IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 August 2022, accepted 3 September 2022, date of publication 8 September 2022, date of current version 19 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3204994

RESEARCH ARTICLE

Analysis and Mitigation of Patterned Read Collisions in Flash SSDs

YUHUN JUN[1], JAEHYUNG PARK[2], JEONG-UK KANG[3], AND EUISEONG SEO[2]
[1]Department of Semiconductor Display, Sungkyunkwan University, Suwon-si, Gyeonggi-do 16419, South Korea
[2]Department of Computer Science and Engineering, Sungkyunkwan University, Suwon-si, Gyeonggi-do 16419, South Korea
[3]Samsung Electronics Co., Ltd., Yongin-si, Gyeonggi-do 17113, South Korea

Corresponding author: Euiseong Seo (euiseong@skku.edu)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant through the Korean Government, Ministry of Science and ICT (MSIT), Research on Edge-Native Operating Systems for Edge Micro-Data-Centers, under Grant 2021-0-00773; and in part by the National Research Foundation of Korea (NRF) under Grant 2021R1A2C2004976.

ABSTRACT A modern flash solid-state drive (SSD) achieves superb throughput by accessing its flash memory dies in parallel. To obtain parallelism in processing writes, the flash translation layer (FTL) of an SSD is allowed to allocate physical pages from idle dies or dies with low loads. However, since the die that holds the page to be read is determined in advance, when multiple read requests head for one die, read collisions occur, and the read operations involved in the collisions must be serialized. These read collisions lead to a significant prolongation of read latency and, thus the degradation of throughput. As the density of flash dies increases, more pages are stored on a die, which is expected to result in more frequent read collisions. Unlike the write collisions that the DRAM buffer can absorb, read collisions directly affect user-experienced latency. Therefore, the severity of the problem is further increased. In this paper, we identify the impact of read collisions on performance with real-world traces. We also propose an approach that distinguishes patterned read collisions from accidental ones and mitigates the performance impact of patterned read collisions. By replicating frequently occurring pages involved in patterned read collisions, the proposed approach improves the average and tail read latency of an SSD. The evaluation through simulation with the 34 MSR Cambridge traces showed that 12 traces out of 34 earned read performance improvement larger than 10% and up to 37%. In addition, the tail read latency of 15 traces was reduced by at least 20% and by up to 53%. Only two traces showed negligible degradation in average and tail read latency by around 1%.

INDEX TERMS Flash memory, read performance, resource contention, SSDs, tail latency.

I. INTRODUCTION

A modern flash SSD is equipped with multiple NAND flash dies. A die can process a primitive flash memory operation: erase, program (write), and read, one at a time. A die is able to read or write at a few tens of MB/s, and flash SSDs can achieve high performance by operating a significant number of dies in parallel. Consequently, maintaining a high degree of die-level parallelism is crucial for the performance of modern SSDs [1].

Maintaining high die-level parallelism in write operations can be carried out effortlessly using the flash translation layer (FTL). The FTL determines a physical page number (PPN) to store a logical page number (LPN), which is an address of an externally visible page and manages the mapping between LPNs and PPNs. During write operations, the FTL tries to allocate PPNs, which will accommodate the incoming LPNs, from as many dies as possible to increase the die-level parallelism [2], [3].

A sequential read operation that accesses multiple pages written in such a way is naturally spread evenly across multiple dies, resulting in maximum read performance. However, since die's location on a target physical page during a read operation is predetermined, a series of read operations can be concentrated on some of the dies. If an outstanding request exists in a die, even if other dies are available, the outstanding request will have to wait its turn.

The associate editor coordinating the review of this manuscript and approving it for publication was Artur Antonyan.

Compared to read operations, erase operations or write operations have a significantly lower impact on the user-perceived performance. Accordingly, to give priority to read operations in a die command queue, reordering of commands in the command queue [4] and suspending an in-progress write or erase operation [5], [6] have been proposed. However, if there exists a preceding read operation, the following read operation has no choice but to wait until the previous one is finished because they are operations of equal priority. Therefore, the read operations biased on a small set of dies significantly degrade die-level parallelism, and consequently, extend read latency and degrade the throughput of the SSD.

Over the last four generations, the capacity of a flash die has increased 8 folds, from 128Gb to 1024Gb [7], [8], [9], [10]. This increased capacity of a die enables the use of fewer dies to provide the same level of capacity for an SSD. Even considering the increasing capacity of an SSD, the number of dies equipped in an SSD is less likely to increase. Consequently, the degree of line-blocking among read operations, which we call read collisions, will remain the same or worsen in the near future. Despite the importance of read collisions' impact on read latency and throughput, few studies have attempted to characterize or mitigate the negative effects of die-level read collisions.

In this paper, we analyze the occurrence patterns of read collisions in various I/O traces and the performance degradation caused by them. Based on the analysis, we present a methodology that separates *patterned read collisions*, which occur due to the LPN-to-PPN mappings mismatching the host I/O patterns, out of accidental ones. Lastly, on top of this, we propose a mitigation scheme of patterned read collisions. Our approach replicates the pages that are frequently involved in patterned read collisions to less congested dies.

The proposed approach was implemented in MQSim [11], a flash SSD simulator reflecting the internal-parallelism of the modern SSD architecture. The MSR Cambridge block I/O trace set [12], [13] was used for the analysis of read collisions and the evaluation of the proposed approach.

The rest of this paper is organized as follows. Section II presents the read collisions observed in various workloads. Based on the obtained implications, in Section III, we propose a read collision mitigation scheme, and, evaluate the proposed approach in Section IV. After introducing the related work in Section V, Section VI concludes the research.

II. BACKGROUND AND MOTIVATION

A. INTERNALS OF MODERN FLASH SSDs

A NAND flash die provides read performance of only a few hundred MB/s, which is far below the transfer rate of several GB/s provided by the PCIe bus. For example, one flash die used by a recent commercial SSD reads 18 KB per 45us, resulting in 409 MB/s, but the SSD can provide over 7 GB/s of read performance [14]. To provide this maximum performance, more than 17 dies in the SSD must read simultaneously.

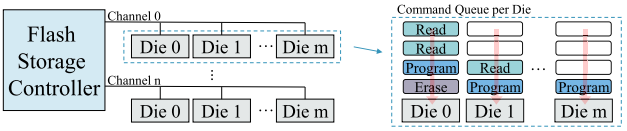


FIGURE 1. Die-level parallelism in a flash SSD.

TABLE 1. Capacity increment over generations of 3D stacked NAND flash dies.

No. of stacked layers	32 [7]	48 [8]	64 [9]	96 [10]
Bits per cell	TLC			QLC
Capacity (Gb)	128	256	512	1024
Areal density (Gb/mm^2)	1.86	2.62	3.98	13.8
Release year	2016	2017	2018	2020

Hence, modern flash SSDs are designed to operate multiple dies in parallel, as shown in Fig. 1. A controller has multiple data channels, and each channel connects to multiple dies. As a die does not fully occupy its data channel during operation, the controller can operate other dies on the channel through die-interleaving. Accordingly, in each n channel operating in parallel, m dies can simultaneously operate.

To keep the dies in the working state as much as possible, recent SSDs maintain a command queue for each die [4]. Operations are inserted into the queue of the target die and processed one by one. As previously stated, read operations take precedence over erase or write operations. Consequently, the use of per-die command queues enables high die utilization as well as improved read latency.

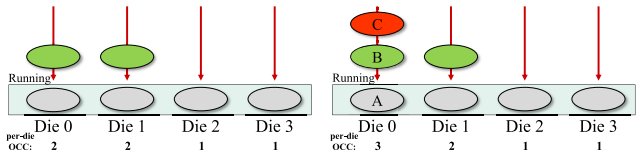
When sequential read requests are issued by an application, because the data locations are predetermined, their flash read commands can be queued to a small set of dies, which results in extended read latency and degraded throughput. For example, as shown on the right side of Fig. 1, read collisions occur at die 0 by consecutively inserting read operations to its command queue.

The occurrence frequency of read collisions is proportional to the number of read operations and inversely proportional to the number of dies in an SSD. Table 1 shows the characteristics of a 3D-stacked NAND flash die across generations. The capacity of a die has increased twice each generation. The SSD manufacturers naturally want to reduce the cost of building an SSD. Therefore, the number of flash dies installed in an SSD of the same capacity tends to decrease [15], [16].

As previously mentioned, read operations directly affect the end-performance of applications because, unlike write operations, buffering of operations is not possible. Therefore, it is expected that the importance of analyzing and mitigating the adverse performance impact of read collisions will keep continuing to grow as the density of flash dies advances.

B. CATEGORIZATION OF READ COLLISIONS

Although read collisions are adverse to the performance of an SSD, we cannot completely remove all of them. When



(a) Balanced collisions (b) Imbalanced collision at die 0

FIGURE 2. Categorization of read collisions.

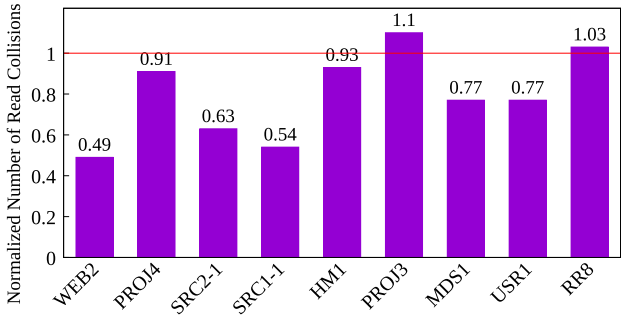


FIGURE 3. Number of read collisions normalized to that of a random read workload with the same level of OCC.

all dies are performing read operations as shown in Fig. 2a, an incoming request will yield a read collision regardless of which die it will be assigned to. If the requests are evenly deployed across all dies, the difference between the maximum outstanding command count (OCC) and the minimum OCC of the die command queues should be at most one, according to the pigeon hole principle [17]. A collision that occurs on a die of which OCC is different from the minimum OCC of the SSD by one or less is called a *balanced collision*, as shown in Fig.2a. The balanced collisions are inevitable and do not degrade throughput, as all dies can keep operating in parallel. Naturally, the prolongation of read latency caused by the balanced collisions should be acceptable.

When the OCC of each die may differ by two or more, *imbalanced collisions* may occur. When a collision occurs at a die and its OCC differs from the minimum OCC of the SSD by two or more, this collision is the result from the biased distribution of operations at the die level. As shown in Fig. 2b, if the read operation C of die 0 could be performed on die 2 or die 3, the read latency would have been reduced. Furthermore, if an imbalanced collision leaves a die idle while there are queued operations in other dies, the throughput will be degraded as well.

The read collisions can be categorized into two types based on their causes. The first is *accidental read collisions*, in which two or more unrelated read operations are accidentally dispatched to a single die. The second is *patterned read collisions* that occur when two or more pages, which are mostly read together by an application, are unfortunately stored in the same die. Although the FTL tries to distribute the contiguous logical pages to as many dies as possible at the time of writing their data, if a sequence of contiguous logical pages are written with a time gap, or the write sequence and the data read sequence are different from each other, the patterned read collisions may occur.

Both accidental read collision and patterned read collision are frequent. However, accidental read collisions are supposed to occur equally across all dies. Therefore, it is less likely that an accidental read collision results in an imbalanced collision than a patterned read collision. In addition, the depth of nested imbalanced collisions occurring at a single die from accidental read collisions tends to be shallow. Consequently, accidental collisions can be considered benign, while patterned collisions are malignant. It is because a patterned collision can occur frequently depending on the application behavior and is likely to end up with an imbalanced collision, which significantly degrades both read latency and throughput.

C. ANALYSIS OF READ COLLISION OCCURRENCES

To analyze the frequency and characteristics of read collisions occurring in operations, through simulation with MQSim [11], we observed the internal activities of an SSD while running diverse workloads. Among the 36 MSR Cambridge workload traces [13], we analyzed eight workloads of which read proportion is over 90%.

For comparison, we also analyzed a synthetic random read workload, which issues 4 KB random reads to keep SSD's OCC at 8. This is denoted as RR8.

A PPN is allocated to an LPN when the LPN is first accessed by a trace. The PPN allocation in our simulation was performed by the CWDP algorithm [18], which determines the channel, way, and die of the PPN based on the remainder of the LPN divided by the number of channels, ways, and dies, respectively. The OCC changes and collision patterns were monitored every 10 μs. Table 2 shows the observed simulation results.

To determine whether read collisions caused by a workload are due to coincidence or access patterns, we need a basis to be compared against. For this, we created a hypothetical workload for each MSR workload trace that continuously maintains the same level of OCC as the corresponding workload but distributes read operations randomly over all dies. The number of read collisions that occurred to the total number of reads when executing this random-read workload is denoted as the *expected collision ratio* in Table 2, and the actual collision ratio of the workload is presented as the *measured collision ratio*. The expected collision ratio tells the read collision ratio caused by pure coincidence under the same level of OCC changes.

Fig. 3 shows, for each workload, the number of measured read collisions normalized to the number of expected read collisions. Given that RR8 is a purely random read workload, it showed a value of 1.0 with a marginal error range, which shows the validity of the collision patterns of the synthetic random read workload.

All seven workloads, except PROJ3, showed fewer collisions than their random-read counterparts. Most of their read

TABLE 2. Read collision characteristics observed during the execution of the MSR traces listed in order of read proportion.

Workloads	RR8	WEB2	PROJ4	SRC2-1	SRC1-1	HM1	PROJ3	MDS1	USR1
Read Proportion	100%	99.25%	98.52%	97.86%	95.26%	95.34%	94.82%	92.88%	91.48%
No. of Die Operations	967430	22755762	15788014	2861418	129830290	1010317	3010108	7232363	177531189
Average OCC	7.8	3.7	12.5	4.8	2.4	14.2	19.8	4.0	5.1
Expected Collision Ratio	83.7%	65.6%	69.5%	67.7%	42.5%	66.2%	55.1%	79.5%	69.3%
Measured Collision Ratio	86.0%	32.0%	62.2%	42.5%	22.8%	61.3%	60.6%	61.0%	53.5%
Imbalanced to All Collisions	100.0%	99.9%	98.8%	91.1%	99.3%	99.9%	100.0%	99.2%	97.1%
Imbalanced Collision pairs	2382491	540624	2288090	1268814	12516416	26823	170431	3376854	12678038
Mean Repetition of Pairs	1.00	9.20	4.56	1.73	1.51	37.89	24.69	1.54	12.47

requests showed a certain degree of sequentiality, and this reduced the occurrences of read collisions. However, despite the average OCC of these seven workloads were significantly smaller than the number of dies, 48% of read operations on average caused collisions. In the cases of HM1 and PROJ4, the numbers of collisions are close to their random read counterparts, respectively. PROJ3 generated even more collisions than the random read workload. This shows that when the LPN-to-PPN mappings are not favorably set to the application's request patterns, a large amount of patterned collisions may occur.

Imbalanced to All Collisions on Table 2 shows the proportion of imbalanced collisions to all read collisions. While executing RR8, idle dies must exist because it always maintains eight outstanding requests while there are 16 dies in the SSD. Therefore, all collisions produced by RR8 must be imbalanced ones. The average OCC of all workloads, except PROJ3, was smaller than 16. Consequently, 90% of total collisions were identified as imbalanced ones.

Even with the increased OCC, biased assignments of operations to some of the dies were consistently observable. The average OCC of PROJ3 was greater than the number of dies. However, for the aforementioned reason, one or more dies stayed idle at 87% of observation time points, and 100% of its collisions were determined as imbalanced ones.

To analyze the impact of frequent collision patterns on overall collision occurrences, we counted the number of *collision pairs* that appeared during the simulation of each workload. A collision pair is a pair of two LPNs involved in a collision. In Fig. 2b, read operations A and B are already queued for die 0. When operation C is queued, it produces an imbalanced collision. At that point, we can extract two collision pairs, <A,C> and <B,C>, from that collision. Given that <A,B> also contributes to the imbalance, <A,B>, is recorded together with <A,C> and <B,C>. The *Imbalanced Collision Pairs* in Table 2 shows the number of collision pairs that contributed to the imbalanced collisions. We also measured the number of occurrences of each collision pair during the execution of each workload.

We measured the number of repetitions of each collision pair and the number of requests deployed to each die observed during the execution of each workload. Fig. 4 shows the measured results. RR8 is not included because every die handles the same number of read operations and every collision pair appeared only once.

TABLE 3. Relative standard deviation of the number of read operations performed on each die during execution of each workload.

Workloads	Relative Standard Deviation
HM1	0.168
PROJ3	0.121
USR1	0.086
WEB2	0.047
SRC1-1	0.023
PROJ4	0.021
MDS1	0.018
SRC2-1	0.016

The upper row of Fig. 4 shows the workloads with the high mean repetition counts of collision pairs while the lower row presents the ones with low mean repetition counts. For HM1, USR1, and WEB2, only 5% of frequently observed collision pairs account for 80% of all collision occurrences. Especially, some of the pairs happened a few thousand times in these three workloads. In PROJ3, 50,000 pairs out of 170,000 accounted for 62% of collisions.

The relative standard deviation (RSD) of the number of read operations performed on each die is calculated as the ratio of their standard deviation to the mean number of read operations for a die. Table 3 shows the RSDs of the workloads presented in Fig. 4, and these numbers represent the degree of bias in the distributions of read operations over the dies.

The workloads in the upper row of Fig. 4 are commonly characterized by uneven distributions of read operations over the dies. On the contrary, the workloads placed in the lower row commonly showed relatively uniform distribution of read operations over the dies. However, even in these workloads, the top few collision pairs showed significantly high repetition counts in comparison to the others. These highly repetitive collision pairs indicate the existence of patterned collisions.

The above observations reveal the existence of patterned collisions in the read-dominant workloads. The collision pairs they generated showed an overwhelmingly higher occurrence frequency than the collision pairs generated by accidental collisions. If these few frequent collision pairs are eliminated, the read latency can be significantly improved.

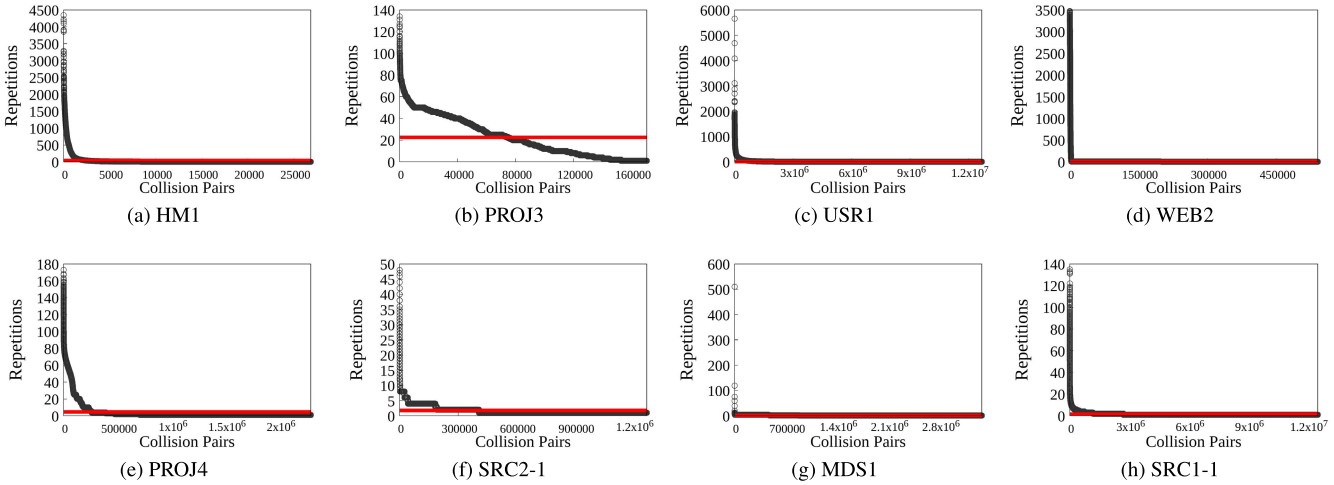


FIGURE 4. Repetition count of each collision pair sorted in descending order, and the distribution of read operations over the 16 dies. The upper row presents the workloads with high mean repetition counts while the lower row shows the workloads with low mean repetition counts.

III. OUR APPROACH

In this study, we propose a method to identify repetitive imbalanced read collision patterns with a small memory footprint and little computation overhead. We also propose a read collision mitigation scheme that can reduce imbalanced read collisions by replicating logical pages that are repetitively involved in read collisions to properly select dies.

A. DETECTION OF MALIGNANT COLLISION PAIRS

When a collision pair repetitively causes imbalanced collisions, we can get rid of the imbalanced collisions by moving one of the pages in the collision pair to other die. This approach requires the SSD to identify the frequently recurring collision pairs without any hints given from the host. Therefore, we propose a cost-benefit analysis model that estimates the expected reduction in imbalanced read collisions by moving one page of a collision pair to other die. If the expected benefit of moving a page of a certain collision pair exceeds the cost to move the page, the mitigation scheme will replicate one of the two pages in the pair to a die that is least likely to cause new collisions by the moved page.

In the proposed scheme, as shown in Fig. 5, the FTL maintains a collision pair list for each die to record the occurrences of patterned collision pairs observed during operation. This list records the number of collisions caused by each collision pair that occurred at the corresponding die based on the categorization criteria explained in Section II-C. When issuing a read command to a die, the FTL updates the collision pair list of the die with the information about the collision, if occurred, generated by the read operation. One collision pair occupies one entry in the list, and a new entry is created when a collision pair that is not in the list is observed. The pages of the collision pairs with high occurrences in this table are subject to the aforementioned page replication to other die.

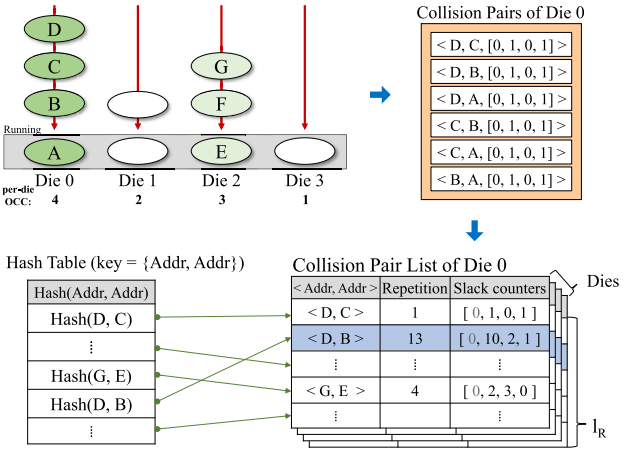


FIGURE 5. Management of per-die collision pair lists.

For page replication, not only the page to be replicated but also the destination die to accommodate the page must be determined. A suitable candidate dies for the page replication is the one with the least amount of outstanding read operations statistically in its queue when the page to be replicated was involved in collisions. Therefore, an entry of a collision pair list records the slack counter for each die. The slack counter of a die is increased by one when the collision pair of the entry generates a collision, and the die's OCC is smaller by at least two than that of the list owner die. As a result, when replicating a page in a collision pair, the die with the largest slack counter in the collision pair entry signifies the most desirable destination for the page replication.

An entry in the collision pair list stores a pair of page addresses, their occurrence counters, and per-die slack counters. The entry is only valid in the die. Therefore, each page address field stores only 26 bits out of a 30 bit PPN address, which is sufficient to address a page in a die. 16 bits are

sufficient for each slack counter based on our experimental results because they are periodically reset by the mechanism to be described later and the overflow of a counter does not cause any critical problem. Therefore, in a 1 TB SSD having 16 dies, one entry occupies 38.5 bytes. The number of entries in the collision pair lists increases the DRAM usage. Therefore, the number of entries of a list must be limited to a predefined threshold value, l_R. For this, the list is implemented as a doubly-linked list. When an entry is updated, the entry node will be inserted right after the head node of its list. Naturally, the tail end of a list will have the least recently used entry of the list. If the number of entries in a list exceeds l_R, the entry at the tail will be removed. To expedite the search time of an entry, the FTL looks up the position of an entry with a hash table, which uses the combination of the page addresses as its hash key.

If a page in a collision pair that caused 4,000 imbalanced collisions for a time interval, which can be observed in Fig. 4a, is moved to another die with a significantly lower per-die OCC, it is expected that the die will yield approximately 4,000 fewer imbalanced collisions in the next time interval. When a page is moved to another die, we can define the probability of collision reduction from this replication, p, as (1) where d is the slack count of the destination die, and the occurrence count of the target pair is o. The page replication will reduce the occurrences of collisions that this page pair is involved in by the rate of p.

$$p = \frac{d}{o} \tag{1}$$

For example, page D and page B in die 0 of Fig. 5 were involved in 13 imbalanced collisions. The slack count of die 1, 10 times in total, is the largest among all in the entry. If either page D or B was stored in die 1, 10 out of 13 collisions could have been avoided. Therefore, when die 1 is chosen as the destination of page D or page B, p becomes 0.77.

Suppose that the number of collisions observed during a certain period is repeated in the next period. In this case, the gain from a page replication is the number of collisions multiplied by p. Therefore, the read latency benefits that can be obtained by replicating a page in a collision pair is defined by (2) where t_{READ} is the time required to finish a read operation on a die.

$$\begin{aligned} benefit &= o \times t_{READ} \times p \\ &= t_{READ} \times d \end{aligned} \tag{2}$$

We can determine whether it is desirable to replicate a page of a collision pair by comparing this predicted gain with the cost of the page replication. The destination die cannot perform any other read operations while it is copying the page to be replicated. If r read operations are issues to the destination die every second, we can expect that read operations as many as $R_{conflict}$ defined in (3) are stalled by the write operation in the destination die. Given that a read operation may arrive at any time during the write operation, the sum of read operations' stall time, which is the cost for the page replication, can be estimated by (4).

$$R_{conflict} = r \times t_{PROG} \tag{3}$$

$$cost = R_{conflict} \times t_{PROG} \times 0.5 \tag{4}$$

The proposed scheme compares the benefit of page replication with its cost by using (2) and (4) when a collision pair entry is updated. If the benefit exceeds the cost, the FTL initiates page replication by selecting the victim page from the pair. Between the two pages in the pair, the page that appears more in the remaining pairs in the collision pair list of the die will be chosen as the victim page. Subsequently, the FTL replicates the victim page to the destination die.

After replicating the victim page to the destination die, the victim page will no longer generate collisions in the source die. Therefore, the collision pair list of the source die needs to be updated to reflect this. However, deleting only the collision pair of the victim page from the list does not fully reflect the changes in collision patterns from the replication. For example, in Fig. 5, the E and G pair in die 2 would not have been recorded as an imbalanced collision pair if F did not cause a collision. Hence, if F has been selected as the victim page and replicated, the E and G pair must be removed from the list as well. Since detecting and removing such derived pairs are technically challenging, we clear all collision pair entries of the source die when a page replication occurs in a die.

B. REPLICATION AND COLLISION AVOIDANCE

To perform a page replication, it is necessary to read the data of the victim page. Given that the decision to replicate is made when the victim page's read operation is in the command queue, the victim page will eventually be read from the flash memory. Therefore, the FTL waits for the read operation in the queue to finish and starts the replication process. By doing so, the FTL reuse the page contents stored in the buffer, and the proposed scheme does not incur additional read overhead for the replication.

Subsequently, the write operation to the destination die is performed with a lower priority than host-issued operations. That is, if host-issued writes or reads exist in die's command queue, a replication write will be placed after the host-issued operations through command queue reordering [11].

After replication, two PPNs will exist that are mapped to the replication target LPN. Therefore, it is necessary to change the FTL structure so that the LPN can point to both PPNs. For this, the proposed scheme builds a replication map, which is an indirect mapping data structure, for a replicated page as illustrated in Fig. 6. A replication map stores the LPN, the original PPN, and the replication PPN.

Each entry of FTL's L2P table has an additional flag that denotes whether the LPN of the entry has the replicated page. When replicating a page, the replication flag of the corresponding LPN entry in the L2P table is set. If the replication flag of an L2P entry is set, the PPN address field of the entry

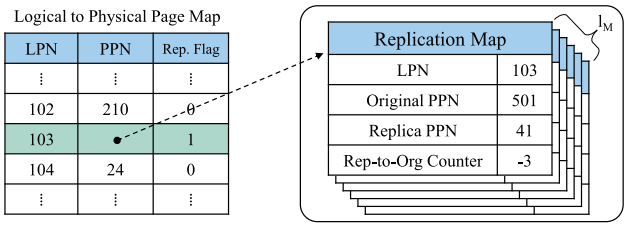


FIGURE 6. Indirect mapping structure of replication map.

is interpreted as the address of the replication map entry, not as the PPN address.

When a request is to read a replicated page, the FTL first finds out the die that has the original PPN and the one that has the replicated PPN by referring to the replication map. In turn, the FTL compares the OCC of the die having the original page with that of the die having the replicated page. Between the two, the FTL accesses the page in the die with the lower OCC. At this point, if the replicated page is selected, the *replication-to-original* counter of the replication map is incremented by one. Conversely, if the original page is chosen, the counter is decremented by one. This counter will be later used to choose a page to leave between the original and replicated pages when the number of replicated pages in an SSD exceeds a predefined threshold.

As the number of replicated pages increases, the amount of additional flash memory for storing them expands. In addition, the size of the replication maps stored in DRAM also grows. Therefore, to limit the DRAM and flash memory space consumption, the proposed scheme defines the maximum allowable number of replicated pages, l_M, and deletes replicated pages from the least recently used ones when the number of replicated pages exceeds l_M. Because the flash space used for storing the replicated pages is provisioned in the over-provisioning (OP) area of an SSD, l_M should be determined so that the replicated pages can fit in the OP space. As the number of replicated pages increases, the effective OP space decreases. It is known that reducing OP space ends up with the escalated write-amplification factor, performance degradation, and shortened life span of an SSD [19], [20]. Consequently, the proper value of l_M should be selected to minimize the adverse effects from the proposed read collision mitigation scheme while maximizing its benefit. The impact of the two parameters of the proposed scheme, l_R and l_M, on the performance gain is analyzed in Section IV.

Replication maps are stored in a doubly-linked list. When a read occurs on a replication map entry, the entry is inserted next to the head node of the list. Consequently, an entry that has not been accessed for the longest time is located at the tail of the list. As stated, a replication map entry is deleted along with its replicated or original physical page when the number of replicated pages exceeds l_M. Before deletion, as previously stated, the FTL decides the page to be left between the original and the replicated ones based on the replication-to-original counter value. If the value of the counter is zero or positive, the FTL will keep the replicated page and invalidate the original one. In the opposite case, the FTL will invalidate the replicated one and keep using the original one. The LPN field of the victim replication map entry is used to accordingly update victim's L2P mapping entry. Finally, the FTL removes the victim replication map from the memory.

A replication map consists of one LPN and two PPN fields, which take 30 bits each, one 32-bit counter, two 32-bit pointers for managing the replication map linked list, and six padding bits for 32-bit alignment of the replication map data structure. To summarize, a replication map entry takes 24 bytes of the DRAM space.

Like the L2P mapping table, the replication maps must be persistent. Therefore, it must be stored in the flash memory when the power is off. In an 1 TB SSD, the replication maps take only up to 32 MB even when l_M is set to 2% of the total capacity. In comparison to the L2P mapping table, which occupies 1 GB in 1 TB SSD, the space usage of the replication maps is negligible. Consequently, syncing the replication maps to the flash memory can be carried out together with the L2P table sync.

Given that the replication map data structure is 4-byte aligned, only 30 bits are required to store the address of a replication map. Therefore, the current PPN field of the L2P table can be used without any modifications to store the address of the replication map instead of the PPN. The only modification to the conventional L2P table is the addition of the replication flag of which size is just 1 bit. To summarize, the space overhead of the metadata management for the proposed mitigation scheme is negligible.

IV. EVALUATION

The proposed scheme was implemented in MQSim, which is able to simulate per-die operation handling, for evaluation. The parameters used in the simulation are introduced in Table 4. The evaluation was conducted with 34 traces out of 36 MSR workload traces set [12], [13], excluding two write-only workloads.

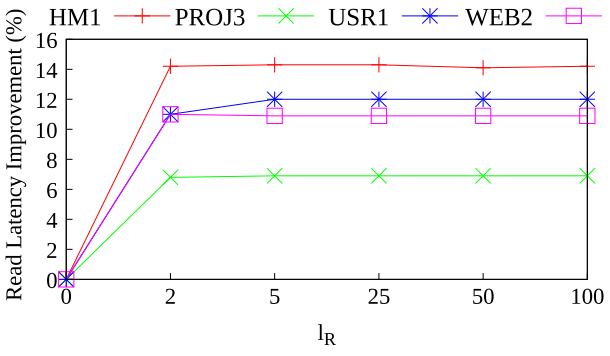
A. PARAMETER SELECTION

As explained in Section III, the proposed approach has two configurable parameters: l_R and l_M. l_R is the maximum number of tracked collision pairs per die, and l_M is the maximum number of replicated pages that can exist in an SSD. This section analyzes the impact of these parameters on the effectiveness of the proposed approach.

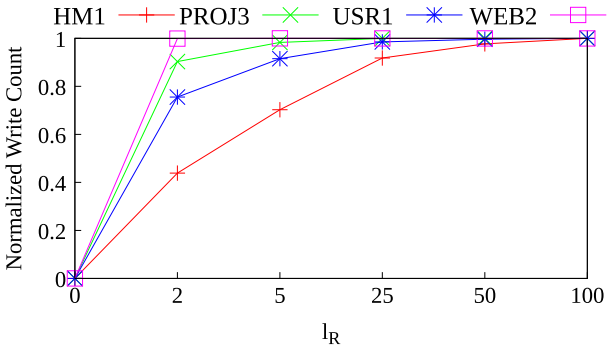
To analyze the effects of l_R, we evaluated our approach with varying l_R. For this analysis, we assumed that l_M is indefinitely large. Fig. 7 shows the average read latency and the numbers of replications observed for the top four workloads with the largest mean repetition counts, which were identified in Section II-C, respectively.

TABLE 4. Parameters of the simulated SSD.

SSD	Capacity	1 TB
	Host Interface	PCI Gen3 x4
	FTL L2P Mapping	Page Mapping [21], [22]
	Channel Count	8
	Dies per Channel	2
	OP Area [23]	7% of Total Capacity
Flash Memory [9]	Bits per Cell	TLC
	Page Size	16 KB + 2 KB (Parity)
	Read Time	60 μs
	Program Time	700 μs
	Erase Time	3500 μs
	P/E Suspend Time	50 μs
	Channel Speed	1 Gbps



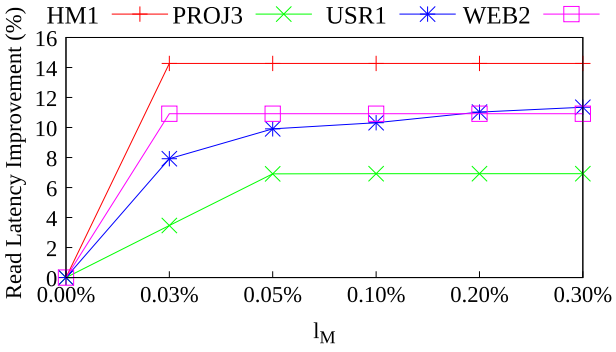
(a) Read Latency Improvement



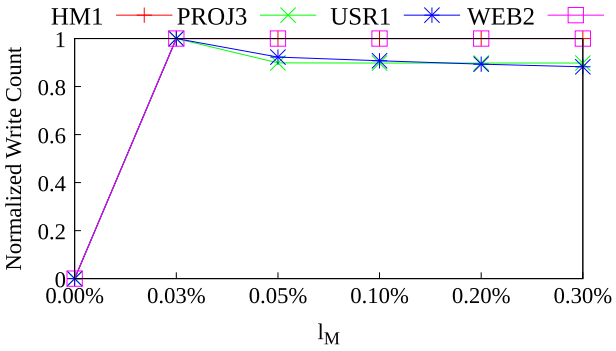
(b) Number of Replications

FIGURE 7. Read latency improvement and number of replications performed depending on varying l_R when l_M is set to be infinitely large.

When l_R is 0, the proposed method does nothing because no collision pairs are kept track of. As l_R increases, more collision pairs are monitored as the candidates of replication. Fig. 7a shows the degree of read latency improvement depending on the varying value of l_R. The read latency was improved by up to 14.6%, which was observed from HM1, as l_R increased. However, the degree of performance improvement saturated in all four workloads after l_R exceeded five. Although PROJ3 showed the larger mean repetition count of collision pairs than USR1 and WEB2, the performance gain for PROJ3 was smaller than theirs. This is because, of USR1



(a) Read Latency Improvement



(b) Number of Replications

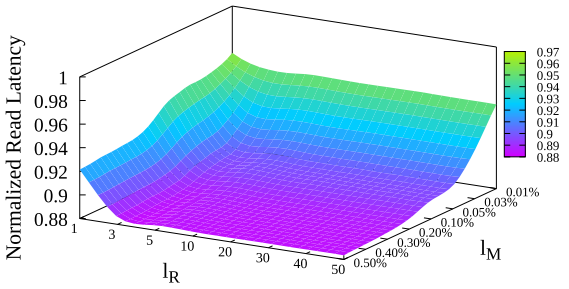
FIGURE 8. Read latency improvement and number of replications performed depending on varying l_M when l_R is set to five. l_M is denoted as the percentage to the total SSD capacity.

and WEB2, a small numbers of collision pairs are extremely popular in comparison to the others, whereas the repetition count distribution over the collision pairs of PROJ3 shows a gentle curve as previously illustrated in Fig. 4.

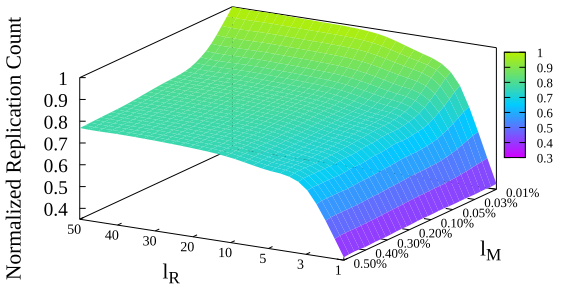
The amount of read latency improvement was trivial when l_R increased from 5 to 100. Rather, as l_R increased, the number of replicated writes increased, as shown in Fig. 7b. This is because, as the number of observed collision pairs increased, collision pairs not having high collision rates were more likely to be selected for the replication.

These results tell that a large performance gain can be obtained by removing only a small number of frequently appearing collision pairs.

Fig. 8 shows the average read latency and number of page replication writes under the proposed approach with varying l_M. In this analysis, based on previous observations, l_R was set to 5. As shown in Fig. 8a, the degree of performance gain rapidly grew as l_M increased until it reached 0.05%, and diminished after that. However, as shown in Fig. 8b, when l_M is smaller than 0.05%, the replicated pages are frequently replaced with each other especially for PROJ3 and USR1, and this results in the increased number of replications. l_M larger than 0.2% of the total capacity did not significantly reduce the number of replication operations nor increase the performance gain.



(a) Average Read Latency



(b) Number of Replications, l_R axis is reversed

FIGURE 9. Normalized read latency of USR1 while varying both l_R and l_M.

Fig. 9 shows the average read latency and replication counts of USR1, which showed the large number of replications, but this time, we varied l_M as well as l_R. When l_M was set to be 0.2% of the total capacity or larger, the performance gain saturated in the region where l_R is greater than five. This assures that only a few collision pairs largely contribute to the total number of read collisions.

We performed the same experiment shown in Fig. 9 to the other MSR traces. Using the unlimited amount of space for the page replication through setting l_R and l_M to ∞ improved 1% or smaller additional performance gain in comparison to the case with $l_R = 5$ and $l_M = 0.2\%$ of the total capacity in 15 out of the 34 traces. Rather, 17 traces showed similar or better read latency under the $l_R = 5$ and $l_M = 0.2\%$ setting by selectively replicating highly influential pages.

Fig. 10 shows the average read latency under both settings, respectively, of the top 8 workloads having the largest performance gap between the two settings. The average read latency under the proposed scheme was normalized to its original value. Only PROJ4 and PRN1 showed 2.62 percent points and 1.46 percent points performance degradation, respectively, by limiting l_R and l_M. This is because, in these workloads, a relatively large fraction of collision pairs yield large collision counts and l_M is not sufficiently large to accommodate them, resulting in frequent and unnecessary replacement of replicated pages. Limiting the resource usage for the page replication, on the contrary, improved the read latency further

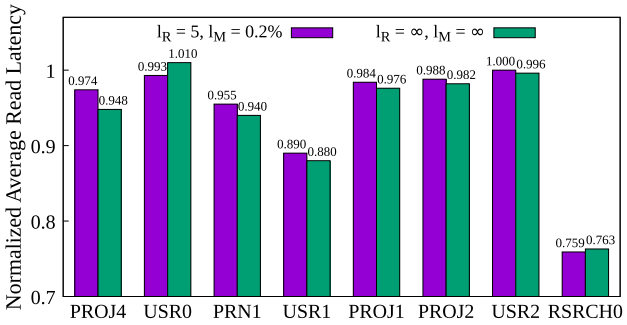


FIGURE 10. Normalized average read latency under two different parameter settings of the top 8 workloads having the largest performance gap between the two settings.

by 1.69 percent points from the unlimited resource usage configuration for USR0.

From these results, we can conclude that the impact of these two parameters on the effectiveness of the proposed scheme is inconsiderable as long as they are not extremely small. We set l_R to 5 and l_M to 0.2% of the total capacity for the following experiments. With these settings, the collision pair lists occupy 5 Kbytes in DRAM, and the replication maps take 3.23 Mbytes in DRAM.

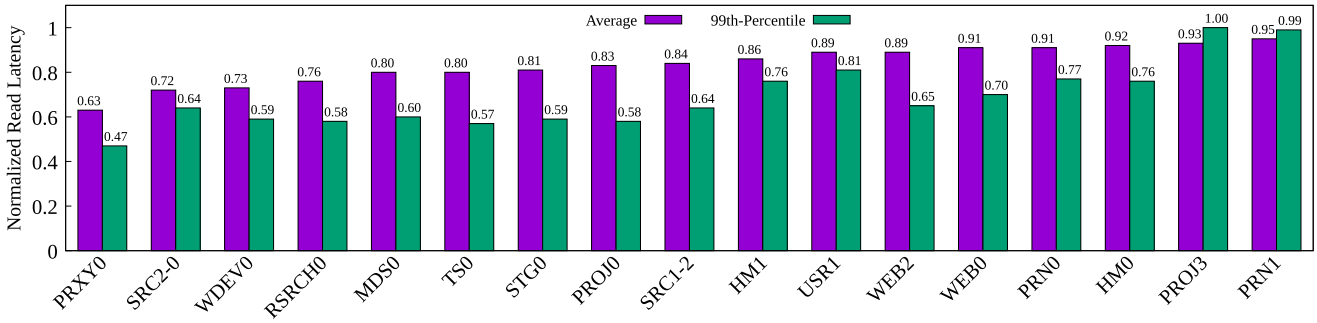
B. EFFECTIVENESS ANALYSIS

The user-experiencing response time of a service is determined by the slowest component of the service architecture [24], and a storage device is one of the primitive components. Consequently, the tail latency of an SSD critically determines the overall service response time while the average latency leads to the throughput of the service.

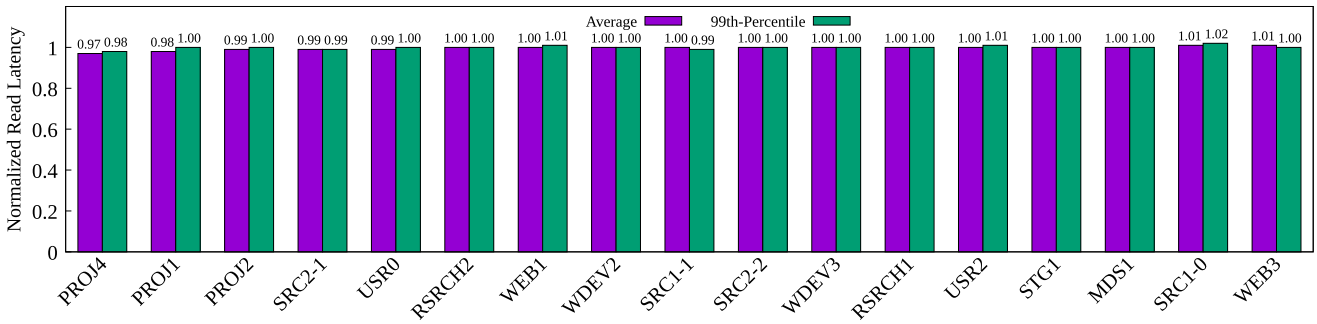
We measured both average and 99th-percentile read latency of the MSR workload traces. Fig. 11 shows the results normalized to that of the original FTL. The workloads are listed in the descending order of average latency improvement. Our approach reduced the average latency for PRXY0 by 37%, and 12 out of the 34 workloads earned more than 10% of read latency improvement. The improvement of the tail latency was greater than that of the average latency. By removing a large portion of the read collisions, the proposed approach could reduce the 99th-percentile read latency by up to 53%, and 15 workloads earned 20% or larger tail latency improvement.

Table 5 shows the changes in the number of imbalanced collisions and the number of replications performed by the proposed scheme. The workloads are listed in the descending order of the collisions reduction ratio ratio on the table. The top 16 workloads in Fig. 11a matches with the top 16 workloads of the table. These workloads share the common property that a small number of collision pairs largely contribute to their total collision counts, and thus a large portion of their collisions, ranging from 10 to 45%, was removed by the proposed approach.

The imbalanced collisions of PRXY0 and SRC2-0, which showed significant performance improvement, were reduced



(a) Top 17 workloads with the largest performance gains



(b) Bottom 17 workloads with the smallest performance gains

FIGURE 11. Average and 99th-Percentile read latency normalized to the original FTL.

by approximately 40%. Notice that the number of reduced collisions per replication indicates the degree of the read latency improvement earned from the proposed approach. For example, a replication during WEB2 and PRXY0 removed 87 and 29 collisions, respectively, on average. The average reduction of imbalanced collisions per replication for the workloads in Fig. 11a was 15.34.

Workloads with relatively few or mostly accidental collisions will not benefit from the proposed approach. Because of this, the proposed approach did not show any significant changes in both average and tail latency for the bottom 17 workloads in Fig. 11b.

Table 5 confirms why the bottom 17 workloads had little performance gain from the proposed approach. Contrary to the intent, the proposed approach increased the number of read collisions of STG1, MDS1, SRC1-0 and USR0. This tells that the replicated pages produced the same or larger number of collisions in the destination dies. However, the performance of STG1 and MDS1 were barely changed. The average read latency of SRC1-0 was prolonged by 0.5% and the tail latency was worsen by 1.7%. Despite the increase in read collisions, the average read latency of USR0 improved because the collisions were spread across all dies, reducing the wait latency on each die.

As previously stated, for some workloads, our approach could not meaningfully reduce the number of collisions. However, because the proposed approach selectively performs replications based on the cost-benefit comparison model, it did not noticeably degrade the read performance even for the workloads with unfavorable characteristics. The only two workloads that the proposed approach worsened the performance were SRC1-0, of which the performance loss was explained previously, and WEB3. WEB3's average read latency was prolonged by 0.7% while its tail latency remained the same. This is because WEB3 is a workload with a small number of read requests, around 10000 or so, and it is difficult to distinguish between pattern collisions and accidental collisions in these small read requests.

The larger the mean repetition is shown in Fig. 4, the greater the number of reduced collisions per replication. As a result, the proposed scheme showed a considerable performance improvement for those workloads. In the case of USR1, a significantly higher performance improvement was obtained compared to its mean repetition of collision pairs because only 1.8% of its collision pairs accounted for 50% of its collision occurrences. Naturally, the proposed scheme could not reduce significant imbalanced collisions for the workloads that obtained little performance improvement. On the contrary, the number of imbalanced collisions was even increased by applying the mitigation scheme. This is because the cause of imbalanced collisions for these workloads was more accidental collisions than repeated host I/O patterns.

The proposed scheme incurs additional writes during page replication. These writes may interfere with the host-issued write operations and degrade the write performance. We

TABLE 5. Change in the number of imbalanced collisions, and the number of replications of each workload caused by the proposed scheme.

	Original	Our Appr.	Change	# of Repl.
PRXY0	1621471	887017	-45.3%	24602
SRC2-0	588577	368177	-37.4%	10417
WDEV0	687519	432423	-37.1%	15423
TS0	404871	256012	-36.8%	18194
RSRCH0	298682	190125	-36.3%	9351
STG0	404187	259984	-35.7%	10221
PROJ0	563821	367340	-34.8%	19820
MDS0	251301	169756	-32.4%	8444
HM1	1016264	704192	-30.7%	17250
SRC1-2	905164	663523	-26.7%	21812
WEB2	4973465	3668995	-26.2%	14923
HM0	709505	532308	-25.0%	49212
WEB0	540993	411293	-24.0%	23406
PRN0	1044003	904225	-13.4%	55999
USR1	158037937	138059990	-12.6%	2306889
PRN1	9403308	8451724	-10.1%	607759
PROJ3	4208654	3917000	-6.9%	212234
PROJ1	15144549	14437805	-4.7%	1399767
WEB1	22299	21297	-4.5%	1068
WEB3	1595	1533	-3.9%	68
PROJ4	10444705	10046809	-3.8%	897243
SRC1-1	18911488	18441783	-2.5%	335869
PROJ2	25547544	24960225	-2.3%	2429163
RSRCH2	149174	146771	-1.6%	24410
USR2	8460061	8375493	-1.0%	610011
SRC2-1	2200816	2189242	-0.5%	48135
SRC2-2	2310640	2309906	0.0%	8546
WDEV2	56	56	0.0%	1
RSRCH1	2	2	0.0%	0
WDEV3	0	0	0.0%	0
STG1	394608	394635	0.0%	5754
MDS1	5212874	5234471	0.4%	528315
SRC1-0	30303775	30482230	0.6%	1934716
USR0	3702948	3859772	4.2%	17659

measured the average latency of host write requests for the 34 traces and found out that the average write latency was prolonged by less than 0.01% in all workloads. This extremely low write overhead was due to the reduction of read collisions along with preferential scheduling of host writes.

V. RELATED WORK

In Flash SSDs, write and erase operations are 60 times and 100 times slower than read operations, respectively. Thus, die-level collisions with write and erase operations critically increase the latency of read operations. Several studies have been conducted to resolve this issue.

Jung *et al.* [4] proposed physically addressed queuing (PAQ), a request scheduler that avoids resource contention resulting from shared SSD resources. It places a command queue for each die and allows reordering operations in the queue to fully utilize multiplane mode operations. Gao *et al.* [25] also proposed parallel issue queuing (PIQ), a host-side I/O scheduler, to minimize flash die operation collisions. The PIQ schedules I/O requests without collisions within the same batch and I/O requests with collisions into different batches. Hence, multiple I/O requests in one batch can be fulfilled simultaneously. Given that PIQ is implemented on the host side, it can utilize the rich resources of the host system.

Significant read latency delays for host-issued read operations can occur intermittently, as garbage collection incurs many write and die collisions. To prevent this problem, Jung *et al.* [26] proposed an I/O scheduler that predicts the GC operation of the FTL in the host interface layer inside the device. The proposed I/O scheduler can avoid the collisions between writes for garbage collection and host-issued read operations. Also, Shahidi *et al.* [27] proposed a technique for proactively performing GC on a plane in an idle state by utilizing plane-level parallelism.

Studies have been conducted to reduce the performance loss ascribed to read collisions by exploiting additional space. Flash on Rails [28] operates an SSD group that only performs read in an array of SSDs, which redundantly stores data in an RAID-like manner to ensure latency of write and mixed read. The set of SSDs in charge of read operations is changed over time and selected by a sliding window. Hot Data Replication (HotR) [29] outsources popular read data to a surrogate space, such as a dedicated spare flash chip or an OP area. If available, conflicting read requests are served by the surrogate flash space. The concept of page replication to avoid collisions is similar to our approach. However, HotR immediately replicates the page when the read operation to it collides with a write request, whereas our study selects the victim pages based on collision statistics. This incurs excessive amount of additional write operations and space overhead. In addition, our approach not only replicates but also permanently migrates a page to another die if beneficial.

He *et al.* [30] also proposed a page replication scheme that counts the number of blocking operations per page and replicates a page to other die when the blocking probability of the operations to the page reaches a certain threshold. Although their rationale behind the approach is similar to ours, keeping tack of the per-page blocking probability imposes a large amount of additional data structures to the per-page metadata. In addition, they did not consider the expected collision probability when choosing the target die for a page replication. Through the carefully designed metadata structures, our approach can choose the page and the destination die for a replication to maximize the benefit with negligible overhead.

Li *et al.* [31] performed fast write with a high error rate to reduce waits caused by collision. The page where a collision occurs for subsequent read operations is reprogrammed at an idle state. Reprogramming the page lowers the error rate and reduces ECC processing time, and, as a result, reduces the waiting time due to collisions.

Lie *et al.* [15] proposed a novel single-operation-multiple-location (SOML) read operation that performs several small intra-chip read operations to different locations

simultaneously, so that multiple requests can be serviced in parallel, thereby mitigating the performance degradation from read collisions. Although this can radically reduce the number of read collisions, it imposes a significant level of complexity on the architecture of flash memory chips and SSD controllers.

VI. CONCLUSION

The ever increasing density of flash memory shrinks the number of flash dies equipped in an SSD with the same capacity. The reduced number of flash dies increases die-level read collisions.

This paper categorizes die-level read collisions into two: benign balanced collisions and malignant imbalanced collisions. Most of imbalanced collisions are induced by the repetitive host I/O patterns, and adversely affect the read latency and throughput of a SSD. This paper analyzed the patterns of read collisions observed in various workloads and their impact to the performance.

Based on the analysis, this paper proposed a read collision mitigation scheme that replicates the pages frequently involved in frequently recurring imbalanced read collisions to appropriate dies. Our evaluation showed that the proposed scheme improved the average read latency by 10% or more for 12 out of the 34 MSR Cambridge traces, and the tail read latency by at least 20% for 15 of them when using only 0.2% of the total storage capacity for the replicated pages. Only two workloads were adversely affected by the proposed approach, however, their performance degradation was negligible.

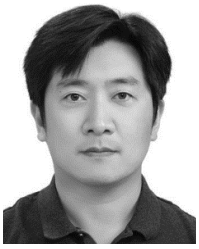
REFERENCES

[1] F. Chen, B. Hou, and R. Lee, "Internal parallelism of flash memory-based solid-state drives," *ACM Trans. Storage*, vol. 12, no. 3, pp. 1–39, 2016.
[2] J.-U. Kang, J.-S. Kim, C. Park, H. Park, and J. Lee, "A multi-channel architecture for high-performance NAND flash-based storage system," *J. Syst. Archit.*, vol. 53, no. 9, pp. 644–658, Sep. 2007.
[3] S.-Y. Park, E. Seo, J.-Y. Shin, S. Maeng, and J. Lee, "Exploiting internal parallelism of flash-based SSDs," *IEEE Comput. Archit. Lett.*, vol. 9, no. 1, pp. 9–12, Jan. 2010.
[4] M. Jung, E. H. Wilson, and M. Kandemir, "Physically addressed queueing (PAQ): Improving parallelism in solid state disks," in *Proc. Annu. Int. Symp. Comput. Archit. (ISCA)*, 2012, pp. 404–415.
[5] S. Kim, J. Bae, H. Jang, W. Jin, J. Gong, S. Lee, T. J. Ham, and J. W. Lee, "Practical erase suspension for modern low-latency SSDs," in *Proc. Annu. Tech. Conf. (ATC)*, 2019, pp. 813–820. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/kim-shine
[6] G. Wu and X. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. Conf. File Storage Technol. (FAST)*, 2012, p. 10. [Online]. Available: https://www.usenix.org/conference/fast12/reducing-ssd-read-latency-nand-flash-program-and-erase-suspension
[7] W. P. Jeong *et al.*, "A 128 Gb 3b/cell V-NAND flash memory with 1 Gb/s I/O rate," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 204–212, Jan. 2016.
[8] D. Kang *et al.*, "256 Gb 3 b/cell V-NAND flash memory with 48 stacked WL layers," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 210–217, Jan. 2017.
[9] C. Kim *et al.*, "A 512-GB 3-b/cell 64-stacked WL 3-D-NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 124–133, Jan. 2018.
[10] H. Huh *et al.*, "13.2 A 1Tb 4b/cell 96-stacked-WL 3D NAND flash memory with 30MB/s program throughput using peripheral circuit under memory cell array technique," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, Feb. 2020, pp. 220–221.
[11] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. Conf. File Storage Technol. (FAST)*, 2018, pp. 49–65. [Online]. Available: https://www.usenix.org/conference/fast18/presentation/tavakkol
[12] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, pp. 1–23, 2008, doi: 10.1145/1416944.1416949.
[13] D. Narayanan, A. Donnelly, and A. Rowstron, "MSR Cambridge traces (SNIA IOTTA trace set 388)," in *SNIA IOTTA Trace Repository*, G. Kuenning, Ed. Colorado Springs, CO, USA: Storage Networking Industry Association, Mar. 2007. [Online]. Available: http://iotta.snia.org/traces/block-io?only=388
[14] Samsung Semiconductor. *Samsung SSD 980 PRO*. [Online]. Available: https://www.samsung.com/semiconductor/minisite/ssd/product/consumer/980pro/
[15] C.-Y. Liu, J. B. Kotra, M. Jung, M. T. Kandemir, and C. R. Das, "SOML read: Rethinking the read operation granularity of 3D NAND SSDs," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Operating Syst.*, Apr. 2019, pp. 955–969, doi: 10.1145/3297858.3304035.
[16] D. Hong, M. Kim, G. Cho, D. Lee, and J. Kim, "GuardedErase: Extending SSD lifetimes by protecting weak wordlines," in *Proc. Conf. File Storage Technol. (FAST)*, 2022, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/fast22/presentation/hong
[17] W. A. Trybulec, "Pigeon hole principle," *J. Formalized Math.*, vol. 2, no. 199, pp. 1–5, 1990.
[18] A. Tavakkol, P. Mehrvarzy, M. Arjomand, and H. Sarbazi-Azad, "Performance evaluation of dynamic page allocation strategies in SSDs," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 2, pp. 1–33, Jun. 2016.
[19] P. Desnoyers, "Analytic modeling of SSD write performance," in *Proc. 5th Annu. Int. Syst. Storage Conf. (SYSTOR)*, 2012, pp. 1–10, doi: 10.1145/2367589.2367603.
[20] B. Van Houdt, "A mean field model for a class of garbage collection algorithms in flash-based solid state drives," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 191–202, Jun. 2013.
[21] A. Ban, "Flash file system," U.S. Patent 5 404 485, Apr. 4, 1995.
[22] C. Intel, "Understanding the flash translation layer (FTL) specification," Intel Corp., Santa Clara, CA, USA, Appl. Note AP-684, 1998.
[23] K. Smith. (2012). *Understanding SSD Over-Provisioning*. Flash Mamory Summit. [Online]. Available: https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2012/20120822_TE21_Smith.pdf
[24] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble, "Tales of the tail: Hardware, OS, and application-level sources of tail latency," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2014, pp. 1–14.
[25] C. Gao, L. Shi, M. Zhao, C. Jason Xue, K. Wu, and E. H.-M. Sha, "Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives," in *Proc. 30th Symp. Mass Storage Syst. Technol. (MSST)*, Jun. 2014, pp. 1–11.
[26] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 289–300, 2014.
[27] N. Shahidi, M. T. Kandemir, M. Arjomand, C. R. Das, M. Jung, and A. Sivasubramaniam, "Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2016, pp. 561–572.
[28] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, "Flash on rails: Consistent flash performance through redundancy," in *Proc. Annu. Tech. Conf. (ATC)*, 2014, pp. 463–474. [Online]. Available: https://www.usenix.org/conference/atc14/technical-sessions/presentation/skourtis
[29] S. Wu, W. Zhang, B. Mao, and H. Jiang, "HotR: Alleviating read/write interference with hot read data replication for flash storage," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1367–1372.
[30] B. He, J. X. Yu, and A. C. Zhou, "Improving update-intensive workloads on flash disks through exploiting multi-chip parallelism," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 1, pp. 152–162, Jan. 2015.
[31] Q. Li, L. Shi, C. Gao, K. Wu, C. J. Xue, Q. Zhuge, and E. H.-M. Sha, "Maximizing IO performance via conflict reduction for flash memory storage systems," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 904–907.

YUHUN JUN received the B.Eng. degree in electrical electronic computer engineering from Dankook University, South Korea, in 2009, and the M.S. degree in computer science and engineering from Sungkyunkwan University, South Korea, in 2016, where he is currently pursuing the Ph.D. degree. He is currently a Senior Engineer with the Flash Solution Development Team, Memory Business Unit, Samsung Electronics Company Ltd. His research interests include operating systems, embedded systems, and flash-based storage systems.

JAEHYUNG PARK received the B.S. degree in computer science and engineering and architecture engineering from Sungkyunkwan University, South Korea, in 2022, where he is currently pursuing the M.S. degree with the Department of Computer Science and Engineering. His research interests include embedded systems and flash-based storage systems.

JEONG-UK KANG received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1998, 2000, and 2007, respectively. He is currently a Master with the Memory Business Unit, Samsung Electronics Company Ltd. His research interest includes flash-based storage systems.



EUISEONG SEO received the B.S., M.S., and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 2000, 2002, and 2007, respectively. He was an Assistant Professor at the Ulsan National Institute of Science and Technology (UNIST), South Korea, from 2009 to 2012, and a Research Associate at Pennsylvania State University, from 2007 to 2009. He is currently a Professor with the Department of Computer Science and Engineering, Sungkyunkwan University, South Korea. His research interests include system software, embedded systems, and cloud computing.