electronics

MDPI

Article

Parallelism-Aware Channel Partition for Read/Write Interference Mitigation in Solid-State Drives

Hyun Jo Lim [1], Dongkun Shin [2] and Tae Hee Han [3,*]

1 Department of Artificial Intelligence, Sungkyunkwan University, Suwon 16419, Republic of Korea
2 Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea
3 Department of Semiconductor Systems Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea
* Correspondence: than@skku.edu

Abstract: The advancement of multi-level cell technology that enables storing multiple bits in a single NAND flash memory cell has increased the density and affordability of solid-state drives (SSDs). However, increased latency asymmetry between read and write (R/W) intensifies the severity of R/W interference, so reads cannot be processed for a long time owing to the extended flash memory resource occupancy of writing. Existing flash translation layer (FTL)-level mitigation techniques can allocate flash memory resources in a balanced manner taking R/W interference into account; however, due to the inefficient utilization of parallel flash memory resources, the effect on performance enhancement is restrictive. From the perspectives of the predicted access pattern and available concurrency of flash memory resources, we propose a parallelism-aware channel partition (*PACP*) scheme that prevents SSD performance degradation caused by R/W interference. Moreover, an additional performance improvement is achieved by reallocating interference-vulnerable page using leveraged garbage collection (GC) migration. The evaluation results showed that compared with the existing solution, *PACP* reduced the average read latency by 11.6% and average write latency by 6.0%, with a negligible storage overhead.

Keywords: data migration; flash memory; garbage collection; page allocation; R/W interference

check for updates

Citation: Lim, H.J.; Shin, D.; Han, T.H. Parallelism-Aware Channel Partition for Read/Write Interference Mitigation in Solid-State Drives. *Electronics* **2022**, *11*, 4048. https://doi.org/10.3390/electronics11234048

Academic Editor: Marco Vacca

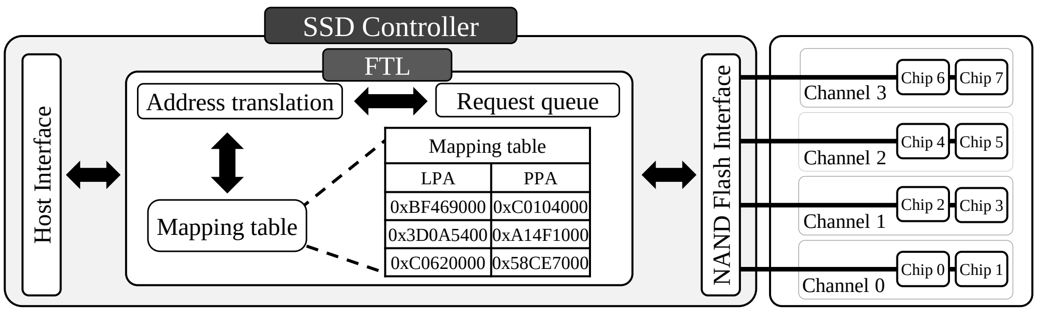
Received: 10 November 2022
Accepted: 2 December 2022
Published: 6 December 2022

1. Introduction

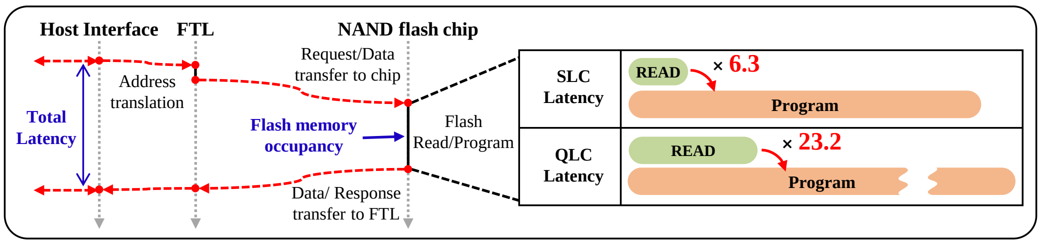
In NAND flash memory, the number of bits per cell is increased to expand the capacity and reduce the bit cost. However, as the number of bits per cell increases, the NAND flash becomes more difficult to store and read correctly considering that the threshold voltage margins become smaller [1]. For example, compared to a single-level cell (SLC) that stores one bit in one cell, the read time of a quad-level cell (QLC) that can store four bits in one cell is approximately five times longer, the write time is approximately twenty times longer, and the latency asymmetry is over four times longer [2]. A solid-state drive's (SSD) overall performance suffers as the read/write (R/W) interference problem becomes more severe owing to the increasing latency asymmetry.

The simplified data flow across the SSD's components for processing requests received from the host is illustrated in Figure 1a. First, the flash translation layer (FTL) converts the logical address to the physical address based on a given data allocation policy. Next, the FTL forwards the corresponding data request to NAND flash chip arrays and logs the translated address information in the mapping table in parallel to expedite the subsequent data accesses.

Figure 1b depicts the timing diagram for this process to highlight the problem to be addressed in this study as a result of the technology trend of NAND flash evolving into a multi-level cell (including MLC, TLC, QLC, and so forth) structure for higher integration. Consequently, data processing latency increases rapidly, and in particular, the imbalance between the read and write is exacerbated significantly. For instance, the write latency for SLC is approximately 6.3 times that of the read latency, whereas the ratio for QLC increased dramatically to 23.2 times [2]. Therefore, ineffective data allocation can frequently cause read processing delays owing to write operations.



(**a**)



(**b**)

Figure 1. (**a**) Data flow across the components in the simplified SSD structure; (**b**) timing diagram for a request flow from host to chip.

Previous studies have focused on allocation strategies that use simple mathematical modeling to determine the target of a flash transaction. For instance, the location of data storage is determined by the quotient and remainder of successive divisions of logical addresses in a predefined order of channel, chip, die, and plane [3–6]. However, this approach makes the latency of the request sensitive to the address pattern of the I/O request, and its static nature can cause conflicts on shared resources. A promising alternative to cope with this challenge is to adopt a dynamic approach when allocating resources depending on the characteristics of the transaction workloads [7]. However, the existing dynamic allocation strategy focuses on resource conflict; thus, it is insufficient to resolve R/W interference through effective request distribution considering the characteristics of the workload.

By addressing the problems of imbalanced flash memory resource utilization and R/W interference in the existing dynamic data allocation methods, we propose a parallelism-aware channel partition (*PACP*) that boosts SSD performance through efficient data allocation. *PACP* allocates SSD resources dynamically based on request types in order to maximize utilization and reduce read processing latency by focusing primarily on dominant request patterns. Furthermore, reallocating interference-vulnerable pages that may occur when changing the allocated area also contributes to an increase in performance.

The contributions of this study are summarized as follows:

1. SSD is partitioned at the channel level using the parallelism-aware channel partition technique to mitigate R/W interference, and latency is reduced by allocating pages leveraging transaction histories.
2. The allocated channel is dynamically updated to maximally utilize the flash memory, even when the workload has highly imbalanced access characteristics.

3. Reallocating some pages vulnerable to R/W interference through a migration scheme is performed to address worst-case scenarios where latency can increase and leveraged garbage collection (GC) to minimize SSD performance overhead.

The remainder of this paper is organized as follows. Section 2 presents related work on R/W interference. Section 3 describes the proposed idea, *PACP*. The simulation results and analysis under various conditions are provided in Section 4. Finally, Section 5 provides the conclusion and future work.

2. Related Work

Various efforts have been conducted to alleviate R/W interference, which can be categorized into program/erase (P/E) suspension, an I/O scheduler at the host level, and performance isolation. Table 1 summarizes the existing R/W interference mitigation schemes for NAND flash memory. Wu et al. proposed a P/E suspension method, which allows for the interruption of long program or erase transactions to process immediate read transactions [8]. However, along with a complicated scheduling technique for proper program and erase interruption, this method requires a page buffer to temporarily store the data of the hindered operation.

Table 1. Summary of R/W interference mitigation schemes.

Category	Scheme	Main Idea	Advantages	Limitations
P/E suspension	Wu et al. [8]	Suspend on-going P/E operations	Significantly decrease the read latency	System & hardware overhead Negative impact on the endurance
I/O scheduling	FIOS [9]	Separates reads and writes in batches	No hardware modification	Read and write requests only separated in batches
	BCQ [10]			
	PIQ [11]			
Performance isolation	Kim et al. [12]	Eliminating interference from different tenants that share the SSD	Significantly decrease R/W interference between tenants	Interference within the workload has not been addressed
	Huang et al. [13]			
	Lv et al. [14]	Allocate different types of requests into different partitions	Significantly decreases R/W interference	Channel utilization problem has not been given sufficient consideration

Typical I/O schedulers are primarily designed to expedite I/O in multi-tasking environments that share a storage device. Although the FIOS [9] describes how to alleviate R/W interference at the host level, it takes a simple approach by sending the read request out first and waiting to send all the writes until the dispatched reading is complete [15]. BCQ [10] presents a similar approach with a higher accuracy for determining R/W costs. Both FIOS and BCQ can cause write delays owing to frequent read access, which can degrade the overall performance of the storage device. In contrast, PIQ [11] minimizes the access interference among I/O requests in a single batch by exploiting the rich parallelism of the SSD. However, all these I/O schedulers isolate read and write requests within a single batch, and the R/W interference problem still exists between batches.

Performance isolation schemes exploit the fact that the main cause of R/W interference is the sharing of flash memory resources among the different types of requests. The objective of performance isolation techniques [12,13,16,17] is to provide predictable latency by preventing interference between tenants sharing an SSD. For example, Huang et al. and Kim et al. deployed a similar approach to data allocation to avoid performance degradation caused by the garbage collection (GC) of other tenants [12,13]. Although performance isolation schemes reduce inter-tenant interference, they overlook R/W interference within a single workload.

Regarding the R/W interference problem within a single workload, Lv et al. [14] proposed a static channel partition for different requests to be handled independently. Compared to previous isolation methods, R/W interference can be addressed to a higher level by statically partitioning channels based on workload access patterns. However, when read and write requests are highly imbalanced, this approach results in low flash memory usage.

3. Parallelism-Aware-Channel-Partition (PACP)

The primary goal of *PACP* is to increase the flash memory resource utilization and mitigate R/W interference through workload-aware page allocation. Figure 2 shows an overview of *PACP* implemented in the SSD controller. First, an access pattern is predicted based on the transaction histories in the modified mapping table to select a flash memory resource where the data are processed depending on the request types. Then, *PACP* handles the SSD resource underutilization by budgeting the area allocated for each request type according to the intensity of the workload and performs R/W-interference-aware page allocation. However, if the request imbalance between reading and writing is severe, partitioning the allocated area can result in a problem where request processing is concentrated on specific flash memory resources. The declining channel utilization is boosted by allowing the use of an alternative channel type in accordance with the access characteristics of the workload. R/W-interference-aware page allocation enables the partial assignment of pages to the other type channels for flash memory resource utilization improvement. Furthermore, *PACP* pursues additional performance improvements with effective reallocation via leveraged GC migration for pages that may become vulnerable to interference.

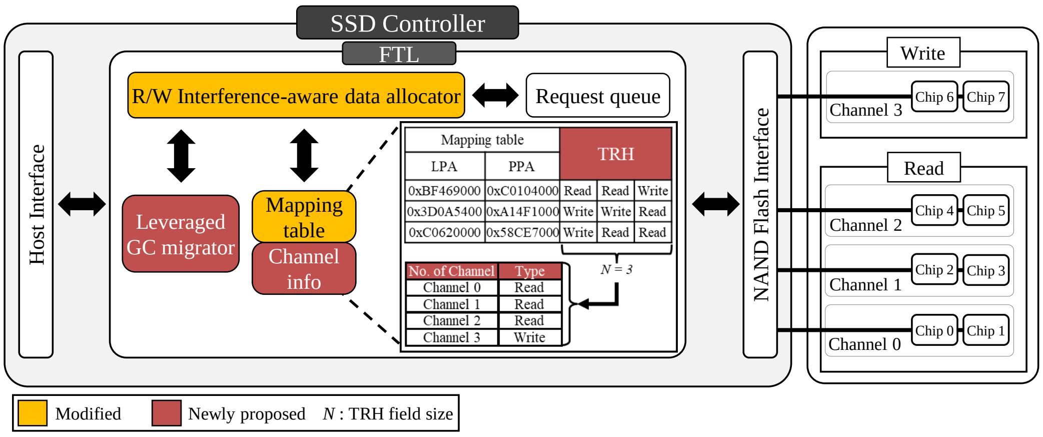


Figure 2. Overview of the parallelism-aware channel partition (PACP).

3.1. Request Prediction on Page

It is vital to characterize the workload being processed on the SSD in order to expose the maximum parallelism of flash memory resources based on the types of requests. To reflect the nature of real-world workloads, we analyzed the Microsoft Research (MSR) trace [18] that profiled workloads running on the Microsoft Research Enterprise Server. Figure 3 shows the read and write percentages processed by pages classified into three categories for nine workloads randomly selected from the MSR traces. A page is classified as read-dominant (write-dominant) if more than 90% of the requests processed are read (write). Otherwise, it is categorized as a mixed page. Although the ratio of reads to writes varies depending on the workload being processed, approximately 91.6% of requests are processed on the dominant pages, as shown in Figure 3. These observations are consistent with previous studies [19,20] and provide the guideline for flash resource partitioning for performance optimization considering the behavior of real-world applications.

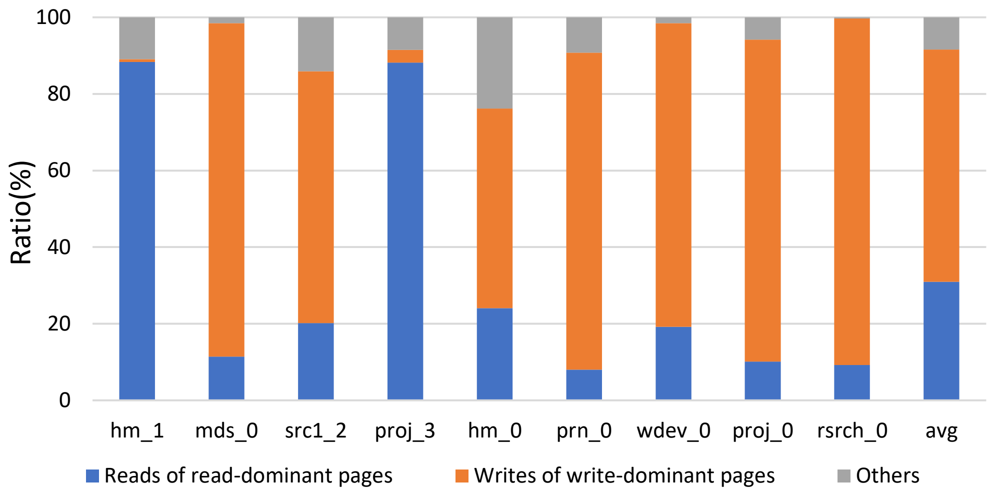


Figure 3. Distribution of reads and writes ratio processed by pages classified into three categories.

To appropriately handle the biased workloads of read or write depending on the addresses of the MSR traces, the associated request type must be logged at run time. As shown in Figure 2, *PACP* requires a transaction history (*TRH*) field in line with the corresponding mapping table entry containing the logical page address (LPA) and physical page address (PPA). When the FTL refers to the mapping table during address translation, *TRH* records the request type and can hold up to N recent requests.

The predicted type is determined by comparing the number of reads and writes in the *TRH*; the greater one is chosen. For example, if *TRH* field tracks three transaction histories (as in Figure 2) and two or more of those are read, the type of request for that page is predicted to be read. However, if an even number of transaction histories are held, the number of reads and writes could be the same. To deal with this case, the request is predicted based on the stored channel information to utilize the evicted past access information. This information is derived from the recorded transaction requests, which will be discussed in Section 3.2.

3.2. *R/W-Interference-Aware Page Allocation*

PACP aims to minimize R/W interference by isolating the flash memory resource for read and write while taking parallelism into account. Channel-wise partitioning is performed to eliminate any conflict caused by physical resource occupancy between read and write requests. The channel partition is determined by the ratio of page types identified by *TRH*. During the workload processing, the channel partition ratio is dynamically adjusted by reflecting the updated read- and write-dominant page ratios, effectively responding to request intensity and maximizing resource utilization. When switching channel types, the channel with the fewest pages of the previous type is chosen to avoid R/W interference caused by different types of pages within the channel.

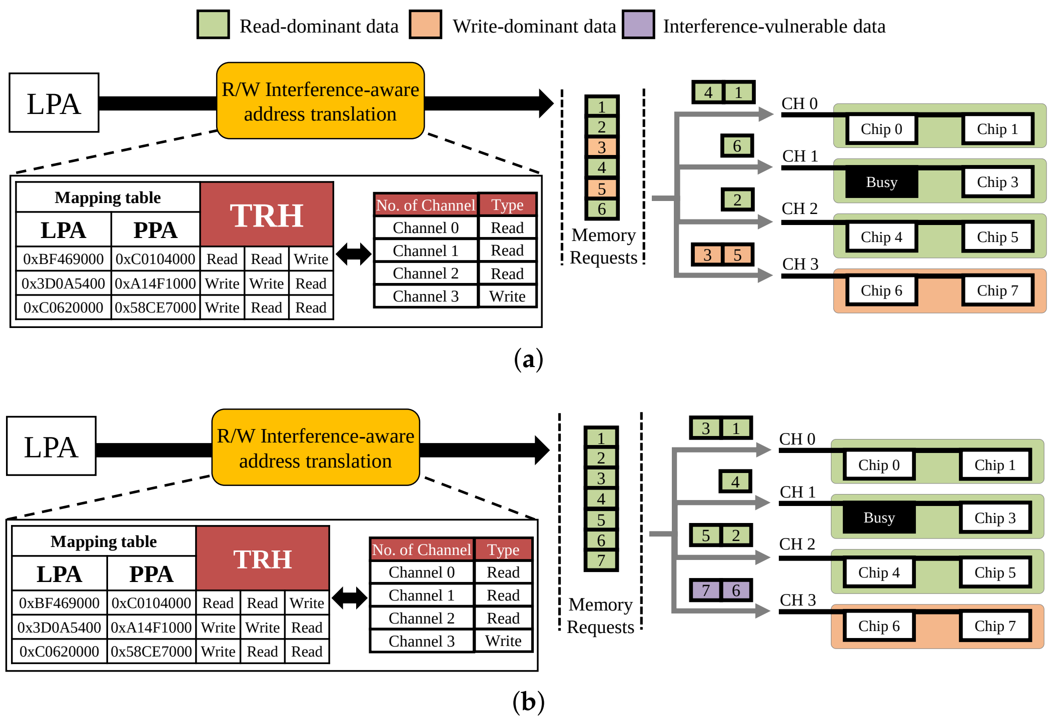
Algorithm 1 presents R/W-interference-aware page allocation to fully utilize the partitioned read and write channels. By default, if there is no transaction history, dynamic page allocation is performed at the channel, chip, die, and plane level order in a round-robin manner. If a resource is busy, it is assigned to the next resource at the same level to utilize flash memory resources efficiently [7]. For requests with transaction history, R/W interference is mitigated through workload-aware page allocation. Considering that the type of request depends on LPA, dynamic allocation is performed in the area partitioned into read and write channels, as shown in Figure 4a.

Algorithm 1 R/W-interference-aware data allocation.

```
Input:
    Each channel type info : $C_{status}$ = {$C_{Read}$, $C_{Write}$}
    Logical page address : $LPA$
    $N$ transaction histories of the LPA : $TRH$
Output:
    Physical page address : $PPA$
1:  $TRH_{Read}$ ← The number of read transaction histories of the $TRH$
2:  $TRH_{Write}$ ← The number of write transaction histories of the $TRH$
3:  if  No $TRH$ then
4:      $PPA$ ← Dynamic_page_allocation($LPA$)
5:  end if
6:  if $TRH_{Read} > TRH_{Write}$ then
7:      if all $Channels$($C_{Read}$) is busy then
8:          $PPA$ ← Dynamic_page_allocation (LPA, $C_{Write}$)
9:      else
10:         $PPA$ ← Dynamic_page_allocation (LPA, $C_{Read}$)
11:     end if
12: else if $TRH_{Read} < TRH_{Write}$ then
13:     if all $Channels$($C_{Write}$) is busy then
14:         $PPA$ ← Dynamic_page_allocation (LPA, $C_{Read}$)
15:     else
16:         $PPA$ ← Dynamic_page_allocation (LPA, $C_{Write}$)
17:     end if
18: else if  $TRH_{Read} == TRH_{Write}$ then
19:     if prvious $PPA$ on $Channels$($C_{Write}$) then
20:         $PPA$ ← Dynamic_page_allocation (LPA, $C_{Write}$)
21:     else
22:         $PPA$ ← Dynamic_page_allocation (LPA, $C_{Read}$)
23:     end if
24: end if
```

The method in Figure 4a may cause processing delays due to insufficient channels being allocated for each type according to the imbalanced requests. For instance, when requests are processed repeatedly only for read-dominant pages, write channel resources may not process any requests. As requests are concentrated only on a specific channel, read requests are buffered for a long time, which has the same effect as reducing the number of channels in the SSD. This is fatal for an SSD with a small number of channels, and the performance degradation from the channel partitioning can cause a bigger problem than the response time improvement that can be obtained through R/W interference mitigation.

When imbalanced requests are processed intensively, as shown in Figure 4b, some pages are partially stored in the other type of flash memory resources. This has the advantage of allowing request processing without waiting for other requests to complete. However, partially allocating data to the other type of area conflicts with design goals of *PACP* by generating data vulnerable to R/W interference. Although it is a problem in the context of considering the worst case in the application of *PACP*, the possibility of effectively solving this problem was explored by leveraging GC, which is an intrinsic operation of SSD.



Mapping table		TRH		
LPA	PPA			
0xBF469000	0xC0104000	Read	Read	Write
0x3D0A5400	0xA14F1000	Write	Write	Read
0xC0620000	0x58CE7000	Write	Read	Read

No. of Channel	Type
Channel 0	Read
Channel 1	Read
Channel 2	Read
Channel 3	Write

Figure 4. R/W-interference-aware page allocation: (**a**) basic allocation flow; (**b**) allocation flow with strongly imbalanced requests.

3.3. Leveraged GC Migration

R/W-interference-aware page allocation partially allocates pages to channels of the other type when excessively imbalanced requests occur to address latency degradation caused by reduced flash-memory resource usage. The page that can intensify interference by using the same hardware resources for read and write operations is reallocated through migration.

The migration processes of read- and write-dominant pages are different owing to the difference between the read and write operations of the SSD. For write operations, data are stored in the physical location, which is determined based on the predefined allocation policy. If data already exist, the page containing that old information is marked invalid, and new data are allocated to a free page and changes its state to valid. When an additional write request occurs for a write-dominant page residing in the read channel, migration is performed to the write channel through R/W-interference-aware page allocation.

However, read-dominant pages in the write channel cannot be reallocated if additional write requests are not made, so the reallocation is performed through migration. Page migration introduces additional writes and occupancy SSD resources, which can degrade the performance of foreground jobs. In order to avoid performance degradation, background migrations should only be performed when the storage system is not in use.

To improve the system performance in the background migration process, page allocation is performed by leveraging GC, which frequently occurs during the SSD lifetime and significantly influences the SSD latency. The GC procedure reclaims invalidated pages by selecting a candidate victim block with many invalid pages, moving any valid pages into a free block, and then erasing the block [21]. As the proportion of invalid pages to the total pages within a block rises, GC efficiency increases [22]. Through GC-aware migration, the block with the largest number of invalid pages is chosen to increase the efficiency of GC that will occur and reallocate page that is vulnerable to interference.

Depending on workload intensity, it may not be possible to sufficiently migrate a read-dominant page stored in a write channel through GC-aware migration. The migration overhead is hidden by transferring the page using an approach similar to the GC-piggybacked migration proposed by Han et al. [23], in which the read-dominant page in the victim block is migrated to a block in a read channel and GC is being performed in the write channel. Consequently, the page migration overhead is concealed by piggybacking the migration on the GC and eliminating unnecessary program operations.

As shown in Figure 5a, the block with the highest number of invalid pages is chosen, and a read-dominant page in the block is migrated to a read channel at idle time. Figure 5b shows the process of migrating the read-dominant page when GC is performed, hiding latency without unnecessary program operations caused by migration.

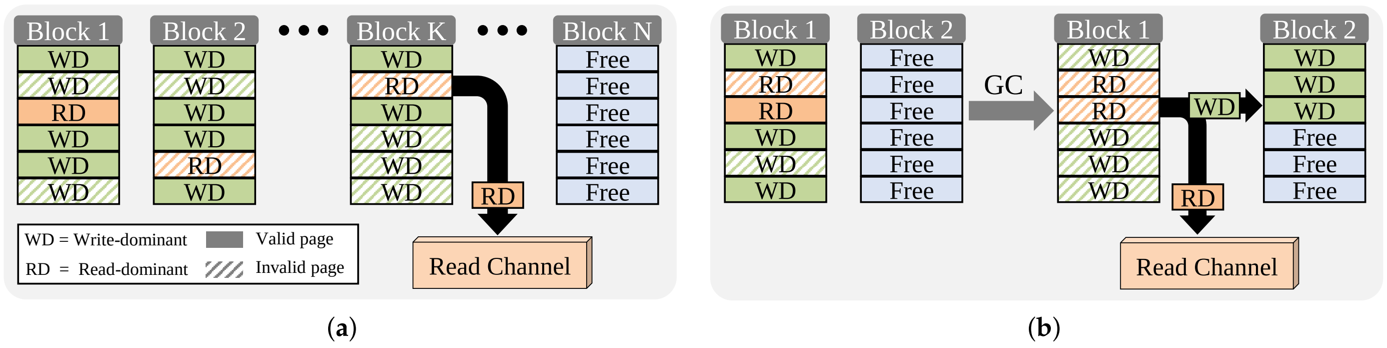


Figure 5. Procedure for read-dominant pages migration vulnerable to R/W interference: (**a**) GC-aware migration at idle time; (**b**) GC-piggybacked migration during GC process.

3.4. Lifespan Discussion

PACP performs channel-wise partitioning and allocates pages referring to transaction histories stored in the mapping table to mitigate R/W interference in SSD. The limited performance improvement stemming from the parallelism decrease is solved by dynamically altering the channel type and reallocating interference-vulnerable pages. However, *PACP* can affect the lifespan by causing uneven utilization of flash memory resources in write-dominant applications due to the physical properties of NAND flash memory.

Existing wear-leveling solutions proposed for partitioned SSDs can address the problem of *PACP* effectively. For example, Huang et al. [12] proposed a method of exchanging partitions when the device is idle, and Kim et al. [24] presented a method of using the write area in round-robin order. The lifetime degradation problem of *PACP* can be alleviated.

4. Evaluation

SSDsim [4], a well-known trace-driven simulator, was utilized to model *PACP*. In the evaluation, flash microarchitecture comprised 8–64 channels, and read latency according to the number of channels was compared with Lv et al.'s scheme [14], which is performance isolation in a workload. The configuration of flash memory resources per channel and chip operation latency is shown in Table 2. The implemented page-mapping-based FTL, GC, and wear leveling of the SSDsim were utilized [4]. Various workloads were used to validate the effectiveness of *PACP* according to the read and write ratios, as shown in Table 3.

Table 2. Configuration of the simulated SSD.

NAND Flash Parameters	Value
Number of Channels	8–64 (8, 16, 32, 48, 64)
Number of Chips per Channel	2
Number of Dies per Flash Chip	2
Number of Planes per Die	2
Number of Blocks per Plane	2048
Number of Pages per Block	256
Page Size (KB)	4
Page Read Latency (µs)	140
Page Program Latency (µs)	3102
Block Erase Latency (ms)	3.5

Table 3. Request ratio of seven workloads selected from MSR.

Workload	Read Ratio (%)	Write Ratio (%)
hm_1	95.3	4.7
src1_2	25.3	74.7
hm_0	35.5	64.5
proj_3	93.5	6.5
prn_0	10.8	89.2
wdev_0	20.1	79.9
web_0	29.9	70.1
average	44.3	65.7

4.1. Prediction Accuracy Based on TRH

A trade-off between prediction accuracy and field size was analyzed for the TRH-based request prediction technique used by the *PACP*. Figure 6 depicts the prediction accuracy according to the field size of 1 to 3 bits. The average accuracy of request type prediction based on the history of a single request was 96.4%. Even though there is only information about the most recent request type, it is still possible to make an accurate prediction about the page type. This is because there is a strong possibility that requests of the same type will be processed on the identical page, as was demonstrated by the results of earlier workload analysis in Section 3. However, the processing of mixed read and write requests appears to be the cause of the incorrect distinction of 3.6%.

When using more than one bit of *TRH*, channel information with a page is used to resolve ambiguity due to the same number of transaction histories. However, the prediction accuracy of the two-bit *TRH* was 94.6%, which was less than that of the one-bit transaction history, owing to the problem of erroneously predicting a page misallocated to channels of the other type. This means that some pages require migration.

In the case of using three transaction histories, a 0.5% increase in accuracy was presented because more information was utilized compared to one bit. Despite an increase of two bits for each mapping entry resulting in an additional 64 MB of overhead based on 512 GB flash with 4 KB pages, there was only a 0.5% improvement in accuracy compared to one bit. TRH field to each mapping entry requires 2 MB of capacity overhead per channel count for including an additional bit. This is not an effective solution considering that SSD's storage capacity increased by approximately 0.1% for every one bit of TRH field size increase. Therefore, *PACP* adopts a one-bit TRH to predict the type of request to be processed on the page.

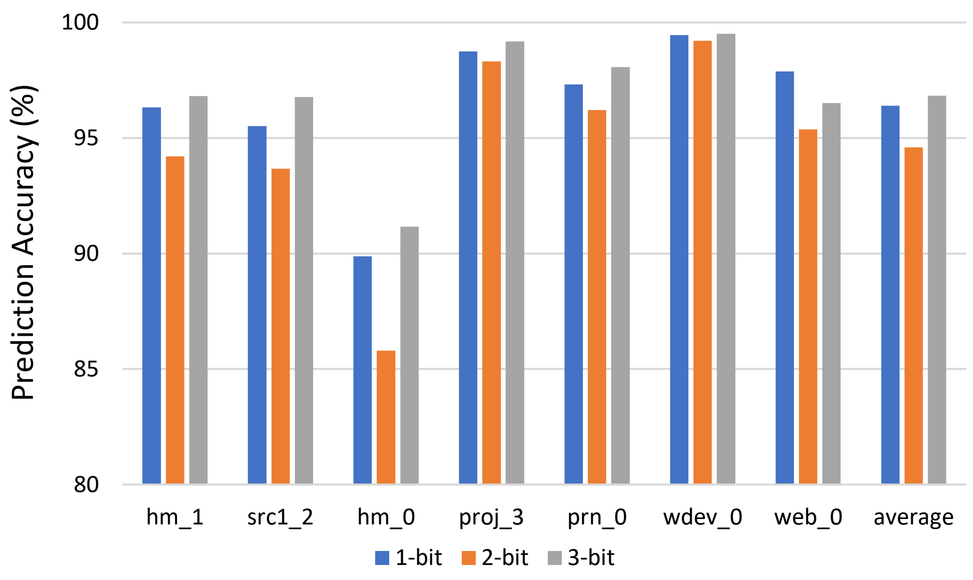


Figure 6. TRH bits count impact on request-type-prediction accuracy.

4.2. PACP Latency Compared to Ideal

A preliminary experiment was conducted in which *PACP* was compared to the ideal case without R/W interference. The root cause of R/W interference is read processing delay due to writes monopolizing flash memory resources. 32-channel SSD was configured on the simulator to make the write-induced resources preemption close to zero to eliminate the read delay caused by the write, showing the ideal case of read delay.

In Figure 7, 'Ideal' represents the read latency when there is no interference in comparison to the baseline dynamic allocation [7] for the seven previously selected workloads in Table 3. Hm_1, proj_3, and src1_2 are workloads with relatively low R/W interference frequencies, making it challenging to anticipate considerable *PACP* effectiveness. In contrast, read processing for hm_0, prn_0, wdev_0, and web_0 is severely delayed due to the write operation. Thus, when *PACP* is used, a significant performance enhancement can be expected.

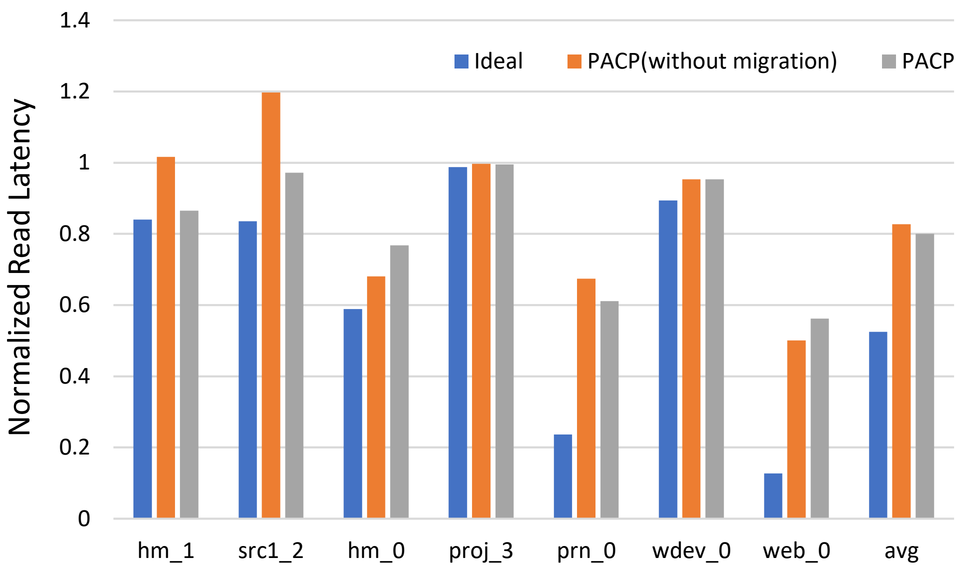


Figure 7. Normalized read latency comparison with the ideal.

Hm_1 and proj_3 are workloads that exhibit insignificant performance degradation owing to R/W interference, as verified in 'Ideal'. Even if *PACP* (without migration) is applied to resolve interference, it shows a similar read latency as the baseline. For hm_0, prn_0, and web_0, the application of *PACP* (without migration) showed a tendency comparable to the 'Ideal', and the latency reduced significantly to 31.9%, 32.6%, and 49.3%, respectively. However, although src1_2 and wdev_0 are workloads vulnerable to R/W interference, the latency reduction is insufficient, and src1_2 increases latency by 19.0%. This problem is predictable because interference-vulnerable pages are inherently handled poorly by R/W-interference-aware page allocation technique.

PACP reallocates the read-dominant pages in the write channel resources to the appropriate location with low overhead through GC leveraged migration. The latency of src1_2 was 21.9% lower than *PACP* (without migration), showing an average read latency comparable to the baseline. Hm_1 showed decreases in latency of approximately 12.9% through *PACP*, although the performance degradation owing to R/W interference was relatively low at 16.0%. By leveraging GC migration, an effective response to the worst case was achieved, as well as additional performance improvement for workloads that are not sensitive to R/W interference. In addition, the average read latency was further reduced for hm_1, proj_3, and prn_0. However, owing to the overhead of the migration process, the latency increased by 8.7% and 6.1% for hm_0 and wdev_0, respectively. Although the migration technique reduced the overall latency by 7.8%, the handling of some workloads suffered.

Overheads caused during the reallocation procedure were analyzed to understand GC-leveraged migration comprehensively. Figure 8 shows the ratio of program counts to total during the migration process. In hm_1 and proj_3, where the read ratios accounted for 93.5% and 95.3%, respectively, migration occurred infrequently. As a result, only 0.2% of the additional programs were processed. For other workloads, the migration ratio varied depending on the intensity of the request. Nevertheless, the migration ratio for all workloads was within 2.7%, and the average migration ratio was around 1.0%. These are negligible program counts considering the average read latency decreased owing to migration.

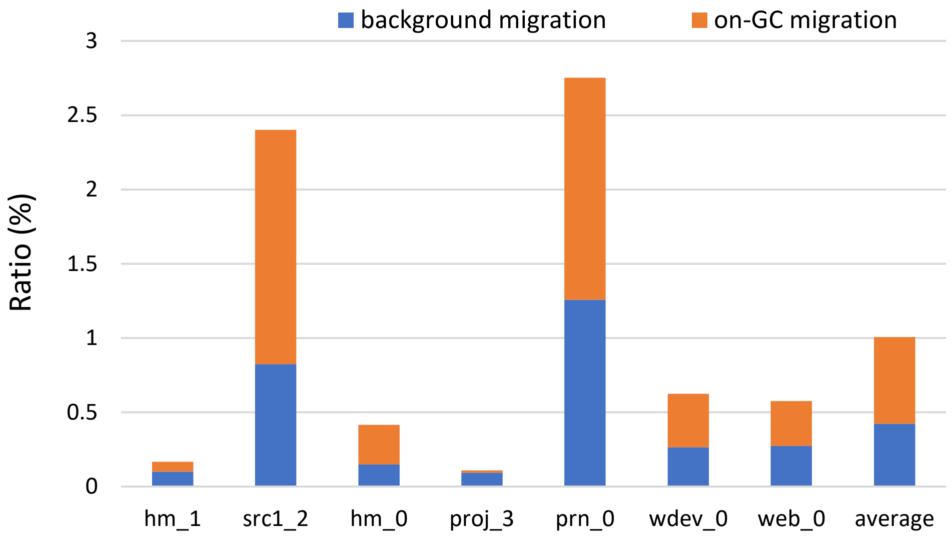


Figure 8. GC-aware migration and GC-piggybacked migration's cost to the total number of program operations.

4.3. Overall Comparison with Existing Solution

The method of Lv et al. [14] and PACP, commonly categorized into R/W interference mitigation by performance isolation (Table 1), were compared from various perspectives.

For performance comparisons based on SSD configuration and workload interference intensity, various workloads were applied under 8 to 64 channel configurations.

Figure 9a shows the latency according to the number of channels for web_0, a workload with high interference. Despite the severe interference, latency increases when there are 16 or fewer channels because Lv et al.'s method decreases channel utilization. However, in environments with a large number of channels, page allocation using statically partitioned channels has proven to be effective because it drastically reduces interference. When the number of channels was 48 or more, the decrease in read latency was bigger than that of *PACP*. Lv et al.'s method reduced latency by 6.0% on average for a workload with high R/W interference.

A comparison of the read latency for the low interference workload hm_1 is shown in Figure 9b. The performance of *PACP*, which uses flash memory resources effectively, was more effective in all cases because performance improvements due to R/W interference mitigation were not significant.

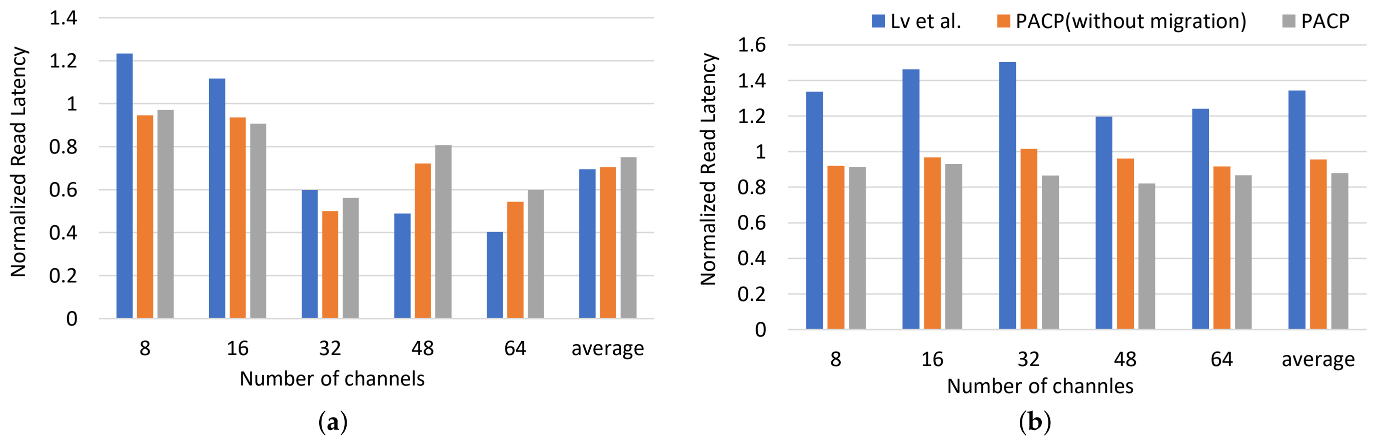


Figure 9. Read latency comparison under different interference conditions: (**a**) high-interference workload (hm_1); (**b**) low-interference workload (web_0).

Figure 10a depicts the average read latency for the seven workloads depending on the number of channels. For cases where the number of channels is 32 or less, the latency of the existing scheme increased, whereas the latency of *PACP* decreased. In addition, when the number of channels exceeded 48, both existing methods and *PACP* decreased latency. However, the *PACP* latency reduction effect was approximately 11.6% for all channels, which showed outstanding performance compared to the existing technique.

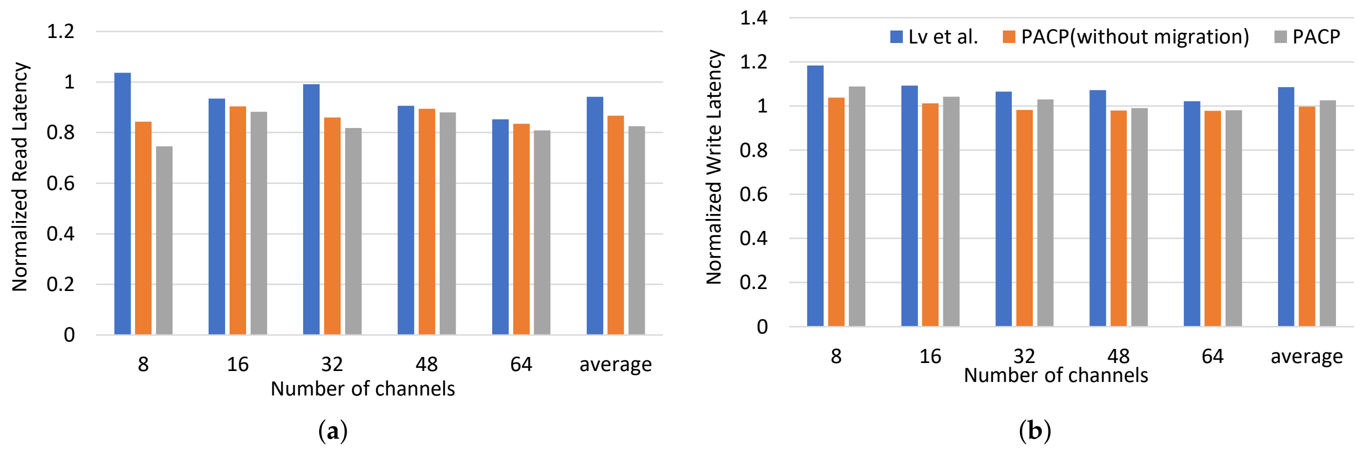


Figure 10. Average latency comparison for all workloads: (**a**) read; (**b**) write.

Despite the fact that both the Lv et al. scheme and the *PACP* reduce the read latency, Figure 10b shows that the write latency increases due to the delay caused by the write. *PACP* caused an increase in latency of 2.7% because it causes an additional program compared to *PACP* (without migration). With *PACP*, the write latency was increased by 2.5% relative to the baseline. However, the write latency was reduced by 6.0% relative to the Lv et al. method.

5. Conclusions

With the development of multi-level cell technology in which multiple bits can be stored in NAND flash memory cells, R/W interference has become a more significant problem. This paper proposed a *PACP* technique that exploits the request access pattern to mitigate R/W interference and increase parallel flash-memory resource utilization. Furthermore, an additional performance improvement was achieved through GC-leveraged migration for pages vulnerable to interference. *PACP* can be implemented in the FTL of the SSDs without dedicated hardware, requiring approximately 0.1% of the SSD storage capacity. Extensive evaluations showed that *PACP* significantly reduces read and write latency compared to the previous solution. The future work of *PACP* includes a wear leveling method for dynamically updated, partitioned SSD.

Author Contributions: Investigation, D.S. and H.J.L.; methodology, H.J.L.; project administration, H.J.L. and T.H.H.; supervision, D.S. and T.H.H.; validation, H.J.L.; writing—original draft, H.J.L.; writing—review and editing, H.J.L. and T.H.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by National R&D Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT under grant 2020M3H2A1076786; in part by the Ministry of Trade, Industry and Energy (MOTIE) under Grant 20011074; in part by Institute of Information & communications Technology Planning & Evaluation (IITP) funded by the Korea government (MSIT) under grant 2019-0-00421 (AI Graduate School Support Program, Sungkyunkwan University); and in part by the MOTIE and KEIT (20010560, Development of system level design and verification for in storage processing architecture based on phase change memory).

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

References

1. Zhao, C.; Jin, L.; Li, D.; Xu, F.; Zou, X.; Zhang, Y.; Song, Y.; Wei, H.; Chen, Y.; Li, C.; et al. Investigation of threshold voltage distribution temperature dependence in 3D NAND flash. *IEEE Electron Device Lett.* **2018**, *40*, 204–207. [CrossRef]
2. Takai, Y.; Fukuchi, M.; Kinoshita, R.; Matsui, C.; Takeuchi, K. Analysis on heterogeneous ssd configuration with quadruple-level cell (qlc) nand flash memory. In Proceedings of the 2019 IEEE 11th International Memory Workshop (IMW), Monterey, CA, USA, 12–15 May 2019; pp. 1–4.
3. 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, Heights, NY, USA, 8–12 June 2009; pp. 338–349.
4. Hu, Y.; Jiang, H.; Feng, D.; Tian, L.; Luo, H.; Zhang, S. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In Proceedings of the International Conference on Supercomputing, Tucson, AN, USA, 31 May–4 June 2011; pp. 96–107.
5. Jung, M.; Kandemir, M.T. An Evaluation of Different Page Allocation Strategies on High-Speed SSDs. In Proceedings of the HotStorage, Boston, MA, USA, 13–14 June 2012.
6. Hu, Y.; Jiang, H.; Feng, D.; Tian, L.; Luo, H.; Ren, C. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE Trans. Comput.* **2012**, *62*, 1141–1155. [CrossRef]
7. Tavakkol, A.; Mehrvarzy, P.; Arjomand, M.; Sarbazi-Azad, H. Performance evaluation of dynamic page allocation strategies in SSDs. *ACM Trans. Model. Perform. Eval. Comput. Syst. (TOMPECS)* **2016**, *1*, 1–33. [CrossRef]
8. Wu, G.; He, X. Reducing SSD read latency via NAND flash program and erase suspension. In Proceedings of the FAST, San Jose, CA, USA, 12–15 February 2013; Volume 12, p. 10.
9. Park, S.; Shen, K. FIOS: A fair, efficient flash I/O scheduler. In Proceedings of the FAST, San Jose, CA, USA, 12–15 February 2013; Volume 12, p. 13.

10. Zhang, Q.; Feng, D.; Wang, F.; Xie, Y. An efficient, QoS-aware I/O scheduler for solid state drive. In Proceedings of the 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, Zhangjiajie, China, 13–15 November 2013; pp. 1408–1415.
11. Gao, C.; Shi, L.; Zhao, M.; Xue, C.J.; Wu, K.; Sha, E.H.M. Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives. In Proceedings of the 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 2–6 June 2014; pp. 1–11.
12. Huang, J.; Badam, A.; Caulfield, L.; Nath, S.; Sengupta, S.; Sharma, B.; Qureshi, M.K. {FlashBlox}: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized {SSDs}. In Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST 17), Santa Clara, CA, USA, 27 February–2 March 2017; pp. 375–390.
13. Huang, S.M.; Chang, L.P. Providing SLO compliance on NVMe SSDs through parallelism reservation. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2018**, *23*, 1–26. [CrossRef]
14. Lv, Y.; Shi, L.; Li, Q.; Xue, C.J.; Sha, E.H.M. Access characteristic guided partition for read performance improvement on solid state drives. In Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 20–24 July 2020; pp. 1–6.
15. Kim, J.; Kim, D.; Won, Y. Fair I/O scheduler for alleviating read/write interference by forced unit access in flash memory. In Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems, Virtual, 27–28 June 2022; pp. 86–92.
16. Nanavati, M.; Wires, J.; Warfield, A. Decibel: Isolation and Sharing in Disaggregated {Rack-Scale} Storage. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA, USA, 27–29 March 2017; pp. 17–33.
17. Song, X.; Yang, J.; Chen, H. Architecting flash-based solid-state drive for high-performance I/O virtualization. *IEEE Comput. Archit. Lett.* **2013**, *13*, 61–64. [CrossRef]
18. 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.
19. Li, Q.; Shi, L.; Xue, C.J.; Wu, K.; Ji, C.; Zhuge, Q.; Sha, E.H.M. Access characteristic guided read and write cost regulation for performance improvement on flash memory. In Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST 16), Santa Clara, CA, USA, 22–25 February 2016; pp. 125–132.
20. Wu, S.; Zhang, W.; Mao, B.; Jiang, H. HotR: Alleviating read/write interference with hot read data replication for flash storage. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 9–13 March 2019; pp. 1367–1372.
21. Tavakkol, A.; Gómez-Luna, J.; Sadrosadati, M.; Ghose, S.; Mutlu, O. {MQSim}: A Framework for Enabling Realistic Studies of Modern {Multi-Queue}{SSD} Devices. In Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST 18), Oakland, CA, USA, 12–15 February 2018; pp. 49–66.
22. Subramani, R.; Swapnil, H.; Thakur, N.; Radhakrishnan, B.; Puttaiah, K. Garbage collection algorithms for nand flash memory devices–an overview. In Proceedings of the 2013 European Modelling Symposium, Manchester, UK, 20–22 November 2013; pp. 81–86.
23. Han, K.; Shin, D. Remap-based Inter-Partition Copy for Arrayed Solid-State Drives. *IEEE Trans. Comput.* **2021**. [CrossRef]
24. Kim, J.; Lim, K.; Jung, Y.; Lee, S.; Min, C.; Noh, S.H. Alleviating garbage collection interference through spatial separation in all flash arrays. In Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC 19), Renton, WA, USA, 10–12 July 2019; pp. 799–812.