GCMix: An Efficient Data Protection Scheme against the Paired Page Interference

SANG-HOON KIM, Virginia Polytechnic Institute and State University and KAIST
JINHYUK LEE, Samsung Electronics Co. and Sungkyunkwan University
JIN-SOO KIM, Sungkyunkwan University

In multi-level cell (MLC) NAND flash memory, two logical pages are overlapped on a single physical page. Even after a logical page is programmed, the data can be corrupted if the programming of the coexisting logical page is interrupted. This phenomenon is called paired page interference.

This article proposes a novel software technique to deal with the paired page interference without any additional hardware or extra page write. The proposed technique utilizes valid pages in the victim block during garbage collection (GC) as the backup against the interference, and pairs them with incoming pages written by the host. This approach eliminates undesirable page copy to backup pages against the interference. However, such a strategy has an adverse effect on the hot/cold separation policy, which is essential to improve the efficiency of GC. To limit the downside, we devise a metric to estimate the benefit of GCMix on-the-fly so that GCMix can be adaptively utilized only when the benefit outweighs the overhead. Evaluations using synthetic and real workloads show GCMix can effectively deal with the paired page interference, reducing the write amplification factor by up to 17.5%compared to the traditional technique, while providing comparable I/O performance.

CCS Concepts: • **Information systems → Information storage technologies**; **Flash memory**;

Additional Key Words and Phrases: Flash memory, flash memory cells, multi-level cells, paired page interference, locality

ACM Reference format:
Sang-Hoon Kim, Jinhyuk Lee, and Jin-Soo Kim. 2017. GCMix: An Efficient Data Protection Scheme against the Paired Page Interference. *ACM Trans. Storage* 13, 4, Article 37 (November 2017), 23 pages.
https://doi.org/10.1145/3149373

1 INTRODUCTION

During the last decade, NAND flash memory, or flash, has become one of the most popular storage media. Flash has been gaining popularity with many attractive characteristics such as high I/O performance, light weight, small form factor, low power consumption, low heat emission, and shock robustness. As a result, many flash-based devices have been introduced, replacing traditional

This work was supported by the National Research Foundation of Korean (NRF) grant funded by the Korea Government (MSIT) (No. 2016R1A2A1A05005494).
Authors' addresses: S.-H. Kim, The Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University, 250 Perry St, Blacksburg, VA 24061, USA; email: sanghoon@vt.edu; J. Lee, Memory Division, Samsung Electronics, Co., 1 Samsungjeonja-ro, Hwaseong 18448, South Korea; email: jinhyuk79.lee@samsung.com; J.-S. Kim, College of Information and Communication Engineering, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon 16419, South Korea; email: jinsookim@skku.edu.

https://doi.org/10.1145/3149373

storage devices such as hard disk drives (HDDs). Especially, solid state drives (SSDs) are increasingly adopted in enterprise environments due to their high I/O performance [2, 4, 20, 21, 24, 25, 34].

Flash is comprised of an array of *cells*, in which logical values are stored. The cells that are connected through the same word line forms a *page*, which is the unit of read and programming (write) operations. As the flash technology evolves, each cell can store more than one bit, enabling multiple logical pages to coexist in a physical page [9, 27]. For instance, in the multi-level cell (MLC) flash memory, the least significant bits (LSB) of cells in a physical page form a logical page called an LSB page. Similarly, the most significant bits (MSB) form an MSB page. These two pages are called paired pages [27].

For a given physical page, the LSB page needs to be programmed prior to the MSB page. While programming the MSB page, the cells in the physical page go through a transient state, in which the original data of the cells cannot be identified. Therefore, if programming an MSB page is interrupted for any reason, the data of the paired LSB page cannot be recovered and it is lost unexpectedly. This phenomenon is called *paired page interference* [12]. The paired page interference is inherent in the MLC flash memory and must be dealt with for reliability and durability. Many counter-interference techniques have been suggested [12, 15, 17, 29, 36, 37]. These techniques, however, require reserved space and/or extra page copy, which increases hardware overhead and manufacturing cost, and hurts the lifespan of MLC flash memory.

In this article, we propose a novel technique to deal with the paired page interference. Modern SSDs usually employ a software layer called flash translation layer (FTL). FTL reclaims invalidated, i.e., logically overwritten or discarded, pages through the garbage collection (GC) process. To produce a free block during GC, FTL picks up a victim block, copies valid pages in the victim block to other block(s), and then erases the victim block. Our scheme called *GCMix* pairs a valid page in the victim block with a page from an incoming write request and programs them together on a physical page. The LSB page is programmed with the data of the valid page from the victim block, whereas the MSB page is with the data of the incoming write request. Under the proposed GCMix scheme, even if programming the MSB page is interrupted and corrupts the data of the LSB page, the original data of the LSB page can be recovered from the victim block. Thus, the paired page interference can be avoided without any additional hardware or extra page copy.

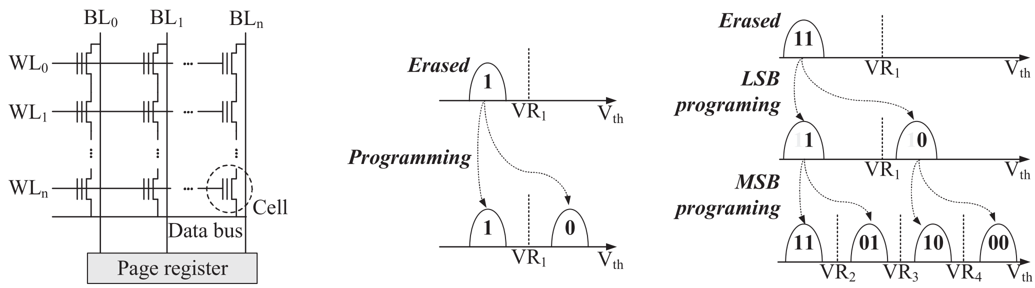
GCMix, however, writes hot incoming write pages and cold valid pages in the same block. This makes it difficult to separate hot data from cold data, which is known to greatly improve the efficiency of GC [3, 7, 9, 16, 18, 23, 26]. Thus, GCMix may induce an adverse influence on the efficiency of GC and hurt the performance and lifetime of the flash device. To address this problem, we propose a sophisticated method based Dynamic dAta Clustering (DAC) [3] to determine whether GCMix is beneficial to the current workload or not. FTL can adaptively apply GCMix based on the decision. Evaluation results obtained from a trace-driven simulator show that the proposed GCMix scheme can tackle the paired page interference with a reduced amount of flash writes, by up to 17.3%, without harming the I/O performance of FTL.

The rest of this article is organized as follows. Section 2 introduces NAND flash memory and FTL. Section 3 reviews related work to deal with the paired page interference. Section 4 presents GCMix and a novel scheme to estimate the benefit of GCMix depending on workloads. Evaluation results are provided in Section 5. Finally, Section 6 concludes the article.

2 BACKGROUND

2.1 NAND Flash Memory

NAND flash memory, or flash shortly, is a non-volatile storage medium that stores and retrieves data electrically. As Figure 1(a) illustrates, the flash typically consists of an array of transistors,



(a) Cell structure of NAND flash (b) Programming SLC NAND (c) Programming MLC NAND

Fig. 1. The structure and programming of NAND flash memory.

called *cells*. The cell can retain electric charges [32] in the floating gate and the amount of charges determines the logical value of the cell. A bit line (BL) connects a column of the cell array and constitutes a bit of the data bus. A word line (WL) connects a row of the array through the control gate of the cells. The cells that share the same word line form a *page*, which is the unit of read and programming operations. To read a page, the corresponding word line is activated, and then the logical values of the selected cells are transferred to the page register through the data bus in a parallel manner. Programming (i.e., flash write) is carried out in a similar way; the corresponding word line is driven to a high voltage and the bit lines are driven to particular voltages so that cells can trap more electric charges. If the page is once programmed, the page can be read repeatedly. The page, however, cannot be programmed again unless it is erased. The *flash erase* is performed in a unit of *erase block* or *block* shortly, which typically consists of hundreds of pages. As the cell experiences the repeated program/erase (P/E) cycles, the thin oxide layer between the floating gate and the silicon substrate deteriorates and eventually becomes unable to electrically isolate the floating gate. This phenomenon is called *flash wear out*, and determines the lifespan of NAND flash memory.

In the single-level cell (SLC) flash memory, each cell represents one logical bit (0 or 1). Figure 1(b) illustrates how the SLC flash memory handles the logical bit in the cell. The x-axis represents the charge level observed from the bit line, and the y-axis represents the distribution of cells corresponding to the charge level. When a block is erased, every cell in the block exhibits the charge level lower than the reference voltage VR_1 and represents the logical value "1." The cell can be programmed to represent "0" by injecting electrons into the floating gate so that the charge level of the cell exceeds VR_1.

The multi-level cell (MLC) flash memory can represent two logical bits (00, 01, 10, or 11) per cell. Programming the MLC flash memory is more complicated than that of the SLC flash memory [37]. Figure 1(c) illustrates the programming of the MLC flash memory. Programming the MLC flash memory is composed of two steps. The first programming step determines the logical value of the LSB of the cells, which is called *LSB programming*. The reference voltage VR_1 determines the logical value of each cell. The second programming step, called *MSB programming*, determines the logical value of the MSB of the cell, and three reference voltages (VR_2, VR_3, and VR_4) differentiate the logical bits of the cells. As the MLC flash memory needs to precisely control the charge levels of the cells, the MLC flash memory exhibits slower operation time and worse reliability than the SLC flash memory [34]. Typically, the MLC flash memory guarantees less than one-tenth P/E cycles of the SLC flash memory and exhibits more than 2× slower program time.

In the MLC NAND flash memory, two logical pages coexist in a physical page. An *LSB page* is the logical page whose contents are comprised of the LSBs of the cells. In the same way, the most significant bits of the cells constitute the contents of an *MSB page*. These two related pages are mutually called *paired pages* [9, 27]. As programming MLC flash is performed in two steps, the data needs to be written on the LSB page prior to the MSB page. During programming the MSB page, the cells in the physical page go through a transient state, from which the original value of the paired LSB page cannot be recovered. Therefore, whenever MSB programming is interrupted by any reasons, such as sudden power-off or programming failure, the original data of the paired LSB page is lost. This phenomenon is called *paired page interference* [12] and it is an inherent problem in the MLC NAND flash memory.

2.2 Flash Translation Layer (FTL)

Due to the erase-before-write characteristic and the discrepancy between the read/program unit (i.e., page) and the erase unit (i.e., block), traditional block I/O interfaces cannot be supported directly on the flash memory. To address the problem, flash-based devices usually employ a software layer called FTL. FTL glues the flash interface and the block I/O interface by dealing with the unique characteristics of the flash memory [1, 6, 11, 13, 19, 22]. FTL maintains the mapping information from a logical address (sector) to a flash address (block and page). FTL translates the logical addresses specified in the incoming requests to the flash addresses and performs the requested operations on the corresponding blocks and pages.

Modern FTLs handle overwrites with a log-structured scheme. The data to be updated is written to the other location and the corresponding mapping is updated accordingly. The page containing the previous version of data is marked as invalid from the mapping, and it is reclaimed through the GC process. GC is similar to the cleaning operation of log-structured file systems [14, 23]. Usually, FTL picks up a victim block and copies valid pages in the victim block to other block(s). Then, the victim block is erased and reused for handling further write requests.

FTLs can be classified into one of three categories according to the granularity of the mapping information. The block mapping FTL [1], as the name implies, maintains the mapping information in a block granularity so that the size of the mapping information is small compared to the other types of FTL. However, as an overwrite to a page may accompany the copy of an entire block, block mapping FTLs show the worst performance. The page mapping FTL [6, 11] maintains mapping information at a page granularity. As the page-level mapping is much more flexible than the block-level mapping, this type of FTL usually outperforms the other types of FTLs. However, the size of the mapping information can be enormous so that the page mapping FTL requires a huge amount of resources to handle the mapping information. The hybrid mapping FTL [13, 19, 22] adopts a hybrid approach of the block mapping FTL and the page mapping FTL. In the hybrid mapping FTL, the data in *data blocks* is managed in a block granularity. When an overwrite happens, the written data is stored in special blocks called *log blocks*, which are managed at a page granularity. Later, if all the log blocks are exhausted, some of them are converted to data blocks. In this way, the hybrid mapping FTL requires a small amount of resources while exhibiting comparable performance to the page mapping FTL in many workloads.

Due to the log-structured approaches for dealing with the erase-before-write limitation, the performance and lifetime of flash-based devices are mostly affected by the efficiency of GC, similar to log-structured file systems [23, 35]. The efficiency of GC is usually measured by the write amplification factor (WAF). It is obtained by dividing the number of flash page writes by the total number of page writes issued from the host. Hence, the smaller WAF implies better efficiency in GC. As each valid page copy accompanies a flash page write, the number of valid pages within victim blocks has significant influences on the WAF. Thus, it is very important to pick the right victim block so as to migrate the least number of valid pages during GC.

The *greedy* policy [3] selects the block containing the minimum number of valid pages as the victim block. The *cost-benefit* policy [11] considers the worthiness of the valid pages as well as the

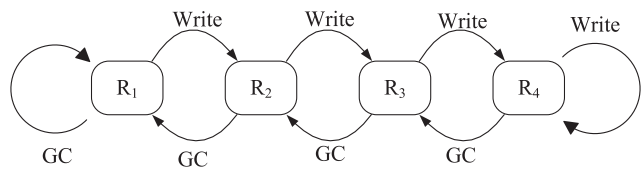


Fig. 2. Regions and page migrations in DAC.

number of valid pages. Due to the temporal locality of data references, the newly written data is considered as *hot*, as it is likely to be overwritten soon. Conversely, *cold* data, which is written and not updated for a long time, is unlikely to be updated in the near future. This implies that if FTL picks up a recently written block as the victim block and migrates valid pages in it, the migrated pages are likely to be invalidated soon and the effort to migrate the pages only entails overhead. The cost-benefit policy takes this fact into account and considers the age of each block; when a number of blocks have the same utilization, cold and old blocks are preferred to hot and young blocks for the victim block.

Several FTLs attempt to utilize the hotness of data more aggressively to improve the efficiency of GC further [3, 7, 9, 16, 18, 23, 26]. These approaches are based on the observation that separating hot data from cold data effectively decreases the average number of valid pages in victim blocks. If hot pages, such as recently written pages by the host, are clustered in blocks, most of the pages in the blocks are likely to be invalidated soon and only a small number of pages remain valid in the blocks. These blocks can be later reclaimed with a small number of valid page copies.

Among the previous studies, DAC [3], suggests a scheme to cluster pages in terms of hotness. DAC maintains a number of regions and each page belongs to one of the regions. Initially, every page belongs to the lowest region. When a page is overwritten, DAC considers the page is getting hot and promotes the region of the page by one unless the page already belongs to the highest region. When GC migrates a valid page, DAC considers the page is getting cold and demotes the region of the page by one unless the page belongs to the lowest region. Figure 2 illustrates an example of the regions and the migration of pages. In this example, there are four regions, R_1, R_2, R_3, and R_4, where R_1 denotes the coldest region and R_4 the hottest one.

3 RELATED WORK

There have been many studies for utilizing the asymmetric characteristics of LSB pages and MSB pages in the MLC NAND flash memory. Grupp et al. [5] present an FTL design to maximize the write performance by utilizing the fast write performance of LSB pages. The FTL uses multiple flash blocks to handle write requests quickly with LSB pages. The utilized LSB pages are refilled during idle time by copying valid pages for GC to MSB pages. Huang et al. [8] presents an FTL technique so-called asymmetric programming. They utilize MSB pages for storing the metadata of FTL and file systems since MSB pages exhibit a lower bit error rate than LSB pages, thereby providing better reliability. These studies, however, do not consider the paired page interference, which is inherent in the MLC NAND flash memory. Whenever the MSB programming currently in progress is interrupted, the original data of the paired LSB page can be broken. Therefore, for the sake of the reliability and durability of data in the MLC NAND flash memory, FTL should keep a copy of an LSB page elsewhere if the LSB page contains valid data and its paired MSB page is being programmed. Previous studies have suggested a number of techniques to protect data from the paired page interference [15, 17, 29, 36, 37].

Yu and Choi [37] propose a straightforward scheme, called *LSB backup*. When an MSB page needs to be programmed, the LSB backup scheme copies the data of its paired LSB page to a

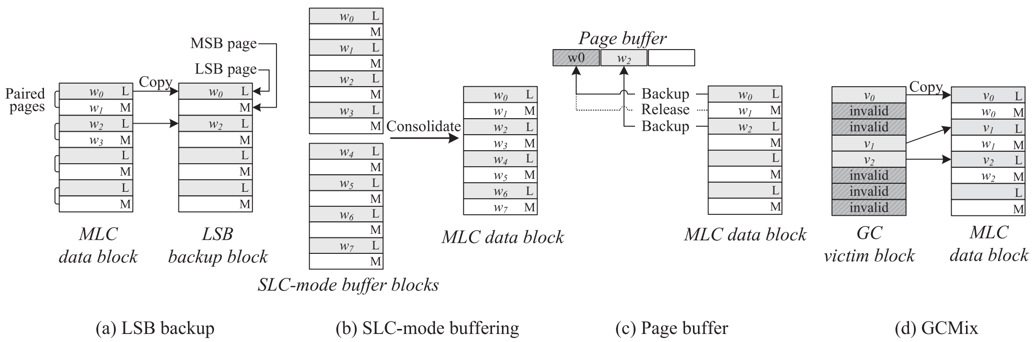


Fig. 3. The schemes to deal with the paired page interference. (w_i represents the incoming data written by the host.)

particular block called *LSB backup block*. To avoid the paired page interference in the LSB backup block itself, only the LSB pages of the LSB backup block are utilized. If the data of the original LSB page is corrupted due to the paired page interference, the original data can be recovered from the copy in the LSB backup block. After its paired MSB page is completely programmed, the copy in the LSB backup block is no longer needed.

Figure 3(a) illustrates how the LSB backup scheme addresses the paired page interference. Assume that an MLC block whose even-numbered pages are the LSB pages, which are paired with the next odd-numbered MSB pages. For example, page w_0 is paired with w_1, page w_2 is with w_3, and so on. In this case, prior to programming the MSB page w_1, the data of its paired LSB page w_0 is copied to the LSB backup block. Similarly, w_2 is copied prior to program w_3. If programming w_3 is interrupted, the data of its paired page (w_2 in this case) can be corrupted. However, it can be recovered from the copy of w_2 in the LSB backup block. This approach, however, needs to reserve a number of blocks as LSB backup blocks, and a single MSB page programming always accompanies one LSB page read and one LSB page programming. Thus, the performance and lifetime of NAND flash memory can be impaired.

Roohparvar [29] investigates a method which utilizes SLC-mode buffer blocks as Figure 3(b) outlines. A number of blocks called *buffer blocks* are operated in SLC-mode, in which only the LSB pages of the block are used to store the incoming write requests. Then, the data is migrated to MLC data blocks by merging the buffer blocks. After the merging, the buffer blocks can be erased and recycled. As the SLC-mode buffer blocks only utilize LSB pages, the paired page interference can only happen while merging the buffer blocks into MLC data blocks. In the event of the interference, the original data can be recovered from the buffer blocks that are being merged. In this way, the paired page interference can be resolved. However, as the data is always buffered in the SLC-mode blocks first and then migrated to MLC data blocks, flash writes are amplified so that the lifetime and performance can be suffering.

Yoon et al. [36] introduce the *page buffer* method. When the data is written to an LSB page, the data is duplicated to the page buffer, which is maintained in the volatile memory, and retained until its corresponding paired MSB page is completely programmed. When the paired page interference occurs, the original data can be recovered from the page buffer. Figure 3(c) illustrates the page buffer method. When programming the LSB page w_0, its data is duplicated to the page buffer. If programming the MSB page w_1 is interrupted, w_0, the data of the paired LSB page can be recovered from the backup in the page buffer. Upon the completion of programming w_1, the page buffer for w_0 is released. Compared to the LSB backup scheme, this approach does not amplify the page write nor requires reserved resources. However, as the page buffer is located in the volatile memory, this approach can only address the paired page interference originated from the programming failure, not from the sudden power-off. In addition, the approach requires as many page buffers as the number of LSB pages whose paired pages are not programmed.

Kwon et al. [15] propose to use *super capacitors* to endure sudden power-off in SSDs. The paired page interference originated from the sudden power-off can be avoided by utilizing the power from the super capacitors to finish the ongoing MSB page programming. However, this scheme cannot deal with the paired page interference originated from the programming failure and requires additional hardware components.

Lee et al. [17] attempt to skip backing up LSB pages by exploiting the architectures for NAND flash memory systems. NAND flash memory systems typically have a write buffer that temporarily keeps the data for write requests. They utilize the write buffer as the copy of LSB pages while processing write requests. When a pair of LSB and MSB pages is assigned to the same write request, the LSB page is not backed up prior to programming the MSB page. Instead, if the MSB page programming is halted by a programming failure, the original data of the LSB page is recovered from the write buffer. In case of a sudden power-off, the LSB page of the pair can be corrupted by the interference. However, modern file systems are capable of recovering partially written requests using journaling techniques [28], which effectively identify and recover the corrupted LSB page. Hence, FTL can deal with the paired page interference in a per-request manner rather than a per-page manner. In addition, they propose a so-called parity page prebackup scheme, which backs up one XOR-ed LSB page for consecutive LSB pages rather than copying each of them. When one of the LSB pages is corrupted, the original data can be recovered by the XOR backup with remaining good pages. However, the parity page prebackup scheme is only effective on the particular system architecture and requires a specific paired page layout, which is uncommon for commercial NAND flash memory products [30, 31].

4 GCMIX

4.1 Motivation and Key Idea

As we reviewed in the previous section, many schemes have been suggested to deal with the paired page interference, which is inherent in the MLC NAND flash memory. However, the previous schemes require extra flash write and/or additional hardware components, reducing the lifetime of MLC flash memory and increasing the manufacturing cost.

We present *GCMix*, a software approach to deal with the paired page interference that requires neither an extra flash write nor any additional hardware. As we discussed in Section 2.2, FTLs reclaim invalidated pages through the GC process, which migrates valid pages in a victim block to other block(s). The key idea behind GCMix is to utilize the valid pages in the victim block as the backup against the paired page interference. When a victim block is picked up, valid pages in the victim block are copied to the LSB pages of MLC data blocks while their paired MSB pages are utilized to store incoming write requests. The erase of the victim block is postponed until all the paired MSB pages are written with the data issued by the host. In this case, the paired page interference can only happen while programming the MSB page with the host data write. However, its paired LSB page contains the copy of a valid page in the victim block, and thus, the contents of the corrupted LSB page can be recovered from the original page in the victim block. If GC is not in progress or there is no valid page to be paired with the incoming host write page, FTL can fall back to the other scheme, such as the LSB backup scheme.

Figure 3(d) depicts the proposed GCMix scheme. When GC picks up a victim block, its valid pages v_0, v_1, and v_2 are copied to the LSB pages of an MLC data block. Meanwhile, their MSB pages are filled with the data written from the host, i.e., w_0, w_1, and w_2. If programming w_2 is interrupted, its paired page can be recovered from the valid page v_2 in the victim block. After writing w_2, the victim block can be erased and GC is continued with the next victim block. In this way, GCMix can deal with the paired page interference without any extra page copy or additional hardware.

4.2 GCMix on Page Mapping FTL

We applied the idea of GCMix on the page mapping FTL and implemented a prototype FTL, named FTL-PM, on an FTL simulator (see Section 5.1 for details of the simulator). FTL-PM is based on the vanilla page mapping FTL. It picks up a free block as an *update block*, to which incoming host writes are written sequentially. The write to the update block also changes the corresponding mapping information. When the current update block becomes full, FTL-PM obtains another free block and designates it as a new update block.

When there are enough free blocks, GC does not have to be performed. Therefore, FTL-PM initially utilizes the LSB backup scheme by maintaining one LSB backup block internally. Before writing any data in the MSB page of the update block, its paired LSB page is copied into an LSB page of the LSB backup block. When the number of free blocks gets equal to or below the low watermark F_{low}, FTL-PM begins to apply the GCMix scheme. It picks up a victim block based on the cost-benefit policy, and copies the valid pages to the LSB pages in the update block. Their paired MSB pages are written with the incoming data from the host. When all the valid pages are paired with the host write data, the victim block is erased and turned into a free block.

As the valid pages are migrated only to the LSB pages, it requires up to two free blocks to reclaim one victim block. Thus, if the number of valid pages in a victim block exceeds a half of the number of pages in a block, the number of free blocks will decrease by one. To prevent free blocks from being completely exhausted, FTL-PM performs GC synchronously when the number of free blocks gets equal to or lower than another watermark F_{min} such that $F_{min} < F_{low}$. This watermark ensures FTL to keep the minimum number of free blocks required to operate the FTL correctly. In this case, the valid pages in the current victim block are copied to the update block sequentially, utilizing both MSB pages and LSB pages within the update block. Note that writing to any MSB page during the synchronous GC is also vulnerable to the paired page interference. Normally, this does not cause any problem since the corrupted data, if any, can be recovered from the original data in the victim block. However, when the first valid page is going to be written to the MSB page in the update block, the existing data in the corresponding LSB page can be lost in the event of programming failure. To cope with this situation, such an LSB page should be temporarily backed up in the LSB backup block.

If less than a half of the victim block are filled with valid pages, reclaiming the victim block can increase the number of free blocks by one. When the number of free blocks gets equal to or exceeds the high watermark F_{high} such that $F_{low} < F_{high}$, GCMix is suspended to prevent excessive GC. The suspended GCMix is resumed when the number of free blocks hits the low watermark F_{low} again. FTL-PM employs the LSB backup scheme during the suspension of GCMix.

4.3 GCMix on DAC

As mentioned in Section 2.2, separating hot data from cold data increases the efficiency of GC by reducing the number of valid pages within a victim block. While applying GCMix to the page mapping FTL, we encounter a shortcoming of GCMix regarding this hot/cold data separation policy. The problem is that the incoming host write pages and the valid pages in victim blocks have different hotness. The host write pages are considered relatively hot because they are likely to be overwritten soon due to the locality of reference. In that sense, the valid pages in a victim block are cold as the pages are programmed in the past but they are not overwritten or discarded until

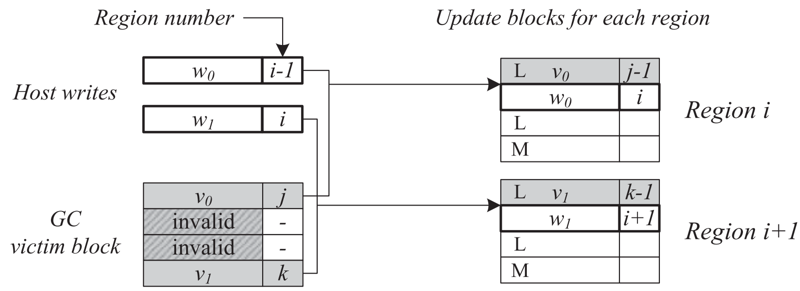


Fig. 4. Pairing host writes with valid pages in victim blocks in FTL-DM.

the block is chosen during GC. GCMix pairs these hot host write pages with cold valid pages and stores them in the same block, which goes against the spirit of the hot/cold data separation policy.

Ideally, a solution to this problem is to classify both host write pages and valid pages according to their hotness and then mix them together in the same block that has a similar level of hotness. Hence, to solve this problem, we apply GCMix to DAC. Compared to the pure page mapping FTL, using GCMix with DAC can improve the overall performance since DAC utilizes the hot/cold data separation policy via estimating the relative hotness of the data and clustering them into different regions.

A naive implementation of GCMix on DAC is to pair the write request to the data currently stored in the region $i - 1$ with the valid page in the region $i + 1$, and put them together in the update block of the region i. However, this strategy has a couple of issues. Firstly, there is no guarantee that we can pick up any victim block in the region $i + 1$. Especially, if there is a skew in the write requests to a certain set of regions, the chance becomes higher that we are not able to find a victim block to be cleaned. Secondly, even if there is a victim block in the region $i + 1$, it is only the best choice in the region $i + 1$, not a globally optimal one. This suggests that integrating GCMix with DAC presents unique challenges yet to be explored.

We have implemented a modified version of DAC called FTL-DM, which integrates GCMix into DAC. FTL-DM globally selects a victim block as in the original DAC and uses its valid pages to fill out the LSB pages of the update block. The main difference between FTL-DM and DAC lies in the way the region number is managed. In the original DAC, all the pages within a block belong to the same region, hence, the region number is maintained in association with each block. FTL-DM, however, tries to maximize the opportunity to combine valid pages in the victim block with the incoming data by allowing pages that belong to different regions to be stored in the same block.

Figure 4 outlines the case for handling a write request in FTL-DM. FTL-DM keeps an update block for each region and picks up a victim block globally via the cost-benefit policy. When existing data in the region $i - 1$ is overwritten by new data w_0 from the host, FTL-DM pairs it with the valid page v_0 in the victim block even though the region number j of the valid page does not match up with the region number $i + 1$ (i.e., $j \neq i + 1$) required for the incoming write. Then, the paired pages are written to the update block for the region i, while the region numbers of LSB (data copied from the valid page) and MSB (data written by the host) pages are set to $j - 1$ and i, respectively. In the same way, a host write w_1 for region i is paired with the valid page v_1 in the victim block, and is written to the update block for region $i + 1$. Note that when the page copied from the victim block is overwritten later by the host, the new data is written to the update block of the region j, not of the region $i + 1$.

Since all the pages in an update block might not belong to the same region anymore, we make FTL-DM explicitly track the region number for each page. The per-page region tracking requires additional space; however, the amount of the space is not significant. For instance, if FTL-DM manages 8KB pages and employs four regions, each page requires two bits for tracking its region. This takes 32KB per 1GB, which only accounts for 0.003% of the capacity. Moreover, the region number does not have to be kept in memory but can be offloaded to the out-of-band (OOB) area of flash pages since it is only required when pages are being promoted or demoted. Thus, we believe the space overhead for the per-page region tracking is acceptable in practice.

Same as FTL-PM, FTL-DM starts to apply GCMix when the number of free blocks gets equal to or below the watermark F_{low}. GCMix is suspended once the number of free blocks hits the watermark F_{high}. It is noteworthy that FTL-DM copes with the paired page interference using LSB backup when GCMix is not applied.

4.4 Applying GCMix Adaptively

We have implemented FTL-DM on an FTL simulator and evaluated its performance with synthetic workloads, which have various degrees of spatial locality in reference. The preliminary evaluation result indicates that FTL-DM outperforms other FTLs in most cases including DAC with LSB backup, since GCMix has been successful in eliminating redundant page copies compared to the LSB backup scheme. However, the result also shows that hot/cold data separation policy is more important than eliminating the redundant page copies when the degree of spatial locality is very high. In other words, when only a small number of logical pages are heavily written, the benefit of the hot/cold data separation policy outdoes the gain obtained by GCMix, thereby making DAC with LSB backup outperform FTL-DM (refer to Section 5.3 for details). This result suggests that if the current workload exhibits a very high degree of spatial locality in reference, it is more desirable to deactivate the use of GCMix.

In order to apply GCMix adaptively, we need to quantify the degree of spatial locality in the workload. For this end, we capture the distribution of page (write) references among the regions and estimate the degree of locality from their distribution. We maintain N regions similar to DAC and monitor the migration of pages among the regions. For a region R_n, we define the *migration ratio* of the region α_n as follows:

$$\alpha_n = \frac{P_n / \sum_{i=1}^{N} P_i}{V_n / \sum_{i=1}^{N} V_i}, \tag{1}$$

where P_i and V_i denote the number of pages promoted from R_i to R_{i+1} and the number of valid pages remaining in the region R_i, respectively. Note that when a page is promoted from R_i to R_{i+1}, V_i is decreased by 1 whereas V_{i+1} is increased by 1 to take account for the promotion of the page. If the workload exhibits no locality, every page has the same probability to be overwritten and the ratio of the promoted pages for the region n to the total number of promoted pages ($P_n / \sum_{i=1}^{N} P_i$) will be proportional to the ratio of the valid pages in the region n to the total number of valid pages ($V_n / \sum_{i=1}^{N} V_i$). Thus, α_n for every region, n will have a similar value to each other around 1.0.

If the workload has high locality, the majority of page writes will be directed to a small number of hot pages, which belong to higher regions. For higher regions, this will make the number of promoted pages become larger than the average while the number of valid pages smaller than the average. Therefore, the values of α_n for higher regions will be much larger than 1.0. On the other hand, the majority of pages is cold and belongs to lower regions. In those lower regions, the values of α_n will be small, close to zero, as only a small number of pages are migrated to upper regions. To summarize, we can see that the value of α_n will have a large variance across regions when the current workload exhibits high locality.

ALGORITHM 1: Determine when to use the GCMix scheme

```
Data: mode: current mode ("LSB Backup" or "GCMix")
Data: n: number of free blocks
Result: "LSB Backup" or "GCMix"
ω ← calculate ω
if ω ≥ τ then
    mode = "LSB Backup"
else
    if mode == "LSB Backup" then
        if n ≤ F_low then
            mode = "GCMix";
        end
    else
        if n ≤ F_min then
            Perform GC synchronously
        end
        if n ≥ F_high then
            mode = "LSB Backup"
        end
    end
end
return mode
```

Based on this observation, we estimate the *workload locality* ω using the variance in the migration ratio of each region as follows:

$$\bar{\alpha} = \sum_{i=1}^{N} \frac{\alpha_i}{N} \tag{2}$$

$$\omega = Var(\alpha_1, \alpha_2, \ldots, \alpha_N) \tag{3}$$

$$= \sum_{i=1}^{N} \frac{(\alpha_i - \bar{\alpha})^2}{N} = \sum_{i=1}^{N} \frac{\alpha_i^2}{N} - \bar{\alpha}^2 \tag{4}$$

ω quantifies the locality in the current workload to a range of $[0, \infty)$, with the lower value of ω indicating the lower locality in the workload. ω can be calculated at a small overhead, as both time and space complexity for obtaining ω are $O(N)$.

We have developed a new FTL called FTL-DML by modifying FTL-DM to consider the degree of locality in the workload. Essentially, FTL-DML operates the same as the original FTL-DM except it calculates the workload locality ω periodically. If ω is less than a threshold τ and the free block constraint of the original FTL-DM is satisfied at the same time, FTL-DML enables the GCMix scheme. Otherwise, FTL-DML falls back to the LSB backup scheme to avoid the adverse effect of GCMix. Algorithm 1 gives a pseudo-code of FTL-DML for determining the scheme to use as a countermeasure for the paired page interference.

5 EVALUATION

5.1 Evaluation Methodology

Simulator. To evaluate the proposed scheme, we have developed an event-driven FTL simulator called FTLSim. FTLSim accepts disk I/O requests comprised of the type of operation (i.e., read or write), the start logical address to operate on, and the length of the operation. FTLSim replays the given I/O requests on a virtual SSD in which various components, including FTL, NAND

Table 1. Specification of the 35nm 2-bit MLC NAND Flash Memory Chip and the Storage

Specification		Value
Unit size	Page	8KB
	Block	1MB (128 pages)
Latency	LSB page read	80 μs
	MSB page read	120μs
	LSB page program	500μs
	MSB page program	1,500μs
	Block erase	1,500μs
Guaranteed P/E cycle		3,000 cycles
Storage configuration	Blocks per chip	4,096 blocks
	# of chips	4
	# of planes	2
	Total size	32GB

controller, and NAND flash chips, can be configured. FTLSim provides an abstract programming interface (API) so that various FTLs can be implemented as modules, and then plugged into FTLSim. FTLSim also has a UI interface by which users can set a number of tunable parameters related to FTLs and NAND flash chips. We validated FTLSim by the following ways; Firstly, we cross-checked the integrity of results using the traces on which the expected behaviors of FTL are known. Secondly, we checked invariants for each component of the simulator (e.g., FTL, NAND flash chips, host interface layer) whenever the part is accessed. Lastly, we verified that the checkpoints for the invariants actually detect malicious operations that we injected in the implementation on purpose. We have simulated 35nm MLC NAND flash memory chips, which are widely used for commercial products [31]. Table 1 describes the specification of the NAND flash chip we used throughout our evaluation.

FTLs. In this article, we consider five FTLs by combining various mapping schemes and counter-interference schemes. All FTLs are configured to perform GC synchronously when the number of free blocks becomes the minimum to run FTLs correctly. The idea behind this approach is to apply the same rule to all FTLs so that the performance comparison between FTLs is fair regardless of their different threshold constraints due to their mechanisms.

The evaluated FTLs can be classified into two groups: the locality-ignorant FTLs and the locality-aware FTLs. FTL-P and FTL-PM are the locality-ignorant FTLs that do not differentiate between hot data and cold data. These FTLs are configured to perform GC synchronously when the number of free blocks becomes one (i.e., $F_{min} = 1$). FTL-P is the baseline FTL that uses the vanilla page mapping with the LSB backup scheme. In order to backup LSB pages, another block is designated as the LSB backup block. FTL-PM represents the page mapping FTL with GCMix that we explained in Section 4.2.

The locality-aware FTLs include the variations of DAC FTL, which separate hot data from cold data and treat them differently. The FTLs belonging to this group are configured to use four regions. If necessary, one block is designated as the LSB backup block. FTL-D represents the vanilla DAC FTL with the LSB backup scheme. It performs synchronous GC when the number of free blocks becomes four since at least one free block is required for each region (i.e., $F_{min} = 4$).

Table 2. A List of Evaluated FTLs

Category	FTL	Description
Locality-ignorant	FTL-PX	Page mapping FTL without LSB backup
	FTL-P	Page mapping FTL with LSB backup
	FTL-PM	Page mapping FTL with GCMix
Locality-aware	FTL-DX	DAC FTL without LSB backup
	FTL-D	DAC FTL with LSB backup
	FTL-DM	DAC FTL with GCMix
	FTL-DML	DAC FTL with adaptive GCMix

FTL-DM integrates the GCMix scheme with DAC FTL. Finally, FTL-DML is a variant of FTL-DM which adaptively applies the GCMix scheme depending on the locality in workloads. FTL-DML evaluates ω on every second and uses $\tau = 10$ as the threshold for the workload locality. Refer to Section 5.3 for the details on how we obtained the particular value for τ. In practice, FTL-DM is evaluated using FTL-DML by setting $\tau = \infty$.

For the watermark to kick in GCMix, we used $F_{low} = F_{min} + 1$. Thus, F_{low} is 2 for FTL-PM, and 5 for FTL-DM and FTL-DML. The rationale behind these particular values are to delay GC as much as possible but GCMix is activated before the synchronous GC is triggered. We set $F_{high} = 10$ for FTL-DM and FTL-DML so that GCMix is suspended when the number of free blocks is doubled (i.e., $F_{high} = F_{low} \times 2$). For fair comparison, we set the same F_{high} value for FTL-PM.

To estimate the cost to counter the paired page interference, we implemented two FTLs, FTL-PX and FTL-DX, each of which is the page mapping FTL and DAC FTL without backing up LSB pages, respectively. Table 2 summarizes a list of FTLs evaluated in this article. All of them are configured to utilize 25% of the total flash space as over-provisioned area and use the cost-benefit policy to pick up the victim block during GC. We omit the results from other counter-interference techniques such as SLC-mode buffering as they show the similar results to those of the LSB backup scheme.

Workloads. In order to evaluate under realistic environments, we use six block I/O traces of the Storage Networking Industry Association (SNIA) Input/Output Traces, Tools, and Analysis Repository [10, 33]. These traces are collected from several servers at Microsoft. Three of them (Befs, Buildserver, and Exchange) are collected from Microsoft's production servers, and three of them (Tpcc2007, Tpcc2008, and Tpce) are from their enterprise servers running online transaction processing (OLTP) workloads. We use these traces over others since they provide enough write requests, causing a meaningful number of GC. Table 3 summarizes the fundamental characteristics of the traces, measured in 8KB request granularity.

5.2 Validating the Workload Locality Metric

First, we validate the proposed workload locality metric ω. We measure ω from FTL-DM at every second while running the workload that generates write requests in the Zipf distribution [38]. Each request is 8KB in size, aligned to the 8KB boundary. The requests spread over the entire user-visible address space which is 24 GB in size and the amount of data written is 128GB in total.

Figure 5 depicts the cumulative distribution of ω on various values of the exponent coefficient z of the Zipf distribution. Note that the x-axis is in a log scale and the larger value of the coefficient z means the higher locality in the write references. For a given z, ω resides in a narrow range of values. As the locality in the workloads increases, the range shifts to higher values. For instance, the 50^{th} percentile of the distributions for $z = 0$, 0.2, 0.4, 0.6, 0.8, and 1.0 are 0.031, 0.037, 0.205,

Table 3. Traces for Evaluation and Their Characteristics

Workload	RW	Amount of I/O (MB)	Accessed area (MB)	Accesses per page	RW ratio (R/W)
Befs	R	218,382	48,930	4.46	1.72
	W	126,922	33,042	3.84	
Buildserver	R	178,366	75,794	2.21	0.92
	W	193,250	129,031	1.49	
Exchange	R	511,936	152,337	3.36	0.71
	W	725,898	66,810	10.87	
Tpcc2007	R	1,428,669	93,987	15.20	1.82
	W	785,372	127,844	6.14	
Tpcc2008	R	1,668,556	72,771	22.92	1.40
	W	1,195,560	180,509	6.62	
Tpce	R	1,591,175	156,459	10.16	6.71
	W	237,157	87,983	2.69	

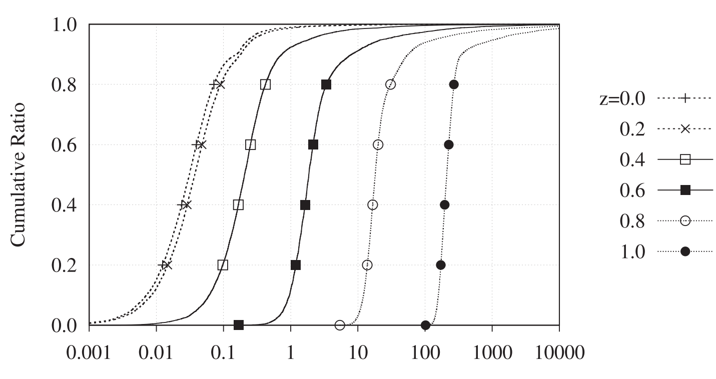


Fig. 5. Distribution of the workload locality metric ω on various Zipf exponent coefficients.

1.861, 18.122, and 210.391, respectively. This result implies that the value of ω is influenced by the degree of locality in workloads.

To verify that ω can practically quantify the locality, we analyze the locality of real workloads and compare it with the ω of the workloads. We obtain the locality of workloads as follows; we pad each write request to be aligned to 8KB boundary and count the number of writes to each 8KB page. Then, we rank the pages with the number of writes to them, and summarize it by cumulating the number of writes from the highest rank page to the lowest rank page. Figure 6 depicts the cumulative fraction of writes for each workload. Note that a linear relationship between the fraction of pages and the fraction of writes means every page contributes the same number of writes to the total writes, which indicates no locality.

We can observe that Exchange has the highest locality; 10% of pages receive 69.6% of writes, and 80% of writes go to 18.1% of pages. Befs shows the second highest locality among the workloads; 80% of writes go to 46.2% of pages. The rest of the workloads show less locality than these two workloads. For example, 60% of writes go to 40% of pages in Buildserver.

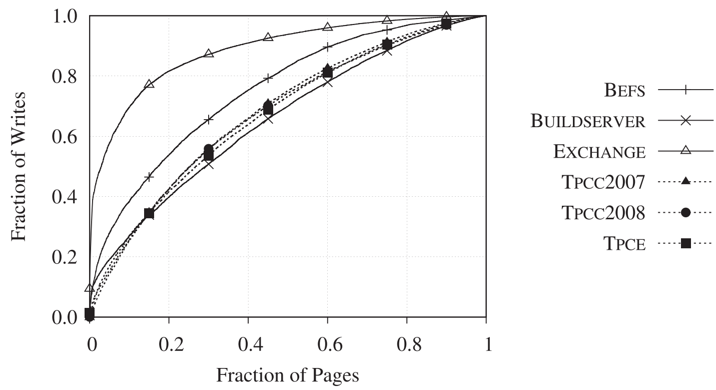


Fig. 6. Distribution of writes to pages on real workloads.

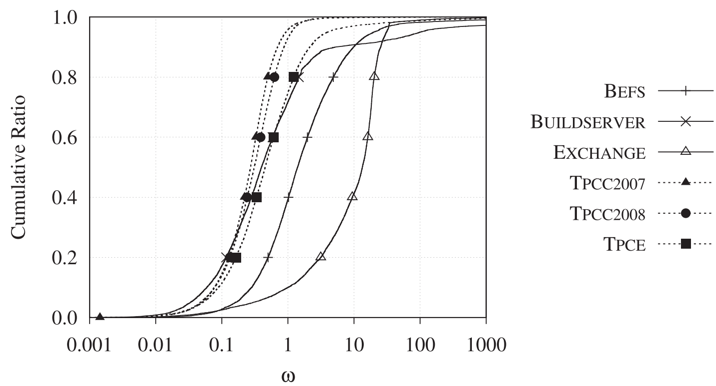


Fig. 7. Distribution of locality metric ω on real workloads.

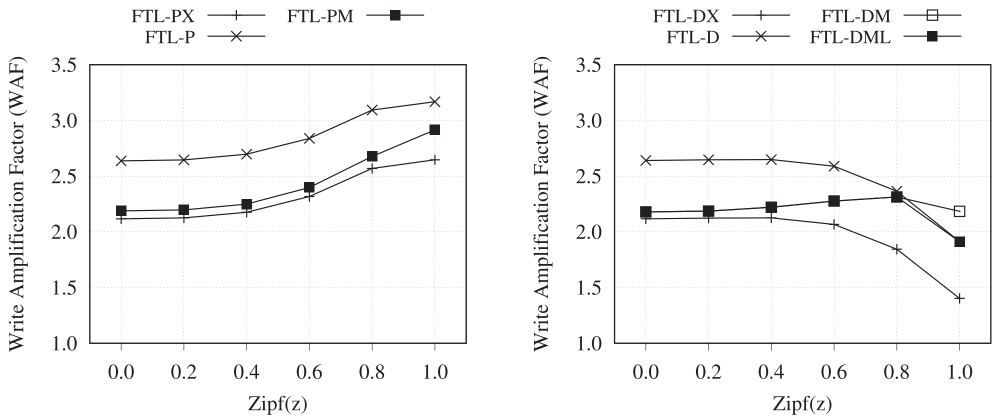
We compare the result with the the cumulative distribution of ω on the workloads illustrated in Figure 7. We can see that Exchange has a large value of ω, which is followed by Befs. The others show a relatively smaller value of ω. This overall trend coincides with the locality of workloads shown in Figure 6. From these results, we can conclude ω is a reliable metric for estimating the degree of locality in the workload.

5.3 GCMix on Synthetic Workloads

To analyze the effectiveness of GCMix, we measure the WAF under the Zipf workloads. We vary the exponent coefficient z from 0 to 1.0. Figure 8 compares the WAFs obtained from the evaluated FTLs.

Firstly, we analyze the cost to counter the paired page interface. The results from ftl-px and ftl-dx illustrate the cost in ftl-p and ftl-d, respectively. Without backing up LSB pages, the WAF of ftl-px is smaller than that of ftl-p by up to 19.7%, and the WAF of ftl-dx does by up to 26.7% compared to that of ftl-d. This indicates that countering the paired page interference incurs high write amplification and considerably shortens the lifespan of NAND flash memory storage.

For the locality-ignorant FTLs shown in Figure 8(a), the WAF grows fast as the locality in the workload increases. Regardless of the degree of locality, ftl-pm always outperforms ftl-p; the WAF of ftl-pm is lower than that of ftl-p by up to 17.0% on low locality and by up to 8.09% on



(a) Locality-ignorant FTLs (b) Locality-aware FTLs

Fig. 8. Comparison of WAF on synthetic Zipf workloads.

high locality. We can also verify that the WAF of FTL-PM is only slightly higher than that of FTL-PX when the locality is low. This implies GCMix effectively eliminates the most of the overhead to counter the paired page interference on low-locality workloads.

The locality-aware FTLs such as FTL-DX, FTL-D, FTL-DM, and FTL-DML exhibit different trends as shown in Figure 8(b). When the locality is low, FTL-DX, FTL-D, and FTL-DM show the similar WAFs to FTL-PX, FTL-P, and FTL-PM, respectively. In this case, FTL-DM shows about 17.5% lower WAF than FTL-D. As the locality increases, the WAF of FTL-D decreases quickly whereas that of FTL-DM does not, and eventually the WAF of FTL-D drops lower than that of FTL-DM. The reason for this phenomenon is due to the negative influence of GCMix on the hot/cold data separation policy. On the high-locality workload, FTL-D benefits much from the hot/cold data separation. FTL-DM, however, cannot benefit from the hot/cold data separation as much as FTL-D, since GCMix puts hot and cold data together in the same block. As a result, FTL-DM performs worse than FTL-D when the locality is high. In addition, the WAF of FTL-DM cannot catch up with the WAF of FTL-DX when the locality is high.

The only cure for this symptom is to disable GCMix when the locality in the current workload is extremely high. Figure 8(b) indicates that the benefit of GCMix is almost offset when $z = 0.8$, and it is completely dominated by the benefit of the hot/cold data separation when $z = 1.0$. This implies that the proper candidate for the threshold τ for FTL-DML will be the value of ω when z is between 0.8 and 1.0. Given the observation, Figure 5 suggests τ=10 as the rationale value for the threshold.

The result of FTL-DML in Figure 8 shows the WAF when τ is set to 10. It confirms that FTL-DML outperforms other FTLs by applying GCMix adaptively based on the locality in the current workload. When the locality is low, FTL-DML utilizes GCMix and operates the same as FTL-DM. When the locality becomes high, FTL-DML detects the high degree of locality by means of ω and switches to the LSB backup scheme so that it exhibits the similar WAF to that of FTL-D.

5.4 Write Amplification Analysis on Real Workloads

To assess the implication of GCMix for the reliability and lifetime of flash-based storage devices, this section focuses on analyzing the write amplification of the FTLs on real environments. Figure 9 compares the WAFs of real workloads measured under various FTLs.

First of all, we can estimate the cost to deal with the paired page interference in real environments by comparing FTL-PX with FTL-P and FTL-DX with FTL-D. Specifically, the WAF increased

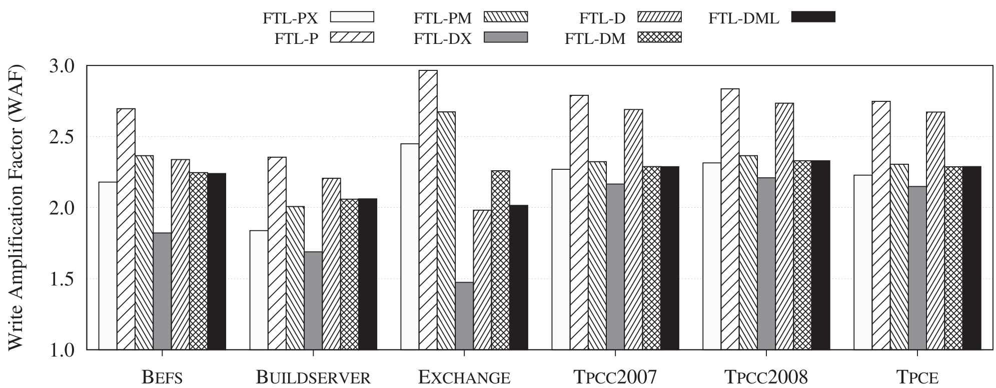


Fig. 9. Comparison of WAF of real server workloads.

by backing up LSB pages accounts for 21.1% to 28.1% of the WAF for FTL-P and 23.7% to 34.5% for FTL-D. This result confirms countering the paired page interference considerably amplifies writes in practice, thereby leaving a chance to improve.

As shown in Figure 9, FTL-D outperforms FTL-P on all workloads to a great or less extent, as DAC always benefits from the hot/cold data separation policy. Especially, on the Exchange workload, which exhibits the highest degree of locality, FTL-D shows 33.2% lower WAF than FTL-P.

For the workloads with low to medium locality (except for Exchange), adopting GCMix always improves the overall WAF and the improvement outweighs the benefit from separating hot data from cold data. Thus, FTL-PM and FTL-DM offer lower WAFs than FTL-P and FTL-D, respectively, by up to 16.7% on Tpcc2007. As the value of ω mostly stays below τ in these workloads, FTL-DML resorts to the GCMix scheme and shows the WAFs comparable to those of FTL-PM.

On the high-locality workload (Exchange), the WAF of FTL-D is reduced significantly compared to those on the low-locality workloads. Given the high degree of locality, DAC reduces the WAF by exploiting the hot/cold data separation policy. This reduction outbalances the benefit of GCMix so that FTL-D outperforms FTL-DM on these workloads. However, FTL-DML detects the presence of high locality in the current workload and switches to the LSB backup scheme so that it avoids the penalty caused by the use of GCMix. In this case, FTL-DML shows the WAFs comparable to those of FTL-D.

In order to further identify the origin of the improvement in the WAF, we examine how many times blocks are erased and what purpose the blocks have been used for. Figure 10 presents the breakdown of the block erase counts for each FTL with the real workloads. Note that the erase counts are normalized to the total erase count in FTL-P on each workload. The dark boxes in Figure 10 represent the fraction of the erased blocks that have stored user data. The light box on top of each bar illustrates the fraction of the erased blocks used for LSB backup.

When the LSB backup scheme is used, each host page write to an MSB page is preceded by a page write to backup the paired LSB page. Thus, the erase count of the LSB backup block is increased in proportion to the total amount of written data. This explains why a significant amount of LSB backup blocks (29.8% to 41.0% of the total blocks erased) are erased in FTL-P and FTL-D, which rely on the LSB backup scheme.

As shown for FTL-PM and FTL-DM, GCMix almost eliminates the erase operation for the LSB backup block for all the workloads. However, GCMix disrupts the hot/cold data separation, reducing the efficiency of GC. This results in the increase of the erase count for normal data blocks in those FTLs compared to their baseline FTLs (FTL-P and FTL-D). Especially, the increase is significant

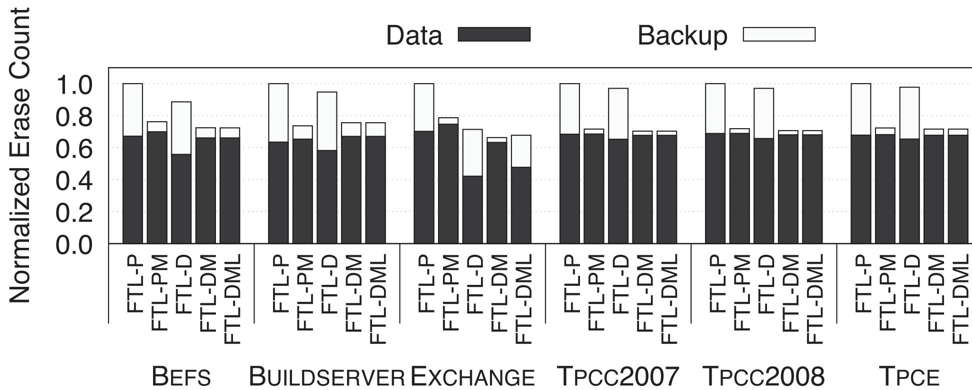


Fig. 10. Breakdown of block erase counts on real workloads.

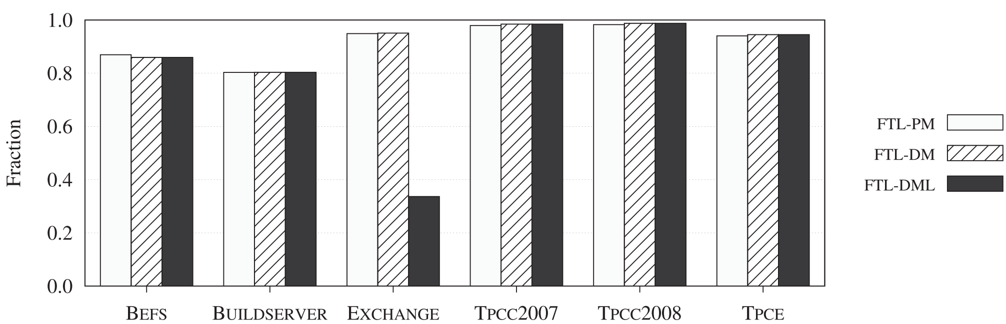


Fig. 11. Fraction of host write pages paired with valid pages by GCMix.

when the locality is as high as Exchange. In these cases, the WAF of FTL-DM is significantly increased due to the reduced GC efficiency as shown in Figure 9. On the other hand, FTL-DML switches to FTL-D for this case to take the benefit of the hot/cold data separation policy. Accordingly, we can observe that FTL-DML works similar to FTL-D for the high-locality workload, while it works similar to FTL-PM otherwise.

To understand how active GCMix is in practice, we analyzed the fraction of host writes that are paired with valid pages by GCMix. Figure 11 summarizes the fractions of host write pages paired by GCMix while running the workloads. Note that the host writes that are not paired by GCMix are written to corresponding update blocks first, and then protected by LSB backups later if necessary.

We can observe that the most host writes are paired by GCMix on FTL-PM and FTL-DM regardless of the workloads. Specifically, FTL-PM pairs 80.3% to 98.3% of total host write pages via GCMix, and FTL-DM does 80.3% to 98.7%. These results indicate that GCMix is activated for most of the running time, actively eliminating the most of LSB backups. FTL-DML runs in the similar way to FTL-DM for low-locality workloads, pairing approximately 80.4% to 98.7% of host writes via GCMix. However, GCMix is deactivated for most of the running time in Exchange workload due to the high locality of the workload, thereby pairing only 33.6% of host writes. This shows the adaptive characteristic of FTL-DML that selectively applies GCMix according to the workload locality.

To sum up, we can confirm that the use of adaptive GCMix (as in FTL-DML) reduces the WAF by efficiently eliminating writes for backing up LSB pages so that the lifetime of MLC NAND flash memory can be extended.

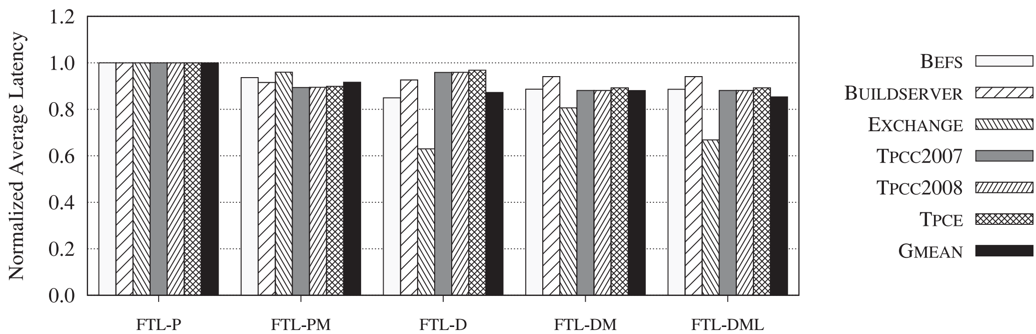


Fig. 12. Comparison of the time to process write requests in workloads.

5.5 I/O Performance Analysis on Real Workloads

Different placement of pages for a given workload results in different I/O performance, and GCMix changes the placement of their baseline FTL. To understand the effect of GCMix on the I/O performance, we measured the time to process each host request on the FTLs. The FTL implementation translates each host request to equivalent flash operations, and each flash operation increases a clock in the simulator according to the timing specification listed in Table 1. Then, the time to process a request is obtained by subtracting the time when processing the request is started from the time when the processing is finished. If FTL needs to perform GC to handle a write request, the time to reclaim victim block(s) is included in the process time of the write request. Figure 12 summarizes the total time to process write requests in the workloads with the FTLs. The time is normalized to the time of FTL-P on each workload, and the smaller value indicates the better performance. GMEAN indicates the geometric mean of the normalized times for the FTL.

The result of FTL-PM indicates GCMix improves the write performance on the locality-ignorant FTL. Compared to FTL-P, the time of FTL-PM is reduced by up to 10.5% and 8.4% in average, respectively. The reason for the improvement is two-fold. First, GCMix eliminates the most of the LSB page backups as shown in Figure 10. Thus, FTL-PM can usually skip backing up LSB pages while processing write requests. Second, the write performance of flash-based storage devices is determined by not only the raw flash write performance but also the GC performance. GC involves reads and writes of one or more flash pages and an erase of a block, which is slower than the raw write performance. Thus, the overall performance of the device is eventually saturated to the GC performance. In FTL-P, valid pages are copied to both LSB and MSB pages. Contrarily, valid pages are migrated to LSB pages in FTL-PM, which effectively improves the throughput of GC. In addition, the accelerated GC can compensate the increased time to write host requests to slow MSB pages. Thus, applying GCMix can improve the write performance.

The effect of GCMix on the write performance is different on the locality-aware FTLs. FTL-DM has smaller write latency than FTL-D for low-locality workloads (i.e., TPCC2007, TPCC2008, and TPCE) whereas the latency is larger for the workloads having medium to high locality (BEFS, BUILDSERVER, and EXCHANGE). As a result, FTL-DM has 0.9% longer write time than FTL-D in average. This is due to the adverse effect of GCMix on the hot/cold data separation, which reduces the efficiency of GC. However, FTL-DML avoids applying GCMix on such workloads, and accordingly, FTL-DML exhibits 2.71% shorter write latency than FTL-D in average.

Figure 13 summarizes the read performance, which is measured and presented in the same way to the write performance. The time to process read requests of FTL-PM and FTL-DM are slightly increased from or close to the time of their baseline FTLs without GCMix (i.e., FTL-P and FTL-D).

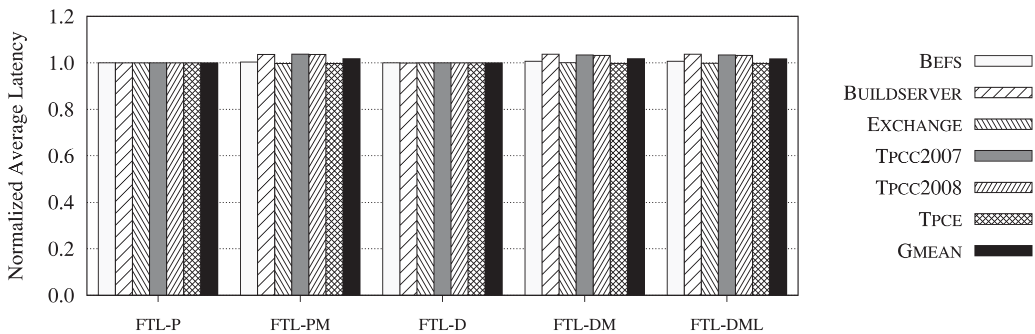
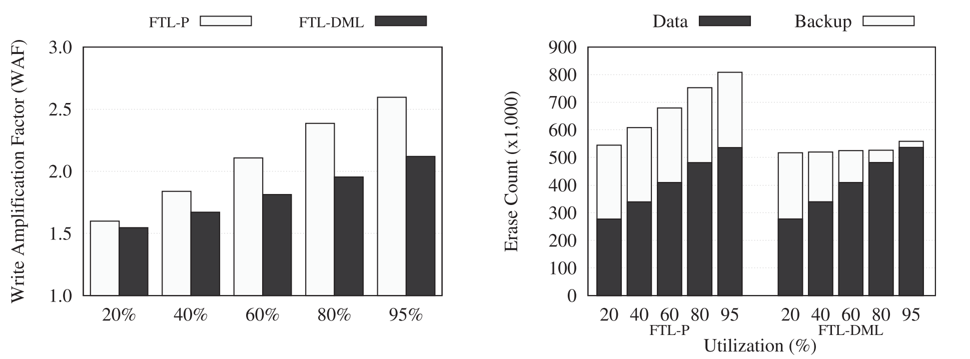


Fig. 13. Comparison of the time to process read requests in workloads.



(a) Comparison of WAF on various storage utilization (b) Breakdown of erase count on various storage utilization

Fig. 14. Performance on various storage utilization.

GCMix can increase the time to process read since GCMix places hot host writes to slow MSB pages. However, the negative effect of the placement is not significant in practice and increases the read time by up to 3.6%. On average, GCMix increases the read time by 1.8%, 1.8%, and 1.7% for FTL-PM, FTL-DM, and FTL-DML, respectively. Thus, we can conclude that the adverse effect of GCMix on read performance is insignificant.

5.6 Effect of Storage Utilization

Now, we analyze the influence of storage utilization on the performance of GCMix. We measure the WAF and erase counts under the uniform random access pattern, which writes 8KB requests in total of 128GB on the given fraction of 32GB address space. Figure 14(a) and (b) show the resulting WAF and the breakdown of erase counts on various storage utilization. We only show the result from FTL-P and FTL-DML for brevity. FTL-D behaves in a similar way to FTL-P, and FTL-PM and FTL-DM behave similarly to FTL-DML under the condition.

From Figure 14(a), we can see that the WAF is increased linearly proportional to the storage utilization in FTL-P. Figure 14(b) reveals that the increase in the WAF is originated from the increase of the data block erase. Each run of the evaluation writes the same amount of data, and the number of LSB backup block erase is proportional to the total amount of written data. Thus, the LSB backup block erase count remains constant. However, the average number of valid pages in a victim block is proportional to the storage utilization; more data blocks are erased on higher storage utilization.

FTL-DML exhibits a different trend. When the storage utilization is low, only a few pages are valid in a victim block so that most of the host writes cannot be paired with valid pages, falling back to the LSB backup scheme. Thus, the WAF and the breakdown of block erase counts follows the results of FTL-P. When the storage utilization becomes large enough, most of the host writes can be paired with valid pages in victim blocks. In this case, we can observe that most of LSB backup operations can be avoided and the overall erase count is decreased.

6 CONCLUSION

As MLC flash memory programs cells in multiple stages, data can be corrupted in the middle of programming operations. This phenomenon called paired page interference is inherent in MLC NAND flash memory and must be dealt with to guarantee reliability and durability.

In this work, we propose GCMix, a novel software technique to ensure data protection against the paired page interference without extra hardware or additional page writes. The key idea behind GCMix is to pair the valid page in the GC victim block with the incoming page written by the host, and then write them to the LSB and MSB page, respectively.

Although it seemingly looks easy to do, GCMix can sometimes harm the overall efficiency as it disrupts the hot/cold data separation, especially when the locality in the workload is high. We address this challenge by proposing FTL-DML, which integrates GCMix to DAC FTL and enables GCMix adaptively. In order to apply GCMix adaptively only when it provides any performance gain, we also develop a metric that can quantify the locality in the workload on-the-fly. Evaluations with both synthetic and real workloads confirm that the proposed scheme can protect data against the paired page interference with less write amplification than the traditional LSB backup scheme. We leave implementing and evaluating the proposed scheme on a real device as future work.

REFERENCES

[1] Amir Ban. 1995. Flash file system. US Patent No. 540485. Filed May 8, 1993; Issued April 4, 1995.

[2] Feng Chen, David A. Koufaty, and Xiaodong Zhang. 2009. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the 2009 ACM SIGMETRICS/Performance*. Seattle, WA, 181–192.

[3] Mei-Ling Chiang, Paul C. H. Lee, and Ruei-Chuan Chang. 1999. Using data clustering to improve cleaning performance for flash memory. *Software-Practice and Experience* 29, 3 (1999), 267–290.

[4] Cagdas Dirik and Bruce Jacob. 2009. The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrenty, device architecture, and system organization. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*.

[5] Laura M. Grupp, John D. Davis, and Steven Swanson. 2013. The harey tortoise: Managing heterogeneous write performance in SSDs. In *Proceedings of the 2013 USENIX Annual Technical Conference (USENIX ATC'13)*. 79–90.

[6] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*.

[7] Jen-Wei Hsieh, Tei-Wei Kuo, and Li-Pin Chang. 2006. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage* 2, 1 (Feb. 2006), 22–40.

[8] Min Huang, Zhaoqing Liu, and Liyan Qiao. 2014. Asymmetric programming: A highly reliable metadata allocation strategy for MLC NAND flash memory-based sensor systems. 14, 10 (Oct. 2014), 18851–18877.

[9] Soojun Im and Dongkun Shin. 2010. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *Journal of Systems Architecture* 56, 12 (2010), 641–653.

[10] Swaroop Kavalanekar, Bruce Worthington, Qi Zhang, and Vishal Sharda. 2008. Characterization of storage workload traces from production windows servers. In *Proceedings of the 2008 IEEE International Symposium on Workload Characterization (IISWC'08)*.

[11] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. 1995. A flash-memory based file system. In *Proceedings of the Winter 1995 USENIX Technical Conference (TCON'95)*.

[12] Hyojun Kim, Ki Yong Lee, JaeGyu Jung, and Kyoungil Bahng. 2008. A new transactional flash translation layer for embedded database systems based on MLC NAND flash memory. In *Proceedings of the 2008 International Conference on Consumer Electronics (ICCE'08)*.

[13] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. 2002. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics* 48, 2 (2002), 366–375.

[14] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, and Satoshi Moriai. 2006. The Linux implementation of a log-structured file system. 40, 3 (July 2006), 102–107.

[15] Min Cheol Kwon, Woon Hyug Jee, Dong Jun Shin, and Shine Kim. 2011. Nonvolatile memory system and related method of preserving stored data during power interruption. US Patent No. 20110093650. Filed June 23, 2010; Issued April 21, 2011.

[16] Jongsung Lee and Jin-Soo Kim. 2013. An empirical study of hot/cold data separation policies in solid state drives (SSDs). In *Proceedings of the 6th International Systems and Storage Conference (SYSTOR'13)*.

[17] Jaeil Lee and Dongkun Shin. 2014. Adaptive paired page prebackup scheme for MLC NAND flash memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 7 (2014), 1110–1114.

[18] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim. 2008. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. *ACM SIGOPS Operating Systems Review* 42, 6 (2008), 36–42.

[19] Sang-Won Lee, Won-Kyoung Choi, and Dong-Joo Park. 2006. FAST: An efficient flash translation layer for flash memory. In *Proceedings of the 2006 International Conference on Emerging Directions in Embedded and Ubiquitous Computing (EUC'06)*. 879–887.

[20] Sang-Won Lee, Bongki Moon, and Chanik Park. 2009. Advances in flash memory SSD technology for enterprise database applications. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD'09)*.

[21] Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, and Sang-Woo Kim. 2008. A case for flash memory SSD in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*. 1075–1086.

[22] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sangwon Park, and Ha-Joo Song. 2007. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Transactions on Embedded Computing Systems* 6, 3 (2007), 18.

[23] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random write considered harmful in solid state drives. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (USENIX FAST'12)*.

[24] Mark Moshayedi and Patrick Wilkison. 2008. Enterprise SSDs. *ACM Queue* (August 2008), 32–39.

[25] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2008. *Migrating Enterprise Storage to SSDs: Analysis of Tradeoffs*. Technical Report MSR-TR-2008-169. Microsoft Research Ltd.

[26] Dongchul Park and David H. C. Du. 2011. Hot data identification for flash-based storage systems using multiple bloom filters. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11)*.

[27] Khyugmin Park. 2008. File system support on multi level cell (MLC) flash in open source. In *Proceedings of the 2008 CELF Embedded Linux Conference (ELC'08)*.

[28] Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2005. Analysis and evolution of journaling file systems. In *Proceedings of the 2000 USENIX Annual Technical Conference (USENIX ATC'05)*.

[29] Frankie F. Roohparvar. 2008. Single level cell programming in a multiple level cell non-volatile memory device. US Patent No. 7366013. Filed December 9, 2005; Issued April 29, 2008.

[30] Samsung Electronics Co., Ltd. 2006. 2G x 8 Bit NAND Flash Memory, K9GAG08B0M, K9GAG08U0M, K9LBG08U1M. http://n2k1.com/n2k1/NB7/PDF/K9GAG08U0E.pdf.

[31] Samsung Electronics Co., Ltd. 2010. 32Gb A-die NAND Flash Datasheet, K9GBG08U0A, K9LCG08U1A, K9HDG08U5A. http://dl.btc.pl/kamami_wa/k9gbg08u0a_ds.pdf.

[32] Marco A. A. Sanvido, Frank R. Chu, Anand Kulkarni, and Robert Selinger. 2008. NAND flash memory and its role in storage architectures. *Proc. IEEE* 96, 11 (Nov. 2008), 1864–1874.

[33] SNIA IOTTA Repository. 2011. Microsoft Enterprise Traces - Exchange Server Traces. Retrieved from http://iotta.snia.org/traces/130.

[34] Anil Vasudeva. 2011. Are SSDs Ready for Enterprise Storage Systems. Retrieved from http://www.snia.org/sites/default/files/AnilVasudeva_Are_SSDs_Ready_Enterprise_Storage_Systemsv4.pdf.

[35] Wenguang Wang, Yanping Zhao, and Rick Bunt. 2004. HyLog: A high performance approach to managing disk layout. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04)*. 145–158.
[36] Han Bin Yoon, Yeong-Jae Woo, and Jung Been Im. Method and apparatus for controlling page buffer of non-volatile memory device. US Patent No. 20110199822. Filed February 16, 2011; Issued August 18, 2011.
[37] Jae-Sung Yu and Jin-Hyeok Choi. 2010. Programming methods of memory systems having a multilevel cell flash memory. US Patent No. 7755950. Filed April 30, 2007; Issued July 13, 2010.
[38] George Kingsley Zipf. 1932. *Selected Studies of the Principle of Relative Frequency in Language*. Harvard University Press.

Received March 2016; revised June 2017; accepted September 2017