electronics

Article

DSFTL: An Efficient FTL for Flash Memory Based Storage Systems

Suk-Joo Chae [1], Ronnie Mativenga [1], Joon-Young Paik [2], Muhammad Attique [3] and Tae-Sun Chung [1,*]

1 Computer Engineering, Ajou University, Suwon 16499, Korea; topchae@ajou.ac.kr (S.-J.C.); ronniematie@ajou.ac.kr (R.M.)
2 Computer Science and Technology, Tiangong University, Tianjin 300387, China; pjy2018@tiangong.edu.cn
3 Department of Software, Sejong University, Seoul 05006, Korea; attique@sejong.ac.kr
* Correspondence: tschung@ajou.ac.kr; Tel.: +82-31-219-1828

Received: 6 December 2019; Accepted: 10 January 2020; Published: 12 January 2020

Abstract: Flash memory is widely used in solid state drives (SSD), smartphones and so on because of their non-volatility, low power consumption, rapid access speed, and resistance to shocks. Due to the hardware features of flash memory that differ from hard disk drives (HDD), a software called FTL (Flash Translation Layer) was presented. The function of FTL is to make flash memory device appear as a block device to its host. However, due to the erase before write features of flash memory, flash blocks need to be constantly availed through the garbage collection (GC) of invalid pages, which incurs high-priced overhead. In the previous hybrid mapping schemes, there are three problems that cause GC overhead. First, operation of partial merge causes more page copies than operation of switch merge. However, many authors just concentrate on reducing operation of full merge. Second, the availability between a data block and a log block makes the space availability of the log block lower, and it also generates a very high-priced operation of full merge. Third, the space availability of the data block is low because the data block, which has many free pages, is merged. Therefore, we propose a new FTL named DSFTL (Dynamic Setting for FTL). In this FTL, we use many SW (sequential write) log blocks to increase operation of switch merge and to decrease operation of partial merge. In addition, DSFTL dynamically handles the data blocks and log blocks to reduce the operations of erase and the high-priced operation of full merge. Additionally, our scheme prevents the data block with many free pages from being merged to increase the space availability of the data block. Our extensive experimental results prove that our proposed approach (DSFTL) reduces the count of erase and increases the operation of switch merge. As a result, DSFTL decreases the garbage collection overhead.

Keywords: flash memory; flash translation layer; file system

1. Introduction

Flash memory is widely used in solid state drives (SSD) and smartphones because of their non-volatility, low power consumption, rapid access speed, and resistance to shocks. These days, the store for solid state drives, which use NAND flash memory, is increasing fast and even making inroads into the hard drive store [1]. Flash-based storage devices are regarded as a new storage medium that can substitute disks and achieve higher performance for database servers [2]. However, operation of overwrite cannot be executed directly and the unit of operations is different in flash memory. An operation of erase is conducted at the unit of a block, which consists of multiple pages and can match up to 1.5ms. On the contrary, a page is the unit, which is where operations of read and write are conducted, can match up to about 80 µs and 200 µs continuously [3]. However, NAND flash memory has a feature of hardware that a page is erased before being written in the equal location, and this is called operation of erase-before-write (out-of-place-update) [4]. Since the memory portion for erasing is different in size from the size for reading or writing [5], a medium software layer, which is called a flash translation layer (FTL) [6], was presented.

There are two categories of mapping schemes depending on the mapping unit, which is namely the page mapping algorithm [7] and the block mapping algorithm [5]. The page mapping algorithm handles the mapping entries at the page, which is the unit of the operations of read and write. A 16 GB flash memory needs approximately 32 MB of SRAM space to save a page mapping table [8]. Thus, page mapping needs a large mapping table to be saved in the SRAM/DRAM. On the contrary, the block mapping algorithm handles the mapping entries at the block, which is the unit of the operation of erase. The size of the mapping table is decreased by a factor of the block size/page size (128 KB/2 KB = 64). However, as the block-mapping allows updates just at the block, the garbage collection overhead increases as not all the pages in a block will be invalid/dirty. To relieve cons of these page and block mapping, the hybrid mapping algorithm [9] was presented. It uses the block mapping technique to obtain the corresponding physical blocks and the page mapping technique to find available empty sectors within the physical block. Therefore, now, the hybrid mapping scheme is the most popular.

Many FTLs [9–15], which used hybrid mapping schemes, were presented with blocks that are separated by two categories that include data blocks and log blocks. Data blocks save real data and are mapped using block mapping. Log blocks are designed to save the updated data and are mapped using page mapping. When the file system communicates to flash memory, the data are first written to the data block. If data already exists in that particular data block, the incoming new data are then written to a log block. Although hybrid mapping schemes are designed to decrease the number of operations of copy and erase that are needed, they still suffer from the garbage collection overhead, which is an issue of concern.

Firstly, the hybrid mapping scheme invokes a garbage collector whenever there are no free log blocks. Garbage collection needs merging the log blocks with the data blocks. In this case, as operation of partial merge should copy pages in comparison with an operation of switch merge, it is fairly beneficial to execute operation of switch merge instead of operation of partial merge. Compared to operation of switch merge, operation of partial merge should copy valid pages in the data block to the log block. Secondly, if one log block is connected with one data block, the space availability of the log block is low. If one log block is connected with all data blocks, a high-priced operation of full merge will be incurred. While the data block represents the ordinary storage space, the log block is used for storing updated data. It is necessary to adjust availability between a log block and a data block dynamically. Finally, even if there are many free pages in the data block, this data block is erased due to the inefficient data block availability of such systems. Realistically, a data block with many free pages should not be merged, and data blocks with the least free pages should be the ones selected as the merge victims. Although many FTLs have addressed these issues, no scheme satisfies all three of the factors mentioned earlier.

In this paper, we propose a new FTL scheme that provides a dynamic setting for flash translation layer (DSFTL scheme) in NAND flash memory. Our proposed DSFTL scheme aims to decrease the garbage collection overhead by increasing operation of switch merge and reducing the number of operation of erase. To increase the frequency of operation of switch merge and decrease the frequency of operation of partial merge, we use many sequential writes (SW) log blocks. In addition, our proposed scheme increases the space availability of the log block and decreases the high-priced operation of full merge by dynamically coordinating the availability of the log block with the data block. Moreover, even if there is a data block connected with the victim log block, DSFTL ensures that the data block with many free pages is not merged. Thus, the space availability of the data block to be merged with the log block is increased. By using our all algorithms, DSFTL avoids many operations of merge. In other words, we can solve the problem of hybrid mapping, which consequently decreases the overall garbage collection overhead.

Unlike BAST [9] and FAST [10], DSFTL controls availability between a data block and a log block dynamically. Therefore, our scheme avoids many operations of merge and high-priced operation of full merge. Compared to Superblock [11], we do not combine the data blocks to alleviate multiple Out-of-Band area reads and writes. In DSFTL, each log block can be connected to all the data blocks to increase the space availability of the log block compared to KAST [12]. In contrast to DA-FTL [13], we copy small valid pages in a victim block to another log block. Thus, DSFTL decreases availability between a data block and a log block. Unlike LAST [14], DSFTL reduces operation of full merge since we detect the hot pages to avoid unnecessary operations of merge. Compared to MAST [15], we find a victim SW log block, whose data are less written to avoid an operation of partial merge.

The contributions of the paper are as follows:

1. By our experiments, we check that the operation of switch merge is occurred more often than operation of partial merge by using many SW log blocks.
2. We show by our experiment that the space availability of the log block is high, and the high-priced operation of full merge is less incurred as the connection of the data block and the log block is managed dynamically.
3. Based on our experimental results, the space availability of the data block is high, since the data block, which has many free pages, avoids operations of merge.
4. As a result, by using DSFTL, we can decrease the garbage collection overhead in hybrid mapping.

2. Related Work

Many FTLs [9–15], which are based on hybrid mapping log-buffer approaches have been proposed.

The Block Associative Sector Translation (BAST) [9] schemes uses two blocks: a data block and a log block. The data blocks are needed to sequential writes, and the log blocks are needed to random overwrites. In this approach, as the log block is connected just one data block, it has a problem that is called log block thrashing and block availability [10]. To overcome this issue, our proposed DSFTL scheme connects one data block with just a log block when the data are written on a log block for the first time.

Fully Associative Sector Translation (FAST) [10] permits log blocks to be connected in all data blocks. In FAST, a single sequential log block is dedicated for sequential updates, while other log blocks are used to perform random writes. However, due to high block availability, FAST incurs high-priced operation of full merge, which degrades its performance. Moreover, it does not have special techniques to handle the temporal locality in random streams. On the contrary, DSFTL connects one data block with just one log block when data has already been written onto a log block. To reduce operation of merge, DSFTL assigns many SW log blocks.

Superblock FTL [11] combines a set of logical blocks into a superblock. To use the existence of block level spatial locality in workloads, superblocks are mapped at bigger unit, while pages inside the superblock are mapped freely at a smaller unit to any location in many physical blocks. However, this causes a lot of Out-of-Band area reads and writes to service the requests. We proposed not to combine the data blocks in DSFTL to alleviate this. Furthermore, while Superblock FTL does not consider promoting space availability of a data block, DSFTL can promote this by copying data from a victim block to another log block.

K-Associative Sector Translation (KAST) [12] configures the maximum log block availability to control the worst case blocking time by a user. By distributing write requests among log blocks to reduce the log block availability, KAST shows better average performance than other FTLs. However, as the number of data blocks connected with the log block is limited, space availability of the log block in a multi-processing system can be reduced. In addition, it is difficult to accurately predict the write operation pattern. While each log block is connected just by the k number of data blocks at maximum in KAST, each log block can be connected to all the data blocks in our proposed DSFTL. To decrease availability, a log block is connected by just a single data block whenever the data have already been written to the log block.

The Dynamic Associative Flash Translation Layer (DA-FTL) [13] dynamically handles the availability of the data blocks and the log blocks in the system. To reduce the number of blocks to be merged, DA-FTL picks out a data block, which has many free pages as a merge victim by copying data to another log block. However, since the data are copied to another log block, the availability between the data block and the log block gradually increases, which results in large merge costs. On the other hand, high-priced operations of full merge are avoided in DSFTL, since small valid pages in a victim block are copied to another log block.

Locality-Aware Sector Translation (LAST) [14] is optimized for access features of normal purpose computing systems. LAST also uses many SW log blocks to reduce operations of full merge and to increase operations of switch merge and partial merge, which is similar to DSFTL. When there is no SW log block where all the pages are valid date in LAST, they use a least recently used (LRU) replacement policy. We do not use an LRU replacement policy, which is unlike LAST. Instead, DSFTL finds a victim SW log block, for which the data are less written. In addition, this victim log block is applied to an operation of partial merge.

Multi-Level Associated Sector Translation (MAST) [15] divides the log block depending on the features of the data, and it dynamically detects the hot pages to avoid unnecessary operations of merge in FTL. We used this architecture for RW log blocks. On the other hand, for SW log blocks, they follow the fundamental rules of FAST [10]. In the case of DSFTL, for SW log blocks, if there is no SW log block, for which all the pages have a valid date, DSFTL finds a victim SW log block, for which data are less written. In addition, this victim log block is applied to operation of partial merge for an increase of operation of switch merge and a decrease of operation of partial merge.

On the other hand, there are many FTL algorithms based on PCM-based memory systems [16–19].

hFTL [16] is a page mapping algorithm for PCM-based memory systems. This scheme handles a page mapping table in the PCM. In addition, it saves the user data in flash memory. However, it needs a high portion of the PCM for its page mapping table. In addition, it has a problem of frequent revision in its page mapping table since it must add or update address mapping entries whenever the file system uses a write request. To solve this problem, we propose adopting either one of the following two mapping mechanisms: page mapping or block mapping.

PAB [17] is a block mapping algorithm while hFTL is a page mapping algorithm. Thus, compared to hFTL, this scheme needs a small mapping table. In this scheme, a single log block is connected with just a data block. However, this causes the generation of many unnecessary operations of erase on flash memory because of its low space availability. On the contrary, a single log block is connected to all the data blocks whenever data are written to a log block for the first time in DSFTL. Therefore, DSFTL increases the space availability of blocks in the system.

WAB-FTL [18] is a block mapping algorithm. A single log block is connected with all the data blocks. Thus, it incurs severe response delay and generates costly operations of merge. To overcome this, a single log block is connected with just one data block in our proposed DSFTL scheme.

Load-FTL [19] revises a log block policy to redirect the updates to the log blocks shared with the least data blocks. When there are no free pages in log block, it selects a victim log block and identifies the hot data in the log block by comparing the update counts of the hot data to a predetermined updated threshold. On the contrary, DSFTL does not select a victim log block since it always maintains two log blocks at any given time to decrease operations of full merge through the promotion of faster operations of switch merge that permits prolonged flash lifespan.

3. Background

3.1. NAND Flash Memory

Flash memory is popular as an attractive long-term storage medium for SSD and smartphones. It is because it has characteristics of non-volatility, low power consumption, rapid access speed, and resistance to shocks. In addition, as there is no mechanical delay, there is no performance penalty of a random access pattern. Thus, it provides great performance. Moreover, flash memory does not need a fluctuating seek time. Thus, it is good to provide a predictable I/O performance to the real-time systems unlike hard disks.

There are two categories of flash memory: NOR flash memory and NAND flash memory. The use of NOR devices is widespread in the industry. These devices provide an easy memory interface, and they are appropriate of code execution, which makes them ideal for devices that do not need data storage. On the contrary, NOR flash memory provides great performance of the operation of read. However, it has many write and erase times, which disqualifies it from being used as a data storage device. However, these days, as devices become increasingly sophisticated, they are expected to offer more features, richer programs, and save more information locally. Doing this needs larger capacities both for code and for data storage, and it has considerably faster erase/write times. NAND flash memory provides all of this as well as cheaper prices for capacities, which range from 8 MB to 512 MB [20]. Therefore, now, NAND flash memories are more popular.

NAND flash memory can be classified into single-level cell (SLC) and multi-level cell (MLC) flash memory [21], and flash memory is limited in the number of times it can be erased before a failure incurs. For example, SLC (single-level cell) flash memory has an endurance life of 100,000 erase cycles before the wear starts to deteriorate the integrity of the storage. Moreover, SLC flash memory can be erased around 100 times more often than MLC flash memory. SLC flash memory can have a considerably greater endurance life than an MLC flash memory, but SLC flash memory is more high-priced than MLC flash memory [22].

NAND flash memory is composed of a number of blocks. The block is the fundamental unit of operations of erase. In addition, the block consists of multiple pages. The page is the fundamental unit of operations of read and write. Each page is composed of a main data area and a spare area. The real data are written in the main data area and the spare area is usually used to save management information such as the error correction code (ECC) to detect errors [23].

Three fundamental operations can be applied to both categories of flash memory, which include operations of read, write, and erase. The operations of erase are significantly slower than the reads/writes.

However, flash memory has a weak point in that overwrites on an already written block of flash memory are impossible to perform unlike hard disks. To make these overwrites possible, an operation of erase on the written block has to be conducted before overwrites, which deteriorates the performance of a flash memory considerably. In addition, flash memory has features of hardware, and a page has to be erased before some data can be written in the equal location. This feature is called erase-before-write [4]. As the memory portion for erasing is different in size from that for reading or writing [5], a medium software layer called an FTL [6] was presented.

3.2. FTL

FTL is located between a file system and a flash memory, and it provides many roles. The one of the roles of FTL is the wear-leveling, which evenly distributes erase operations to the whole memory blocks. Another role is to recover from a sudden power-off. That is, metadata such as mapping information should be recovered in spite of a sudden power off. Moreover, the main role of FTL is to change the logical addresses from the file system to the physical addresses in flash memory.

There are two categories of mapping schemes depending on the mapping unit: a page mapping algorithm [7] and a block mapping algorithm [5]. In the page mapping algorithm, all logical pages are mapped to a corresponding physical page. The page mapping scheme handles the mapping table as a page that is a unit of operation of read/write. Thus, page mapping requires a large mapping table to be saved in the SRAM.

Block mapping algorithms were presented, as the page mapping algorithm needs a lot of memory space. In block mapping, a logical sector offset within a logical block is the same as a physical sector offset within a physical block. Thus, the block mapping scheme needs a smaller amount of mapping information than the page mapping scheme. However, if the file system needs write operations with the same logical sector numbers, a lot of operations of copy and erase are needed, which degrades the performance.

As both page and block mapping have cons, a hybrid mapping algorithm [9] was presented. In this algorithm, all of the physical blocks were separated between the data blocks and the log blocks. The data blocks are handled by the block mapping scheme while the log blocks use page mapping. For example, first, if a block mapping algorithm is used to get the corresponding physical block, a page mapping algorithm is used to locate an available vacant sector within the physical block. In this algorithm, the page mapping table for a limited number of blocks is retained in addition to the block mapping table. Thus, it satisfies the size limitations of mapping information and reduces the erase-before-write problem drastically. Therefore, hybrid mapping is the most popular these days.

4. Motivation

In this section, we will describe the different categories of merges and why we need to reduce operations of partial merge and increase operations of switch merge. Furthermore, we will explain the availability between a data block and a log block. Finally, we will show why the space availability of the data block should be high.

4.1. Operation of Merge

In the log based FTL scheme, the blocks are separated by two blocks: a data block and a log block. While the data block is used for the ordinary storage space, the log block is used for storing updated data. There are two cases where a new log block should be assigned. The first case is, if there is no log block corresponding to a specific data block, a new log block is assigned. In BAST, as the log block is assigned for just one data block, this problem incurs occasionally. The second case is when the log block is full. As the log block does not have free pages, FTL should assign a new log block. When a new log block is assigned, the valid data in the data block and the log data of the log block need to be merged into a free block. This work is called operation of merge.

There are three categories of operations of merge: a switch merge, a partial merge, and a full merge. Figure 1 shows these operations of merge. Figure 1c shows the operation of full merge. It assigns a free block and then copies a valid page of the data block or the log block to a free block. After it copies all valid pages, the free block becomes the data block, and the previous data block and the log block are erased. As operation of full merge needs many operations of copy and erase, it is the most high-priced operation of merge. The operation of full merge is needed when the pages are updated in a random request. Thus, many hybrid mappings are designed to reduce operations of full merge.

As depicted in Figure 1a, FTL simply erases the data block with just the invalid pages and then changes the log block into a data block. As the operation of switch merge needs just one operation of erase, it is the cheapest operation of merge. It performs just an SW. As shown in Figure 1b, operation of partial merge is similar to operation of switch merge. Compared to operation of switch merge, operation of partial merge should copy valid pages in the data block to the log block. In other words, it needs additional copy operations, and it is executed when the SW does not fill up a block. The copy of the pages causes the garbage collection overhead. However, many authors just approach reducing operation of full merge.

As we can see, operation of partial merge is more high-priced than operation of switch merge. Therefore, we should ensure that FTL uses more operations of switch merge than operations of partial merge.

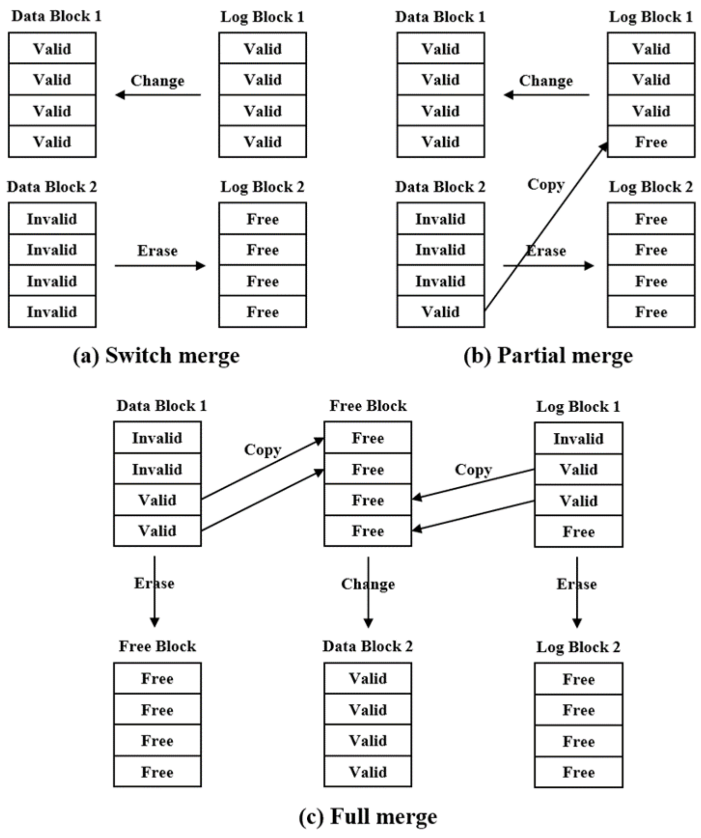

Figure 1. Three categories of operations of merge.

4.2. Availability between a Data Block and a Log Block

The log-buffer-based FTL schemes have a data block and a log block, and these two blocks can be connected with each other. In many existing FTLs, the availability between the data block and the log block is different. For example, in the case of BAST [9], the log block is assigned for just one data block. On the contrary, FAST [10] allows log blocks to be connected to all the data blocks. In the KAST [12] scheme, the number of data blocks that can be connected with one log block is limited to less than k. However, each of these FTLs has problems related to availability.

The BAST [9] scheme has a block thrashing problem [10]. As depicted in Figure 2, we assumed that there are four data blocks and two log blocks. Furthermore, the write pattern is the sequence of P0, P4, P8, and P12. As the log block is assigned for just one data block, P0 is written in *Log* and P4 is written in *Log Block 2*. When the file system issues write P8, a new log block is assigned as P8 does not have space to write. Thus, *Log Block 1* is merged even though it has many free pages. This problem is also related to the space availability of blocks, which will be discussed in the next section.

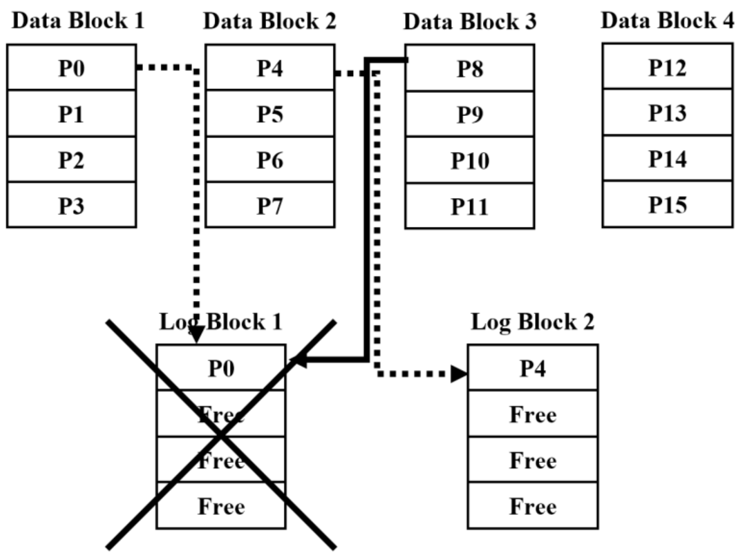

Figure 2. Problem of 1:1 availability.

FAST [10] allows log blocks to be shared by all the data blocks. Therefore, it effectively improves the storage availability of the log blocks and delays the operation of merge considerably longer. However, since the merge is possible for many data blocks connected with the log block, a high-priced merge may occur. As shown in Figure 3, we assumed that there are four data blocks and one log block. Furthermore, the write pattern is the sequence of P1, P5, P9, P13, and P2. When the file system issues write P2, a high-priced merge is executed, since *Log Block 1* is connected with many data blocks. In this case, one log block and four data blocks should be merged.

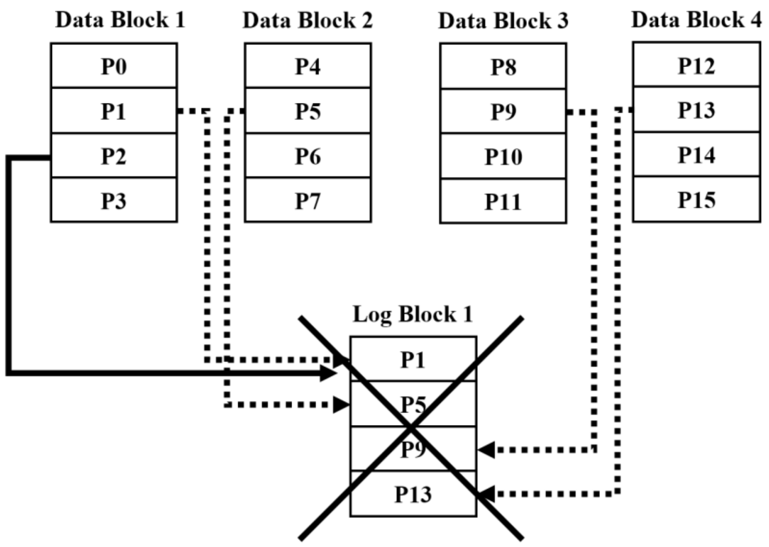

Figure 3. Problem of M:N availability.

In the KAST [12] scheme, to decrease the problems caused by the high availability in FAST [10], the number of data blocks that can be connected with one log block is limited to less than k. However, as the number of data blocks connected with the log block is limited, the space availability of the log block can be reduced. Moreover, it is difficult to accurately predict the write operation pattern.

This causes various problems depending on the availability between the data block and the log block. If the availability between the data block and the log block is low, there is the problem of block thrashing. In the opposite case, a high-priced merge is likely to occur. Then, we should consider the availability.

4.3. Space Availability of a Data Block

When a new log block is assigned, the valid data in the data block and the log data of the log block should be merged into a free block. In this case, even though the blocks to be merged have many free pages, they should be erased. As flash memory has erase/program cycles [24], the lifetime of flash memory is shortened if the blocks to be erased have many free pages. There are many cases where the space availability of a log block and a data block is lowered.

Figure 2 shows the low space availability of a log block. When the file system issues write P8, *Log Block 1* is merged even though it has many free pages. In this case, the space availability is 0.25, and the log block just uses one page out of four pages. This is the problem in BAST [9] where the log block is assigned for just one data block. In other words, if the availability between the log block and the data block is increased, the space availability of the log block is increased.

The important point is how to increase the space availability of a data block. As depicted in Figure 4, we assumed that there are four data blocks and one log block, and the write pattern is the sequence of P0, P4, P8, P12, and P1. When the file system issues write *P1*, four data blocks and one log block are merged. In this case, the space availability of *Log Block 1* and *Data Block 1* is 1. Furthermore, the space availability of *Data Block 2* is also high at 0.75. However, *Data Block 3* and *Data Block 4* have a low space availability at 0.25. Many blocks are merged, and the space availability of the blocks is low.

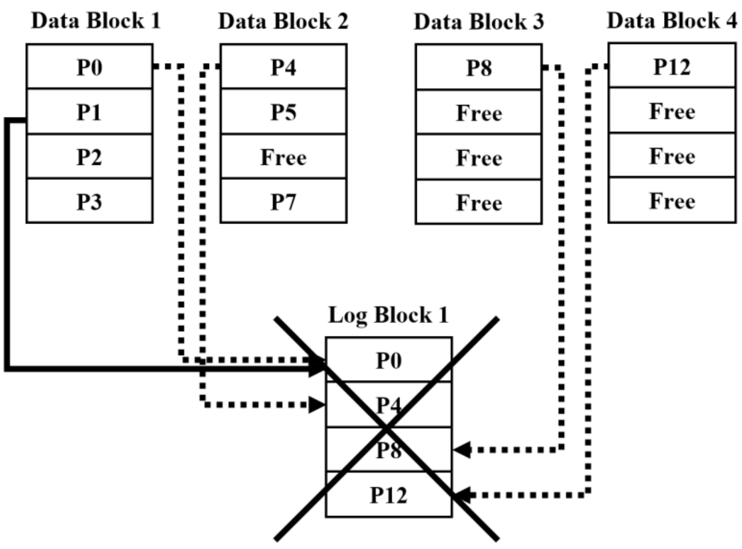

Figure 4. Low space availability of a data block.

We need to increase the space availability of the blocks to be merged. To be specific, it is important to increase the space availability of the data block. If space availability of the data block is increased, the count of erase is reduced. In addition, the lifetime of a flash memory is increased. To increase the life time of a flash memory, we should solve this problem.

5. System Design

In this section, we described the design principles and the implementation of the proposed system DSFTL. Our goal is to decrease the garbage collection overhead.

5.1. System Architecture

We add three algorithms to overcome the disadvantages of existing hybrid mapping. Firstly, DSFTL reduces the occurrence of operation of partial merge, which is more high-priced than operation of switch merge, and it causes operation of switch merge to occur occasionally. Secondly, as our new scheme dynamically adjusts the availability of a data block and a log block, it increases the space availability of the log block as well as avoids a high-priced operation of full merge. Finally, space availability of the data block is high, as the data block that has many free pages avoids operation of merge. As a result, DSFTL decreases the garbage collection overhead.

5.2. Increase of a Switch Merge and the Decrease of a Partial Merge

As noted in the Motivation section, there are three categories of operations of merge: a switch merge, a partial merge, and a full merge. As operation of full merge needs many operations of copy and erase, it is more high-priced than operations of switch merge and partial merge. Therefore, to decrease the garbage collection overhead, operation of switch merge or partial merge are used more occasionally than operation of full merge. To solve this problem, we use MAST [15] architecture, which is the state-of-the-art in hybrid mapping. MAST mitigates asymmetric read and write performance. In addition, it divides log block depending on the features of the data. It then dynamically finds hot pages to avoid unnecessary operations of merge and handles hot and cold pages on attributed areas in the log block. Therefore, we can handle RW log blocks and operations of full merge efficiently.

To handle SW log blocks, we use many SW log blocks to increase operation of switch merge and to reduce operation of partial merge. MAST [15] and LAST [14] also use many SW log blocks. However, in the case of MAST, they follow the fundamental rules of FAST [10]. Thus, there are still many operations of partial merge in this scheme. In the case of LAST, when all the SW log blocks are exhausted, LAST first searches the log block where all the pages are valid data, and it does operation of switch merge. If there is no such log block, LAST selects the victim log block using the LRU (Least Recently Used) replacement policy and then applies operation of partial merge [14]. On the other hand, in the case of DSFTL, if there is no SW log block that has all the pages with valid dates, we do not use the LRU replacement policy. Instead, DSFTL finds a victim SW log block where the data are less written. Then, this victim log block is applied to operation of partial merge to increase operation of switch merge.

Data, for which the page offset is zero, is written in the SW log block. Then, if the page offset is continuous, the data are written in the SW log block. Otherwise, they are written in the RW log block. Furthermore, an operation of switch merge is executed when the SW log block is full. However, if the SW log block is not empty and the data with a page offset that is zero is written, an operation of partial merge is executed. In other words, because of the data whose page offset is zero, the data written in the SW log block loses the opportunity for operation of switch merge. Therefore, DSFTL increases the probability of operation of switch merge by using many SW log blocks. The use of many SW log blocks prevents data written in the SW log block from being erased even if the data whose page offset is zero is written.

Algorithm 1 shows how to operate many SW log blocks in DSFTL. When data with page offset zero is written in the SW log blocks, DSFTL can perform each operation according to three conditions. Firstly, if the data are not written in all the SW log blocks, the data are written to any one SW log block. As all the SW log blocks are empty, it does not matter which SW log block the data are written to. This case is the same as FAST. Secondly, if the data are written to some SW log blocks, new data are written to other SW log blocks. Furthermore, the data in the existing SW log blocks do not do operations of partial merge. The existing SW log blocks have an increased probability of operation of switch merge. On the contrary, in FAST, the data in the existing SW log block operations of partial merge when new data are written. Finally, if the data are written in all SW log blocks, DSFTL finds the SW log block where less data are written. Then, this SW log block does operations of partial merge. Next, new data are written in this SW log block. Except for three cases, we do not need to handle anymore, as an operation of switch merge occurs if the SW log block is full.

Algorithm 1: When data with page offset zero is written

```
   Input: Data
begin
1  if all SW log blocks are not written
2      write data to any SW log block
3  else if some SW log blocks are written
4      do not erase data of these SW log blocks
5      write data to other SW log blocks
6  else /* all SW log blocks are written */
7      erase data of SW log block which data is less written
8      write data to this SW log block
9  end if
```

Figure 5 shows the difference between the FAST and DSFTL when new data are written in the SW log block. While FAST has just one SW log block, DSFTL has many SW log blocks. In this example, FAST has one SW log block and DSFTL has two SW log blocks. In addition, we assumed that the data block is already filled with data, and the write pattern is the sequence of 0, 1, 2, 4, and 3. It is the equal until *data* 2 is written. When *data 4* is written, the SW log block conducts operation of partial merge. Furthermore, *data 4* is written in the SW log block. As *data 3* does not have a sequential pattern, it is written in an RW log block. If the order of *data 4* and *data 3* is changed, the operation of switch merge may occur. Moreover, *data* 3 is not written in the RW log block. In the case of DSFTL, even though *data 4* is written, the data already written in SW log block 1 does not conduct operation of partial merge. Instead, *data 4* is written to another SW log block 2. When *data* 3 is written in SW log block 1, the operation of switch merge occurs.

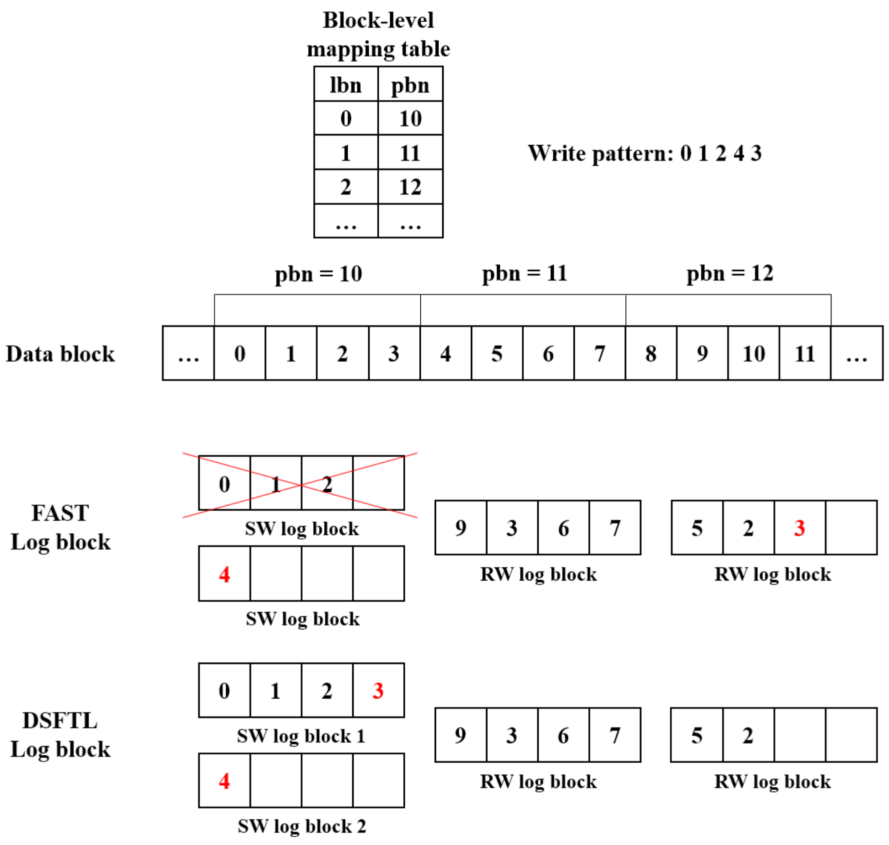

Figure 5. When data are written in an SW log block.

In the case of DSFTL, more operations of switch merge or partial merges can be executed instead of operation of full merge by using the MAST architecture. In addition, more operations of switch merge can be executed instead of operation of partial merges by using many SW log blocks. Unlike previous hybrid mapping schemes, the data for operation of switch merge does not conduct operation of operations of partial merge in the proposed scheme. Moreover, DSFTL prevents all the data from being written to the RW log blocks.

5.3. Adjust Availability Dynamically

There are two blocks in the log-buffer-based FTL scheme. A data block represents the ordinary storage space, and a log block is used for updated data. As noted in the Motivation section, because of the availability between the data block and the log block, there are many problems in FTL.

If the log block is assigned for just one data block, which occurs in BAST, a high-priced operation of full merge is not executed, since there is just one data block connected with the log block selected as the victim block. However, the space availability of the log block is likely to be low. This is called the block thrashing problem. On the contrary, if the log block shares all the data blocks, which occurs in FAST, the probability of operation of merge is reduced, since the space availability of the log block is high. However, as the log block can be connected to all the data blocks, a high-priced operation of full merge may occur. Therefore, we need to adjust the availability between the data block and the log block.

To solve these problems, DSFTL mixes the features of the BAST and the FAST. When the data of the data block are first written to the log block, the data are written to the log block with less written data. The log block can be shared by all the data blocks, which is similar to that of FAST. As the space availability of the log block is increased, there is a lower occurrence of operation of merge. Furthermore, after being written to the log block first, the data of the data block are assigned to just one log block where the data of the data block is already written. Thus, one data block is shared just with one log block. This is similar to BAST. Because of this feature, a high-priced operation of full merge is not executed in DSFTL.

As depicted in Algorithm 2, we can see how DSFTL dynamically adjusted the availability between the data block and the log block. When the data in a data block are written to a log block, DSFTL checks whether the data have already been written in the log block or not. If the data has not already been written in the log block, the data are written to the log block for the first time. Therefore, DSFTL finds a log block with less written data due to the increased space availability of the log blocks in this case. Furthermore, the data are written in this log block. On the contrary, if the data are already written in the log block, the data are written to the log block more than once. The data are written to the log block, which has already been connected because of the reduction of the high-priced merge.

Algorithm 2: When data in a data block is written to a log block

```
  Input: Data
begin
1  if data is not already written in a log block
2     find a log block which data is less written
3     write data to this log block
4  else /* data is already written in the log block */
5     write data to only this log block which have already been connected
6  end if
```

Figure 6 shows the difference between the previous schemes and DSFTL when the data in a data block are written to a log block. We assumed that the data block is already filled with data. Furthermore, the write pattern is the sequence of 0, 4, 8, 12, 16, 20, 13, and 17, and we assumed that there are three log blocks. In BAST, *data 0*, *data 4*, and *data 8* are written to different log blocks, as one data block is connected to just one log block. When *data 12*, *data 16*, and *data 20* are written, the previous log blocks are merged with each data block, as there is no log block connected to the new data blocks. Therefore, there are three operations of merge in this example in the case of BAST. In the case of FAST, all the data are written to Log block 1 and Log block 2, since the data block is shared by all the log blocks. In this example, a merge does not occur. Thus, it seems to be better than BAST. However, because Log block 1 is connected to four data blocks (data blocks 1, 2, 3, and 4), a high-priced operation of merge is executed when Log block 1 is selected by the victim block.

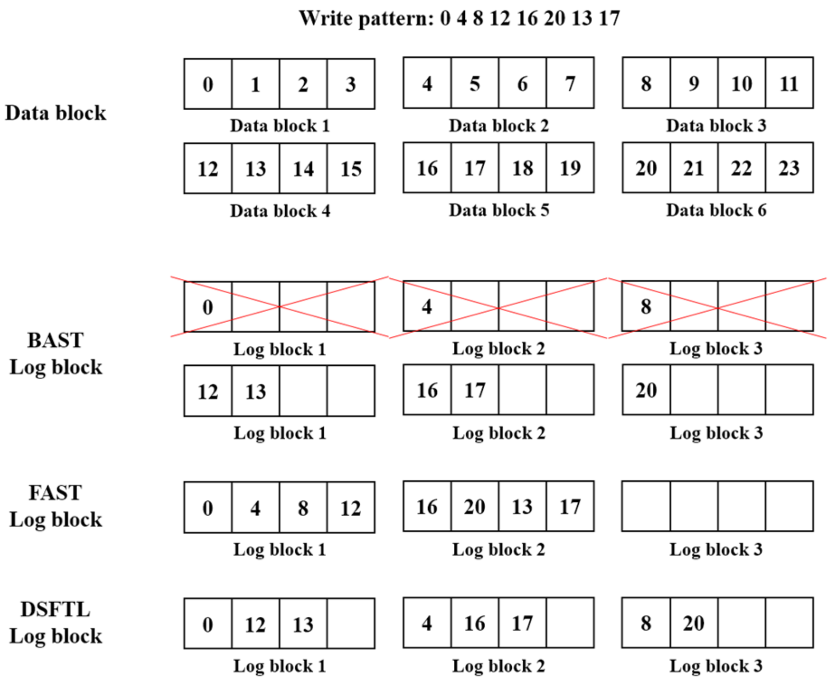

Figure 6. When data in a data block is written to a log block.

In the case of DSFTL, *data 0*, *data 4*, and *data 8* are written to different log blocks because of the increased space availability. Moreover, *data 12*, *data 16*, and *data 20* are written to each log block. At each stage, DSFTL finds a log block that has less written data since the data are written to the log block for the first time. Thus, the data of different data blocks are distributed evenly. When *data 13* and *data 17* are written, the data from the data block is written to the log blocks more than once. In this case, the data are written to the log block that has already been connected. Thus, *data 13* is written to Log block 1, and *data 17* is written to Log block 2. Because DSFTL has this feature, it not just decreases the number of operations of merge, but also decreases the number of high-priced operations of full merge.

There are two advantages in DSFTL. First, the space availability of log blocks is increased, as the new data of a data block are written to a log block with less written data. Thus, the number of operations of merge needed is reduced. Second, the availability between the data block and the log block is reduced because the data block is connected to just one log block. Therefore, DSFTL can avoid the high-priced operation of full merge. Consequently, by adjusting the availability between a data block and log block, we can decrease the garbage collection overhead.

5.4. Increasing Space Availability of a Data Block

As noted in the Motivation section, a data block and a log block can be merged when a new log block is assigned. Irrespective of the space availability of the data block and the log block, these are merged and erased. Because a flash memory has a limited number of erase operation, the lifetime of a flash memory is shortened if the blocks to be merged have many free pages. The space availability of the log block can be increased by increasing the availability between the log block and the data block.

Furthermore, it is an important thing to solve this problem, since there is no way to increase the space availability of the data block.

Algorithm 3 shows the merging of a log block. When a log block is merged, DSFTL operates four things to increase the space availability of the data block. Firstly, if a log block is selected as a victim block, DSFTL checks this log block. In this log block, our scheme finds a data block with less written data. We think that the data block with less written data in the victim block is likely to have low space availability. Secondly, DSFTL finds a log block whose availability with the data block is the lowest. This log block will save the data of the data block that had less written data written in the victim block. Thirdly, DSFTL copies the data in the victim block with the found log block. It is possible to prevent the data block with low space availability from being merged. Finally, DSFTL invalidates the data in the victim log block and considers the copied data valid. As a result, the life time of flash memory is extended, as the data block with low space availability is not merged.

Algorithm 3: When a log block is merged

```
  begin
1 if a log block is selected as a victim block
2     find the data block that data is less written in this log block
3     find the log block that availability is the lowest
4     copy data in the victim log block with the found log block
5     invalidate data in the victim log block
6 end if
```

Figure 7 shows how to increase the space availability of the data block in DSFTL. We assumed that not all the data blocks are filled with data. For example, there is just *data 0* in Data block 1. In this case, the space availability is 0.25. We also assumed that the data are written according to the availability of DSFTL. Log block 1 is connected with two data blocks that include Data block 1 and Data block 4. When Log block 1 is selected as the victim block, DSFTL finds a data block with less written data in Log block 1. In this case, while Data block 1 uses one page (*data 0*), Data block 4 uses three pages (*data 12*, *data 13*, and *data 14*). Therefore, the data of Data block 1 written in the victim block will be copied later. Furthermore, DSFTL finds a log block with the lowest availability with the data block except for the victim block. While the availability of Log block 2 is two (Data block 2 and Data block 5), that of Log block 3 is just one (Data block 3). As Log block 3 has the lowest availability, *data 0* of the victim block is copied to Log block 3, and then *data 0* in the victim block is considered invalid.

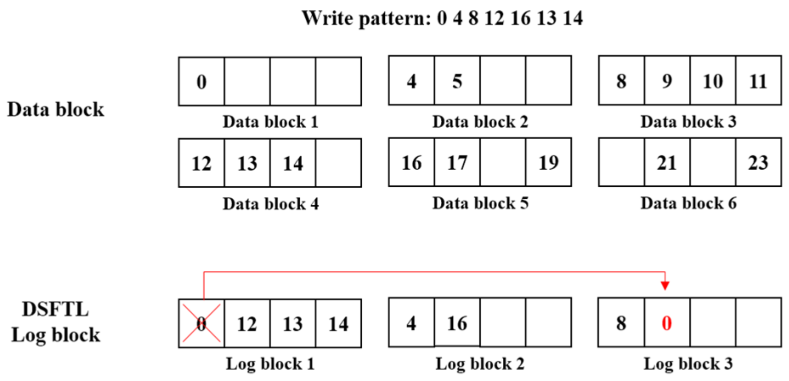

Figure 7. When a log block is merged.

To prevent a data block with low space availability from being merged, DSFTL copies the data of the data block with less data written in the victim block to another log block that has the lowest availability between the data block and the log block. Thus, the data block with low space availability is not merged. As a result, we can decrease the garbage collection overhead.

6. Evaluation

In this section, we evaluated performance of our proposed scheme by running experiments with real world workload traces, which we instrumented on a 100 GB multi-channel SSD and through mathematical models that we used to compare DSFTL with previous FTLs. We implemented DSFTL in conjunction with many hybrid mapping schemes for simulated experimental evaluation comparisons.

6.1. Experiment Setup

We implemented our DSFTL scheme onto a trace-driven EagleTree simulator [25] because it can simulate multi-channel SSDs [26] and does not only simulate SSDs but also the OS and applications that utilize it. EagleTree allows for various configurations of specific flash memory chip categories from Single-Level Cell (SLC) to Multi-Level cell (MLC) [27] configurations while supporting it for advanced commands. We configured a 100 GB multi-channel SSD to allow for parallelism both internal and external to improve the I/O throughput and system availability. Table 1 shows our full 6-channel SSD configuration parameters from an 8 KB page size up to the number of chips/planes per channel. Both approaches were implemented on the same SSD, which included many FTLs and DSFTLs.

Table 1. SSD configuration parameters.

Parameter	Value (Fixed)-Varied
Page Size (KB)	8
Pages per Block	1024
Blocks per Plane	1024
Plane per Die	1
Die per Chip	2
Chip per Channel	1
Channel number	6
Erase time	1500 µs
Program time	800 µs
Buffer service time	1000 ns

6.2. Workloads

Our experiments were conducted out using realistic workloads that included Financial 1 (uniform random write requests), Financial 2 (small random and large random write requests), MSNSF (uniform small sequential and random write requests), and RADIUS (mixed small and large sequential write requests) traces that were collected from various storage and image/video files, and they constituted various request sizes and read/write compositions, which are described in Table 2. These multimedia traces were retrieved from the UMASS Trace Repository [28] and the SNIA IOTTA Repository [29]. We used a set of traces that reflected the workloads listed in Table 2 in order to study the number of counts of erase and the execution time. These workloads came from the real workloads released by the International Network Storage Industry Association (SNIA) and Microsoft.

Table 2. Characteristic of workloads.

Traces	Financial 1	Financial 2	MSNFS	Radius
Read Ratio	0.41	0.23	0.85	0.09
Write Ratio	0.59	0.87	0.15	0.91
Ave Read size (KB)	2.5	18.6	14.8	7.1
Ave Write size (KB)	3.1	21.4	10.6	8.4

6.3. Experimental Results

In our experiments, we evaluated the counts of erase and the categories of operations of merge to compare DSFTL performance with BAST [9], FAST [10], LAST [14], and MAST [15]. In the case of the count of erase, we further tested the impact of the log blocks availability to both systems by increasing the number of log blocks from 4 up to 256 for each configuration. Figure 8 shows the comparison with Financial 1, Financial 2, MSNFS, and the Radius Traces. In this figure, DSFTL reduced count of erase, as the number of log blocks were increased. This is because DSFTL can increase the space availability of a log block and a data block. In addition, DSFTL can avoid a high-priced operation of full merge, since we adjusted the availability between a data block and a log block dynamically.

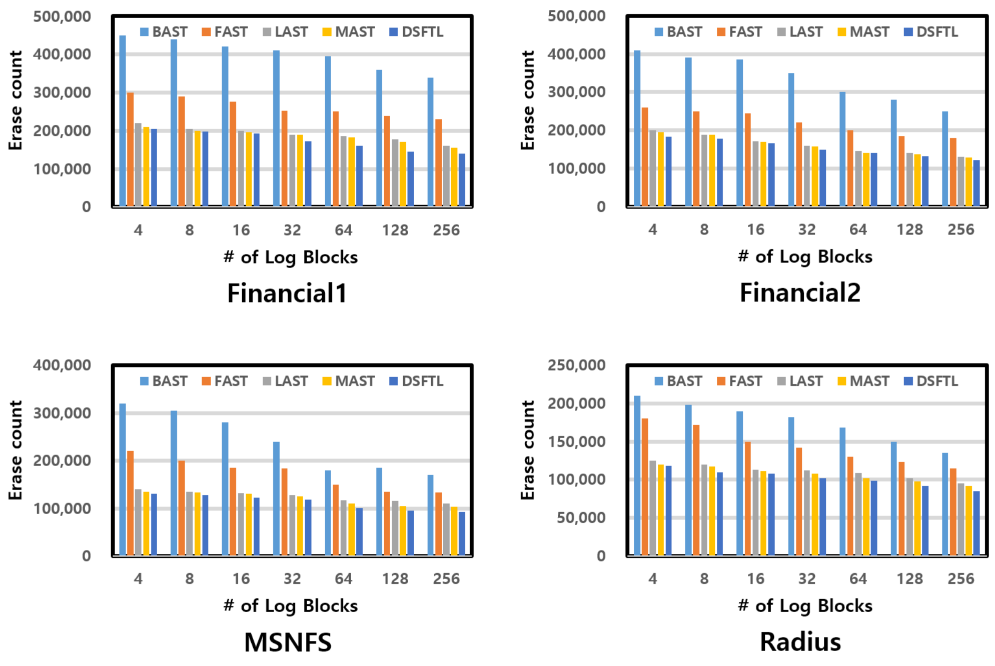

Figure 8. The number of counts of erase with four traces.

The uniform random write requests of the Financial 1 and the Financial 2 traces proved to be costly for the BAST, which is witnessed from Figure 8. The number of counts of erase in the BAST is over 400,000 in the Financial 1 and the Financial 2 traces. On the other hand, DSFTL decreased the number of counts of erase under 200,000 in two traces. This is because operation of write was placed into a dedicated block that caused operations of merge and led to the non-improvement with the BAST performance even as the number of blocks increased, and it just improved from 64 log blocks where we saw an improvement in its space availability. On the other hand, FAST, MAST, and LAST showed better reduction in space availability, as it could append operation of write into the end sector of a log block, which thereby resulted in lesser operations of write compared to the BAST. On the contrary, DSFTL reduced the operations of write by adjusting the availability between a data block and a log block, which consequently avoided a high-priced operation of full merge. This in turn reduced the number of operations of erase.

With MSNFS traces, we realized that DSFTL still outperformed the other FTLs, and it showed a very low count of erase. This is because the BAST performance remained identical regardless of the varied block numbers, whereas, with other FTLs, the sectors in a victim block have more chances to be invalidated, which resulted in the corresponding data blocks being less likely to be merged. On the contrary, DSFTL reduced the number of counts of erase, since the number of log blocks were increased. This is because the operations of erase and the operations of full merge decreased the garbage collection overhead. DSFTL assigned a lot more log blocks, which thereby increased the space availability of the data blocks more than its counterparts. In all traces, DSFTL avoided over 1000 counts of erase compared to the MAST.

Even though previous hybrid mappings reached the same level of reduction in the counts of erase between 16 and 32 number of log blocks due to BAST's intelligent operation of switch merge, which reduced the count of erase in the small random overwrites, we witnessed a continuation of DSFTL outperforming other FTLs in radius traces. We further checked the efficiency of DSFTL against large sequential writes by counting the number of operations of switch merge and comparing them with those from other hybrid mappings. Consequently, we decreased the number of counts of erase for all traces by increasing the space availability of a log block and a data block while avoiding high-priced operations of full merge in order to improve the system performance and the lifespan. To accomplish this, we adjusted the availability between a log block and a data block while avoiding operations of merge, which were previously discussed.

Figure 9 shows the ratio merge operation between our proposed DSFTL, BAST, FAST, LAST, and MAST FTLs with four traces. We realized that our DSFTL, which employed MAST architecture for RW log blocks and MAST FTL, had the lowest rate of operations of full merge. Furthermore, DSFTL also realized an increase in operations of switch merge than the rest, as it used many SW log blocks.

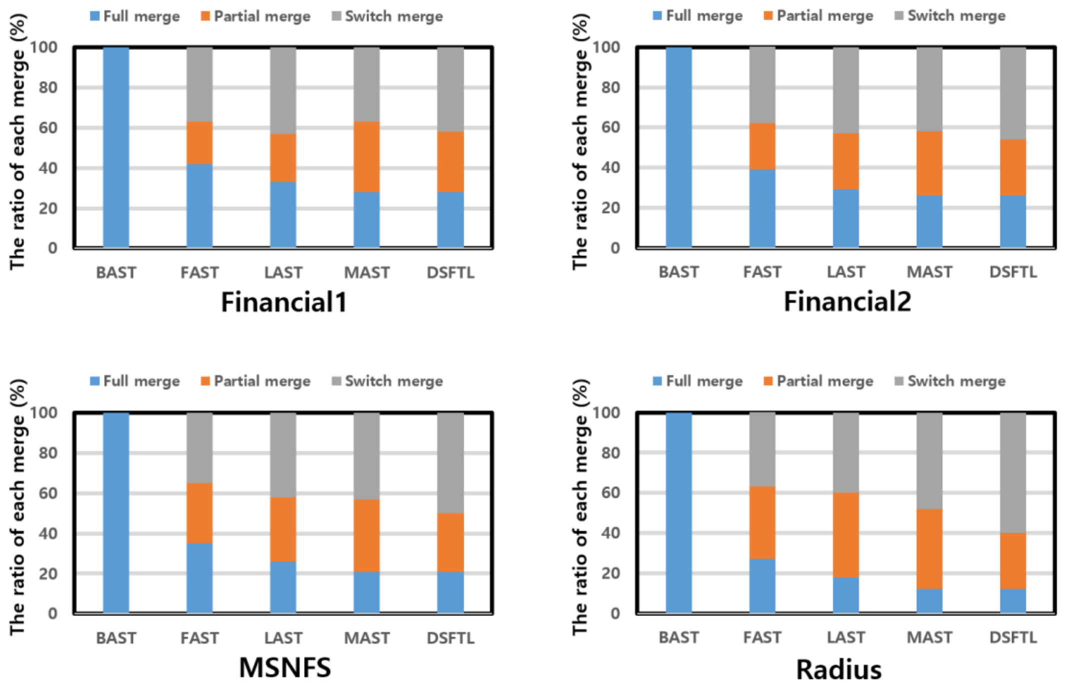

Figure 9. The ratio of each operations of merge with four traces.

In the case of an operation of switch merge, when data with a page offset of zero is written in an SW log block, DSFTL can perform each operation with many SW log blocks. Therefore, many SW log blocks increase the probability of operation of switch merge to occur. On the other hand, in other FTLs, data in an existing SW log block conducts operation of partial merge when new data are written. As we can see in Figure 9, DSFTL incurred more operations of switch merge in all traces cases, since it avoids operations of full merge and partial merge by using MAST architecture and many SW log blocks in order to decrease the garbage collection overhead. DSFTL performs the operations of switch merge more than 2% and operation of partial merge less than 2% compared to MAST.

For BAST, all of the operations of merge are conducted by the operation of full merge because of block thrashing, while, with FAST, which uses M:N availability between a log block and a data block, the victim block incurred operations of switch merge and partial merge. Most of the operations of merge are conducted by operations of switch merge and partial merge in the case of LAST, even though it still experiences a lot of operations of full merge. In MAST, we witnessed a decrease in the operations of full merge compared to the other FTLs. Compared to DSFTL, there are many operations of partial merge in MAST because it uses the FAST architecture for SW log blocks. For DSFTL, operations of switch merge are increased

7. Conclusions

In the hybrid FTL, there are many factors that increase the garbage collection overhead. Firstly, in the case of operation of merge, an operation of full merge is more high-priced than operations of switch merge and partial merge. In addition, since an operation of partial merge should copy pages in comparison with an operation of switch merge, it is quite beneficial to execute an operation of switch merge instead of an operation of partial merge. Secondly, availability between a data block and a log block has a direct effect on the performance of a flash memory. Lastly, the space availability of the data block can directly increase the garbage collection overhead. Therefore, we proposed DSFTL approach that dynamically considers all the above-mentioned concerns for a system in this study. Our proposed system induces more operations of switch merge than operations of full merge and partial merge. In addition, by adjusting the availability between a log block and a data block, we can avoid high-priced operations of full merge and the space availability of log block is increased. Moreover, DSFTL makes the space availability of a data block increase by avoiding operation of merge. The experimental results showed that our proposed scheme decreases the count of erase better than traditional block mapping FTLs.

However, there are some limitations in our algorithm. First, if workload includes many random write requests, some operations of merge will be occurred in DSFTL since we use some SW log blocks instead of RW log blocks. On the other hand, our scheme shows good performance in sequential write requests. Second, compared to page mapping scheme, DSFTL cannot avoid operation of merge since our scheme is a hybrid mapping scheme. However, DSFTL solved problems of a page mapping scheme.

In the future, we will try to solve these limitations. In addition, we plan to further study a system that applies machine learning capabilities to decrease the garbage collection overhead.

Author Contributions: Investigation, S.-J.C.; Methodology, T.-S.C., J.-Y.P., and M.A.; Software, R.M. The authors equally contributed in the work presented in the paper. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: This research was partially supported by the Basic Science Research Program through the NRF, Korea funded by the Ministry of Education (2019R1F1A1058548) and the Ajou University research fund.

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

References

1. Kinam, K. *Symposium on VLSI-TSA Technology*; IEEE: Piscataway, NJ, USA, 2008; pp. 5–9.
2. Gray, J.; Fitzgerald, B. Flash Disk Opportunity for Server-Applications. Available online: http://www.research.microsoft.com/~{}gray (accessed on 10 January 2020).
3. Mativenga, R.; Paik, J.-Y.; Kim, Y.; Lee, J.; Chung, T.-S. RFTL: Improving performance of selective caching-based page-level FTL through replication. *Clust. Comput. J.* **2019**, *22*, 25–41. [CrossRef]
4. Samsung Electronics. *Nand Flash Memory & Smartmedia Data Book*; Samsung Electronics: Suwon, Korea, 2007.
5. Amir, B. Flash File System Optimized for Page-mode Flash Technologies. U.S. Patent No. 5,937,425, 10 August 1999.
6. Chung, T.-S.; Park, D.-J.; Park, S.; Lee, D.-H.; Lee, S.-W.; Song, H.-J. A survey of flash translation layer. *J. Syst. Arch.* **2009**, *55*, 332–343. [CrossRef]
7. Amir, B. Flash File System. U.S. Patent No. 5,404,485, 4 April 1995.
8. Gupta, A.; Kim, Y.; Urgaonkar, B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. *ACM* **2009**, *44*, 229–240.
9. Kim, J.; Kim, J.M.; Noh, S.H.; Min, S.L.; Cho, Y. A space-efficient flash translation layer for CompactFlash systems. *IEEE Trans. Consum. Electron.* **2002**, *48*, 366–375.
10. Lee, S.-W.; Park, D.J.; Chung, T.S.; Lee, D.H.; Park, S.; Song, H.J. A log buffer-based flash translation layer using fully-associative sector translation. *ACM Trans. Embed. Comput. Syst.* **2007**, *6*, 18. [CrossRef]
11. Kang, J.; Jo, H.; Kim, J.-S.; Lee, J. A superblock-based flash translation layer for NAND flash memory. In Proceedings of the 6th ACM & IEEE International Conference on Embedded Software, Seoul, Korea, 22–25 October 2006.

12. Cho, H.; Shin, D.; Eom, Y.I. KAST: K-associative sector translation for NAND flash memory in real-time systems. In Proceedings of the 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 20–24 April 2009.
13. Forouhar, P.; Safaei, F. DA-FTL: Dynamic associative flash translation layer. In Proceedings of the 2017 19th International Symposium on Computer Architecture and Digital Systems (CADS), Kish Island, Iran, 21–22 December 2017.
14. Lee, S.; Shin, D.; Kim, Y.-J.; Kim, J. LAST: Locality-aware sector translation for NAND flash memory-based storage systems. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 36–42. [CrossRef]
15. Kim, J.; Kang, D.H.; Ha, B.; Cho, H.; Eom, Y.I. MAST: Multi-level associated sector translation for NAND flash memory-based storage system. In *Computer Science and Its Applications*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 817–822.
16. Kim, J.K.; Lee, H.G.; Choi, S.; Bahng, K. A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems. In Proceedings of the 8th ACM International Conference on Embedded Software, Atlanta, GA, USA, 19–24 October 2008.
17. Wei, Q.; Zeng, L.; Chen, J.; Chen, C. A popularity-aware buffer management to improve buffer hit ratio and write sequentiality for solid-state drive. *IEEE Trans. Magn.* **2013**, *49*, 2786–2793. [CrossRef]
18. Liu, D.; Wang, T.; Wang, Y.; Qin, Z.; Shao, Z. A block-level flash memory management scheme for reducing write activities in PCM-based embedded systems. In Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, Dresden, Germany, 12–16 March 2012.
19. Kwon, S.J. Non-volatile translation layer for PCM + NAND in wearable devices. *IEEE Trans. Consum. Electron.* **2017**, *63*, 483–489. [CrossRef]
20. Tal, A. *Two Technologies Compared: NOR vs. NAND*; M-Systems White Paper: Kfar Saba, Israel, 2003.
21. Sun, G.; Joo, Y.; Chen, Y.B.; Niu, D.; Xie, Y.; Chen, Y.R.; Li, H. A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. In *Emerging Memory Technologies*; Springer: New York, NY, USA, 2014; pp. 51–77.
22. Moshayedi, M.; Seyed, J.S. SLC-MLC Combination Flash Storage Device. U.S. Patent No. 8,825,941, 2 September 2014.
23. Harari, E.; Robert, D. Norman, and Sanjay Mehrotra. Flash Eeprom System. U.S. Patent No. 5,602,987, 11 February 1997.
24. Robinson, K.B.; Elbert, D.K.; Levy, M.A. Block-Erasable Non-Volatile Semiconductor Memory Which Tracks and Stores the Total Number of Write/Erase Cycles for Each Block. U.S. Patent No. 5,544,356, 6 August 1996.
25. Dayan, N.; Svendsen, M.K.; Bjorling, M.; Bonnet, P.; Bouganim, L. EagleTree: Exploring the design space of SSD-based algorithms. In Proceedings of the VLDB Endowment, Riva del Garda, Italy, 26 August 2013; pp. 1290–1293.
26. Park, S.-H.; Ha, S.-W.; Bang, K.; Chung, E.-Y. Design and analysis of flash translation layers for multi-channel NAND flash-based storage devices. *IEEE Trans. Consum. Electron.* **2009**, *55*, 1392–1400. [CrossRef]
27. Chang, Y.-H.; Kuo, T.-W. A management strategy for the reliability and performance improvement of MLC-based flash-memory storage systems. *IEEE Trans. Comput.* **2011**, *60*, 305–320. [CrossRef]
28. UMASS Trace Repository. Available online: http://traces.cs.umass.edu/ (accessed on 10 January 2020).
29. SNIA IOTTA Repository. Available online: http://iotta.snia.org/ (accessed on 10 January 2020).

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).