Reorder the Write Sequence by Virtual Write Buffer to Extend SSD's Lifespan

Zhiguang Chen, Fang Liu, Yimo Du

Reorder the Write Sequence by Virtual Write Buffer to Extend SSD's Lifespan

Zhiguang Chen, Fang Liu, Yimo Du

School of Computer, National University of Defense Technology
Changsha, China
chenzhiguanghit@gmail.com, liufang@nudt.edu.cn, poshand@163.com

Abstract. The limited lifespan is the Achilles's heel of Solid State Drive (SSD) based on NAND flash memory. NAND flash has two drawbacks that degrade SSD's lifespan. One is the out-of-place update. Another is the sequential write constraint within a block. To extend the lifespan, SSD usually employs a write buffer to reduce write traffic to flash memory. However, existing write buffer schemes only pay attention to the first drawback, but fail to overcome the second one. We propose a virtual write buffer architecture, which covers the two aspects simultaneously. The virtual buffer consists of two components, DRAM and the reorder area. DRAM is the normal write buffer which aims at the first drawback. It endeavors to reduce write traffic to flash memory as much as possible by pursuing higher hit ratio. The reorder area is actually a part of SSD's flash address space. It focuses on reordering write sequence directed to flash chip. Reordering write sequence helps to overcome the second drawback. The two components work together just like the virtual memory adopted by operating system. So, we name the architecture as virtual write buffer. Our virtual write buffer outperforms traditional write buffers because of two reasons. First, the DRAM can adopt any existing superior cache replacement policy, it achieves higher hit ratio than traditional write buffers do. Second, the virtual write buffer reorders the write sequence, which hasn't been exploited by traditional write buffers. We compare the virtual write buffer with others by trace-driven simulations. Experimental results show that, SSDs employing the virtual buffer survive longer lifespan on most workloads.

Keywords: flash memory; SSD; lifespan; virtual write buffer; storage

1 Introduction

Flash memory is of hot topic in recent years. Solid State Drives (SSDs) based on NAND flash have been widely deployed in storage systems. Different from hard disks, flash memory is absolute semiconductor chip without any mechanical components. Flash chip is hierarchically organized. Typically, a chip is comprised of thousands of blocks. A block is further divided into tens of pages. Flash memory supports three basic operations, write, read and erase. Write must be preceded by the erase. An alternation between write and erase is named as an E/W (Erase/Write) cycle. Read and write are performed at the granularity of page, while, erase is performed at block level.

Even though flash memory exhibits some superior features, there still exist some critical drawbacks, which hinder much wider deployment of SSDs. We focus on three of these drawbacks. One is the limited lifespan of flash memory. Generally, each block could only survive limited E/W cycles. When a block has been written and erased too many times, it becomes unreliable and even leads to bit flips. Another concerned drawback is the out-of-place update. It's unable to directly overwrite a page for flash memory. If a page that has already contained data will be rewritten again, it must be erased beforehand. Exactly, the whole block containing the page must be erased because erase is performed at block level. The last concerned drawback is the sequential write constraint within a block. Pages in a physical block cannot be written randomly. They must be written one by one in the same order as they are arranged in the block.

To overcome the first drawback discussed above, SSD employs a write buffer to reduce write traffic, thus extends its lifespan. Correspondingly, various write buffer management schemes have been proposed [e.g. 1, 2, 3, 4, 5, 6 and 7]. These schemes mostly aim at decreasing the write traffic directed to flash memory. However, the number of write requests is not the only one factor affecting lifespan. Actually, SSD's lifespan is impaired by both out-of-place update and sequential write constraint, which are the last two drawbacks as discussed above, respectively. Unfortunately, existing write buffer schemes mostly pay attention to the out-of-place update. They endeavor to reduce the quantity of updates to flash memory, but neglect to optimize the write sequence.

In this paper, we propose a novel virtual write buffer architecture to extend SSD's lifespan. The virtual write buffer consists of two components. One is the DRAM, which is also employed by traditional write buffers. The other is a part of SSD's address space. Actually, our virtual buffer is the same with virtual memory employed by Operating Systems (OS). These two components focus on the two factors affecting SSD's lifespan, respectively. DRAM is used to overcome the out-of-place update drawback. It buffers the frequently updated pages to reduce write traffic directed to flash chips. The flash based component is responsible for reordering the write sequence, which helps to overcome the sequential write constraint.

The DRAM can adopt any existing high performance cache replacement policy, such as 2Q [8], ARC [9], and LIRS [10]. So, it achieves higher hit ratio, therefore reduces write traffic significantly compared with other SSD-oriented write buffers. The flash based component reorders the write sequence directed to flash memory. This optimization is neglected by traditional write buffers completely. As the virtual buffer covers both two factors affecting the lifespan, it's not surprising that, our virtual buffer outperforms other write buffers constantly. We carry out trace driven simulations to evaluate our virtual buffer. Experimental results show that, the virtual write buffer erases less blocks on most workloads compared with CLC [1], PUD-LRU [2], and BPLRU [4]. Operating Systems name the part of hard disk used for virtual memory as swap area. Similarly, the flash based component in the virtual write buffer is named as *reorder area*, just as its function implies.

The rest of paper is organized as follows. Section 2 introduces the backgrounds and related works. Section 3 presents the principle of our virtual buffer elaborately. Section 4 evaluates different write buffers. The last section concludes our work.

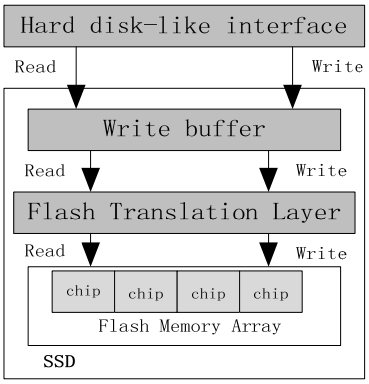

Fig. 1. Overall architecture of SSDs.

2 Backgrounds and related works

The first subsection reviews the typical architecture of SSDs briefly. The second subsection analyzes the negative impacts imposed on SSD's lifespan by the two drawbacks of flash memory. They are the motivations of our work. The last subsection introduces some related works.

2.1 SSD Overview

Compared with hard disks, flash memory has some peculiarities. It cannot be applied to traditional storage system directly. To be compatible with existing software stack, several flash chips are packaged into an SSD. A typical architecture of SSD is shown in Figure 1. SSD exhibits the same interface with hard disks by employing the Flash Translation Layer (FTL). FTL hides these peculiarities of flash memory. Due to the out-of-place update characteristic, flash memory can't overwrite a page. An alternative method to accommodate updates is to direct updated data to other free physical pages. The original pages are marked as invalid. In other words, a logical page can be written to any free physical page. Correspondingly, a logical address would point to a new physical page once its data are updated. The dynamic mapping relationship between logical and physical addresses is maintained by FTL. Ideally, a logical address can map with any physical page. However, the fully-associative mapping between physical and logical pages requires a large table to maintain the mapping entries. Instead, practical SSDs usually map a logical block with a physical block. Logical pages (in the logical block) and physical pages (in the physical block mapping with the logical block) that have the same offset in respective blocks maps with each other by default. This mapping scheme is usually named as block-level mapping. This coarse-grained mapping scheme leads to a more compact mapping table. Besides the address mapping, FTL is also responsible for reclaiming invalid pages. As updated data is directed to other free pages, the original data is invalid. FTL must convert these invalid pages to be free to guarantee that, SSD supplies constant

storage capacity to users. The erase operation is used for converting invalid pages to be free. The process reclaiming invalid pages is called garbage collection.

As shown in Figure 1, there is a write buffer between SSD's interface and FTL. Write buffer is mostly used for decreasing write traffic to flash array. Since the write requests are reduced, fewer pages become invalid. Garbage collection erases fewer blocks to reclaim these invalid pages. Thus, SSD's lifespan is extended. However, we argue that, the lifespan is not determined by the quantity of write traffic completely. Usually, SSD's lifespan is depleted by inner-generated write traffic, rather than user's write requests. Reducing the inner-generated write traffic is a more attractive way to extend SSD's lifespan. Supposing that a block contains some invalid pages, to reclaim these invalid pages, the whole block must be erased (erase operation is performed at block level). Unfortunately, the block still contains some valid pages, which must be moved to other locations. Moving these valid pages leads to additional write requests generated by SSD itself, rather than by user. This phenomenon is called write amplification [11]. So, write buffer should not only reduce write traffic, but also overcome write amplification.

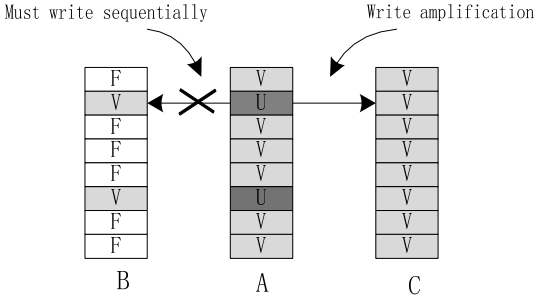

Fig. 2. Write amplification induced by out-of-place update and sequential write constraint.

2.2 Write Amplification Analysis

Practical SSDs usually map physical address with logical address at the granularity of block. Pages in the two blocks are mapped sequentially. Block is the atomic unit of address management. So, updates to any page in a logical block lead to rewriting the entire block to another physical block. Figure 2 shows this drawback elaborately.

Figure 2 presents the negative impacts imposed on SSD's lifespan by out-of-place update and sequential write constraint. As shown in the figure, we assume that a block contains 8 pages. Pages marked by V contain valid data. Pages marked by U had been updated. Pages marked by F are free and have not been written.

In Figure 2, all pages of block A are originally valid. Two pages (marked by U) are to be updated. Since flash cannot overwrite these pages, it must direct updated data to other pages. Consequently, another block B is allocated to accept the updates. As block-level mapping scheme demands that two pages (coming from logical and physical blocks, respectively) with the same offset map with each other, ideally, we expect that updates are directed to the new block with their page offset unmodified. As shown in block B, two pages contain valid data. Unfortunately, this is not permitted by flash memory due to the sequential write constraint. Pages in a block

must be written sequentially beginning from the first one. Instead, we update the two pages in another manner illustrated by block C. In this manner, not only are the updated pages written to block C, but also these unchanged pages are moved to block C as well. This update policy conforms to the principle of block-level mapping scheme. Pages in physical and logical blocks are mapped with their offsets unmodified. However, this policy requires moving valid pages. Even though there are only two pages updated, SSD moves six additional pages. The process of moving these still valid pages to the new block is named as padding [4]. Padding leads to write amplification directly.

Based on the discussion, we can conclude that, two factors lead to write amplification. One is out-of-place update. The other is the sequential write constraint. The first factor requires that, updated data must be written to a new block. The second factor does not allow writing the new block randomly. Ultimately, it needs to rewrite the whole block, which results in write amplification. Write amplification impairs the lifespan remarkably. Write buffer is an effective way to reduce write amplification. But, existing write buffer management policies mostly focus on reducing write traffic, which helps to reduce the chance of out-of-place updates. They scarcely pay attention to overcome the sequential write constraint. The next subsection will introduce some typical write buffer policies in detail. By contrary, our virtual write buffer not only reduces the updates, but also makes write sequence to be sequential. So, it achieves superior performance.

2.3 Related Works

Write buffers for SSDs are different from buffers of hard disks. Buffers of hard disks try to encapsulate several trivial requests into a long sequence, reducing the chance of seek and rotation of disk heads. Buffers of SSDs aim at reducing write traffic and alleviating write amplification. The first goal requires frequently updated pages to be kept in write buffer for as long time as possible. The last goal demands to reduce padding. They conflict with each other sometimes. Write buffers for SSDs mostly compromise between the two.

CFLRU. Clean first LRU (CFLRU) [13] is a policy used for managing the cache of both write and read requests. For flash memory, write is more expensive than read, so, write requests should be kept in cache for a longer time. CFLRU always attempts to select a clean page as the replaced victim. Dirty pages are protected. CFLRU is found to be able to reduce the average replacement cost by 26% compared with LRU. But, it deals with the cache containing both read and write. The scope of this paper is constrained to write buffer only.

FAB. Flash aware buffer (FAB) [5] is also a policy for write buffer of SSDs. In the write buffer, all pages coming from the same block are packed into a group. All the groups are organized into an LRU list. Once a group is accessed, or a new page joins the group, the group is moved to the MRU-end (most recently used). If replacement is needed, FAB selects the group having most pages as the victim. If more than one groups meet this demand, the least recently used group is evicted. FAB evicts the group with most pages to reduce padding pages, despite that the evicted group may be accessed frequently. The arbitrary policy may lead to thrash.

BPLRU. Just like FAB does, Block Padding LRU (BPLRU) [4] also packs pages belonging to a block into a group. It selects the least recently used group as the victim. Maybe, some pages of the block are not in write buffer. BPLRU reads them from flash memory and pads the block, then writes the whole block to flash memory. PBLRU pads blocks explicitly before they are written to flash.

CLC. FAB evicts the largest group. BPLRU evicts the least recently used group. CLC [1] incorporates these two policies together. It divides the write buffer into two parts. One part employs BPLRU policy, the other employs FAB. Compared with FAB, it protects some of the recently used large groups. Compared with BPLRU, it gives higher priority to large groups to be replaced, because replacements of large groups reclaim more free capacity for write buffer.

PUD-LRU. Hu et. al. proposed a new erase-efficient write buffer management algorithm called PUD-LRU, PUD-LRU [2] divides blocks in the write buffer into two groups based on a combined measure of frequency and recency, Predicted average Update Distance (PUD). One group is used for keeping frequently or recently updated blocks. The other is used for selecting replaced blocks. It evicts the block with the largest PUD value, because the block is considered to be the least frequently and recently updated. By doing so, PUD-LRU maximizes the efficiency of destaging operation.

Wu. [3] et. al. proposed a hybrid page/block write buffer architecture along with an advanced replacement policy, called BPAC [3]. BPAC also divides the buffer into two parts. One exploits temporal locality, the other exploits spatial locality. BPAC analyzes the workloads constantly and adapts to workloads dynamically. Sun. et. al. had made use of Phase Change Random Accessed Memory (PCRAM) as SSD's write buffer [6]. Griffin [7] extends SSD's lifespan by disk-based write buffer.

In conclusion, write buffers described above mostly focus on reducing write traffic by caching frequently updated pages. Some of them reduce padding pages by evicting the block containing the most updated pages. All of them fail to optimize write order.

3 The Virtual Write Buffer Architecture

Our virtual write buffer consists of two components. The DRAM is used to cache frequently updated pages, thus reduce write traffic. The reorder area is responsible for reordering write sequence, thus decreasing padding pages. They are introduced in the next two subsections, respectively.

3.1 The DRAM Component

DRAM used in the virtual write buffer is absolutely a traditional cache. It attempts to achieve as higher hit ratio as possible. By contrary, write buffer management schemes described above do not center around hit ratio completely. Take CLC as an example, it not only pursues higher hit ratio, but also tries to evict the largest group to decrease padding pages. These two goals may conflict with each other. Therefore, write buffers adopting CLC usually obtain poor hit ratio.

To achieve higher hit ratio, we manage the DRAM at page level (Contrarily, several write buffers described above pack some pages belonging to a block into a group, they are managed at block level). Only frequently accessed pages are prevented from being evicted. As the DRAM purely exploits temporal locality, which is also pursued by traditional caches, so, some classical cache replacement policies, such as 2Q [8], ARC [9], and LIRS [10], can be reused. This is an important superiority of our virtual buffer over the others. Exactly, we adopt ARC [9] as the replacement policy for DRAM. ARC is so superior a cache replacement policy that, it has been adopted by ZFS [14] and IBM DS8000 storage server. The rationale of ARC is explained as follows.

ARC is well-known due to its simplicity and high performance. It maintains two caches, a physical cache and a virtual cache (the virtual cache is also named as ghost cache). The physical cache is organized into two lists. One list focuses on frequency, the other focuses on recency. The ghost cache contains pages that had been evicted out of the physical cache recently. Actually, the ghost cache only keeps the index entries of these pages. Data in these pages are discarded. In other words, the ghost cache just stores replaced history. Replacements in the physical cache are guided by the feedback of the ghost cache. If a page hits in ghost cache, and it had been evicted from the recency list in physical cache, it means that pages in the recency list are more valuable. ARC gives priority to replacing the pages in frequency list. On the other hand, if an access hits in the ghost cache, and the target page had been evicted from the frequency list in physical cache, ARC is prone to replace pages in the recency list. ARC adapts itself to the workloads continually and dynamically by changing the size of each list. So, it achieves high hit ratio on most workloads.

Compared with conventional write buffers of SSDs, the DRAM achieves high hit ratio for two reasons. The first is to manage the buffer at page level, which helps to exploit temporal locality aggressively. The second is to inherit the classical high performance cache replacement policy. Higher hit ratio means that, the write traffic is reduced significantly.

3.2 Reorder area

The reorder area is a part of SSD's flash address space. It is used for accommodating out-of-order updates. To explain the reordering process clearly, we first introduce a concept, *prior page*. For a given page, its prior pages are those belonging to the same block, but their page offsets are smaller than that of the given page. For example, in logical block 0, Page 2 has smaller offset than Page 3 does, Page 2 is one of the prior pages of Page 3. Similarly, Page 1 is also a prior page of Page 3. The rationale of reordering write sequence can be summarized as follows. When a page is evicted from DRAM, the virtual write buffer detects whether there are some prior pages of the evicted page existing in DRAM. If there is such a page in DRAM, the evicted page cannot be written to its physical block sequentially, because a page prior to it hasn't been written to the block. Instead, the page is written to the reorder area to wait for the right sequence. The process will be explained in detail by resorting to Figure 3. In the figure, there are seven scenarios denoted by S_i, where i belongs to {0, 1, 2, 3, 4, 5, 6}. On each scenario, DRAM evicts a page out.

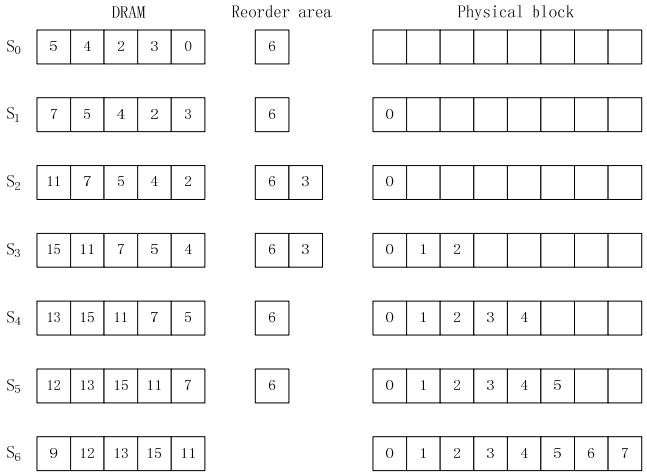

Fig. 3. The process of reordering.

On Scenario 0, the size of DRAM is five pages. The page numbered by 6 is in the reorder area. In another word, Page 6 had been written to flash due to its out-of-order update. A free physical block is prepared for accepting replaced pages. As DRAM is full, the page numbered by 0 is going to be replaced.

On Scenario 1, Page 0 is replaced. As it is the first page of a logical block, it should be written to the first page of a physical block as well, because a page in the logical block has to locate in the same offset in corresponding physical block for block level mapping scheme. Since there is no prior page of Page 0 in DRAM, Page 0 can be written to its physical block directly. There is no need to write it to the reorder area.

On Scenario 2, Page 3 is replaced. However, Page 2 is still in DRAM, and it is a prior page of Page 3. Therefore, Page 3 cannot be written to the physical block sequentially. Instead, it's directed to the reorder area.

On scenario 3, Page 2 is replaced from DRAM. As none of the pages prior to it (such as Page 0 and 1) is in DRAM, Page 2 could be written to its physical sequentially. However, it requires reading Page 1 from flash memory to pad the block. As shown in the figure, Page 1 and 2 are written to the physical block sequentially. Reading Page 1 and writing it back to flash are I/O requests generated by SSDs, rather than by users.

On Scenario 4, Page 4 is replaced. In order to write Page 4 in the physical block sequentially, it needs to pad Page 3 beforehand. As a result, Page 3 and 4 are written to the physical block. So, when Page 5 is replaced, it can be written to the block directly, as presented on Scenario 6. When Page 7 is replaced from the DRAM, we pad Page 6 as well, as shown on the last scenario.

In conclusion, the DRAM doesn't concern about the write sequence. It evicts pages depending on temporal locality and the adopted cache replacement policy. If a page is replaced, the write buffer considers whether it is able to be written sequentially. If so, the page is written to flash directly, otherwise, the page is directed to reorder area, which is also a part of flash address space. Whether a page can be written to flash

directly is determined by whether there are some prior pages in the DRAM. If there are no prior pages in DRAM, the replaced page can be written to flash directly. But maybe, it needs to pad some pages.

The virtual buffer can be compared with other write buffers as follows. For a write buffer mentioned in the related works, when a page is evicted out of buffer, the entire block containing the page is replaced. All the absent pages are padded to the block. Then, the block is written to flash. For our virtual write buffer, replacement of a page does not influence other pages in the same block. This is the first improvement to traditional write buffers. We don't write the entire block to flash arbitrarily, but write its pages one by one. For a given page, its prior pages won't be padded to the block until the given page is replaced from DRAM. As these prior pages haven't been written to the target physical block, updates to them will not lead to write amplification. Delaying page padding as late as possible is the second improvement to traditional write buffers. To keep the sequential write order, we employ the reorder area to register these out-of-order replaced pages.

Actually, the reorder area is a part of flash address space. Directing out-of-order replaced pages to reorder area still shortens SSD's lifespan. But, as the DRAM is able to filter more updates than other write buffers do, and the padding pages is less as well, our virtual buffer erases less blocks all the same compared with others, as shown in the next section.

4 Performance Evaluation

This section evaluates the performance achieved by our virtual write buffer. The first subsection introduces the experimental setup. Subsection 2 compares our virtual write buffer with other buffers. Actually, the virtual buffer is similar with an FTL called FAST (Fully Associative Sector Translation) [15]. The last subsection compares the virtual write buffer with FAST.

4.1 Experimental Setup

We evaluate different write buffers by trace driven simulations. Write buffers mostly aim at extending SSD's lifespan, so, performance is measured by the number of erased blocks. We expect that the erased blocks are as few as possible.

Characteristics of adopted traces are listed in Table 1. They are all enterprise class traces delegating typical applications. Among these traces, *SRC*, *USR*, *Proj* and *Proxy* were gathered from servers at Microsoft Research Cambridge. *LiveMaps* was gathered from *LiveMaps* backend server for duration of 24 hours. *Develop* was collected from Developers Tools Release server for a duration of a day. *MSNFile* was collected from MSN Storage file server for duration of 6 hours. *Exchange* was collected from Exchange Server for duration of 24 hours. All the traces can be downloaded from internet [16]. They were used by Dushyanth N. [17].

The flash chip configuration adopted in experiments is summarized as follows. The size of a page is 4KB. A block contains 64 pages. These parameters come from the specification of a flash chip with small blocks from Sumsung [18].

Table 1. Characteristics of trace.

Traces	Read	Write	R/W
USR	41362884	3837166	10.78/1
SRC1	43544675	2155325	20.2/1
SRC2	21110497	16289503	1.3/1
Proj	47281632	9818368	4.82/1
Proxy	110389290	58210710	1.9/1
LiveMaps	38616344	11033656	3.5/1
Develop	12324273	5865727	2.1/1
MSNFile	19725455	9614545	2.05/1
Exchange	25246721	31823279	1/1.26

4.2 Comparison with Other Buffers

We compare the virtual write buffer with BPLRU [4] and PUD-LRU [2]. BPLRU is a classical write buffer policy for SSDs. It has been extensively evaluated. PUD-LRU is a recently proposed policy achieving high performance. It inherits the design principle of classical cache replacement policies.

The experimental results are presented in Figure 4. As shown in the figure, SSDs employing the virtual write buffer erase the least physical blocks on most workloads. To further explain why the virtual buffer achieves higher performance, we compare the hit ratios and the numbers of padding pages related to different write buffers, which are presented in Figure 5 and 6, respectively. Higher hit ratio means that the write traffic is reduced more remarkably. So, we expect the hit ratio to be as high as possible. Inversely, padding pages are additional write traffic generated by background operation. A superior write buffer should help to reduce padding pages, thus alleviate write amplification. So, we expect that padding pages are as few as possible. As padding a block indicates migrating pages from one physical block to another, hereafter, we refer padding pages as *padding pages* or *migrated pages*, interchangeably.

BPLRU and PUD-LRU pack updated pages in a logical block into a group. Groups are the atomic units for replacement in write buffer. BPLRU organizes these groups into an LRU list. It always replaces the least recently updated group. Hu et al. argued that, the LRU list only focuses on recency, but neglects access frequency [2]. They proposed PUD-LRU to exploit both of the two aspects. PUD-LRU improves BPLRU by employing a new measurement called Predicted average Update Distance, rather than the simple recency. So, PUD-LRU usually erases less blocks compared with BPLRU, as shown in Figure 4. Actually, the importance of frequency is extensively evaluated in the research domain of cache replacement policy.

Policies discussed above mostly aim at improving hit ratio and reducing write traffic to flash memory. But, as shown in Figure 5, hit ratios achieved by different write buffers are low. Even when the cache size is 64MB, the write buffer still cannot reduce write traffic significantly. Only on workloads *Prxy*, the erased blocks are reduced with the expanding of buffer size. So, simply using write buffer to reduce write traffic is not an effective way to extend SSD's lifespan.

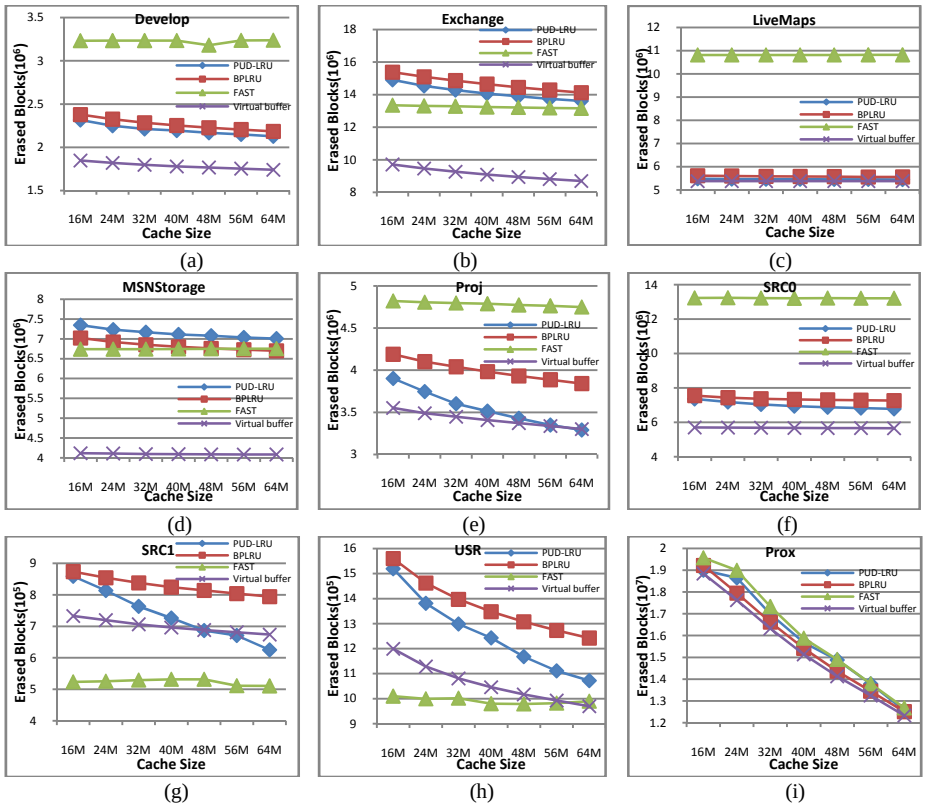

Fig. 4. SSDs employing our virtual write buffer erase the least blocks on most workloads. That's because our virtual write buffer optimizes both cache replacement and write sequence.

We propose to reorder the write sequence in virtual write buffer. Reordering write sequence avoids writing physical blocks randomly. Usually, random write results in write amplification, which will shorten SSD's lifespan. As shown in Figure 6, SSDs employing the virtual write buffer move the least pages for padding. It means that the write amplification is prohibited. That's why SSDs equipped with the virtual buffer erase fewer physical blocks. Actually, out-of-order written pages are directed to the reorder area, which is a part of flash address space. These pages also deplete SSD's lifespan. But, these pages can be written to any physical pages in flash chips. There is no need to migrate any page for them. They will not bring about write amplification. So, the erased blocks induced by out-of-order written pages are negligible.

Except for the reordering optimization, our virtual buffer does not neglect hit ratio. The DRAM devotes to improving hit ratio. DRAM is managed at the page level. It is able to exploit temporal locality completely. Furthermore, it can reuse existing high performance cache replacement policies. So, the DRAM achieves higher hit ratios on most workloads, as shown in Figure 5. Higher hit ratio means to reduce write traffic. This is pursued by traditional write buffers. Based on the two aspects (higher hit ratio and reordering optimization), the virtual write buffer reduces erased blocks.

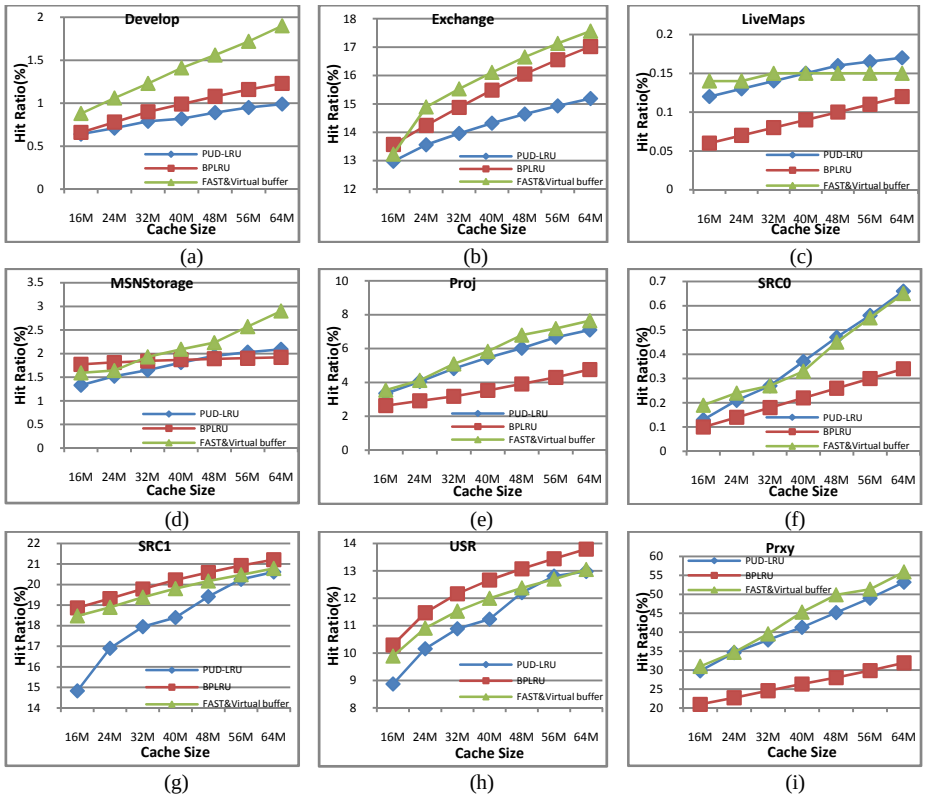

Fig. 5. The virtual write buffer achieves higher hit ratio than others do.

4.3 Comparison with FAST

Reorder area is similar with the log area adopted by FAST [15]. So, we compare the virtual write buffer with FAST further. The rationale behind FAST can be described as follows. FAST allocates a set of blocks called log blocks. Left blocks used by users are called data blocks. Data blocks contain most of the user's data. Log blocks are prepared to accept updated pages. The log set is shared by all data blocks. Updates to any data blocks are directed log blocks sequentially. If the log set is used up, FAST merges valid data in the log set to data blocks. And another log area is allocated. These log blocks are similar with the reorder area adopted by our virtual write buffer. To compare the two schemes fairly, we assume that FAST is equipped with the same DRAM as our virtual write buffer owns. And it adopts ARC [9] as the cache replacement policy as well. The difference between two schemes relies on the direction of updated pages evicted from DRAM. For FAST, pages evicted from DRAM are written to log blocks arbitrarily. When the log set is used up, log blocks are merged with data blocks. For our virtual buffer, SSD tries its best to write pages to data blocks directly. Only when a page is evicted from DRAM in out-of-order manner, the page is written to the reorder area.

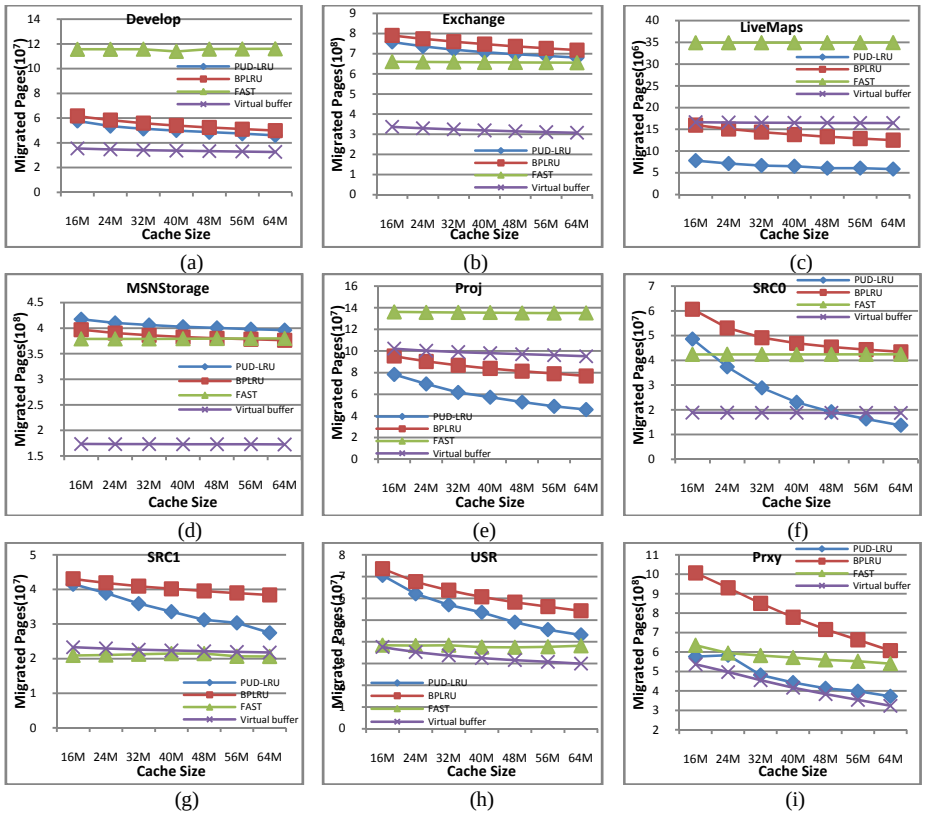

Fig. 6. SSDs employing the virtual write buffer migrate the least number of pages.

As two schemes adopt the same cache replacement policy to manage DRAM, they achieve the same hit ratios. But, the virtual buffer usually erases fewer blocks. FAST writes all updates to log area arbitrarily. When the log area is used up, it merges log blocks with data blocks. The merge operation requires to moving many pages. As shown in Figure 6, FAST usually moves more pages than the virtual write buffer does. On the contrary, the virtual buffer gets rid of merge operation. It reorders the write sequence, and then writes pages of a logical block to a physical block with page offset unchanged. There are no page migrations except for padding. So, the virtual buffer erases less blocks.

5 Conclusion

SSDs usually employ write buffer to extend the lifespan. However, existing write buffers haven't optimized the write sequence. We propose a virtual write buffer. The buffer consists of two components, the DRAM and reorder area. DRAM devotes to improving hit ratio, as pursued by traditional write buffers. Since DRAM is managed at page level and adopts existing superior cache replacement policy, it gets higher hit

ratio. The reorder area reconfigures the write sequence, thus prohibits write amplification. Based on the two optimizations, the virtual write buffer enables SSDs to erase less physical blocks, and thus to extend the lifespan.

Acknowledgments. We are grateful to the anonymous reviewers for their valuable suggestions to improve this paper. This work is supported by the National Natural Science Foundation of China (NSFC60736013, NSFC60903040, NSFC61025009), Program for New Century Excellent Talents in University (NCET-08-0145).

References

1. S. Kang, S. Park, H. Jung, H. Shim, and J. Cha, Performance trade-offs in using nvram write buffer for flash memory-based storage devices. IEEE Transactions on Computers, vol. 58, no. 6, 744--758 (2009)
2. Jian Hu, Hong Jiang, Lei Tian, Lei Xu. PUD-LRU: An Erase-Efficient Write Buffer Management Algorithm for Flash Memory SSD. In: MASCOTS'10. August, (2010)
3. G. Wu, B. Eckart, and X. He, BPAC: An Adaptive Write Buffer Management Scheme for Flash-based Solid State Drives. In: MSST2010, May, (2010)
4. H. Kim and S. Ahn, Bplru: A buffer management scheme for improving random writes in flash storage. In: FAST2008. (2008)
5. H. Jo, J. U. Kang, S. Y. Park, J. S. Kim, and J. Lee, Fab: flashaware buffer management policy for portable media players. IEEE Transactions on Consumer Electronics, vol. 52, no. 2, 485--493 (2006)
6. G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement. in: HPCA2010. IEEE, 141--153, Jan, (2010)
7. G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, Extending ssd lifetimes with disk-based write caches. In: FAST2010. USENIX, Feb, (2010)
8. Johnson T, Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In: VLDB1994. Santiago de Chile, 297--306. (1994)
9. Megiddo N, Modha D S. ARC: a self-tuning, low overhead replacement cache. In: FAST2003. San Francisco, 115--130 (2003)
10. Jiang S, Zhang X D. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: SIGMETRICS2002. 31--42. 2002.
11. X. Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In: SYSTOR 2009. (2009)
12. Chanik Park, Jeong-Uk Kang, Seon-Yeong Park, and Jin-Soo Kim. Energy-aware demand paging on NAND flashbased embedded storages. In: ISLPED2004. New York, NY, USA, 338--343 (2004)
13. Seon-Yeong Park, Dawoon Jung, Jeong-Uk Kang, Jin-Soo Kim, and Joonwon Lee. CFLRU: a replacement algorithm for flash memory. In: CASES2006, New York, NY, USA, 234--241 (2006)
14. ZFS Introduction. http://en.wikipedia.org/wiki/ZFS
15. S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song. A Log Buffer based Flash Translation Layer Using Fully Associative Sector Translation. IEEE Transactions on Embedded Computing Systems, 6(3):18. ISSN 1539–9087. (2007)
16. http://iotta.snia.org/traces.
17. Dushyanth Narayanan, Austin Donnelly, and Antony Rowstron. Write Off-Loading: Practical Power Management for Enterprise Storage. In: FAST08. (2008)
18. K9WBG08U5M-KCJ0 Datasheet. www.samsung.com.