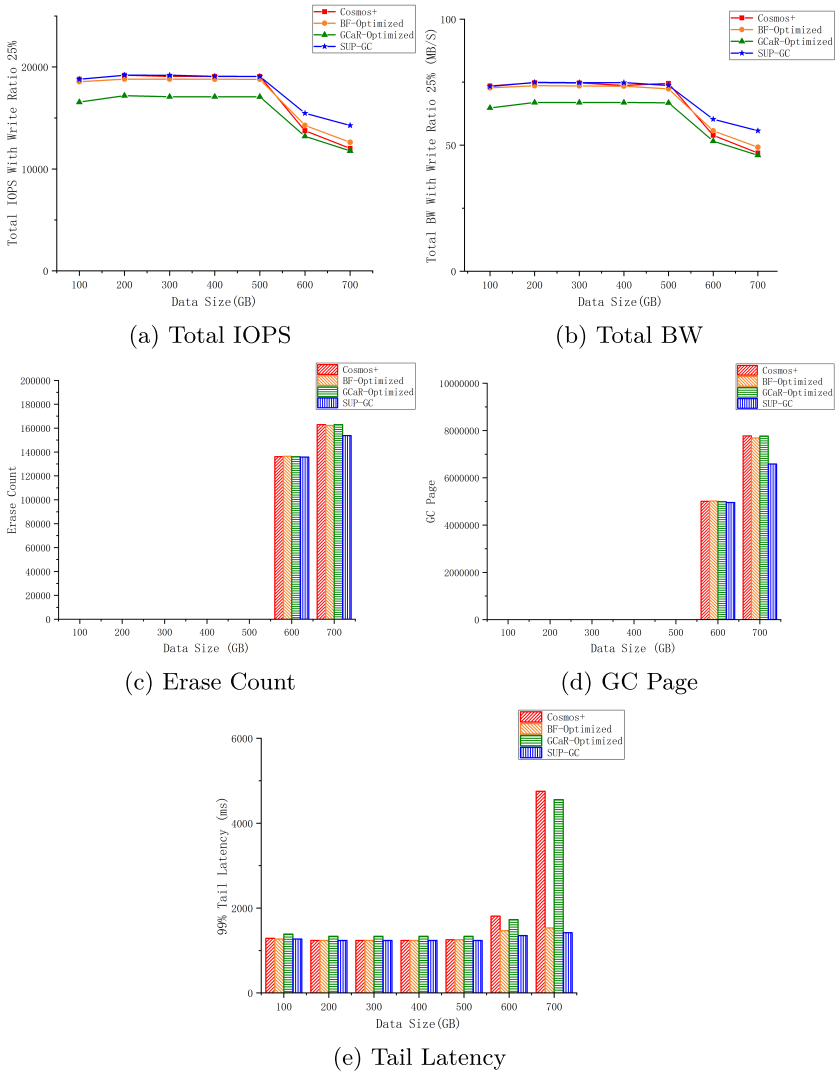



Fig. 15. FIO 25% write ratio test.

In Figure 15, when the FIO write ratio increases to 25%, the entire testing process changes. When the data volume reaches 500 GB, the performance of IOPS and BW starts to decline, accompanied by an increase in block erase counts and GC page move counts. Tail latency also increases with the onset of GC. Therefore, this experimental observation further reinforces the notion that GC is indeed a primary factor influencing SSD performance. As shown in Figure 15, when the data volume reaches 600 GB and 700 GB, all schemes experience varying degrees of performance decline due to the impact of GC. However, SUP-GC outperforms the other comparison schemes in terms of both IOPS and BW. This advantage can be attributed to SUP-GC's data recognition strategy based on natural data attributes and efficient block-level utilization in flash memory. Additionally, BF-Optimized, which employs hot and cold data separation, outperforms Cosmos+ and GCaR. This is because BF-Optimized's data partitioning strategy enhances efficiency during GC, reducing unnecessary flash page migrations and thereby improving SSD I/O performance.

Similarly, constrained by computational complexity, GCaR fails to achieve satisfactory results in this scenario, even performing worse than the baseline. This is because the performance improvement brought by the GCaR strategy itself is insufficient to offset the disadvantages introduced by its performance overhead.

Specifically, when the data reaches 700 GB, SUP-GC performs excellently under medium-to-low write ratios and large data volumes: IOPS is about 15.7% higher than Cosmos+, 11.5% higher than BF-Optimized, and 17,5% higher than GCaR; BW is approximately 15.8% higher than Cosmos+, 11.6% higher than BF-Optimized, and 17.6% higher than GCaR; block erase count is about 5.7% lower than Cosmos+, 5.2% lower than BF-Optimized, and 5.6% lower than GCaR; GC page movement count is about 15.2% lower than Cosmos+, 14.3% lower than BF-Optimized, and 15.2% lower than GCaR; 99% tail latency is about 70.2% lower than Cosmos+, 7.4% lower than BF-Optimized, and 68.9% lower than GCaR.

As we increase the write ratio to 50%, we observe more pronounced differences in performance between the schemes.

In Figure 16, as the write ratio increases to 50%, GC is triggered more frequently and earlier. At 300 GB, the block erase count and GC page movement begin to rise (as shown in Figure 16(c) and (d)), while IOPS and BW (as shown in Figure 16(a) and (b)) start to decline, and tail latency in Figure 16(e) sharply increases. The trends observed in these metrics are consistent with those in Figure 15. However, as the write ratio increases and GC is triggered more frequently, the performance differences between the schemes become more pronounced. Notably, towards the end of the 700 GB data load, the performance of the baseline scheme Cosmos+ significantly deteriorates due to insufficient SSD space for GC data migration, leading to a substantial increase in block erase counts. In contrast, SUP-GC, by efficiently allocating data in blocks based on their natural properties, shows satisfactory performance even under frequent GC triggers.

Specifically, when the data reaches 700 GB, SUP-GC performs excellently under medium-to-high write ratios and large data volumes: IOPS is about 35.1% higher than Cosmos+, 10.1% higher than BF-Optimized, and 35.2% higher than GCaR; BW is approximately 35.3% higher than Cosmos+, 10.2% higher than BF-Optimized, and 35.3% higher than GCaR; block erase count is about 22.1% lower than Cosmos+, 2% lower than BF-Optimized, and 22% lower than GCaR; GC page movement count is about 44.7% lower than Cosmos+, 5.6% lower than BF-Optimized, and 44.6% lower than GCaR; 99% tail latency is about 87.2% lower than Cosmos+, 68.3% lower than BF-Optimized, and 87.2% lower than GCaR.

In summary, SUP-GC, under medium-to-high write ratios and large data volumes, significantly enhances the overall performance of SSDs through its simplified design principles and effective block-level utilization, demonstrating excellent performance and efficiency. The advantage of SUP-GC lies in its lack of additional computational overhead, relying on the natural behavior of data for classification, and efficiently allocating data in blocks, simplifying the GC process. This significantly reduces block erasures and data migrations, greatly lowering write amplification, and improving system responsiveness and efficiency. In comparison, BF-Optimized improves efficiency during GC through a hot and cold data partitioning strategy, but still falls short of SUP-GC, though it outperforms Cosmos+ and GCaR. GCaR, constrained by its high computational complexity, performs poorly under high data volume and medium-to-high write ratios, even lagging behind the baseline Cosmos+. These results further validate the effectiveness and advantages of SUP-GC in enhancing SSD performance.

When the write ratio is further increased to 75%, the system faces an even more write-intensive workload, providing insights into the schemes' behavior under heavy load.

In Figure 17, when the number of write operations surpasses the read operations, reaching a write ratio of 75%, the entire testing system enters a write-intensive workload scenario.

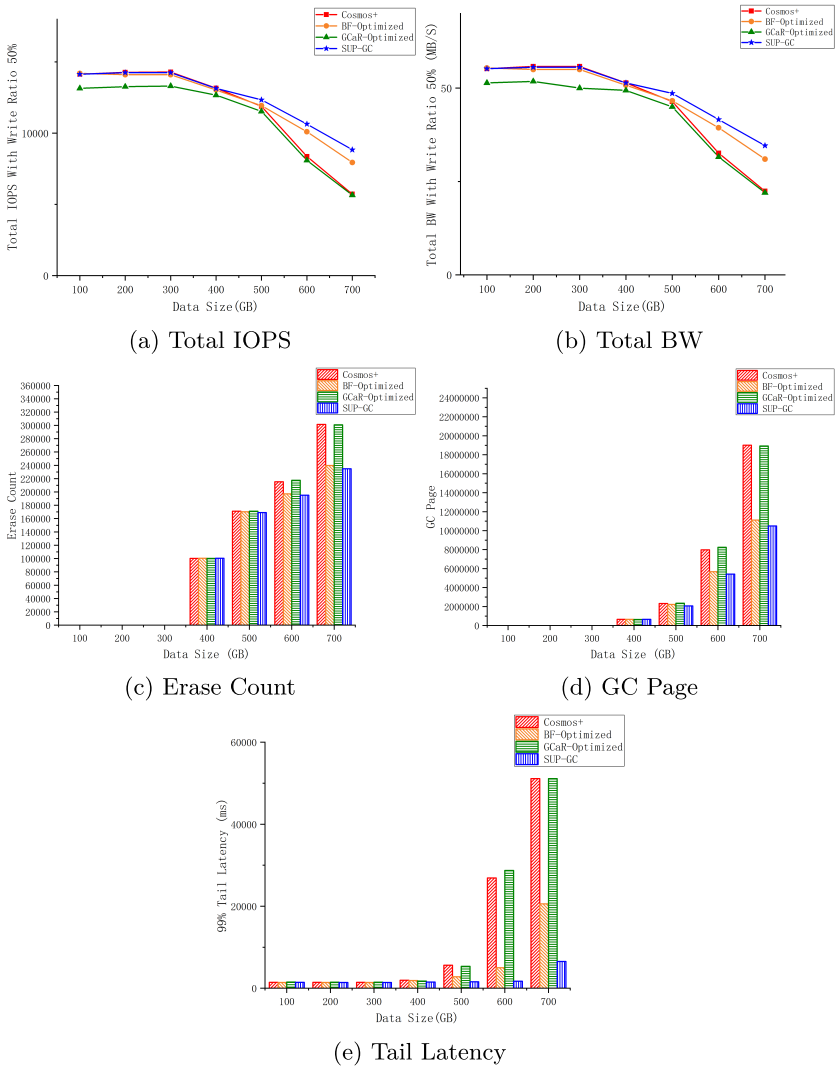



Fig. 16. FIO 50% write ratio test.

From Figure 17(c) and (d), it is evident that when the data volume reaches 200 GB, both the block erase count and GC page move count start to rise, accompanied by a decline in IOPS and BW. This indicates the initiation of GC in the SSD. In this scenario, due to the frequent occurrence of GC triggers, all schemes experience a substantial decline in performance during GC phases. However, due to the adoption of hot and cold data partitioning strategies, the performance degradation of SUP-GC and BF-Optimized is less noticeable compared to Cosmos+ and GCaR.

Specifically, when the data reaches 700 GB, SUP-GC performs excellently under medium-to-high write ratios and large data volumes: IOPS is about 73.5% higher than Cosmos+, 72.6% higher than BF-Optimized, and 75.6% higher than GCaR; BW is approximately 73.6% higher than Cosmos+, 72.6% higher than BF-Optimized, and 75.8% higher than GCaR; block erase count is about 35% lower than Cosmos+, 32.7% lower than BF-Optimized, and 35.3% lower than GCaR; GC page movement

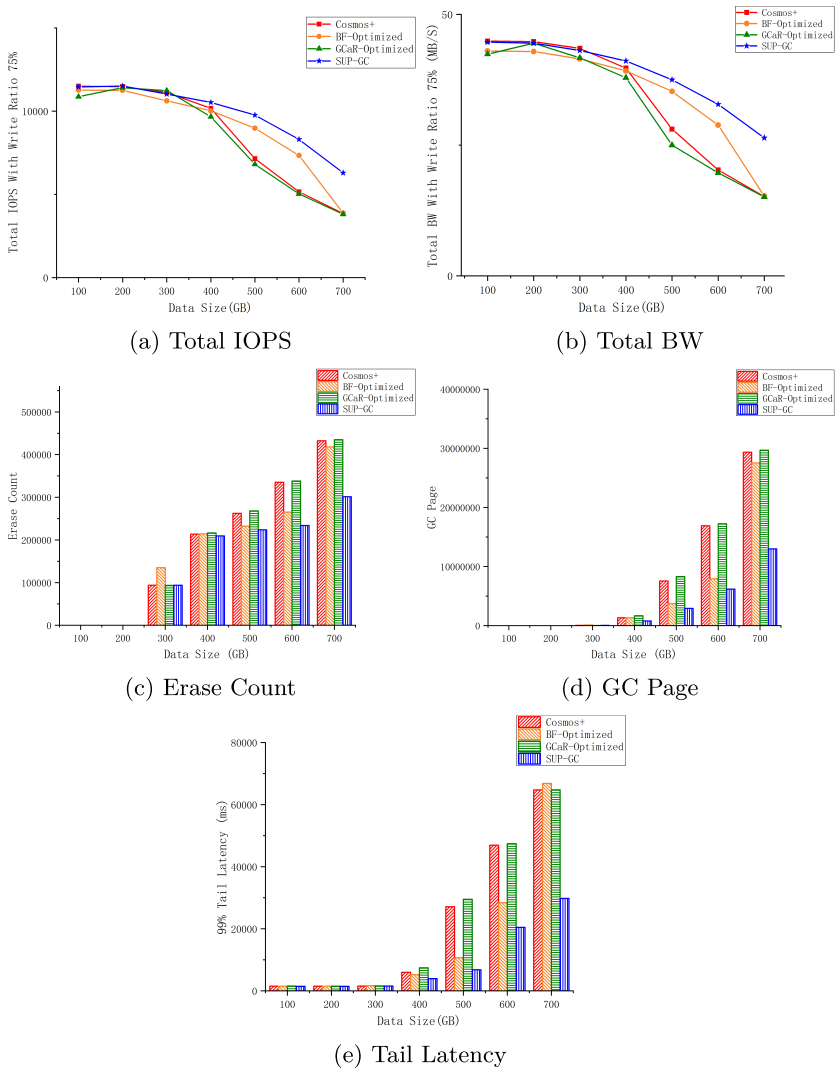



Fig. 17. FIO 75% write ratio test.

count is about 65.9% lower than Cosmos+, 63.7% lower than BF-Optimized, and 66.1% lower than GCaR; 99% tail latency is about 87.2% lower than Cosmos+, 68.3% lower than BF-Optimized, and 87.2% lower than GCaR.

In the tests with a data volume of 700 GB, SUP-GC significantly outperforms other methods, achieving substantial performance improvements compared to Cosmos+, GCaR, and BF-Optimized, and also significantly reducing block erase counts. Additionally, at data volumes of 500 GB and 600 GB, BF-Optimized performs better than Cosmos+ and GCaR. However, when the data volume reaches 700 GB, the performance of BF-Optimized sharply declines, slightly falling below Cosmos+ and GCaR. This counterintuitive result may be due to BF-Optimized's inability to effectively distinguish between hot and cold data states under extremely high GC scenarios, leading to ineffective partitioning and isolation of hot and cold data, and thus failing to achieve the expected performance gains.

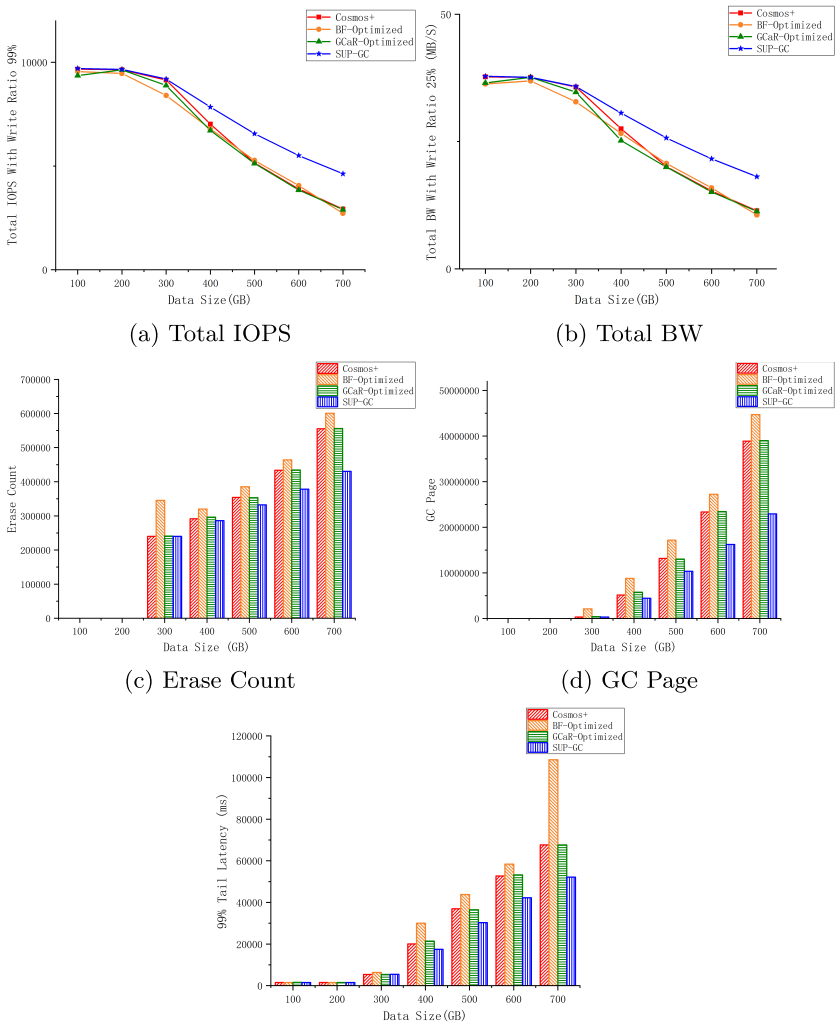



Fig. 18. FIO 99% write ratio test.

Regarding write amplification, as indicated by GC page movements in Figure 17(d), SUP-GC shows a significantly lower GC page move count, indicating reduced write amplification compared to the other methods. This lower write amplification contributes to better overall performance and longevity of the SSD.

Moreover, Figure 17(e) shows the 99% tail latency for different schemes. At 700 GB, SUP-GC demonstrates significantly lower tail latency compared to Cosmos+, BF-Optimized, and GCaR-Optimized. This indicates that SUP-GC not only maintains higher overall performance but also ensures better responsiveness under heavy write-intensive workloads.

Finally, we test the performance under an extreme write ratio of 99%, pushing the system to its limits.

In Figure 18, when the write ratio reaches 99%, indicating a substantial amount of write operations along with extensive GC activities, all strategies experience severe performance degradation

in this extremely write-intensive scenario. However, even under such extreme conditions, SUP-GC demonstrates commendable performance. In the final test with a data volume of 700 GB, our proposed SUP-GC significantly outperforms other comparative methods, exhibiting a performance advantage of 57.9%, 69.9%, and 59.8% over Cosmos+, BF-Optimized, and GCaR-Optimized, respectively. Additionally, the block erase count is also notably reduced by 22.5%, 28.3%, and 22.6%, respectively. In this scenario with an exceptionally high write ratio, the IOPS and BW performance of BF-Optimized generally fall short of Cosmos+ and GCaR. Simultaneously, we observe in Figure 18(c) and (d) that the block erase count and GC page move count of BF-Optimized are consistently higher than those of Cosmos+ and GCaR. This phenomenon corroborates our speculation from the analysis in Figure 17, suggesting that in extremely high GC scenarios, BF-Optimized struggles to effectively distinguish between hot and cold data states, leading to inadequate partitioning and isolation of hot and cold data and consequently failing to achieve the expected performance improvement.

Regarding write amplification, as indicated by GC page movements in Figure 18(d), SUP-GC shows a significantly lower GC page move count, indicating reduced write amplification compared to the other methods. This reduced write amplification contributes to the better overall performance and longevity of the SSD.

Moreover, Figure 18(e) shows the 99% tail latency for different schemes. Similarly, SUP-GC demonstrates significantly lower tail latency compared to Cosmos+, BF-Optimized, and GCaR. This indicates that SUP-GC not only maintains higher overall performance but also ensures better responsiveness under heavy write-intensive workloads.

Overall, the SUP-GC scheme demonstrated significant performance improvements in file system layer tests. Through a series of evaluations using the FIO tool under different read/write ratios, we observed that SUP-GC consistently outperformed BF-Optimized, GCaR, and the baseline Cosmos+ in multiple key performance metrics. SUP-GC efficiently handled a 700 GB data volume without introducing additional computational overhead. By treating newly written data as hot and efficiently managing cold data during GC, SUP-GC reduced block erasures and GC page movements, thereby lowering write amplification. This not only improved throughput performance (higher IOPS and BW) but also ensured lower 99% tail latency, enhancing system responsiveness. SUP-GC's strategy, based on natural data behavior, simplified data processing by eliminating the need for complex real-time data state analysis, reducing the computational burden on the SSD microcontroller. Consequently, SUP-GC maintained superior performance even under high write ratios and heavy workloads, whereas other strategies underperformed in these scenarios.

*5.3.3 Application Layer.* This section compares the performance of different strategies in real-world application-level database system tests. We use Sysbench combined with MySQL to evaluate the performance of SUP-GC and other strategies. To comprehensively test database performance, we preheat with 1 TB of data and then conduct comprehensive tests with 100 tables, each containing 9,000,000 records, in both only write and mixed read/write scenarios. Evaluation metrics include transactions and queries, which directly reflect the transaction and query processing capabilities of the database running on SSDs; latency, which reflects the average read/write performance of the database; block erase counts and GC page move counts, which indicate block utilization efficiency and write amplification performance; and 95% tail latency, which demonstrates the tail latency performance of the database running on SSDs during the tests.

Figure 19 illustrates the performance of the database under only write conditions when combining Sysbench and MySQL. In Figure 19(a), the OpenSSD with the SUP-GC scheme shows an increase in database transaction processing capacity by 54.1%, 32.1%, and 52.4% compared to

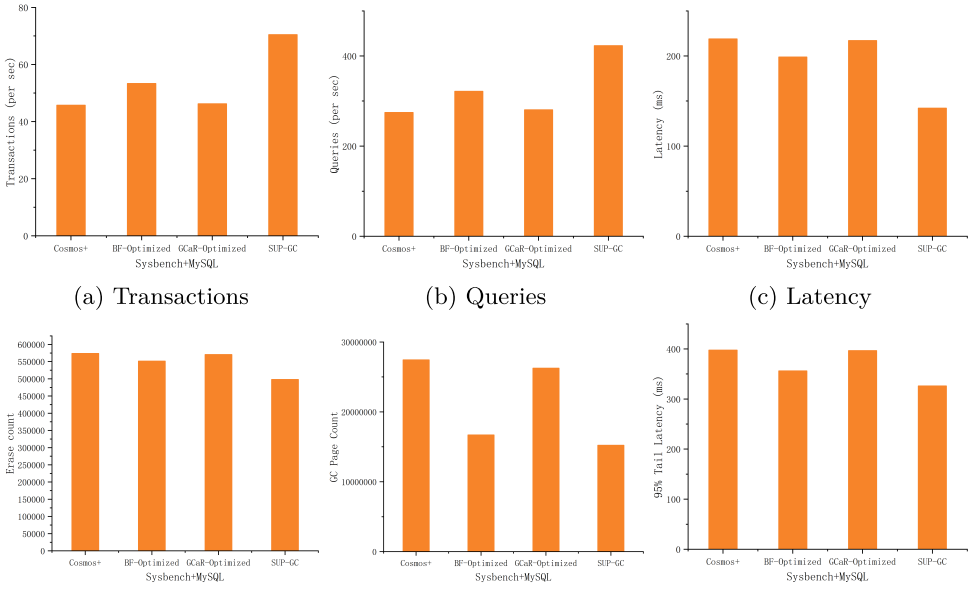



Fig. 19. Sysbench+MySQL.

other OpenSSD schemes. Similarly, in Figure 19(b), SUP-GC demonstrates the same advantage in database query capability. The number of transactions and queries executed per second directly reflects the database's performance, and OpenSSD based on the SUP-GC scheme achieves the best results in both aspects. This phenomenon occurs because data migration during GC can severely affect the normal read and write operations of the database. Moreover, SUP-GC effectively reduces the overhead of migrating invalid data during GC by adopting a data distribution method that perfectly matches natural data behavior, thereby minimizing the impact of GC data migration on normal data writes.

Although the BF-Optimized scheme uses a hot and cold data partitioning strategy, this attempt to precisely partition the write data not only introduces additional computational overhead but also cannot achieve perfect partitioning. Additionally, any change in workload characteristics requires parameter readjustment (in the database tests, we readjusted the parameters used in FIO to better suit the database for BF-Optimized to perform as expected), which severely restricts its application prospects in real-world scenarios. Furthermore, the OpenSSD based on GCaR-Optimized did not achieve the expected advantages in this test. Although GCaR-Optimized considers the impact of data distribution in the cache on GC performance, its performance improvement is limited, and the computational overhead and data blocking caused by cache traversal also severely hinder its performance.

In Figure 19(c), the OpenSSD based on the SUP-GC scheme also achieves the best results in average database latency compared to other schemes. It reduces the latency by 35.1%, 28.5%, and 34.6% compared to Cosmos+, BF-Optimized, and GCaR-Optimized SSDs, respectively. In Figure 19(d), due to the perfect allocation of hot and cold data, SUP-GC reduces the number of block erasures by 13.2%, 9.7%, and 12.7% compared to other schemes. Figure 19(e) reflects the number of page migrations related to GC within OpenSSD. This metric effectively reflects the write amplification performance of each algorithm. Compared to Cosmos+, BF-Optimized, and GCaR-Optimized, SUP-GC reduces write amplification by 44.5%, 32.6%, and 42.1%, respectively. Figure 19(f) shows the tail

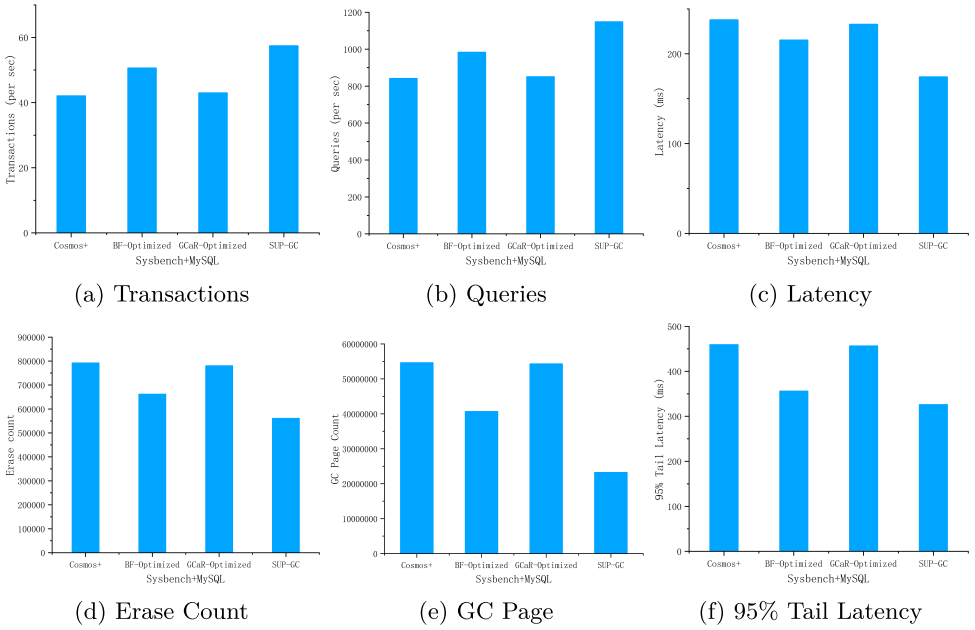



Fig. 20. Sysbench+MySQL.

latency performance among the strategies for the 95th percentile tail latency. Similarly, SUP-GC achieves the best performance in this tail latency metric as well.

Figure 20 illustrates the database performance under read-write mixed conditions when combining Sysbench and MySQL. Similar to the performance observed in Figure 19, in Figure 20(a), the OpenSSD using the SUP-GC scheme shows an improvement of 36.5%, 13.5%, and 33.6% in database transaction processing capacity compared to other OpenSSD schemes. Likewise, in Figure 20(b), SUP-GC demonstrates the same advantages in database query capacity. It is evident that in read-write mixed scenarios, SUP-GC effectively reduces the overhead of migrating invalid data during GC by adopting a data distribution method that perfectly matches the natural data behavior, thereby minimizing the impact of GC data migration on normal data read and write operations.

The BF-Optimized scheme attempts to optimize performance by partitioning hot and cold data, but under read-write mixed conditions, the additional computational overhead and the problem of imperfect partitioning introduced by this method still exist. Similarly, the GCaR-Optimized scheme did not show the expected advantages in this test. Although this scheme considers the impact of data distribution in the cache on GC performance, its performance improvement in read-write mixed scenarios is limited, and the computational overhead and data blocking issues caused by cache traversal still exist.

In Figure 20(c), the OpenSSD using the SUP-GC scheme continues to perform excellently in terms of average database latency. Compared to Cosmos+, BF-Optimized, and GCaR-Optimized SSDs, SUP-GC reduces latency by 26.7%, 19.1%, and 26.5%, respectively. Figure 20(d) shows that due to the reasonable allocation of hot and cold data, the SUP-GC scheme also reduces the number of block erasures by 29.2%, 15.2%, and 28.1% compared to the other schemes. Figure 20(e) shows that under read-write mixed conditions, SUP-GC continues to perform best in terms of write amplification. Compared to Cosmos+, BF-Optimized, and GCaR-Optimized, SUP-GC reduces page migration by 57.5%, 42.9%, and 57.2%, respectively, significantly reducing the write amplification effect. Figure 20(f) demonstrates the performance of each strategy in terms of 95th percentile

tail latency, with the SUP-GC scheme also performing best in this metric, further validating its excellent performance in read-write mixed scenarios.

Overall, the SUP-GC scheme demonstrated significant performance improvements in application-level database system tests. Through a series of evaluations using Sysbench combined with MySQL, we observed that SUP-GC consistently outperformed BF-Optimized, GCaR, and the baseline Cosmos+ in multiple key performance metrics. SUP-GC efficiently handled large data volumes without introducing additional computational overhead. By treating newly written data as hot and efficiently managing cold data during GC, SUP-GC reduced block erasures and GC page movements, thereby lowering write amplification. This not only improved throughput performance (more transactions and queries) but also ensured lower 95% tail latency, enhancing system responsiveness. SUP-GC's strategy, based on natural data behavior, simplified data processing by eliminating the need for complex real-time data state analysis, reducing the computational burden on the SSD microcontroller. Consequently, SUP-GC maintained superior performance even under high write ratios and mixed workload.

## 6 Related Work

In this section, we introduce works related to SUP-GC.

In SSD research, simulators are commonly used to evaluate their designs. Popular simulators include DISKSim [36], SSDSim [39], WiscSim [38], FlashSim [37], and MQSim [3]. While these simulators can emulate the state of real SSDs to a certain extent, they still exhibit discrepancies when compared to actual SSDs.

To address the limitations of simulators, some researchers have turned to using real SSDs for their studies. The first open-source SSD platform, Jasmine OpenSSD [40], employs a commercial SOC as the storage controller and allows users to freely modify its firmware. Although Jasmine OpenSSD has many advantages, its strategies are outdated and do not support the latest high-performance SSD protocols. Recently, researchers have developed the first open-source SSD that supports the NVMe protocol, named Cosmos+. Cosmos+ OpenSSD [32] not only permits the free modification of its software and hardware components but also makes its source code publicly available. This advancement positions Cosmos+ OpenSSD as the most advanced real SSD device currently available, significantly advancing SSD research.

The identification and classification of hot and cold data have always been key challenges in SSD research. To effectively identify hot data in flash memory, Chang et al. [31] adopted a two-level LRU linked list to distinguish between hot and cold data. PGIS [26] utilizes a lightweight linked list to identify and classify hot and cold data, isolating them at the block level. MHF [29] employs multiple hash functions and a Bloom filter to identify hot and cold data, which results in low computational and memory overhead, making it well-suited for use in flash memory. To address the issue of recency information in MHF, MBF [30] uses multiple Bloom filters to capture finer-grained recency information. Although these schemes achieve a certain level of performance in terms of identification accuracy, they still face unresolved issues related to computational and spatial overhead. Additionally, they cannot guarantee consistent accuracy across all scenarios. Furthermore, these schemes have been validated through simulators, lacking verification on actual SSD devices, which makes their real-world performance evaluation insufficient. Li et al. [42] and Yang et al. [43] both leveraged machine learning techniques to predict data heat and manage SSD performance more effectively. Li et al. utilized historical access patterns to improve prediction accuracy and adaptability, while Yang et al. aimed at reducing GC overhead based on workload prediction. However, both methods suffer from high implementation complexity and significant computational overhead, which limit their practicality in resource-constrained environments.

In addition to the aforementioned strategies that improve GC efficiency through hot and cold data classification, there are other methods that optimize GC from different angles. GCaR [17] considers the impact of data distribution in the cache on GC and gives higher priority to cached data blocks belonging to flash memory chips in the GC state to reduce conflicts between user input I/O operations and GC-induced I/O operations. However, GCaR uses a sequential method to search for GC-related data in the cache, which adds extra time overhead and leads to performance degradation. PaGC [16] utilizes plane-level parallelism to improve GC efficiency, but PaGC is only applicable to SSDs that can take advantage of plane-level parallelism, making its application range narrower than that of block-level parallelism. Kim et al. [48] and Wu [49] both proposed suspension techniques to optimize the timing of GC operations, aiming to minimize their impact on I/O performance. Kim et al. specifically focused on suspending erase operations in modern low-latency SSDs, while Wu extended this approach by suspending both program and erase operations to further reduce read latency. These techniques mitigate GC interference by strategically managing timing but do not fundamentally enhance the efficiency of GC itself. In scenarios with intensive write workloads and limited idle time, their effectiveness is reduced, which may degrade overall system performance. FaGC+ [12] considers the impact of block-level parallelism on GC and implements four types of hot and cold block classifications. However, the relocation of each page in FaGC+ is overly cumbersome, which can hinder the execution of normal I/O operations in real-world scenarios, leading to excessive blocking issues. TTFLASH [14] reduces GC-induced tail latencies by utilizing intra-plane copy operations. This approach accelerates the copying of pages during the GC process by skipping error correction code checks, thereby minimizing the impact of GC on user input/output operations. However, the TTFLASH strategy has significant disadvantages due to its heavy reliance on intra-plane copy operations, which are not supported by many existing hardware platforms. Furthermore, the implementation and validation of TTFLASH on actual hardware face numerous challenges, limiting its potential for performance improvements and possibly compromising data reliability. These limitations restrict the broad applicability of the TTFLASH strategy. RLGC [19], utilizing dynamic state management and reinforcement learning-assisted garbage collection, aims at reducing SSD long-tail latency. However, this method has drawbacks: increased computational overhead, the need for tuning Q-table cache size, and it may not always outperform alternative methods. Implementing this technique in SSD firmware could face challenges due to limitations in memory and code size. FIOS [41] method reduces write amplification, improving SSD performance and durability by optimizing data organization. It lowers internal operation overhead, increases device BW, and enhances workload throughput. However, its implementation is complex and resource-intensive, requiring feature extraction, matrix construction, and multithreaded K-means clustering. Different workload features affect WAF differently, making it challenging to select the optimal feature combination. The method's high computational overhead and its applicability only to multi-stream SSDs further limit its practicality.

## 7 Conclusion

This article introduced SUP-GC, an innovative and potentially disruptive GC scheme for SSDs. By categorizing all newly written data as hot and simplifying the data processing workflow, while storing valid pages as cold data in cold blocks during the GC process, SUP-GC eliminates the need for complex real-time data state analysis, thereby reducing computational overhead and significantly improving SSD efficiency and responsiveness. Implemented and validated on the Cosmos+ FPGA OpenSSD platform, experimental results show that SUP-GC significantly enhances read and write response speeds during GC. This approach not only streamlines data management, reduces system overhead, and extends the lifespan of SSDs, but also demonstrates practical feasibility and efficiency across various workloads. By effectively utilizing existing hardware resources, SUP-GC

achieves efficient GC while minimizing system resource usage. This not only highlights the theoretical advantages of the scheme but also proves its significant effectiveness in practical applications. We expect that SUP-GC will bring groundbreaking advancements in GC management, making significant contributions to the development of SSD technology. Looking ahead, since our current evaluation has not yet been conducted on commercial SSDs, we plan to apply SUP-GC to commercial SSD products in the future to further validate its effectiveness in real-world environments.

## References


[1] Chin-Hsien Wu, Liang-Ting Chen, Ren-Jhen Hsu, and Jian-Yu Dai. 2023. A state-aware method for flows with fairness on NVMe SSDs with load balance. *IEEE Transactions on Cloud Computing* 11, 3 (2023), 3040–3054.

[2] Hui Sun, Shangshang Dai, Jianzhong Huang, and Xiao Qin. 2021. Co-active: A workload-aware collaborative cache management scheme for NVMe SSDs. *IEEE Transactions on Parallel and Distributed Systems* 32, 6 (2021), 1437–1451.

[3] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A framework for enabling realistic studies of modern multi-queueSSD devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*. USENIX Association, 49–66.

[4] Haodong Lin, Jun Li, Zhibing Sha, Zhigang Cai, Yuanquan Shi, and Balazs Gerofi. 2022. Adaptive management with request granularity for DRAM cache inside NAND-based SSDs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 8 (2022), 2475–2487.

[5] M. Bjørling, A. Aghayev, H. Holmberg, A. Ramesh, D. Le Moal, G. R. Ganger, and G. Amvrosiadis. 2021. ZNS: Avoiding the block interface tax for flash-based SSDs. In *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC'21)*. USENIX Association, 689–703.

[6] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir. 2014. HIOS: A host interface I/O scheduler for solid state disks. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 289–300.

[7] R. Liu, Z. Tan, Y. Shen, L. Long, and D. Liu. 2024. Fair-zns: Enhancing fairness in zns ssds through self-balancing I/O scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 7 (2024), 2012–2022.

[8] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. 2008. Design tradeoffs for SSD performance. In *Proceedings of the 2008 USENIX Annual Technical Conference (USENIX ATC'08)*.

[9] J. Li, X. Xu, Z. Cai, J. Liao, K. Li, B. Gerofi, and Y. Ishikawa. 2022. Pattern-based prefetching with adaptive cache management inside of solid-state drives. *ACM Transactions on Storage* 18, 1 (2022), 1–25.

[10] H. Lin, J. Li, Z. Sha, Z. Cai, Y. Shi, and B. Gerofi. 2023. Adaptive management with request granularity for DRAM cache inside NAND-based SSDs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 8 (2023), 2475–2487.

[11] Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, and Yaohua Wang. 2018. FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives. In *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 397–410.

[12] Hua Yan, Yong Huang, Xinzhi Zhou, and Yinjie Lei. 2018. An efficient and non-time-sensitive file-aware garbage collection algorithm for NAND flash-based consumer electronics. *IEEE Transactions on Consumer Electronics* 65, 1 (2018), 73–79.

[13] Y. Pan, M. Lin, Z. Wu, H. Zhang, and Z. Xu. 2021. Caching-aware garbage collection to improve performance and lifetime for NAND flash SSDs. *IEEE Transactions on Consumer Electronics* 67, 2 (2021), 141–148.

[14] S. Yan, H. Li, M. Hao, M. H. Tong, S. Sundararaman, A. A. Chien, and H. S. Gunawi. 2017. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. *ACM Transactions on Storage* 13, 3 (2017), 1–26.

[15] Y. Kim, Sarp Oral, Galen M. Shipman, Junghee Lee, David A. Dillow, and Feiyi Wang. 2011. Harmonia: A globally coordinated garbage collector for arrays of solid-state drives. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*. 1–12.

[16] Narges Shahidi, Mahmut T. Kandemir, Mohammad Arjomand, Chita R. Das, Myoungsoo Jung, and Anand Sivasubramaniam. 2016. Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'16)*. 561–572.

[17] S. Wu, B. Mao, Y. Lin, and H. Jiang. 2017. Improving performance for flash-based storage systems through GC-aware cache management. *IEEE Transactions on Parallel and Distributed Systems* 28, 10 (2017), 2852–2865.

[18] Sungjin Lee, Dongkun Shin, and Jihong Kim. 2012. BAGC: Buffer-aware garbage collection for flash-based storage systems. *IEEE Transactions on Computers* 62, 11 (2012), 2141–2154.

[19] Wonkyung Kang and Yoo Sungjoo. 2018. Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD. In *Proceedings of the 55th Annual Design Automation Conference (DAC'18)*. 1–6.

[20] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. *ACM SIGPLAN Notices* 44, 3 (2009), 229–240.

[21] H. Kim and S. Ahn. 2008. BPLRU: A buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. USENIX Association, 1–14.

[22] Dongzhe Ma, Jianhua Feng, and Guoliang Li. 2011. LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (SIGMOD'11)*. 1–12.

[23] Yudong Yang, Misra Vishal, and Dan Rubenstein. 2015. On the optimality of greedy garbage collection for ssds. *ACM SIGMETRICS Performance Evaluation Review* 43, 2 (2015), 63–65.

[24] Van Houdt and Benny. 2013. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. *ACM SIGMETRICS Performance Evaluation Review* 41, 1 (2013), 191–202.

[25] Y. Li, P. P. Lee, and J. C. Lui. 2013. Stochastic modeling of large-scale solid-state storage systems: Analysis, design tradeoffs and optimization. In *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*. 179–190.

[26] J. Guo, Y. Hu, B. Mao, and S. Wu. 2017. Parallelism and garbage collection aware I/O scheduler with improved SSD performance. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'17)*. IEEE Computer Society, 1184–1193.

[27] Y. Hu, H. Jinag, D. Feng, L. Tian, H. Luo, and C. Ren. 2012. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE Transactions on Computers* 62, 6 (2012), 1141–1155.

[28] Feng Chen, Binbing Hou, and Rubao Lee. 2016. Internal parallelism of flash memory-based solid-state drives. *ACM Transactions on Storage* 12, 3 (2016), 1–39.

[29] Jen-Wei Hsieh, Tei-Wei Kuo, and Li-Pin Chang. 2006. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage* 2, 1 (2006), 22–40.

[30] Dongchul Park and David H. C. Du. 2011. Hot data identification for flash-based storage systems using multiple bloom filters. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–11.

[31] Li-Pin Chang and Tei-Wei Kuo. 2002. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 187–196.

[32] J. Kwak, S. Lee, K. Park, J. Jeong, and Y. H. Song. 2020. Cosmos+ OpenSSD: Rapid prototype for flash storage systems. *ACM Transactions on Storage* 16, 3 (2020), 1–35.

[33] A. Kopytov. 2004. Sysbench: A system performance benchmark. [Online]. Available: http://sysbench.sourceforge.net/. [Accessed: Jan. 31, 2025].

[34] J. Axboe. FIO (Flexible IO Tester). [Online]. Available: http://git.kernel.dk/?p=fio.git;a=summary. [Accessed: Jan. 31, 2025].

[35] Oracle Corporation. 2024. MySQL Community Edition (Version 8.0.36). https://dev.mysql.com/. Accessed: 2025-01-31.

[36] J. K. Park, D. H. Suh, and S. Baek. 2016. A study of divided disk cache performance using DiskSim. In *Proceedings of the International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT'16)*. ACM, New York, NY, USA, 1867–1872.

[37] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar. 2009. Flashsim: A simulator for nand flash-based solid-state drives. In *Proceedings of the 2009 1st International Conference on Advances in System Simulation*. IEEE, 125–131.

[38] J. He, S. Kannan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. 2017. The unwritten contract of solid state drives. In *Proceedings of the 12th European Conference on Computer Systems (EuroSys'17)*. 127–144.

[39] E. Lee, J. Kim, H. Bahn, S. Lee, and S. H. Noh. 2017. Reducing write amplification of flash storage through cooperative data management with NVM. *ACM Transactions on Storage* 13, 2 (2017), 1–13.

[40] D. Brown, T. O. Walker III, J. A. Blanco, R. W. Ives, H. T. Ngo, J. Shey, and R. Rakvic. 2021. Detecting firmware modification on solid state drives via current draw analysis. *Computers and Security* 102 (2021), 102149.

[41] J. Bhimani, Z. Yang, J. Yang, A. Maruf, N. Mi, R. Pandurangan, and V. Balakrishnan. 2022. Automatic stream identification to improve flash endurance in data centers. *ACM Transactions on Storage* 18, 2 (2022), 1–29.

[42] B. Li, C. Deng, J. Yang, D. Lilja, B. Yuan, and D. Du. 2019. Haml-ssd: A hardware accelerated hotness-aware machine learning based ssd management. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD'19)*. 1–8.

[43] P. Yang, N. Xue, Y. Zhang, Y. Zhou, L. Sun, W. Chen, and K. Kwon. 2019. Reducing garbage collection overhead in SSD based on workload prediction. In *Proceedings of the11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*.

[44] D. Purandare, P. Wilcox, H. Litz, and S. Finkelstein. 2022. Append is near: Log-based data management on ZNS SSDs. In *Proceedings of the 12th Annual Conference on Innovative Data Systems Research (CIDR'22)*.

[45] H. R. Lee, C. G. Lee, S. Lee, and Y. Kim. 2022. Compaction-aware zone allocation for LSM based key-value store on ZNS SSDs. In *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'22)*. 93–99.

[46] Matias Bjørling, Javier Gonzalez, and Philippe Bonnet. 2017. LightNVM: The linux open-channelSSD subsystem. In *Proceedings of the15th USENIX Conference on File and Storage Technologies (FAST'17)*. 359–374.

[47] X. Zhang, F. Zhu, S. Li, K. Wang, W. Xu, and D. Xu. 2021. Optimizing performance for open-channel ssds in cloud storage system. In *Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 902–911.

[48] S. Kim, J. Bae, H. Jang, W. Jin, J. Gong, S. Lee, and J. W. Lee. 2019. Practical erase suspension for modern low-latency SSDs. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC'19)*. 813–820.

[49] Guanying Wu and Xubin He. 2012. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. 10.

[50] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim, and J. Lee. 2006. CFLRU: A replacement algorithm for flash memory. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'06)*. 234–241.

[51] P. Jin, Y. Ou, T. Härder, and Z. Li. 2012. AD-LRU: An efficient buffer replacement algorithm for flash-based databases. *Data and Knowledge Engineering* 72 (2012), 83–102. DOI: 10.1016/j.datak.2011.09

[52] J. Gu, C. Wu, and J. Li. 2017. Hotis: A hot data identification scheme to optimize garbage collection of ssds. In *Proceedings of the 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE, 331–337.

Received 26 May 2024; revised 11 February 2025; accepted 17 February 2025