IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 15, 2020, accepted July 3, 2020, date of publication July 21, 2020, date of current version July 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3010804

Increasing the Lifetime of Flash Memory Based SSDs by Improving the Merge Operation in Flash Translation Layer

PEYMAN FOROUHAR, (Graduate Student Member, IEEE), AND FARSHAD SAFAEI, (Associate Member, IEEE)
Faculty of Computer Science and Engineering, Shahid Beheshti University G. C., Tehran 1983963113, Iran
Corresponding author: Farshad Safaei (f_safaei@sbu.ac.ir)

ABSTRACT The impressive features of Solid-State Drives (SSDs) have made them be used in a wide range of storage systems. NAND Flash memory, as a good choice for SSDs, caused many challenges. The impossibility of data overwritten in this type of memories is their biggest challenge since this constraint ultimately leads to a decrease in memory lifetime. Numerous designs have been developed to overcome the above challenge in the interface layer of flash memories known as Flash Translation Layer (FTL). In this paper, we present a new FTL that aims to improve the lifetime of memory based on an equation. Using this equation, two major challenges in SSDs can be targeted simultaneously and both can be improved. Reduction of the number of unused pages erased in merge operations and simultaneous increase of the number of invalid pages released in the Garbage Collection (GC) operations, are two of the achievements of our proposed FTL. The results of evaluation on real workloads indicate that parameters of unused erased pages decreased by 5% to 11%, invalid pages released increased by 10% to 28%, redundant written pages in garbage collection operations decreased by 8% to 25% and finally the number of erased blocks decreased by 3% to 12%. Therefore, the significant improvement of these four parameters has a direct impact on the memory lifetime.

INDEX TERMS Solid-state disk, flash translation layer, garbage-collection, merge operation, life-time.

I. INTRODUCTION

NAND flash memory is used as a non-volatile memory in Solid-State Disks (SSDs) because of the high density of data storage. This type of memories is generally classified into two categories: Single-Level Cell (SLC) and Multi-Level Cell (MLC). The memory lifetime in SLCs is longer than MLCs, which is due to the physical properties of its construction and the concept of Floating Gate in their MOSFET transistors [1], [2]. In NAND-based SSDs, the reading and writing unit is a single page, which consists of a number of SLC or MLC transistors. However, the erasing unit in the flash memory is a block, which is made up of a number of pages. A unique feature of SSDs is the impossibility of data overwriting. For example, if at time t1, a page is written in the address x of physical memory, then it will not be possible to write a new page at address x at time t2. In this case, the controller will allocate an empty page on another address, so that the new page can be written at that address, and then the address x of the memory will be put in the invalid state. This operation is hidden from the user; however, this page will not be available for reuse unless the block associated with the page at the address x is erased [3].

One of the other most important challenges for flash memories is their limited lifetime. Repeated writing and erasing of memory blocks, over time, cause them to be worn out and, as a result, the information in some parts of the memory could not be properly read [4]–[6]. Clearly, the more we reduce the amount of unnecessary write operations in flash memories, the longer will be their associated lifetime. The write operation, which is carried out in the memory backend section and commanded by the Flash Translation Layer (FTL), can be related to running commands such as garbage collection or merge operation, which the user has no interference with their productions [7].

The FTL is a software layer that manages and controls various memory sections; commands such as garbage collection, mapping, wear leveling, and merge operation are issued and run in FTL [8]–[11]. The following presents a brief description for each of these commands.

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

The mapping space in flash drives is divided into physical and logical addresses. The logical addresses are user-defined and used constantly; each logical address is mapped to a physical address in the main memory and multiple referrals over time to a logical address will repeatedly change its related physical address, which, as previously indicated, is obvious because of the impossibility of overwriting. The mapping of the logical addresses to the physical addresses is carried out in the Mapping Tables. Mapping through mapping tables can take place at the level of the page, block, or a combination of both. There is a physical address in each logical address of a page in the page mapping. In this way, the data access speed is very high, and, on the other hand, a very high space of memory is needed to store the page table. In the block mapping, there is a physical block address for each logical block address, and access to pages of that block is made possible through the offsets. In this method, less space is needed to store the block table. However, when updating a page, you must copy all block pages related to that page in a new block, which results in erosion of memory. The third method, which is the most widely used, is a hybrid of the two preceding methods, so that the storage space is divided into two general parts. The first part is called data blocks, and the second part is called update blocks. Write operations are carried out in data blocks and pages are updated in update blocks or log blocks. Data blocks access Log blocks through block mapping, and accessing the pages in the log blocks is through the page mapping [12]–[14]. This method greatly improves the limits of the other two methods, and most modern FTLs are based on the third method.

As mentioned earlier, due to the update of pages in memory, many invalid pages are created in different blocks that are practically non-usable. The only way to release these pages for reuse is to erase the blocks associated with them. To do so, this requires firstly to copy the valid pages of the corresponding block to another empty block, then to erase the block to release the corresponding invalid pages. This is called garbage collection (GC), implementation of which is efficient when the number of invalid pages of a block is more than an accepted threshold. It is clear that to run this command, some unnecessary ''write'' operations must be done. The more the number of these operations is reduced, the greater the erosion of memory will be prevented [15], [16].

The merge operation can take place in three general forms: partial merge, switch merge, and full merge. Figure. 1 shows the different merge forms. The least costly one is the switch merge; then the partial merge and, ultimately, the full merge are in the next ranks respectively. Therefore, improving full merge can greatly increase the lifetime of the memory [17]. In Log-block based methods of mapping, each log block associates with a number of data blocks from which it has received the data.

In the merge operation, the valid pages of the data blocks are merged with the valid pages of their associating log blocks, and are written in an empty data block, and finally the log blocks and their associating data blocks are then

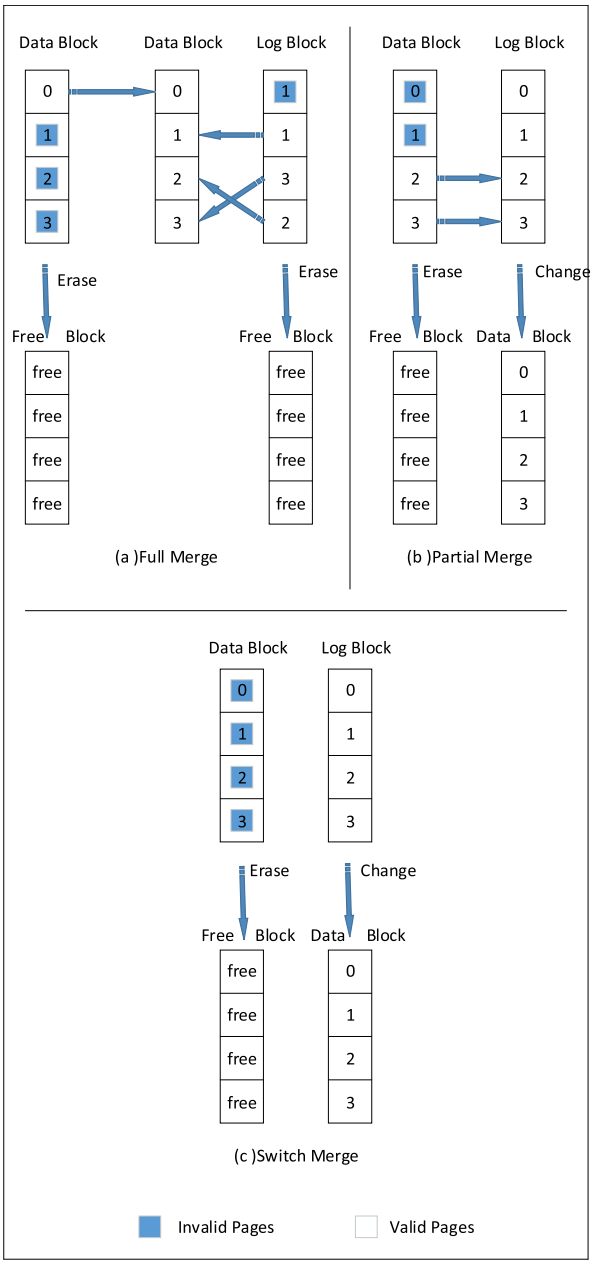

FIGURE 1. (a) An example of full merge, (b) an example of partial merge and (c) an example of switch.

erased and added to the list of usable empty blocks. Here, the maximum cost of memory erosion is associated with the FTL. The obvious reason for this is the transfer of valid pages that carries out multiple unnecessary writings; in addition, many unused pages are erased unnecessarily in the process of erasing.

Wear Leveling helps to make a fair distribution of data at the level of memory blocks, and it causes the lifetime of those sections of the memory on which the hot data is written not to expire earlier than other blocks so that the information stored in that section will not be lost [18], [19].

Focusing on lifetime of the flash memories, especially in SSDs, is important for applications for which the memory

lifetime is critical. For example, in large IT corporations that billions of videos and data are daily uploaded to their servers, it is essential to reduce the memory erosion. Replacing flash-technology-based storage systems and transferring their data to prevent loss of data is very costly and delaying a disk failure (that is, increasing the lifetime of a memory disk) can be very cost-effective [20].

In section II, we will examine some of the most widely used, basic and up-to-date FTLs, and investigate the advantages and disadvantages of each one. Then, in the section III, we introduce the new FTL and show its performance with an example. In the Section IV, a comparison is made to compare the simulation results of the new FTL and the other FTLs previously presented. Finally, the future work and conclusions are expressed in Section V.

II. RELATED WORKS

In this section, some important FTLs are reviewed. Fully Associative Flash Translation Layer (FAST) is considered as the most basic FTL, which is the starting point for most of today's advanced FTLs. The next Flash translation layer is Reuse aware NAND Flash Translation Layer (RN-FTL); this structure, like most modern FTLs, relies on update blocks, and restricts associativity with a parameter called K. In other words, it can be called extended K-Associative Sector Translation Layer (KAST). Ultimately, we will review one of the newest flash translation layers, Dynamic Associative Flash Translation Layer (DA-FTL,) which is a flexible structure in accordance with the conditions governing the system.

In the FAST structure, the physical memory block is divided into two separate sections. The first section consists of data blocks where the initial data is stored. This section covers most of the memory block and is called data block. The second section is made up of update blocks that are called log blocks, and only the updated pages are stored there. This section has a small share of the overall memory space [21]. It is noteworthy to mention that this partitioning is done programmatically and there is no physical change in memory space. When a workload is applied to the storage system, the initial writings are stored in the storage cells in the data block section. If a logical address is referenced to for the second time, then it means that the contents of its physical address should be updated, and, as data overwriting is impossible, a new physical address will be allocated to it. In FAST, this address will be selected from the mapping space associated with log blocks. Communications between the data blocks and their associating log blocks are stored in a block-based mapping table. Each log block in FAST is allowed to receive and store updated pages from all data blocks. The reason for such naming is the unrestricted distribution of data. In the best case, a log block can only be associated with one data block; this occurs if the writing is done serially and consecutively, and in the worst case, a log block can be associated with a number of data blocks equal to its pages. In fact, there will be a problem when there is no space for storing a new page in the log blocks section. In this case, the Merge command is called. Once the command is run, a log block is selected as the victim, and then the associated data block is marked. Each data block is merged with the existing pages in that victim block and then they are written in an empty data block. This process continues for the entire data blocks associated with the victim block, and then the victim block, along with its entire associated data blocks, is erased and added to the empty block list. Finally, a new log block is assigned to the log blocks list to store the recent updated page. The data blocks associated with the log block contain many unused pages, and given the fact that there is no limitation for associativity, the number of blocks containing unused pages can be very large. These unused pages are unnecessarily erased in merge operations. In block-erasing step, this leads to severe erosion of memory. One of the advantages of this structure is the full use of the pages of log blocks, and it is to be ensured that the log block selected as the victim does not contain an empty page. However, due to the limited storage space of the log blocks, the Merge command is executed repeatedly and many unused pages are unnecessarily erased.

RN-FTL as already mentioned, is the advanced form of KAST [22]. In KAST, a constraint K is considered for associativity of log blocks with data blocks. It means that each log block is allowed to be associated with only K data blocks. This provides a guarantee for calculation of the worst case in the merge operation, and a log block in merge operation requires to be merged with at most K data blocks. Reduction of the number of merges prevents the severe erosion that occurs in structures like FAST. However, this constraint itself causes some problems, and one of the most important of these is that there is no guarantee that all pages of a log block used before are to be erased in merge operation. In fact, in KAST, filling log blocks is not the only required condition for entering the merge phase. In addition, there will be a situation where a page of a data block is updated, but no log block has already been allocated to this page. Therefore, it is necessary to assign a new log block to this data block. However, there will be blocks in the list of log blocks that have empty space for the new data storage, but their K limit does not allow them to receive new blocks. The RN-FTL structure provides a solution to solve this problem [23]. The idea of this design is that if a log block is forced to join the merging process because its k capacity is full, while there are plenty of unused pages, then, after merging it with its associated data block, it is added to the list of random log blocks and it will not be erased. This method improves efficiency and prevents erosion of memory blocks largely. Yet, as mentioned before, a limited number of memory blocks can be placed in the update blocks, and if the list of log blocks exceeds the standard limit, we will have to erase some of them and transfer them to the data block list; the repeated erase operations will lead to a decrease in memory lifetime.

By reviewing different FTLs and different solutions presented in them, DA-FTL by a new structure is presented

that is highly flexible [24]. This structure is designed based on update blocks, and has benefited from ideas presented in KAST and other FTLs. The main difference of this FTL is in its merge operation section. When a log block is to be selected as the victim for merge operation, the data blocks associated with this log block are checked before entering the merging phase. The data blocks that repeatedly undergo the merge operation will impose considerable costs in terms of erosion to memory; therefore, we move it to another log block or in other certain circumstances, if there is an empty space in the victim log block, we can move the available data block in other log blocks that are ready to be merged to the victim log block before the merge operation so that we can make a proper merge operation by creating an ideal state. Obviously, the K constraint in the KAST structure will not allow any transfer; but in DA-FTL, the K value in the update blocks has changed from the static state to a dynamic one, so the K value of a log block can exceed its constant limit considering the governing circumstances to improve the merge operation. Sometimes this limit will be lower depending on the situation. The results show remarkable improvement during the merge operation. However, the overhead due to the transfer of pages of a data block to log blocks and the change of association in different logs cannot be ignored, and sometimes the excessive increase of association in a log block can create difficult conditions for future input/output (I/O) operations; this requires an efficient management to prevent the crisis. This is what makes the structure of the FTL extremely complex.

III. THE PROPOSED FLASH TRANSLATION LAYER

In this section, a new FTL architecture is proposed. It is called Optimized Victim Select (OVS). Like other recent architectures, this FTL is based on update blocks, and the memory space is programmatically divided into two sections, the data blocks and log blocks. The idea behind this design is to use the ability to restrict association in log blocks without causing any physical or software changes in the main structure of the memory. This structure attempts to avoid selecting log block with high merging costs as victim block in merging operations as much as possible, and practically, to delay their merge so that they will be in a better situation when this command is run for them. The main goal of this FTL is to reduce the number of unused pages that are unnecessarily erased in the merge operation and thereby cause a significant reduction in the lifetime of the memory cells. It has also attempted to release as much invalid pages as possible in the merge operation to be reused.

Log blocks are divided into random and serial log blocks, so that we may benefit from the advantages of the switch and partial merge operations as much as possible. Table 'A' (Associative table) is considered for each log block to specify the associated data block. An association constraint is considered in the new FTL as in KAST. We store such information as the number of valid and invalid pages of each data block in this table.

A. BASIC DEFINITIONS

Addresses at the user level and in the workloads are recognized as I/O_Address and according to (1); the Logical Block Address (LBA) is obtained by dividing I/O_Address by the block size.

$$LBA = \frac{I/O\_Address}{Block\_Size} \tag{1}$$

Valid_Count, the number of valid pages and Invalid_Count, the number of Invalid pages available at a Physical Block Address (PBA) are obtained from the Valid_List and Invalid_List tables according to the (2) and (3).

$$Valid\_Count = Valid\_List(PBA) \tag{2}$$

$$Invalid\_Count = Invalid\_List(PBA) \tag{3}$$

Unused_Page, the number of unused pages contained in a data block is calculated of (4).

$$Unused\_Page = Block\_Size - [Valid\_List(PBA) + Invalid\_List(PBA)] \tag{4}$$

In (3) and (4), Valid_Count and Invalid_Count are respectively the equivalents of the number of valid and invalid pages of physical block in the PBA address, and Unused_Pages represents the number of unused pages of a physical block in the PBA address.

B. OPTIMIZED VICTIM SELECT ALGORITHM

Suppose that after applying a string of Inputs/Outputs (I/Os) to a flash-based storage system such as an SSD, there is no possibility of updating a new page of a data block due to the lack of an empty association capacity in log blocks. In this case, a log block is selected as the victim and is put into merge operation so that the new page can be updated and stored by releasing it. Log blocks are placed in a linked list. The proposed FTL checks log blocks from the beginning to the end of the list. For each log block, we obtain the SEL parameter, which is computed from the (5).

$$SEL = \sum_{i=1}^{K} (Invalid\_list(PBA_i) - Unused\_Pages) \rightarrow$$

$$SEL = \sum_{i=1}^{K} [2 * Invalid\_List(PBA_i) + Valid\_List(PBA_i) - Block\_Size] \tag{5}$$

In (5), the difference between the number of invalid pages and the number of unused pages is calculated for each of the data blocks associated with that log block, and finally they summed together. The reason for choosing such an equation is that the more the number of invalid pages supposed to be released in the merge operation is, the better it is. Thus, we select the positive sign for the Invalid_List parameter. On the other hand, the lower the number of unused_Pages of a data block is, the better is the situation of that block for

the merge operation. Therefore, we consider a negative sign for the parameter of the number of unused pages of a block, which is obtained based on the (6).

$$Unused\_Pages = Block\_Size - [Valid\_List(PBA_i) + Invalid\_List(PBA_i)] \quad (6)$$

We calculate this difference for all data blocks associated with that log block which is equivalent to K, and then we sum up them together to obtain a suitable criterion for choosing a victim log block. Now in the linked list of the log blocks, SEL(Log_i) is compared to SEL(Log_j), and the bigger one is considered as the Victim parameter. This will continue for the entire linked list, and at the end, the Victim is the address of the log block to be included in the merge operation.

Regarding the reasons for the parameter SEL setting, the proposed FTL provides a way to run Garbage Collection and merge functions in the most optimal conditions. The general view is that a log block should be selected for merge operations that has two features. First, the total number of unused pages of the associated data blocks is the lowest, and second, the sum of invalid pages of the associated data blocks has the most number. This is achieved by providing the SEL equation. By using SEL, a log block is selected for the merge operation to apply the least amount of erosion to the memory blocks after the merge function is executed. It is important to note that the weight value of releasing an invalid page (to become a valid and usable page) is the same as erasing an unused page, because in both cases there is a discussion about using part of the memory space is the size of the cells that make up a page. In (5), the sum of the values of the negative parameter affecting the lifetime of the flash memory is subtracted from the sum of the values of the positive parameter affecting the lifetime of the memory (given that the values of these parameters are equal) and a number is obtained for SEL. It should be noted that if the FTL selects the victim log block without any calculation and prioritization, many unused pages may be erased for no reason, or the least invalid pages in the merge operation may be released for reuse, which can lead to erosion.

We will investigate the proposed FTL pseudocode in the following. This section includes four basic functions that we explain briefly. In the function ''write'' of Fig. 2, once an I/O command is read from the workload, the part related to the address is extracted so that the data could be written in its corresponding address. The logical block address is obtained through dividing this address by the block size. The offset of the page is computed based on the remainder of division of LBA by the block size. In a granular mapping table, at the block level, it is examined whether a physical block has already been assigned to this logical address or not. In the case that this section of the mapping table is empty, a block data is selected from the free physical block space of the flash memory and its address is mapped onto the LBA in the mapping table. Then, the data is written in the correct offset of the PBA, and the Valid_List counter increases by

```
1   write(event)
2    BEGIN
3      LBA: =event.address div BLOCK_SIZE; //LBA : Logical Block
    Number
4      offset: =LBA mod BLOCK_SIZE;
5      if(map.table(LBA) == EMPTY) // map.table: map  LBA to PBA
6        PBA=get.new_BLOCK; // PBA: Physical Block Number
7        map.table(LBA)=PBA;
8       write data at PBA ;
9       Valid_List(PBA)++;  //  Increase PBA valid pages
10     end of if
11     else
12       if (PBA.offset == EMPTY)
13         write data at PBA offset;
14         Valid_List(PBA)++;  // Increase PBA valid pages
15       Else
16         Invalid_List(PBA)++;  // Increase PBA invalid pages
17         call: write_to_log_block(LBA,PBA,offset,DATA)
18      end of else
19    END
```

FIGURE 2. "wrtite" function of proposed FTL.

```
1   write_to_log_block(LBA,PBA,offset,data)
2     BEGIN
3       if (offset ==0)
4         write DATA to the serial_log_block;
5       else
6         if(page.table(PBA) == random_log) // page.table :map PBA to
    random_log BLOCKs
7           if (there is unused page in random_log)
8             write DATA at the next unused page in random_log;
9           else
10            call: merge(random_log)
11        else
12          if (there is a free BLOCK in random_log linked-list)
13            page.table(PBA)=random_log;
14            write DATA at the first unused page in random_log;
15          else
16            scan the random_log linked-list:
17              if (associativity of a random_log is less than K) // K: limit on
    associative
18                write DATA at the first unused page in random_log;
19              else
20                call: OVS_function;
21            end of scan
22          end of else
23        end of else
24      end of else
25    END
```

FIGURE 3. "wrtite_to_log_bock" function of proposed FTL.

one (according to the writing of the new page). However, if a physical address has already been assigned to the LBA, there will be two scenarios. In the first scenario, the offset of the page is free in the physical block and the page will be written in that offset and then, the Valid_List increases by one (according to the writing of the new page). In the second scenario, the corresponding offset contains a value that shows the page update operation in that offset, and then, the Invalid_List increases by one (according to the page updated of PBA). Finally, function ''write_to_log_block'' is called.

In Function ''write_to_log_block'' of Fig. 3, the updated page is written in the update blocks, that is, the log blocks. If the offset of the updated page is zero, this page is written in the serial log blocks. In the case that a random log block is assigned to a PBA in the related mapping table and the log block has an unused free page, the updated page will be written in the first free cell in the random log block; otherwise, the function ''merge'' is called. However, if no random log block is previously assigned to PBA, there will be two scenarios. In the first scenario, if there is a free block

```
1  OVS_function(random_log)
2    BEGIN
3      for (i=1; i<K;i++)
4        Invalid(random_log(x)): =sum(Invalid_List(PBA(i))); // sum of
   invalid pages in all associated data blocks of random_log(x)
5        unused(random_log(x)): =sum (BLOCK_SIZE-[Valid_List(PBA(i))
   +Invalid_List(PBA(i))) ;//sum of unused pages in all associated data blocks
   of random_log(x)
6        SEL (random_log(x)): =Invalid(random_log(x)-
   unused(random_log(x)); // SEL is the Best Victim Choice Indicator
7      end of for
   /* Updated Valid_List()  and  Invalid_List() are available from previous
   functions, and you don't need to scan any data blocks to get these values. */
8      for(i=start;i!=end-1;i++) // start/end is the first/end in random log blocks
   linked-list
9        if (SEL (random_log(i)<SEL (random_log(i+1))
10         victim: =random_log(i+1);
11       else
12         victim: =random_log(i);
13     end of for
14     merge(victim);
15   END
```

FIGURE 4. "OVS" function of proposed FTL.

in the list of random log blocks, we select a free block, assign it to the PBA and write the updated page in the first free cell of the log block. In the second scenario, there is no unused log block; thus, the linked list of random log blocks is scanned and if a used log block is found with an associative limit less than the parameter K, the page will be written in the first unused free cell in the log block. Nonetheless, if all random log blocks have reached their associative limit and there are still unused pages, the function "OVS" is called.

We define the 'SEL' parameter in the function "OVS" to select the best random log block for the merge operation. This parameter is defined to select the log blocks with the largest number of invalid pages and the least number of unused pages in all data blocks corresponding to that log block. In this function shown in Fig. 4, all the data blocks corresponding to a random log block are examined in a 'for' loop. First, the sum of all invalid pages is computed in data blocks; then, the sum of all invalid pages in all data blocks corresponding to the random log block is calculated (the number of unused pages of a data block is obtained by subtraction of the block size and sum of valid and invalid pages of that block).

It is of a significant importance to note that updated Valid_List and Invalid_List are available from the other functions, and there is no need to scan any data blocks to get these values, so the SEL is obtained in the shortest possible time with a simple calculation. In the 'SEL' equation the sign of the total invalid pages is positive (that is, the greater the number of invalid pages be released in the merge operation, the better it is) and the sign of the total unused pages is negative (that is, the fewer the number of unused pages be erased in merge operation, the better it is). Then, the log blocks with greater 'SEL' parameter are selected as the victim for the merge operation. Finally, this random log block is sent as the argument to the function "merge".

In the function "merge" of Fig. 5, we first find all of the data blocks corresponding to the victim log block. Next, the valid pages of the data blocks and the victim log block are copied in a new data block in the list of free data blocks, and, for each page copy in the new free block,

```
1  merge(victim)
2    BEGIN
3      find(associated.data_blocks) ;//find all data blocks associated to
   the victim block
4      copy_to_freeblock(valid_pages) ;//copy all valid pages in the data
   blocks and victim block to a new data block
5      increase_valid_pages(freeblocks); // Increase valid pages counter of
   new data blocks for each copy of the page
6      erase(victim);
7      erase (associated data blocks);
8      ADD_FreeBlock(victim and associated data blocks);
9      clear(valid_pages_counter); //Clear valid pages counters of all associated
   data blocks
10     clear(invalid_pages_counter);  //Clear invalid pages counters of all
   associated data blocks
11     update map tables;
12   END
```

FIGURE 5. "Merge" function of proposed FTL.

the Valid_List counter increases by one. After GC operation, the victim block and all its corresponding data blocks are erased and added to the list of reusable empty block. Then clear Valid_List and Invalid_List counters of all associated data blocks. Finally the mapping tables are updated.

C. AN EXAMPLE

Here is an example to explain the proposed solution in details. Suppose the physical memory space has 16 blocks, each block size being 4 pages. There are 12 blocks in data block section, and 4 blocks in log block section; there is one log block for serial writing, and the other three blocks are for random writing. The numbers above each block indicate their physical address, (that is, PBA). The association rate of a K is defined to be half the size of a block, which is 2 in this case. After applying a string of I/Os, the memory schema will be as shown in Fig. 6.

By executing the next command on the I/O workloads, the data of 'L' update command is issued by data block number 5 that has not been previously assigned to any log block. There is empty space in log blocks, but the K value has reached the maximum amount (that is, 2) in this case, in all of them. Therefore, a log block should be selected as the victim and put into the merge operation so that we can execute the current command after it is erased. At this stage, the tables presented in the proposed FTL design section are reviewed. The Decision Table is adjusted as Table 1 according to the equations presented in Section III.

In the last column, you can see the SEL parameter for each of the log blocks. Given that SEL is the largest in log14, this block is selected as the victim block, and is included in the merge operation.

Table 2 shows how many unused pages will be erased and how many invalid pages will be released if each of the log blocks are selected as the victim block in the merge operation.

As you can see in Table 2, when Log14 is selected as the victim block the lowest number of the unused pages are erased and the highest number of invalid pages is released in the merge operation.

IV. EVALUATION

To evaluate the proposed FTL, it is compared with the FAST as a base FTL, RN-FTL as the closest design to the proposed

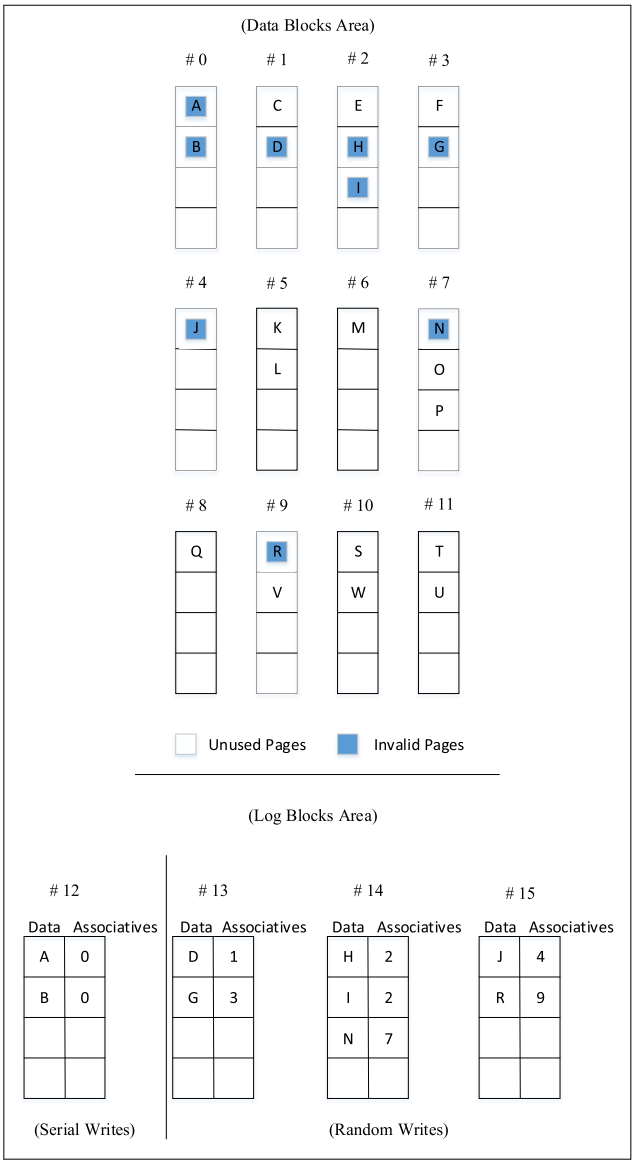

FIGURE 6. Example of a Memory Structure after Running a String of I/O Operations.

TABLE 1. The Decision Table to Select the Victim Log Block.

LOG-Block	Associative	valid	invalid	unused	SEL
13	1	1	1	2	-2
	3	1	1	2	
14	2	1	2	1	+1
	7	2	1	1	
15	4	0	1	3	-3
	9	1	1	2	

structure and DA-FTL as a new FTL. The FlashSim simulator is used to execute and run each of the FTLs [25]. Simulations are driven by four open access workloads called, Fin1, Fin2, MSN and LiveMapsBE. The Fin1 and Fin2 traces were collected at a large financial institution. Financial1 is write intensive and 77.9% of the commands are "write" type. Financial2 is read intensive and only 18% of the commands are "write" type [26]. MSN [Microsoft Enterprise Traces]

TABLE 2. The Number of Unused Pages Erased and the Number of Invalid Pages Released in the GC Operation If Each of the Log Blocks were selected as the Victim Block.

Victim LOG Blocks	Unused Pages Erased	Invalid Pages Released
13	2+2=4	1+1=2
14	1+1=2	2+1=3
15	3+2=5	0+1=1

TABLE 3. Simulated SSD Specifications.

Characteristics	Description	Samsung 4GB
SSD_Size	Chips per SSD	4
Package_Size	Dies per Chip	8
Die_Size	Planes per Die	4
Plane_Size	Blocks per Plane	128
Block_Size	Pages per Block	128(Page)
Page_Size	Words per Page	2KB

was collected at Microsoft's several live file servers, and unlike financial traces, their requests have larger sizes and are sequentially burly. LiveMapsBE [Microsoft Production Server Traces] was collected for LiveMaps back-end server for a duration of 24 hours, and the number of switch merge and partial merge operations is higher than the number of full merge operations [27].

Using different workloads with unique features can challenge the proposed FTL. The proposed scheme is implemented in simulation of the SSD-Samsung K9G4G08U0A 4GB platform with the specifications provided in Table 3 [28].

The defined parameters for evaluating and comparing the FTLs include the total number of block erase commands at the end of each workload, the total number of unused pages that are erased during the merge operation, the total number of invalid pages that are released at the end of each workload and the total number of full merge operations. The main cause for selecting each of the above parameters is the efficient effect each of them can have on the memory block lifetime.

Figure. 7 compares the number of unused pages that are erased in GC operations in different FTLs over 4 real workloads. As indicated in the Section III, the priority for OVS-FTL is to reduce the number of unused pages that are unnecessarily erased. This is accomplished with the correct selection of the victim block before the execution of the merge operation.

Choosing a victim block that contains the smallest number of unused pages inside itself followed by choosing its associated data blocks will allow us not only to obtain the most optimal parameter, but also it makes it possible to give the other blocks with higher number of unused pages the opportunity to find a better position by continuing the process of input/output commands. Consequently, they will impose lower cost on memory erosion if they are selected as the victim block. The proposed FTL, considering Fig. 7, clearly improves the parameter of the number of unused pages erased in GC operations and decreased up to 11%.

Figure. 8 shows the number of invalid pages in the victim log block and its associated data blocks, which are released

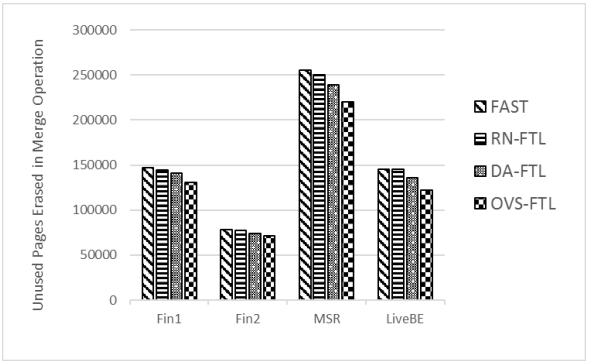

FIGURE 7. Comparison of the Number of Unused Pages Erased in GC Operations in Different FTLs.

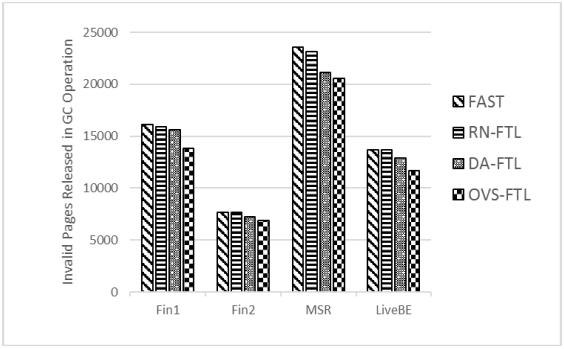

FIGURE 8. Comparison of the Number of Invalid Pages Released in GC Operations in Different FTLs.

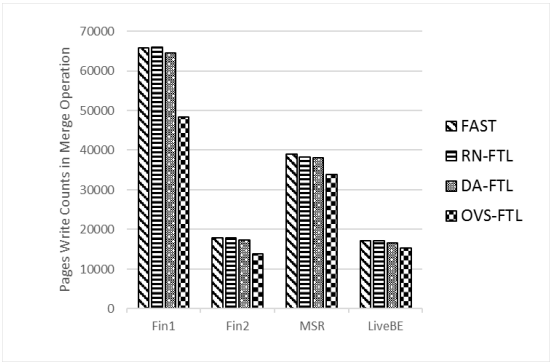

FIGURE 9. Comparison of the number of pages write during the merge operation in different FTLs.

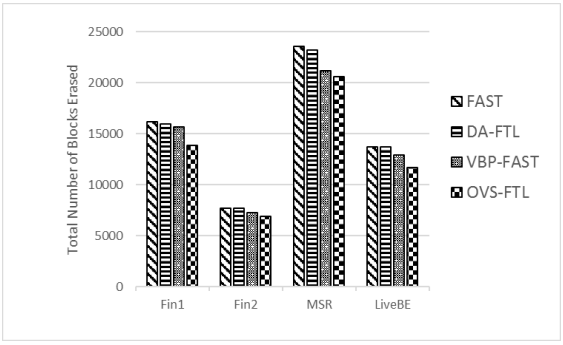

FIGURE 10. Comparison of the total number of erased blocks during the merge operation in different FTLs.

in GC operations. As we know, the larger this parameter, the higher efficiency of memory space it has.

In the proposed FTL, the parameter affecting the selection of a log block as the victim is derived from the (5) (the number of invalid pages minus the number of unused pages). The lower is the number of unused pages in a log block and its associated data block, the higher will be the chance of that log block to be selected as the victim block. The result of this can be seen in the evaluation shown in Fig. 7. This equation also shows that the higher is the number of invalid pages of a log block and its associated data block, this equation will have a larger number, and again the chances of selection of that log block as a victim will be higher. Improvement in the number of invalid pages released following the merge operation, comparing the proposed FTL to the other FTLs, can be seen in Fig. 8. This parameter is increased to 28% in the proposed FTL.

Figure. 9 and Figure. 10 show 25% decrease in the number of pages written and copied during the merge operation, and 12% decrease in the number of blocks erased in the GC operation in the proposed FTL compared to other FTLs; the obvious reason for this is the postponing of the merging operations in the log blocks for which the resulted number of differential parameters are smaller than the rest of the log blocks, which consequently improves the use of empty space in their associated data blocks. Postponing the merge operation reduces the number of referrals to the Merge command; in other words, it reduces repeated calls for the Merge command, which in turn reduces block-erasing operations in GC operations.

What is remarkable is the addition of overhead time to perform calculations and search for the best victim block among log blocks. Clearly, there is a trade-off between getting a good parameter and losing another. In this FTL, to get a longer lifespan, we have applied an overhead time to select the best victim block for merge operations. All available FTLs to select the victim block in the merge operation have performed a process that ultimately applies an overhead to the system. The important point is that the priority of flash-based storage systems, according to the limited number of write and erase operations in them (which is related to their physical characteristics) increases their lifespans, and given the very high speed of SSDs (due to the elimination of mechanical parts and full-electronic structure), increasing lifespan as a major challenge in flash memory is a much higher priority than process execution speed.

The worst-case scenario in the proposed FTL occurs when the amount of SEL between log blocks during the merge function does not differ significantly, and this indicates that the additional calculations performed were somewhat worthless and only applied overhead to the system. The greater the difference between the maximum SEL value and the minimum SEL value, the better the proposed FTL performance. In OVS FTL, SEL parameter variables are updated in the associated data blocks after each command of the input workload (this variable is located in OOB). Consequently, this helps

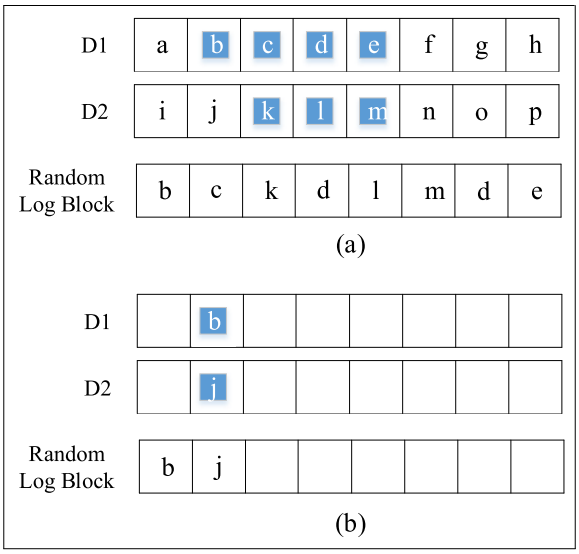

FIGURE 11. The best and worst memory situation for OVS-FTL performance.

us to get SEL for each log block with a simple calculation and perform a simple search to find the best SEL parameter among the log blocks during the merging process. In the worst case, the process can be as long as log-num, which is the total number of log blocks. It is noteworthy that in FTLs based on the hybrid structure, the space allocated for update blocks is much smaller than the data blocks, so it will not take long for this search. This helps to ensure that, if at the time t0, the merge process is not significantly improved by selecting the victim block using the proposed method and using the SEL parameter, then at the time t1 of the merge function, it is not necessary to perform all calculations to obtain SEL for each of the log blocks as the update of the main SEL parameters for all data blocks has already been done and can be easily calculated.

For example, suppose a log block corresponds to two data blocks D1 and D2, and each block has 8 pages. In the best case, the data blocks can be filled as shown in Fig. 11-a, and 7 updated pages from D1 and D2 are written in the random log block. It is of a great value to mention that the update of the eighth page of one of the two data blocks fills the capacity of the log block. So this block is sent to the merge function to release its pages without interruption. It is assumed that the call to the merge function is due to the association limitation. In this case, SEL is +7 according to the (5). In the worst case, the data blocks can be filled as shown in Fig. 11-b and in this case, SEL is equal to -12. Therefore, in this memory with 8-page blocks, the SEL value can be in the range between -7 to +12. The greater the difference between the lowest value and the highest value of SEL when selecting the victim block is, the better the proposed FTL will perform. Conversely, if this difference is minimal, the OVS efficiency decreases.

It is important to note that the existence of valid pages in data blocks makes more copies when the merge function executes, but it is obvious that if a page remains unused, it is

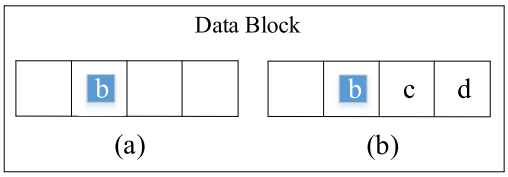

FIGURE 12. Compare SEL for a data block filled with valid pages or unused pages.

equivalent to erasing a page that was not used in the merge operation, so it is a negative score for SEL and if it is filled with a valid page, a page transfer operation will occur when merging data block and log block. So, again, a negative score will be generated when the unused page is erased, leading to memory erosion. Therefore, these two situations will have the same effect on SEL for a page when performing the merge function.

For example, suppose we want to get SEL for the data block in the Fig. 12, and consider copying valid pages as a negative factor, in which case the SEL equation changes as (7).

$$SEL = (invalid\_pages) - (unused\_pages) - (copy\_pages) \tag{7}$$

So according to the new SEL equation, in Fig. 12-a, which does not have a valid page, SEL is equal to -2, and for Fig. 12-b, which assumes two valid pages, SEL is equal to -2. Obviously, the result will not change.

In presenting of (5), the negative impact of page copy has been eliminated according to the presented explanations. Therefore, this helps to prioritize the selection of blocks that have more valid pages and fewer unused pages in the implementation of the merge function; as a result, it increases the efficiency of memory space.

V. CONCLUSION

In this paper, we presented a new FTL with the main purpose of increasing the lifetime of the memory blocks. The proposed solution relies on reducing the number of unused pages that are erased in merge operations, increasing the number of invalid pages that are released in GC operations, reducing the number of erased blocks and reducing the number of pages copied during the merge operation. To evaluate the proposed solution, we implemented it together with three other FTLs in the FlashSim simulator; the used I/Os should be real workloads if an accurate evaluation is intended. The proposed FTL significantly improved each of the parameters affecting the memory lifetime. The remarkable point is that the focus is on random data, and we know that the highest cost in memory blocks erosion is related to the merge of random log blocks. It was attempted to use workloads with a greater impact in order to challenge the proposed FTL because in the section related to serial I/O operations, our FTL acts like other FTLs. The value of association between the log blocks and the data blocks for the FTLs that are based on this parameter, such as our proposed FTL and RN-FTL, has been obtained by repeated executions and various configurations in an optimal state. As a suggestion for future works, it will be beneficial to

present a relationship to calculate the exact optimum value of associativity between log blocks and data blocks. In addition, scanning update blocks to select the best victim block applies a time overhead that can be improved as a goal by providing a solution to this bad parameter.

REFERENCES

[1] R. Baker, *NAND Flash Memory Technologies* (IEEE Press Series on Microelectronic Systems). Hoboken, NJ, USA: Wiley, 2009.

[2] G. Sun, Y. Joo, Y. Chen, Y. Chen, and Y. Xie, "A hybrid solid-state storage architecture for the performance, energy consumption, lifetime improvement," in *Proc. High-Perform. Comput. Archit.*, 2010, pp. 51–77.

[3] J. Kim, J. M. Kim, S. H. Nog, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.

[4] M. Kang, W. Lee, and S. Kim, "Subpage-aware solid state drive for improving lifetime and performance," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1492–1505, Oct. 2018.

[5] P. Papavramidou and M. Nicolaidis, "Test algorithms for ECC-based memory repair in ultimate CMOS and post-CMOS," *IEEE Trans. Comput.*, vol. 65, no. 7, pp. 2284–2298, Jul. 2016.

[6] A. Grossi, L. Zuolo, F. Restuccia, C. Zambelli, and P. Olivo, "Quality-of-Service implications of enhanced program algorithms for charge-trapping NAND in future solid-state drives," *IEEE Trans. Device Mater. Rel.*, vol. 15, no. 3, pp. 363–369, Sep. 2015.

[7] X. Y. Hu and R. Haas, "The fundamental limit of flash random write performance: Understanding analysis and performance modeling," Int. Bus. Mach., Endicott, NY, USA, Tech. Rep. RZ 3771, 2010.

[8] P. Forouhar and F. Safaei, "Improving utilization and life-span in parallel aware MLC-based SSD using virtual blocks," *IEEE Access*, vol. 8, pp. 48212–48225, Mar. 2020.

[9] Z. Xu, R. Li, and C.-Z. Xu, "CAST: A page-level FTL with compact address mapping and parallel data blocks," in *Proc. IEEE 31st Int. Perform. Comput. Commun. Conf. (IPCCC)*, Stockholm, Sweden, Dec. 2012, pp. 142–151.

[10] R. Chen, Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "On-demand block-level address mapping in large-scale NAND flash storage systems," *IEEE Trans. Comput.*, vol. 64, no. 6, pp. 1729–1741, Jun. 2015.

[11] D. Ma, J. Feng, and G. Li, "A survey of address translation technologies for flash memories," *ACM Comput. Surv.*, vol. 46, no. 3, pp. 23–31, Jan. 2014.

[12] Y. Guan, G. Wang, C. Ma, R. Chen, Y. Wang, and Z. Shao, "A block-level log-block management scheme for MLC NAND flash memory storage systems," *IEEE Trans. Comput.*, vol. 66, no. 9, pp. 1464–1477, Sep. 2017.

[13] H.-Y. Sung and C.-H. Wu, "Increasing multi-controller parallelism for hybrid-mapped flash translation layers," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.*, Berlin, Germany, 2014, pp. 567–570.

[14] D. Liu, K. Zhong, T. Wang, Y. Wang, Z. Shao, E. H.-M. Sha, and J. Xue, "Durable address translation in PCM-based flash storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 2, pp. 475–490, Feb. 2017.

[15] M. Lin and Z. Yao, "Dynamic garbage collection scheme based on past update times for NAND flash-based consumer electronics," *IEEE Trans. Consum. Electron.*, vol. 61, no. 4, pp. 478–483, Nov. 2015.

[16] Y. L. Wang, K. T. Kim, B. Lee, and H. Y. Youn, "A novel buffer management scheme based on particle swarm optimization for SSD," *J. Supercomput.*, vol. 74, no. 1, pp. 141–159, Jan. 2018.

[17] C. Feng, T. Luo, and X. D. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proc. USENIX FAST*, San Jose, CA, USA, 2011, pp. 6–13.

[18] J. Liao, F. Zhang, L. Li, and G. Xiao, "Adaptive wear-leveling in flash-based memory," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 1–4, Jan. 2015.

[19] L. Zhu, Z. Chen, F. Liu, and N. Xiao, "Wear leveling for non-volatile memory: A runtime system approach," *IEEE Access*, vol. 6, pp. 60622–60634, May 2018.

[20] J. Meza, Q. Wu, S. Kumar, and O. Mutlu, "A large-scale study of flash memory failures in the field," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, Portland, OR, USA, 2015, pp. 177–190.

[21] D.-J. Park, "FAST: An efficient flash translation layer for flash memory," in *Emerging Directions in Embedded and Ubiquitous Computing*. Berlin, Germany: Springer-Verlag, 2006, pp. 879–887.

[22] H. Cho, D. Shin, and Y. I. Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems," in *Proc. Conf. Design, Automat. Test (DATE)*, Nice, France, 2009, pp. 507–512.

[23] D. Liu, Y. Wang, Z. Qin, Z. Shao, and Y. Guan, "A space reuse strategy for flash translation layers in SLC NAND flash memory storage systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 6, pp. 1094–1107, Jun. 2012.

[24] P. Forouhar and F. Safaei, "DA-FTL: Dynamic associative flash translation layer," in *Proc. 19th Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, 2017, pp. 1–5.

[25] *FlashSim*. Accessed: Feb. 2020. [Online]. Available: https://github.com/MatiasBjorling/flashsim

[26] *OLTP Trace From UMass Trace Repository*. Accessed: Feb. 2020. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[27] *BlockIOTraces*. Accessed: Feb. 2020. [Online]. Available: https://iotta.snia.org/traces/list/BlockIO

[28] *Samsung K9G4G08U0A (V1.0)-4GB MLC NAND Flash Data Sheet*, Samsung Electron., Suwon-si, South Korea, Sep. 2006.

PEYMAN FOROUHAR (Graduate Student Member, IEEE) received the B.Sc. degree in computer hardware engineering and the M.Sc. degree in computer architecture engineering from Azad University, Arak, Iran, in 2008 and 2011, respectively. He is currently pursuing the Ph.D. degree in computer architecture engineering with the Department of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran. His current research interests include storage systems, flash memory, and solid state-disk.

FARSHAD SAFAEI (Associate Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the Iran University of Science and Technology (IUST), in 1994, 1997, and 2007, respectively. He is currently an Associate Professor with the Department of Computer Science and Engineering, Shahid Beheshti University, Tehran, Iran. His research interests include performance modeling/evaluation, interconnection networks, computer networks, and high-performance computer systems.