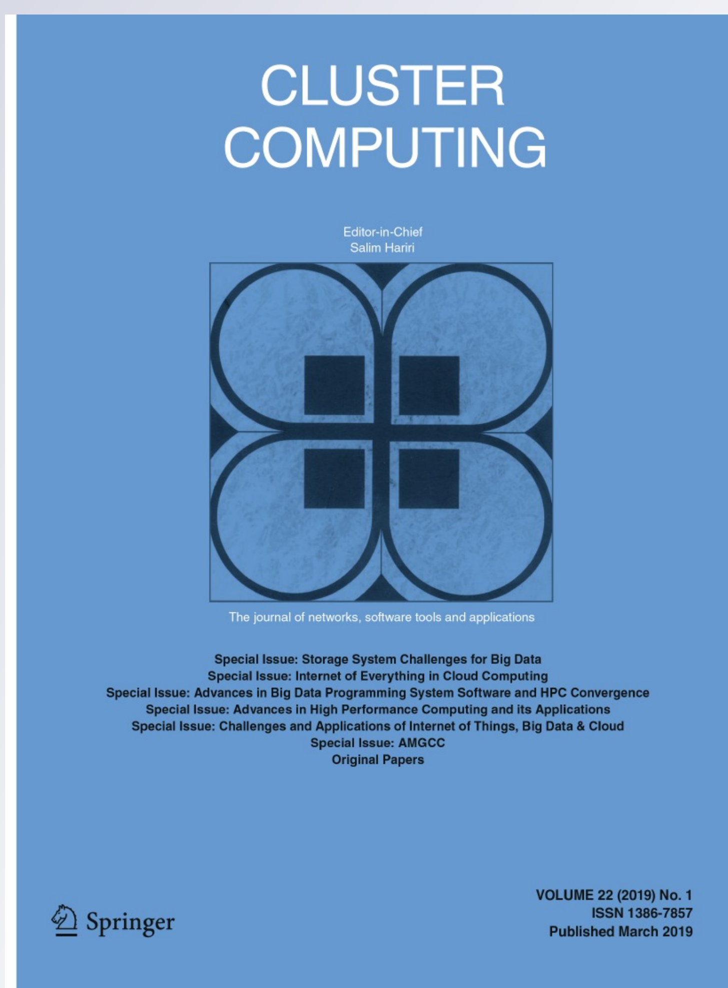
RFTL: improving performance of selective caching-based page-level FTL through replication

Ronnie Mativenga, Joon-Young Paik, Youngjae Kim, Junghee Lee & Tae-Sun Chung

Cluster Computing
The Journal of Networks, Software Tools and Applications

ISSN 1386-7857
Volume 22
Number 1

Cluster Comput (2019) 22:25-41
DOI 10.1007/s10586-018-2824-5



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science+Business Media, LLC, part of Springer Nature. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com”.

Springer

Author's personal copy

Cluster Computing (2019) 22:25–41
https://doi.org/10.1007/s10586-018-2824-5

RFTL: improving performance of selective caching-based page-level FTL through replication

Ronnie Mativenga[1] · Joon-Young Paik[1] · Youngjae Kim[2] · Junghee Lee[3] · Tae-Sun Chung[1]

Received: 28 November 2017 / Revised: 6 February 2018 / Accepted: 17 July 2018 / Published online: 25 July 2018
© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract

The internal nature of flash memory technology, makes its performance highly dependent on workload characteristics causing poor performance on random writes. To solve this, Demand-based Flash Translation Layer (DFTL) which selectively caches page-level address mappings, was proposed. DFTL exploits temporal locality in workloads and when low, high cache miss rates are experienced. In this paper, we propose a replication based DFTL, called RFTL, which aims at minimizing the overhead caused by miss penalty from the cached mapping table in SRAM. We developed an analytical model for studying the range of performance for RFTL. We extended EagleTree simulator to implement RFTL. Our experimental evaluation with synthetic workloads endorses the utility of RFTL showing improved performance over DFTL especially for read-dominant workloads. With 80% read dominant workload, RFTL's cumulative distribution function shows a 20% improvement and under 80% write dominant workload, it outperforms DFTL by 10% on I/O throughput.

Keywords Cached mapping table · Flash memory · Flash translation layer · Solid-state drive

1 Introduction

Unlike hard disk drives (HDDs), solid state drives (SSDs) have revolutionized the storage system ecosystem from mobile to enterprise-class storage server environments [1]. The price of SSD is now cheaper than HDD and supports higher performance. Flash memory capacity continues to grow as new memory technologies such as triple-level cell (TLC) [2] or 3D stacking memory are discovered to help increase flash memory density. As a result, more and more memory blocks are being integrated into a single die. For example, TLC has created 32 layers of TLC cells vertically, resulting in much higher storage densities for each die (3D stacking). The price of the SSD is \$ 0.17 per GB and the price has been reduced by 16.5% per year from \$ 0.99 per GB in 2012 and the 16TB SSD is on the market [3]. SSDs feature low operating temperature, low power consumption, vibration and durability against external shocks, light weight, flexible design and low access times. SSD is finally becoming a key storage device for persistent data storage.

The problem arises when SSD capacity increases. The page-mapped FTL table size increases as the SSD capacity increases and even requires a larger SRAM to load the corresponding mapping table. Thus this pure page-mapping is not feasible to implement especially in the current SSDs because of the high cost price/byte of SRAM. In an attempt to overcome this SRAM space limitation problem of the pure page-mapped FTL, the demand-based selective caching of page-level address mappings has been proposed [4]. A cache mapping table (CMT) maintains recently accessed mapping entries. When a cache miss

✉ Youngjae Kim
youkim@sogang.ac.kr

Ronnie Mativenga
ronniematie@ajou.ac.kr

Joon-Young Paik
lucadi@ajou.ac.kr

Junghee Lee
junghee.lee@utsa.edu

Tae-Sun Chung
tschung@ajou.ac.kr

[1] Ajou University, Suwon 443-749, South Korea

[2] Sogang University, Seoul 13557, South Korea

[3] University of Texas at San Antonio, San Antonio, TX 78249, USA

Author's personal copy

occurs, it has to involve page read and write operations to update CMT, which is called cache miss penalty. This penalty can become higher if there is interference with internal GC or any other ongoing operations. Hence, minimizing the cache miss penalty is critical for efficient DFTL.

Most state-of-the-art SSDs employ multi-channels, which allows access to data on different channels in parallel [5]. In this paper, we propose a method of replicating the entire page-level translation table on flash memory called *RFTL* to minimize the cache miss penalty when the size of the working set is larger than the size of the cached mapping table in the SRAM (CMT). The key idea of *RFTL* is to take advantage of the multi-channel architecture of SSDs where all channels are not always busy when a CMT miss occurs. *RFTL* utilizes the multi-channel architecture of an SSD and replicates the entire page-level translation table across all the channels such that page mapping entries do exist in isolation across channels. The *RFTL* uses the replicated mapping entries of the other idle channels. When a CMT miss occurs, *RFTL* can allow for an opportunity to skip busy channels and utilize replicated page mapping entries on different idle channels.

This paper has the following contributions.

- We propose a RFTL method to solve the high CMT miss penalty problem of DFTL by making several FTL copies in flash memory. When a mapping entry is replicated to exist in isolation on several different channels and a CMT miss occurs, the *RFTL* can skip the active channel and dynamically utilize the replicated page mapping entries in the idle channel for fast address translation.
- *RFTL* allows FTL to understand the underlying flash memory channel topology and replicate FTL to skip channels in use due to GC or other ongoing I/O services when a CMT miss occurs. Replicas reduce CMT miss penalty overhead. Specifically, our approach demonstrates better performance than a baseline without FTL replication for small random write workloads.
- In this paper, we used both the modeling method and the simulation method to analyze the effect of the RFTL method (*RFTL*). In particular, we used the Eagle Tree [6], which is popularly known as a SSD simulator, for the constraints of the mathematical modeling approach. Our evaluation shows that *RFTL* can minimize the CMT miss penalty without significantly reducing the lifetime of the NAND flash memory.

The remainder of this paper is organized as follows: Section 2 discusses background and motivation. Section 3 shows the architectural design and implementation of the RFTL method and Sect. 4 analyzes the boundary of the *RFTL* performance improvement with a mathematical model. Section 5 presents our experimental evaluation and Sect. 6 discusses the related works. We then conclude in Sect. 7.

2 Background and motivation

2.1 Background

The Flash Translation Layer (FTL) is one of the key elements in flash memory-based SSD. The FTL maintains a mapping table of virtual addresses to the physical address on flash memory. It helps emulate the functionality of a generic block device by displaying only read/write operations to the upper software layer and hiding the flash memory-based specific erase operations. Flash-based SSDs are asymmetric in read and write. The flash memory device can read the page (a unit of read/write), but can only write to the page with the special status that it clears. Flash memory is designed to erase at a block, a much larger unit than a page, because page-level erasing is extremely costly.

The Page-based FTL is ideal for performance, but it requires expensive, high-capacity SRAM memory, making it difficult to use in the real-world SSD system. On-demand selective caching-based FTL (DFTL) has been proposed as a method for selectively caching address mapping entries [4, 7]. This approach solves the huge memory requirement of the page-based FTL and successfully solves the overhead problem of the Full Merge operation of the Hybrid FTL [4, 7]. DFTL separates pages into data pages and translation pages.

In DFTL, the mapping entries stored in SRAM are maintained by Cached Mapping Table (CMT). And the translation pages are then maintained in a table called, Global Translation Directory (GTD), locate translation pages scattered across the chip if missed from cache. However, since this DFTL does not cache all address translations, it needs access to the address translation request for the mapping item of the flash memory when there is no mapping entry corresponding to the address translation request in the SRAM. Therefore, the actual performance of the DFTL is greatly influenced by the working set size ratio of the workload to the SRAM size, that is, the SRAM hit ratio.

The SSD performs garbage collection internally. In flash memory, the page is valid and has an invalid state and a clean state. When the number of invalid pages reaches a certain threshold, a GC operation is performed to create clean pages. The GC includes page read and write and erase operations. The page write/programming time is about 200 us, while the block erase takes about 1.5 ms [8]. GC is the most expensive operation in flash memory [3]. There is a background GC method that runs the GC when

Author's personal copy

the system/channel is idle to minimize the conflict with I/O processing [9]. More worse, specifically in DFTL, when a CMT miss occurs, the address translation performance becomes worse if there is a conflict between flash memory access and internal GC operation for address translation. Therefore, we address this issue by redirecting it to a replicated address translation entry using a multi-channel SSD architecture.

Most SSDs are implemented using multiple channels to take advantage of the abundant parallelism of current NAND flash memory [10–12]. Figure 1 shows the internal parallel architecture of a multi-channel SSD. Every chip consists of multiple dies, each containing multiple blocks. Each block consists of several pages, and the Translation page contains the address mapping of the page based FTL. The SSD system considers die and plane as internal parallel units and can be thought of as an internal parallel device at the top of the channel and flash memory chips. In this paper, we propose a technique to replicate FTL and utilize replicated FTL to minimize CMT miss penalty in multi-channel SSD using DFTL. For example, a conflict with GC during a CMT miss will increase the FTL cache miss latency [6]. Internal I/O for address translation must wait for the GC to complete its turn. However, if a replica entry page in the idle channel is available, it can avoid conflicts with the GC.

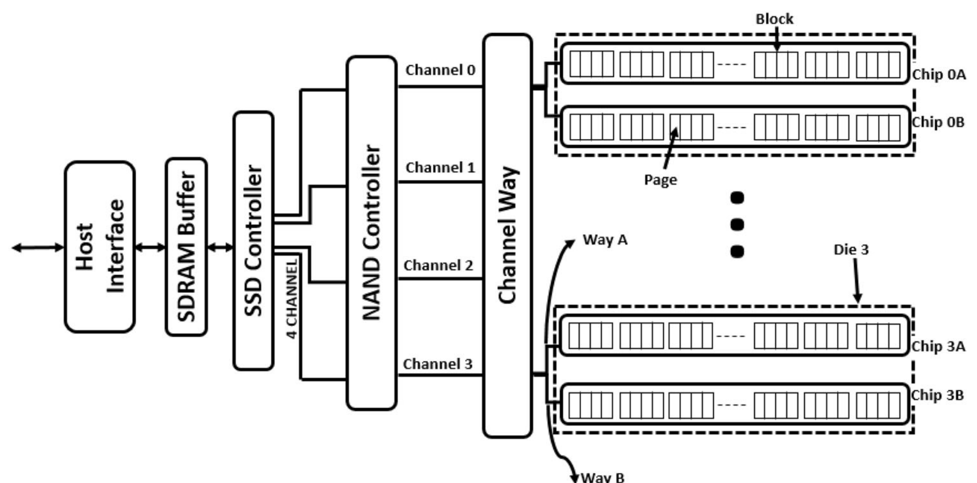
2.2 Motivation

In SSDs using the Selective Caching-based Page-level FTL [4, 7], CMT hit or miss ratio can be determined according to the mapping entries of cached mapping table (CMT) loaded in SRAM and the working-set size of workload. When a CMT miss occurs, flash memory reads and writes occur because the mapping page storing the corresponding mapping entry must be accessed from flash memory. Since flash memory access latency is several hundred times slower than SRAM access latency, the time overhead of processing CMT misses from flash memory is quite high. This is called the *CMT miss penalty*.

In particular, when a CMT miss occurs, the channel storing the mapping page may be busy. In this case, you must wait for the channel to become idle. The CMT miss penalty time becomes larger because CMT miss can not be processed. Figure 2 illustrates the worst-case situation of a CMT miss penalty for a select-page-level FTL. When a CMT miss occurs, it attempts to access Channel#2 to access the corresponding mapping page. However, Channel#2 is currently busy with GC as an internal GC job. That is, the Channel#2 access collision occurs with the GC operation. CMT misses cannot be processed immediately. This situation includes not only conflicts with GC, but also conflicts with normal read writes for other I/O operations. Therefore, in this paper, the case of channel collision considering two cases will be examined.

Figure 3a shows that flash memory I/O to process CMT misses conflicts with normal I/Os accessing flash memory on the same channel. In the figure, R_{CMT_Miss} is a CMT miss flash memory I/O, and it requires to access the Channel#0. On the other hand, at that moment when R_{CMT_Miss} arrives, there are three ongoing normal I/Os (W_{normal}, R_{normal}, and E_{normal}) to access Channel#0. R_{CMT_Miss} must wait until these three normal I/Os finish. Also, I/O for processing CMT miss may conflict with internal GC I/Os. That is when R_{CMT_Miss} arrives while GC is still running. R_{CMT_Miss} will conflict with ongoing GC operations such as R_{GC}, W_{GC}, and E_{GC} in the figure. As R_{CMT_Miss} must wait until these internal GC operations finish, the R_{CMT_Miss} processing will be delayed due to the shared channel resource conflict with the GC or normal I/Os. However, as shown in Fig. 3b, if a copy of the flash memory page for processing the CMT miss is present on Channel#1, it can be processed in parallel with normal I/Os on Channel#0. On the other hand, if a copy of the page that is needed to process R_{CMT_Miss} on Channel#1 is available,

Fig. 1 Structure of a multiple-channel SSD



Author's personal copy

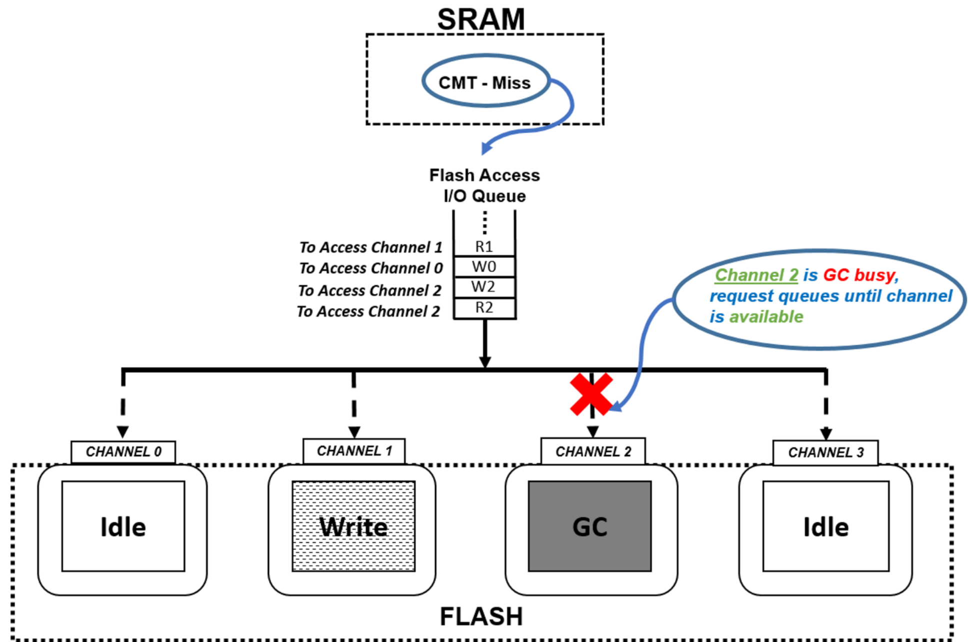


Fig. 2 CMT miss penalty example: flash memory I/O for CMT miss processing is currently delayed due to a collision when accessing Channel#2 with flash memory I/Os for internal GC operations

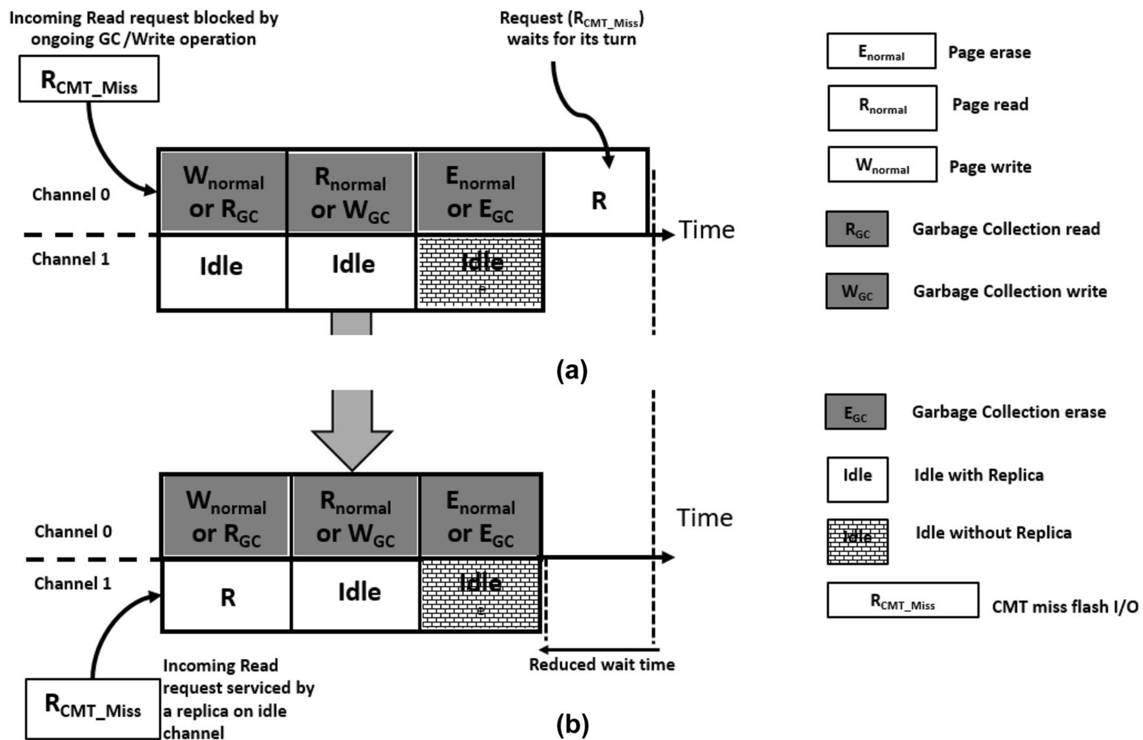


Fig. 3 **a** Flash memory I/O to process CMT miss conflicts with internal GC operations or Normal write operation. **b** A copy of the flash memory page on channel#1 is used to skip GC busy channel#0 and process the CMT miss in parallel with GC operation

then, it can also be immediately processed in parallel with GC operations on Channel#1.

To minimize the processing time of CMT miss penalty in selective caching-based page-level FTL, we propose the idea of replicating mapping pages across multiple channels. This allows for fast address translation, minimizing this CMT miss penalties. As shown on Fig. 2, the flow processes experienced during DFTL's worst case are expensive, that is on performance and this is made even worse when requests are queued due to access conflicts [13].

Author's personal copy

3 Design and implementation for RFTL

RFTL is designed to prevent performance degradation when a request undergoes a cache miss penalty with selective caching-based page-level FTL. The request may need to wait if the channel where the request should go is busy, until the channel becomes available. The channel can be unavailable due to other ongoing normal I/O operations and GC operations. By duplicating the page mapping table across multiple channels, it can dynamically use the duplicated mapping entries available in the idle channel for fast address translation. The following section illustrates the RFTL architecture and its advantages compared with the selective caching-based page-level FTL.

3.1 RFTL architecture

Figure 4 shows a descriptive diagram of the architecture of RFTL. In RFTL, the entire page-level translation table will be stored in flash. Flash blocks are logically divided into Data Block and Translation Block. The data block stores the user's data. The translation Block stores the address translation pages of the entire page-level translation table. The translation block is divided into the original translation block and the replica translation block. Depending on the value of the RFTL's replication factor, several replicated translation blocks may exist. In particular, a translation page stores only address translation entries that translate logical page addresses into physical page addresses whereas a normal page stores the user's data. SRAM only caches frequently accessed translation entries, which are maintained by RFTL Cached Mapping Table (RCMT).

When a single entry in the translation page is modified, a page write operation is performed. A new free page is allocated and written according to the out-of-update rule of NAND flash. The newly written page needs to be kept track of and it's managed by RFTL Global Translation Directory (RGTD). The GDT is stored in the SRAM along with the RCMT and is mainly used for location tracking when the physical page position of the translation page changes. A GDT table entry consists of a virtual page number (MVPN) and multiple physical page numbers (MPPN and MRPNs). The MPPN in the RGTD represents an address of the original translation page while the MRPNs are addresses of multiple replicated translation pages.

In RFTL, RGTD requires more memory space than the existing DFTL GTD because all the original translation pages and all duplicate translation pages must be tracked. However, since the RGTD size is smaller than the RCMT, this is not a big problem. If 4bytes are needed to represent the physical page address on flash with each translation page capable of storing 512 mappings. Therefore a 1 GB flash would require 1024 translation pages and its GTD size would only be 4 KB [4]. If we assign two replicas per each translation page, then our RGTD would require only about 8 KB of SRAM space for the same 1 GB SSD.

Considering 4 bytes for page addressing and 512 mapping entires clubbed in a 4 KB page, the flash space overhead to store the entire page-level translation table is only 0.2%. A 10 GB flash device requires only about 20 MB of space to store all the mappings in the traditional DFTL [4]. In other words, total size of FTLs on flash is highly dependent on the number of replicas assigned to each translation page. For example, considering a single replica of the entire page-level translation table, total size of flash memory required to store the entire page-level translation table including its replication is 40 MB of the 10 GB SSD. Three replica requires 60 MB of space. Overall RFTL can require more flash memory space required for keeping original FTL and SRAM space for our page-mapped table also known as page-level translation table on flash, but its space overhead is minimal.

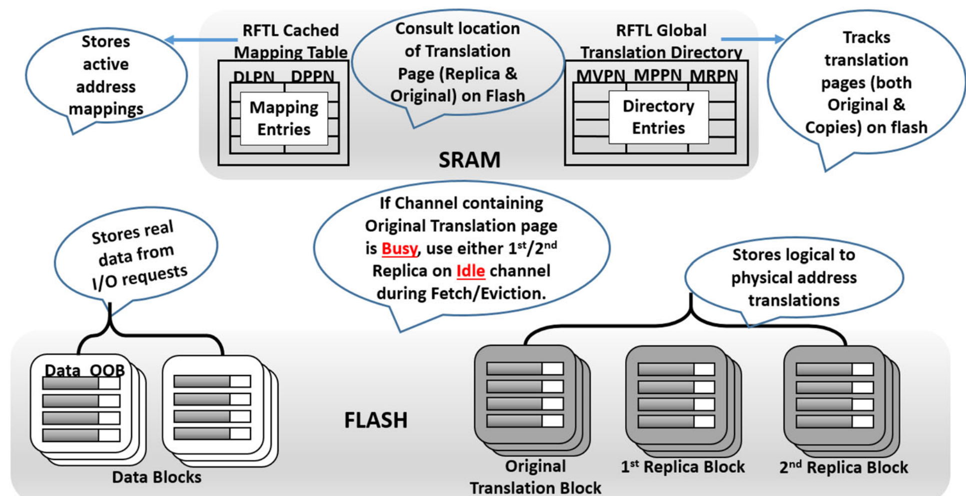


Fig. 4 The schematic design of RFTL

Author's personal copy

Algorithm 1: RFTL Address Translation

```
1  Input: Request's Logical Page Number (request_lpn), Request's Size
     (request_size)
2  Output: NULL
3  while request_size ≠ 0 do
4    if request_lpn miss in RFTL Cached Mapping Table then
5      if RFTL Cached Mapping Table is full ; /* select entry
         for eviction using segmented LRU replacement
         algorithm */
6       then
7        victim_lpn ← select_victim_entry()
8        if victim_last_mod_time ≠ victim_load_time ;     /* victim_type:
           Translation or Data Block
           Translation_Page_victim: Physical
           Translation-Page Number containing victim
           entry */
9         then
10         Translation_Page_victim ← consult_RGTD (victim_lpn)
           victim_type ← Translation Block
11         if victim_request_channel is Busy then
12           Get(victim_rpn) ;    /* get alternative entry
               from victim's Replica page on an idle
               channel */
13         end
14         RFTL_Service_Request(victim)
15       else
16         erase_entry(victim_lpn)
17       end
18     else
19       Translation_Page_request ← Consult_RGTD(request_lpn) ;
          /* load map entry of request from flash
          memory chip into RFTL Cached Mapping Table
          */
20       if request_channel is Busy then
21         Get(request_rpn) ;   /* get an alternative entry
             from Translation Page's Replica on an
             idle channel or wait until channel is
             idle */
22       end
23       load_entry(Translation_Page_request)
24     end
25   else
26     request_type ← Data Block
27     request_ppn ← RCMT_lookup(request_lpn)
28     RFTL_Service_Request(request)
29     request_size
30   end
31 end
```

Author's personal copy

3.2 RFTL operation

3.2.1 Mapping operation

The mapping table (RCMT) of *RFTL* maintains two or more physical locations (Data Physical Page Number: *DPPN*) of the same mapping information per every given logical page number (Data Logical Page Number: *DLPN*) depending on the number of replicas allocated for each mapping page (*MRPN*). In other words, unlike *DFTL* where there is a single DPPN for every DLPN in the mapping table, a multiple number of alternative copies of the same mapping page are stored across multiple channels in case the corresponding DPPN is inaccessible due to its channel being busy with other operations. As illustrated from our Algorithm 1 which show the mapping process, *RFTL* can use this opportunity to utilize idle channels with same copy (DRPN) of the requested mapping page. Figure 5 again shows the whole operation and here only a single copy of every mapping page is maintained for illustration purposes.

Due to out-of-place updates policy in flash memory, translation pages are scattered throughout the chip. To keep track of these translation page locations, the RFTL Global Translation Directory (RGTD) on Fig. 5 stores all the mapping locations of these pages. Unlike DFTL, it also contains multiple reference (Physical Translation Page Number: *MPPN* and RFTL Physical Translation Page Number: *MRPNs*) for each logical page number (Logical Translation Page Number: *MLPN*). The MRPNs are the translation page copies for each original translation page (MPPN). This means that a single translation page has a copy or more across different chip locations.

3.2.2 Read operation

Once address translation is completed, flash memory page read operation is initiated and serviced directly. For a mapping read during RCMT miss, *RFTL* check is invoked to assess the channel status first. If the channel is busy then an alternative page (replica) on an idle channel is provided otherwise if idle then the original mapping page is read directly. Algorithm 1 shows the steps that are followed through the whole read operation until requested data has been located on flash memory.

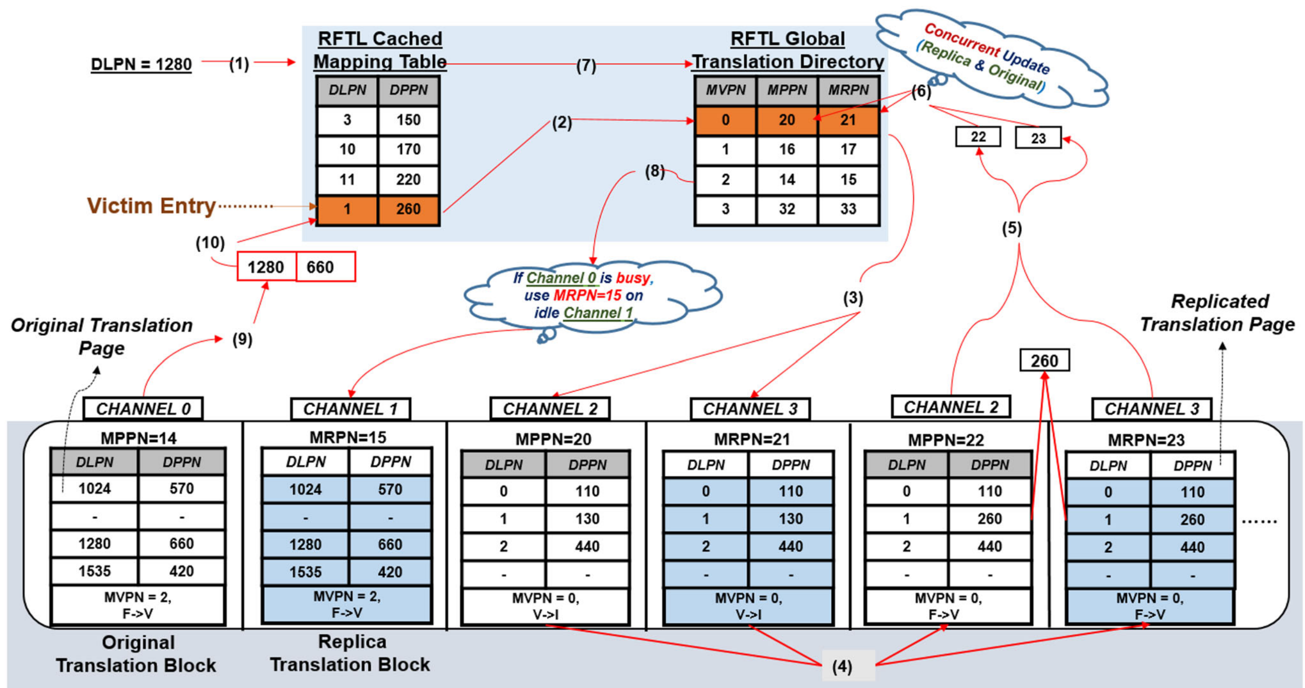


Fig. 5 (*1*) Request to DLPN 1280 incurs a miss in RFTL Cached Mapping Table (RCMT), (*2*) Victim entry DLPN 1 is selected, its corresponding translation page MPPN 20 and its copy MRPN 21 are located using RFTL Global Translation Directory (RGTD), (*3*)–(*4*) If channel 2 is idle, MPPN 20 is read else MRPN 21 on channel 3 is read, both pages updated (DPPN 130 → DPPN 260) and concurrently written to free translation pages (MPPN 22 and MRPN 23), (*5*)–(*6*) RGTD is updated (MPPN 20 → MPPN 22) and (MRPN 21 → MRPN 23) and DLPN 1 entry is erased from RCMT. (*7*)–(*11*) The original requests (DLPN 1280) translation page is located on flash (MPPN 14 and MRPN 15) and if channel 0 with MPPN 14 is busy, it is skipped and a replica (MRPN 15) on an idle channel 1 is used instead. The mapping entry is loaded into RCMT and the request is serviced. Note that each RGTD entry maps 512 logically consecutive mappings

Author's personal copy

3.2.3 Translation page write

When a translation page is updated on flash memory chip, it is written in batches together with its copies (replicas). This technique is called *batch-updating*. For example, on Fig. 5, each translation page has a single copy on a different channel. This means that each page is written/updated onto two different locations (one original translation page and its copy) on chip at the same time. The reasons being, (1) to reduce the write overhead and (2) for a synchronous location update on RGTD and RCMT after the write operation is completed.

During a cache miss , a victim translation page entry is updated on flash memory before it is evicted from RCMT (write-back policy) this is because of *Lazy copying* feature of *RFTL* do as to reduce write latency thereby improving the overall system's response time. To facilitate for a quick update, *RFTL* checks the channel status and if busy then an alternative location on an idle channel containing a copy of the same translation page is availed for faster address translation. This process is done each time he system tries to access the chip during a RCMT miss.

3.2.4 Fault tolerance

RFTL maintains a full page-level mapping table (FMT) on flash to ensure and maintain the consistency of its translation pages. If power is lost unexpectedly, a portion of the FMT is uploaded from the non-volatile flash into SRAM and becomes our cached mapping table (RCMT) after reboot. Furthermore, mapping write requests are propagated to all the target MRPNs Die registers by the controller pending execution. For example, an MVPN write request will have pointers to 1 original MPPN plus multiple replica MRPNs on flash. So if machine crashes before all replicas are completely written, after system reboot, the information in the non-volatile Die registers (including replica write request) is retained. This enables the write execution to proceed, that is, ensuring thatwrites are complete only when all replicas are created.

3.2.5 Garbage collection

Our approach (*RFTL*) maintains a GC threshold which is the minimum number of available free blocks per given time. If this limit is reached, then GC is triggered to claim all the invalidated pages from both Data Blocks and Translation Blocks. The garbage collector selects its victims based on an analysis adopted from [14] and if the victim is a Data block then all the valid data pages are copied to the Current Data Block and *RFTL* updates all the translation pages and RCMT entries associated with these pages. On the other hand if the victim is a translation block then the valid pages are copied to the Current Translation Block and the RGTD is updated.

Table 1 Comparison of number of replica writes and RCMT size

FTL type	Replicas (#)	Total FTL (MB)	E
Baseline	1	200	0
RFTL (1)	2	400	0.4
RFTL (3)	4	800	1.2
RFTL (7)	8	1600	2.8
RFTL (15)	16	3200	6

The number in parentheses denotes the number of replicas of the page mapped table. E = extra writes due to RCMT miss/total writes

Table 1 shows the total space overhead required for RFTL and DFTL on a 100 GB SSD. If the required mapping information of a request is present in RCMT, it is processed directly through the available location of its actual data and this is called a *RCMT-Hit*. On the other hand, if the mapping information is absent from the RCMT *(RCMT-Miss)*, then this it has to be retrieved from the flash memory chip into the RCMT, Algorithm 1 clearly shows this. When a request is issued, there are two states in which a mapping table can be at any given time, that is NOT-FULL (*Best Case*) or FULL (*Worst Case*).

3.3 Best and worst cases for CMT misses

Best case When space is available in RCMT (*Best-Case*), the incoming request entry is inserted into the mapping table and its corresponding mapping information fetched from the flash memory chip via RGTD that is used to locate the exact position of the mapping page on flash memory. It is during this flash memory access stage where *RFTL* will check the mapping-page location's *channel* to see whether it is *busy* or not and if busy, an alternative location with a copy of the requested mapping page, a (*RFTL*) on an idle channel is provided. Instead of the incoming request being queued until the its turn comes or till the ongoing operation is completed, mapping information is availed faster from the mapping page copy on an idle channel thus minimizing the miss penalty.

Worst case Otherwise if RCMT is full, then a victim has to be selected (using algorithm-LRU) [15] for eviction while keeping in mind that most selective page-level mapping schemes including DFTL adopts Lazy copying (write-back) policy for cached entries. Before updating the victim's entries on flash memory, *RFTL* also checks the channel status of the required chip to provide a mapping page copy from an alternative channel whenever the status returns busy state. In this scenario (*Worst Case*), all channels will be busy including the ones with replicas

Author's personal copy

therefore forcing the request to queue for its turn. Algorithm 1 describes this process for servicing a request during the worst-case scenario. This means that the victim has to be first updated on flash memory (read, Channel status checked, updated and then re-written onto a new physical location) before it is evicted from RCMT. The RGTD is then updated with the victim's new physical location prior to its eviction from the RCMT. The incoming request is then inserted into RCMT and follows the same steps of fetching mapping reference from flash memory via RGTD and channel status checking (*RFTL* check) as mentioned above during the best-case scenario when there is space in the mapping table (RCMT).

4 Modeling and analysis of RFTL

4.1 Performance modeling

In this section, we study the performance and lifespan of our *RFTL* approach and evaluate its efficacy against that of DFTL, which we present as our Baseline approach. We analyze the efficacy of RFTL against DFTL using a simple analytical modeling approach. For modeling, we used several notations, which are detailed in Table 2.

Baseline approach (DFTL) The original DFTL approach is considered as our baseline approach. In DFTL, if a CMT miss occurs for a request and its address translation requests should be served by a busy channel in SSD, those page read and write requests associated with the address translation process should wait until the channel comes to idle.

By considering the probability of a channel being busy (P_{GC}), (P_{GC}=1) in DFTL wost case together with two-reads and a single-write when a cache miss occurs while CMT is full, average CMT access time can be formulated as:

$$F_{(base)} = [C + GC + (2R + W)] \times M + [C \times (1 - M)] \quad (1)$$

RFTL In *RFTL*, an entire page-level translation table is replicated on flash memory. During a CMT miss, mapping pages on idle channels can be utilized. Such pages on idle channels, if any, can be used to avoid accessing busy channels during a CMT miss. By considering the probability of getting a replica on an idle channel (*P*) and miss ratio (*M*), we modeled the following scenarios:

Best case For a single CMT miss, when a channel that contains its corresponding mapping page is busy and its replica page is available on an idle channel (*P* = *1*), We have:

$$F_{(RFTL,best)} = [C + (2R + W)]M + [C(1 - M)] \quad (2)$$

Worst case: For a single CMT miss, if a channel with that corresponding mapping page is busy and there is no replica page available on any channel (*P* = *0*), it has to wait until the busy channel comes to idle. Compared with the best case, the CMT miss is delayed for a longer time by GC because it has to wait for completion of the ongoing GC. This situation can be modeled as:

$$F_{(RFTL,worst)} = (C + GC + (2R + W)) \times M + [C \times (1 - M)] \quad (3)$$

4.2 Performance analysis

In this subsection, we will use the mathematical equations for each approach as presented in Sect. 3 to run experiments and then compare the performance and lifespan results of *RFTL* against our Baseline approach that implements a simple DFTL. For evaluation, we considered SSD configurations as shown in Tables 3 and 4.

Performance For comparison, we considered similar test environments for DFTL (P_{GC}) and *RFTL* (P) approaches as shown in Table 1. From Fig. 6, we can observe that *RFTL* is superior to DFTL with an overall improvement of around 20%. We also compared the average response times of DFTL and *RFTL* by varying CMT miss ratio from 0 to 1.

Table 2 Descriptions about parameters used in modeling

Constants	Description
C	DRAM (CMT access time)
GC	Garbage collection wait time
R	Read latency
W	Write latency

Variables	Description
F	Channels (#)
M	Miss ratio
N	Replicas (#)
P	N / F (probability of replica on idle channel)
P_{GC}	Probability channel being GC busy
E	Value of extra writes

Table 3 SSD settings

Configuration	Value
DRAM (ns)	100
SSD channels (#)	8, 16
GC wait (µs)	900
Read latency (µs)	25
Write latency (µs)	200

Author's personal copy

Table 4 Configurations

	Baseline	RFTL
P_{GC}	0, 0.5, 1	–
P	–	0.25, 0.5, 0.75
F	8, 16	8, 16
N	–	2, 4, 6

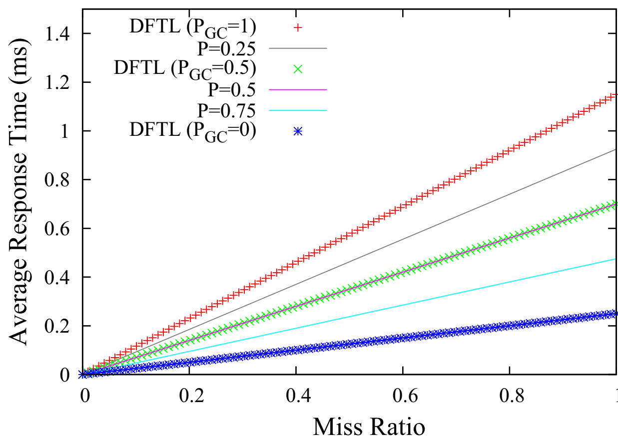


Fig. 6 Mapping response time with increased miss ratio on an 8 channel SSD

Considering the worst case of DFTL implementation (where P_{GC}=1), we observe that request latency can increase up to 70% when miss ratio gets to 0.6. Taking *RFTL* into account, we clearly observe its effectiveness as it showed about 50% delay in average response time. In all test cases, we can clearly notice that *RFTL* can outperform baseline (DFTL).

Lifespan In our experiments, we considered a 100 GB SSD where I/O requests arrival rate is 1000 IOPS. We varied the number of copies of our entire page-level translation table (*RFTLs*) on each scenario to compare extra writes required. *E* is a variable representing the ratio of extra writes. Figure 7 shows additional write overheads for both DFTL and *RFTL*. *RFTL* can incur more writes compared to DFTL, however for Replica (1), overall writes on SSD are around 40% higher than DFTL. It will only take 0.2% additional flash memory space to cover the replicated entire page-level translation table while improving 20% overall response time. As we maintain more replicas of the entire page-level translation table, it will increase more writes to synchronize all the replicated entire page-level translation table on flash memory. However, this will increase the chances of utilizing replicated mapping pages on idle channels for a CMT miss. A summary of our test configuration settings and results is shown in Table 1.

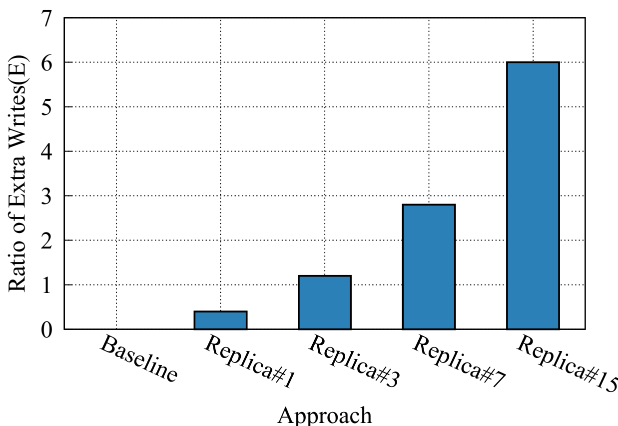


Fig. 7 Extra writes ratio (E = extra writes due to CMT miss/total writes) for baseline and *RFTL*

5 Evaluation

The mathematical model presented in Sect. 4 models only the performance of the CMT miss penalty and is therefore limited to analyzing the overall I/O system performance when SSD employs RFTL. Therefore, we instrumented a well-regarded trace driven SSD simulator called EagleTree [6] to implement RFTL to evaluate and analyze the actual Input/output operations per second (IOPS) of RFTL compared with baseline (DFTL).

5.1 Experiment setup

The EagleTree [6] can simulate multi-channel SSDs [12]. It allows for various configurations of specific flash memory chip types from Single-Level Cell (SLC) to Multi-Level cell (MLC) [2] configurations while supporting for advanced commands like *pipelining* and *copy-backs*. For the SSD configurations, we modeled an 4 GB SSD, which is configured with 4 channels with each channel equipped with a single chip. Each Chip is composed of 2 Dies, and each die is constructed with a single plane. Each Plane contains 1024 flash memory Blocks. Each flash memory block consists of 128 Pages with 4 KB page. Access times that the simulator used are shown in Table 5. The Greedy GC method is employed and it limits the maximum number of concurrent GC operations less than and equal to 2 channels at a time. The I/O scheduler uses NOOP. The over provisioning ratio is set to 30% of the SSD, which complies with the setting in several recent works [8, 16].

For the data allocation scheme, the most widely used Channel-Chip-Die scheme is applied since it has the best performance in term of the parallelism of SSD [8]. The die level parallelism is called internal parallelism where

Author's personal copy

Table 5 Operational SSD parameters and access timings

Parameter	Access time (µs)
Page read delay	25
Page write delay	200
Bus data delay	100
Block erase delay	1500
Parameter	**Size (KB)**
I/O size	4
Page size	4
Event size	4

multiple dies can be accessed simultaneously while differing from the internal parallelism, the channel and chip level parallelism are commonly supported parallelism, which enables host I/O requests to be processed in parallel at different channels and chips.

Synthetic workload benchmarks were used to evaluate various I/O patterns in the workload. In particular, we considered both write dominant and read dominant workloads. Following 80/20 rule, the write dominant workload is 80% writes and 20% reads and the read dominant workload is 20% writes and 80% reads. To compare the efficacy of RFTL against DFTL, we compare Input/output operations per second (IOPS) and lifetime of RFTL and DFTL. Table 6 shows the sizes of SRAM used to adjust the miss ratio for our above configured 4 GB SSD.

To run simulations with varied number of replicated mapping pages for our proposed approach, we used RFTL approaches described by Table 7.

Table 6 4 GB SSD mapping table sizes in SRAM at each miss ratio

SRAM size (KB)	Miss ratio
32	0.9
68	0.8
147	0.6
208	0.4
247	0.2
512	0

Table 7 RFTL implementation based on number of replicated tables

RFTL approach	Description
RFTL(1)	1 replica copy written
RFTL(2)	2 replica copies written
RFTL(3)	3 replica copies written

The number in parentheses is the replication factor

5.2 Experimental results

In our evaluation, we ran experiments for mapping reads and writes I/O performance check. After this evaluation, we then compared the overall system performance from RFTL and Baseline (DFTL).

5.2.1 Mapping I/O performance

Since RFTL aims to improve performance by speeding up address translations during a cache miss, considering mapping I/O performance is of importance to our evaluation analysis. We used these to check this performance improvement by RFTL and then compared them with the ones obtained from our mathematical models in Sect. 4 in-order to validate of our proposal. We further compared the actual response time for every I/O for both read and write-intensive workloads by placing them into ranges of response time against their cumulative distribution frequency (CDF).

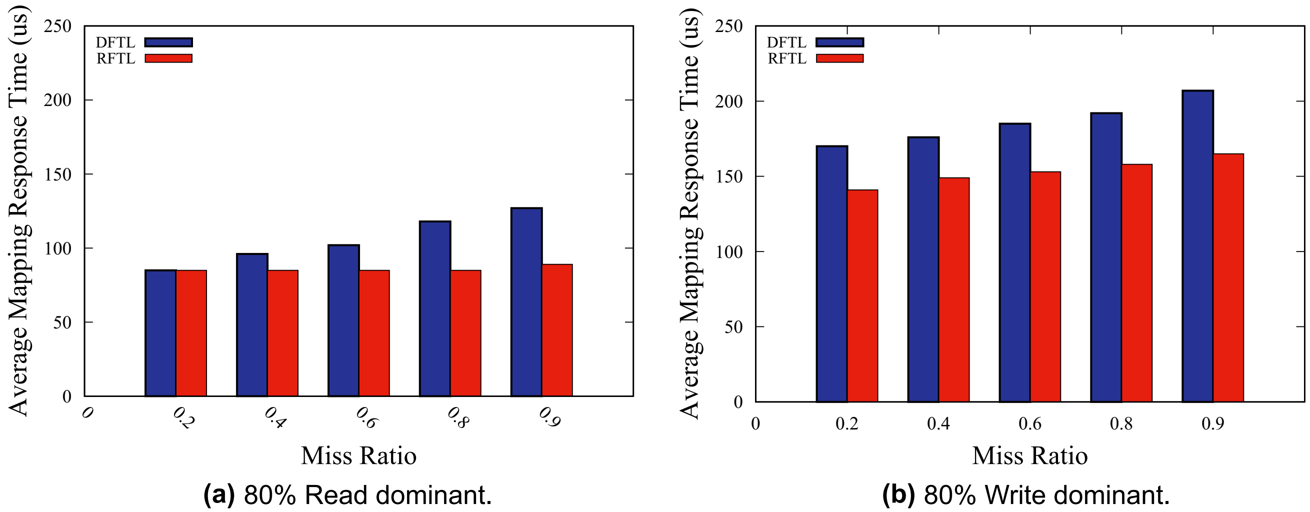
Figures 8 and 9 shows mapping reads and mapping writes consecutively from both read-dominant (a) and write-dominant (b) workloads. From Fig. 8, RFTL out performs DFTL by an average of 17% improvement as the miss ratio increases. While on the other hand, Fig. 9b shows RFTL's mapping write operations from a write-dominant workload environment averaging around 55% on performance improvement as compared to DFTL. This is because write intensive workloads increase the chances of garbage collection (GC) which results in more more delay due to increased internal conflicts while GC is operational. DFTL suffers greatly from this effect but as seen from the results, RFTL's performance is exceptional because it can skip GC-busy channels and utilize replicas on idle ones.

On Fig. 10, RFTL shows more than 90% of the I/Os having response time below 300 µs while DFTL has only below 65% of I/Os with the same response time for Mapping Writes. This is due to the high percentage of reads ratio that increases cache access demand thereby increasing chances of CMT miss for such Read intensive workload. When the workload is write intensive (Fig. 11), we see RFTL having 100% of I/Os with response time between 50 and 100 µs while DFTL at this range only having below 55% for Mapping-writes. Same applies for Mapping-reads where DFTL having below 80% of its I/O ranging below 100 µs and 10% of I/Os exceeding 350 µs response time.

5.2.2 End-to-end I/O performance

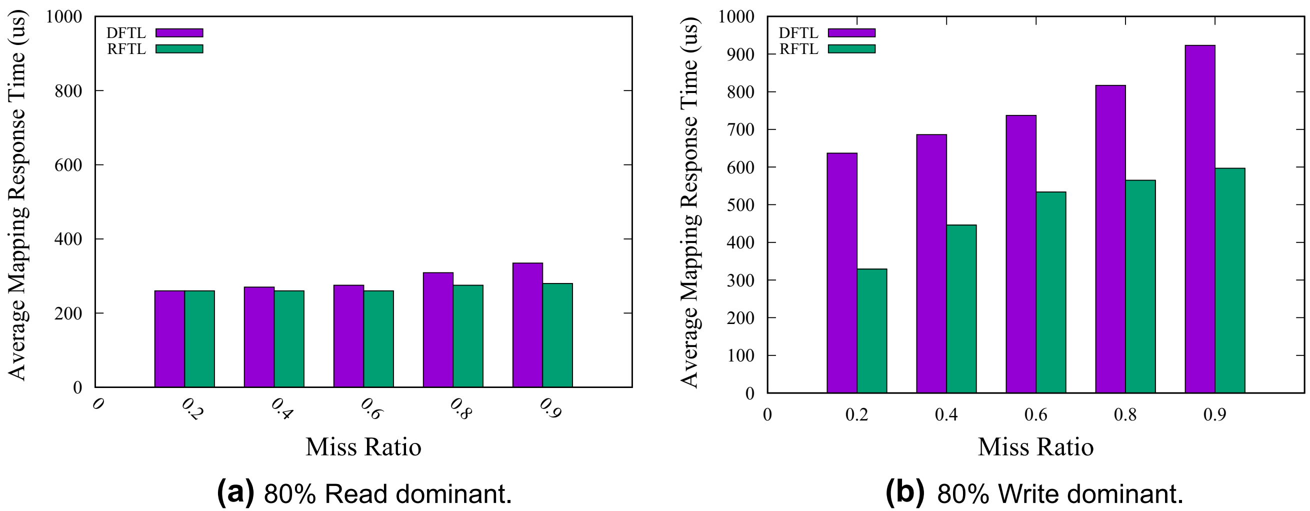
The miss ratio of the cache mapping table in SRAM is important for a fair comparison of RFTL and DFTL

Author's personal copy



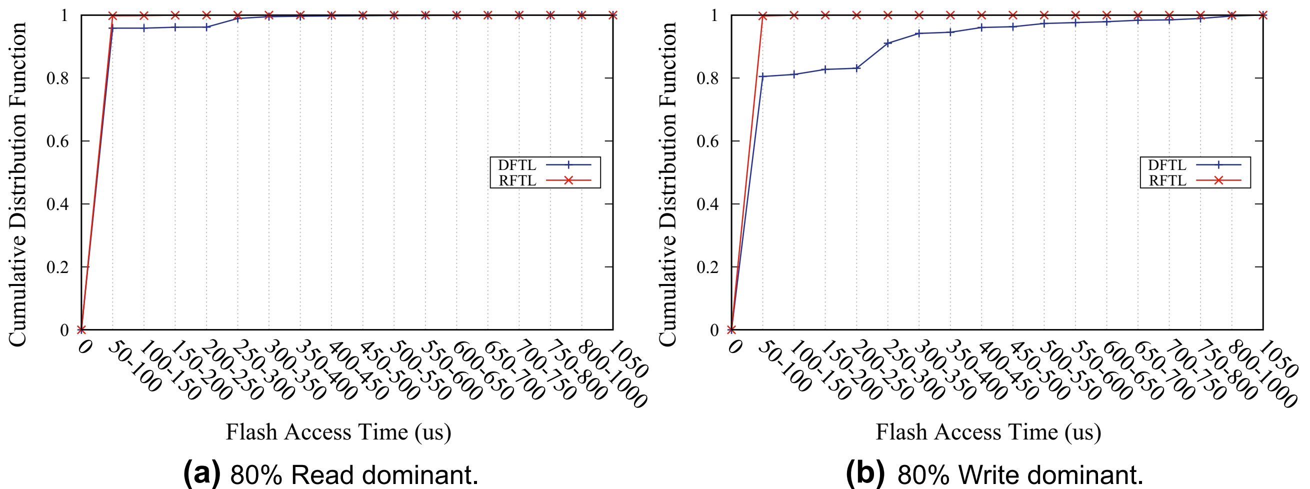
(a) 80% Read dominant. **(b)** 80% Write dominant.

Fig. 8 Comparing the response time for mapping reads of input/output operations between RFTL and DFTL



(a) 80% Read dominant. **(b)** 80% Write dominant.

Fig. 9 Comparing the response time for mapping writes of input/output operations between RFTL and DFTL



(a) 80% Read dominant. **(b)** 80% Write dominant.

Fig. 10 Comparing the cumulative distribution for mapping reads of input/output operations between RFTL and DFTL

Author's personal copy

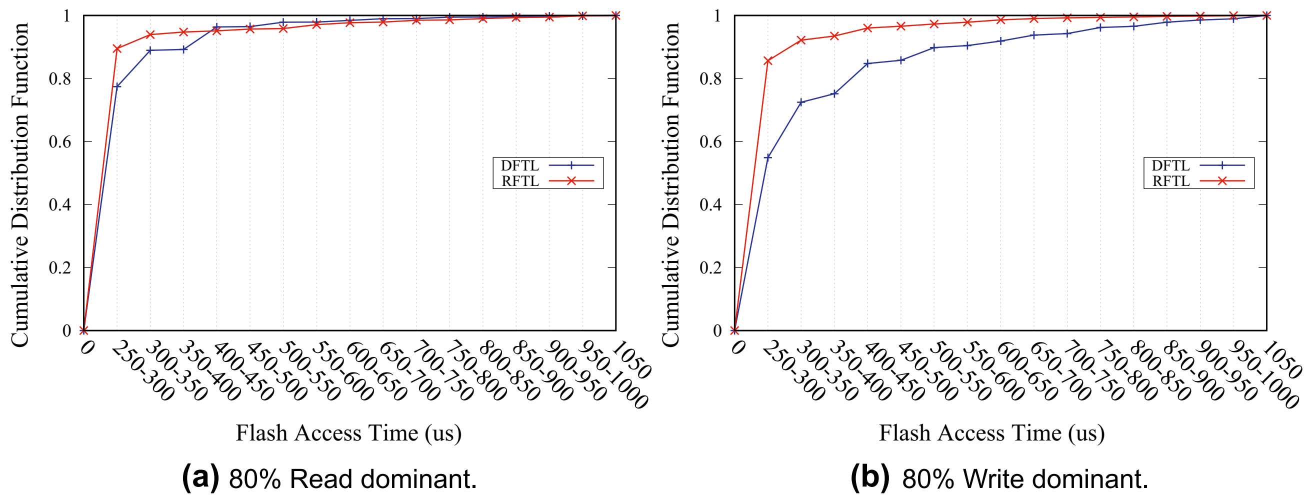


Fig. 11 Comparing the cumulative distribution for mapping writes of input/output operations between RFTL and DFTL

performance. In order to consider this, experiments were carried out while varying the SRAM size to change the miss ratio. Furthermore, we compared the actual response time for every I/O for both read and write-intensive workloads by placing them into ranges of response time against their cumulative distribution frequency (CDF). When exposed to write intensive workloads, RFTL shows extended improvement compared to DFTL. For example, from Fig. 12a we see a 100% of I/Os falling below 50 µs whereas DFTL has only around 70%. Figure 13 shows all RFTL's I/Os taking less than 250 µs for end-to-end overall writes I/O in read-dominant workload while DFTL having over 20% of I/Os going beyond 600 µs on the same Figure.

Figure 14 shows the results of RFTL and DFTL in terms of Input/output operations per second (IOPS) for various miss ratios with with read and write dominant workloads. Fig. 14a shows the results for read dominant workload. Overall we see that the RFTL approach can offer higher throughput than baseline. This is because the higher the cache miss rate, the greater the chance of using replicas for faster address translation.

Especially, we observe RFTL(1) improves total system performance by 2.5% at 0.2% miss ratio compared to DFTL and even its improvement gets greater as the miss ratio increases, for example at 0.9% miss, RFTL outperforms DFTL by almost 6%. However, The situation is different for 2 replicas (RFTL(2)) and 3 replicas (RFTL(3)). We see that the approach show reduced performance as we increase the replication factor even though they still out perform our baseline. We witness the performance improvement gets lower as the replication factor increases (RFTL(2) and RFTL(3)), even though even though they still out perform DFTL. As miss ratio increases, the gap between RFTL(3) and DFTL narrows down to

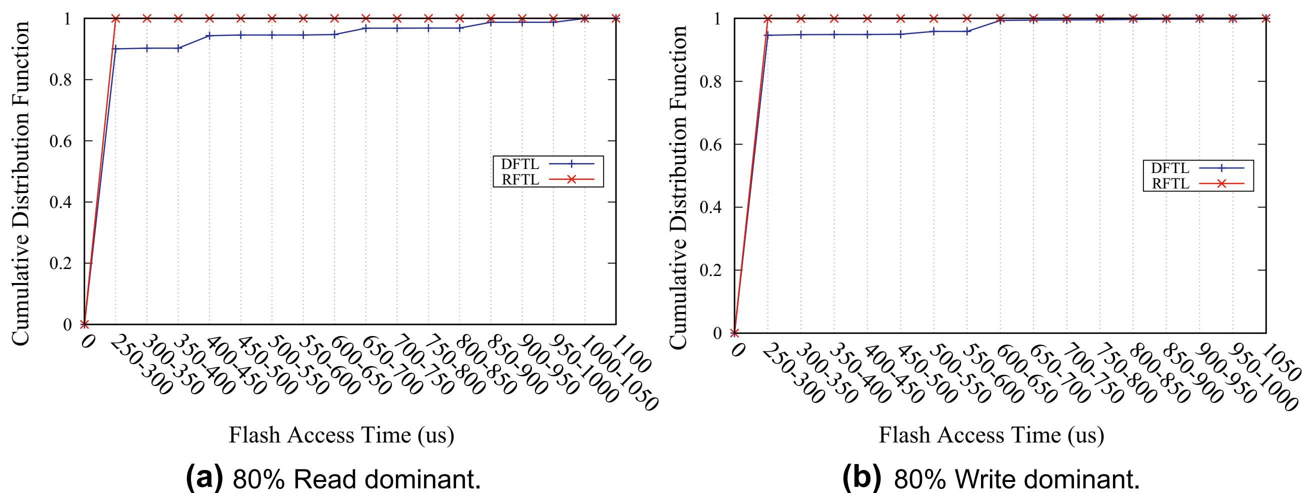


Fig. 12 Comparing the cumulative distribution for overall writes of end-to-end input/output operations between RFTL and DFTL

Author's personal copy

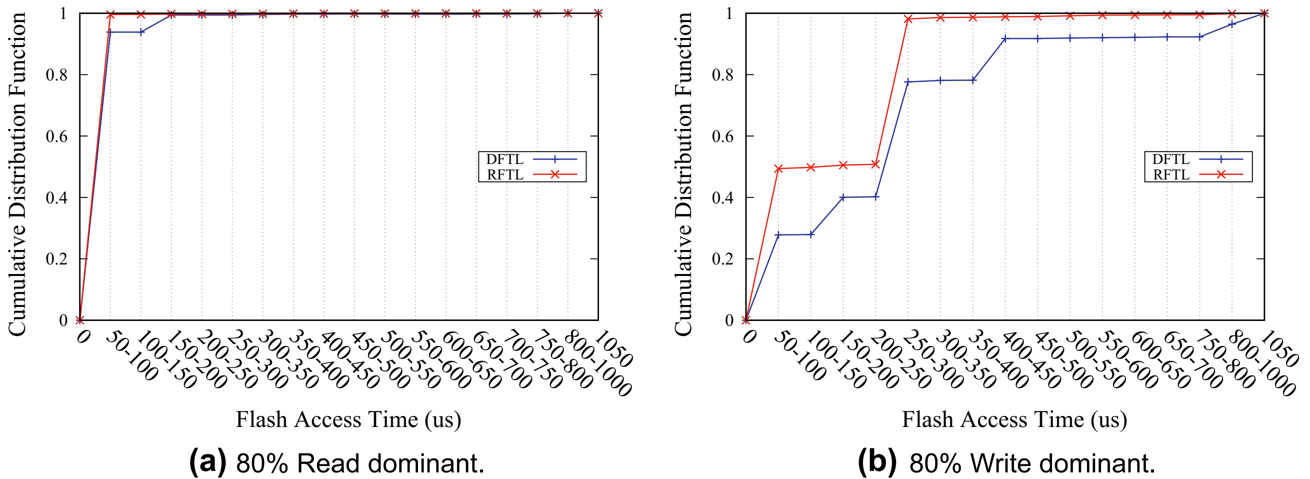


Fig. 13 Comparing the cumulative distribution for overall reads of end-to-end input/output operations between RFTL and DFTL

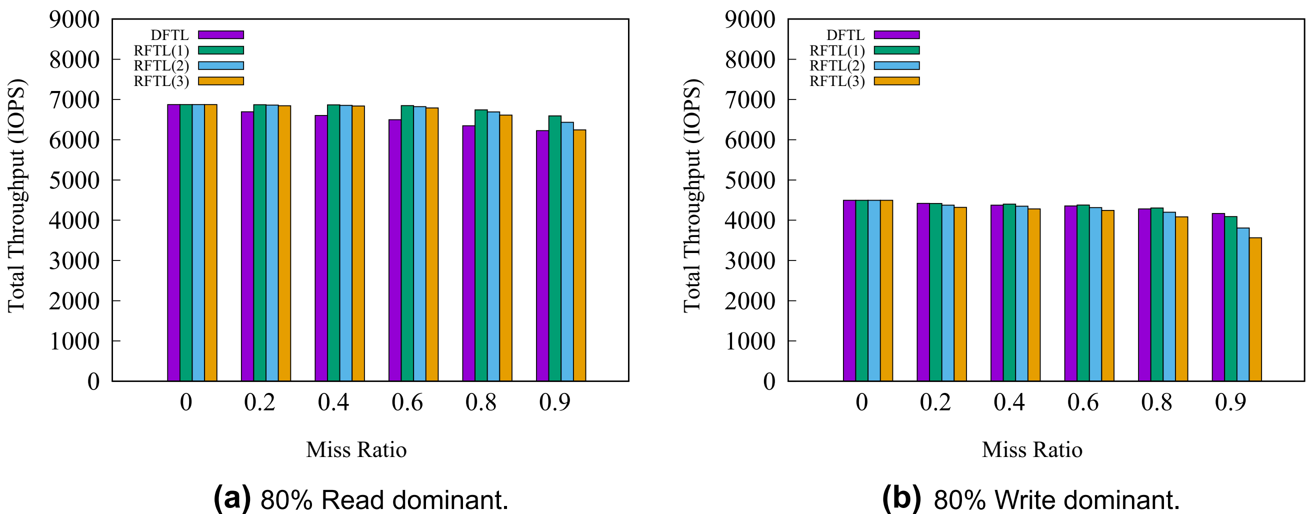


Fig. 14 Comparing input/output operations per second (IOPS) for RFTL and DFTL

only 0.3%. This is because the more replicas, the greater the total number of writes, which increases the GC number.

Figure 14b shows the results when the workload is write-dominant. Unlike read-dominant workload results, overall RFTL performance improvement is observed to be negligible. If the workload is write-dominant, the demand for write operations increases and more GCs are triggered. Excessive GC execution can result in more writes, thus reducing the opportunity to benefit from using replicas in the system when a CMT miss occurs. Comparing RFTL (1) and DFTL results, RFTL (1) shows higher performance than DFTL by 4.6% as the miss ratio increases. However, when the number of replicas increases, the IO throughput of RFTL (2) and RFTL (2) is lower than that of DFTL regardless of the miss ratio. It is because of the increased writes by the replicas causing more delay in RFTL(2) and (3) over RFTL(1).

Since the lifetime of the flash memory is limited by the number of erases, it is also important to evaluate the impact of RFTL on the NAND flash lifecycle. For this analysis, we calculated the number of mapping writes for each approach while changing the CMT miss ratio. In Fig. 15, we see that as the number of replicas increases, the number of mapping writes linearly increases. If a CMT miss occurs and the selected victim entry needs to update its original mapping page on flash memory's page-level address translation table, it issues additional writes by the number of replicas. These additional mapping writes can affect flash memory lifetime. However, the amount of mapping write is very small compared to the number of writes in the actual workload, so it will not have a significant effect on lifetime. This is not the case since and assigning a single copy (RFTL(1)) per each translation page would double this to 0.4%. If we assign 3 copies (RFTL(3)) or n-copies

Author's personal copy

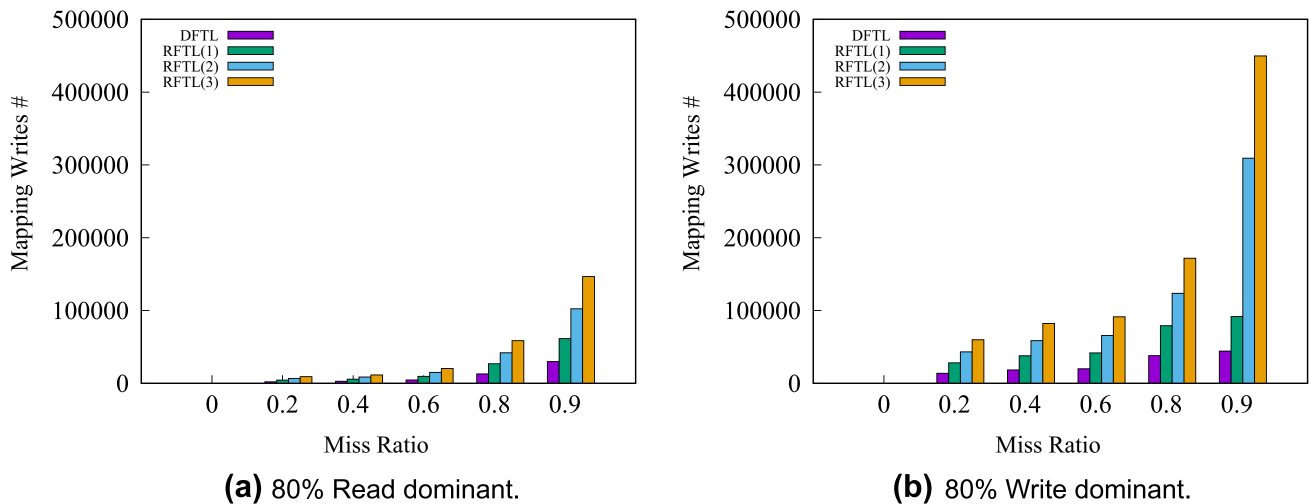


Fig. 15 Comparing the number of mapping writes (#) for RFTL and DFTL

(RFTL(N)), this means we will occupy only 0.2% × N of the total flash memory which is quit negligible.

5.2.3 Analysis

From the above mentioned results we realized that RFTL Mapping I/O performance, especially its mapping-writes performed well above Baseline under write dominant workloads as depicted by Fig. 9b. This is because RFTL takes advantage of replicas on free channels to skip GC busy channels induced by write intensive workloads. On the other hand, the overall performance results for End-to-End I/O results on Fig. 14a shows that RFTL overall performance benefit is evident when exposed to read-dominant workloads. RFTL tries to improve DFTL's miss penalty that is experienced when a mapping request is not found in cache. While trying to fetch this entry, it might face access conflicts with internal I/Os. In other words, increased read requests subsequently increases chances of cache-misses and the miss penalty of DFTL. This gives RFTL increased opportunity to utilize replicas, thus evident from Fig. 14a. Read dominant workload increase mapping requests which in-turn increases opportunity for our proposed approach.

6 Related work

A pure page mapping FTL maintains a page-level address mapping table [5] and is known as the most efficient FTL because it can store all address mappings in SRAM in flash memory [4]. However, loading a full page-level address mapping table requires huge SRAM, and as storage capacity of an SSD increases, SRAM demand increases. DFTL solved this SRAM space problem by loading only the necessary mapping entries as needed. However, DFTL can have a huge SRAM cache miss penalty issue during address translation. The performance of DFTL is highly dependent on the working-set size of the workload on SSD. This study aims to reduce the miss penalty of DFTL and to improve an overall flash memory performance by avoiding long GC queuing delay on busy channels during cache miss. To achieve high performance in an SSD, an SSD consists of multiple flash memory chips and multiple independent channels [12]. The parallel architecture in an SSD allows multiple I/Os to be processed in parallel on different channels and on different flash memory chips. This study proposes an FTL replication scheme on flash memory across different channels in the SSD to minimize the miss penalty of address translation from SRAM in DFTL.

A lot of work has been done to solve the performance issue in an SSD, caused by Garbage Collection (GC) [3, 17]. In an SSD, the erase operation takes around 1.5 m and write/program operations take around 200 µs [8, 9] whereas read operation takes less than 50 µs. When GC is triggered on SSD, many page read/write operations and block erase operations occur internally. Current work has covered the conflict between host I/O requests and internal activities even though the approaches differ from our *RFTL* [3, 16, 18]. Such interference occurs when internal activities such as GC are activated within a flash memory chip while another host I/O request is issued to the same chip before the ongoing activity is completed for example. This will therefore imply queuing the host request until chip release thereby causing long wait time if this ongoing process is GC. A GC aware request scheduling scheme is proposed by delaying the request scheduling to the chip on which the GC is being processed [18].

Author's personal copy

In order to avoid such high delay overhead by GC, several ideas to preempt GC have been proposed [3]. The work proposed the preemptive GC scheme by dividing GC process into several atomic processes, which can be delayed and prioritizing the service of upcoming host I/O requests. However, this approach has drawbacks when taking into account the reason for GC being triggered. If the incoming request is a write for example then it has to wait for GC to provide new fresh blocks since flash memory might be in short of free blocks for the incoming write operation. [16] proposed a dynamic page replication (DPR) approach through issuing conflict requests to different chips where these requests can be processed in parallel with internal activities but this approach does not consider chip idleness and moreover, is not selective to which type of requests to be replicated thereby becoming uneconomical to SSD space and lifespan.

Our work differs from the existing work because we propose to exploit the parallelism and idleness in multichannel SSD chips by using only translation page replicas to skip busy channels so as to reduce the cache-miss penalty of DFTL. Replicating only translation pages has a negligible effect on the SSD space and lifespan but reduces flash memory access time during a cache-miss thereby improving the overall system performance.

7 Conclusion

In this paper, we investigate the problem of high address translation miss penalty caused by flash memory access when the size of the working set is larger than the SRAM that stores the FTL in the existing Demand-based Flash Translation Layer (DFTL). In order to minimize the aforementioned high miss penalty of address translation, We present an idea for replicating the FTL across multiple channels (RFTL). When an address translation miss occurs from the SRAM, the address translation time can be greatly increased when the busy channel is accessed. On the other hand, RFTL can minimize the address translation time by using the address translation page on the idle channel. We evaluated the performance and lifetime of RFTL through both analytical modeling and simulation. Our evaluation results show that significant performance improvements can be achieved without significantly degrading the lifetime of the flash memory.For example, using one replica can result in a 20% performance increase.

Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2015R1C1A1A01052105).

References

1. Schroeder, B., Lagisetty, R., Merchant, A.: Flash reliability in production: the expected and the unexpected. In: 14th USENIX Conference on File and Storage Technologies (FAST 16), pp. 67–80. USENIX Association, Santa Clara (2016). http://usenix.org/conference/fast16/technical-sessions/presentation/schroeder
2. 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. **60**, 305–320 (2010)
3. Lee, J., Kim, Y., Shipman, G.M., Oral, S., Kim, J.: Preemptible I/O scheduling of garbage collection for solid state drives. In: IEEE Transaction on CAD of Integrated Circuits and Systems, vol. 32, no. 2, pp. 247–260 (2013). http://dx.doi.org/10.1109/TCAD.2012.2227479
4. Gupta, A., Kim, Y., Urgaonkar, B.: DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS XIV, pp. 229–240 (2009)
5. Ma, D., Feng, J., Li, G.: Lazyftl: a page-level flash translation layer optimized for nand flash memory. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '11, pp. 1–12. ACM, New York (2011). http://doi.acm.org/10.1145/1989323.1989325
6. Dayan, N., Svendsen, M.K., Bjørling, M., Bonnet, P., Bouganim, L.: Eagletree: exploring the design space of SSD-based algorithms. In: Proceedings of VLDB Endowment, vol. 6, no. 12, pp. 1290–1293 (2013). http://dx.doi.org/10.14778/2536274.2536298
7. Kim, Y., Gupta, A., Urgaonkar, B.: A temporal locality-aware page-mapped flash translation layer. J. Comput. Sci. Technol. **28**(6), 1025–1044 (2013)
8. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M.S., Panigrahy, R.: Design tradeoffs for SSD performance. In: Proceedings of 2008 USENIX Annual Technical Conference, Boston, MA, USA, June 22-27, pp. 57–70 (2008). http://www.usenix.org/events/usenix08/tech/fullpapers/agrawal/agrawal.pdf
9. Chen, F., Koufaty, D.A., Zhang, X.: Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In: Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems, ser. SIGMETRICS '09, pp. 181–192. ACM, New York (2009). http://doi.acm.org/10.1145/1555349.1555371
10. Park, S.Y., Seo, E., Shin, J.Y., Maeng, S., Lee, J.: Exploiting internal parallelism of flash-based SSDs. IEEE Comput. Archit. Lett. **9**, 9–12 (2010)
11. Jung, M., Kandemir, M.: Revisiting widely held SSD expectations and rethinking system-level implications. In: Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, ser. SIGMETRICS '13, pp. 203–216. ACM, New York (2013). http://doi.acm.org/10.1145/2465529.2465548
12. Park, S.-H., Ha, S.-H., Bang, K., Chung, E.-Y.: Design and analysis of flash translation layers for multi-channel nand flash-based storage devices. IEEE Trans. Consum. Electron. **55**(3), 1392–1400 (2009)
13. Mativenga, R., Paik, J., Lee, J., Chung, T., Kim, Y.: Minimizing CMT miss penalty in selective page-level address mapping table. In: 2016 IEEE International Conference on Cluster Computing, CLUSTER 2016, Taipei, Taiwan, September 12-16, pp. 152–153 (2016). http://dx.doi.org/10.1109/CLUSTER.2016.81
14. Kawaguchi, A., Nishioka, S., Motoda, H.: A flash-memory based file system. In: Proceedings of the USENIX 1995 Technical

Author's personal copy

Conference Proceedings, ser. TCON'95, pp. 13–13. USENIX Association, Berkeley (1995)
15. Karedla, R., Love, J.S., Wherry, B.G.: Caching strategies to improve disk system performance. IEEE Comput. **27**(3), 38–46 (1994)
16. He, B., Yu, J.X., Zhou, A.C.: Improving update-intensive workloads on flash disks through exploiting multi-chip parallelism. IEEE Trans. Parallel Distrib. Syst. **26**(1), 152–162 (2015)
17. Lee, J., Kim, Y., Shipman, G.M., Oral, S., Wang,F., Kim, J.: A semi-preemptive garbage collector for solid state drives. In: ISPASS, pp. 12–21. IEEE Computer Society, Washington, DC (2011). http://dblp.uni-trier.de/db/conf/ispass/ispass2011.html#LeeKSOWK11
18. Mao, B., Wu, S.: Exploiting request characteristics and internal parallelism to improve SSD performance. In: 2015 33rd IEEE International Conference on Computer Design (ICCD), vol. 00, pp. 447–450 (2015)

Ronnie Mativenga received his B.S.C Honors degree in Computer Science from National University of Science and Technology (NUST), Bulawayo, Zimbabwe in 2010. He is currently a PhD student in Computer Engineering at Ajou University, Suwon, Korea. His current research interests include All-Flash Array storage, building a high-performance, reliable all Flash (SSD) based storage system, non-volatile memory systems, large database systems and simulation tools.

Joon-Young Paik received the B.S., M.S., and Ph.D. degrees in computer science and engineering from Chungnam National University, Daejeon, South Korea, in 2008, 2010, and 2013, respectively. He is currently a Research Professor with the Department of Software, Ajou University, Suwon, South Korea. His current research interests include flash memory storages and storage security.

Youngjae Kim received his Ph.D degree in Computer Science and Engineering from Pennsylvania State University, University Park, PA, USA in 2009. He is currently an assistant professor in the department of computer science and engineering at Sogang University, Seoul, Republic of Korea. Before joining Sogang University, Dr. Kim was a staff scientist in the US Department of Energy's Oak Ridge National Laboratory (2009–2015) and an assistant professor in Ajou University, Suwon, Republic of Korea (2015–2016). Dr. Kim received the BS degree in computer science from Sogang University, Republic of Korea in 2001, and the MS degree from KAIST in 2003. His research interests include distributed file and storage, parallel I/O, operating systems, emerging storage technologies, and performance evaluation.

Junghee Lee received his B.S. and M.S. degrees in Computer Engineering from Seoul National University in 2000 and 2003, respectively, and his Ph.D degree in Electrical and Computer Engineering at Georgia Institute of Technology in 2013. From 2003–2008, he worked at Samsung Electronics on electronic system level design of mobile system-on-chip. Since 2014, he is with the Department of Electrical and Computer Engineering of University of Texas at San Antonio as an assistant professor. His research interests include architecture design of microprocessors, memory hierarchy, and storage systems for high performance computing and embedded systems.

Tae-Sun Chung received the B.S. degree from KAIST, Daejeon, South Korea, in 1995, and the M.S. and Ph.D. degrees from Seoul National University, Seoul, South Korea, in 1997 and 2002, respectively, all in computer science. He is currently a Professor with the Department of Software, Ajou University, Suwon, South Korea. His current research interests include flash memory storages, XML databases, and database systems.