Exploiting Chip Idleness for Minimizing Garbage Collection Induced Chip Access Conflict on SSDs

CONGMING GAO, College of Computer Science, Chongqing University, Chongqing, P.R. China
LIANG SHI, College of Computer Science, Chongqing University, Chongqing, P.R. China
YEJIA DI, College of Computer Science, Chongqing University, Chongqing, P.R. China
QIAO LI, College of Computer Science, Chongqing University, Chongqing, P.R. China
CHUN JASON XUE, Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong
KAIJIE WU, College of Computer Science, Chongqing University, Chongqing, P.R. China
EDWIN SHA, College of Computer Science, Chongqing University, Chongqing, P.R. China

Solid State Drives (SSDs) are being widely deployed in mobile devices, personal computers, data centers, and cloud storages. Different from hard disk drives, SSDs are normally constructed with a number of parallel-accessible flash chips, where host I/O requests can be processed in parallel. In addition, there are many internal activities in SSDs, such as garbage collection and wear leveling induced read, write and erase operations, to solve the issues of inability of in-place-updates and limited lifetime. When internal activities are triggered on a chip, the chip will be blocked. Our preliminary studies on several workloads show that when internal activities are frequently trigged, the host I/O performance will be significantly impacted because of the *access conflict* between them. In this work, in order to improve the access conflict induced performance degradation, a novel *access conflict* minimization scheme is proposed. The basic idea of the scheme is motivated by an interesting observation in SSDs: several chips are idle when other chips are busy with internal activities and host I/O requests. Based on this observation, we propose to schedule internal activities induced operations for minimized access conflict by exploiting the idleness of the multiple chips of SSDs. This approach is realized by two steps: First, read internal activities accessed data to the controller; Second, by exploiting the idle chips during internal activities, write internal activities accessed data back to these idle chips. With this scheme, the internal activities can be processed with minimized access conflict to the host requests. Simulation results show that the proposed approach significantly reduces the access conflict, and in turn leads to a significant performance improvement of SSDs.

CCS Concepts:•**Computer systems organization** → **Embedded software;** *Architectures; Parallel architectures;*•**Hardware** → External storage;

Additional Key Words and Phrases: Garbage Collection, Chip Idleness, Access Conflicts, Parallelism, SSD

ACM Reference Format:
Congming Gao, Liang Shi, Yejia Di, Qiao Li, Chun Jason Xue, Kaijie Wu, and Edwin Sha 2016. Exploiting Chip Idleness for Minimizing Garbage Collection Induced Chip Access Conflicts on SSDs. *ACM Trans. Des. Autom. Electron. Syst.* 9, 4, Article 39 (March 2010), 29 pages.
DOI: 0000001.0000001

1. INTRODUCTION

NAND flash memory based solid state drives (SSDs) are widely deployed in mobile devices, personal computers, data centers, and cloud storages, thanks to its well-

This work is partially supported by the Fundamental Research Funds for Graduate Scientific Research and Innovation Foundation of Chongqing, China (Grant No.CYB14043), the Fundamental Research Funds for the Central Universities (106112016CDJZR185512), NSFC (61402059, 61472052, and 61572411), and National 863 Programs 2015AA015304.

DOI: 0000001.0000001

identified advantages, such as high density, high random access performance, low power consumption, and light-weight form factors [Agrawal et al. 2008]. SSDs are normally constructed with a parallel architecture, which includes multiple parallel channels with each channel containing multiple NAND flash chips, each chip containing multiple dies, and each die containing multiple planes. Each plane is further organized with multiple blocks and each block has a fixed number of pages [Agrawal et al. 2008][Chen et al. 2011a][Dirik and Jacob 2009] [Hu et al. 2011][Jung et al. 2012b][Shin et al. 2009]. The rich parallelism offered by the parallel architecture of SSDs is the major contributions to the performance since host I/O requests can be processed simultaneously. On the other hand, due to the inability of out-of-place updates and limited lifetime features of flash memory, there are always a large amount of internal activities, such as garbage collection (GC) and wear leveling (WL), which would induce several internal read, write, and erase operations. GC is designed to reclaim the invalid space of SSDs [Hu et al. 2011][Chang and Wen 2014][Chang et al. 2004][Jung et al. 2012a]. WL is designed to level the wears of flash memory [Hu et al. 2011][Murugan and Du 2011][Pan et al. 2013][Chen et al. 2015]. Both these two schemes would induce several read, write and erase operations, where GC has been identified as the dominate source of internal activities and it is able to highly reduce the performance of SSDs [Mao and Wu 2015][Shahidi et al. 2016][He et al. 2015][Lee et al. 2013]. When internal activities are triggered, host I/O requests accessing these chips occupied by internal activities are going to be blocked and delayed. In this case, even though rich parallelism is provided by SSD inherent architecture, the performance of SSD is significantly degraded due to the internal activities induced chip blocking. In this work, we will propose approaches to solve the access conflict induced performance degradation.

Previous work has been proposed to identify the negative effects of these internal activities [Agrawal et al. 2008][Chang and Wen 2014][Chen et al. 2009][He et al. 2015][Shahidi et al. 2016]. In order to clearly identify the effects of these internal activities, we have done experiments to evaluate the impact from internal activities. The details of the experiment setups are presented in Section 5. Figure 1 presents results on the request access latency increase between with and without internal activities impact for host I/O requests. Compared to the case without internal activities, where internal activities take zero latency [Shahidi et al. 2016], the read and write latencies are degraded by 4.7 times and 2.9 times, respectively. With the increasing access intensity of modern workloads, the internal activities of SSDs have been a key challenge for the performance improvement [Mao and Wu 2015][Lee et al. 2013][He et al. 2015]. In this work, we propose approaches to reduce the performance impact from internal activities.

Recently, access conflict between internal activities and host I/O requests has been widely studied [Mao and Wu 2015][Lee et al. 2013][He et al. 2015]. Among the different types of internal activities, GC induced internal operations have been identified as the dominant source. GC always includes several reads, writes and a block erase operation, and is activated when there are no enough free space for data writes. Lee et al. [Lee et al. 2013] found that GC induced internal activities are dominated, and GC induced access conflicts commonly exist during the runtime of SSDs. Based on this feature, they proposed a preemptive GC scheme to prevent host I/O requests from being conflicted by GC processes. When a host request is issued on a chip where a GC operation is ongoing, the controller preempts the GC process and switches to serve the pending host I/O request in advance. With this approach, the GC activities induced access conflicts can be minimized. However, the large amount of GC induced internal activities are still pending in the chip, which would cause new conflict to the following host requests to the chip. Mao et al. [Mao and Wu 2015] proposed a GC-aware request scheduling scheme, which is designed to prioritize the requests to the chips without GC activities.

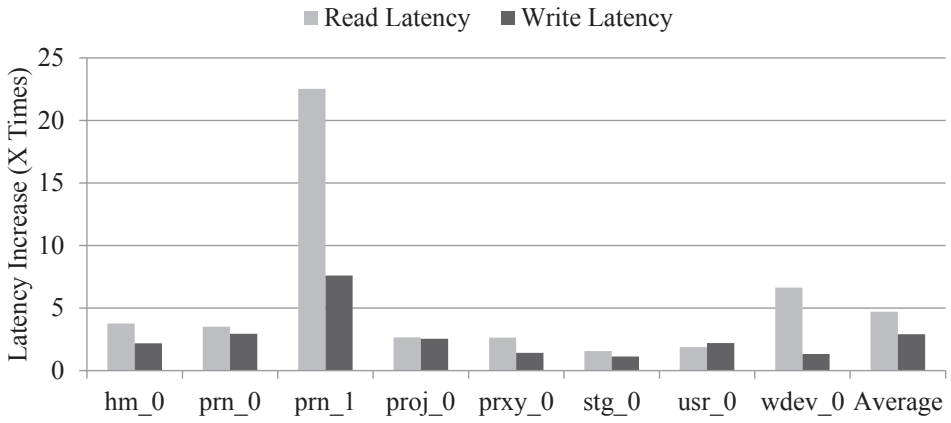


Fig. 1. The performance impacts resulted from the internal activities induced access conflict.

With this approach, access conflict between host I/O requests and GC activities can be avoided through delaying the requests issued to the chips with GC activities. He et al. [He et al. 2015] proposed a dynamic page replication (DPR) approach, which replicates the conflict pages in multiple chips in advance. With this approach, the access conflict between host I/O requests and internal activities can be avoided by redirecting the conflict requests to other chips, where these I/O requests can be processed in parallel with internal activities. These work tried to minimize the conflict through avoiding the access to these chips with GC activities passively. However, the time-consuming GC activities are still potential in inducing access conflicts with upcoming host I/O requests.

In this work, only GC induced access conflict is taken into consideration since GC has been identified as the dominating sources of internal activities. We propose a novel GC-induced access conflict minimization scheme for SSDs. This work is motivated by an observation that there are a large amount of idle chips during GC process. Based on this observation, the basic idea of the proposed scheme is to smartly schedule the activities in GC processes to exploit the rich parallelism and idleness among the flash chips on an SSD. In order to realize the basic idea, the proposed scheme is designed in two steps: First, a small buffer is maintained in the SSD controller to hold the valid pages from the to-be-collected block when a GC process is activated. The buffer can be a partition of the RAM in the SSD controller. With the first step, these valid pages transferred to the buffer can be scheduled to other parallel chips more flexible. Since read operations are much faster than write operations, reading the valid pages into the buffer is fast and can minimize the access conflict resulted from the time-consuming write operations. After valid pages in the to-be-collected block are transferred to the controller buffer, the block can be erased, and then the occupied chips can be released immediately for serving upcoming host I/O requests. Second, the valid pages read into the buffer are then written back to chips by exploiting the parallelism and idleness available in the SSD. Note that, the number of chips in an SSD, which is the foundation of the parallelism of an SSD, has been increased steadily over generations. According to recent work [Fusionio 2015], the most recent SSDs are equipped with more than 128 chips. Hence, based on the combination of parallelism and idleness of SSDs, the GC induced access conflict can be minimized for performance improvement.

With the above two-step design, the GC-induced access conflict can be significantly reduced. This is also confirmed by the results obtained from extensive experiments shown in Section 5. In summary, this work makes the following contributions:

— Verified that GC-induced access conflict leaves a negative impact on the performance of SSDs;
— Proposed a minimization scheme targeting GC-induced access conflict to improve the host I/O performance;
— An efficient implementation of the proposed approach with little overhead in the controller is presented. Experimental results show that the proposed approach is effective in reducing the access conflict and is able to achieve significant performance improvement.

The rest of the paper is organized as follows. Section 2 depicts the background and the most related work. Section 3 presents the motivation. Section 4 presents the proposed approach. Experiments are presented in Section 5. Finally, Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

In this section, the background of SSDs and related work are presented. First, the background of SSD organization and GC process is introduced. Then, the most related work is presented.

2.1. Background

2.1.1. SSD Organization. SSDs are constructed with multiple channels, and each channel is connected with multiple flash chips. Each chip is constructed with multiple dies, and each die is further composed of multiple planes. Figure 2 shows an example of the parallel architecture of SSD. This SSD is equipped with 4 channels, 2 chips per channel, 2 dies per chip, and 2 planes per die. This four level parallelism has been widely studied and exploited for performance improvement [Chen et al. 2011a][Gao et al. 2014][Gao et al. 2017][Hu et al. 2011]. Within each level of parallelism, host I/O requests can be processed independently. For the four levels of parallelism, the die and plane level parallelism are called internal parallelism, which are enabled by the support of advanced commands [Hu et al. 2011]. 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. Due to the non-transparent feature of SSDs [Agrawal et al. 2008], this four levels of parallelism are hard to be fully exploited at host system side. Hence, the re-design of controller is a straightforward method for exploiting the rich parallelism of SSDs as much as possible.

In the SSD controller, there are several important components, including flash translation layer (FTL), data allocation, WL and GC. With the equipment of these components, all issues of SSDs are well dealt with. FTL is designed to maintain mappings between logical addresses and physical addresses. When data are programmed or updated, the new or updated mapping entries are recorded in the mapping table for future accesses. Data allocation is designed to manage the flash array of SSDs and decides the distribution of the data. Hence, access conflict between internal activities and host I/O requests is highly related with the data allocation of SSDs, which determines the access location of each I/O request. In order to fully exploit the parallelism of SSDs, several data allocation schemes have been studied in [Hu et al. 2011][Jung and Kandemir 2012][Shin et al. 2009][Bucy et al. 2008]. In this work, the default data allocation applied in [Hu et al. 2011][Bucy et al. 2008] is adopted. GC is used to reclaim invalid pages due to the out-of-place feature of SSDs. WL is designed to wear blocks evenly for prolonging the lifetime of SSDs. NAND flash cell is a floating gate transistor, which can be worn by program and erase operations. In order to prolong the lifetime of SSDs, WL is in charge of programming data to flash chips evenly.

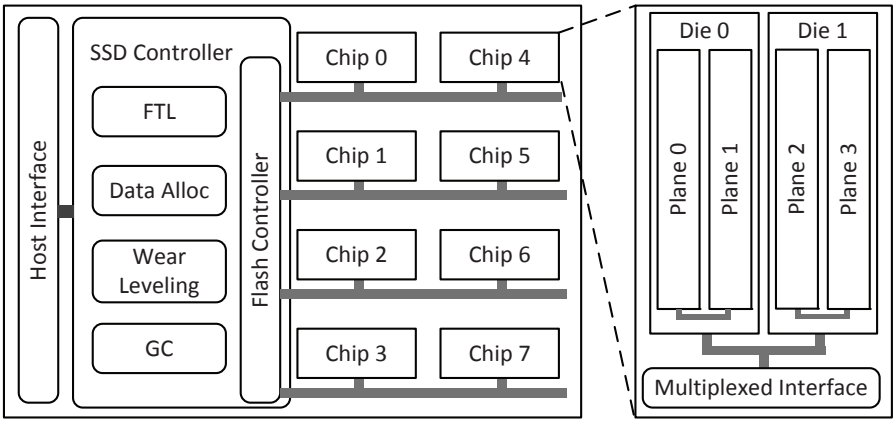


Fig. 2. Parallel Architecture of SSDs.

2.1.2. Garbage Collection in SSDs. GC is one of unique components of SSDs. Due to the out-of-place update feature of SSDs, GC is designed for reclaiming invalid pages [Agrawal et al. 2008]. A flash page can be programmed only when its residing block has been erased. After programming data into a flash page, the state of this page is switched from free to valid. Once there exist update operation on this valid page, its state will be switched from valid to invalid. Hence, when there are not enough free space for data updating or programming, GC is triggered to reclaim these invalid pages. If there are valid pages in the target block during GC, several read and program operations are created to copy them to free pages in other blocks [Agrawal et al. 2008][Chiang et al. 1999][Lee et al. 2013][Min et al. 2012][Shahidi et al. 2016]. Take Figure 3 as an example. Assume that each flash block has 4 pages, and totally 4 blocks reside in the flash chip. When GC is triggered, the block, block #1, with the largest number of invalid pages is selected for GC. Note that, the greedy GC scheme, which takes the block with the largest number of invalid page as the target block, is adopted for space reclaim in this work [Agrawal et al. 2008][Hu et al. 2011][Bucy et al. 2008]. Then, these two valid pages in block #1 are read, transferred and programmed to the free pages of block #3. Finally, block #1 is erased. During this process, chips held these blocks reject servicing host I/O requests until the completion of GC.

From the GC process, we can find that the whole GC process is composed of four steps: The first one is reading valid pages from the target block; The second one is transferring these valid pages to free blocks; The third one is programming valid pages to free pages of other blocks; The last one is erasing the to-be-collected block, which is executed after all valid pages have been programmed to other free pages. Due to the asymmetry of read, write and erase operations, the time cost of read operation is negligible compared with program and erase operations. That is, the cost of program and erase operations dominates the total cost of GC process. In addition, the cost of GC process is highly related with the number of valid pages of target block. The larger the number of valid pages is, the larger the cost of GC process will be. This is the reason why the greedy GC scheme is applied in this work for minimizing the cost of GC process. In order to identify the costs of these four parts, an experiment on the cost of each part is measured on a well configured SSD, where the details are presented in Section 5. As shown in Figure 4, the cost of program operation is the dominated one, which directly results in high cost of GC process and poor performance of SSDs verified

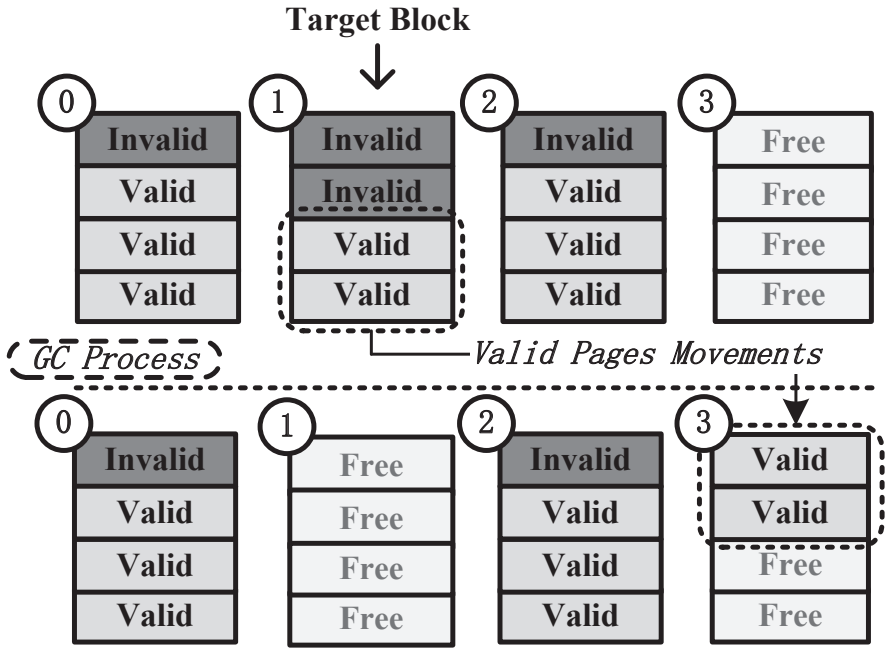


Fig. 3. The process of garbage collection.

in Figure 1. This result tells us that optimizing the cost of program operations is the key challenge of the proposed approach.

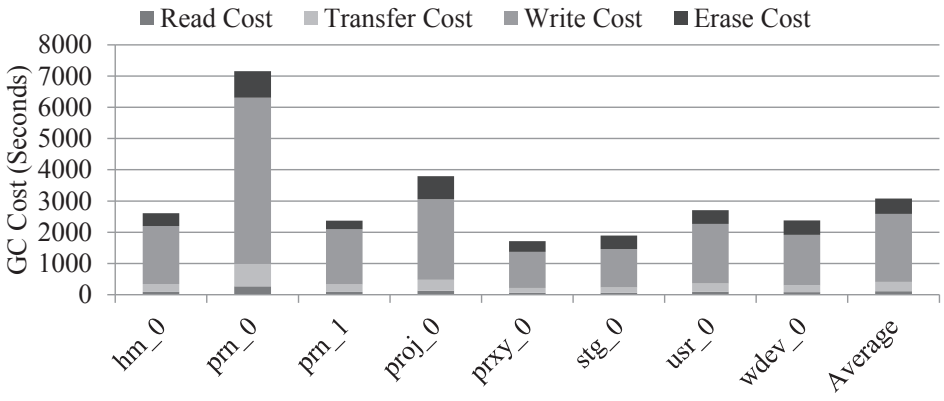


Fig. 4. The cost of garbage collection.

2.2. Related Work

In this section, the related work regarding the minimization of access conflict on SSDs is presented. Access conflict on SSDs has been identified and well studied in previous work. Currently, there are at least two types of access conflict having been identified: access conflict among host I/O requests [Chen et al. 2012][Jung et al. 2012b][Gao et al.

2014] and access conflict between host I/O requests and internal activities [Lee et al. 2013][Mao and Wu 2015][He et al. 2015].

2.2.1. Access Conflict among Host I/O Requests. Access conflict among host I/O requests happens when several host requests are issued to the same chip within a short time interval. In this case, latter requests need to wait for the finish of the front requests. Considering this type of access conflict, several work has been proposed. Chen et al. [Chen et al. 2012] observed that latency of host I/O requests is highly related with the state of chips. If a chip is occupied, these host I/O requests issuing to this chip are going to be delayed for a while. Hence, they proposed a cache management approach for SSDs to wisely buffer data resided in frequently accessed chips. Jung et al. [Jung et al. 2012b] proposed a source conflict aware request scheduling scheme, physical address queueing under flash translation layer. That is, before scheduling host I/O requests in the SSD controller, the physical accessing locations of I/O requests have been obtained. With this information, more host I/O requests can be issued to parallel chips without conflict. Similarly, Gao et al. [Gao et al. 2014] proposed a parallel issuing queueing scheme at the host side, where the conflict information can be obtained with considering the logical addresses and mapping scheme in advance. With this approach, host I/O requests are scheduled and batched without access conflict. Each batch can fully utilize the parallelism of SSDs. Different with the above methods, Elyasi et al. [Elyasi et al. 2017] proposed to schedule sub-requests for reducing the latency variation among several sub-requests belonging to the same host I/O requests. With this approach, the access conflict among sub-requests can be used to improve the performance of SSDs. However, none of these work take the large amount of high cost internal activities of SSDs into consideration.

2.2.2. Access Conflict between Host I/O Requests and Internal Activities. Access conflict between host I/O requests and internal activities happens when internal activities are activated within a chip when there exists a host I/O request which is going to be issued to this chip within a short time interval. In this case, the host request is blocked until the release of chips occupied by internal activities. Considering this type of access conflict, several work has proposed. Lee et al. [Lee et al. 2013] 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. Mao et al. [Mao and Wu 2015] proposed a GC aware request scheduling scheme through delaying the requests scheduling to the chip where GC is being processed. He et al. [He et al. 2015] 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. However, the time-consuming GC activities are still potential in access conflict with upcoming host I/O requests. In this work, we also proposed approach to minimize the access conflict between host I/O requests and internal activities. Differing from the existing work, we propose to exploit the parallelism and idleness of multiple chips of SSDs to reduce the costs resulted from the internal activities so that internal activities induced access conflict can be minimized. Recent SSDs have employed hundreds of flash chips [Agrawal et al. 2008]. If there are idle flash chips and the time-consuming GC process is activated, the access conflict can be minimized with little overhead by smartly processing the GC activities through exploiting the parallelism and idle flash chips.

3. MOTIVATION

Based on previous studies, GC activities have been identified as the dominant reason for the occupation of chips. If the time-consuming GC activities can be optimized, the GC induced access conflict can be reduced. In addition, we also understand that

current SSDs have been equipped with a large amount of flash chips, which can be accessed in parallel. Based on this understanding, we are thinking about optimizing the GC process by exploiting the multiple chips to minimize the GC induced access conflict. However, this is challenge since GC process is blind with the potential parallelism and idleness of flash chips in SSDs. In this section, we present a motivation in fully utilizing flash chips for GC activity optimization.

Figure 5 is a motivation example. In this figure, we assume there are four chips equipped for parallel request processing. In current state, there are three idle chips #1, #2 & #3 and one busy chips #0 with GC activated. Assume there are several requests currently pending in the I/O queue, where R 0 & 1 access chip #0 & #1, W 0 & 1 access chip #1 & #2. In this case, W 0 & 1 can be issued to chip #1 & #2. While, R 0 is blocked and delayed until the GC releases the occupation of this busy chips. In this example, we find that there is an idle chip, #3. If we exploit the idle chip to speed up GC process, the conflict requests can be issued to chips with less waiting time and the performance can be improved.

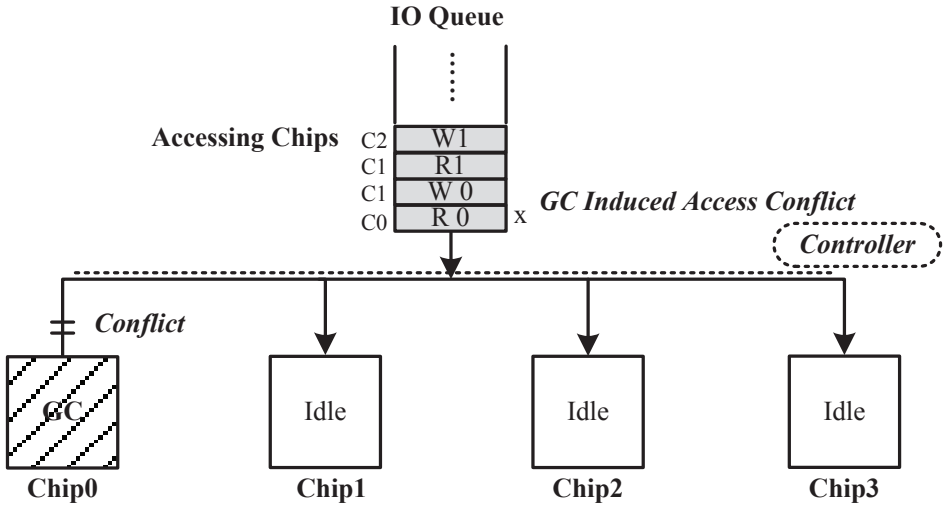


Fig. 5. An example of GC activities induced access conflicts.

In order to understand the potential in exploiting idle chips for GC process, we have evaluated two metrics: the percentages of GC activated during idle period, and the percentages of idle chips when GC is activated and host I/O requests are pending. On the one hand, the first metric shows the maximum potential for exploiting the idleness of flash chips, where all flash chips are idle, except the GC occupied one. On the other hand, when there exist host I/O accesses, the percentages evaluated by second metric shows the rest potential for exploiting the idle flash chips. These two metrics are evaluated on a well configured SSD with 16 channels, 1 chips in each channel. The detailed experiment settings can be found in Section 5. Figure 6 shows the experimental results on a set of block I/O traces of enterprise servers at Microsoft Research Cambridge [Narayanan et al. 2009]. On average, only 17.7% GC activities are activated during idle period and more than 21% of chips are idle when host I/O requests are accessing.

From the experiment results, we can find that a part of GC activities are activated during idle period and there are a large amount of idle chips during GC activities when

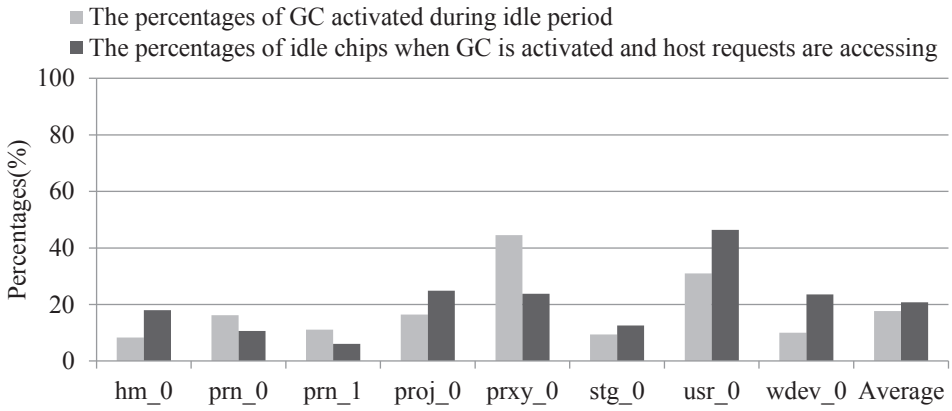


Fig. 6. Percentages of GC activated during idle period and idle chips when GC is activated and host I/O requests are accessing.

these exist host I/O request accesses. If these idle chips are exploited for GC activities, the access conflict between GC induced internal activities and host I/O requests can be minimized so that these host I/O requests accessing occupied chips can be serviced with less waiting time. This observation motivates us to design a GC activities induced access conflict minimization approach with considering the parallelism and idleness of flash chips.

4. GARBAGE COLLECTION INDUCED CHIP ACCESS CONFLICTS MINIMIZATION

In this section, we propose approaches to exploit the idle chips to minimize the conflict between GC activities and host I/O requests. Based on the studies in the motivation, we can find that there are a large amount of idle chips in the SSDs during GC process. These idle chips can be potentially exploited in reducing the cost of GC processes. However, exploiting them for GC process is challenging, which requires the re-design of the process of traditional GC scheme. Traditional GC scheme is designed to write valid pages resided in the to-be-collected block to another free block sequentially, and this valid page movement process is time-consuming. Hence, a direct method for reducing the cost of GC process is to move the valid pages from the to-be-collected block to idle chips when access conflicts between GC activities and host I/O requests occur. However, writing back valid pages to an idle chip may introduce new conflict when there exist new host I/O requests, which arrive before the completion of writing valid pages to this idle chip. In order to overcome the challenges and realize the basic idea, there are two steps: first, transferring valid pages from the to-be-collected block to the SSD controller, and second, writing the valid pages back to SSDs by exploiting the parallelism and idleness of flash chips. Once all valid pages from the same GC process have been transferred into SSD controller, the erase operation is executed for reclaiming the space of to-be-collected block. With this approach, GC occupied chips can be released as soon as possible, and access conflicts between GC activities and host I/O requests can be minimized as well. Figure 7 is an overview for the two steps, which will be detailed in the following.

4.1. Transfer Valid Pages to SSD Controller

For the first step, the valid pages from the to-be-collected block are transferred to the SSD controller. In order to accommodate the valid data, a buffer is equipped in the controller. For maintaining valid pages in buffer, a buffer management policy is

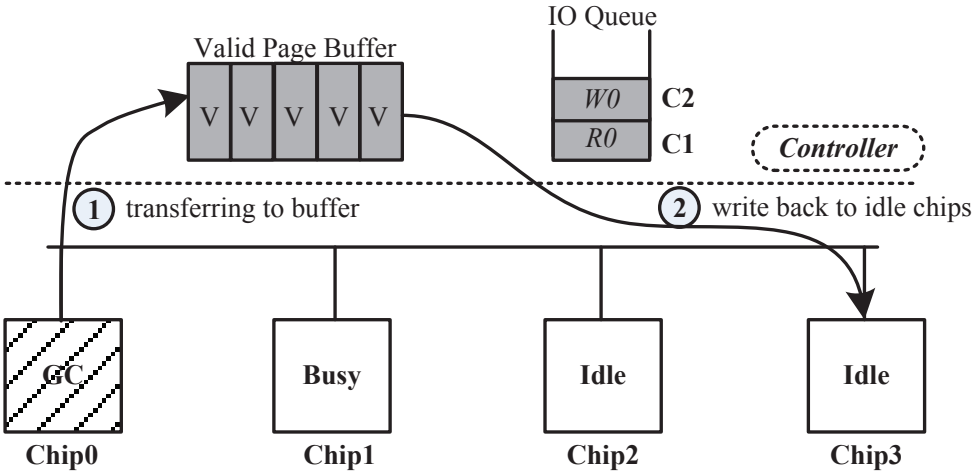


Fig. 7. Overview of the proposed approach.

designed. During the process of transferring valid pages to buffer, since read operations of flash memory are much faster than program operations, transferring the valid pages to the buffer can be quite fast. Once the valid pages are read and transferred into SSD controller, the to-be-collected block can be erased and the occupied chip will be released. Then, pending requests, which are going to access the chip, can be issued to this chip without conflict. Figure 7 shows an example on the first step. Note that there could be more than one chip in GC state. In this case, all valid pages from these to-be-collected blocks are transferred to the valid page buffer in parallel.

There are several concerns for the buffer design. For the equipment of the valid page buffer, first, the buffer can be a newly designed buffer or a shared partition with an existing data buffer in the SSD controller. Currently, data buffer has been a common component in the SSD controller [Kim and Ahn 2008][Shi et al. 2011], especially for high performance SSDs. In this case, we do not need to add new buffer in the controller. Second, the size of the buffer is closely related with the relaxation of access conflict. If the capacity of buffer is too small, few valid pages can be held in the buffer. Otherwise, if the capacity of buffer is too large, there may exist many empty slots which can not be fully utilized and induce waste resources. In this work, we propose to use a small buffer. If the buffer is full and more valid pages are needed to be transferred, these valid pages are supposed to be processed via traditional GC scheme without transferring them to valid page buffer. In the experiment section, we will evaluate the percentages of valid pages transferred to the buffer and sensitiveness of the proposed approach on the buffer size. Thirdly, the reliability issue of valid page buffer is also taken into consideration. If the block is erased and the system is crashed or suddenly powered off, the valid data in the buffer would be lost. In order to solve this issue, capacitor supported buffer can be used to hold the valid pages since capacitor has been widely used in modern SSDs [Kang et al. 2014][Zheng et al. 2013][Wikipedia 2010][Chen et al. 2011b]. Once system crashes unexpectedly, all data can be programmed back with the charge stored in the internal capacitor. In addition, a small piece of emerging non-volatile memories, such as Phase Change Memory (PCM) [Wang et al. 2015] or Spin Torque Transfer memory (STT-RAM) [Wen et al. 2014], which have been widely studied during the last decades, can also be used as the buffer to avoid the data loss.

Figure 8 shows the overview of proposed valid page buffer. There exist three key components, including *Valid Page Movement Streams*, *Checking Buffer*, and *Writing Back Policy*.

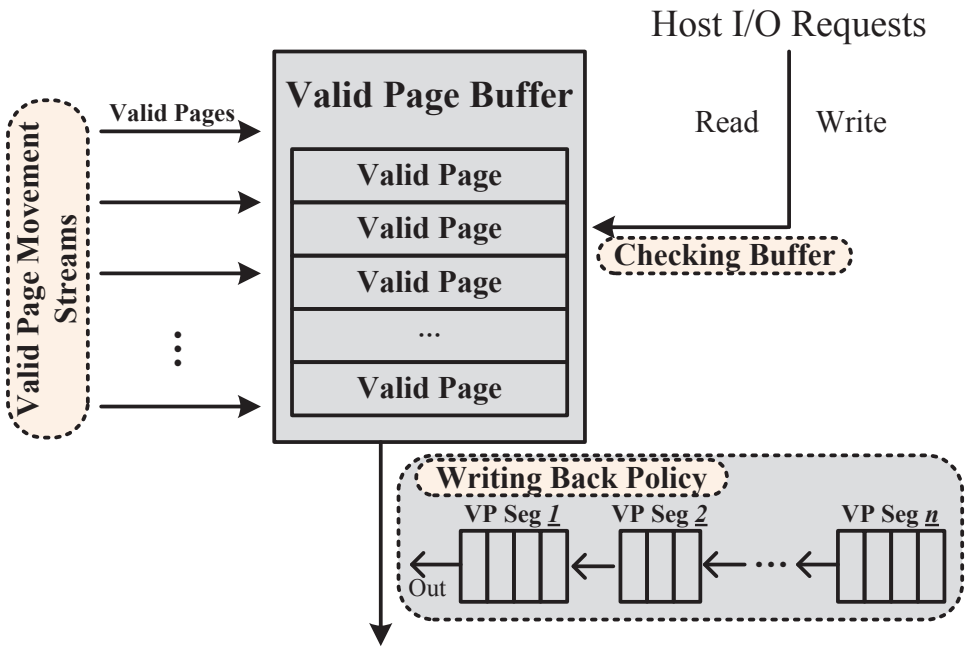


Fig. 8. Overview of the proposed valid page buffer management policy.

4.1.1. Valid Page Movement Streams. If there are empty slots in the valid page buffer for receiving valid pages, the valid pages can be transferred into this buffer continually. Otherwise, these valid pages, which still reside in the to-be-collected block, are going to be processed via traditional GC scheme without transferring to valid page buffer.

During the process of transferring valid pages to the buffer, there may exist multiple GC processes. In this case, there will be several valid page movement streams, where each stream is responsible for transferring valid pages from one GC process. If the space of valid page buffer is not large enough for holding all valid pages from current GC processes, the interference among GC processes will directly affect the exploiting of rich parallelism and idleness of flash chips. This is because that GC occupied chips cannot be released in time since a part of valid pages still reside in the to-be-collected block. Hence, we suggest that each GC process should obtain the space of valid page buffer profitably and greedily. Two rules are designed to regulate the valid page movement streams.

— Only GC process resulting in access conflict between GC process and host I/O requests is regarded as valid page movement stream.
— Once GC process starts to transfer valid pages to the buffer, a fixed number of buffer slots are reserved in advance for receiving valid pages from this GC process.

With the first rule, we can make a full utilization of valid page buffer for reducing access conflict between GC processes and host I/O requests. With the second rule, valid

pages of each GC process can be transferred to valid page buffer greedily for avoiding the degradation of parallelism and idleness of flash chips.

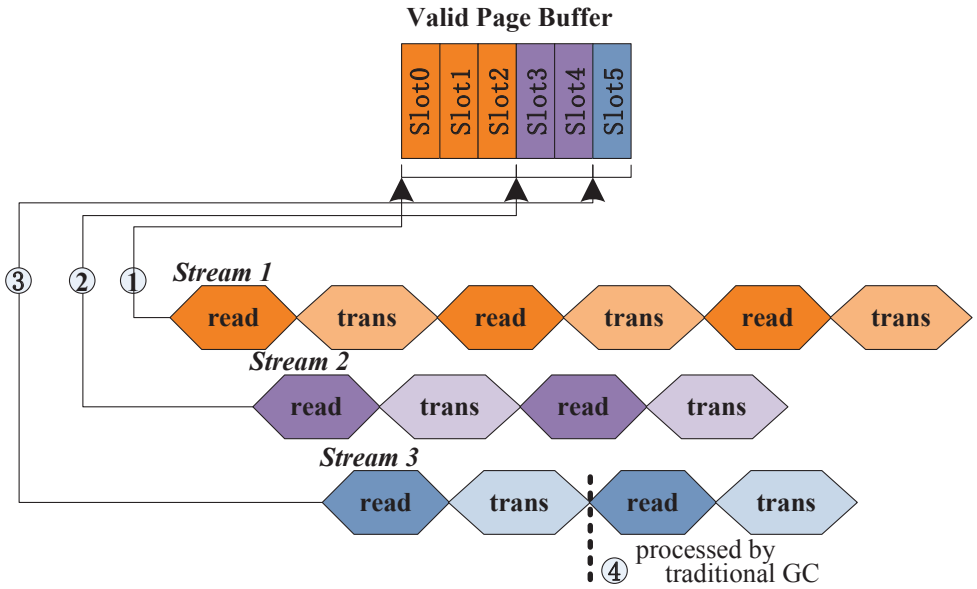


Fig. 9. Process of valid page movement streams.

Figure 9 presents an example on the valid page movement. In the figure, we assume that there exist 6 empty slots in valid page buffer. Within flash memory, there are 3 to-be-collected blocks which are going to be reclaimed by GC activities. These 3 to-be-collected blocks contain 3, 2 and 2 valid pages, respectively. When the first GC activity is triggered, the first stream is going to read and transfer 3 valid pages from to-be-collected block to valid page buffer. 3 slots are occupied in advance even though these 3 valid pages still have not been transferred into the buffer. After that, if the first GC activity has not completed and another GC is triggered, then another stream starts to read and transfer 2 valid pages to the buffer. For the second stream, only the fourth and fifth slots in valid page buffer can be obtained. Then, if the third GC activity is triggered and the third stream starts to read and transfer 2 valid pages into this buffer. Since the first two streams have occupied 5 slots in the valid page buffer, only one slot left in the buffer can be used to keep one valid page from the third stream. Hence, the third GC activity only reads and transfers one valid page to the valid page buffer and the rest valid page in to-be-collected block is supposed to be migrated via traditional GC process.

4.1.2. Checking Buffer. When the valid pages of to-be-collected blocks are transferred and temporarily held in the valid page buffer, host I/O requests are potential in accessing these valid pages before writing them back. In order to maintain the data consistency, valid page buffer should be checked in advance before issuing host I/O requests into flash chips. If the requested data have been buffered, the host I/O requests can obtain the corresponding data directly from valid page buffer (for read requests), or be released (for write requests). Note that, write requests, which can obtain data from valid page buffer, is supposed to be released in this work instead of releasing valid pages resided in the buffer. As we mentioned above, valid pages in the buffer can

make a full use of rich parallelism and idleness of flash chips via writing them back to idle chips without access conflicts. Hence, compared with host I/O requests, these GC induced valid pages resided in the buffer have larger potential for performance improvement.

Figure 10 shows the process of checking buffer. There exist a valid page buffer mapping table, which is composed of logical page number (*LPN*), buffer page number (*BPN*) and *type*. *LPN* maintains the logical address of valid page, which can be obtained when it is read from to-be-collected block. When the LPN of valid page is the same as that of host I/O request, this host I/O request is going to directly read data back from valid page buffer (for read request) or be released (for write request). *BPN* records the address of valid page in valid page buffer. Once valid page is added into valid page buffer, this area should be recorded for the following access. Differing from the above two components, *type* is a component added in the mapping entry for grouping valid pages from the same GC process together. Mapping entries of valid pages from the same GC process are set with a same type value in the *type* area. This *type* helps to group valid pages from the same GC process into the same segment in valid page buffer. Hence, the *type* based valid page organization method can be used by the following writing back policy, which aims at writing valid pages back quickly and smartly.

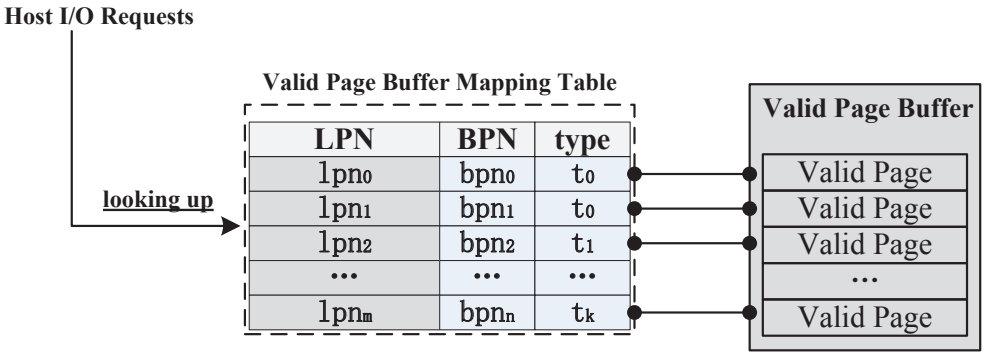


Fig. 10. The structure of valid page buffer mapping table.

For the writing back policy, which shows the structure and writing order of valid pages , we first present it in Section 4.2. Then, the processes of writing host I/O requests and valid pages from valid page buffer are presented.

4.2. Writing Requests Back to SSDs

4.2.1. Writing Back Policy of Valid Page Buffer. The structure of valid page buffer is composed of several segments, where each segment contains valid pages from the same GC process. With applying this structure, the segment is defined as the unit of valid pages written back to flash. We require that these valid pages resided in the same segment should be written back one time and the issuing order is determined by the completion time of transferring all valid pages from the same GC process to valid page buffer. This is a basic First-In-First-Out policy, which can guarantee the fairness among segments. When valid pages are written back to idle chips, the corresponding mapping entries, which record logical-to-physical address mappings, should be constructed and stored in mapping table.

For selecting idle chips, we maintain a chip state vector and each slot in this vector only requires one bit (1 for free state and 0 for busy state). In this case, once there exist accesses in flash chips, the corresponding slots are marked as 0. If accesses complete,

the corresponding slots are marked as 1. Hence, when there exist idle chips, a round-robin policy shown in the Figure 11, is adopted to select idle chips for writing valid pages back to flash memory.

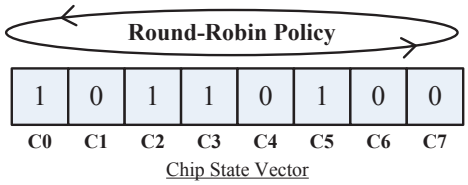


Fig. 11. The chip state vector for selecting idle chips.

4.2.2. Writing Host I/O Requests and Valid Pages Back to SSDs. The writing order of valid pages in the buffer has been determined by the above writing back policy. In this work, we propose to exploit the idleness of flash chips for the valid page write back in parallel. The basic idea of the scheme is to write the valid pages to the idle flash chip dynamically. In order to support this kind of page writes, the valid page writing back policy is able to redirect valid pages to another idle chip. This redirection method is the same as the applied dynamic data allocation scheme in this work, which has been well studied and designed to write data to any idle chips [Hu et al. 2011][Bucy et al. 2008].

With above settings, the process of writing requests (including host I/O requests and GC induced write requests) back to SSDs is presented as following: First, host I/O requests in the pending queue are issued to flash chips based the computed physical addresses from FTL. For read requests, their locations are determined by checking the mapping table. If the corresponding chip is idle, it is issued. Otherwise, it is blocked in the queue. For write requests, they are issued if the chips where they access are idle. Second, when there are no pending host I/O requests, the valid pages are written back. After the host I/O requests are issued based on above rules, if there are idle chips, the valid pages can be written to these chips as regular host I/O requests.

As shown in Figure 7, since there exist idle chips where no host I/O requests are mapped to these chips, valid pages in the valid page buffer can be sent to these chips and written back. Based on the observation in the motivation section, we believe it is easy to find the idle chips to process the writing back of valid pages.

4.3. Implementation

The implementation of the proposed approach in SSDs is presented in Figure 12. There are two components implemented in the SSD controller: valid page buffer and buffer management. The valid page buffer is designed for holding the valid pages from the to-be-collect blocks from chips. The buffer management is designed for maintaining and managing valid pages resided in this buffer.

Algorithm 1 shows the implementation of proposed approach. The algorithm is partitioned into two steps: read and transfer valid pages to the controller (Line 1 to 4), and write valid pages back to SSDs (Line 10 to 14). As shown in Algorithm 1, once GC is activated, the GC is executed to reclaim the invalid pages (Line 1). First, the valid page buffer will be checked if there exist enough slots for receiving the valid pages from the target block (Line 2). If there are empty slots for receiving valid pages, these valid pages are transferred into valid page buffer (Line 3). Traditional GC is activated if there are not enough slots for the valid pages (Line 6). After transferring all valid

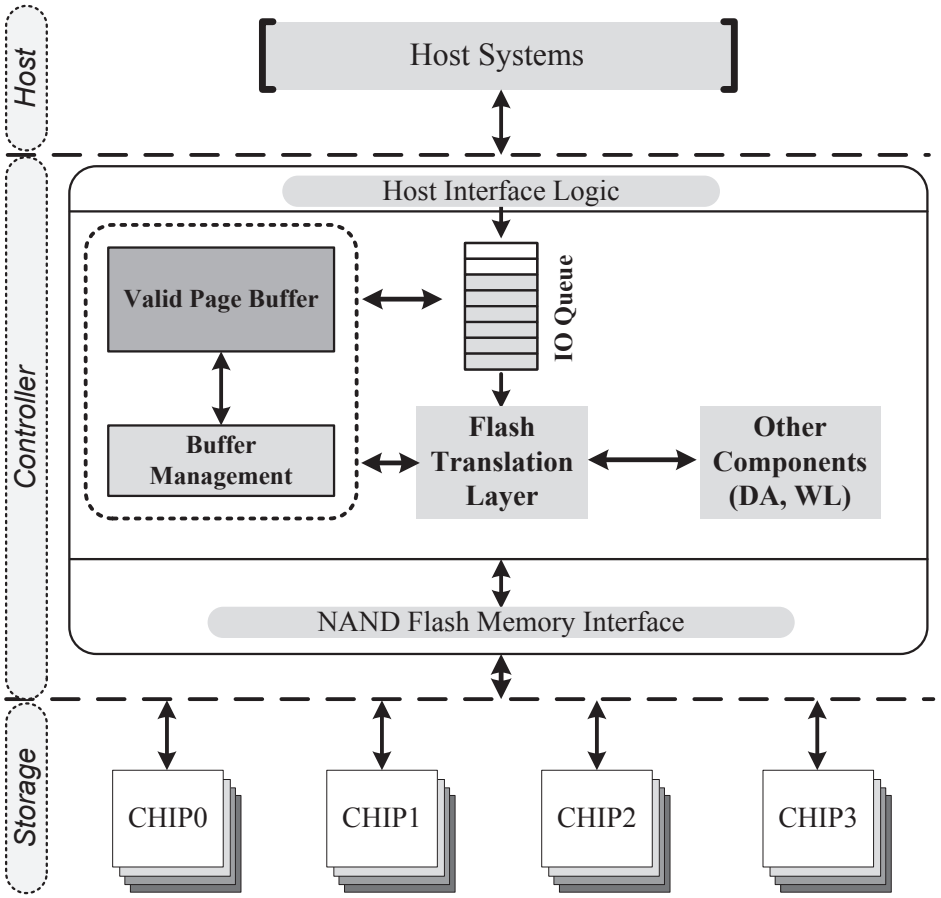


Fig. 12. The implementation of proposed approach in the SSD controller.

pages into valid page buffer or this buffer is full, the erase operation is issued to reclaim the target block (Line 9). If there are no host request being issued and there are valid pages in the buffer, valid pages are going to be written back to SSD (Line 10). If there are idle chips, valid pages are written to the these chips and the corresponding copies in valid page buffer are released (Lines 11 to 13).

Overhead analysis: The main space overhead of the proposed approach is on the valid page buffer. A slot in the buffer needs to host a page of data. The number of slots in the buffer depends on many factors. First of all, it depends on the IO throughput requested by the host. The more IO requests from the host system, the more slots are needed in the buffer to hold the valid pages. Secondly, it also depends on the maximum number of concurrent GC processes as well as the maximum number of valid pages involved in each GC process, which are related to the data locality of workloads. In the experiment, we will evaluate the effects on the size of valid page buffer. The valid page writing back process is realized by traditional method, which is responsible for writing host I/O requests into flash chips. Hence, the cost of valid page writing process is negligible. In addition, since valid page buffer in SSD controller uses RAM module to achieve fast access, the time cost of reading and writing in valid page buffer is faster than the time cost of accessing flash memory [Gupta et al. 2009]. Hence, the time cost

ALGORITHM 1: Minimizing GC Induced Chip Access Conflicts

Require:
$Valid\_Page$ – GC Induced Valid Page Movement;
Buf – Valid Page Buffer in the SSD Controller;
1: **if** $Chip\_State$ == GC **then**
2: **while** $Valid\_Page$!= NULL && full(Buf) != TRUE **do**
3: Transfer_to_Buf ($Valid\_Page$, Buf);
4: **end while**
5: **if** $Valid\_Page$!= NULL **then**
6: Activate Traditional GC;
7: **end if**
8: **end if**
9: Erase Target Block;
10: **while** empty($HostIOs$) == TRUE && empty(Buf) != TRUE **do**
11: Find_Idle_Chips (i);
12: Write_to_Chip (i, $Buf.head$);
13: Remove_Page($Buf.head$);
14: **end while**

of reading valid pages to valid page buffer and writing them back to flash memory is negligible as well. The other space overhead comes from valid page buffer mapping table, which contains three components. The space cost of *BPN* and *type* is determined by the space of valid page buffer. In this work, the size of valid page buffer is set as 512KB for keeping 128 valid pages. That is, the valid page location in valid page buffer and the type of GC process only need 7 bits, respectively. Combined with traditional mapping table [Gupta et al. 2009], where each mapping entry containing logical and physical addresses only need 4 bytes, we can find that the space overhead of buffer mapping table is negligible as well.

5. EXPERIMENT AND ANALYSIS

In this section, we first present the experimental methodology. Then, the experimental results of the proposed approach are presented with the performance improvement, and the sensitive studies on the size of valid page buffer.

5.1. Experiment Methodology

In this work, a trace drive simulator, SSDsim [Hu et al. 2011], is used to verify the proposed approach. SSDsim has been widely applied in the performance exploration of SSDs, especially for the parallelism exploration [Gao et al. 2014][Hu et al. 2011]. The proposed approach is implemented in the controller of the SSDs. For the experiment configurations, a 128GB SSD is simulated in default, with 16 channels, each channel with 4 chip, each chip with 2 dies, and each die with 2 planes [Agrawal et al. 2008][Chen et al. 2011a][Hu et al. 2011][Li et al. 2016]. In the controller, default management mechanisms are implemented, including page mapping based flash translation layer [Agrawal et al. 2008], static wear leveling scheme [Chang et al. 2007] and dynamic data allocation scheme [Hu et al. 2011][Bucy et al. 2008]. Note that, in order to avoid the performance impact from different types of flash translation layers, we assume that all mapping entries are stored in mapping cache. Greedy based GC scheme [Agrawal et al. 2008][Hu et al. 2011][Prabhakaran and Wobber 2009][Bucy et al. 2008] is implemented for its excellent performance. The proposed approach also works with other GC schemes. The reserved space is set to 15% of the total SSD and GC is activated when the free space is smaller than 5%, complying with previous works [Agrawal et al. 2008]. The I/O queue length is set as 64 and the valid page buffer includes 128 page entries in default. In order to simulate an aged SSD that has been used for a long time, the simulated SSD is warmed up with random data where invalid pages are

Table I. Parameter Settings for the Simulated SSD.

Parameters	Value	Parameters	Value
Number of Channels	16	T_r (in Page)	0.03 ms
Chips per Channel	4	T_p (in Page)	0.6 ms
Dies per Chip	2	T_e (in Block)	3 ms
Planes per Die	2	$T_{transfer}$ (in Byte)	10 ns
Blocks per Plane	1024	Reserved Space	15%
Pages per Block	64	GC Threshold	5%
Page Size	4 KB	IO Queue Length	64
Valid Page Buffer	128		

Table II. Characteristics of Traces.

Traces	WR	WSS	AS	ARS	AWS	SR	IA Time	Pct of B-Reqs
HM_0	62.9%	1.9 GB	33.3 GB	11.2 KB	12.4 KB	75.7%	14.4 ms	76.72%
PRN_0	89.3%	1.6 GB	72.1 GB	26.6 KB	13.9 KB	67.7%	12.0 ms	80.82%
PRN_1	30.8%	7.1 GB	43.3 GB	18.3 KB	13.8 KB	99.9%	5.9 ms	73.04%
PROJ_0	79.1%	1.7 GB	58.7 GB	20.8 KB	19.5 KB	96.0%	30.6 ms	44.63%
PRXY_0	97.4%	22.6 GB	32.0 GB	11.3 KB	7.5 KB	40.1%	4.84 ms	25.76%
STG_0	76.9%	6.2 GB	82.2 GB	33.6 KB	12.7 KB	88.7%	27.4 ms	79.18%
USR_0	62.9%	2.1 GB	99.7 GB	47.4 KB	13.5 KB	95.9%	27.5 ms	48.77%
WDEV_0	79.9%	0.2 GB	44.9 GB	16.6 KB	12.1 KB	93.8%	52.8 ms	84.83%

Note that, WR:Writes Ratio; WSS: Working Set Size; AS: Accessing Size; ARS: Average Read Size; AWS: Average Write Size; SR: Sequential Requests Ratio; IA Time: Inter-arrival Time; Pct of B-Reqs: Percentages of Burst Requests.

initially scattered in the SSD. The detailed settings of the SSD are listed in Table I, where typical MLC flash memories are simulated.

Since the proposed approach aims at reducing the impact from GC activities, the simulated SSD is filled with data in advance so that GC activities can be triggered timely. In this work, we fill the simulated SSD with 84% valid data, 10% invalid data and 6% free space. Once the free space drops to 5% of the total size of SSD, GC activities will be triggered for reclaiming free space.

Eight traces are selected from a series of MSR Cambridge servers [Narayanan et al. 2009], including hm_0, prn_0, prn_1, proj_0, prxy_0, stg_0, usr_0, and wdev_0. These traces have been widely used in previous works for the studies of SSD performance [Gao et al. 2014][Hu et al. 2011][Jung and Kandemir 2012][Shin et al. 2009][Li et al. 2016]. All these traces contain a large number of write requests to trigger enough GC activities. For each trace, at least 3,000,000 I/O requests are simulated to produce a long running process. The characteristics of all traces are presented in Table II. Since the internal activities are highly related with the characteristics of trace, we collect several attributes in Table II, such as write ratio, working set size, average read/write size, sequential/random requests ratio, inter-arrival time and the percentages of burst requests. Among these attributes of traces, the inter-arrival time indicates the average arrival time of two adjacent arriving host I/O requests. However, since there exist burst I/O requests accesses, we evaluate another attribute called the percentages of burst requests, which records the percentages of burst I/O requests. In this work, host I/O requests arriving within 0.2ms are identified as burst requests.

In addition, the SSD behaviors, such as GC rate, which is defined as the average number of written flash pages for each triggered GC activity, and migrated valid pages in a GC, are also highly related with trace characteristics. Hence, we collect GC rate and average migrated valid pages and show them in Figure 13. In this figure, we can find that GC activities are frequently triggered among all traces, and each GC activity needs to migrate 22.1 valid pages to other free space, on average. Note that, in order to evaluate the efficiency of the proposed scheme, an aged SSD is simulated, of which the SSD is fully filled with valid data and there are only a limited number of free space for

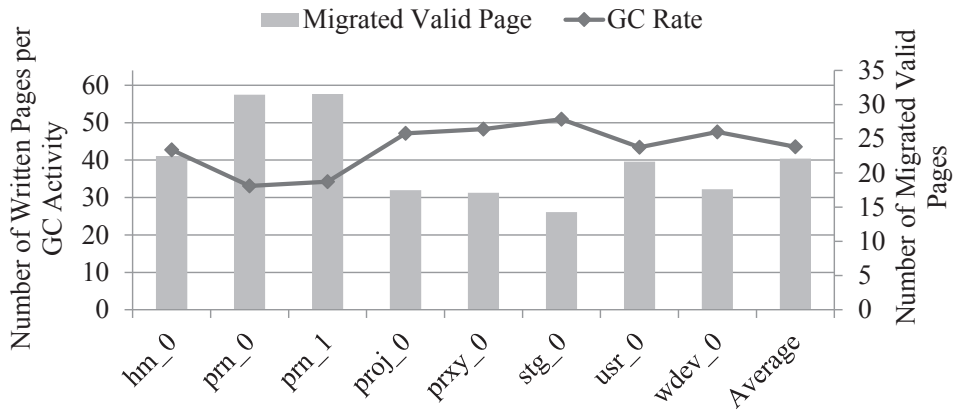


Fig. 13. GC rates and average migrated valid pages among traces.

coming requests. In this case, the GC rates and the number of valid pages during each GC is some similar for each workloads. In addition, GC is frequently activated and has significant impacts on performance of SSDs. On average, 43.5 flash pages are written for triggering one GC activity and each GC activity will migrate 22.1 valid pages to other free space.

5.2. Experimental Results

In this section, four schemes are evaluated in term of I/O performance and bandwidth, including traditional scheme, preemptive scheme [Lee et al. 2013] and the DPR scheme [He et al. 2015], and the proposed scheme.

5.2.1. I/O Performance and Bandwidth Improvement. In this subsection, the performance and bandwidth results are presented and analyzed.

Read Performance: Figure 14 shows the read access latency comparison among evaluated four schemes. As shown in Figure 14(a), the read access latency of the proposed approach is reduced to 59.5%, 67.3% and85.4%, compared with the traditional scheme, the preemptive scheme, and DPR scheme, respectively. The significant improvement of read latency is achieved through reducing the latency of the conflicted read requests. In order to understand the latency reduction for conflicted read requests, the percentages of conflicted read latency of these four schemes are collected, which are presented in Figure 14(b). In the result, the percentages of conflicted read latency are reduced compared with traditional and preemptive schemes. In most case, we can find that the significant read performance improvement is brought by the highly degradation of the percentages of conflicted read latency. However, we also found that some workloads, such as prn_0 and proj_0, show little read performance improvement even with significant percentage reduction of conflicted read latency. The reason can be found in Figure 1, where we can find that the GC induced conflict for these two workloads are not serious. As shown in Figure 1, the read latencies are increased only by 3.5 and 2.6 for these two workloads, respectively.

For the preemptive scheme, it is proposed to preempt the on-going GC process and insert the host I/O requests into the preemption points. With this approach, the latency of conflict read requests is able to be reduced and read bandwidth can be improved. As shown in Figure 14(b), the percentages of conflicted read request latencies are reduced to 85.3%, on average. However, the preemptive scheme cannot insert more requests into the on-going GC when the rest space of SSD is further reduced. When requests

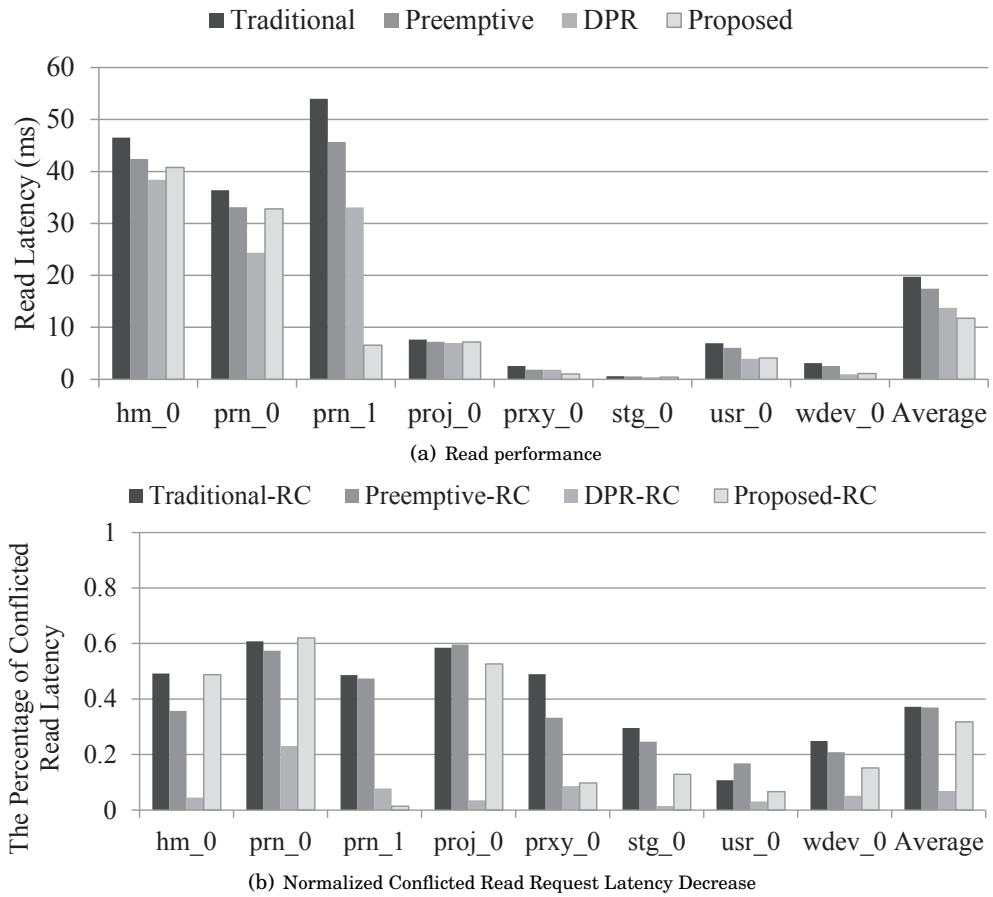


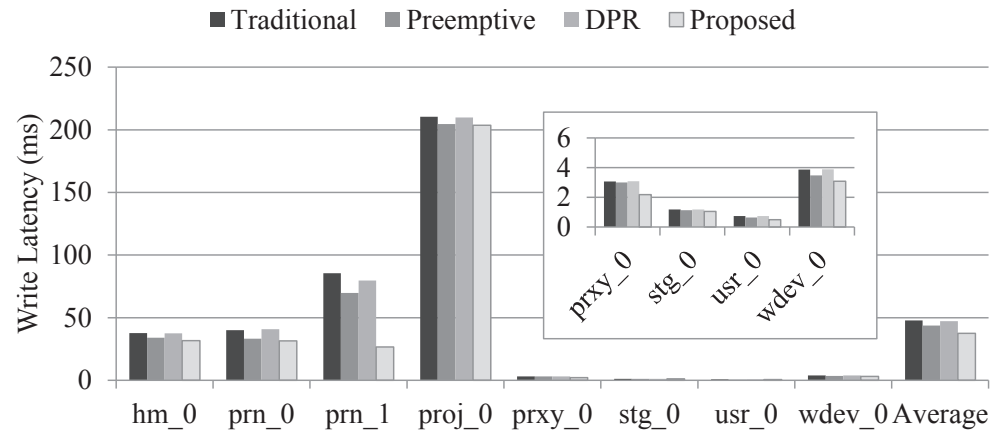
Fig. 14. Read performance improvement and normalized conflicted read request latency decrease among traditional scheme, Preemptive scheme, DPR scheme and proposed scheme.

are bursty and the space of SSDs is further utilized, the achieved read performance and bandwidth improvement would be limited. Differently, we noted that the achieved degradation of conflicted read latency of DPR is better than proposed scheme while the reduced read latency of DPR is worse than proposed scheme. The reason is from that DPR only moves the frequently accessed data to different chips. In this case, there still exist conflict read requests which are going to access the un-moved data in the GC occupied chips. Therefore, the average latencies of conflicted read requests are still large and the achieved read latency degradation is limited as well.

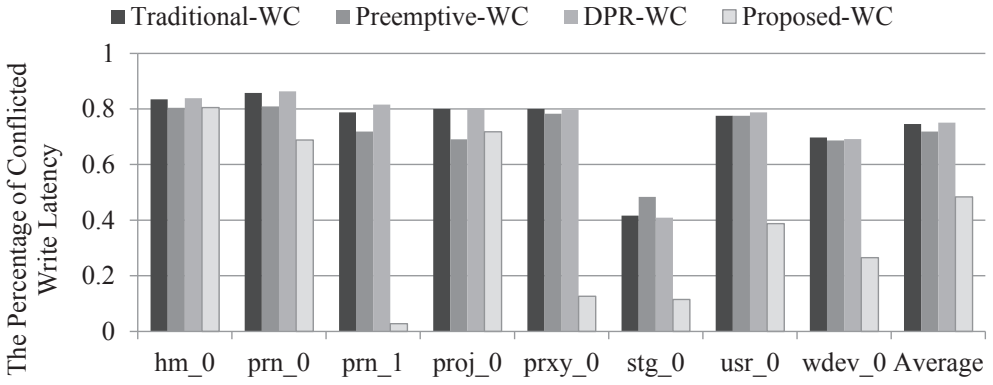
Write Performance: Figure 15 presents the write access latency comparison among evaluated four schemes. As shown in Figure 15(a), the write latency of the proposed approach is smaller than all other three schemes as well. Compared with the traditional scheme, the preemptive scheme and DPR scheme, the proposed approach reduces the write latency to 78.4%, 85.7% and 79.6%, respectively. Similarly, the highly write latency degradation also comes from the degradation of conflicted write request latency. The percentages of conflicted write latency among these four schemes are also presented, in Figure 15(b). As shown in the results, the percentages of conflicted write latency are significantly reduced. Whats more, the result for DPR scheme is the worst one, which also induce bad read performance. We also found that prn_0 shows little write performance and write bandwidth improvement even with significant percent-

age reduction of conflicted write latency. The reason can be found in Figure 1, where we can find that the impact of GC induced conflicts for prn_0 is not serious. As shown in Figure 1, the write latencies are increased only by 2.9 for prn_0.

Comparing with the other two schemes, the proposed scheme improves the write performance significantly. For preemptive scheme, GC induced access conflict occurs when the space of SSDs is further utilized. Similar to read requests, the performance and bandwidth improvement of write requests are going to be limited when the incoming requests are bursty and the space of SSDs is further utilized quickly. In addition, when all parallel chips have been fully utilized, other incoming write requests still are going to highly reduce the performance and bandwidth of SSDs due to on-going time-consuming GC processes. Thus, the write latency of the preemptive scheme only are reduced to 91.4% on average, where the percentages of conflict write request latency only is reduced to 96.3%. For the DPR scheme, the evaluated write latency is similar with the traditional scheme, which does not try to avoid the write conflicts.



(a) Write performance



(b) Normalized Conflicted Write Request Latency Decrease

Fig. 15. Write performance improvement and normalized conflicted write request latency decrease among traditional scheme, Preemptive scheme, DPR scheme and proposed scheme.

Overall Performance: As shown in Figure 16, the I/O latency of the proposed scheme is reduced to 75.8% compared with the traditional scheme, on average. In the meanwhile, the I/O latency of the preemptive and DPR schemes are reduced to 91.0%

and 94.6% compared with the traditional scheme on average, respectively. The access latency decreases of requests is highly related with the number of valid pages transferred to valid page buffer and the number of reduced conflicts. This is because that the more valid pages resided in to-be-collected blocks are transferred into valid page buffer, the more host I/O requests can be issued into SSD without being conflicted by GC activities.

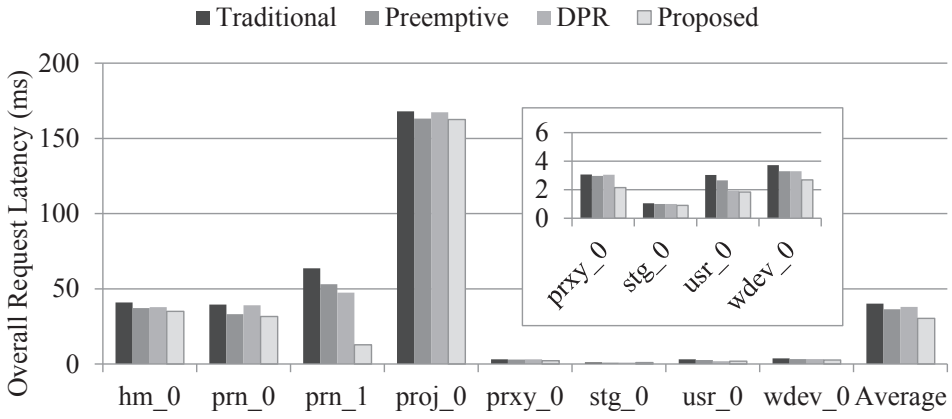


Fig. 16. Overall performance improvement among traditional scheme, Preemptive scheme, DPR scheme and proposed scheme.

Figure 17 presents the result on the distributions of valid pages moved to buffer and moved by traditional scheme and the percentages of reduced conflicts with applying proposed approach. For the distributions of valid pages, we can find the proposed approach is able to reallocate a large number of valid pages to other free space in parallel. Figure 17 shows that most of valid pages are transferred, with 48.6% on average. Take stg_0 as an example. It has 98.6% valid pages transferred and written back in parallel, which achieves significant GC cost degradation. With large number of valid pages transferred into valid page buffer, the cost of GC activities can be highly reduced. Figure 18 presents the cost of GC activities of the proposed approach, which is highly reduced compared with traditional scheme that presented in Figure 4. However, the proposed approach can introduce significant performance improvement only when GC impact is significant as well. If there exists a workload (e.g. proj_0, shown in Figure 1) on which GC induced impact is not serious, valid page movements introduced performance improvement is limited as well. Different from traditional scheme, the proposed scheme improves the performance of SSD by quickly moving the valid pages to the buffer in the SSD controller. In this case, the conflict between GC and host requests are minimized. Whats more, valid pages in the buffer are written back to flash memory by exploiting the idleness of chips. For reduced conflicts, since the proposed approach can not only reduce the conflicts between host I/O requests but also may create new conflicts when valid pages are reallocated to other locations. In this case, we evaluate the percentages of the total reduced conflicts in Figure 17. In the results, we can find that although there exist new conflicts, the total conflicts are highly reduced. With the large number of reallocated valid pages and the degradation of conflicts, the cost of GC activities can be reduced so that the performance can be improved. From the results, we can find that the proposed approach can minimize the impact of GC induced access conflict.

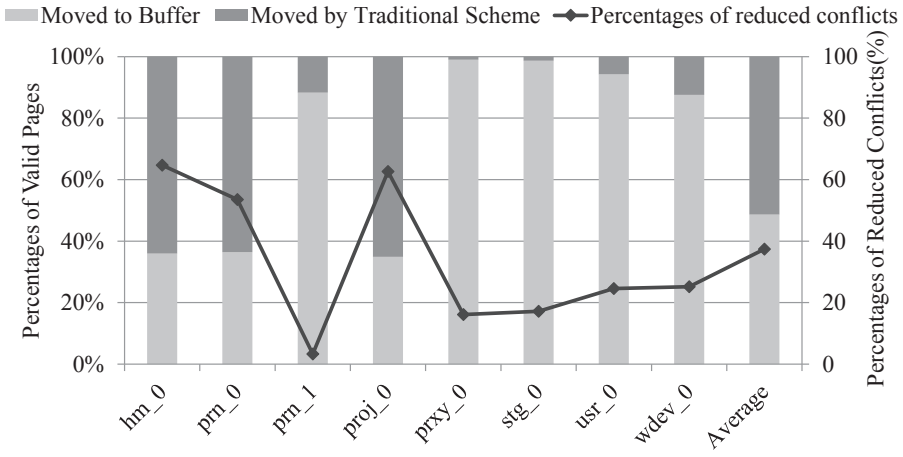


Fig. 17. The distributions of the valid pages moved to buffer and moved by traditional scheme and the percentages of reduced conflicts.

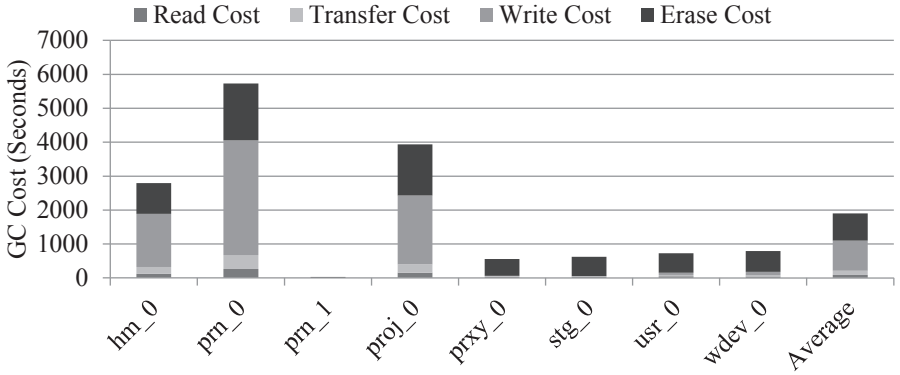


Fig. 18. The cost of GC activities of the proposed approach.

Table III. The raw bandwidth of the simulated SSD.

Bandwidth Tests	Seq-W (64KB)	Seq-R (64KB)	Rand-W (4KB)	Rand-R (4KB)
Value(MB/s)	192.5	202.5	102.8	92.1

Bandwidth Improvement: In this part, the raw bandwidth of the simulated SSD is presented in Table III. From the table, we can find that bandwidth of sequential write, sequential read, random write and random read are 192.5MB/s, 202.5MB/s, 102.8MB/s and 92.1MB/s, respectively. The sequential bandwidths are measured when the page unit is 64KB and the random bandwidths are measured when the page unit is 4KB. From the collected data, we can find that the simulated SSD is reasonable and credible. In the following, we will present the bandwidth improvement by the proposed scheme with the configured SSD under various workloads.

In Figure 19, the results for read, write and overall bandwidth improvement compared among the four evaluated schemes are evaluated to show the effectiveness of the proposed approach. From the results, we can find that the read and write bandwidths are improved compared with all the other three schemes. In addition, the overall bandwidth is approaching 150MB/s. Note that the evaluated bandwidth is relatively

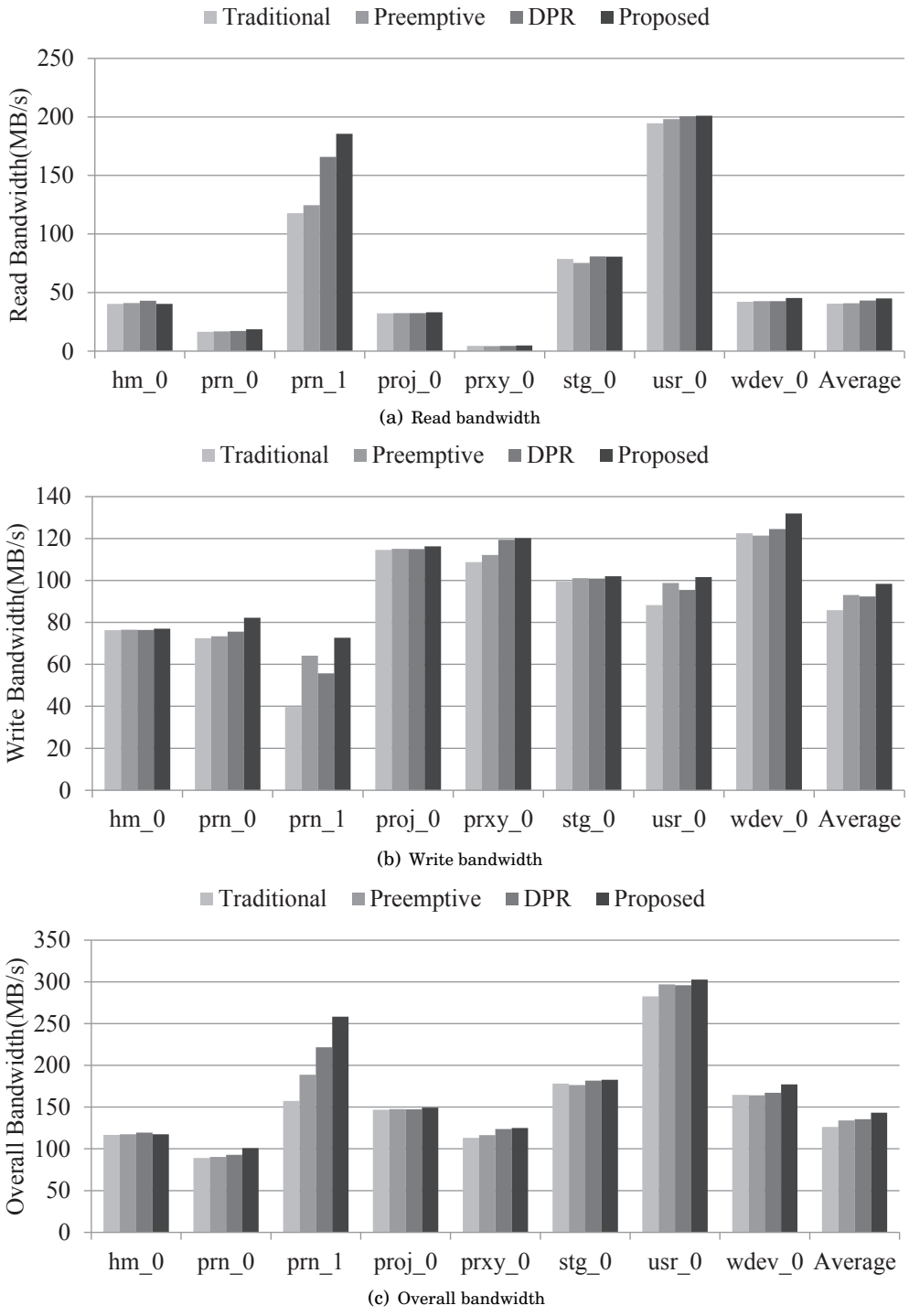


Fig. 19. Bandwidth improvement among traditional scheme, Preemptive scheme, DPR scheme and proposed scheme.

smaller than the raw bandwidth is due to the high rate of GC in the SSD, which takes many IO clock in the storage.

For read bandwidth, as shown in Figure 19(a), compared with traditional scheme, preemptive scheme and DPR scheme, the proposed approach increases the read bandwidth 1.11, 1.09 and 1.04 times on average, respectively. For write bandwidth, as shown in Figure 19(b), proposed scheme is able to increase write bandwidth 1.14, 1.05 and 1.06 times on average, respectively. For overall bandwidth, as shown in Figure 19(c), compared with traditional scheme, preemptive scheme and DPR scheme, proposed scheme is able to increase overall bandwidth 1.14, 1.07 and 1.06 times on average, respectively. When the number of conflicts between GC activities and host I/O requests are highly reduced, more host I/O requests can be issued into SSD and completed in time so that bandwidth is improved.

5.2.2. Reliability Issue Analysis. In this work, in order to maximize the performance improvement with applying the proposed approach, we adopt capacitor or NVM in SSD controller. In order to analyze the worst case when SSD is crashed or powered off, we collect the number of average valid pages resided in valid page buffer and the waiting time for the write-back operations. The results are presented in Figure 20.

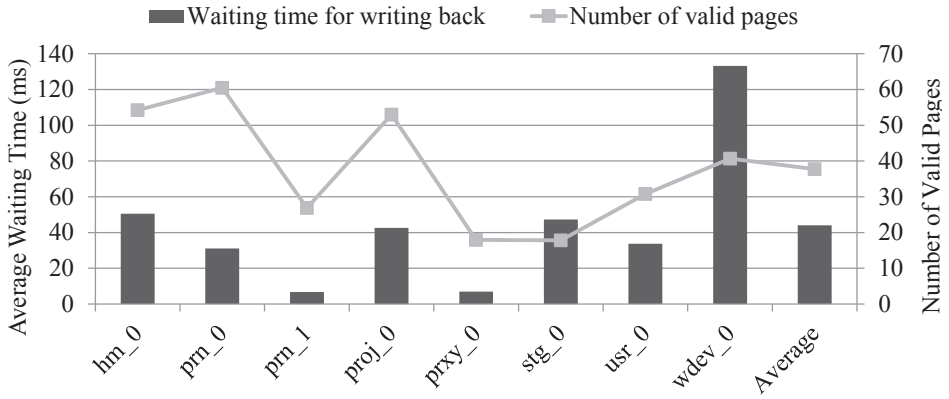


Fig. 20. Average waiting time of valid pages for writing back and the average number of valid pages resided in valid page buffer.

In this figure, we can find that the average waiting time of valid pages for writing back is 44ms, which is similar with the average request latency of traditional scheme. That is, valid pages resided in valid page buffer can be written back in time. In addition, for the cost of writing all valid pages back when system is crashed or powered off, we assume that DRAM is equipped as the valid page buffer and its read latency is set as 50ns [Saxena et al. 2012]. Hence, the formula of calculating the latency of writing all valid pages resided in valid page buffer can be presented as follows:

$$T_{write\_back} = Num_{valid\_page} \times T_{DRAM_read} + \lfloor \frac{Num_{valid\_page}}{Num_{chips}} \rfloor \times (T_{bus\_trans} + T_{flash_write}); \quad (1)$$

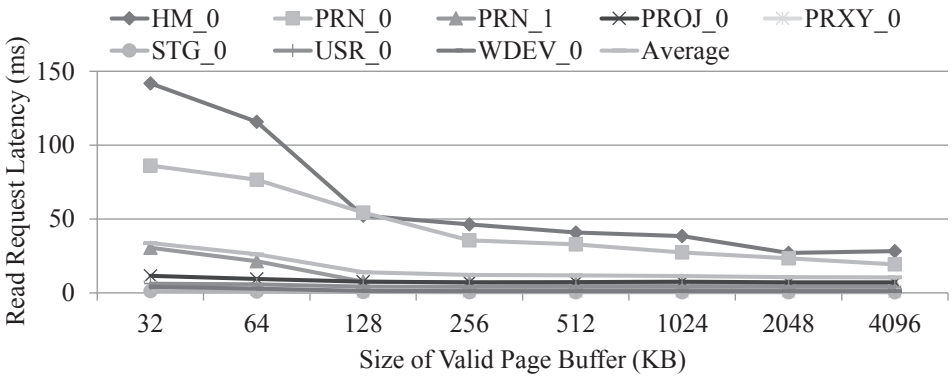
where T_{write_back} means the total time cost of writing valid pages back to SSD, Num_{valid_page} and Num_{chips} mean the number of valid pages resided in buffer and the number of parallel chips, T_{DRAM_read}, T_{bus_trans} and T_{flash_write} mean the time cost of reading one valid page from DRAM, time cost of transferring one valid page via bus and the time cost of writing one valid page to flash page. When SSD is crashed

or powered off, these valid pages can be written back through fully using the parallel chips. Based on this formula, we get the time-cost of write-back operations, for 1.84ms on average. Hence, we can see that the time cost of writing all valid pages resided in buffer is small.

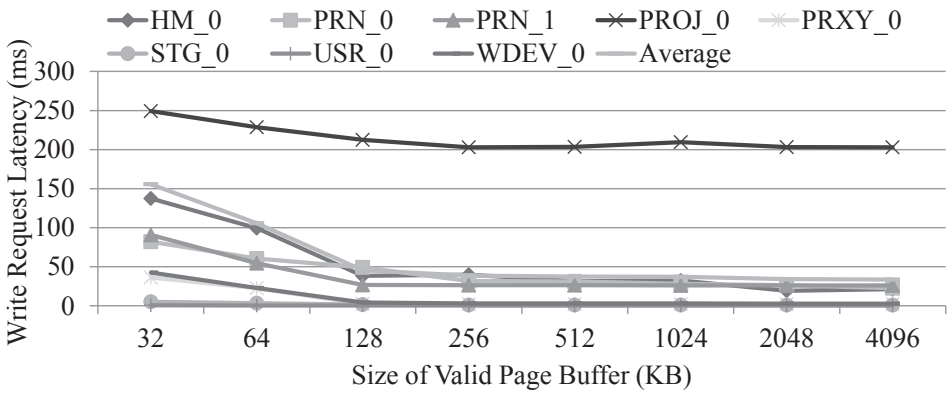
5.2.3. Sensitive Studies. In this subsection, three types of sensitive studies are discussed. First, the sensitive study of varying the size of valid page buffer is evaluated. Then, we vary the size of SSD to show the its impact on performance of SSDs. Last, the number of chips is taken into consideration.

Size of Valid Page Buffer: As mentioned in Section 4.1, the size of buffer is closely related with the relaxation of access conflict. In this part, we vary the sizes of buffer, from 32KB to 4096KB, to evaluate the latency of read and write requests.

Figure 21 shows the results on the normalized read and write latency reduction with the increasing of valid page buffer size. In this figure, we can find that the increases of



(a) Read Latency



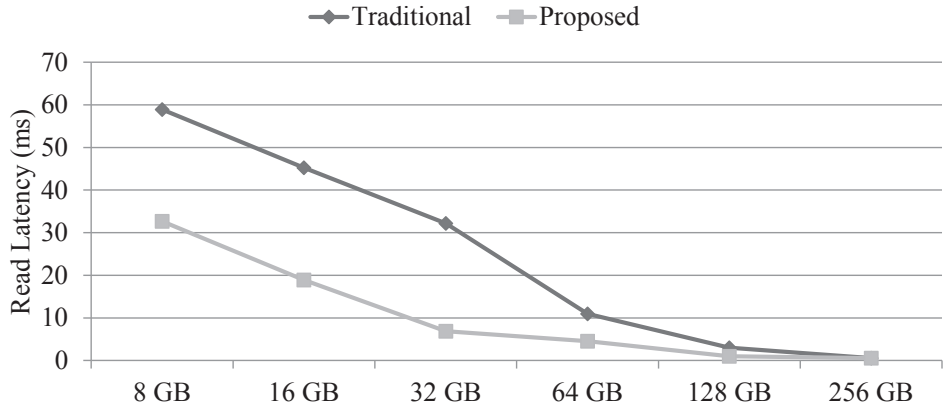
(b) Write Latency

Fig. 21. Sensitive study on the size of valid page buffer.

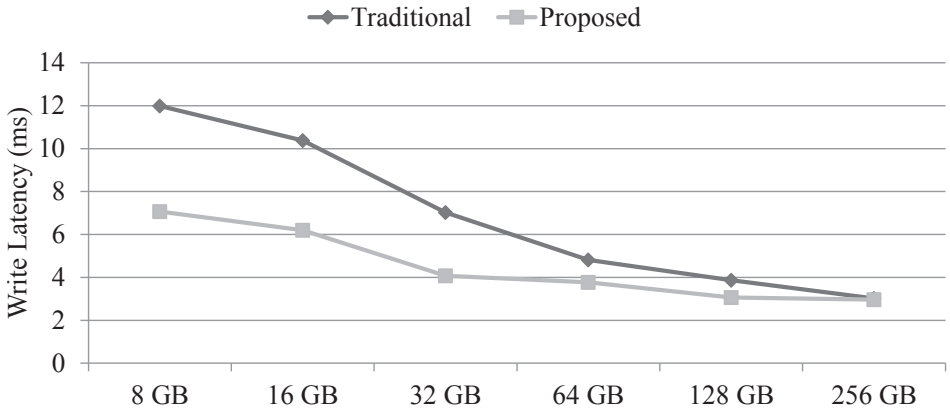
the size of buffer have significant improvement on the performance of read and write requests. For example, from 32KB to 512KB, the latency of read and write requests are reduced by 65.1% and 75.9% on average, respectively. The reason is that with the increases of buffer sizes, more valid pages can be transferred to buffer. However, the performance improvement is limited when the size of buffer is increased to a threshold.

When the size of buffer increases from 512 KB to 4096 KB, the latency of read and write requests are reduced only by 11.1% and 10.3%, respectively. The reason is that when the buffer size is large enough, there is little more valid pages are transferred to buffer. Hence, the size of buffer is set with 512 KB in this work.

Size of Storage: In this part, we vary the storage from 8GB to 256GB with the same number of flash chips. For each type of varied size of SSD, we only change the number of blocks within each plane. In this way, the impact from parallelism and idleness of flash chips can be removed. Take wdev_0 as an example, Figure 22 shows the read and write request latencies of traditional scheme and proposed scheme under various storage size.



(a) Read latency of traditional scheme and proposed scheme



(b) Write latency of traditional scheme and proposed scheme

Fig. 22. Sensitive study on the size of SSD.

Three conclusions can be found from the results: 1) With varying the size of SSD, the proposed approach is still able to achieve better performance than traditional scheme. That is, the proposed approach is able to achieve better performance compared with traditional scheme. 2) Request latencies are decreased when the size of SSD is increased. Since the larger size of SSD is equipped with larger over-provision space, which can be used for reducing the number of GC activities. Hence, when the size of

SSD is large, GC activities induced latency increase can be degraded. 3) When the size of SSD is set as 256GB, the read and write latencies of proposed scheme are similar with the latencies of traditional scheme. This is because that the reserved space of the simulated SSD is large enough for servicing incoming write requests so that a few incoming write requests induced GC activities are triggered.

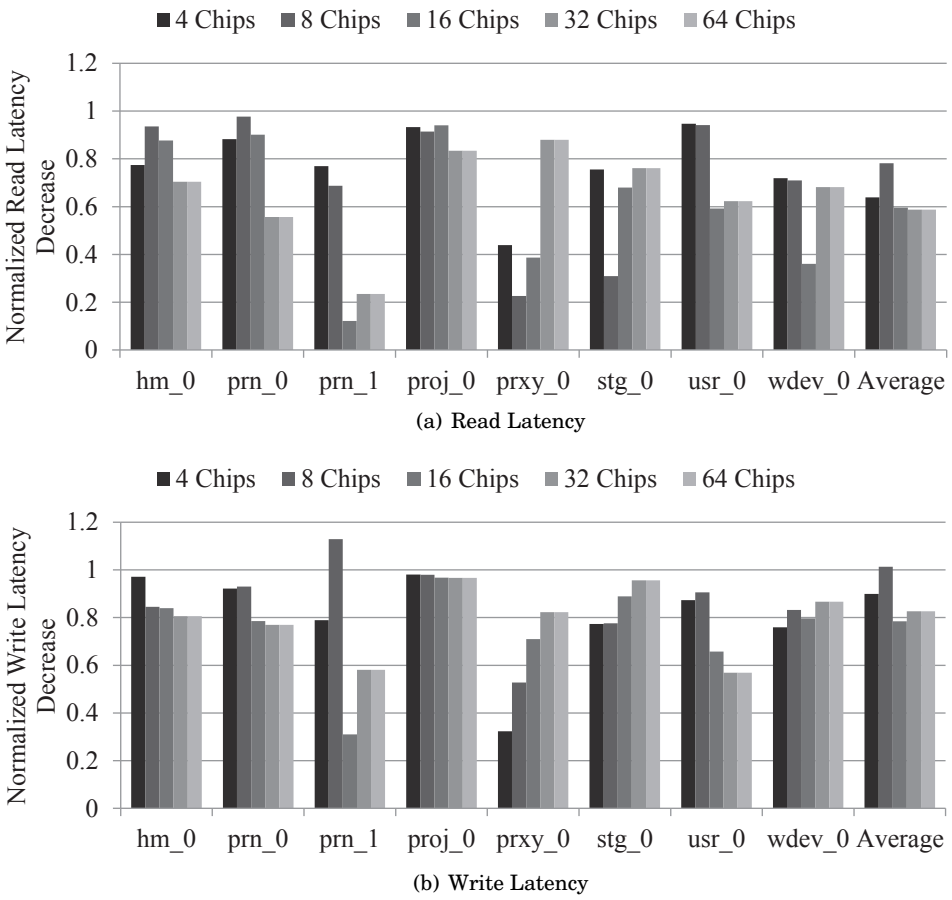


Fig. 23. Sensitive study on the number of flash chips.

Number of Chips: In this work, the achieved performance improvement is highly related with the number of idle chips. In this part, we vary the number of flash chips from 4 chips to 64 chips to evaluate the latency of requests.

Figure 23 shows the results of the normalized read and write latency reduction compared with traditional scheme from 4 to 64 chips. As shown in results, we can find that the proposed approach is consistently better than the traditional scheme no matter how many chips in the SSDs, except for prn_1, which has a few number of write requests. This observation shows the effectiveness of the proposed approach on different SSD settings. In addition, we also can find that the achieved benefit of the proposed approach various under different number of flash chips. The reason is simple. When the number of chips is increased, on the one hand, more idle chips are able to be used for reducing the cost of GC activities. On the other hand, when there are more flash

chips, more host I/O requests are potential to be conflicted by GC activities. That is, the achieved benefit is highly related with the access pattern of traces.

6. CONCLUSIONS

In this paper, a GC induced access conflict minimization approach is proposed through exploiting the parallelism and idleness chips of SSDs. An efficient implementation of the proposed approach with little overhead is presented. Experimental results show that the proposed approach is able to significantly improve the performance of SSDs. In the future works, we are going to analyze the impact from different GC policies and other internal activities, such as wear leveling. For different GC policies, based on the different selection policies, the number of transferred valid pages and the timing of triggering GC activities are going to be various so that the GC induced impact on performance will be different as well. For other internal activities, they are going to move valid page among flash chips as well. In order to reduce the performance decrease induced from these valid page movements, the proposed approach should be modified.

REFERENCES

Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark Manasse, and Rina Panigrahy. 2008. Design tradeoffs for SSD performance.. In *ATC*. 57–70.

John S Bucy, Jiri Schindler, Steven W Schlosser, and Gregory R Ganger. 2008. The disksim simulation environment version 4.0 reference manual. *Parallel Data Laboratory* (2008), 26.

Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. 2004. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *TECS* 3, 4 (2004), 837–863.

Li-Pin Chang and Chen-Yi Wen. 2014. Reducing asynchrony in channel garbage-collection for improving internal parallelism of multichannel solid-state disks. *TECS* 13, 2s (2014), 63.

Yuan-Hao Chang, Jen-Wei Hsieh, and Tei-Wei Kuo. 2007. Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design. In *DAC*. 212–217.

Feng Chen, David A Koufaty, and Xiaodong Zhang. 2009. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37. ACM, 181–192.

Feng Chen, Rubao Lee, and Xiaodong Zhang. 2011a. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *HPCA*. 266–277.

Feng Chen, Tian Luo, and Xiaodong Zhang. 2011b. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives.. In *FAST*, Vol. 11.

Fu-Hsin Chen, Ming-Chang Yang, Yuan-Hao Chang, and Tei-Wei Kuo. 2015. PWL: A progressive wear leveling to minimize data migration overheads for NAND flash devices. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1209–1212.

Zhiguang Chen, Nong Xiao, and Fang Liu. 2012. SAC: Rethinking the cache replacement policy for SSD-based storage systems. In *SYSTOR*. 13:1–13:12.

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

Cagdas Dirik and Bruce Jacob. 2009. The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization. In *ACM SIGARCH Computer Architecture News*, Vol. 37. 279–289.

Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T Kandemir, Chita R Das, and Myoungsoo Jung. 2017. Exploiting intra-request slack to improve SSD performance. In *ASPLOS*. ACM, 375–388.

Fusionio. 2015. PCIe SSDs. https://www.micron.com/products/solid-state-storage/bus-interfaces/pcie-ssds#/. (2015).

Congming Gao, Liang Shi, Cheng Ji, Yejia Di, Kaijie Wu, Chun Jason Xue, and Edwin H.-M. Sha. 2017. Exploiting parallelism for access conflict minimization in flash-based solid state drives. *TCAD* PP, 99 (2017), 1–1.

Congming Gao, Liang Shi, Mengying Zhao, Chun Jason Xue, Kaijie Wu, and Edwin H.-M. Sha. 2014. Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives. In *MSST*. 1–11.

Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 2009. *DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings*. Vol. 44. ACM.

Bingsheng He, Jeffrey Xu Yu, and Amelie Chi Zhou. 2015. Improving update-intensive workloads on flash disks through exploiting multi-chip parallelism. *Parallel and Distributed Systems, IEEE Transactions on* 26, 1 (2015), 152–162.

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *ICS*. 96–107.

Myoungsoo Jung and Mahmut Kandemir. 2012. An evaluation of different page allocation strategies on high-speed SSDs. In *Hotstorage*. 9–9.

Myoungsoo Jung, Ramya Prabhakar, and Mahmut Taylan Kandemir. 2012a. Taking garbage collection overheads off the critical path in SSDs. In *Middleware*. 164–186.

Myoungsoo Jung, Ellis H. Wilson, III, and Mahmut Kandemir. 2012b. Physically addressed queueing (PAQ): improving parallelism in solid state disks. In *ISCA*. 404–415.

Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh. 2014. Durable write cache in flash memory SSD for relational and NoSQL databases. In *SIGMOD*. 529–540.

Hyojun Kim and Seongjun Ahn. 2008. BPLRU: A buffer management scheme for improving random writes in flash storage.. In *FAST*, Vol. 8. 1–14.

Junghee Lee, Youngjae Kim, Galen M Shipman, Sarp Oral, and Jongman Kim. 2013. Preemptible I/O scheduling of garbage collection for solid state drives. *TCAD* 32, 2 (2013), 247–260.

Qiao Li, Liang Shi, Wu K Xue CJason, C Ji, Q Zhuge, and EHM Sha. 2016. Access Characteristic Guided Read and Write Cost Regulation for Performance Improvement on Flash Memory. In *FAST*. 125.

Bo Mao and Suzhen Wu. 2015. Exploiting request characteristics and internal parallelism to improve ssd performance. In *ICCD*. IEEE, 447–450.

Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: random write considered harmful in solid state drives. In *FAST*. 12–12.

Muthukumar Murugan and David HC Du. 2011. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *MSST*. IEEE, 1–12.

Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating server storage to SSDs: analysis of tradeoffs. In *EuroSys*. 145–158.

Yangyang Pan, Guiqiang Dong, and Tong Zhang. 2013. Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes. *TVLSI* 21, 7 (2013), 1350–1354.

Vijayan Prabhakaran and Ted Wobber. 2009. SSD extension for DiskSim simulation environment. *Microsoft Reseach* (2009).

Mohit Saxena, Michael M. Swift, and Yiying Zhang. 2012. FlashTier:A lightweight, consistent and durable storage cache. In *ACM European Conference on Computer Systems*. 267–280.

Narges Shahidi, Mohammad Arjomand, Myoungsoo Jung, Mahmut T Kandemir, Chita R Das, and Anand Sivasubramaniam. 2016. Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 48.

Liang Shi, Jianhua Li, Chun Jason Xue, Chengmo Yang, and Xuehai Zhou. 2011. ExLRU: A unified write buffer cache management for flash memory. In *EMSOFT*. 339–348.

Ji-Yong Shin, Zeng-Lin Xia, Ning-Yi Xu, Rui Gao, Xiong-Fei Cai, Seungryoul Maeng, and Feng-Hsiung Hsu. 2009. FTL design exploration in reconfigurable high-performance SSD for server applications. In *ICS*. 338–349.

Rujia Wang, Lei Jiang, Youtao Zhang, and Jun Yang. 2015. SD-PCM: Constructing reliable super dense phase change memory under write disturbance. In *ASPOLS*. 19–31.

Wujie Wen, Yaojun Zhang, Mengjie Mao, and Yiran Chen. 2014. State-restrict MLC STT-RAM designs for high-reliable high-performance memory system. In *DAC*. 1–6.

Wikipedia. 2010. Battery or supercap. https://en.wikipedia.org/wiki/Supercapacitor. (2010).

Mai Zheng, Joseph Tucek, Feng Qin, and Mark Lillibridge. 2013. Understanding the robustness of SSDS under power fault. In *FAST*. 271–284.

Received February 2007; revised March 2009; accepted June 2009