Concurrency and Computation: Practice and Experience

WILEY

RESEARCH ARTICLE

Innovative Bloom Filter-Based Multilevel Block Classification: A Practical Approach to Enhancing SSD Performance and Endurance

Keyu Wang[1] | Huailiang Tan[1] | Zaihong He[1] | Jinyou Li[1] | Keqin Li[2]

[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha, China | [2]Department of Computer Science, State University of New York, New Paltz, NY, USA

Correspondence: Zaihong He (hezaihong88@hnu.edu.cn)

Received: 16 January 2025 | **Revised:** 16 April 2025 | **Accepted:** 21 May 2025

Funding: The authors received no specific funding for this work.

Keywords: bloom filter | cold and hot data division | flash translation layer | garbage collection | solid-state drive

ABSTRACT

In modern solid-state drives (SSDs), managing hot and cold data is key to improving performance and extending lifespan. However, previous Bloom filter-based methods were often complex and lacked solid empirical validation under high-performance conditions. To address this, we propose a more efficient multilevel Bloom filter classification strategy tailored to SSDs. Our approach optimizes classification accuracy while reducing computational and storage overhead through careful parameter tuning. By utilizing SSD block-level parallelism to group similar data access patterns, we enhance garbage collection efficiency and extend block life. Unlike previous studies relying on simulations, we validate our method on real SSD hardware. Experimental results show that our strategy improves SSD performance and endurance, offering valuable insights for future firmware optimization.

1 | Introduction

NAND flash memory, renowned for its high performance, strong shock resistance, and ease of use, has established a dominant position in numerous storage devices [1–3]. Its nonvolatile nature, compact size, and low power consumption have made it the storage medium of choice across a wide range of applications. As the cost of flash technology continues to decrease, flash-based Solid-State Drives (SSDs) are gradually replacing traditional mechanical hard drives. This shift not only revolutionizes storage in personal electronics such as laptops and desktop computers but also plays an increasingly pivotal role in data centers and high-performance computing environments [4–6].

However, this trend introduces challenges inherent to the NAND flash architecture, particularly in terms of data management and maintenance. Garbage Collection (GC) emerges as a key process in this regard [7–9]. Because NAND flash memory can only be erased at the block level, the update or deletion of data generates a substantial amount of invalid data fragments. To maintain optimal SSD performance and prolong its lifespan, these fragments must be periodically cleaned through the GC process.

Yet, GC often entails additional write operations, thereby causing performance degradation. This underscores the importance of efficient GC algorithms, as their effectiveness directly influences the overall performance and durability of SSDs. In response, researchers have proposed various advanced GC strategies to

Abbreviations: ABC, a black cat; DEF, doesn't ever fret; GHI, goes home immediately.

minimize performance overhead and fully harness the potential of NAND flash memory in diverse application scenarios [10–15].

Relying solely on general strategies is insufficient for further optimizing GC performance and extending SSD lifespan. Distinguishing between hot data (frequently accessed) and cold data (infrequently accessed) is a critical approach to improving GC [16–18]. Inefficient handling of cold data can significantly compromise SSD endurance and efficiency.

Accurately identifying and segregating hot and cold data has thus become a key research direction for enhancing GC efficiency [19, 20]. Traditional methods, such as linked lists, have been used [21], but their computational and storage overhead is substantial, making them unsuitable for high-speed, large-scale parallel SSD environments. Bloom filters provide a probabilistic, lightweight, and efficient solution for data classification. Although previous works such as MHF [19] and MBF [20] used Bloom filters to improve GC efficiency, they relied on outdated block-level mapping techniques and did not fully exploit the block-level parallelism of NAND flash. Moreover, these strategies were not verified on real SSD devices.

To address these issues, we propose a multiblock partitioning strategy based on Bloom filters, called BF-GC. This approach thoroughly considers the architectural characteristics of SSDs. Compared to conventional, overly complex implementations, our Bloom filter module is more streamlined and efficient, and its parameters and structure are optimized to meet the demands of high-performance SSDs. By classifying data according to the access counts of logical page addresses, and storing data with similar access frequencies in the same flash block, we significantly enhance GC efficiency and improve block endurance. Unlike studies relying solely on simulations, we validate BF-GC on real SSD hardware. The results demonstrate that this strategy effectively boosts SSD performance and longevity in actual scenarios, providing a scalable and practical solution to the challenges of modern SSD data management.

The main contributions of this paper are as follows:

1. By thoroughly analyzing NAND flash and the GC process, we identified key shortcomings in existing approaches, particularly in distinguishing hot and cold data and utilizing block-level parallelism. These insights guided the development of our more efficient BF-GC strategy.
2. We propose the BF-GC strategy, based on Bloom filters, with parameters and structure optimized for high-speed, parallel SSD environments. By leveraging block-level parallelism to store data with similar access characteristics together, we substantially improve GC efficiency and extend SSD lifespan.
3. Unlike previous research limited to simulation, we validate our approach on actual SSD devices. The experimental results confirm the practical effectiveness and high efficiency of our strategy, providing valuable guidance for the design and optimization of SSD firmware.

The remainder of this paper is organized as follows. Section 2 introduces the background and motivation. Section 3 details the BF-GC scheme. Section 4 presents the performance evaluation results. Section 5 reviews related work. Finally, Section 6 concludes the paper.

2 | Background and Motivation

2.1 | Background

In this section, we introduce the key features of SSDs related to this research.

Solid-state drives offer core advantages such as fast data access speeds and excellent shock resistance, making them the preferred storage solution for modern electronic devices and data centers. As shown in Figure 1, a typical architecture of these drives consists of a microprocessor, cache buffer, and multiple flash memory chips. Each component plays a crucial role in the process of data handling and storage.

The host interface connects the host system to the SSD, facilitating the transmission of commands and data for quick and reliable transfer. The processor, acting as the command center, executes firmware operations, manages the flash translation layer (FTL), and handles various data management protocols, directly influencing the drive's responsiveness and processing capacity. The cache buffer temporarily stores data, such as mapping tables, to

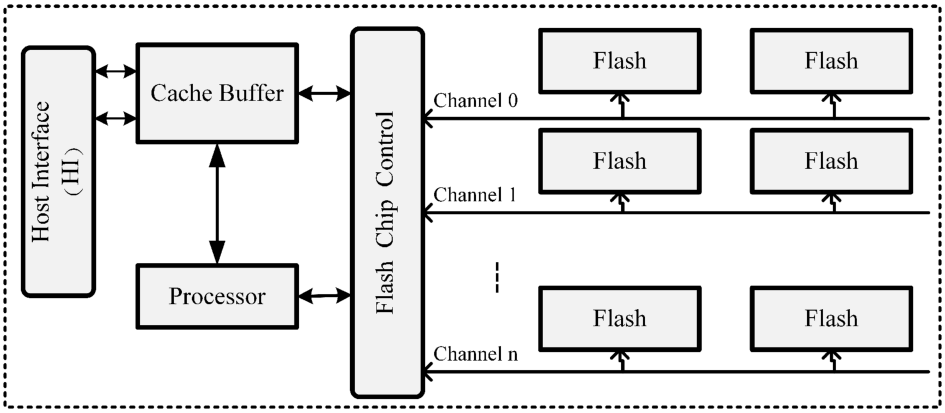

FIGURE 1 | Solid-state storage system architecture overview.

enable fast access and writing. The flash controller coordinates the operations of flash memory chips across multiple channels, optimizing parallel processing and data flow. Flash memory chips store user data, organized into multiple blocks, with each block containing several pages. By connecting several chips to the flash controller across different channels, SSDs can process data in parallel, significantly boosting performance.

As technology advances, the performance improvements of solid-state drives largely depend on enhancements to their hardware components, such as the adoption of faster buses, increased processor frequencies, and the parallelism of flash memory chips. However, beyond hardware upgrades, the optimization of data management strategies is also a key to enhancing SSD performance. In particular, intelligent categorization of data into hot and cold is crucial for improving cache efficiency, reducing write amplification, and optimizing the garbage collection process.

Existing research has proposed various data classification-based optimization methods to improve the efficiency of GC in SSDs. For example, MHF uses multiple hash functions to map data blocks into different access frequency categories, reducing misjudgments associated with a single hash approach. Similarly, MBF utilizes multiple Bloom filters with varying thresholds to more accurately distinguish between hot data and cold data. These methods optimize the GC process by minimizing unnecessary data movement and perform well in low-load scenarios. However, their limitations become evident in high-load environments. MHF and MBF rely on a single-level classification mechanism, which fails to dynamically adapt to rapidly changing access patterns. This leads to confusion between hot and cold data during the partitioning process, increasing write amplification and consequently reducing SSD performance and lifespan. Furthermore, both MHF and MBF are designed based on outdated block-level mapping mechanisms, meaning their data degradation approaches do not account for the computational overhead of handling large volumes of data in high-load conditions.

2.2 | Motivation

In large-capacity, high-performance SSD applications, effective segmentation of hot and cold data is the core of optimizing SSD data management strategies. The traditional linked-list method distinguishes hot and cold data through precise tracking, but when handling large amounts of data, the storage overhead and performance bottlenecks of the linked-list method become particularly significant, making it difficult to meet the demands of high-speed parallel processing. On the other hand, Bloom filter-based methods, due to their high space efficiency and query speed, have become a common solution. Bloom filters significantly reduce storage overhead and accelerate data queries, particularly in SSDs with multichannel architectures, where they can take advantage of parallel processing. However, existing Bloom filter-based solutions still have certain limitations. For instance, many methods still rely on outdated block-level mapping techniques, failing to fully utilize the block-level parallelism of SSDs. Moreover, some schemes introduce complex computations (such as exponential decay), which increase the system's computational overhead and impact scalability in practical applications.

To address these issues, we propose an entirely new data management strategy specifically designed for large-capacity, high-performance SSD environments. Our solution optimizes the parameters of Bloom filters to make them more suitable for the characteristics of SSDs and simplifies the complex computational steps found in traditional methods. Specifically, we use a direct counting reset approach to reset the state, avoiding the complexity of exponential decay calculations, thereby significantly reducing computational overhead. Additionally, by fully leveraging the block-level parallelism of SSDs, we group data with similar access frequencies into the same flash block. This not only enhances the efficiency of garbage collection but also effectively extends the lifespan of flash blocks, further improving overall performance.

The innovation of this research lies in simplifying and optimizing the existing Bloom filter methods, addressing the bottlenecks in high-performance SSD applications, and validating the strategy's effectiveness and efficiency through real hardware experiments. Our approach not only reduces system complexity but also fully utilizes SSD's parallel processing capabilities, offering a scalable and efficient solution for future SSD data management strategies.

3 | BF-GC

In this section, we describe the details of the BF-GC scheme. First, we present a system overview of BF-GC. Then, we will describe the bloom Classifier and block storage optimizer of BF-GC.

3.1 | System Overview

The BF-GC scheme represents an integration of advanced data classification and storage optimization techniques within the SSD management framework. Its purpose is to tackle the challenges related to data volume, access patterns, and storage efficiency in SSDs. This scheme operates on two distinct levels: data classification using the bloom Classifier and data storage optimization facilitated by the block storage optimizer.

Through the bloom Classifier, data is categorized based on access frequency, enabling efficient identification of hot and cold data. Meanwhile, the block storage optimizer focuses on optimizing the placement of data within the SSD's memory blocks and streamlining garbage collection processes. By strategically organizing data and minimizing unnecessary write and erase cycles, BF-GC significantly extends the SSD's lifespan and enhances its overall performance.

In Figure 2, we present the system overview of an SSD-based storage system implementing the BF-GC scheme. The host side comprises three layers: the application layer, file system layer, and block device layer. Connectivity between the host and SSD is established through the PCIe interface, leveraging the NVMe protocol. When the host generates I/O requests, they are transmitted to the SSD via this interface. Subsequently, the BF-GC within the SSD processes these requests and coordinates data storage operations within the flash memory.

BF-GC comprises two components: the bloom Classifier and the block storage optimizer. In Figure 2, when the SSD receives a

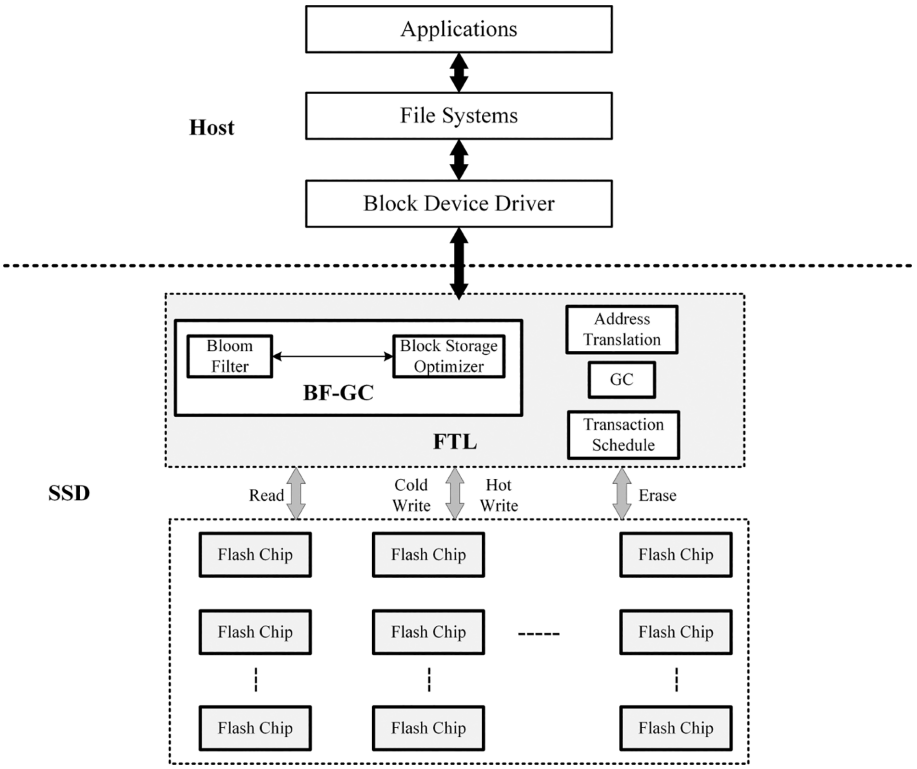

FIGURE 2 | Exploring the architecture of solid-state drives with BF-GC enhanced storage system.

write request from the host, the bloom Classifier determines the logical address corresponding to the request using the Bloom filter. It then records the number of hits for the logical address in a piece of metadata. Based on the number of hits for the logical address, the block storage optimizer determines the hotness or coldness of the data. Subsequently, the block storage optimizer parallelly transmits the data, divided by the bloom Classifier, to the appropriate hot and cold flash memory blocks at the block level. By distinguishing and segregating hot and cold data, BF-GC can effectively enhance the efficiency of data migration during garbage collection, thereby improving the performance and durability of the SSD. In the subsequent sections, we will delve into the specifics of the bloom Classifier and the block storage optimizer.

3.2 | Bloom Classifier

To efficiently manage data access patterns in SSDs and reduce unnecessary data migration during garbage collection, we propose a data classification mechanism based on Bloom filters. A Bloom filter is a space-efficient, fast querying probabilistic data structure that can quickly determine whether an element is a member of a specific set. In large-scale data environments, accurately classifying hot and cold data is crucial. Proper data classification not only optimizes storage layout and the GC process but also enhances SSD performance and extends its lifespan.

In this approach, we use 8 independent hash functions to capture the logical page address (LSA) hit information. As illustrated in Figure 3, each LSA is processed through eight independent hash functions, and the resulting outputs set the corresponding bits in the Bloom filter bitmap, where relevant bits are set to 1 for new LSA requests, while bits remain 0 for LSAs that are not hits. Compared to traditional 4-hash function architecture, modern SSDs, with their larger address spaces, benefit from 8 independent hash functions that provide more precise mapping of logical addresses, effectively reducing conflict and collision rates. The output of each hash function is mapped to the Bloom filter's bitmap, where relevant bits are set to 1 for new LSA requests, while bits remain 0 for LSAs that are not hits.

To balance the false positive rate and memory usage, we configure 8 hash functions and limit the bitmap size to 5% of the cache space, resulting in a false positive rate of approximately 5%. Studies show that this design is reasonable within the memory resources available in modern SSD controllers and can significantly improve query accuracy and performance.

By optimizing the number of hash functions and bitmap configuration, the proposed Bloom filter design effectively balances query accuracy and memory overhead in large-capacity SSDs. This design, by more precisely capturing data access patterns, helps optimize data storage layout and the garbage collection process, thereby improving SSD performance and extending its lifespan.

Frequency. To track hit frequency information, unlike the 4-bit strategy in the multihash function framework [19], we introduce a data variable named state in the logical page address to record the number of hits, as we need to classify each hit in a more fine-grained manner. In other words, the number of hits is not recorded in the Bloom filter, but in the state variable of the LSA. When a write request arrives, we first check the hit status in the Bloom filter. If the write request does not hit, it indicates that the LSA is a new write data, and we set the corresponding bit of the

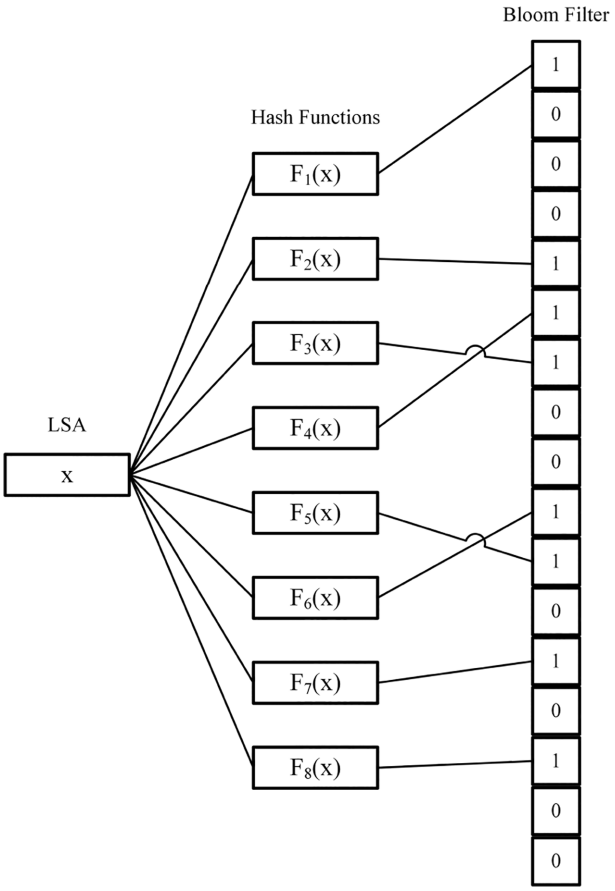

FIGURE 3 | The Bloom filter framework.

Bloom filter to 1. For the state variable in the LSA, since the initial value is zero, we do not perform any operations on it, meaning that the write request is for new data. For data that has already been hit, when the value of the LSA in the Bloom filter corresponding to the new write request is 1, it indicates that the LSA has been hit at least once. The value in the Bloom filter remains 1, and the state in the LSA is incremented by 1 to record the number of LSA hits. Compared to the simple 0 and 1 recording in the multihash architecture, our scheme can record the information of each hit in a more fine-grained manner, allowing the data to be divided into multiple states, rather than just simple cold and hot states.

How to Alleviate the Computational Burden of Zeroing Operations?. In modern SSD systems, as storage capacities continue to increase, efficiently managing large-scale data, especially during data zeroing operations, has become a critical challenge. Traditional zeroing methods, such as record-by-record or bitwise operations, are effective for small datasets but lead to significant computational overhead as the data size grows, resulting in reduced processing efficiency. This issue is particularly prominent in modern SSDs, which utilize page-level mapping in the FTL architecture, handling much larger data volumes than those accommodated by traditional LBA (Logical Block Addressing) methods. To address this challenge, we propose a direct zeroing mechanism. This approach monitors the SSD's runtime and sets a threshold value. When the SSD reaches this threshold, the memset function is used to immediately reset the state of all entries in the Bloom filter to zero. This method performs a bulk reset of data states, avoiding the computational burden typically associated with zeroing large amounts of data individually. By reducing the computational load, the direct zeroing mechanism not only enhances SSD performance but also maintains data integrity and the efficiency of the garbage collection process. Therefore, this method offers an efficient and scalable solution for high-performance SSDs, meeting the data management demands of large-capacity storage while ensuring optimal performance.

ALGORITHM 1 | Description of Bloom Classifier.

Require: The logicalSliceAddr requests *Lsa_1*, *Lsa_2*, ..., *Lsa_i*.
Ensure: Bloom Classifier (Lsa_i)

```
if Bloom_filter_contains(Lsa_i) then
    LSA.state = LSA.state + 1
else
    Bloom_filter_add(Lsa_i)
    LSA.state = LSA.state
end if
if time_to_reset_Bloom_filter() then
    Reset Bloom filter using memset()
end if
End Procedure
```

The algorithm 1 of the Bloom Classifier is as described in the pseudocode.

In Algorithm 1, the process begins by assessing if the newly arrived LSA is present in the Bloom filter. Upon a hit in the Bloom filter, the corresponding state within the LSA is incremented by 1. Conversely, if there is no hit, the program adds the LSA to the Bloom filter and keeps the state of the LSA unchanged. When the program reaches the predefined time threshold, the Bloom filter undergoes a reset via the Memset function, effectively clearing its contents. This mechanism ensures dynamic data management, enabling efficient tracking of data access patterns, and maintaining the Bloom filter's effectiveness through periodic resetting.

3.3 | Block Storage Optimizer

After the Bloom Classifier has executed data partitioning, the Block Storage Optimizer undertakes the critical task of efficiently allocating these categorized data to flash storage. During this process, the Block Storage Optimizer relies on the temperature of the data—a measure indicating the frequency of data access—to determine and write the data to flash blocks that match its temperature. The strategy is designed with the intent to optimize data access efficiency as well as to extend the lifespan of flash storage. To illustrate the characteristics of our scenario in detail, we use Figure 4 as a typical example.

Figure 4 illustrates the difference in data transmission between the BF-GC system and traditional high-performance systems. The flash memory architecture is configured with 8 channels and 8 ways, where the intersection of each channel and way forms a die. Data is ultimately stored in the blocks of these dies. Specifically focusing on Die 0, the data queue contains various data categorized by their access frequency or temperature. In

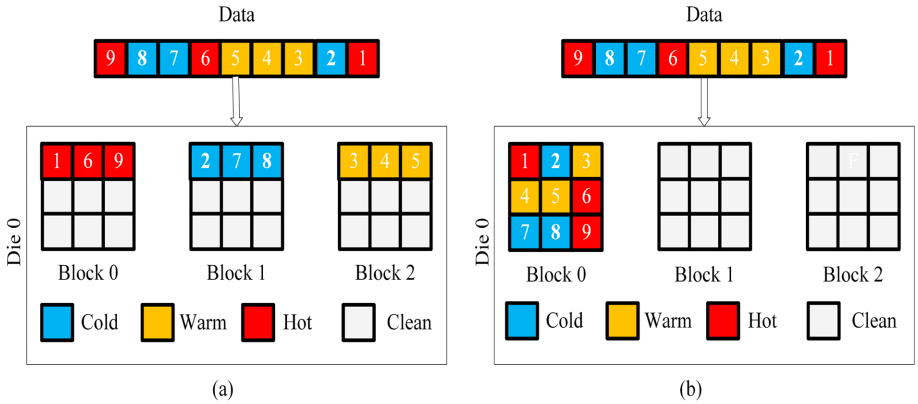

FIGURE 4 | The differences in the data transmission process between BF-GC and traditional scheme.

this context, data labeled 1, 6, and 9 are classified as hot data, indicating a very high access frequency. Data 2, 7, and 8 are categorized as warm data, showing a medium access frequency. Lastly, 3, 4, and 5 are classified as cold data, indicating the lowest access frequency.

In Figure 4a, the data queue for Die 0 is arranged in order from data 1 to data 9, demonstrating a strategic method of data storage based on temperature or access frequency. Initially, when data 1 is written, it occupies the first empty block, block 0, due to its unique temperature classification. Subsequently, data 2, having a different temperature than data 1, is allocated to the next available block, block 1, illustrating the system's approach to segregating data based on temperature. As the process continues, data 3, exhibiting a temperature distinct from both data 1 and data 2, is placed in block 2. However, when data 4 is up for writing, since it shares the same temperature as data 3, it is stored in block 2 alongside data 3, showcasing an efficient use of space by grouping together data of similar temperatures. Similarly, data 5, 6, 7, 8, and 9 are written into different blocks corresponding to their own temperatures. This meticulous organization ensures that data of varying temperatures are isolated within different flash blocks, effectively preventing the mixing of different data types. Such a structured approach not only optimizes data retrieval by closely aligning data storage with access frequencies but also enhances the overall efficiency and lifespan of the flash memory by minimizing unnecessary read/write cycles among different data types. By categorizing data according to temperature and storing them in appropriate flash blocks, the BF-GC system is able to optimize data access efficiency and extend the lifespan of flash memory. This is because it reduces the wear on flash memory that could occur due to frequent access to hot data. This method not only improves the overall performance of the storage system but also extends the device's lifespan by effectively managing the storage location of data.

Compared to the BF-GC method shown in Figure 4a,b presents a more basic data storage strategy adopted by traditional high-performance flash memory systems. Under this strategy, once data 1 is written, it initially occupies the empty space in block 0. This practice contrasts with the previous approach of allocating different types of data to specific blocks based on their "temperature." In this mode, as long as block 0 is not full, subsequent data items continue to be sequentially written to the same block. Thus, when data 2 arrives, it is stored in block 0 right after data 1, given that there is still space available. This sequential writing process also applies to the following data items 3, 4, 5, 6, 7, 8, and 9, which are successively accumulated in block 0 until its capacity is saturated. This strategy does not fully utilize the block-level parallelism of flash memory but instead opts for simple, continuous writing operations to a single block. Only when block 0 is completely filled does the system start to write new data to the next free block, which is block 1. Although this linear or sequential data storage method simplifies the data management process to some extent, it shows certain shortcomings in enhancing data migration efficiency and extending the lifespan of flash memory compared to the temperature-based isolation storage strategy described in Figure 4a. Because it does not optimize the placement of data based on access frequency, frequently accessed data may be dispersed across multiple blocks, potentially increasing the number of read/write operations and thereby affecting the overall performance and lifespan of the flash memory.

4 | Performance Evaluation

In this section, we first describe the experimental setup and test benchmarks. Then we use the benchmarks of FIO to evaluate the performance of BF-GC.

4.1 | Experimental Setup

The Cosmos+ FPGA OpenSSD [22] development platform is equipped with HYNIX H27Q1T8YEB9R flash memory chips, which are of the MLC NAND type with 16KB pages and 128 pages per block. Each die contains 8192 effective blocks, and there are a total of 64 dies, organized into 8 channels and 8 ways for efficient data processing. The microcontroller utilizes Xilinx's ZYNQ-7000 series chip, featuring two ARM Cortex-A9 embedded CPUs. Additionally, the controller includes 1 GB DRAM for storing metadata such as the FTL mapping table and buffer cache data. Connectivity with the host is established through the PCIe interface with NVMe protocol. The NVMe protocol version is 1.2, and the PCIe interface utilizes Xilinx 7 series IP cores (PCIe 2.0 version), ensuring efficient command queuing and data transfer mechanisms. These specifications collectively contribute to the robust performance and reliability of the Cosmos+

FPGA OpenSSD, making it suitable for high-speed data processing requirements in storage systems.

The host machine features an Intel Core i7-4790K processor running at 4.4 GHz, complemented by 16 GB DDR3 DRAM and a WD SN750 Black 500 GB SSD. The operating system used is Ubuntu 16.04, built on the Linux kernel version 4.15, utilizing the ext4 file system. This configuration provides a robust foundation for supporting various computational tasks and storage operations. The combination of a high-performance processor, ample memory capacity, and solid-state storage ensures efficient data processing and rapid access to files and applications under the Ubuntu 16.04 environment.

4.2 | Benchmark and Comparison

FIO (Flexible I/O tester), cited as [23], is a versatile tool tailored to generate multiple threads or processes, executing predefined I/O operations as directed by users. Typically, FIO benchmark tests are used to gauge the efficiency of files and storage systems. In our investigation, we leverage FIO to evaluate the influence of each strategy on the random read and write performance of the file system across different proportions. The FIO parameters are set to: use the libaio engine, random read-write mode, 4KB block size, 10 concurrent jobs, and test 100 GB of data.

In our comparative analysis of the Cosmos+ FPGA OpenSSD development platform, we use Cosmos+ as the baseline for comparison. As a leading open-source SSD device, Cosmos+ uses an FTL strategy that is also adopted by many mainstream commercial high-performance SSDs. To conduct a more in-depth analysis, besides our proposed BF-GC scheme, we have also implemented an improved version of the Multiple Hash Functions (MHF) scheme on the Cosmos+ platform as another comparative approach. Considering that the original MHF scheme was designed for an older emulation environment, many of its design concepts no longer meet the requirements of modern high-performance SSDs. Therefore, while maintaining the core idea of using multiple hash functions along with a single Bloom filter in the MHF scheme, we updated other outdated design concepts to ensure its seamless operation on the Cosmos+ OpenSSD. This revised version is named the MHF-Optimized version. By selecting the MHF scheme as a point of comparison, we effectively showcase a mainstream SSD optimization strategy based on Bloom filters, providing a reliable basis for our analysis.

4.3 | Performance

To rigorously evaluate SSD performance under real-world conditions, we developed a comprehensive testing framework. The process begins with writing and erasing 1TB of data to replicate usage after an initial period. This step transitions the SSD to a "steady state", ensuring full activation of management protocols, including garbage collection.

After the warm-up phase, performance testing continues with 300 GB of data to evaluate the SSD in its steady state. This phase assesses performance after the SSD has adapted to regular read-write operations, ensuring a precise and realistic measurement of its capabilities. This approach reveals how SSDs evolve with prolonged use, providing critical insights for informed decision-making. By simulating real usage patterns, it enables a thorough evaluation of SSD resilience and efficiency under heavy workloads, helping consumers and businesses choose reliable storage solutions.

To thoroughly evaluate SSD performance in different read/write scenarios, we performed tests using five specific ratios: 1%, 25%, 50%, 75%, and 99%. These ratios represent a wide spectrum of workload intensities, allowing us to analyze SSD behavior under various conditions. Our evaluation focused on two key performance metrics. IOPS and bandwidth directly indicate the SSDs' data processing capabilities. In addition to throughput, we examined block erase counts and garbage collection page moves to assess block usage efficiency and the effectiveness of page migration during garbage collection. By integrating these aspects, our analysis provides a comprehensive view of SSD performance, shedding light on their operational efficiency, durability, and adaptability to different workload demands.

In Figure 5, we observe the SSD's performance metrics under an FIO write ratio of 1%. Despite the increment from zero in block erase counts and GC page move counts, as detailed in Figure 5e,f, the Input/Output Operations Per Second (IOPS) and bandwidth (BW) depicted in Figure 5a–c, and d exhibit negligible variance. This minimal variation can be attributed to the predominance of read operations over write activities due to the low write ratio. As a result, the performance distinctions across different scenarios become inconsequential.

This observation suggests that at a 1% write ratio, the SSDs' performance is largely unaffected by variations in FTL strategies or GC activities. Since the majority of operations are read-centric in this context, the potential impacts of differing FTL mechanisms or the efficiency of GC processes on overall performance are minimized. Hence, in scenarios where read operations significantly outweigh write actions, the choice of FTL strategy or the nuances of GC mechanisms are less likely to influence the SSD's performance outcomes.

In Figure 6, we observe the performance of SSDs under a 25% FIO write ratio condition. Throughout the 300 GB testing sequence, the BF-GC scheme, which uses a multiblock strategy, surpasses the performance of the other two schemes. Notably, at the 100 GB data checkpoint, Cosmos+ demonstrates slightly better performance compared to MHF-Optimized. However, as the data volume increases to 200 GB, the performances of Cosmos+ and MHF-Optimized begin to converge. When the test data volume reaches 300 GB, MHF-Optimized starts to outperform Cosmos+, indicating that the efficacy of these schemes changes with increasing data loads.

The data reveal that the BF-GC scheme not only provides superior and more stable performance but also outperforms the single hot and cold block strategy used by MHF-Optimized. Compared to the existing architecture of Cosmos+, the block-level parallelism used by BF-GC effectively reduces the overhead of GC operations. In testing scenarios like the 300 GB test, BF-GC proves to offer better performance over the other two schemes, thereby highlighting the importance of block-level strategies and

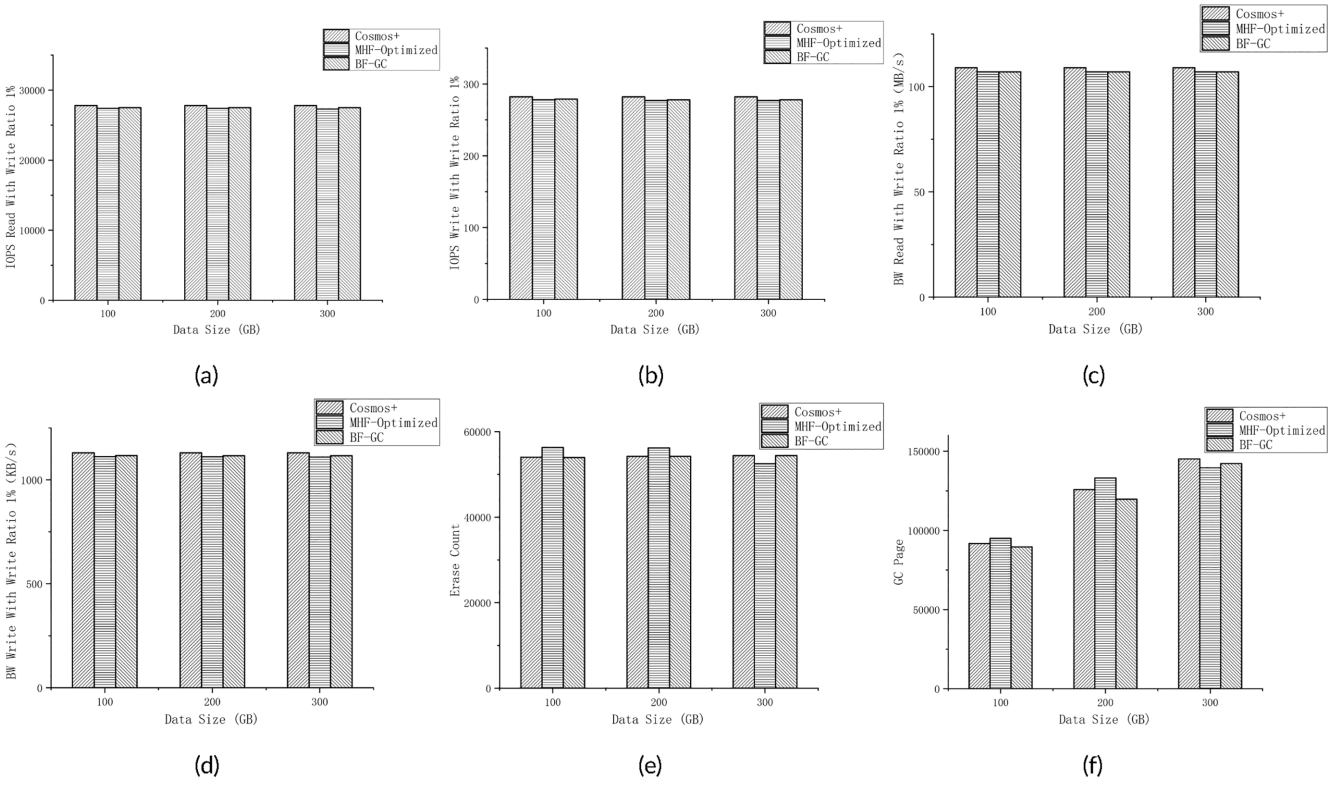

FIGURE 5 | FIO: 300 GB data with 1% write ratio. (a) IOPS Read, (b) IOPS Write, (c) BW Read, (d) BW Write, (e) Erase Count, and (f) GC Page.

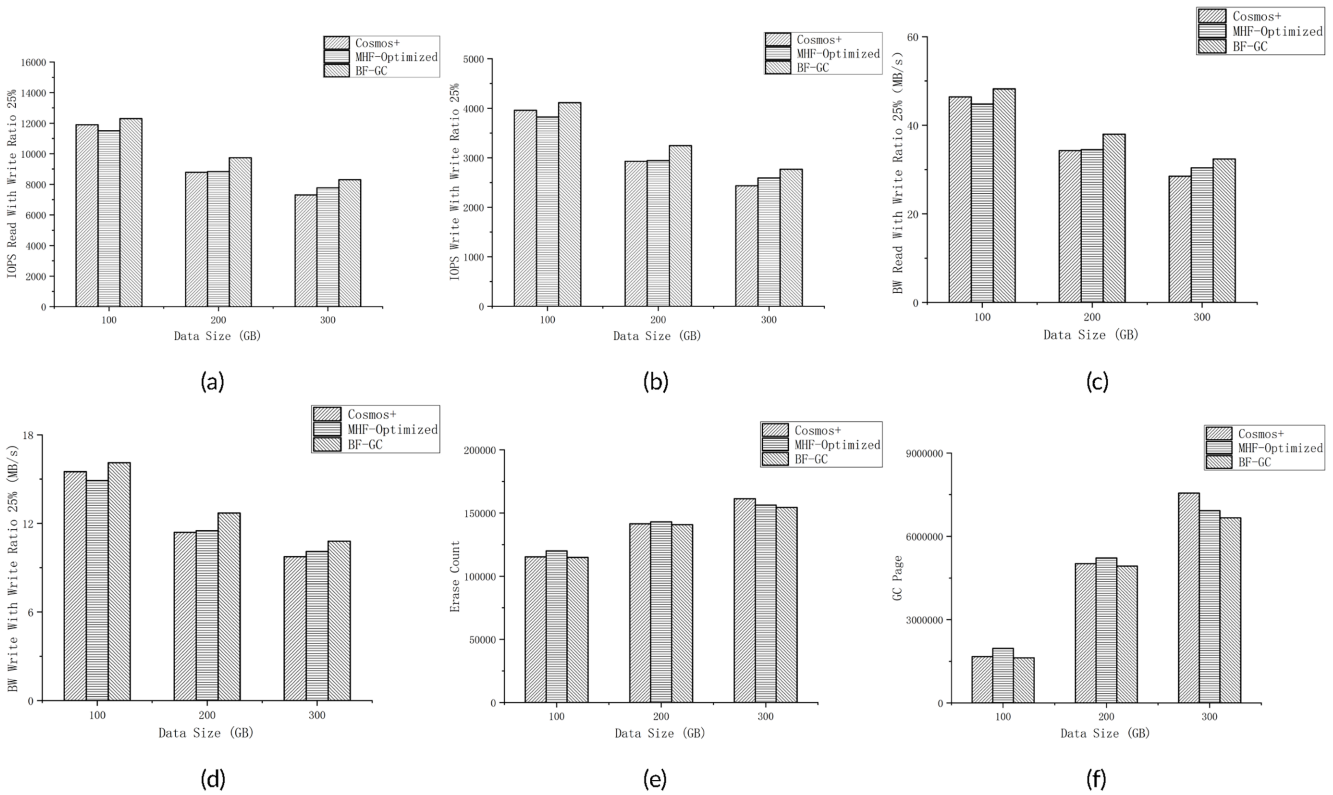

FIGURE 6 | FIO: 300 GB data with 25% write ratio. (a) IOPS Read, (b) IOPS Write, (c) BW Read, (d) BW Write, (e) Erase Count, and (f) GC Page.

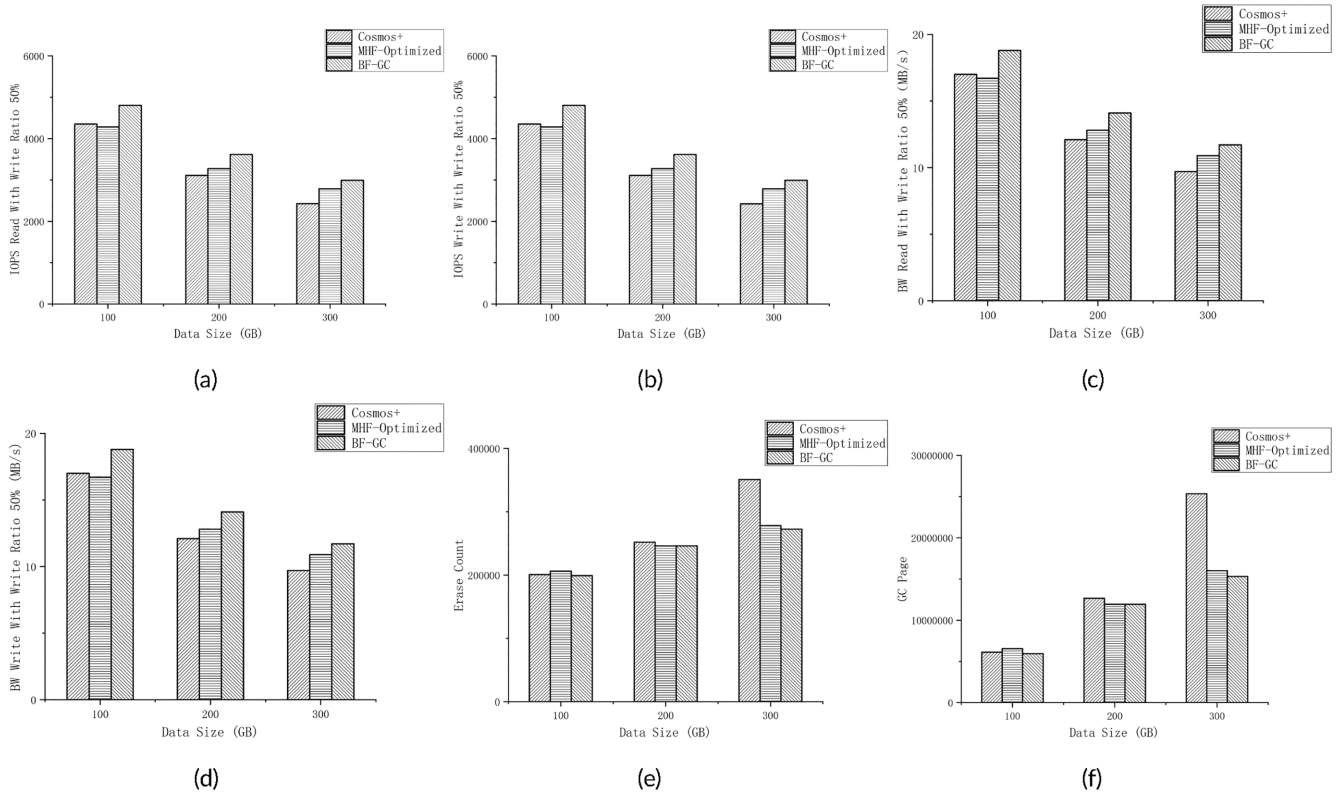

FIGURE 7 | FIO: 300 GB data with 50% write ratio. (a) IOPS Read, (b) IOPS Write, (c) BW Read, (d) BW Write, (e) Erase Count, and (f) GC Page.

their impact on enhancing GC efficiency in SSDs. These insights suggest that for tasks with moderate to low write loads, the BF-GC architecture may offer significant advantages in maintaining high-performance levels.

In Figure 7, we analyze SSD performance under a 50% FIO write ratio, where the advantages of BF-GC become significantly more pronounced. Throughout the 300 GB testing phase, BF-GC consistently outperforms the other two schemes in all performance metrics. Compared to the 25% write ratio scenario in Figure 6, this performance gap becomes even more evident, demonstrating how more frequent GC triggers at a 50% write ratio further enhance BF-GC's efficiency.

Figure 7 presents a comprehensive analysis of SSD performance under a 50% FIO write ratio, highlighting the significant advantages of BF-GC over Cosmos+ and MHF-Optimized across all key performance metrics. In terms of IOPS, BF-GC consistently outperforms the other two schemes, as illustrated in Figure 7a and b. Notably, in write IOPS, BF-GC demonstrates a considerable lead, indicating its ability to efficiently handle data migration and garbage collection processes under a high write ratio. This efficiency reduces write amplification while improving overall throughput. Similarly, bandwidth performance, as shown in Figure 7c and d, further corroborates this advantage. Specifically, in write bandwidth, BF-GC achieves improvements of 23.5% and 10.4% over Cosmos+ and MHF-Optimized, respectively, demonstrating its ability to maintain high data throughput even under frequent GC triggers. This stable bandwidth performance minimizes performance fluctuations caused by GC operations, making BF-GC particularly suitable for write-intensive workloads.

Moreover, BF-GC exhibits clear optimizations in GC-related metrics. As shown in Figure 7e, BF-GC records significantly lower erase counts compared to Cosmos+ and MHF-Optimized, indicating a more efficient GC mechanism that reduces unnecessary erase operations, thereby mitigating write amplification and extending SSD lifespan. Additionally, Figure 7f depicts the number of pages involved per GC event, where BF-GC significantly reduces GC Page through optimized data migration strategies, further minimizing the overhead caused by garbage collection. Overall, Figure 7 provides a comprehensive illustration of BF-GC's advantages under a 50% write ratio scenario, demonstrating its superior performance in IOPS and bandwidth while also achieving lower erase counts and reduced GC overhead. These findings confirm BF-GC's exceptional adaptability to write-intensive environments, making it a highly efficient choice for demanding workloads.

In Figure 8, we examine SSD performance under a 75% FIO write ratio, highlighting the dynamics under higher write loads. Initially, at 100 GB, BF-GC surpasses both Cosmos+ and MHF-Optimized, with MHF-Optimized slightly lagging behind Cosmos+. As the data volume increases to 200 GB and 300 GB, BF-GC maintains its lead, showcasing superior performance over MHF-Optimized, which in turn outperforms Cosmos+. This performance pattern aligns with observations from the 25% and 50% write ratio scenarios, emphasizing the growing performance edge of the BF-GC as the write ratio escalates.

Figure 8 presents a comparative analysis of SSD performance under a 75% write ratio, highlighting the distinct performance trajectories of BF-GC, Cosmos+, and MHF-Optimized. BF-GC

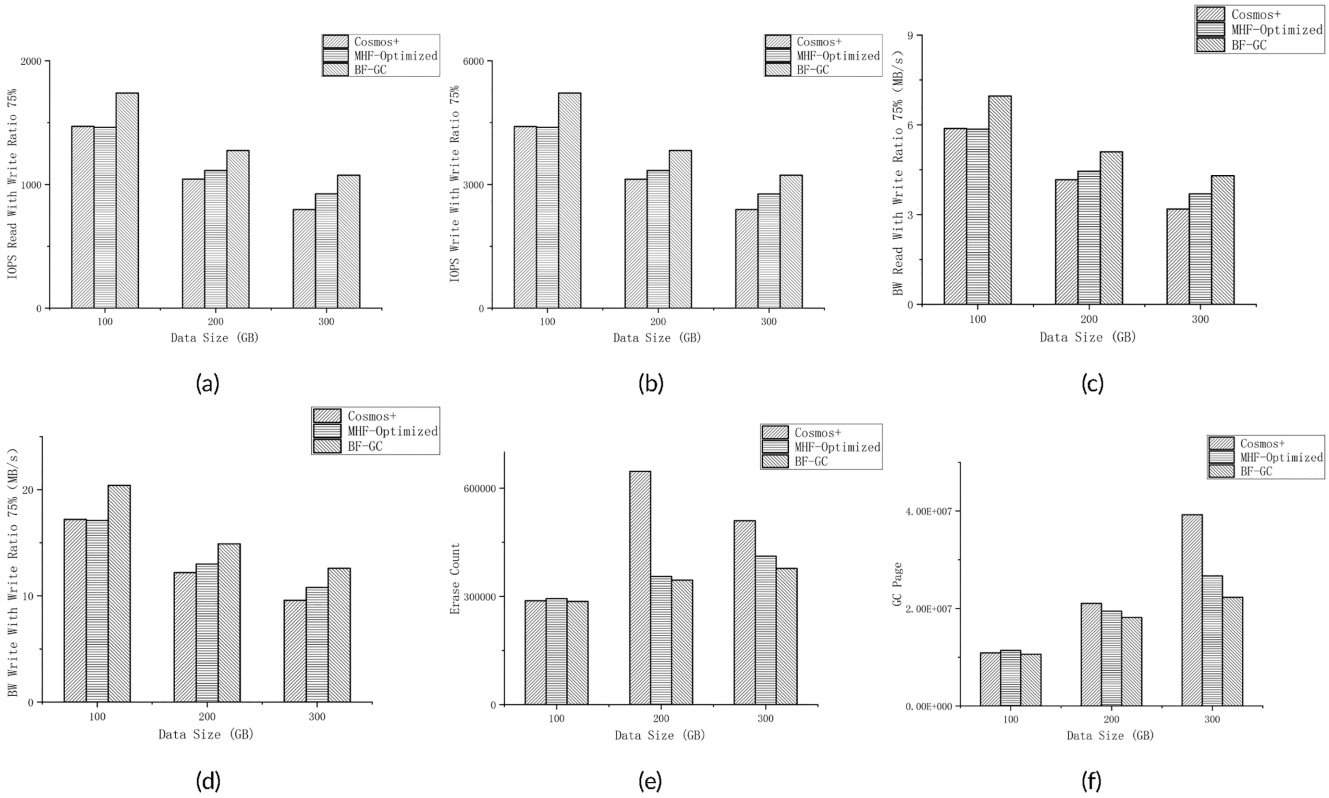

FIGURE 8 | FIO: 300 GB data with 75% write ratio. (a) IOPS Read, (b) IOPS Write, (c) BW Read, (d) BW Write, (e) Erase Count, and (f) GC Page.

demonstrates a consistent and robust performance across various metrics, particularly as write demands intensify. As shown in Figure 8, BF-GC maintains a significant advantage in IOPS and bandwidth, consistently outperforming the other two schemes. The consistency in BF-GC's performance can be attributed to its granular data partitioning strategy and the direct count reset mechanism, both of which enable more effective data management while minimizing computational overhead. In contrast, MHF-Optimized exhibits performance fluctuations, initially trailing Cosmos+ but eventually surpassing it, only to experience performance declines as data volumes increase. This volatility, particularly under heavy write operations, underscores MHF-Optimized's instability in such scenarios.

Specifically, under the most favorable conditions, BF-GC achieves a performance uplift of 34% over Cosmos+ and 16% over MHF-Optimized, as depicted in Figure 8. This advantage is most pronounced in write IOPS and write bandwidth, where BF-GC shows superior handling of high write loads and remains resilient despite the frequent GC triggers inherent to such environments. In terms of erase count and GC page involvement, BF-GC again leads, demonstrating its efficiency in managing write amplification and minimizing the overhead associated with garbage collection.

Overall, Figure 8 underscores BF-GC's exceptional ability to maintain high performance, even as write demands increase. Its superior efficiency and stability, coupled with its resilience to frequent GC triggers, make it an ideal choice for applications requiring high endurance and consistent performance under substantial write workloads.

Figure 9 illustrates the performance of SSDs under a 99% FIO write ratio, indicating that such a high write proportion significantly increases the burden on the GC mechanism. Under these extreme write conditions, BF-GC demonstrates significant performance advantages, validating the effectiveness of its methods in optimizing write operations and GC, especially under high write load scenarios. Similarly, in tests with data volumes of 200 GB and 300 GB, BF-GC outperforms the MHF optimization approach. This advantage can primarily be attributed to BF-GC's adoption of a more fine-grained partitioning strategy, coupled with a direct counting zeroing mechanism, which effectively reduces computational overhead, thereby achieving superior performance in high-load environments.

Figure 9 further illustrates the effectiveness of BF-GC in mitigating GC-related performance overhead and reducing block erasures, particularly when compared to the traditional Cosmos+ architecture. As shown in Figure 9a,b, BF-GC consistently outperforms both Cosmos+ and MHF-Optimized in read and write IOPS, demonstrating its ability to sustain high-speed operations even under extreme write conditions. Similarly, in bandwidth performance, BF-GC maintains a significant advantage, particularly in write bandwidth, where it ensures stable data throughput, despite the high write ratio. This highlights BF-GC's efficiency in managing high write loads while minimizing performance degradation caused by frequent GC triggers.

Additionally, BF-GC exhibits a clear advantage in GC-related metrics, as shown in Figure 9e,f. The erase count in Figure 9e is substantially lower for BF-GC compared to Cosmos+ and MHF-Optimized, indicating a more efficient GC mechanism that

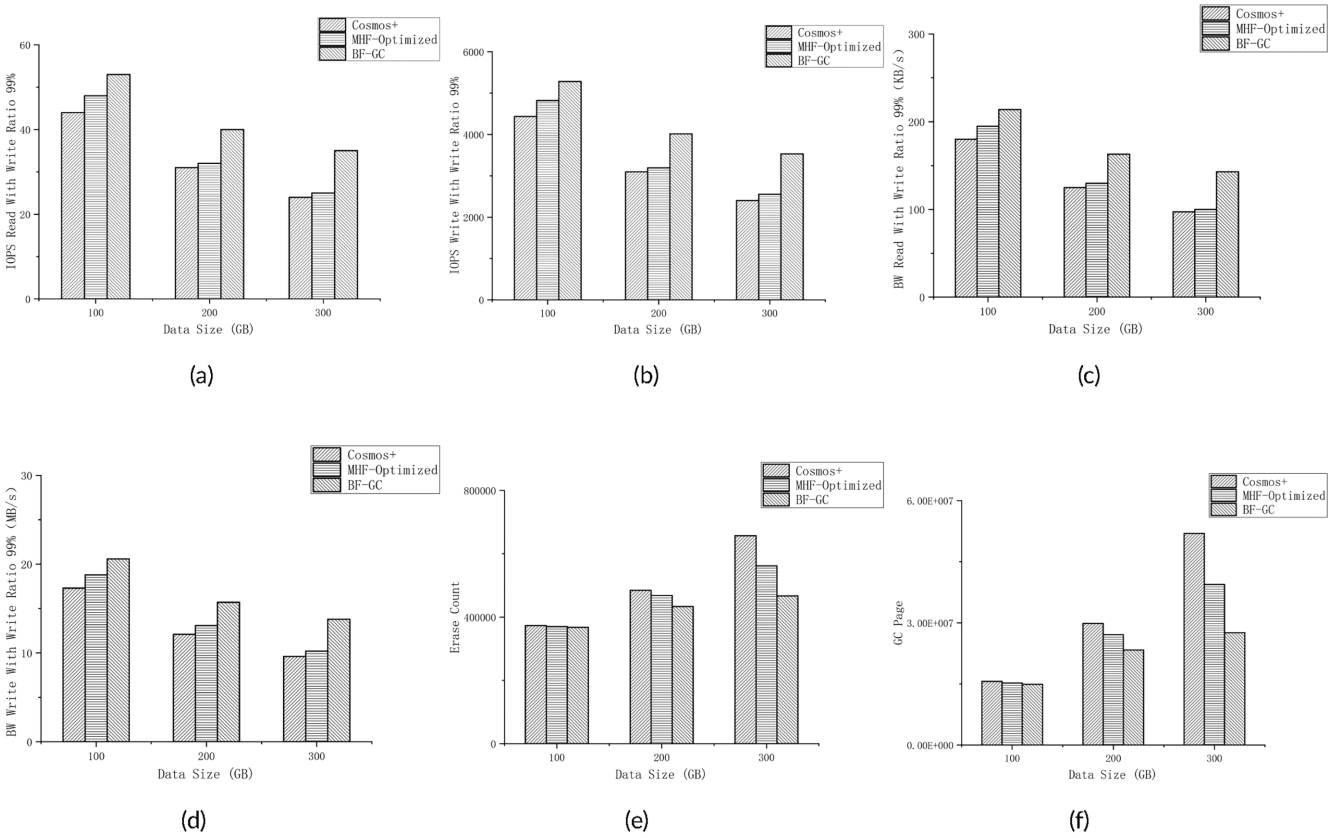

FIGURE 9 | FIO: 300 GB data with 99% write ratio. (a) IOPS Read, (b) IOPS Write, (c) BW Read, (d) BW Write, (e) Erase Count, and (f) GC Page.

minimizes unnecessary erase operations, thereby reducing write amplification and prolonging SSD lifespan. Moreover, Figure 9f shows that BF-GC involves fewer pages per GC event, further underscoring its efficiency in managing garbage collection overhead. In contrast, while MHF-Optimized achieves competitive results in specific scenarios through simple hot and cold data partitioning, its overall stability and performance remain inferior to BF-GC, especially under sustained high write workloads.

Overall, compared to the mainstream Cosmos+ architecture, BF-GC effectively mitigates the impact of GC on SSD performance while significantly reducing block erasures, ultimately enhancing SSD durability. Unlike MHF-Optimized, which relies solely on a basic data partitioning scheme, BF-GC demonstrates superior stability and performance across a wide range of workloads. These findings reinforce BF-GC's suitability for applications that require sustained high write endurance, ensuring long-term reliability and efficiency.

5 | Related Work

In the research focused on optimizing garbage collection for SSD, victim block selection and hot data identification emerge as pivotal aspects. The greedy algorithm [24, 25], by selecting blocks with the fewest valid pages for GC, offers a method to minimize data movement and thereby enhance overall SSD performance. The D-choice algorithm [26], inspired by the greedy approach, selects a victim block from multiple candidates, aiming to expedite the selection process while balancing efficiency and wear leveling. Furthermore, advancements in hot data identification techniques play a vital role in improving GC efficiency. Chang et al. [21]. used a two-level LRU list structure to identify impending hot write requests, whereas Park et al.'s multiple Bloom filter scheme [19] focuses on identifying hot data within flash memory. These methods, by accurately pinpointing frequently accessed data, contribute to optimized data storage strategies and improved access efficiency. The MBF [20] scheme, utilizing multitiered Bloom filters, further enhances the rationale and efficiency of hot data identification, targeting optimized GC processes and elevated SSD performance through improved handling of hot data. These studies illustrate the potential of innovative algorithms and techniques in addressing the challenges of SSD GC and data management, with each approach contributing toward enhanced storage efficiency and extended device lifespan.

Building on these foundational approaches, our proposed BF-GC scheme introduces a novel application of Bloom filters for data partitioning, with a focus on refined granularity and enhanced block-level parallelism. Unlike previous schemes that primarily utilize simulators for design and validation, our approach leverages real-world devices to authenticate and refine our design. This practical validation underscores the efficacy of the BF-GC scheme in real-world applications, demonstrating significant improvements in GC efficiency and SSD performance. By meticulously crafting data partition strategies and harnessing the power of block-level parallel processing, the BF-GC scheme effectively minimizes unnecessary data movements and optimizes the overall GC process. Our research not only validates the benefits of incorporating advanced data identification and partitioning

techniques but also highlights the importance of real-device testing in developing practical solutions for SSD optimization.

6 | Conclusion

This study proposes a multilevel classification strategy based on Bloom filters, specifically designed for modern SSDs. The method efficiently distinguishes between hot and cold data while significantly reducing computational and storage overhead. By leveraging SSD block-level parallelism, our approach enhances garbage collection efficiency and extends block lifespan. Experimental results demonstrate that, compared to traditional Bloom filter methods, our strategy achieves a better balance between computational efficiency and storage overhead, while proving its effectiveness and reliability on real SSD hardware. Nevertheless, the generalizability of this method across different types of SSDs still requires further validation. Therefore, in future work, we plan to expand the scope of our experiments by testing BF-GC on a wider range of SSD devices. We will also explore further optimization strategies to enhance its adaptability and robustness across diverse storage architectures.

Data Availability Statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

1. N. Agrawal, V. Prabhakaran, T. Wobber, et al., "Design Tradeoffs for SSD Performance," in *2008 USENIX Annual Technical Conference (USENIX ATC 08)* (2008).

2. H. Sun, S. Dai, J. Huang, and X. Qin, "Co-Active: A Workload-Aware Collaborative Cache Management Scheme for NVMe SSDs," *IEEE Transactions on Parallel and Distributed Systems* 32, no. 6 (2021): 1437–1451.

3. F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems," in *Proceedings of the International Conference on Supercomputing* (2011), 22–32.

4. D. Narayanan, E. Thereska, A. Donnelly, et al., "Migrating Server Storage to SSDs: Analysis of Tradeoffs," in *Proceedings of the 4th ACM European Conference on Computer Systems* (2009), 145–158.

5. I. Shin, "Early Dirty Buffer Flush With Second Chance for SSDs," *Micromachines* 14, no. 4 (2023): 796.

6. S. Wu, B. Mao, X. Chen, and H. Jiang, "LDM: Log Disk Mirroring With Improved Performance and Reliability for SSD-Based Disk Arrays," *ACM Transactions on Storage* 12, no. 4 (2016): 1–21.

7. J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A Semi-Preemptive Garbage Collector for Solid State Drives," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software* (IEEE, 2011), 12–21.

8. S. Lee, D. Shin, and J. Kim, "BAGC: Buffer-Aware Garbage Collection for Flash-Based Storage Systems," *IEEE Transactions on Computers* 62, no. 11 (2012): 2141–2154.

9. J. Hu, H. Jiang, L. Tian, and L. Xu, "GC-ARM: Garbage Collection-Aware RAM Management for Flash Based Solid State Drives," in *Proceedings of the 2012 IEEE Seventh International Conference on Networking, Architecture, and Storage* (IEEE, 2012), 134–143.

10. S. Wu, B. Mao, Y. Lin, and H. Jiang, "Improving Performance for Flash-Based Storage Systems Through GC-Aware Cache Management," *IEEE Transactions on Parallel and Distributed Systems* 28, no. 10 (2017): 2852–2865.

11. S. Yan, H. Li, M. Hao, et al., "Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs," *ACM Transactions on Storage* 13, no. 3 (2017): 1–26.

12. Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu, "WARM: Improving NAND Flash Memory Lifetime With Write-Hotness Aware Retention Management," in *Proceedings of the 2015 31st Symposium on Mass Storage Systems and Technologies (MSST)* (IEEE, 2015), 1–14.

13. Y. Ge, J. Bao, X. Xu, et al., "Hybrid Write Strategy Based on Hot Data Recognition and Channel Busyness Perception for Consumer Solid State Drives," *IEEE Transactions on Consumer Electronics* 69, no. 4 (2023): 1082–1090.

14. F. Chen, B. Hou, and R. Lee, "Internal Parallelism of Flash Memory-Based Solid-State Drives," *ACM Transactions on Storage* 12, no. 3 (2016): 1–39.

15. G. Wu, P. Huang, and X. He, "Reducing SSD Access Latency via NAND Flash Program and Erase Suspension," *Journal of Systems Architecture* 60, no. 4 (2014): 345–356.

16. Y. Pan, M. Lin, Z. Wu, H. Zhang, and Z. Xu, "Caching-Aware Garbage Collection to Improve Performance and Lifetime for NAND Flash SSDs," *IEEE Transactions on Consumer Electronics* 67, no. 2 (2021): 141–148.

17. S. Wang, Y. Zhou, J. Zhou, F. Wu, and C. Xie, "An Efficient Data Migration Scheme to Optimize Garbage Collection in SSDs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, no. 3 (2020): 430–443.

18. H. Yan, Y. Huang, X. Zhou, and Y. Lei, "An Efficient and Non-Time-Sensitive File-Aware Garbage Collection Algorithm for NAND Flash-Based Consumer Electronics," *IEEE Transactions on Consumer Electronics* 65, no. 1 (2018): 73–79.

19. J. W. Hsieh, T. W. Kuo, and L. P. Chang, "Efficient Identification of Hot Data for Flash Memory Storage Systems," *ACM Transactions on Storage* 2, no. 1 (2006): 22–40.

20. D. Park and D. H. Du, "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, 2011), 1–11.

21. L. P. Chang and T. W. Kuo, "An Adaptive Striping Architecture for Flash Memory Storage Systems of Embedded Systems," in *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium* (IEEE, 2002), 187–196.

22. J. Kwak, S. Lee, K. Park, J. Jeong, and Y. H. Song, "Cosmos+ Openssd: Rapid Prototype for Flash Storage Systems," *ACM Transactions on Storage* 16, no. 3 (2020): 1–35.

23. H. J. Kim, Y. S. Lee, and J. S. Kim, "{NVMeDirect}: A User-Space {I/O} Framework for Application-Specific Optimization on {NVMe}{SSDs}," in *the 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)* (2016).

24. W. Bux and I. Iliadis, "Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives," *Performance Evaluation* 67, no. 11 (2010): 1172–1186.

25. Y. Yang, V. Misra, and D. Rubenstein, "On the Optimality of Greedy Garbage Collection for SSDS," *ACM SIGMETRICS Performance Evaluation Review* 43, no. 2 (2015): 63–65.

26. B. Van Houdt, "A Mean Field Model for a Class of Garbage Collection Algorithms in Flash-Based Solid State Drives," *ACM SIGMETRICS Performance Evaluation Review* 41, no. 1 (2013): 191–202.