IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 23, 2021, accepted May 5, 2021, date of publication May 12, 2021, date of current version May 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3079520

Internal Task-Aware Command Scheduling to Improve Read Performance of Embedded Flash Storage Systems

GYEONGYONG LEE[1], JAEWOOK KWAK[1], JOONYONG JEONG[1], DAEYONG LEE[1], MOONSEOK JANG[1], (Student Member, IEEE), JUNGWOOK CHOI[1], AND YONG HO SONG[1,2], (Member, IEEE)

[1]Department of Electronics and Computer Engineering, Hanyang University, Seoul 04763, South Korea
[2]Samsung Electronics Company Ltd., Hwaseong 18448, South Korea

Corresponding author: Yong Ho Song (yhsong@hanyang.ac.kr)

This work was supported by the Ministry of Trade, Industry and Energy/Korea Evaluation Institute of Industrial Technology (MOTIE/KEIT) through the Research and Development Program (Developing Processor-Memory-Storage Integrated Architecture for Low-Power, High-Performance Big Data Servers) under Grant 10077609.

ABSTRACT Many NAND flash storage systems access flash memories by generating flash commands to process input/output (I/O) requests from a host system. Because the order in which flash commands are processed affects the I/O performance, command scheduling has been performed in previous studies to prioritize flash read commands for improving the read performance. However, in addition to the flash commands for accessing user data requested by the host, flash commands for internal tasks that improve the I/O performance and operation efficiency of the flash storage are issued. Particularly in embedded flash storage, the flash commands by the map cache management have a significant influence on the latency of I/O requests. The processing time of the I/O request depends on the execution time of the flash commands for mapping information and the flash commands for user data. In this paper, we propose command scheduling to improve read performance to address the occurrence of map flash commands. Priority is given to the flash read and program command from the read request, and among these commands, the flash command originating from the read request, which has a shorter processing time, is executed first. Consequently, the waiting time of the flash command is reduced, thereby improving the read latency. Experiments conducted with real workloads show that the proposed scheduling scheme reduces the average read latency by up to 51% compared with the existing scheduling scheme and demonstrates an effective performance improvement in a small map cache.

INDEX TERMS Command scheduling, map cache, address translation, flash translation layer, NAND flash storage, NAND flash memory.

I. INTRODUCTION

An input/output (I/O) request sent from the host system to the storage system is translated into flash commands to be processed in the NAND flash storage system. The I/O request usually accesses user data of the storage using logical block addressing, but the storage data is stored in flash memories. Therefore, it is necessary to know the physical address of the flash memory corresponding to the logical address. The flash translation layer (FTL), which is a software layer in the flash storage, has an address translation function [1], [2]. It obtains a physical address from a logical address and generates a flash command to read or program a flash page from the physical address.

Research has been conducted to improve the I/O performance of flash storage systems through flash command scheduling. Because an I/O request is completed when all translated flash commands are processed, the I/O performance depends on which flash command is prioritized. In particular, the host requires synchronous read requests to be processed before asynchronous write requests [3]; therefore, a command scheduling that prioritizes flash read commands has been performed in many studies [4]–[6]. In addition, because the average latency can be reduced if an

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Ching Ying.

I/O request with a smaller size is processed first [7], the flash read commands should have different priorities according to the size of the read request from which each command originates. Consequently, in addition to prioritizing the flash read command, a scheme has been proposed to schedule the flash commands based on the original request size of each flash command [4].

The FTL generates not only direct flash commands to access user data requested by the host, but also indirect flash commands through internal tasks that improve the I/O performance and operation efficiency of the flash storage. The buffer management scheme temporarily stores I/O request data in a memory device with a shorter access time than the flash memory. Because the buffered write request data must eventually be issued to the flash memory, indirect flash commands may occur when processing an I/O request. Garbage collection (GC) and wear leveling (WL) make flash commands clean flash blocks that have a combination of valid and obsolete data. The flash commands by GC and WL can indirectly affect the processing time of requests because they can appear not only during storage idle time but also when processing I/O requests. However, in an embedded flash storage, the flash commands because of the map cache significantly influence the latency of I/O requests. To quickly find the address of the flash memory where user data are stored, a mapping table in RAM containing the entire mapping information is usually used. Embedded flash storage has a limited amount of RAM; therefore, it uses a map cache scheme that places only a part of the mapping table in RAM [8]–[11]. When mapping information for an address not existing in the map cache is required, a flash command is issued to fetch the information. Because the address translation procedure is essential for processing I/O requests, the flash command generated by the map cache should be considered in command scheduling.

When an I/O request arrives, the FTL searches the map cache before generating a flash command to access the user data. If there is no mapping information of the requested address, a map page containing the mapping information is read, and multiple mapping entries are added to the cache line of the map cache. When there is no space to add a new cache line, the existing cache line is evicted. In particular, a map page is programmed to update the mapping information of the dirty cache line in the flash memory. However, there is a case where the desired mapping entry is in the cache line but is not ready. If the map page read command generated by the previous I/O request has not been executed yet, the mapping entry is referenced only after the corresponding command is completed. In addition, the map page is composed of multiple cache lines; therefore, the mapping information of multiple cache lines can be updated together to reduce the number of flash commands generated by the dirty line eviction. To bring the mapping entries to the clean cache line to be secured at this time, the map page program command from the previous I/O request must be completed. In summary, to complete an I/O request, all *flash command sets* consisting of the flash commands for mapping information and the flash commands for user data must be executed. In particular, the map flash command is developed while the cache line is managed, so it can be shared among multiple I/O requests. Therefore, the existing scheme of scheduling flash commands based on their request size without considering the flash commands related to the I/O requests has a limitation in reducing the average latency.

We propose a command scheduling to improve the read performance of embedded flash storage systems based on the fact that the processing time of the I/O request is determined by the flash command sets. First, a priority is given to flash read and program commands from the read requests instead of any flash read commands so that the read requests are processed first according to the demand of the host. Second, to reduce the waiting time of the flash command, a flash command originating from an I/O request with a shorter processing time is prioritized. A command scheduling using both schemes reduces the average read latency by allowing shorter read requests to be processed first. In addition, among multiple I/O requests that share the flash command by the map cache, read requests with the least processing time are scheduled to be prioritized, further reducing the read latency.

The proposed command scheduling was evaluated using a trace-driven simulator to model the map cache flash storage. Experimental results show that the proposed scheme reduces the average read latency by up to 51% compared with the existing scheduling. In addition, the significant effect of command scheduling in the small map cache was demonstrated.

The remainder of this paper is organized as follows. Section II presents the background of map cache flash storage for understanding this study, and related studies are briefly introduced in Section III. Section IV provides the motives behind the proposed command scheduling scheme. In Section V, a method for performing the scheduling scheme to reduce the read latency is described. Section VI details the experimental environment and results of the proposed scheme. We conclude this study in Section VII.

II. BACKGROUND

A. ADDRESS TRANSLATION IN FLASH STORAGE WITH MAP CACHE

In a NAND flash storage with a map cache, the entire mapping table is managed in the NAND flash memory. Flash blocks consist of map blocks containing a mapping table and data blocks containing user data. Some of the mapping tables are cached in the RAM for faster address translation. DFTL, the first map cache management scheme proposed, caches mapping information in RAM in a mapping entry unit [8]. Because this method is not efficient when addresses are requested sequentially, a scheme for fetching mapping information into multiple mapping entries simultaneously has been devised. The dual granularity cache management scheme reads a map page and adds multiple mapping entries to the cache line of the cached line table (CLT) [11].

The location information of the map page required for the caching requires a small memory space; therefore, a global translation directory (GTD) containing the entire location information exists in RAM.

Fig. 1 presents how the requested logical page number (LPN) 11, which is a cache miss in the map cache with four mapping entries per map page, is processed (1). As the CLT is full and the victim line is dirty, the map page to which the entries of the line belong must be updated. The map page corresponding to victim's starting LPN (SLPN) 4 is virtual page number (VPN) 1, and therefore, physical page number (PPN) 14 is read by referring to GTD (2). After the mapping information is updated, it is programmed in a new map page to be reflected in the flash again (3). In the map page containing the mapping entry of the originally requested address, two entries, which is the size of the cache line, are fetched into the CLT (4). The data block is accessed with information that LPN 11 is mapped to PPN 7 (5).

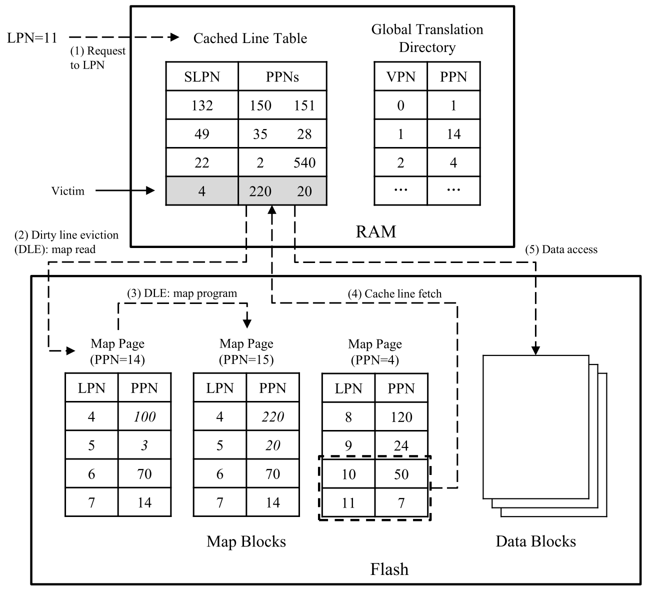

FIGURE 1. Architecture of the map cache fetching mapping entries into a cache line.

Fig. 2 depicts the flash commands generated in the aforementioned process in a chronological order. When reading or writing a requested logical page, the data page is read or programmed after having time to access the map pages in common. This series of flash commands is called a *flash command set*. An I/O request consists of one or more logical pages according to its size, resulting in one or more flash command sets. However, herein, in some cases an I/O request is assumed to be a single page-sized request for simplicity; Fig. 2 is such a case.

B. I/O REQUEST PROCESSING FLOW

The host using a NAND flash memory as a secondary storage device makes I/O requests for user data. I/O requests are usually divided into read I/O requests (R) and write I/O requests (W). The requests are sent to the flash storage device according to the host interface specification and queued at the I/O request queue. The order in which requests at the queue are processed depends on the I/O scheduling policy. Fig. 3 illustrates the processing of I/O requests in a flash storage device.

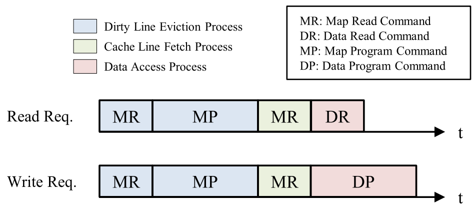

FIGURE 2. There are flash commands that must be processed prior to data access when the largest cost of cache miss occurs during the address translation.

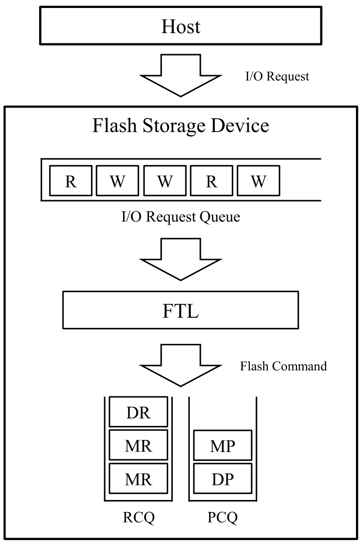

FIGURE 3. Processing I/O requests sent by the host in the flash storage device. RCQ: read command queue, PCQ: program command queue.

To process the R or W in the I/O request queue, access to the flash memory is required. The target that an I/O request reads or writes is a logical address, and the flash memory corresponds to a physical address; therefore, an appropriate address translation must be performed. The FTL translates I/O requests into flash commands. If there is a map cache, a flash command directed to the map block or data block is generated. Flash storage is usually composed of multiple flash chips in a multi-channel and multi-way architecture; thus, a flash command queue exists for each chip. The generated command is inserted into the appropriate flash command queue according to its physical address.

A flash command queue is generally divided into read command queue (RCQ) and program command queue (PCQ). In a storage without a map cache, the read command is prioritized to prevent the R from being blocked by the W. Thus, the read performance is improved by a command scheduling that gives priority to RCQ over PCQ until the RCQ is empty. However, if there is a map cache, the pattern of flash commands generated from the I/O request changes, thereby necessitating different command scheduling methods.

III. RELATED WORK

A. ADDRESS TRANSLATION SCHEMES FOR RAM-LIMITED FLASH STORAGE

In a NAND flash storage, page-level mapping is a naive and intuitive address translation scheme designed for a flash operation unit, a page. However, in an environment with limited RAM, its application is limited because of the excessive size of the mapping table. Therefore, block-level mapping and hybrid mapping that can be applied even in smaller RAMs have emerged [12]–[15], but their performance is insufficient compared with that of page-level mapping. To overcome this, DFTL, a map cache based on page-level mapping, has been proposed [8]. DFTL, which manages mapping information in a mapping entry unit, is a design that exploits the temporal locality of the workload. Frequently accessed entries continue to reside in the map cache owing to the least recently used replacement algorithm, saving the cost of fetching mapping entries from the flash. However, because it is a design that cannot exploit the spatial locality of the workload, algorithms have been proposed to compensate for this [9], [10]. They manage the map cache in a map page unit; thus, in the case of consecutively accessed addresses, only one flash read operation needs to be performed without having to fetch the mapping entries one by one. However, some of the fetched mapping entries may not be referenced until they are evicted from the map cache, which is also known as cache pollution. Dual granularity (DG) manages a map cache in a cache line unit to reduce cache pollution [11]. The cache line consists of a plurality of mapping entries and is set to a smaller size than the map page.

B. IMPROVING PERFORMANCE OF FLASH STORAGE WITH MAP CACHE

Studies have been conducted to optimize the performance of flash storage away from the perspective of managing the map cache. MAP+ [16] employs an I/O scheduling scheme that distinguishes the map cache hit/miss of I/O requests and prioritizes hit requests. Missing requests are grouped together with related addresses to be processed as a batch, and the batches are scheduled to further reduce the I/O latency. Parallel-DFTL [17] improves DFTL so that NAND flash parallelism can also be utilized in address translation operations. Because data access and address translation operations are coupled in the existing method, different queues are introduced to separate them. Consequently, both operations can simultaneously access the flash.

One of the bottlenecks caused by the introduction of the map cache is that address translation involves flash operations. To substantially reduce the translation time, research has been conducted to use resources other than RAM in a flash storage device. HPB [18] uses a part of the host system memory as a map cache to improve random read performance in a mobile environment with only SRAM. In [19], the host memory buffer, which is one of the characteristics of NVMe, is utilized to improve the performance of the DRAM-less SSD, and a map cache is concluded as an optimal usage among several other methods. HAT [20] stores the entire mapping table in a phase-change memory (PCM) to eliminate the time required to access the flash for fetching mapping information. However, the access speed of PCM is lower than that of RAM; therefore, frequently referenced mapping information is managed in the existing RAM.

C. I/O SCHEDULING POLICIES FOR FLASH STORAGE

I/O requests created by a host system to access data go through several queues while they are issued to and processed by the flash storage system. Because scheduling in the queue greatly affects I/O performance, studies on I/O scheduling have been conducted on the host and storage sides. Host applications typically issue read requests in a synchronous manner; therefore, Individual Read Bundled Write FIFO with Read Preference (IRBW-FIFO-RP) gives read requests a higher priority over write requests, and write requests are collected in bundles of appropriate size to be processed [3]. Similarly, algorithms have been developed to prioritize read requests and perform I/O scheduling considering the size of the requests and the GC state of the flash chips [7]. Storage-side I/O scheduling exploits the internal structure of a storage device. A scheduling algorithm for native command queuing has been proposed based on the fact that I/O service times vary according to the state of the data buffer [21]. A host interface I/O scheduler (HIOS) performs I/O scheduling to reduce the number of I/O requests that cannot meet the deadline owing to channel resource conflicts and GC overheads [22]. To exploit NAND flash parallelism, studies have been conducted to group flash commands generated from I/O requests and execute them out of order [4], [23]. Slacker has been proposed as a scheduler to reduce slack time between sub-requests of I/O requests [5]. These scheduling approaches have been proposed for general flash storage. In storage with a map cache, new flash commands to access mapping information are created; therefore, research focusing on this application is needed.

IV. MOTIVATION

The flash command scheduling method should depend on the presence of the map cache in a flash storage. In a full-map storage without a map cache, only flash commands to access data exist, whereas in a storage with a map cache, commands to access the map as well as data exist. Fig. 4(a) depicts the scheduling in the flash command queue of the map cache storage. The command in the queue is the first command in the flash command set generated from the I/O request, and the remaining blurry commands are sequentially generated and processed. In other words, the flash command is a part of the flash command set, and therefore, command scheduling should be based on the set instead of the command.

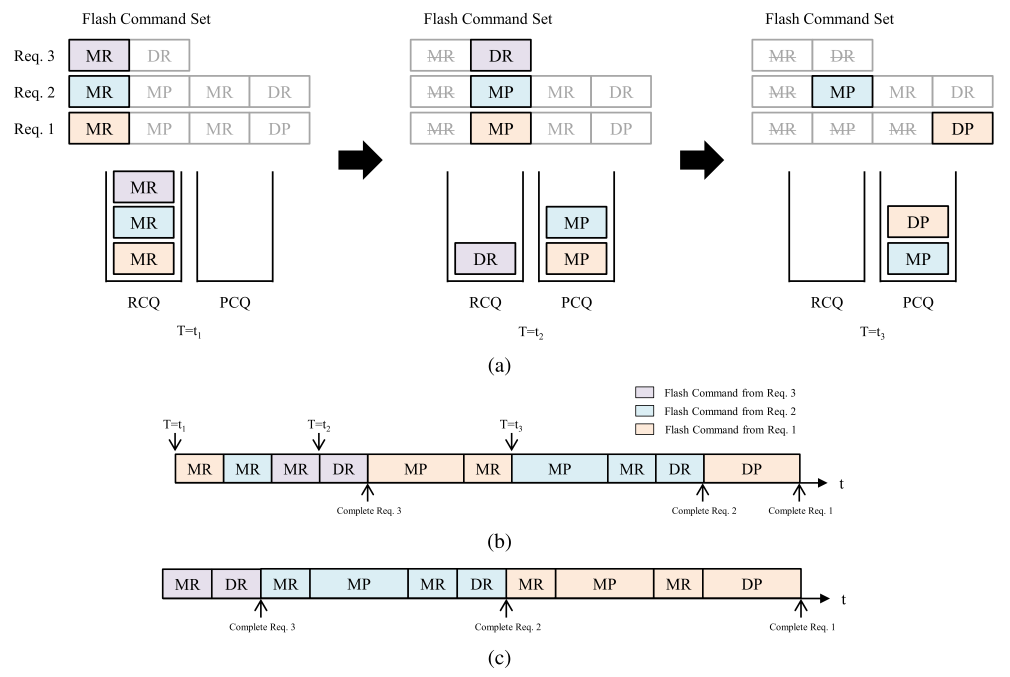

FIGURE 4. Read performance varies according to the flash command scheduling method. (a) Existing command scheduling for full-map flash storage. It manages the flash command queue by dividing it into RCQ and PCQ. (b) Command processing order in the existing command scheduling. (c) Command processing order in command scheduling proposed for map cache flash storage.

The existing command-based scheduling for a full-map storage divides the flash command queue into RCQ and PCQ to distinguish between flash read and program commands. In a full-map storage, flash read commands are generated only from read requests. The result of a scheduling that prioritizes the read commands to process read requests first is illustrated in Fig. 4(b). The first queued command is prioritized among the commands in the RCQ or PCQ; however, the commands in RCQ are always preferred to PCQ. Therefore, at T=t_2, MP of Req. 2 loses its turn to the MR, which follows MP in the flash command set of Req. 1. Although the read request is not completed, the command of the write request is processed. In addition, the flash command set of a request is not completed, but commands of other requests are alternately processed, resulting in unnecessary queue waiting time.

Fig. 4(c) depicts a scheduling based on the flash command set so that the read request is processed first and unnecessary queue waiting time is avoided. To reduce the average latency of a read request, based on the shortest-job-first scheduling [24], the flash command belonging to the set of read requests with a shorter processing time is prioritized. Therefore, the commands of Reqs. 2 and 3, which are read requests, are processed with priority over Req. 1, which is a write request, and the commands of Req. 3, which have a shorter set than Req. 2, are processed first. Consequently, the period for which the flash command waits in the queue is reduced, thus improving the read latency. Therefore, scheduling based on the flash command set is effective for the map cache storage.

V. EFFECTIVE COMMAND SCHEDULING FOR MAP CACHE STORAGE

In this section, we present a method of identifying flash commands related to read requests and reordering the flash commands so that requests with a shorter processing time are completed first to improve the read performance of the map cache flash storage. A flash command set is the basic unit for accessing a logical page. Because an I/O request reads or writes one or more logical pages, the command scheduling described in the following subsections is based on the flash command set.

A. READ REQUEST FIRST SCHEDULING

The flash storage usually schedules flash commands so that read requests can be processed first. Host applications generally make synchronous read requests to the storage [25]. The application is blocked until the data are received by the read request. In contrast, most write requests occur asynchronously [24]. In this case, the application performs its own work without waiting for the write request sent by itself to finish. Therefore, if read requests are processed quickly through command scheduling, the host's responsiveness increases.

We divide the flash command queue into a read request queue (RRQ), a write request queue (WRQ), and GC queues to propose a read request first (RRF) scheduling. An I/O request is translated into flash command sets in the FTL. Because both flash read commands and flash program commands can exist in the flash command set, it is difficult

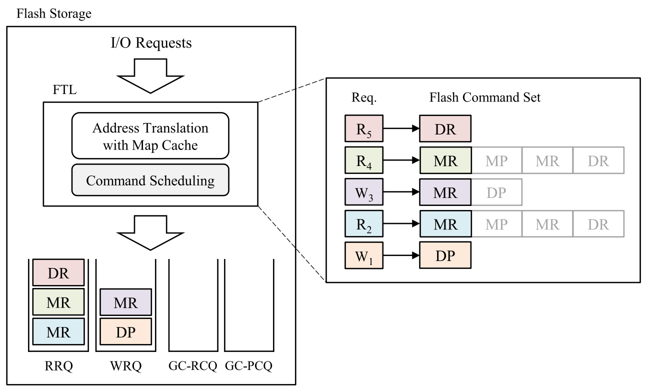

FIGURE 5. Flash command queue is divided into a read request queue (RRQ), a write request queue (WRQ), and GC command queues so that read requests are processed first.

to process read requests first if the flash command is queued based on its opcode. In Fig. 5, the flash read and program commands are queued together in the RRQ or WRQ. Of course, the data read command (DR) exists only in the RRQ, and the data program command (DP) only exists in the WRQ. To achieve RRF scheduling, the flash command in the RRQ is first processed until the RRQ is empty. WRQ starvation caused by priority scheduling is prevented by setting a deadline. The expiration time of the flash command in the WRQ is set to 5 seconds, which is the value for the write request in the deadline I/O scheduler, and the command is prioritized after this period.

Flash commands generated to process I/O requests, including the map access commands, are prioritized over flash commands issued to perform GC. When GC is triggered, flash read and program commands are generated to migrate a valid page of a victim block to a new block, and a flash erase command is generated after all migrations are completed. Moreover, when a data block is selected as a victim block, flash commands for updating the changed mapping information are also issued. Because responsiveness is not required for the GC operation, GC flash commands are queued into GC-RCQ or GC-PCQ based on their opcodes, and the erase commands are inserted into the GC-RCQ. Because the time required for the flash read operation is shorter than that of other operations, the flash read commands are prioritized [6]. When GC is triggered because of insufficient free blocks, the GC commands take precedence over the flash commands from I/O requests as an exception.

B. COMMAND SCHEDULING WITH FLASH COMMAND SET

Flash commands in the RRQ or WRQ can be further scheduled based on their I/O requests. I/O requests are translated to flash command sets in the FTL, and flash commands become the scheduling target while the set is being sequentially processed. Because all translated flash commands must be processed to complete an I/O request, command scheduling considering the request can improve I/O performance. If command scheduling is performed based on the size of the I/O request, the average latency of the request can be reduced [4]; however, even the flash command set must be considered for effective scheduling in the map cache storage. Therefore, it is necessary to examine the flash command set that must be processed to complete the I/O request.

1) IMPACT OF MAP CACHE OPTIMIZATION ON FLASH COMMAND SET

Owing to the map cache optimization, the flash command set of an I/O request can be composed of its own flash commands and the flash commands of the previous request. DFTL performs a *batch update* to reduce the number of flash programs that can occur when replacing mapping entries [8]. When a cache space to be secured by batch update is allocated to an I/O request in which a cache miss occurs, a flash command to fetch a mapping entry is created. However, this command can only be processed when the cache space is ready, so the command should wait for the dirty victim eviction procedure to complete. Therefore, the flash command set of the I/O request consists of *original flash commands*, which are flash commands created directly to process the request, and *delaying flash commands*, which are flash commands of the previous request that delay the execution of the original flash commands. *K-entry caching*, which compensates for the shortcomings of DFTL [11] is another case where original flash commands are delayed. If an I/O request is a cache hit to the cache line in which fetching the mapping entries is in progress, only a flash command to access user data is generated. Similarly, this command can be processed only after the cache line fetch process of the previous request is completed. The batch update can also be applied to the cache lines, causing the original commands that are delayed during the dirty line eviction process to still appear.

Fig. 6 presents the flash command sets of I/O requests that occur when the batch update is performed in the map cache to which a 2-entry caching is applied. The map page has eight mapping entries, and each request assumes that a cache miss occurs for the map cache in its initial state. The four requests are processed in order as follows. When the CLT is full, the dirty victim line is evicted for Req. 1, accompanied by the batch update. Consequently, SLPNs 8 and 10 belonging to the same map page are programmed together. Req. 2 is assigned a clean line, and the caching is enabled without any delay. In contrast, Req. 3 is assigned a clean line to be secured after the batch update; therefore, it must wait until the dirty line eviction of Req. 1 is finished. As the mapping entry required by Req. 4 is included in the cache line of Req. 1, Req. 4 can be processed after being delayed until the completion of the cache line fetch. Because the flash command set of Req. 1 is created by evicting the victim, it consists of only the original flash commands, and the flash command sets of the rest of requests include delaying flash commands because of the Req. 1 commands.

2) FLASH OPERATION TIME-BASED SCHEDULING

Flash operation time (FOT), as the actual size of an I/O request, is defined as the sum of the time periods spent

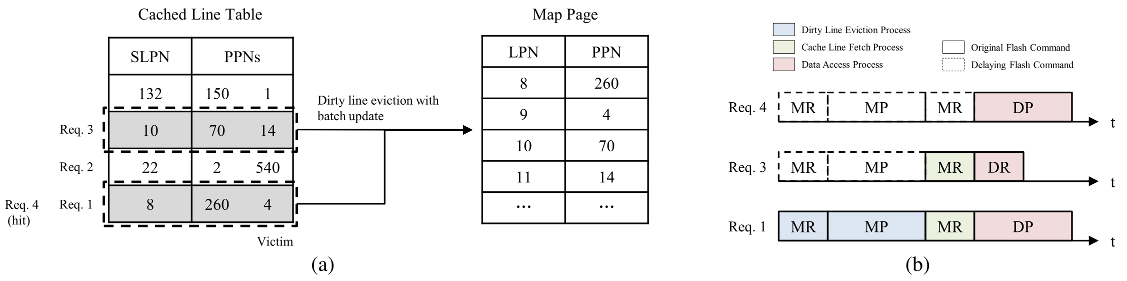

FIGURE 6. Scenario in which a delaying flash command occurs in a map cache with a 2-entry caching. (a) Situation where Reqs. 1–4 are processed sequentially when there are two dirty lines, including the victim in CLT. (b) Flash command set for Reqs. 3–4 can be executed after the delaying flash command originating from Req. 1 is completed.

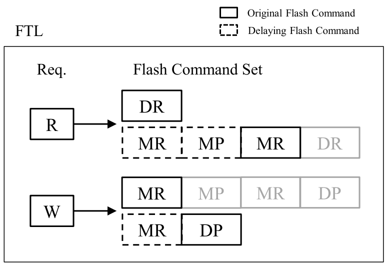

FIGURE 7. Owing to a characteristic of the map cache, I/O requests are completed only when all directly or indirectly related flash commands are processed.

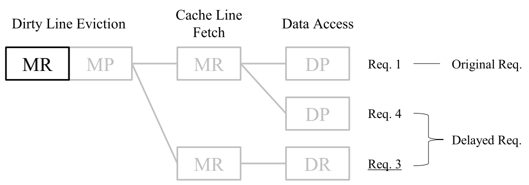

FIGURE 8. Representation of the delay relationship between the requests in the scenario of Fig. 6. MR, which is the original command for Req. 1, becomes the delaying command for Reqs. 3 and 4. Because the MR is included in each flash command set of delayed requests as well as in Req. 1, FOT-based scheduling should be performed using Req. 3, which has the smallest FOT value as a read request among all requests (though there is only one candidate).

processing all commands in the translated flash command sets. An I/O request is translated into one or more flash command sets depending on the number of logical pages accessed in the FTL. These command sets represent the characteristics of the request because they are the minimum commands that must be processed to complete it. In contrast, although GC flash commands can affect the latencies of requests, they are excluded from the definition of FOT because they are dedicated to overall storage management. The FOT is determined according to the number of flash command sets and the length of each set. Fig. 7 depicts an example of a read and write request accessing two logical pages, translated in the FTL. Between flash command sets caused by the cache hit, the number of flash commands to be processed is different when the mapping information can be referenced immediately (the top set of read request) or not (the bottom set of write request). Therefore, the FOT is defined as the sum of the processing times of the original flash commands and the delaying flash commands.

The FOT-based scheduling prioritizes the flash command originating from I/O requests with a shorter FOT. When command scheduling is performed based on the size of the request in the map cache storage, there is a limit to reducing the average latency because the map cache factor is not considered. Therefore, the average latency can be further reduced by scheduling the command based on the FOT calculated from the request size and the flash command set together. The FOT is determined based on the flash command sets generated when the I/O request is translated, and the value is stored in the flash command. Once generated, the command set occupies the allocated cache space until the mapping information fetch process is completed, so the determined FOT cannot be changed by a subsequent request. In addition, when there is a desired mapping entry in the cache, a data access command is created immediately by referencing the entry; therefore, even if a subsequent request replaces its mapping entry, the FOT is not affected. Command scheduling is performed based on the stored FOT value. Section V-D provides a detailed description of scheduling operations. Because FOT-based scheduling occurs in addition to RRF scheduling, the expiration time of the flash command in the RRQ is set to 500 ms, which is the value for the read request in the deadline I/O scheduler, to prevent starvation from the RRQ.

C. DELAYED REQUEST SCANNING

When the delaying flash command is executed, not only the request that creates the delaying command but also other requests are processed at the same time. Fig. 8 presents the delay relationship between the flash commands in Fig. 6, and Reqs. 3 and 4 are the delayed requests to the MR, the original flash command of Req. 1. Specifically, Req. 3 is a *DLE-delayed request* that is handled after the completion of the dirty line eviction (DLE) process, and Req. 4 is a *CLF-delayed request* that is handled after the completion of the cache line fetch (CLF) process. The flash commands of the DLE are delaying commands that affect all delayed requests, but the MR for the CLF of Req. 1 only affects Req. 4. After the DLE process is completed, Req. 3 is no longer influenced by the delaying command, and only the handling of the original commands remains.

Delayed request scanning (DRS) searches all I/O requests related to the delaying command and includes them in the target of FOT-based scheduling. In Fig. 8, if the scheduling is performed using only Req. 1, which is a write request, the MR is processed with a low priority according to the RRF scheduling. In contrast, when delayed requests are considered in the scheduling, we have Req. 3, which is the read request; therefore, the MR is prioritized to further reduce the read latency. Therefore, if there is at least one read request among the requests affected by the flash command in DRS, the command is inserted into the RRQ, otherwise, it is inserted into the WRQ. In Fig. 8, the MR and MP of the DLE are inserted into the RRQ, but MR for the CLF of Req. 1 is inserted into WRQ because only write requests are affected by the MR. The priority of a RRQ (or WRQ) flash command is determined according to the FOT-based scheduling performed with the request, which has a minimum FOT value as a read (or write) request among the command's original request and the delayed requests.

D. IMPLEMENTATION OF DRS

To implement the proposed command scheduling, attributes of the flash command are added, and RRQ and WRQ are managed as a queue sorted by priority based on these attributes. Because DRS is an FOT-based scheduling scheme that considers delayed requests on top of RRF scheduling, other scheduling schemes are not described separately.

1) ATTRIBUTES OF FLASH COMMAND

Request information (reqInfo) is added to the attributes of the flash command because it is necessary to know the I/O requests associated with a command when scheduling the commands. Request information is structured as an array because it needs to contain the information of the original request and delayed request of the command. As members of the structure, it has a *type* to distinguish between read and write requests, and a *flash operation time* (fot) for priority scheduling in the queue. In the case of FOT-based scheduling, information on delayed requests is not managed, and thus the request information is implemented without being in an array form.

The request information is an array of three sizes. Three request information (reqInfo[3]) are needed to express the original request, CLF-delayed requests, and DLE-delayed requests. ReqInfo[0] contains original request information. ReqInfo[1] contains a smaller value between the minimum FOT among CLF-delayed request FOTs and reqInfo[0].fot and type at that time. Finally, reqInfo[2] contains a smaller value between the minimum FOT among DLE-delayed request FOTs and reqInfo[1].fot and type at that time. In other words, the following equation holds for the fot:

$$reqInfo[n].fot = min(min(FOT_{req\_n_1}, FOT_{req\_n_2}, \cdots), reqInfo[n-1].fot) \quad (1)$$

for $n = 1, 2$. *req*_1 and *req*_2 denote CLF-delayed requests and DLE-delayed requests, respectively. However, when a read request is compared with a write request, the read request takes precedence and is stored in the request information, regardless of the FOT value.

2) DRS ALGORITHM

Algorithm 1 presents the process of inserting a flash command into the RRQ or WRQ. Because reqInfo[2] stores the information of the request with the minimum FOT among the original request and all delayed requests, the queue in which the flash command is to be inserted is based on the request type (lines 1–5). Of course, if there is any read request among the requests affected by the flash command, the type will be the Read, so the queue is set as RRQ. Because RRQ or WRQ is a sorted queue, the queue is traversed to insert the input command at the appropriate location (line 6). The queue is sorted in an ascending order based on the FOT value; therefore, the input command is inserted before the flash command with a larger FOT value than the input command (lines 7–10). If no such command is in the queue, the input command is inserted as the last element of the queue (lines 11–14). When inserting the commands into the sorted queue, a binary search is possible; however, it is expressed as a sequential search to focus on the comparison of the commands' FOT values.

Algorithm 1 RRQ/WRQ Management

```
Input: Flash command to be queued at RRQ/WRQ
Output: Sorted RRQ/WRQ according to priority
1:  if inCommand.reqInfo[2].type = Read then
2:      queue ← RRQ;
3:  else
4:      queue ← WRQ;
5:  end if
6:  for command in queue do
7:      if inCommand.reqInfo[2].fot <
8:        command.reqInfo[2].fot then
9:          insert inCommand before command;
10:         return queue;
11:     else
12:         if command = queue[last] then
13:             insert inCommand after command;
14:         end if
15:     end if
16: end for
17: return queue;
```

Algorithm 2 presents the process of updating the request information for the next flash command in the flash command set after a flash command is completed. If the completed command is MP, it means the DLE process is over, and therefore, the flash command set becomes irrelevant for DLE-delayed requests. Therefore, reqInfo[2] is updated with reqInfo[1], which contains the information of requests still related to the flash command set (lines 1–2). When the MR of the CLF

Algorithm 2 Request Information Update

Input: Completed flash command

Output: Updated request information for next flash command in the flash command set

```
1: if command is MP then
2:     command.reqInfo[2] ← command.reqInfo[1];
3: else
4:     if command is MR of Cache Line Fetch then
5:         command.reqInfo[2] ← command.reqInfo[0];
6:     end if
7: end if
8: return command.reqInfo;
```

process is completed, the flash command set no longer affects the delayed requests; therefore, the information of the original request is assigned to reqInfo[2] (lines 3–6). As reqInfo[2] is continuously updated in this manner, it can be used for a comparison of queue managements in Algorithm 1.

3) OVERHEAD OF DRS

Because DRS maintains sorted queues based on the reqInfo for the command scheduling, the overhead of managing the reqInfo and sorted queues becomes DRS overhead. First, the sorted queue also exists in other priority scheduling; therefore, if a priority of the flash command can be obtained in a constant time, additional overhead is not incurred. Second, because the reqInfo is a fixed-sized array with two member variables, the additional RAM requirement is constant. The overhead of initializing the reqInfo appears to be O(n) according to (1), but this is an expression of all the comparisons to find the final minimum value at once. In practice, a comparison is performed whenever there is a delay relationship with the previous flash command while processing an I/O request, so the initialization overhead can be regarded as a constant time. Because the time to access the reqInfo to which the value is assigned is constant, the additional time complexity of DRS as a priority scheduling method is O(1).

VI. EXPERIMENT

A. EVALUATION SETUP

To model the I/O request queue and flash command queue, and to check the effect of command scheduling, an in-house simulator was implemented based on FlashSim [26]. Based on DFTL supported by FlashSim, a map cache managed in a cache line unit was implemented. In addition, a batch update policy was applied to the map cache. Through trace-driven simulations, we conducted command scheduling experiments with several workloads.

Some of the embedded flash storage, such as UFS and flash cards, are configured as a single channel owing to cost limitations [27], [28]. In addition, the chip capacity has increased owing to an increase in bits per flash cell [29], and thus, the storage capacity can be fulfilled using a single way. We configured the storage with a single-channel and single-way architecture to evaluate the performance of low-end devices. The default capacity of the map cache was set to 1 kB, and the page size of the flash memory was set to 4 kB. The times required for flash read operation and program operation were 60 μs and 700 μs, respectively [30]. The default flash command queue depth was set to 256. The simulation was performed without pre-conditioning to experiment in an environment with minimal effect of GC. Moreover, because on-demand GC, which triggers GC whenever a free block is needed, was applied, GC flash commands are simply processed with the highest priority during the GC procedure. Therefore, a scheduling method for the purpose of optimizing GC is not considered in the proposed scheme.

We measured the latency of I/O requests to evaluate five types of command scheduling. Two existing scheduling schemes that do not consider the flash command set generated by the map cache and the three proposed scheduling schemes are compared.

1) *RCF* is a command scheduling to which a read command first (RCF) scheduling algorithm is applied. The flash read command, which has the lowest cost among the flash commands, is prioritized.
2) *Size* is a command scheduling that also considers the size of the I/O request in RCF. The read commands from smaller-sized requests are processed first.
3) *RRF* prioritizes flash commands from read I/O requests described in Section V-A. The command translated from the read request is inserted into the RRQ, and the command translated from the write request is inserted into the WRQ.
4) *FOT* is a command scheduling to which the FOT-based scheduling algorithm is applied. In the RRQ and WRQ of the RRF, the flash command with a smaller FOT value is processed first.
5) *DRS* is an enhanced version of FOT. The FOT value is obtained by considering both the original and delayed requests affected by the flash command. This scheduling is also based on the preference of read requests.

Because a meaningful timestamp is needed to measure the latency of I/O requests, we experimented with real workloads. Nexus5 workloads collected from an embedded flash storage were used [31]. Table 1 presents the characteristics of each workload. From Movie to Radio, workloads are collected when individual applications are running. The other workloads are gathered when Radio and Message/WebBrowsing are running simultaneously.

B. STORAGE PERFORMANCE

Among all command scheduling schemes, DRS shows the most improved average latency of read I/O requests. Fig. 9 presents the average latency of each scheduling scheme as a value normalized to the RCF. In the case of write latency and I/O latency, DRS has the shortest latency in most workloads. Because DRS prioritizes read requests over write requests, the performance improvement in write latency does not differ significantly from that of Size. Rather, Booting has the

TABLE 1. Characteristics of workloads used in the experiment.

Workload	Avg. Read Size	Avg. Write Size	Write Req.	Req. Arrival Rate
Movie	27.5 kB	17 kB	5.40 %	4.79
YouTube	19.5 kB	13.5 kB	97.50 %	0.44
Booting	61 kB	37.5 kB	33.07 %	460.4
Facebook	28.5 kB	23.5 kB	74.42 %	3.5
AngryBird	51 kB	25 kB	84.51 %	1.59
Radio	36 kB	19.5 kB	98.68 %	1.31
Radio/Msg	17.5 kB	13 kB	98.15 %	16.82
Radio/WB	29 kB	19.5 kB	72.02 %	9.78

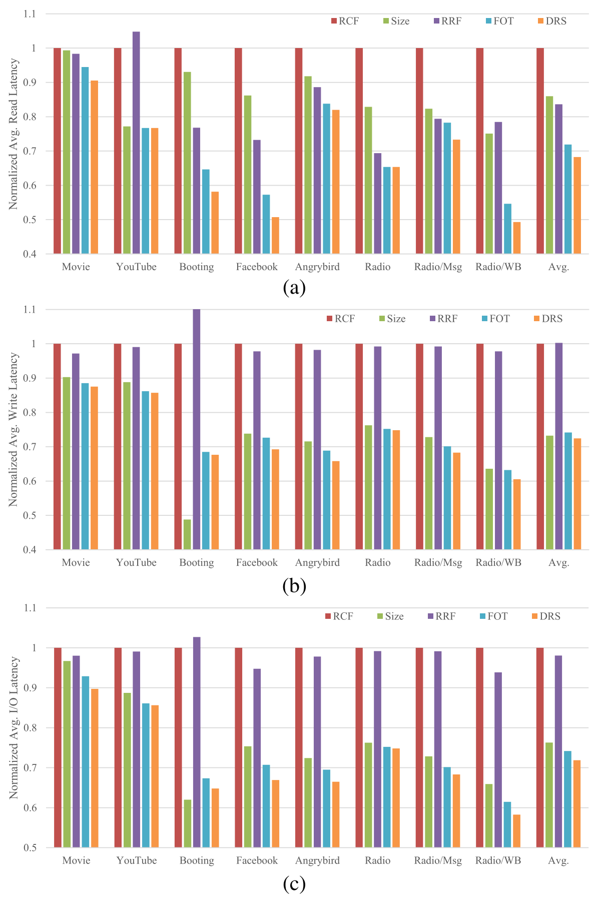

FIGURE 9. Comparison of average latency of I/O requests by command scheduling. (a) Read latency. (b) Write latency. (c) I/O latency.

shortest write latency in Size. Because Size does not classify the flash command by request type and schedules based on the size of the request, the write request may be processed first. In addition, Booting is a read-dominant workload, and the average write size is much smaller than that of read, which makes the result possible. Nevertheless, DRS significantly reduces the read latency, which also improves the I/O latency to a meaningful level.

The read latency of RRF showed a better performance than that of RCF on an average. This is a predictable result because the flash read command does not come only from read I/O requests. However, RCF processes read commands from different flash command sets in a round-robin manner, and therefore, read requests with smaller sizes may be partially prioritized. This prioritization is pronounced in YouTube. Because the workload is write-dominant and the average read size is relatively small, the read latency of RRF is higher than that of RCF. In Radio/Msg, which has similar characteristics to YouTube, read requests are more affected by program commands, and therefore, the read latency of RCF that does not prioritize program commands is higher than that of RRF.

It can be seen that the latency is further reduced by scheduling the command based on the characteristics of the I/O request. The size scheme additionally considers the size of the request in the RCF, and the FOT scheme additionally considers the FOT of the request in RRF, which reduces the latency. Of course, Size does not consider the flash command set generated by the map cache, and therefore, the read performance is lower than that of the FOT. The read latency of the FOT was the lowest in the mixed workload of Booting, Facebook, and Radio/WB. DRS further reduces the latency by considering the delayed requests in FOT. The Radio series shows that the read latency reduction effect of DRS is remarkable in combination with workloads.

C. QUEUE WAITING TIME

To check the performance improvement factor of the proposed command scheduling, we observed the change in queue waiting time. The I/O request is completed when flash operations are performed after going through the I/O request queue and flash command queue. Therefore, by breaking down the latency into the queue waiting time (QWT) and the flash operation time, we can see what causes the latency to decrease. Here, QWT was measured by dividing it into I/O request QWT and flash command QWT.

DRS significantly reduces the flash command QWT, which shortens the average I/O latency. Fig. 10 depicts the I/O latency breakdown based on the workload. DRS is compared with the baseline RCF, and their latencies are expressed from I/O request QWT to the flash operation time. The flash operation time does not change because of a command scheduling characteristic that only reorders the flash commands, and the flash command QWT decreases the most. This means the command processing speed increases, resulting in a slight decrease in I/O request QWT in some workloads that have bottlenecks in the request queue. Eventually, the proposed scheduling scheme improves I/O latency by reducing the total QWT.

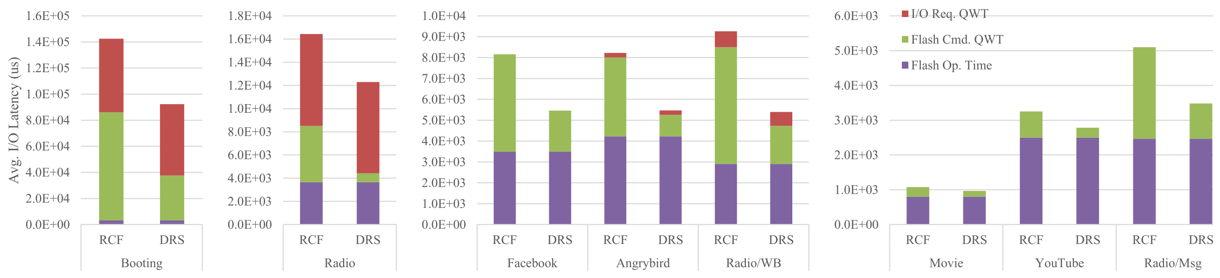

FIGURE 10. Queue waiting time is shortened, reducing the I/O latency.

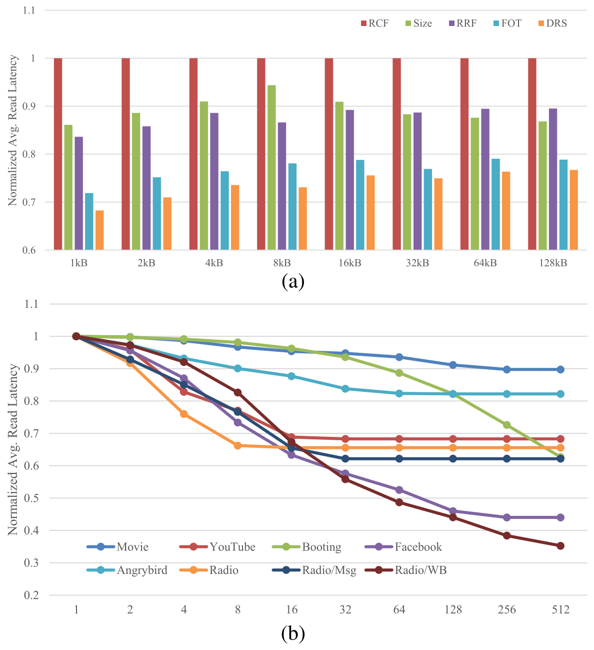

FIGURE 11. Average read latency varying (a) map cache size and (b) flash command queue depth.

D. PARAMETER STUDIES

Because DRS is a proposed command scheduling scheme for map cache flash storage, we verified how the map cache size and flash command queue depth affect performance through experiments. Fig. 11(a) shows the read latency of the average workload according to the cache size. Because DRS is a scheduling scheme that considers flash commands caused by cache misses, its performance is the best with the smallest cache size. In contrast, Size does not consider the map cache characteristics, and therefore, it is difficult to find a decreasing trend in performance with an increase in cache size. This result indicates that the proposed command scheduling is effective for embedded flash storage using a small amount of RAM.

Fig. 11(b) presents the result of measuring the average read latency by changing the queue depth from 1 to 512 in DRS. As the queue depth increases, the number of candidates for scheduling increases, thus improving the performance. Of course, the point at which performance is saturated depends on the workload. The larger the size and arrival rate of the I/O request in the workload, the more requests that can be queued, resulting in the saturation of performance at a larger queue depth. The performance saturation points of some workloads are not illustrated in the chart, such as Booting. Other workloads are rapidly saturated at a small queue depth, such as Radio.

VII. CONCLUSION

This paper presents a flash command scheduling scheme that exploits the characteristics of a map cache. The proposed scheme obtains the FOT of the I/O request from different flash command sets caused by cache hit/miss and reduces the average read latency by prioritizing the flash command generated from the read request with a shorter FOT. The FOT value of the flash command is recalculated considering not only the original request but also the delayed requests sharing the command, thereby further enhancing the effect of FOT-based scheduling. Experimental results demonstrate that the proposed scheduling scheme primarily reduces the flash command queue waiting time and improves the average latency. A trace-driven simulation shows that the average read latency of the proposed scheduling is reduced by up to 51% compared with the existing scheme, and the performance improvement is effective in a small map cache.

REFERENCES

[1] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *J. Syst. Archit.*, vol. 55, nos. 5–6, pp. 332–343, May 2009.

[2] F. Chen, T. Zhang, and X. Zhang, "Software support inside and outside solid-state devices for high performance and high efficiency," *Proc. IEEE*, vol. 105, no. 9, pp. 1650–1665, Sep. 2017.

[3] J. Kim, Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Disk schedulers for solid state drivers," in *Proc. 7th ACM Int. Conf. Embedded Softw.*, 2009, pp. 295–304.

[4] X. Xie, L. Xiao, D. Wei, Q. Li, Z. Song, and X. Ge, "Pinpointing and scheduling access conflicts to improve internal resource utilization in solid-state drives," *Frontiers Comput. Sci.*, vol. 13, no. 1, pp. 35–50, Feb. 2019.

[5] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve SSD performance," in *Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, Apr. 2017, pp. 375–388.

[6] R. Wang, Z. Chen, N. Xiao, M. Zhang, and W. Dong, "Assimilating cleaning operations with flash-level parallelism for NAND flash-based devices," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.*, Sep. 2014, pp. 212–219.

[7] B. Mao, S. Wu, and L. Duan, "Improving the SSD performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.

[8] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, Feb. 2009.
[9] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen, "S-FTL: An efficient address translation for flash memory by exploiting spatial locality," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, May 2011, pp. 1–12.
[10] Z. Chen, N. Xiao, F. Liu, and Y. Du, "2F: A special cache for mapping table of page-level flash translation layer," in *Proc. IEEE 16th Int. Conf. Parallel Distrib. Syst.*, Dec. 2010, pp. 585–592.
[11] H. Kim, S. Jung, and Y. Song, "Map cache management using dual granularity for mobile storage systems," *IEEE Trans. Consum. Electron.*, vol. 60, no. 4, pp. 644–652, Nov. 2014.
[12] J. Kim, J. Min Kim, S. H. Noh, S. Lyul Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.
[13] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, p. 18, Jul. 2007.
[14] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim, "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 4, pp. 1–23, Jul. 2008.
[15] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "Last: Locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, pp. 36–42, 2008.
[16] C. Ji, L.-P. Chang, C. Wu, L. Shi, and C. J. Xue, "An I/O scheduling strategy for embedded flash storage devices with mapping cache," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 4, pp. 756–769, Apr. 2018.
[17] W. Xie, Y. Chen, and P. C. Roth, "Exploiting internal parallelism for address translation in solid-state drives," *ACM Trans. Storage*, vol. 14, no. 4, pp. 1–30, Dec. 2018.
[18] W. Jeong, H. Cho, Y. Lee, J. Lee, S. Yoon, J. Hwang, and D. Lee, "Improving flash storage performance by caching address mapping table in host memory," in *Proc. 9th Workshop Hot Topics Storage File Syst.*, 2017, p. 5.
[19] K. Kim and T. Kim, "HMB in DRAM-less NVMe SSDs: Their usage and effects on performance," *PLoS ONE*, vol. 15, no. 3, Mar. 2020, Art. no. e0229645.
[20] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang, "Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–12.
[21] Y. Cho and T. Kim, "An efficient scheduling algorithm for NCQ within ssds," *IEICE Electron. Exp.*, pp. 12, May 2015, Art. no. 20150066.
[22] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "Hios: A host interface I/Q scheduler for solid state disks," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 289–300, 2014.
[23] M. Jung, E. H. Wilson, and M. Kandemir, "Physically addressed queueing (PAQ): Improving parallelism in solid state disks," *ACM SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 404–415, Sep. 2012.
[24] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*. Hoboken, NJ, USA: Wiley, 2013.
[25] S. Iyer and P. Druschel, "Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/Q," in *Proc. 18th ACM Symp. Oper. Syst. Princ.*, 2001, pp. 117–130.
[26] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simul.*, Sep. 2009, pp. 125–131.
[27] *Product Brief Document of SM2750 (UFS 2.1 Controller)*, Silicon Motion, 2020.
[28] *Products-Flash Card-Silicon Motion*. Accessed: Feb. 10, 2021. [Online]. Available: https://www.siliconmotion.com/products/Flash-Card/detail
[29] K. Smith, "Using QLC SSDs to improve cost/performance tradeoffs for warm data," in *Proc. Flash Memory Summit*, Santa Clara, CA, USA, 2019.
[30] C. Kim, "11.4 A 512Gb 3b/cell 64-stacked WL 3D V-NAND flash memory," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 202–203.
[31] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/O characteristics of smartphone applications and their implications for eMMC design," in *Proc. IEEE Int. Symp. Workload Characterization*, Oct. 2015, pp. 12–21.

GYEONGYONG LEE received the B.S. degree from the Department of Electronic Engineering, Hanyang University, South Korea, in 2014. He is currently pursuing the Ph.D. degree with the Department of Electronics and Computer Engineering, Hanyang University.

His research interests include embedded computing and NAND flash memories.

JAEWOOK KWAK received the B.S. degree from the Department of Electronic Engineering, Hanyang University, South Korea, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electronics and Computer Engineering, Hanyang University.

His research interests include computer architecture, embedded systems, and NAND flash-based storage systems.

JOONYONG JEONG received the B.S. degree from the Department of Information System, Hanyang University, Seoul, South Korea, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electronics and Computer Engineering, Hanyang University.

His research interests include NAND flash-based storage systems, databases, and key-value stores.

DAEYONG LEE received the B.S. degree from the School of Electronic Engineering, Soongsil University, Seoul, South Korea, in 2014, and the M.S. degree from the Department of Electronics and Computer Engineering, Hanyang University, Seoul, in 2017, where he is currently pursuing the Ph.D. degree.

His research interests include embedded systems and NAND flash memories.

MOONSEOK JANG (Student Member, IEEE) received the B.S. degree in electronic engineering from Hanyang University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree with the Department of Electronics and Computer Engineering.

His research interests include computer architecture, embedded systems, and flash memory storage.

JUNGWOOK CHOI received the B.S. and M.S. degrees in electrical and computer engineering from Seoul National University, South Korea, in 2008 and 2010, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, USA, in 2015.

From 2015 to 2019, he worked as a Research Staff Member with the IBM T. J. Watson Research Center. He is currently an Assistant Professor with Hanyang University, South Korea. His main research interest includes the efficient implementation of deep learning algorithms.

He has received several research awards, such as the DAC 2018 Best Paper Award, and has actively contributed to academic activities, such as being in a Technical Program Committee of DATE 2018-2020 (Co-Chair) and DAC 2018-2020 and a Technical Committee (DiSPS) of the IEEE Signal Processing Society.

YONG HO SONG (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, Korea, in 1989 and 1991, respectively, and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 2002.

He is currently a Professor with the Department of Electronic Engineering, Hanyang University, Seoul, and the Senior Vice President of Samsung Electronics Company Ltd. His current research interests include system architecture and software systems of mobile embedded systems, including SoC, NoC, multimedia on multicore parallel architecture, and NAND flash-based storage systems.

Dr. Song has served as a Program Committee Member of several prestigious conferences, including the IEEE International Parallel and Distributed Processing Symposium, the IEEE International Conference on Parallel and Distributed Systems, and the IEEE International Conference on Computing, Communication, and Networks.