electronics

MDPI

Article

High-Performance Garbage Collection Scheme with Low Data Transfer Overhead for NoC-Based SSDC

Seyeon Ahn, Donghyuk Im, Donggon You and Youpyo Hong *

Division of Electronics and Electrical Engineering, Dongguk University-Seoul, Seoul 04620, Republic of Korea; 2021111916@dgu.ac.kr (S.A.); 2020111875@dgu.ac.kr (D.I.); ehdrhs1013@dgu.edu (D.Y.)
* Correspondence: yhong@dgu.edu; Tel.: +82-2-2260-3818

Abstract: Solid-state drives (SSDs) have become the preferred storage solution for performance-critical applications due to their high speed, durability, and energy efficiency. However, the inherent characteristics of NAND flash memory, such as block-level erasure and data fragmentation, necessitate frequent garbage collection (GC) operations to reclaim storage space. These operations, while essential, introduce significant performance overhead, particularly in modern SSD controllers (SSDCs) that utilize network-on-chip (NoC) architectures. In such architectures, GC requires substantial data transfer over interconnects for error correction, leading to increased latency and reduced throughput. This paper presents a novel GC scheme designed to minimize latency in NoC-based SSDCs. Unlike conventional methods that unconditionally transfer data for error correction, the proposed approach selectively determines the data transfer path based on the presence of errors. By leveraging the low error probability of NAND flash memory, this scheme avoids unnecessary data traversal across the interconnect, significantly reducing GC overhead. A hardware implementation using task queues ensures efficient parallelism without disrupting other operations. The experimental results demonstrate that the proposed scheme improves SSD performance across various real-world workloads, achieving up to a 26.9% reduction in average latency and a 50.0% reduction in peak latency compared to traditional GC methods. These findings highlight the potential of optimizing data traversal paths in NoC architectures, providing a scalable solution for enhancing SSD performance for diverse applications.

Keywords: solid-state drive; garbage collection

Citation: Ahn, S.; Im, D.; You, D.; Hong, Y. High-Performance Garbage Collection Scheme with Low Data Transfer Overhead for NoC-Based SSDC. *Electronics* **2024**, *13*, 4838. https://doi.org/10.3390/electronics13234838

Academic Editor: Luis Gomes

Received: 11 October 2024
Revised: 4 December 2024
Accepted: 6 December 2024
Published: 7 December 2024

1. Introduction

Solid-state drives are increasingly favored for applications demanding rapid data retrieval and processing, thanks to their low access times and energy efficiency compared to traditional storage devices. In addition to offering faster performance, SSDs exhibit higher reliability due to their resistance to physical damage and longer lifespan. These advantages have driven the widespread adoption of SSDs across diverse domains, including cloud computing, data centers, and consumer electronics [1–3].

However, the performance and reliability of SSDs are heavily influenced by how efficiently they manage the storage medium, NAND flash memory. A major challenge arises from the inherent characteristics of NAND flash memory, which requires data to be erased in blocks before it can be rewritten. Over time, this leads to fragmentation, where valid data are scattered across blocks, reducing the available storage space. To address this issue, SSDs rely on a process called garbage collection, which consolidates valid data and reclaims storage space by erasing blocks with obsolete data [4,5]. While GC improves storage utilization, it comes at the cost of increased internal data transfer and latency, significantly degrading SSD performance, especially under high workloads [6,7].

This trade-off between storage efficiency and performance is a critical bottleneck in SSD design. Frequent GC operations not only consume significant bandwidth but also

increase the wear on NAND flash memory, shortening its lifespan. Researchers have explored strategies to optimize GC by reducing its frequency or minimizing the amount of data transferred during the process. For example, prioritizing blocks with the highest number of invalid pages or predicting the lifespan of data can reduce unnecessary GC operations. However, these methods face challenges such as limited accuracy in data lifespan prediction or inefficiencies in managing data transfer during GC [8–12].

A more recent trend in SSD architecture involves the adoption of network-on-chip interconnects in SSD controllers. NoC-based architectures provide scalable, high-throughput communication between SSD components, addressing the limitations of traditional bus systems. However, the use of NoC interconnects introduces significant latency during GC operations, as data often need to traverse multiple routing slices for error correction before being written back to NAND flash memory. This delay is particularly pronounced when applying low-density parity-check (LDPC) error correction coding, which is computationally intensive and typically performed on the host-side CPU [13–16]. Despite the growing use of NoC-based SSDCs, previous research has largely overlooked how interconnect latency impacts GC performance.

To address this gap, we propose a novel GC scheme designed specifically for NoC-based SSDCs. Unlike conventional approaches that unconditionally transfer data from flash memory to the CPU for error correction during GC and write the ECC-checked data back to flash memory [17,18], our scheme selectively determines the data transfer path based on the presence of errors. By leveraging the low error probability of NAND flash memory, the proposed method minimizes unnecessary data traversal across the interconnect, reducing latency and improving SSD performance. Our contributions are as follows:

1. We introduce a selective copy-back mechanism that prioritizes shorter data paths when no errors are detected during GC, significantly reducing interconnect delays.
2. We present a hardware implementation using task queues to efficiently manage the proposed selective data flow, enabling parallelism with minimal overhead.
3. We evaluate the proposed scheme under real-world workloads, demonstrating up to a 26.9% improvement in average latency and a 50.0% reduction in peak latency compared to conventional methods.

By tackling the latency challenges associated with NoC interconnects in SSD controllers, our approach offers a scalable solution that enhances SSD performance while maintaining storage efficiency. To the best of our knowledge, there is currently no academic research that specifically addresses NoC-based SSD controllers.

The remainder of this paper is organized as follows: Section 2 provides background on SSD architecture, GC mechanisms, and NoC-based SSDCs. Section 3 details our proposed GC scheme and its hardware implementation. Section 4 presents the experimental setup and results, and Section 5 concludes with a discussion of implications and future directions.

2. Background

To understand the motivation behind our proposed work, it is important to first discuss the evolution of SSD architectures. Therefore, this section provides an overview of conventional SSD architecture, the concepts of GC and ECC, and the introduction of NoC-based SSD controllers.

2.1. Conventional SSD Organization

Traditional SSD architectures typically comprise a CPU, DRAM, and multiple NAND flash memory units, referred to as channels. As shown in Figure 1, the CPU manages read and write requests from the host and temporarily stores data in DRAM. Given that the read and write speeds of NAND flash memory are significantly slower than those of the CPU and DRAM, parallelism is employed through multiple NAND flash channels.

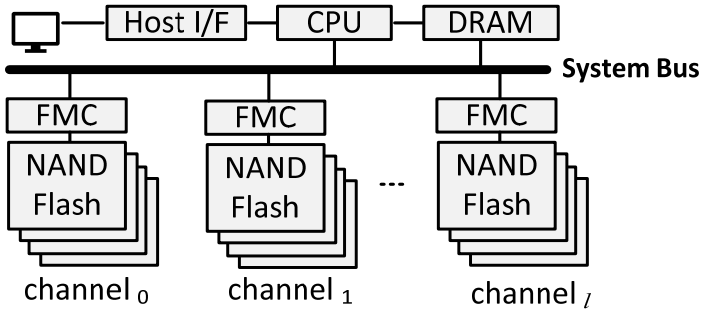


Figure 1. Conventional SSD organization (FMC: flash memory controller).

Historically, these components have been interconnected using a system bus, which serves as the communication highway. However, this bus-based approach comes with several inherent limitations. Buses are constrained in terms of bandwidth, which can lead to bottlenecks when multiple components attempt to communicate simultaneously. As the number of NAND flash channels increases, the bus must handle more requests, which exacerbates latency issues and limits scalability. This results in inefficient data transfer, particularly during critical operations such as GC, where data must be moved frequently and rapidly.

Additionally, the bus architecture creates contention among different components, leading to delays that can severely impact overall SSD performance. The rigid addressing and communication methods of bus systems hinder the flexibility required for modern SSD workloads that demand high throughput and low latency. Thus, while conventional bus-based SSD controllers were suitable for early SSD designs, they struggle to meet the performance demands of contemporary applications.

In contrast, the adoption of interconnect-based architectures, such as network-on-chip, offers significant advantages in handling the increasing complexity and performance requirements of SSDs. NoC systems facilitate better parallelism and scalability, allowing for non-blocking communication among components, which mitigates the limitations associated with traditional bus systems.

2.2. Garbage Collection in SSDs

The erase operation in flash memory differs significantly from that in other data storage types, such as hard disks. In flash memory, each memory cell must be initialized before future writes can occur, and erasure must be performed at the block level due to the physical structure of the memory. A key aspect of the erase operation is that it is executed in units of blocks; if a page within a block is marked for erasure, it is considered invalid until the entire block has been erased. Furthermore, overwriting data in flash memory requires a preceding erase operation because of the device's physical characteristics. When a page in a block needs to be updated, the existing page is invalidated, and the new data are written to another valid page. This operational mechanism implies that SSDs can contain many blocks with numerous invalid pages, leading to significant storage inefficiencies. To reclaim this storage space, blocks with a high number of invalid pages undergo a process known as GC. During this process, valid data from the target block is copied to another block—a procedure referred to as "copy-back"—after which the entire original block is erased.

Figure 2 illustrates a snapshot of SSD where GC is in progress. $Block_1$ in $channel_0$ is full of invalid pages except one valid page which is $page_1$. This means a significant waste of storage, and it is desirable to initialize $block_1$ after moving $page_1$ in another place such as $page_1$ in $channel_1$ as indicated by the red arrow. This process is called copy-back. Once the copy-back operations are complete for the GC target block, an erase operation is applied to $block_1$ in $channel_0$ to reclaim the storage.

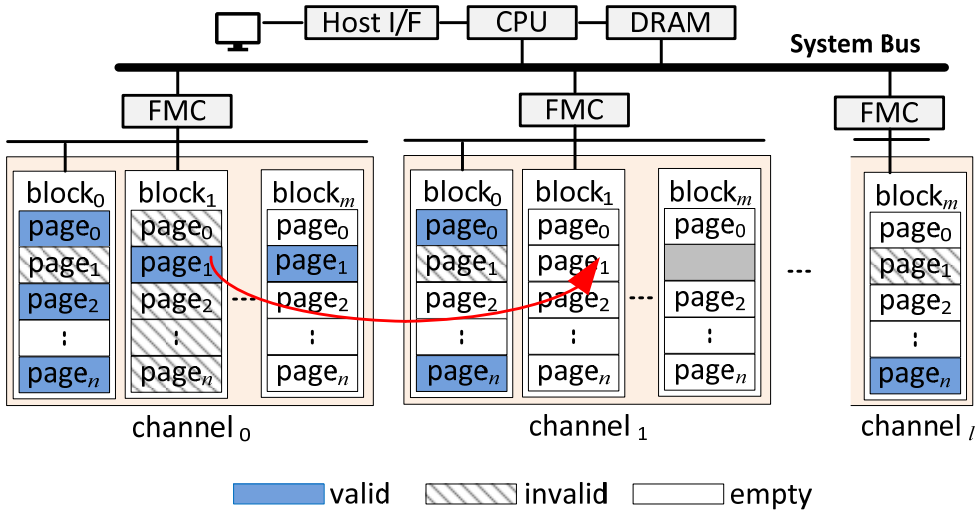


Figure 2. Illustration of garbage collection in SSD.

While the GC increases the available storage space, it does so at the cost of SSD execution time. That is, frequent copy-back operations lead to a significant increase in the write and read operations to the flash memory requested by the host. Also, the frequent GC leads to a reduction in the device's overall lifespan because NAND flash memory cells have a relatively short lifetime compared to other types of storage.

2.3. NoC-Based SSD Controller

As SSD capacities and performance demands continue to grow, traditional bus-based SSD architectures face scalability and bandwidth limitations. To address these challenges, modern SSD controllers increasingly adopt NoC architectures. NoC-based SSDCs consist of a CPU core, SRAM, DRAM, and multiple NAND flash memories connected by an interconnect as shown in Figure 3. This interconnect functions as a communication backbone within the chip, providing scalability and enabling parallelism across multiple NAND flash memory channels.

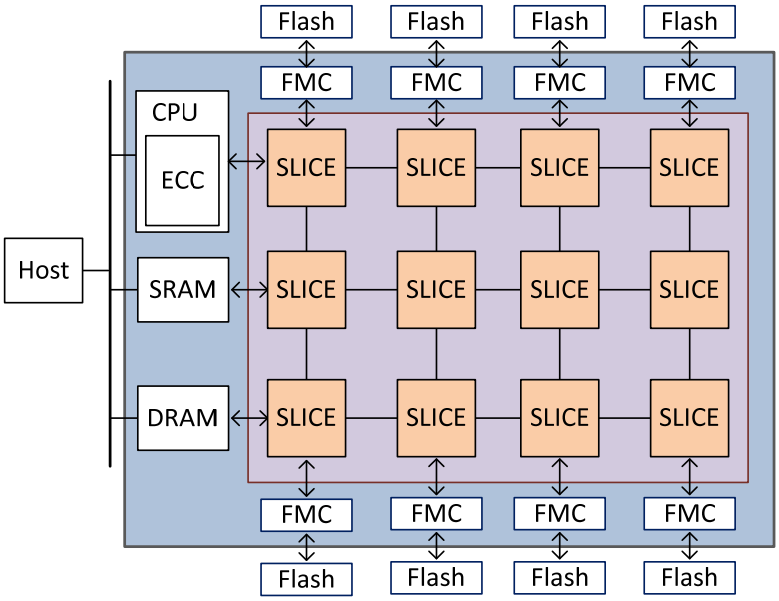


Figure 3. NoC-style SSD controller architecture.

The NoC interconnect is composed of routing slices, each of which serves as a buffer with routing capabilities. Data communication occurs as packets traverse these slices, and the total latency depends on the number of slices along the path. For GC operations, this means valid data from a NAND flash block must travel through multiple slices to reach the host-side CPU, where error correction is performed. Once processed, the data are routed back through the interconnect to its destination NAND flash memory. This round-trip significantly increases latency due to the cumulative delays introduced by slice traversal.

3. Proposed Garbage Collection Scheme

As highlighted in the previous section, the evolution of SSD controllers has significantly influenced performance, particularly through the adoption of interconnect-based architectures. While these architectures provide advantages such as higher throughput and improved timing closure, they also introduce significant latency challenges during GC operations. This section outlines the core concepts behind our proposed GC scheme, which is specifically designed to mitigate interconnect-induced latency.

3.1. Adaptive Data Path Selection for Garbage Collection Efficiency

In NoC-based SSD controllers, the GC process involves several key steps:

1. Target Block/Page Identification: Blocks with a high number of invalid pages are selected for GC. Valid pages from these blocks are identified and copied to a temporary buffer in the flash memory controller (FMC).
2. Error Correction: Valid data in the FMC buffer are routed via the NoC interconnect to the CPU for error correction using the low-density parity-check algorithm.
3. Data Write-Back: Once corrected, the data are written to an empty page in the target NAND flash block.

This process incurs significant latency due to two main factors:

- Round-Trip Traversal: Each copy-back operation involves two interconnect traversals—one to the CPU for error correction and another to return the data to the target NAND flash memory.
- Slice-Based Delays: Data packets pass through multiple interconnect slices, with each slice adding latency based on the interconnect size and the number of active channels.

Existing GC optimization methods primarily focus on reducing GC frequency or prioritizing blocks with higher invalid page counts. While beneficial in some scenarios, these approaches fail to address the core issue of interconnect latency during GC operations. Similarly, techniques that pre-empt GC during high-priority host requests overlook inefficiencies in the data transfer process, leaving potential performance gains untapped.

The proposed GC scheme effectively addresses these limitations by selectively optimizing the data flow paths based on real-time error detection, as illustrated in Figure 4. When no errors are present, the CPU is bypassed, allowing data to flow directly from the FMC buffer to the NAND flash memory, thus significantly reducing traversal time. The process is outlined as follows:

1. Copy-Back Operation: During the copy-back process, valid pages from the GC target block are temporarily stored in the FMC buffer, as presented in Figure 4a.
2. Error Analysis: Data packets are routed to the CPU via the NoC for error analysis by the ECC unit, as presented in Figure 4a:
    - No Errors Detected: If no errors are found, the data stored in the FMC buffer are routed directly back to an available location in the NAND flash memory, as shown in Figure 4b.
    - Errors Detected: If errors are detected, the ECC unit within the CPU performs the necessary corrections, after which the corrected data are routed back to the NAND flash memory, as shown in Figure 4c.

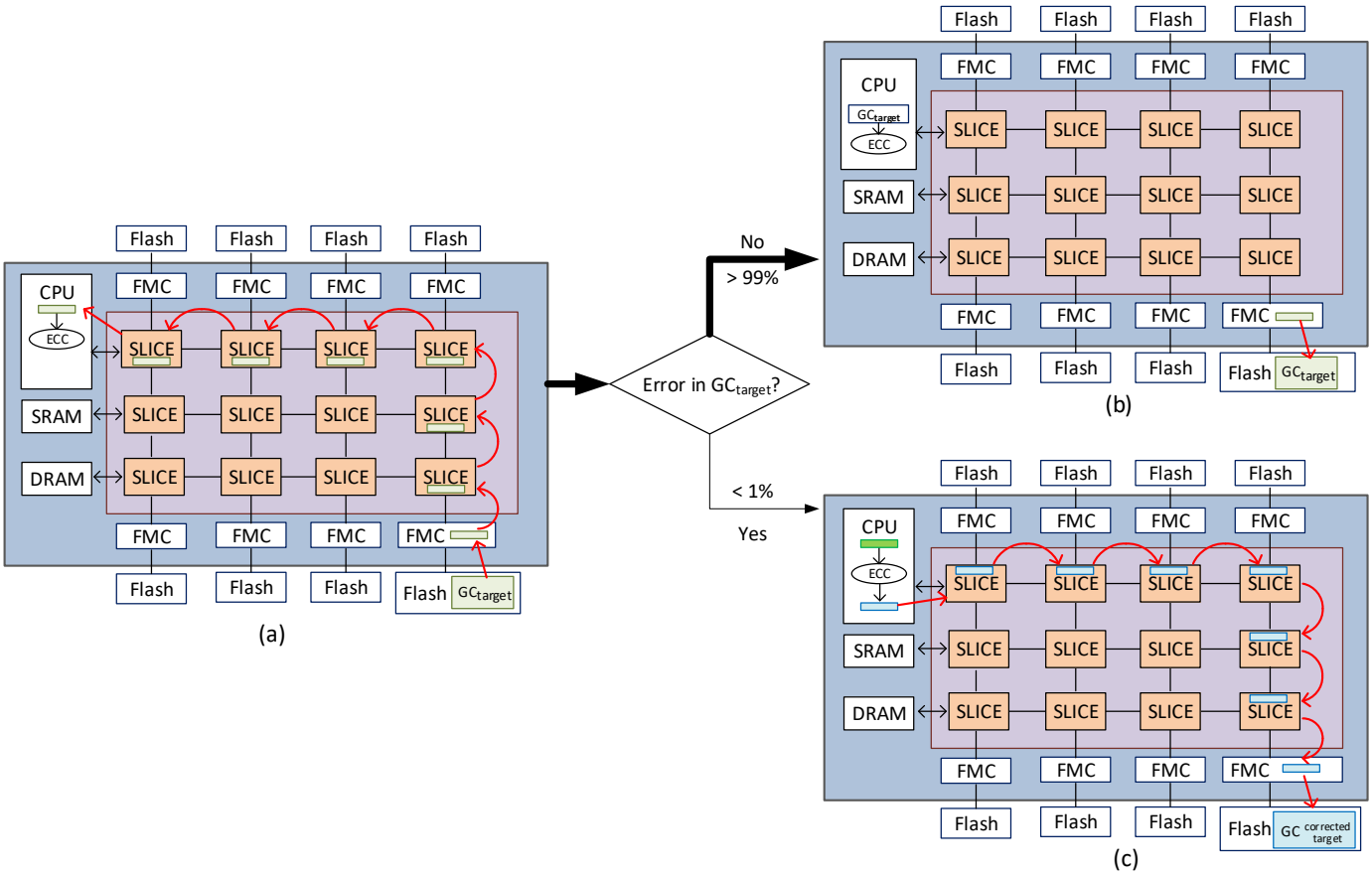


Figure 4. Copy-back data path selection by the proposed GC scheme.

By eliminating unnecessary round-trip traversal for error-free data, this approach significantly reduces latency while maintaining data reliability. Hardware-implemented task queues manage the selective path optimization seamlessly.

This method leverages the typically low error rates of NAND flash memory, making full round-trip traversal for LDPC correction unnecessary in most cases. By dynamically analyzing error conditions during the copy-back operation, the scheme minimizes traversal length without compromising data integrity.

Key innovations include the following:

1. Selective Data Path Optimization: A dynamic mechanism determines the shortest path for copy-back operations. Error-free data bypasses the CPU and is directly written back to NAND flash memory, reducing interconnect latency by roughly half.
2. Error-Adaptive Control: For data with errors, full round-trip traversal ensures compatibility with standard LDPC correction processes, preserving reliability.

Unlike conventional GC schemes that require unconditional data transfers to the CPU for error correction, this method adapts data flow based on error conditions. By aligning with the parallelism inherent in NoC architectures, the proposed scheme reduces interconnect delays and enhances overall SSD performance.

The originality of this approach lies in addressing interconnect latency by exploiting low error probabilities, a factor often overlooked in previous research. The result is a scalable solution that mitigates latency challenges in NoC-based SSD controllers while ensuring data reliability and efficient storage management.

3.2. Efficient Data Handling Through Task Queue Architecture

The proposed GC scheme intelligently governs the flow of data to be copied back, requiring a precise hardware mechanism for its implementation. A defining characteristic

of multi-channel SSDs is their capacity to perform multiple tasks simultaneously—such as processing host-oriented read and write requests while executing GC tasks—thereby maximizing overall SSD throughput. In prior work, we introduced an efficient method for managing multiple tasks in bus-based SSD controllers using a task queue [10].

To effectively support parallelism in NoC-based SSD controllers, each flash memory channel is paired with a dedicated task queue that organizes all requests for that channel, as illustrated in Figure 5. Each channel is associated with four distinct task queues: read requests from the host (R_{host}), read requests for GC (R_{gc}), write requests from the host (W_{host}), and write requests for GC (W_{gc}). Tasks in these queues are processed sequentially by dedicated hardware that operates independently for each channel.

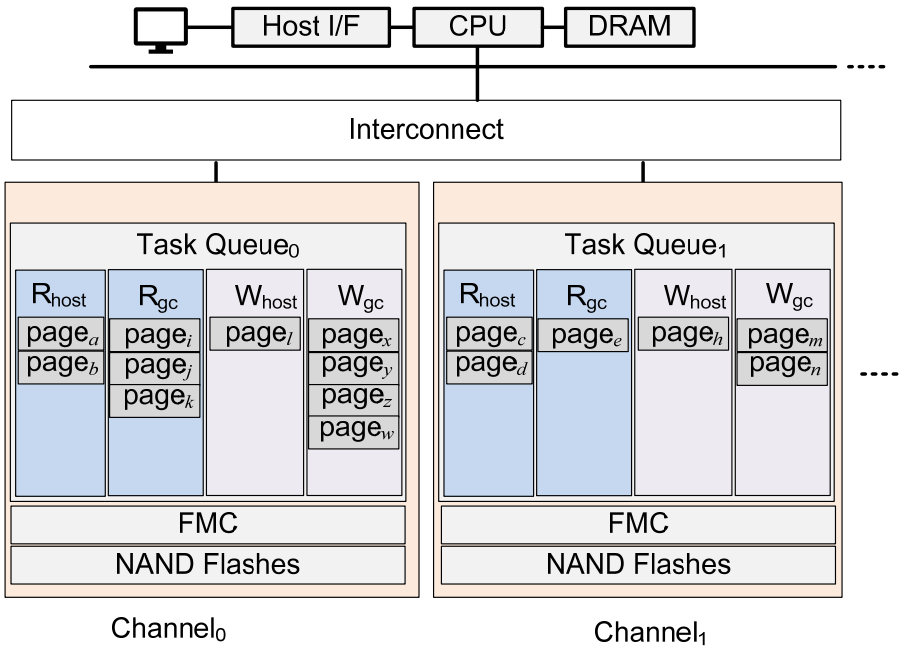


Figure 5. Configuration of dual task queue.

The criteria for enqueueing tasks in the W_{gc} (write requests for GC) queue are determined by the analysis results from the ECC unit. If the ECC unit detects errors in the page data, the corrected data will be stored in an empty page of flash memory, and the corresponding request will be added to the W_{gc} queue. Conversely, if the ECC unit confirms that the page data are error-free, the data temporarily held in the FMC buffer will be written back to an available page in the flash memory, and no new task will be enqueued in the W_{gc} queue.

Key innovations include the following:

1. Efficient Task Queue Integration: The approach integrates with task queue architectures in NoC-based SSD controllers, enabling smooth implementation alongside host read and write operations.
2. Integration with ECC Analysis: The task queue enqueues GC write requests (W_{gc}) only if errors are detected and corrected. Otherwise, error-free data from the FMC buffer are directly written back without generating additional W_{gc} tasks.

4. Evaluation

4.1. Experimental Setup

We first extracted trace files from an SSD for various workloads using tools such as ftrace [19], blktrace, and blkparse [20]. Next, we modeled an eight-channel SSD system in C and measured its performance across these workloads.

For trace extraction, we used a 1 TB Samsung T5 Portable SSD, running on a host system with an Intel® Core™ i7-9700K CPU and 32 GB of RAM. The operating system was Red Hat Enterprise Linux Workstation release 7.9, with Linux kernel version 3.10.0-1160.59.1.el7.x86_64. Table 1 summarizes the key features of our SSD model configuration, which are adapted from previous studies on GC [4,21–24].

Table 1. Parameters of SSD configuration.

Features	Value
Number of Channels	8
Planes per Channel	4
Blocks per Plane	512
Pages per Block	64
Page Size (KB)	4

Table 2 summarizes the delay parameters for one page read or write with the SRAM, the flash memory, and the interconnect used in our SSD model. For SRAM, we extracted delay information from single-port 16-bit SRAM from a Xilinx Spartan-7 FPGA (model xc7s100fgga676-1), manufactured by AMD, located in Santa Clara, California, USA, using Xilinx Vivado 2021.2. The delay information of NAND flash memory is from Micron MT29F4G08ABADAH4 NAND flash model [25].

Table 2. Delay of SSD components for one page read or write.

Delay	Value
SRAM read (µs)	12
SRAM write (µs)	12
NAND read (µs)	25
NAND write (µs)	200
NAND erase (µs)	700
Interconnect (µs)	714

In order to identify a realistic interconnect delay, we designed a 16 × 8 interconnect using Verilog HDL for the eight-channel SSD and measured the timing from the synthesis results. Each task queue for a channel requires an interconnect port, similar to the DMA controllers for NAND flash memories and DRAMs, which is why we chose an interconnect dimension of 16 × 8. We synthesized the Verilog HDL targeting the Xilinx Spartan-7 FPGA from which we extracted SRAM delay parameters. According to the synthesis report, the critical path delay between two neighboring slices is 87 ns.

To calculate the total interconnect delay for transmitting one page between the host and NAND flash memory, we use the following formula:

$$1\ page\ interconnect\ delay = \left(\frac{2^{15}}{4} + 13.5 - 1\right) \times 87\ ns, \quad (1)$$

Here, $(2^{15}/4)$ represents the number of movements required to transmit one page. Since a page is 4 KByte and 1 byte equals 8 bits, a single page contains 2^{15} bits. By dividing one page by the data transmission unit of 4 bits, we obtain the number of movements required to transmit one page. Additionally, 13.5 is the average number of slices a data transmission unit passes through, calculated by considering all possible cases. The reason for subtracting one at the end is that the actual number of movements is one less than the number of slices. Therefore, the total delay for transmitting one page across the interconnect is 8204.5 × 87 ns, which equals approximately 714 µs.

As stated before, the bit error rate of MLC ranges from 1×10^{-9} to 1×10^{-4}. In our simulation, the probability that the GC target page has errors is set to 1×10^{-4}.

4.2. Trace File Analysis

Table 3 shows the profiles of the trace files extracted from six different workloads used in our experiment. These trace files were extracted by installing a database and benchmark programs, such as MySQL based on sysbench [26,27], Yahoo Cloud Serving Benchmark (YCSB) family [28], Cassandra [29], MongoDB [30], RocksDB [31], and Phoronix Test Suite [32] family SQLite and Dbench.

Table 3. Characteristics of trace files.

Workloads	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Read Ratio (%)	80.2	0.0	0.0	0.0	36.9	50.5
Write Ratio (%)	19.8	100.0	100.0	100.0	63.1	49.5
Seq. Request (%)	56.8	13.2	9.8	18.7	18.7	9.9
Rand. Request (%)	43.2	86.8	90.2	81.3	81.3	90.1
Aver. Read Size (KB)	70.5	4.0	0.0	4.0	6.4	4.9
Aver. Write Size (KB)	103.7	7.6	5.8	4.9	4.7	4.5

4.3. Experimental Results

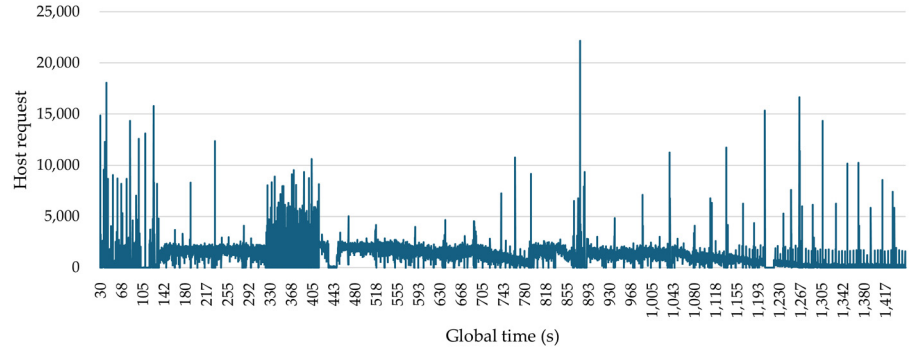
We conducted simulations using our C model of the SSD, along with trace files extracted from various workloads. To evaluate the effectiveness of the proposed GC scheme, we measured the read and write latency of the SSD for each workload. While we acknowledge that most commercial SSDCs now use interconnects, to the best of our knowledge, no academic research has yet addressed this approach. Therefore, we compared the performance of our proposed method with a conventional approach that uses an unconditional copy-back scheme, as explained in this paper.

Table 4 summarizes the average latency for both the conventional GC scheme, which unconditionally copies back ECC results, and our proposed scheme, which replaces this process with a local copy-back. As shown in Table 4, the proposed GC scheme improved average latency by up to 26.9%, demonstrating the effectiveness of the proposed scheme. It is important to note that the MySQL, Dbench, and SQLite workloads do not include any read operations, so corresponding measurement data are not available.

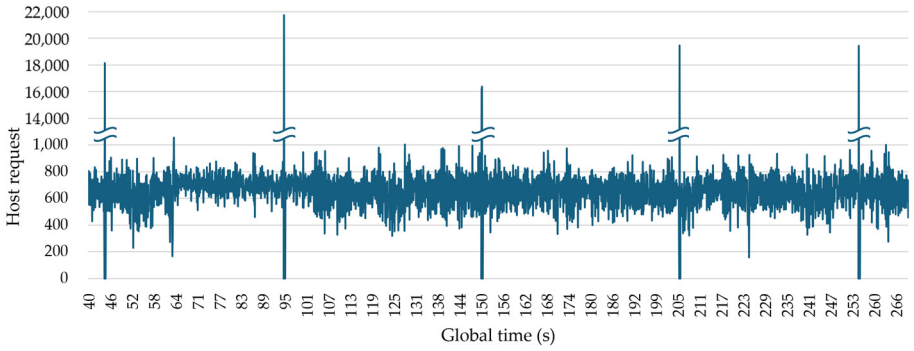
Table 4. Average latency of read/write requests.

Avg. Read Latency (ms)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Conventional	1.729	-	-	-	0.826	0.856
Proposed	1.712	-	-	-	0.824	0.855
Avg. Write Latency (ms)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Conventional	20.714	8.981	1.159	1.151	1.118	1.467
Proposed	15.137	6.811	1.076	1.097	1.067	1.461
Proposed/Conventional Improvement Ratio (%)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Avg. Read latency	0.983	-	-	-	0.242	0.117
Avg. Write latency	26.924	24.162	7.161	4.692	4.562	0.409

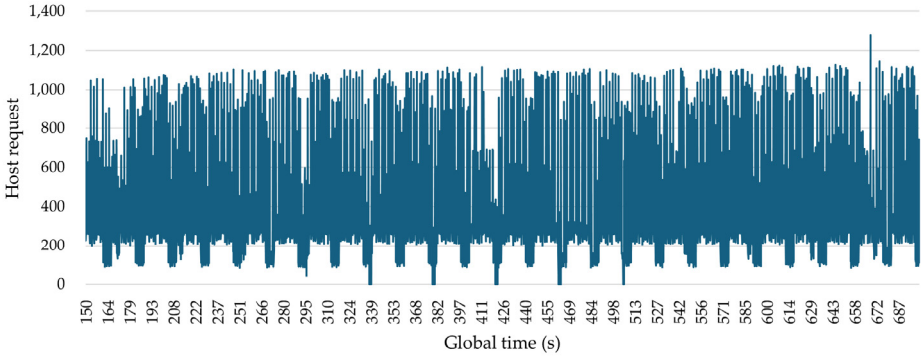
As shown in Table 4, the average write latency improvements vary across workloads, with Cassandra showing the most significant improvement, followed by Dbench, MySQL, SQLite, MongoDB, and RocksDB. To understand the reasons behind these differences, we analyzed the host request patterns for each workload, as illustrated in Figure 6. Cassandra, Dbench, MySQL, and RocksDB workloads display highly dynamic request patterns, and SQLite and MongoDB workloads show relatively static host request patterns.



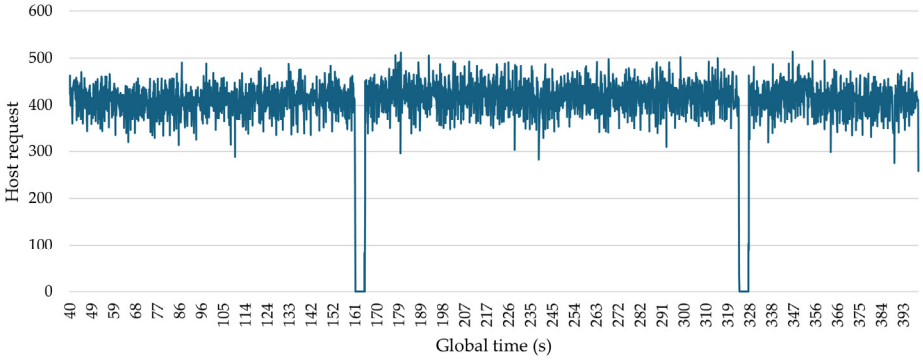
(**a**) Cassandra



(**b**) Dbench

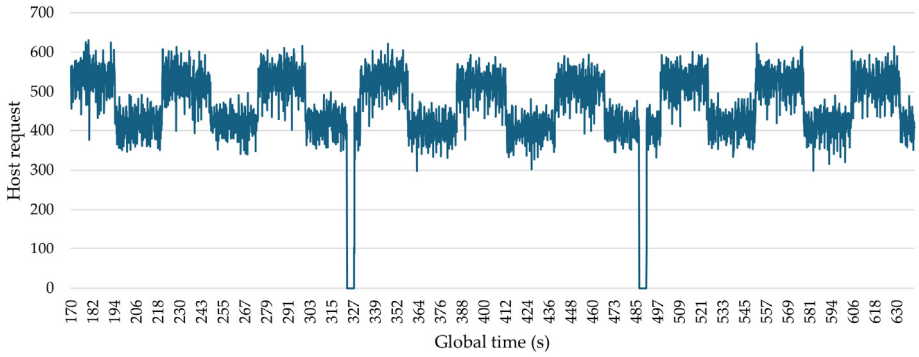


(**c**) MySQL

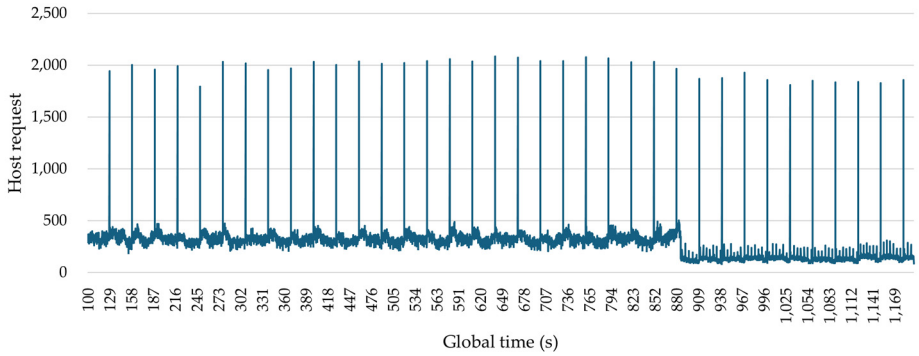


(**d**) SQLite

Figure 6. *Cont*.



(**e**) MongoDB

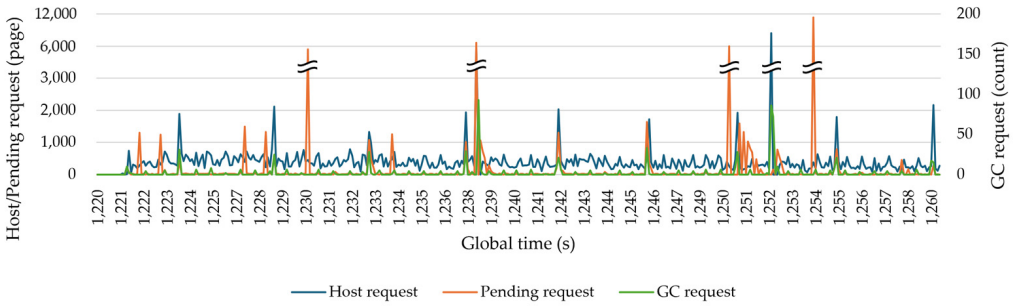


(**f**) RocksDB

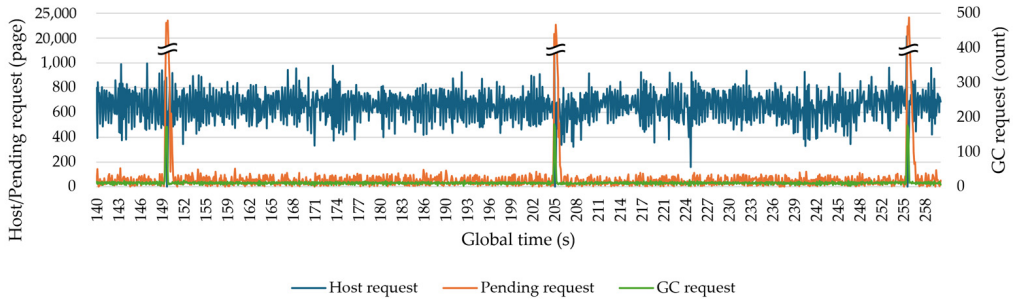
Figure 6. Host request patterns.

To gain deeper insights, we measured several key metrics: the number of read/write requests from the host, the number of pending requests, and the number of requests generated by the GC process. Figure 7 presents the results across all workloads. Cassandra exhibits a highly dynamic pattern of read/write requests, with occasional spikes that lead to a significant increase in pending requests. In contrast, MongoDB follows a more periodic pattern, where moderate request volumes are generated over a period, followed by intervals of inactivity. This cycle repeats regularly. Because the request load in MongoDB remains well within the SSDC's capacity, the number of pending requests remains minimal. In other words, read/write requests from the host are promptly serviced as they are queued in the task queues when using MongoDB. The proposed scheme effectively reduces the GC load and has a more pronounced impact on improving SSD latency, particularly when the host request pattern is more dynamic.

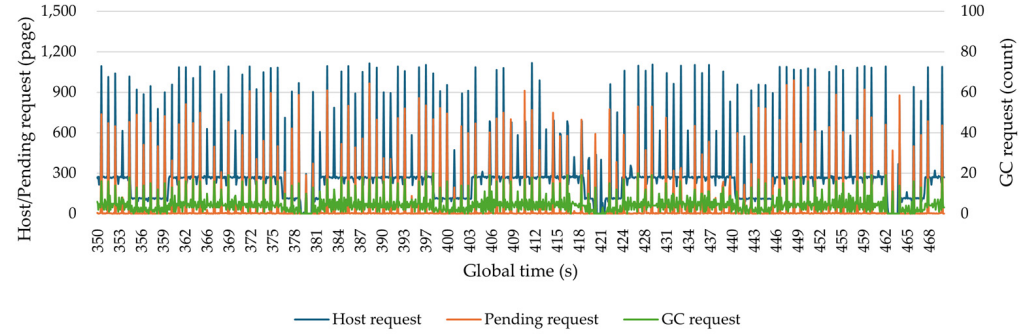
Interestingly, the proposed GC method does not lead to significant improvements in latency for RocksDB benchmark, which shows occasional surges of host requests. This is because the number of copy-back operations in RocksDB is notably lower compared to the other benchmarks as shown in Figure 8. In RocksDB's case, journal data occupy almost 70% of the entire data. Journal data refer to the log information used by Linux file systems to ensure data integrity and recoverability in the event of a system crash or power failure. The total amount of journal data is fixed, and old journal data are periodically overwritten by new journal data. Therefore, GC target page that stores journal data only needs erase operation without copy-back operation. This is why RockDB has a very low copy-back count and the proposed scheme does not show significant improvement.



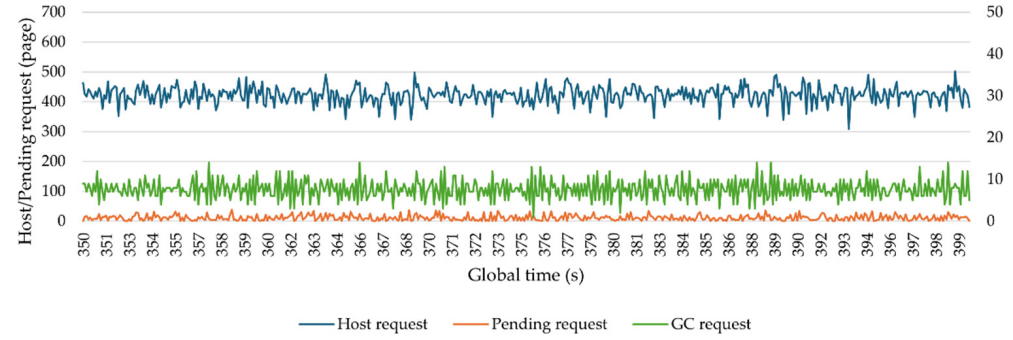
(**a**) Cassandra



(**b**) Dbench

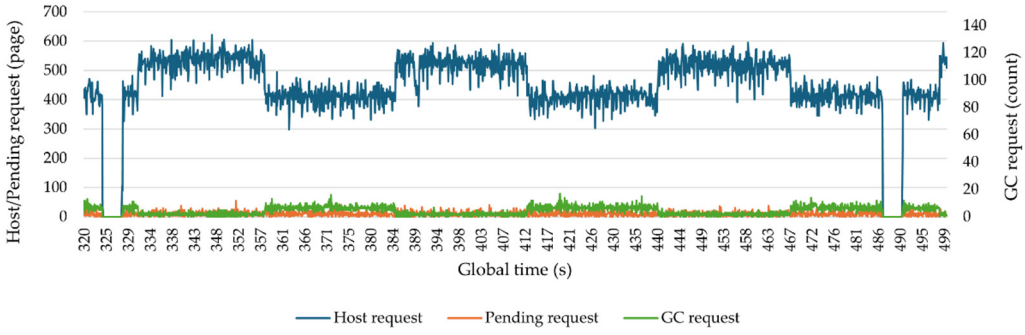


(**c**) MySQL

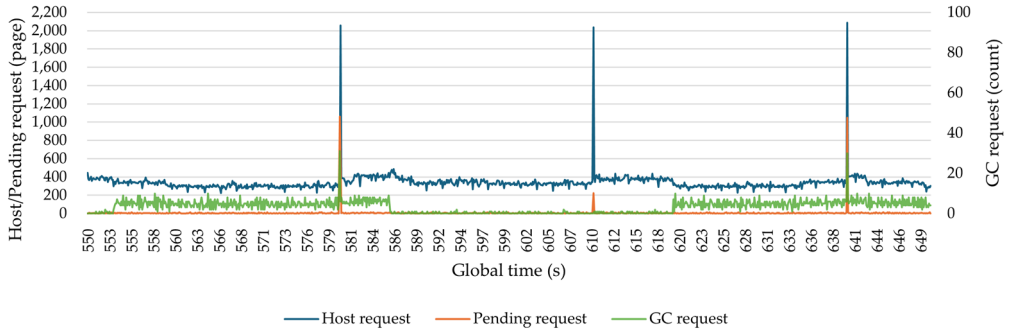


(**d**) SQLite

Figure 7. *Cont.*



(**e**) MongoDB



(**f**) RocksDB

Figure 7. Number of host requests, GC requests, and pending requests.

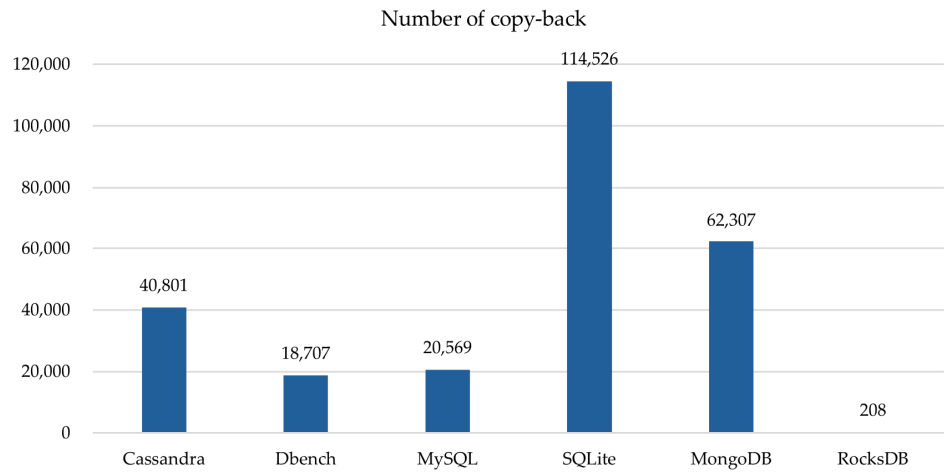


Figure 8. Number of copy-backs.

However, a higher copy-back count does not always correspond to a greater performance improvement. For example, SQLite exhibits a significantly higher copy-back count than MySQL, yet their improvement rates are similar. As shown in Figure 7d, SQLite experiences minimal pending requests due to GC, meaning that host requests are typically processed immediately. As a result, the proposed GC scheme has a relatively small impact on SQLite's performance, which is reflected in the lower improvement rate shown in Table 4. In contrast, Figure 7c illustrates that MySQL frequently experiences pending requests due to GC, with a higher volume of such requests. Consequently, the latency improvements from our algorithm are more pronounced, as they apply to host requests that are delayed by GC. This explains why MySQL, despite having fewer copy-backs, achieves a performance improvement similar to SQLite's.

An interesting observation regarding the Cassandra workload is that the performance improvement is significantly more pronounced for write requests. To investigate this further, Figure 9 presents a graph that separates Cassandra's read and write requests. While read requests follow a relatively monotonic pattern, write requests are highly irregular, with sudden surges in volume. This suggests that the accumulation of pending requests in Cassandra, as observed earlier, is primarily driven by these sharp increases in write requests, while read requests remain stable and do not exhibit similar fluctuations.

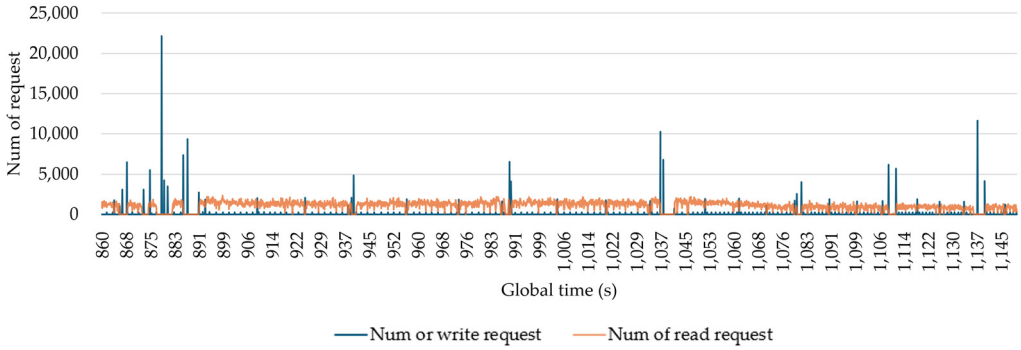


Figure 9. Read and write request patterns of Cassandra.

Table 5 compares the peak latency of the conventional GC scheme with that of the proposed GC scheme. As shown, the proposed scheme reduces peak latency by up to 50.0%. Similar to the pattern observed for average latency, the improvement in maximum latency correlates with the irregularity of host request patterns. When listing the workloads by the extent of improvement, the order is as follows: Cassandra, Dbench, MySQL, SQLite, MongoDB, and RocksDB. Excluding RocksDB, which has an exceptionally low copy-back count, this suggests that greater irregularity in host request patterns leads to a more significant improvement in peak latency.

Table 5. Peak latency of read/write requests.

Max. Read Latency (ms)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Conventional	181.257	-	-	-	2.832	21.686
Proposed	90.625	-	-	-	2.474	20.625
Max. Write Latency (ms)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Conventional	1055.123	977.507	29.450	3.776	3.602	51.816
Proposed	758.686	836.451	25.372	3.341	3.200	50.637
Proposed/Conventional Improvement Ratio (%)	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Max. Read latency	50.002	-	-	-	12.641	4.893
Max. Write latency	28.095	14.430	13.847	11.520	11.160	2.275

To account for operation uncertainties in our experimental results, we conducted a total of five trials for our algorithm, calculating the standard deviation for the experimental results of each workload. As shown in Table 6, the standard deviation for average latency is nearly zero. In contrast, there is a tendency for a larger standard deviation in maximum latency.

Table 6. Standard deviation of read/write latency.

Standard Deviation	Cassandra	Dbench	MySQL	SQLite	MongoDB	RocksDB
Avg. read latency	0.003	-	-	-	0.000	0.000
Avg. write latency	0.430	0.395	0.002	0.001	0.000	0.001
Max. read latency	1.758	-	-	-	0.057	0.676
Max. write latency	6.893	6.227	1.199	0.117	0.133	1.053

5. Conclusions

In this paper, we have addressed the latency issues associated with GC operations in SSDs that utilize NAND flash memory, particularly in NoC architectures. Our proposed GC scheme intelligently minimizes unnecessary data transfers by selectively determining the data flow based on the presence of errors, thereby reducing the overhead typically incurred during GC.

Through extensive experimentation, we demonstrated that our approach can significantly enhance SSD performance, achieving an average latency improvement of up to 26.9% and a peak latency reduction of up to 50.0%, particularly benefiting workloads characterized by dynamic request patterns. The insights gained from this study underscore the importance of optimizing data traversal paths in the presence of error correction needs, which has been largely overlooked in previous research.

Our results were based on an eight-channel SSD model with 64 pages per block. Increasing the number of channels would expand the interconnect size and increase the number of slices, which, in turn, would lead to a greater improvement from the proposed GC scheme. This is because the scheme effectively reduces the impact of interconnect delays. In contrast, as NAND flash memory technology advances, manufacturers focus on increasing memory density, which often results in a higher number of pages per block. A larger number of pages per block is expected to reduce the frequency of GC, and as a result, we anticipate a smaller improvement factor from the proposed GC scheme. Given these trends, it would be valuable for future work to explore how the proposed GC scheme can be adapted to accommodate these developments in SSD architecture.

Author Contributions: Conceptualization, S.A., D.I., D.Y. and Y.H.; methodology, S.A., D.I., D.Y. and Y.H.; software, S.A., D.I. and D.Y.; validation, S.A., D.I., D.Y. and Y.H.; formal analysis, S.A., D.I., D.Y. and Y.H.; investigation, S.A., D.I. and D.Y.; resources, S.A., D.I. and D.Y.; writing—original draft preparation, S.A., D.I., D.Y. and Y.H.; writing—review and editing, S.A., D.I., D.Y. and Y.H.; visualization S.A., D.I., D.Y. and Y.H.; supervision, Y.H.; project administration, Y.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Narayanan, D.; Thereska, E.; Donnelly, A.; Elnikety, S.; Rowstron, A. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In Proceedings of the 4th ACM European Conference on Computer Systems, Nuremberg, Germany, 1–3 April 2009; pp. 145–158.
2. Deng, Y. What Is the Future of Disk Drives, Death or Rebirth? *ACM Comput. Surv. (CSUR)* **2011**, *43*, 1–27. [CrossRef]
3. Micheloni, R.; Marelli, A.; Eshghi, K.; Wong, G. SSD Market Overview. In *Inside Solid State Drives (SSDs)*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 1–17.
4. Guo, J.; Hu, Y.; Mao, B.; Wu, S. Parallelism and Garbage Collection Aware I/O Scheduler with Improved SSD Performance. In Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, FL, USA, 29 May–2 June 2017; pp. 1184–1193.
5. Bux, W.; Iliadis, I. Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives. *Perform. Eval.* **2010**, *67*, 1172–1186. [CrossRef]

6. Google. Google: Taming the Long Latency Tail—When More Machines Equals Worse Results. Available online: http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html (accessed on 12 March 2012).
7. Dean, J.; Barroso, L.A. The Tail at Scale. *Commun. ACM* **2013**, *56*, 74–80. [CrossRef]
8. Gupta, A.; Pisolkar, R.; Urgaonkar, B.; Sivasubramaniam, A. Leveraging Value Locality in Optimizing NAND Flash-Based SSDs. In Proceedings of the FAST 2011, San Jose, CA, USA, 15–18 February 2011; pp. 91–103.
9. Yadgar, G.; Yaakobi, E.; Schuster, A. Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes. In Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST 15), Santa Clara, CA, USA, 16–20 February 2015; pp. 257–271.
10. Ae, J.; Hong, Y. Efficient Garbage Collection Algorithm for Low Latency SSD. *Electronics* **2022**, *11*, 1084. [CrossRef]
11. Zhang, Q.; Li, X.; Wang, L.; Zhang, T.; Wang, Y.; Shao, Z. Lazy-RTGC: A Real-Time Lazy Garbage Collection Mechanism with Jointly Optimizing Average and Worst Performance for NAND Flash Memory Storage Systems. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2015**, *20*, 1–32. [CrossRef]
12. Cheng, W.; Wang, X.; Zhang, S.; Li, J. Lifespan-Based Garbage Collection to Improve SSD's Reliability and Performance. *J. Parallel Distrib. Comput.* **2022**, *164*, 28–39. [CrossRef]
13. Kim, J.; Kang, S.; Park, Y.; Kim, J. Networked SSD: Flash Memory Interconnection Network for High-Bandwidth SSD. In Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), Chicago, IL, USA, 1–5 October 2022; pp. 388–403.
14. Kim, J.; Jung, M.; Kim, J. Decoupled SSD: Rethinking SSD Architecture Through Network-Based Flash Controllers. In Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), Orlando, FL, USA, 17–21 June 2023; Association for Computing Machinery: New York, NY, USA, 2023; Article 61, pp. 1–13.
15. Mielke, N.; Alam, S.; Goodman, A. Bit Error Rate in NAND Flash Memories. In Proceedings of the IEEE International Reliability Physics Symposium, Phoenix, AZ, USA, 27 April–1 May 2008; pp. 9–19.
16. Cai, Y.; Haratsch, E.F.; Mutlu, O.; Mai, K. Error Patterns in MLC NAND Flash Memory: Measurement, Characterization, and Analysis. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 12–16 March 2012; pp. 521–526.
17. Chang, W.; Lim, Y.; Cho, J. An Efficient Copy-Back Operation Scheme Using Dedicated Flash Memory Controller in Solid-State Disks. *Int. J. Electr. Energy* **2014**, *2*, 13–17. [CrossRef]
18. Pang, S.; Wang, X.; Zhang, Y.; Chen, L.; Liu, J.; Wu, D. PcGC: A Parity-Check Garbage Collection for Boosting 3-D NAND Flash Performance. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2023**, *42*, 4364–4377. [CrossRef]
19. Rostedt, S. Ftrace Linux Kernel Tracing. In Proceedings of the Linux Conference Japan, Tokyo, Japan, 5–6 October 2010.
20. Brunelle, A.D. *Block I/O Layer Tracing: Blktrace*; Hewlett Packard Company: Cupertino, CA, USA, 2006; p. 57.
21. Lee, J.; Kim, Y.; Shipman, G.M.; Oral, S.; Wang, F.; Kim, J. A Semi-Preemptive Garbage Collector for Solid State Drives. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, Austin, TX, USA, 4–6 April 2011; pp. 12–21.
22. Lee, J.; Kim, Y.; Shipman, G.M.; Oral, S.; Kim, J. Preemptible I/O Scheduling of Garbage Collection for Solid State Drives. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 247–260. [CrossRef]
23. Zhang, Q.; Li, X.; Wang, L.; Zhang, T.; Wang, Y.; Shao, Z. Optimizing Deterministic Garbage Collection in NAND Flash Storage Systems. In Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Seattle, WA, USA, 13–16 April 2015.
24. Kishani, M.; Ahmadian, S.; Asadi, H. A Modeling Framework for Reliability of Erasure Codes in SSD Arrays. *IEEE Trans. Comput.* **2020**, *69*, 649–665. [CrossRef]
25. Micron. NAND Flash Memory MT29F4G08ABADAH4 16Gb Asynchronous/Synchronous NAND Features Datasheet. Available online: https://www.micron.com/products/nand-flash/slc-nand/part-catalog/mt29f4g08abadah4-ait (accessed on 15 July 2020).
26. Sysbench Manual. Available online: http://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf (accessed on 22 December 2020).
27. MySQL 8.0 Reference Manual. Available online: https://dev.mysql.com/doc/refman/8.0/en (accessed on 22 December 2020).
28. Cooper, B.F.; Silberstein, A.; Tam, E.; Ramakrishnan, R.; Sears, R. Benchmarking Cloud Serving Systems with YCSB. In Proceedings of the 1st ACM Symposium on Cloud Computing (SOCC 10), Indianapolis, IN, USA, 10–11 June 2010.
29. Apache Cassandra Documentation v3.11.13. Available online: https://cassandra.apache.org/doc/3.11.13/ (accessed on 22 December 2020).
30. MongoDB Documentation v6.0. Available online: https://www.mongodb.com/ko-kr/docs/v6.0/ (accessed on 13 July 2021).
31. RocksDB Documentation. Available online: http://rocksdb.org/docs/getting-started.html (accessed on 13 July 2021).
32. Phoronix Test Suite v10.8.4 User Manual. Available online: https://github.com/phoronix-test-suite/phoronix-test-suite/blob/master/documentation/phoronix-test-suite.pdf (accessed on 21 June 2023).