electronics

MDPI

Article

On-Demand Garbage Collection Algorithm with Prioritized Victim Blocks for SSDs

Hyeyun Lee, Wooseok Choi and Youpyo Hong *

Division of Electronics and Electrical Engineering, Dongguk University-Seoul, Seoul 04620, Republic of Korea; hylee0142@dgu.ac.kr (H.L.); zz0123@dongguk.edu (W.C.)
* Correspondence: yhong@dgu.edu; Tel.: +82-2-2260-3818

Abstract: Because of their numerous benefits, solid-state drives (SSDs) are increasingly being used in a wide range of applications, including data centers, cloud computing, and high-performance computing. The growing demand for SSDs has led to a continuous improvement in their technology and a reduction in their cost, making them a more accessible storage solution for a wide range of users. Garbage collection (GC) is a process that reclaims wasted storage space in NAND flash memories, which are used as the memory devices for SSDs. However, the GC process can cause performance degradation and lifetime reduction. This paper proposes an efficient garbage collection (GC) scheme that minimizes overhead by invoking GC operations only when necessary. Each GC operation is executed in a specific order based on the expected storage gain and the execution cost, ensuring that the storage space requirement is met while minimizing the frequency of GC invocation. This approach not only reduces the overhead due to GC, but also improves the overall performance of SSDs, including the latency and write amplification factor (WAF) which is an important indicator of the longevity of SSDs.

Keywords: flash memory; solid-state drive; garbage collection

Citation: Lee, H.; Choi, W.; Hong, Y. On-Demand Garbage Collection Algorithm with Prioritized Victim Blocks for SSDs. *Electronics* **2023**, *12*, 2142. https://doi.org/10.3390/electronics12092142

Academic Editor: Manuel Mazzara

Received: 14 March 2023
Revised: 26 April 2023
Accepted: 5 May 2023
Published: 7 May 2023

1. Introduction

The fast access times of solid-state drives (SSDs) make them a more attractive option for tasks that require quick data retrieval and processing than conventional storage devices. In addition, they have a long lifespan and are less prone to physical damage, making them a more reliable storage option. Another advantage of SSDs is their lower power consumption compared to that of hard disk drives (HDDs), which make them a popular choice for use in mobile devices, laptops, and other battery-operated devices [1,2]. Because of their numerous advantages, SSDs are increasingly being used in a wide range of applications, including gaming, data centers, cloud computing, and high-performance computing [3].

The limitation of the infeasibility of in-place overwrite operations in NAND flash memory is due to the way the memory is constructed and how it operates. NAND flash memory is organized into blocks, and data can only be written to a block if it has been completely erased first. This restriction on the number of erase operations that can be performed and the size of the blocks creates challenges for SSDs in terms of managing storage space and ensuring efficient operation [4].

The obsolete data in NAND flash memory can cause fragmentation, which leads to the wastage of storage space. To address this issue, SSDs use a process called garbage collection (GC), which reclaims the wasted storage space by collecting and consolidating the obsolete data. This is necessary because over time the memory can become fragmented, with obsolete data scattered throughout the storage space. GC operations collect the obsolete data and consolidate them into a single area, allowing a SSD to reclaim the wasted storage space and operate more efficiently [5,6].

However, GC operations can also be a major source of performance degradation in SSDs. This is because the process of reclaiming wasted storage space requires a significant

amount of internal bandwidth, which can slow down the SSD and reduce its overall performance. This can lead to long wait times for data retrieval and processing, which can negatively impact user experience [7,8]. Therefore, reducing the frequency of GC operations while maintaining the necessary storage space requirements is a crucial aspect of optimizing SSD performance. This is important for ensuring that SSDs continue to be a fast, reliable, and efficient storage solution for a wide range of applications. Accordingly, minimizing the impact of GC operations on the performance of SSDs while ensuring that storage space requirements are met is important. This can be accomplished through the use of intelligent algorithms and techniques for managing storage space and prioritizing GC operations. The performance of SSDs can be optimized by reducing the frequency of GC operations, ensuring that they remain a fast, efficient, and reliable storage solution for various applications [9,10].

Plane blocking is more aggressive but has a lower performance penalty than channel blocking does [11]. This is because only a small portion of blocks are blocked for GC, so that the majority of blocks are accessible for regular read/write operations. In [12], a novel GC scheme that reduces the overhead of GC with low complexity was proposed. The scheme intelligently controls the priorities among various operations on SSDs, minimizing the overhead of GC while maintaining the storage space requirement. The scheme was evaluated in terms of latency, throughput, and storage utilization, and it was shown to significantly improve the performance and extend the lifetime of SSDs compared to the existing GC schemes.

GC is a critical factor for the performance and lifetime of SSDs. Although various GC schemes have been proposed, reducing performance degradation caused by GC remains a challenge. A novel GC scheme that balances the trade-off between the performance degradation and storage utilization by intelligently controlling the priorities among various operations on SSDs is an attractive solution to this problem.

The existing approaches for reducing GC overhead include fine-grained blocking of operations and interrupting GC if a new RW operation is required; however, these approaches have limitations in terms of data hazards and pre-emptive schedule management [13]. Another approach is to distribute data over multiple flash memory chips in a way that reduces the frequency of GC, but this requires careful management of the hotness of data and proper placement of data. Further research is needed to improve the performance of SSDs by effectively managing GC operations.

Another approach to reduce the GC overhead is to optimize the GC process itself. This can be carried out by reducing the amount of data that needs to be moved during the GC process or by improving the efficiency of the GC process. For example, some GC algorithms prioritize blocks that have the highest number of invalid pages and collect those blocks first, while others prioritize the blocks according to their age or the time since they were last collected [14]. Thus, the choice of GC algorithm can significantly affect the overall performance of an SSD.

A relatively new approach for minimizing the GC overhead is to optimize the underlying NAND flash memory technology. For example, some new NAND flash technologies, such as 3D NAND, have a higher page count per block than previous generations do, which reduces the number of blocks that need to be collected during the GC process [15,16].

In their study, Yu et al. [17] presented an algorithm for efficiently determining the destination block during a copy-back operation. The algorithm works by measuring the overwrite frequency of data and grouping similar data based on their frequency during garbage collection. This approach ensures that data are moved in a way that optimizes the use of available space and improves overall performance.

Cheng et al. [18] proposed a method for predicting data lifetimes and allocating resources accordingly. Their approach involves measuring the average lifetime of stored data and determining the characteristics of data lifetimes based on the type of application load. It is worth noting that these characteristics can change over time, requiring regular updates to ensure accurate predictions.

In their study, Lee et al. [19] presented a garbage collection (GC) scheme that organizes blocks based on the update counts of their pages. The selection of the GC target block is carried out through a formula that takes into account several factors, such as the cost of copy-back operations, program–erase cycle, and more. While the update frequency of data is a valuable metric, it is only one of the many factors associated with GC. Additionally, the percentage of pages that experience updates may not always be high. Therefore, relying solely on update frequency to organize blocks may not always result in the optimal arrangement. The proposed algorithm in this paper is aimed at addressing the problem of performance degradation due to GC in SSDs. In traditional GC schemes, GC operations are performed when the wasted storage space reaches a certain threshold. However, this approach often results in excessive GC invocations, leading to a performance degradation. The proposed GC scheme operates differently by considering the trade-off between the storage space requirement and the overhead of GC operations. GC is applied on the basis of the priority of GC target blocks according to their impact on the wasted storage space, allowing the SSD to operate more efficiently as it reduces GC call frequency.

2. Background

Modern SSDs typically comprise multiple NAND flash channels, which can be accessed in parallel as shown in Figure 1 [20]. The read and write operation unit of the NAND flash memory is a page, which is the smallest unit that can be read or written. Pages typically range in size from 4 to 16 Kbytes. An erase operation is performed in the unit of the block, which is a group of pages. By accessing multiple channels simultaneously, SSDs can achieve higher read and write speeds than those possible with a single-channel configuration [21,22].

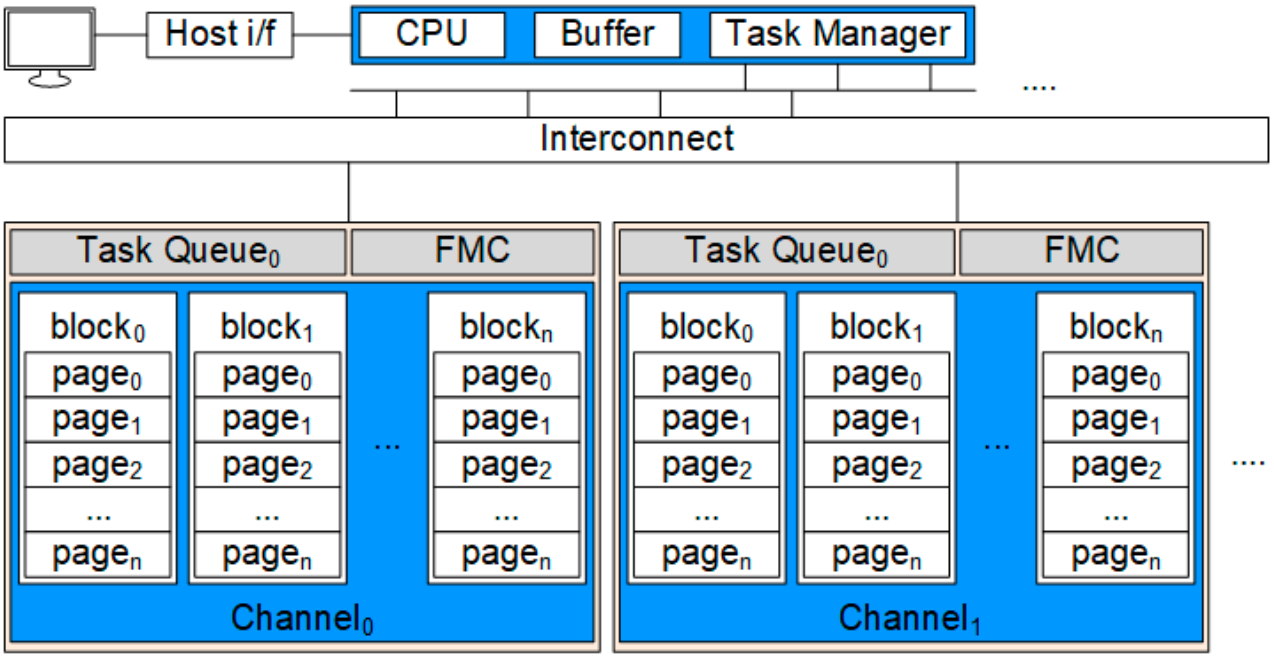

Figure 1. Configuration of multichannel solid-state drive (SSD).

The conventional procedure of a file write operation in SSDs is as follows. After a host generates a write request for a file, the SSD controller receives the file from the host and stores it in a temporary buffer. Then, the flash translation layer (FTL) functions as a bridge between the host interface and flash memory controller including the address-mapping task [4]. Intelligent algorithms are used to distribute the file to multiple channels to maximize the performance and lifetime of the SSD system. To reduce the processing

time, hardware logic may be used to perform read or write operations on the NAND flash memory in units of pages.

By distributing the file to multiple channels, the SSD system can achieve higher performance and a longer lifetime [21]. The FTL plays a key role in managing the mapping of logical addresses to physical locations in the flash memory, which helps to optimize the performance and reliability of the SSD [23]. Overall, a multichannel SSD system uses sophisticated algorithms and hardware to provide high-speed, reliable storage for a wide range of applications.

An erase operation is performed on a block unit owing to the physical structure of the flash memory. This means that if a page in a block is to be erased, the entire block must be erased. Furthermore, if a page in a block is marked as invalid, the entire block is considered invalid until it is erased. In SSDs, an overwrite operation in flash memory must be preceded by an erase operation because of the physical characteristics of the flash memory. If a page in a block is to be updated, the page with the existing data must be invalidated first and then the new data are written to another valid page. This means that the old data remain in the block until they are erased, which can affect the performance of the SSD over time.

In particular, GC occupies the internal bandwidth of the SOC inside SSDs and causes performance degradation. This is because the GC process requires copying valid data from multiple blocks into a new block, which takes time and resources [12]. Furthermore, GC operations are performed in the background, which reduces the available bandwidth for other operations, causing further performance degradation. Reducing the frequency of GC invocation while maintaining the storage space requirement is one way to remedy this problem. To address this issue, researchers have proposed various GC schemes with different trade-offs between performance, lifetime, and complexity.

The GC operation in SSDs is necessary for reclaiming wasted storage space; however, it can also negatively impact the performance and lifetime of SSDs. One solution to this problem is the use of a multistream drive, in which pages with similar data lifetimes are grouped together in the same block to minimize GC frequency. However, accurately predicting the lifetime of data is challenging and remains an active area of research [24,25].

While SSDs are generally faster than HDDs in terms of basic read and write operations, if a read or write operation is preceded by multiple GC operations, the access time of SSDs can be significantly longer than that of HDDs [26]. Therefore, intelligent algorithms that reduce or hide the GC overhead are essential for SSDs.

3. Related Work

Garbage Collection

GC is a task for SSDs to free up wasted storage space in flash memory. The basic unit of a write operation in an SSD is a page; however, the erase operation is allowed in one unit of a block, which is a group of pages. Therefore, there are three possible pages in a block: free, valid, and invalid pages. Initially, a block comprises free pages to which new data can be written. Once data are written to a free page, it becomes a valid page. In case a delete operation is applied to the valid page, the SSD controller marks the existing page as invalid. Because an in-place overwrite operation is not feasible in flash memories, the valid pages are marked as invalid if an overwrite operation is performed to a valid page. Therefore, valid and invalid pages are mixed inside a block as the SSD is in use. The storage occupied by invalid pages is waste, which needs to be minimized. GC is an operation that initializes such invalid pages to free pages. The problem is that the erase operation is allowed only in a block unit.

As shown in Figure 2, the SSD copy-back operation separates valid and invalid pages by moving valid pages to another block before performing GC. If the copy-back operation is performed before GC, only invalid pages remain in the block, and new empty pages can be created. However, frequent copy-back operations mean that many additional write

operations of flash memory occur. In other words, as the number of copy-backs increases, the lifetime of flash memory could be shortened.

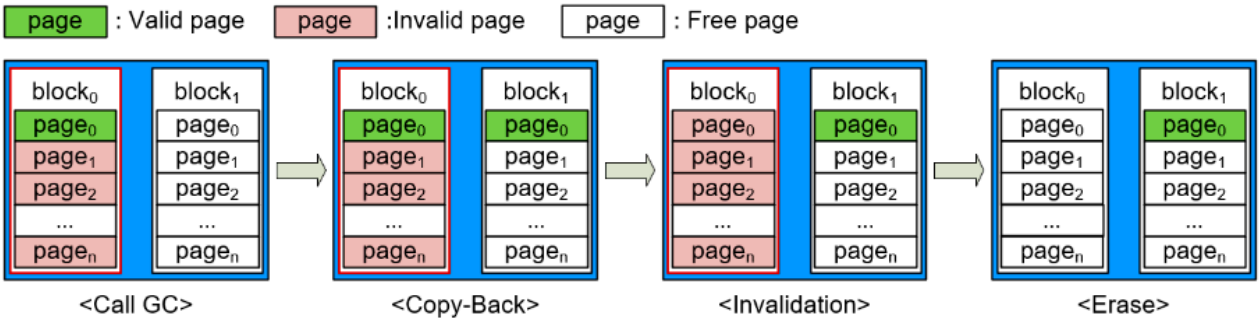

Figure 2. Garbage collection process.

4. Proposed Garbage Collection Algorithm

4.1. Motivation

Although GC is an essential operation for making SSDs practical by reducing storage waste, extensive research on the decision criterion for GC invocation timing is lacking. A simple yet widely accepted approach is to apply GC to a block if the percentage of invalid pages in the block reaches a predefined threshold. Although this approach fulfills the basic requirements for reclaiming wasted storage, it has fundamental limitations.

The valid pages copied back to another block may soon be invalidated in the new block. If it is known that such pages would be invalidated soon, GC would be delayed until such valid pages become invalidated to avoid unnecessary copy-back. By contrast, if valid pages in a block are destined to stay valid for a very long period of time, it may be desirable to apply GC to the block even if the predetermined threshold has not been not reached thus far.

To verify the feasibility of our claim, we measured the number of blocks with 70% invalid pages that reached the state with more than 90% invalid pages eventually; the corresponding results are presented in Table 1. In some cases such as RocksDB, most blocks with 70% invalid pages eventually become full of invalid pages. In this case, it would be reasonable to set the GC invocation criteria tight so that GC is executed after the target block is full of invalid pages to minimize copy-back overhead.

Table 1. Statistics of invalid page increase.

Ratio of Victim Blocks (%)	Cassandra	MongoDB	MySQL	RocksDB	Dbench	SQLite
70% invalid blocks	59.827	80.908	92.859	68.494	88.577	92.712
90% invalid blocks	51.855	74.036	84.326	68.396	76.184	42.322
90%/70%	86.676	91.506	90.811	99.857	86.012	45.648

Notably, most blocks with 70% invalid pages eventually reach more than 90% invalid pages. SQLite workload is an exception where only half of the blocks reach 90% invalid pages. This means that such blocks are not expected to have more invalid pages even after a long duration. In this case, delaying the application of GC in such blocks contributes to significant storage waste.

Therefore, we decided to measure the expected gain for an individual GC in each target block and execute GCs in the expected gain.

4.2. Invalidation Rate-Based GC-Triggering Algorithm

Our goal is to control the GC invokation schedule depending on how fast the percentage of invalid pages increases for each block. If a block is being invalidated relatively quickly, it is desirable to delay the GC invocation time because the entire block is likely to

be invalidated in the near future. However, if a block is being invalidated at a slow rate, GC should ideally be applied promptly once the percentage of invalid pages reaches the predetermined threshold to reclaim wasted storage without delay.

The primary task for accomplishing such a goal is to measure the rate at which the number of invalid pages increases for each block. The basic idea is to calculate the ratio of invalid pages to the total number of valid pages and divide it by the lifetime of the block:

$$invalidation\ rate = \frac{(N_{invld} - 1)/N_{total}}{T_c - T_f}, \quad (1)$$

where N_{invld} is the number of invalid pages in a block, N_{total} is the total number of pages in a block, T_c is the current time, and T_f is the time when an invalid page is first created in that block. We propose to measure and update such parameters, including invalidation rate values, whenever an invalid page occurs, using the invalidation rate to determine the block order in which GC should be applied.

To support the proposed GC scheme, the block status table (BST) associated with each block, as shown in Figure 3, stores the information to calculate the invalidation rate. In our implementation, the blocks with invalid page ratios of 70% or higher are stored and managed in the victim block list.

Block No.	First invalid time	Last invalid time	Invalid page ratio	Invalidation rate
0000	0.050	0.786	78.1%	1.05
0001	0.333	0.655	39.1%	1.18
0010	0.600	2.090	53.1%	0.35
⋮	⋮	⋮	⋮	⋮

Figure 3. A sample block status table (BST) configuration.

If a block becomes full of invalid pages, GC is applied to the block to reclaim the storage. If the amount of space reclaimed by this process is insufficient, GC must be applied to the block with a high number of invalid pages. In the conventional scheme, GC is applied to all blocks with a higher number of invalid pages than a predetermined threshold has. This is a straightforward way to reclaim storage; however, it may invoke more GC than required, which leads to very high wear of the device and long latency in assessing the SSD due to GC overhead.

In the proposed GC scheme, blocks with a high number of invalid pages are ranked according to their invalidation rate rather than being treated equally as in the conventional scheme. This ranking takes into account the expected time until a block becomes completely full of invalid pages, which allows the algorithm to prioritize GC for blocks that are less likely to become completely full in the near future.

In Figure 4, two blocks, A and B, which contain more than a predetermined number of invalid pages, are illustrated. Suppose block A is expected to become completely full of invalid pages in the near future, while block B is expected to remain mostly unused for a long time. In this case, it would be more efficient to apply GC in block B first, even though it has a lower percentage of invalid pages compared to block A. This is because delaying GC for block B would not provide any benefit in terms of reducing unnecessary copy-back operations, whereas delaying GC for block A would minimize the number of copy-back operations needed.

However, the absence of a remaining valid page means that there is no need for GC delay anymore. In this case, we give GC top priority. This is because there is no valid page; thus, latency and additional writes due to copy-back do not occur, and it is the most efficient way to secure SSD storage space.

Figure 4. Illustration of two blocks that contain more than a predetermined number of invalid pages.

5. Evaluation

5.1. Experimental Setup

SSDs were modeled using C code and tested with trace files created from various workloads by extracting data using ftrace [27], blktrace, and blkparse [28]. The SSD used in the experiment was 1-TB Samsung T5 Portable SSD. The Linux kernel version was 3.10.0-1160.59.1.el7.x86_64, and the OS version was Red Hat Enterprise Linux Workstation release 7.9. In addition, the CPU corresponding to the host used Intel(R) Core™ i7-9700K with 32 GB of RAM.

Table 2 summarizes the key parameters of the flash memory used for the SSD model with 4 channels, 4 planes per channel, 512 blocks per plane, and 128 pages per block. The read, write, and erase operation times were 25 µs, 230 µs, and 0.7 ms, respectively [29].

Table 2. Parameters of SSD configuration.

Parameter	Value
Number of channels	4
Planes per channel	4
Blocks per plane	512
Pages per block	128
Page size (KB)	4
Page read latency (µs)	25
Page write latency (µs)	230
Erase latency (ms)	0.7

When calculating the time required to perform GC for one block in a channel, we can assume that GC is applied when the number of invalid pages in the block exceeds 70% [30]. The copy-back operations for the 30% pages of the block take 9.8 ms, and erasing one block takes 0.7 ms, resulting in a total GC time of 10.5 ms for one block [31].

The overall flow of the proposed GC scheme is outlined in the following pseudocode. The Algorithm 1 below shows the process of giving priority to victim blocks, i.e., blocks with an invalid page ratio of 70% or higher. Several global variables are used to prioritize the GC of each block in the order of the lowest invalidation rate.

5.2. Trace File Analysis

Table 3 is an analysis table that provides information on the trace files extracted from six different databases used in an experiment. These trace files were extracted by installing a database and a manual of each benchmark program, such as MySQL based on sysbench [32,33], Yahoo Cloud Serving Benchmark (YCSB) family [34], Cassandra [35], MongoDB [36], RocksDB [37], Phoronix Test Suite [38] family SQLite and Dbench.

Algorithm 1. Prioritize victim blocks.

Global Variables

1. S_{used}: percentage of used storage on SSD;
2. T_c: current time when an invalid page is created in the block;
3. T_f: time when an invalid page is first created in the block;
4. *Block_status_table*: table that stores information for calculating the *invalidation_rate*;
5. *Victim_block_list*: list of blocks with invalid page ratios of 70% or higher.

Main Procedure

1. **while (1) perform**
2. **if** an invalid page is created inside the block^i **then**
   a. Update block^i's T_c, T_f and an invalid_page_ratio to Block_status_table;
   b. Update S_{used};
3. **if** *invalid_page_ratio* of block^i $\geq$ 70% **then**
   a. Copy information of block^i stored in *Block_status_table* to *Victim_block_list*;
4. **end if**
5. **end if**
6. **if** $S_{used} \geq$ 70% **then**
   a. PRIORITIZE_VICTIM_BLOCK( );
   b. Perform garbage collection on the block at the top of the *Victim_block_list*;
7. **end if**
8. **end while**

end procedure
Procedure PRIORITIZE_VICTIM_BLOCK ( );

9. **for** (i = 0**; i <** Num_of_victim_block; i++) **begin**
10. Calculate the *invalidation_rate* of block^i using the information in *Victim_block_list*;
11. **end for**
12. Sort *Victim_block_list* in ascending order based on block's *invalidation_rate*;
13. **for** (i = 0**;** i **<** Num_of_victim_block; i++) **begin**
14. **if** *invalidate_ratio* of block^i == 100% **then**
   a. Reorder block^i to the top of the *Victim_block_list*;
15. **end if**
16. **end for**

end procedure

Table 3. Characteristics of trace files.

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

5.3. Experimental Results

We implemented the conventional GC and proposed GC scheme and compared their performances. As a framework to test GC performance, we selected the plane-blocking GC (PBGC) scheme presented in [11] and block-blocking GC (BBGC) scheme from [13]. GC was enabled if 70% of the entire SSD storage was full for all schemes. The blocks with more than 70% invalid pages were chosen as target blocks. If the conventional GC scheme

was applied to PBGC and BBGC, all the target blocks were reclaimed by GC. As shown in Figure 5, the portion of used disk space significantly dropped once GC was enabled.

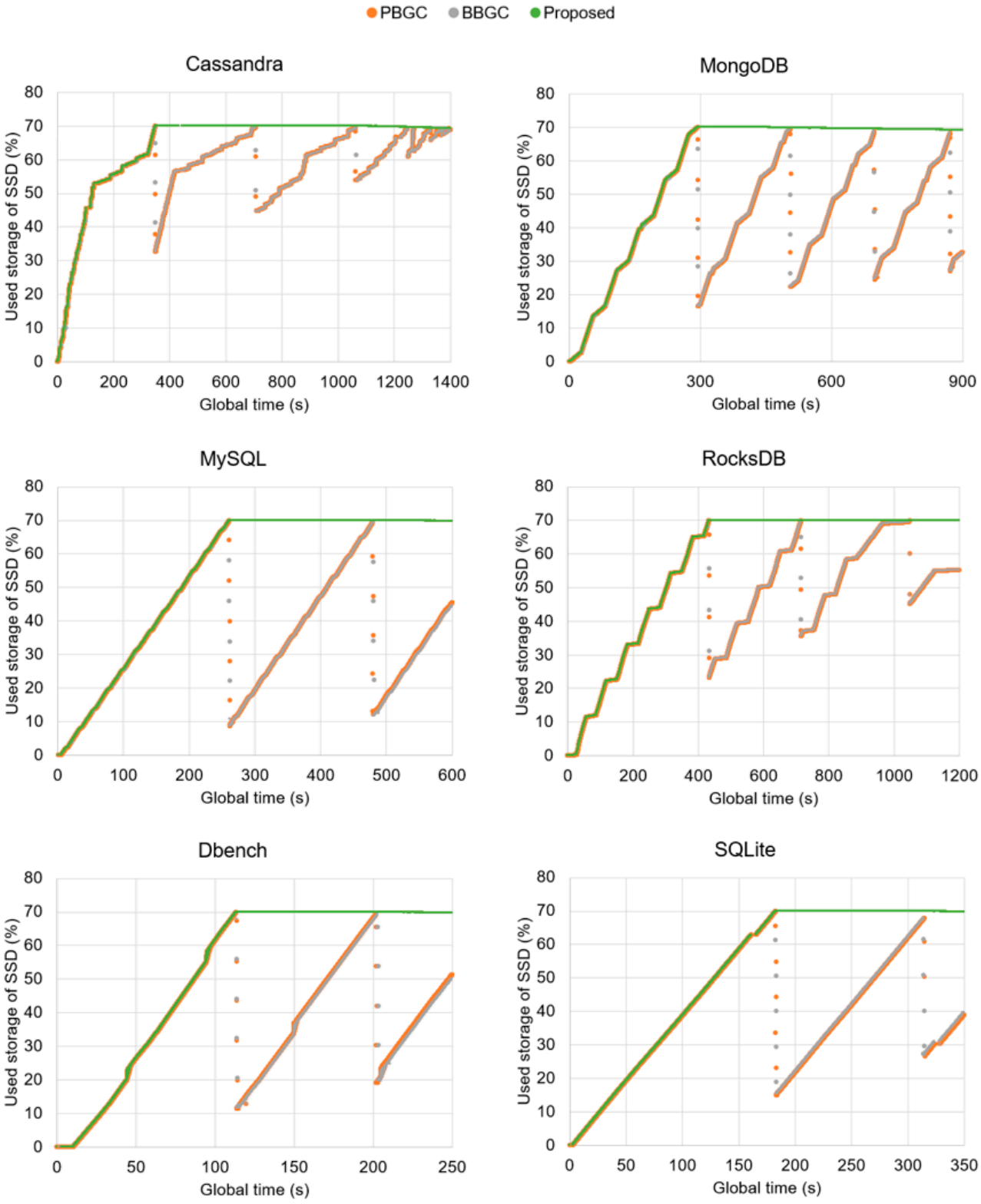

Figure 5. Used storage of SSD.

The GC was enabled under the same condition of a 70% full disk for conventional GC; the proposed GC scheme was executed minimally to maintain a disk occupancy rate of below 70%, as shown in Figure 5.

We measured the average latency for read and write operations to the SSD. Table 4 reveals that the proposed GC algorithm improved the latency up to 99.4% in the case of a read operation and 99.6% in the case of a write operation. Note that the MySQL benchmark does not include any read operations.

Table 4. Average latency of read/write request.

Avg. read latency (ms)	Cassandra	MongoDB	MySQL	RocksDB	Dbench	SQLite
PBGC	531.064	28.149	-	0.122	-	-
BBGC	1.382	20.364	-	0.122	-	-
Proposed	0.615	0.119	-	0.123	-	-
Avg. write latency (ms)	**Cassandra**	**MongoDB**	**MySQL**	**RocksDB**	**Dbench**	**SQLite**
PBGC	65.509	13.033	19.699	2.922	58.035	142.699
BBGC	51.303	13.614	17.435	2.913	57.111	140.553
Proposed	56.050	0.531	1.195	1.683	37.143	0.614
Proposed/BBGC Improvement ratio (%)	**Cassandra**	**MongoDB**	**MySQL**	**RocksDB**	**Dbench**	**SQLite**
Avg. read latency	55.499	99.416	-	−0.820	-	-
Avg. write latency	−9.253	96.100	93.146	42.225	34.963	99.563

It should be noted that in the case of MySQL and Dbench, read access constitutes only a negligible portion of the benchmark results, and as such, these results were omitted from the table.

Regarding the RocksDB benchmark, our proposed scheme exhibits a slightly longer delay than that of BBGC. Figure 6 illustrates that data are accessed regularly, with idle periods occurring between consecutive accesses. As long as the execution time of garbage collection (GC) is not longer than that of the regular read or write operations, which is not the case, all three GC scheme choices are not expected to have a significant impact on the overall latency.

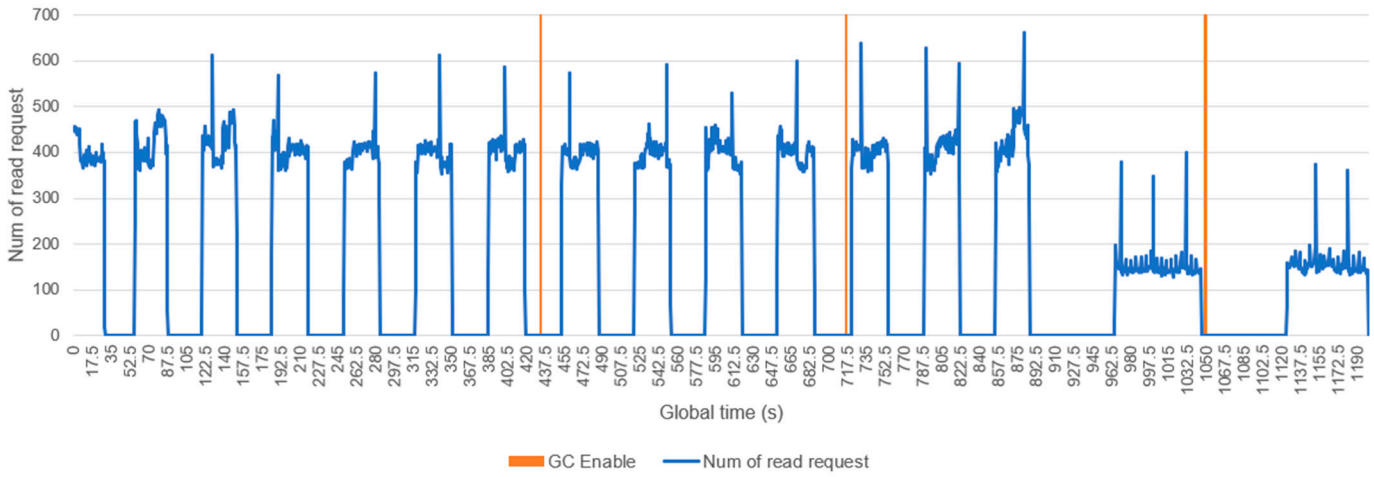

Figure 6. RocksDB data access pattern and GC-enabled timing.

According to Table 3, the proposed scheme exhibits a longer average write latency than BBGC does for the Cassandra benchmark, which consists of 56.8% sequential access. This is likely due to the proposed GC scheme frequently intervening during sequential access, as space usage is controlled at a fine grain level. As a result, the proposed scheme has an adverse effect on latency. Additionally, the average data size for write access are much larger than those for read access, contributing to overall loss. It would be valuable to develop an intelligent scheme that avoids penalties on sequential access.

In addition, we measured the maximum latency for read and write operations to the SSD. Table 5 reveals that the proposed GC algorithm improved the latency by a minimum of 99.3% in the case of a read operation and 99.8% in the case of a write operation.

Table 5. Maximum latency of read/write request.

Max. read latency (ms)	Cassandra	MongoDB	MySQL	RocksDB	Dbench	SQLite
PBGC	5123.887	5726.358	-	38.044	-	-
BBGC	486.446	5350.930	-	38.155	-	-
Proposed	4.192	36.029	-	39.340	-	-
Max. write latency (ms)	Cassandra	MongoDB	MySQL	RocksDB	Dbench	SQLite
PBGC	2317.631	5933.481	8972.502	689.303	7054.371	21,329.661
BBGC	2317.248	6215.630	7881.995	697.057	7001.079	21,109.462
Proposed	2757.480	36.519	50.490	99.357	1694.329	34.678
Proposed/BBGC Improvement ratio (%)	Cassandra	MongoDB	MySQL	RocksDB	Dbench	SQLite
Max. read latency	99.138	99.327	-	−3.106	-	-
Max. write latency	−18.998	99.412	99.359	85.746	75.799	99.836

The total numbers of GC operations are summarized in Figure 7. The GC operation call was minimized in the proposed scheme, and the results confirmed that the number of GC calls significantly decreased.

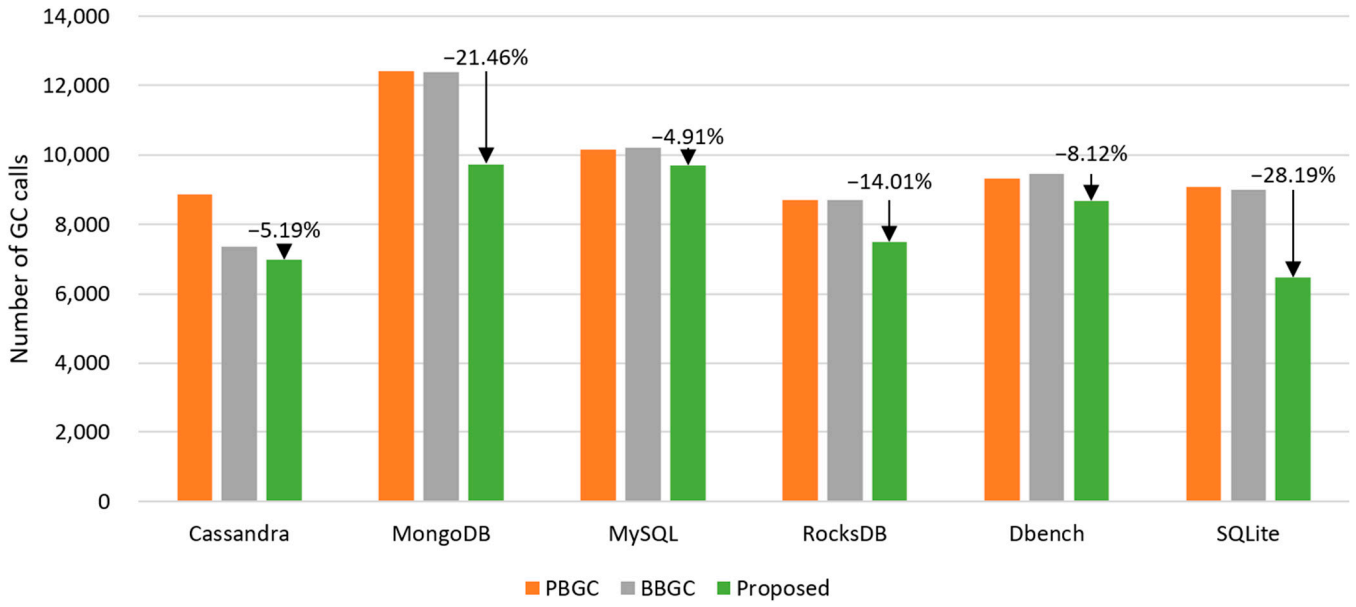

Figure 7. Number of GC calls.

The write amplification factor (WAF) indicates the ratio of total write operations to the original write operation requested by the host. In particular, a high WAF indicates that several copy-back operations have been performed. A high WAF not only degrades SSD performance in terms of latency but also reduces SSD lifetime because a flash memory cell has a limitation in the number of write operations.

WAF is calculated using the following formula [39]:

$$\mathrm{WAF} = \frac{\text{number of arrived pages} + \text{number of copied pages}}{\text{number of arrived pages}}. \quad (2)$$

Figure 8 shows that the proposed GC algorithm significantly reduced the WAF, which means that it is expected to improve both the performance and lifetime of SSDs.

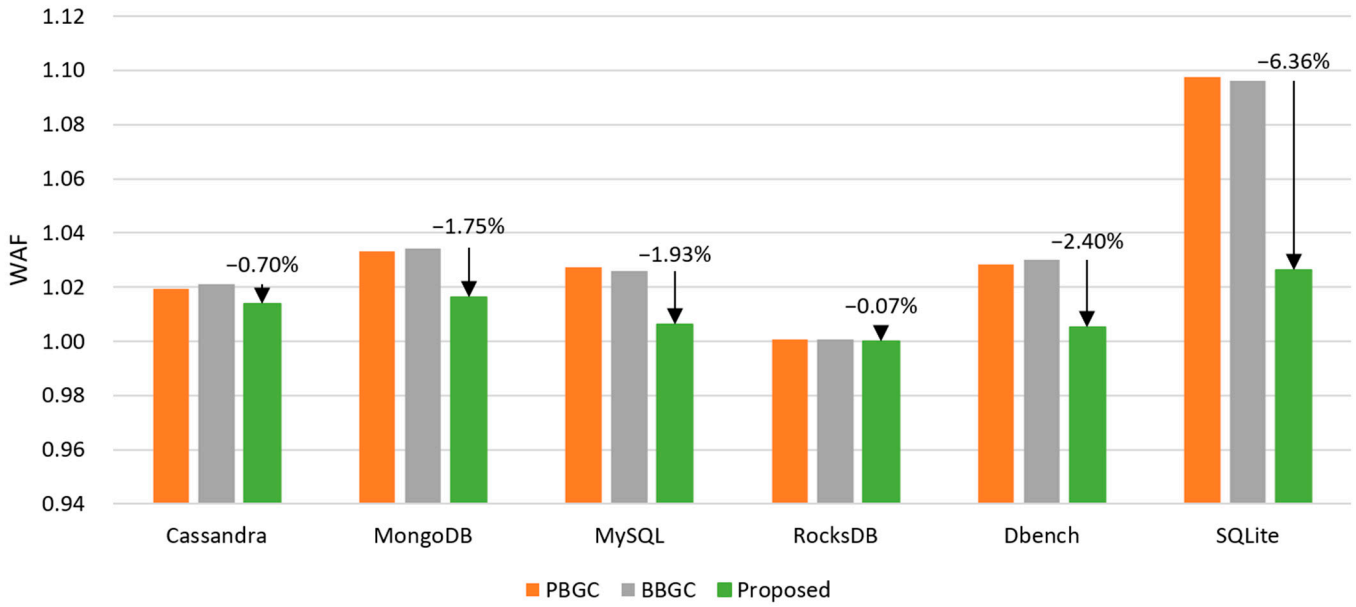

Figure 8. Write amplification factor (WAF) comparison.

5.4. Comparison with Relevant Work

In the study by Cheng et al. [18], the allocation mechanism proposed aims to group data with similar lifetimes together in the same flash blocks. To achieve this, the mechanism takes into account various factors such as data arrival time, device number, logical page address, I/O size, and I/O flag. They use these factors to compute the "hotness" of the data and assign it to a block associated with the same hotness. However, the paper does not provide specific details on how the hotness of data is computed or how the destination block is chosen when multiple candidate blocks meet the condition to store the data. While Cheng et al.'s work offers a unique approach that could be compared to our own, their study lacks specificity on how they collected and incorporated data to arrive at their lifetime predictions. Although they mention using empirical values, they do not provide further details on how these values were obtained or applied, so qualitative performance evaluation of their mechanism compared to our algorithm was not feasible.

In the study by Lee et al. [19], the GC target block is determined using a formula that takes into account multiple factors, including the cost of copy-back operation and the program–erase cycle. Qualitatively comparing their work with ours could be interesting, but the detailed descriptions of the parameters used in their cost formula are not presented in their study. As a result, a direct comparison between the two approaches is not feasible.

6. Conclusions

In this paper, we propose a new GC scheme that minimizes the invocation of GC operations to reduce overhead. Additionally, the scheme orders GC target blocks according to the expected gain and penalty, with the goal of improving read/write latency and WAF while reducing the number of GC operations.

The experimental results presented in this paper indicate that the proposed GC scheme significantly improves read/write latency and WAF while also reducing the number of GC operations. This suggests that the proposed GC algorithm has the potential to improve not only SSD performance but also the lifetime of SSDs.

This new GC scheme has the potential to considerably improve the performance and lifetime of SSDs, making them a more attractive choice for a variety of applications. With its low complexity, it is also an accessible solution for both manufacturers and users.

Author Contributions: Conceptualization, H.L., W.C. and Y.H.; methodology, H.L., W.C. and Y.H.; software, H.L. and W.C.; validation, H.L., W.C. and Y.H.; formal analysis, H.L., W.C. and Y.H.; investigation, H.L. and W.C.; resources, H.L. and W.C.; writing—original draft preparation, H.L., W.C. and Y.H.; writing—review and editing, H.L., W.C. and Y.H.; visualization H.L., W.C. 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 work was supported by the Dongguk University Research Fund of 2023.

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

Conflicts of Interest: The authors declare no conflict 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. Chung, T.S.; Park, D.J.; Park, S.; Lee, D.H.; Lee, S.W.; Song, H.J. A survey of flash translation layer. *J. Syst. Archit.* **2009**, *55*, 332–343. [CrossRef]
5. Guo, J.; Hu, Y.; Mao, B.; Wu, S. Parallelism and garbage collection aware I/O scheduler with improved SSD performance. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, FL, USA, 29 May –2 June 2017; pp. 1184–1193.
6. Bux, W.; Iliadis, I. Performance of greedy garbage collection in flash-based solid-state drives. *Perform. Eval.* **2010**, *67*, 1172–1186. [CrossRef]
7. Google. Google: Taming The Long Latency Tail—When More Machines Equals Worse Results. 2012. Available online: http://highscalability.com/blog/2012/3/12/google-taming-the-long-latency-tail-when-more-machines-equal.html (accessed on 3 March 2023).
8. Dean, J.; Barroso, L.A. The tail at scale. *Commun. ACM* **2013**, *56*, 74–80. [CrossRef]
9. Gupta, A.; Pisolkar, R.; Urgaonkar, B.; Sivasubramaniam, A. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In *FAST*; USENIX Association: Washington, DC, USA, 2011; pp. 91–103.
10. 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, 16–19 February 2015; pp. 257–271.
11. Yan, S.; Li, H.; Hao, M.; Tong, M.H.; Sundararaman, S.; Chien, A.A.; Gunawi, H.S. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. *ACM Trans. Storage (TOS)* **2017**, *13*, 1–26. [CrossRef]
12. Lee, J.; Kim, Y.; Shipman, G.M.; Oral, S.; Kim, J. Preemptible I/O scheduling of garbage collection for solid state drives. *IEEE Trans. Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 247–260. [CrossRef]
13. Ae, J.; Hong, Y. Efficient Garbage Collection Algorithm for Low Latency SSD. *Electronics* **2022**, *11*, 1084. [CrossRef]
14. 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]
15. Chen, T.Y.; Chang, Y.H.; Ho, C.C.; Chen, S.H. Enabling sub-blocks erase management to boost the performance of 3D NAND flash memory. In Proceedings of the 53rd Annual Design Automation Conference, Austin, TX, USA, 5–9 June 2016; pp. 1–6.
16. Choi, E.S.; Park, S.K. Device considerations for high density and highly reliable 3D NAND flash cell in near future. In Proceedings of the 2012 International Electron Devices Meeting, San Francisco, CA, USA, 10–13 October 2012; pp. 9.4.1–9.4.4.
17. Yu, F.; Yan, H. An Efficient Hot-Cold Data Separation Garbage Collection Algorithm Based on Logical Interval in NAND Flash-based Consumer Electronics. In *IEEE Transactions on Consumer Electronics*; IEEE: New York, NY, USA, 2023.
18. Cheng, W.; Luo, M.; Zeng, L.; Wang, Y.; Brinkmann, A. Lifespan-based garbage collection to improve SSD's reliability and performance. *J. Parallel Distrib. Comput.* **2022**, *164*, 28–39. [CrossRef]
19. Lee, D.; Jin, Y.; Jang, H.; Lee, B. CATDOG: Cost-Age-Time Data Organized Garbage Collection. In Proceedings of the 2022 IEEE International Conference on Networking, Architecture and Storage (NAS), Philadelphia, PA, USA, 3–4 October 2022; IEEE: New York, NY, USA, 2022; pp. 1–6.
20. Jung, M.; Choi, W.; Srikantaiah, S.; Yoo, J.; Kandemir, M.T. HIOS: A host interface I/O scheduler for solid state disks. *ACM SIGARCH Comput. Archit. News* **2014**, *42*, 289–300. [CrossRef]
21. Kang, J.U.; Kim, J.S.; Park, C.; Park, H.; Lee, J. A multi-channel architecture for high-performance NAND flash-based storage system. *J. Syst. Archit.* **2007**, *53*, 644–658. [CrossRef]
22. Yu, S.; Chen, P.Y. Emerging memory technologies: Recent trends and prospects. *IEEE Solid-State Circuits Mag.* **2016**, *8*, 43–56. [CrossRef]

23. Shin, J.Y.; Xia, Z.L.; Xu, N.Y.; Gao, R.; Cai, X.F.; Maeng, S.; Hsu, F.H. FTL design exploration in reconfigurable high-performance SSD for server applications. In Proceedings of the 23rd International Conference on Supercomputing, Yorktown Heights, NY, USA, 8–12 June 2009; pp. 338–349.
24. Kang, J.U.; Hyun, J.; Maeng, H.; Cho, S. The multi-streamed solid-state drive. In Proceedings of the 6th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 14), Philadelphia, PA, USA, 17–18 June 2014.
25. Yang, J.; Pandurangan, R.; Choi, C.; Balakrishnan, V. AutoStream: Automatic stream management for multi-streamed SSDs. In Proceedings of the 10th ACM International Systems and Storage Conference, Haifa, Israel, 22–24 May 2017; pp. 1–11.
26. Rizvi, S.S.; Chung, T.S. Flash SSD vs. HDD: High performance oriented modern embedded and multimedia storage systems. In Proceedings of the 2010 2nd International Conference on Computer Engineering and Technology, Chengdu, China, 16–18 April 2010; Volume 7, pp. V7-297–V7-299.
27. Rostedt, S. Ftrace Linux kernel tracing. In Proceedings of the Linux Conference Japan, Tokyo, Japan, 31 August–1 September 2010.
28. Brunelle, A.D. *Block I/O Layer Tracing: Blktrace*; HP: Gelato-Cupertino, CA, USA, 2006; p. 57.
29. Li, J.; Xu, X.; Peng, X.; Liao, J. Pattern-based write scheduling and read balance-oriented wear-leveling for solid state drivers. In Proceedings of the 2019 35th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 20–24 May 2019; pp. 126–133.
30. Agrawal, N.; Prabhakaran, V.; Wobber, T.; Davis, J.D.; Manasse, M.S.; Panigrahy, R. Design tradeoffs for SSD performance. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, 22–27 June 2008; Volume 57.
31. Micron NAND Flash Memory MT29F16G08ABABA 16Gb Asynchronous/Synchronous NAND Features Datasheet. Available online: https://www.micron.com/products/nand-flash/slc-nand/part-catalog/mt29f16g08ababawp-ait (accessed on 15 July 2020).
32. Sysbench Manual. Available online: http://imysql.com/wpcontent/uploads/2014/10/sysbench-manual.pdf (accessed on 22 December 2020).
33. MySQL 8.0 Reference Manual. Available online: https://dev.mysql.com/doc/refman/8.0/en/ (accessed on 22 December 2020).
34. 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.
35. Apache Cassandra Documentation v4.0-beta4. Available online: https://cassandra.apache.org/doc/latest/ (accessed on 22 December 2020).
36. MongoDB Documentation v5.0. Available online: https://docs.mongodb.com/manual/ (accessed on 13 July 2021).
37. RocksDB Documentation. Available online: http://rocksdb.org/docs/getting-started.html (accessed on 13 July 2021).
38. Phoronix Test Suite v10.0.0 User Manual. Available online: https://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf (accessed on 22 December 2020).
39. Chun, Y.; Han, K.; Hong, Y. High-performance multi-stream management for SSDs. *Electronics* **2021**, *10*, 486. [CrossRef]