CAVE: Channel-Aware Buffer Management Scheme for Solid State Disk

Sung Kyu Park, Youngwoo Park, Gyudong Shim, and Kyu Ho Park
Korea Advanced Institute of Science and Technology (KAIST)
305-701, Guseong-dong, Yuseong-gu, Daejeon, Korea
{skpark, ywpark, gdshim, kpark}@core.kaist.ac.kr

ABSTRACT

Exploiting a multi-channel architecture of Solid State Disk (SSD) is a role of Flash Translation Layer (FTL) until now. A multi-channel FTL scheme increases I/O parallelism by spreading out pages in a logical block to multiple channels. However, this scheme has high garbage collection overhead for reclaiming invalid pages, thus resulting in the performance degradation. In order to overcome this problem, we assign a write buffer to exploit a multi-channel architecture. In this paper, we propose a novel buffer management scheme, called Channel-Aware Victim Eviction (CAVE). The key idea of the CAVE scheme is to evict multiple victims whose number is equal to the number of NAND flash memories when a write buffer is full for increasing I/O parallelism. Because a write buffer exploits a multi-channel architecture, we can use a 1-channel FTL scheme, thus reducing garbage collection overhead. We develop a trace-driven simulator for evaluating the CAVE scheme. The hit ratio and execution time are used as performance metrics. In the hit ratio, the CAVE scheme can achieve a similar result with a conventional method which uses a multi-channel FTL scheme though it evicts more victims at a time. In the execution time which consists of the write time and garbage collection time, a result shows that the CAVE scheme can reduce it by 55.5% - 97.4% in block-level LRU using SYSMARK compared to the conventional method.

Categories and Subject Descriptors

D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*

General Terms

Design, Management, Performance, Experimentation

Keywords

Solid State Disk, Write buffer, Multi-channel architecture, I/O parallelism, Garbage collection

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

1. INTRODUCTION

Nowadays, multiple NAND flash memories are widely used in a personal or server storage system as a form of Solid State Disk (SSD) constructed in a multi-channel architecture [3] [6] [18]. SSD embeds a special layer, called Flash Translation Layer (FTL), to emulate a block device interface by hiding erase-before-write characteristic of NAND flash memory.

The main roles of FTL are address mapping and garbage collection. Address mapping is necessary to translate a logical address from a file system to a physical address of NAND flash memory. Garbage collection is necessary to reclaim invalid pages of a block. Many researches like BAST [9], FAST [11], and LAST [12] have an effort to optimize address mapping and garbage collection schemes. FTL also takes a role in exploiting a multi-channel architecture of SSD. Thus, multi-channel FTL schemes like MCsplit [10] and Subgroup [14] are developed.

Multi-channel FTL schemes spread out pages in a logical block across multiple channels for increasing I/O parallelism. However, this scheme can bring about high garbage collection overhead. The reason is that many blocks are associated with garbage collection, thus making many erase operations and valid page copies. Thus, multi-channel FTL schemes would rather be able to reduce the performance compared to 1-channel FTL schemes because garbage collection overhead is higher than the effect of I/O parallelism.

We are motivated by this high garbage collection overhead problem. Thus, we assign a write buffer to exploit a multi-channel architecture instead of FTL. In this paper, we propose a novel buffer management scheme, called Channel-Aware Victim Eviction (CAVE). The key idea of the CAVE scheme is to evict multiple victims when a write buffer is full. The number of victims selected at a time is equal to the number of NAND flash memories in SSD for achieving full I/O parallelism. Because a write buffer exploits a multi-channel architecture, we can use a 1-channel FTL scheme which has low garbage collection overhead instead of a multi-channel FTL scheme. Thus, we can achieve high I/O parallelism and low garbage collection overhead.

There are many previous researches to exploit a write buffer such as CFLRU [15], FAB [5], BPLRU [8], REF [16], and CLC [7]. They have an effort to reduce the number of write requests, increase a sequentiality of written data, and reduce garbage collection overhead, thus improving the performance of SSD. However, they only focus on the eviction order. In these schemes, one victim is evicted when a write buffer is full. It is hard to exploit a multi-channel

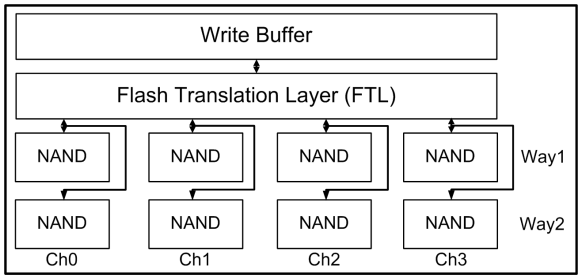


Figure 1: Internal Architecture of SSD

architecture without a multi-channel FTL scheme. Different from previous buffer management schemes, the CAVE scheme considers the eviction rate which means how many victims are evicted when a write buffer is full. Therefore, we can achieve I/O parallelism in a multi-channel architecture by evicting multiple victims.

The CAVE scheme has another advantage that is possible to combine the CAVE scheme with previous buffer management schemes. This is because the CAVE schemes only considers the eviction rate, but previous ones only focus on the eviction order. Thus, we can exploit both strengths of the CAVE scheme and previous buffer management schemes such as the high hit ratio, high I/O parallelism, and low garbage collection overhead.

The remainder of this paper is organized as follows. This paper starts from background which is about Solid State Disk (SSD). In section 3, we describe previous buffer management schemes and FTL algorithms. We then describe our proposed buffer management scheme in section 4. The performance evaluation results are placed in section 5 and section 6 concludes this paper and presents further work.

2. SSD BACKGROUND

Solid State Disk (SSD) is made up of three main components: Write buffer, Flash Translation Layer (FTL), and NAND flash memories as shown in Figure 1.

Write Buffer: SSD has a small amount of SDRAM or NVRAM. It is used for storing a mapping table managed by FTL. It is also able to be available for a write buffer. Since small-sized and frequently updated write requests cause high garbage collection overhead, we have to reduce these small-sized write requests. A write buffer whose location is above FTL exploits temporal and spatial localities of write requests. In a write buffer, write requests are updated and reordered, thus reducing garbage collection overhead.

Flash Translation Layer (FTL): SSD has a special software layer, called FTL. FTL makes NAND flash memories to look like Hard Disk Drive (HDD) by translating a logical address from a file system to a physical address of NAND flash memory. Another role of FTL is to recycle invalid pages, called garbage collection which contains block erase operations and valid page copy operations. Because it significantly affects the performance of SSD, FTL should have an effort to reduce garbage collection overhead. Finally, FTL is responsible for prolonging a life time of NAND flash memories, called wear-leveling. Because each cell of NAND flash memory has a limited number of erase count about 10,000 - 1,000,000 times, FTL has to reduce the erase count and make all cells have even erase counts.

NAND Flash Memories: The main storage medium of SSD is NAND flash memory which has non-volatility, reliability, low-power consumption, and shock resistance. SSD uses multiple NAND flash memories for expanding storage capacity and improving the performance. A multi-channel and multi-way architecture is adopted in SSD, thus making possible to access a number of NAND flash memories simultaneously.

3. RELATED WORK

Solid State Disk (SSD) has been extensively studied by many researchers. They have especially focused on developing write buffer management and FTL schemes. In this section, we discover our contributions based on a careful analysis about previous works.

3.1 Write Buffer Management Scheme

In write buffer management schemes for SSD, the eviction order and the eviction rate should be considered. The eviction order is used for exploiting temporal and spatial localities, thus increasing the hit ratio of a write buffer and reducing garbage collection overhead. The eviction rate means how many victims are evicted when a write buffer is full. It is important to increase I/O parallelism.

There are many related works about buffer management schemes which consider the eviction order. Park et al. [15] proposed Clean-First LRU (CFLRU) scheme which evicts a clean page in a clean-first region, thus reducing the replacement cost. Jo et al. [5] presented Flash-aware buffer management policy (FAB) which selects a victim block that has many valid pages, thus reducing garbage collection overhead and increasing the hit ratio by delaying to evict small metadata files which are frequently updated. Kim et al. [8] proposed Block Padding Least Recently Used (BPLRU) scheme which improves the random write performance by delaying to evict a frequently used block and increasing the number of switch merge operations instead of the full merge operations. Seo et al. [16] proposed Recently-Evicted-First (REF) scheme which chooses a page having same logical block number that is recently evicted as a victim for reducing a block merge number and associativity. Kang et al. [7] introduced Cold-Largest-Cluster (CLC) scheme which selects a cold and large cluster as a victim, thus increasing the hit ratio and avoiding garbage collection overhead.

By considering the eviction order, previous buffer management algorithms improve the performance. However, they don't regard the eviction rate, but evict one victim when a write buffer is full. It is hard to exploit a multi-channel architecture without a multi-channel FTL scheme.

Seol et al. [17] proposed a buffer replacement scheme exploiting multi-chip parallelism by considering the state of NAND flash chips. Thus, it can avoid the chip-waiting problem in SSD. However, this scheme has an overhead for knowing both the mapping information of FTL and the page information of a buffer cache. Because it does not consider the eviction rate, it is also hard to exploit a multi-channel architecture when an interval between requests is too long.

In order to exploit a multi-channel architecture without a multi-channel FTL scheme, we should consider the eviction rate in a write buffer. By considering the eviction rate when designing a write buffer scheme, we can achieve high I/O parallelism and low garbage collection overhead.

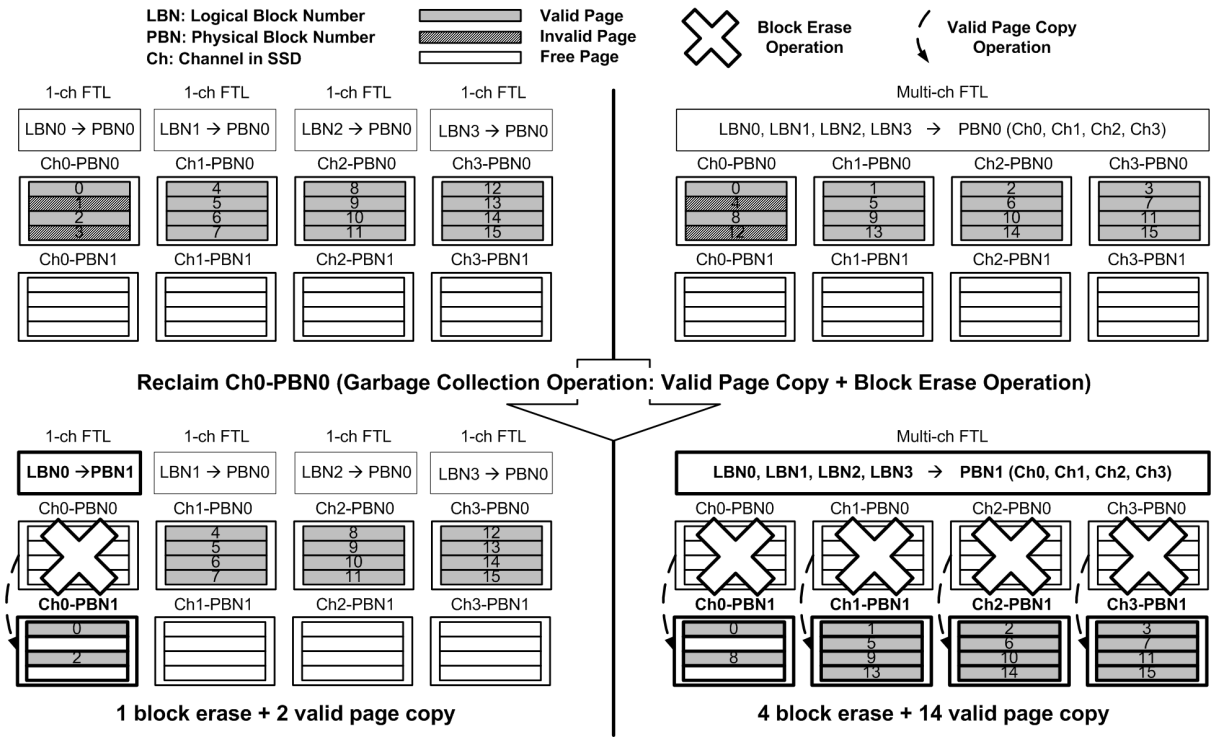


Figure 2: Garbage Collection Overhead with 1-channel FTL and multi-channel FTL

3.2 FTL Scheme

Previous FTL schemes are developed for reducing the garbage collection overhead and increasing I/O parallelism. Kim et al. [9] proposed the BAST scheme that one data block is associated with only one log block. The BAST scheme is nice for the sequential requests, while bad for the random requests because of a low utilization of log blocks, called a block thrashing problem. In order to overcome this block thrashing problem, Lee et al. [11] suggested the FAST scheme that one log block is shared by a number of data blocks. The FAST scheme is good for a space utilization of log blocks, while making many full merge operations which degrade the performance. Lee et al. [12] proposed the LAST scheme which exploits the locality of storage access patterns, thus reducing the merge operation overhead. These 1-channel FTL schemes could use a small memory space and improve the performance. However, they are not appropriate for a multi-channel architecture.

Kim et al. [10] presented the MCSplit scheme with a multi-chip striping configuration, thus reducing the number of expensive erase and data copy operation. Park et al. [14] described the SubGroup scheme which is a new superblock management scheme to overcome the problem of increased erase operations which results from increasing the degree of striping of memory banks. These multi-channel FTL schemes try to improve the performance by exploiting a multi-channel architecture. However, it is hard to satisfy both high I/O parallelism and low garbage collection overhead.

This problem can be solved with help of a write buffer. In this paper, we focus on designing a write buffer management scheme for improving I/O parallelism. By using a 1-channel FTL scheme, we can also reduce garbage collection overhead.

4. CAVE: CHANNEL-AWARE VICTIM EVICTION

This section consists of the motivation and algorithm description of Channel-Aware Victim Eviction (CAVE). We describe the problem of previous approaches and explain how to solve it by using the CAVE scheme.

4.1 Motivation

In order to exploit a multi-channel architecture, Solid State Disk (SSD) uses a multi-channel Flash Translation Layer (FTL) scheme which spreads out pages in a logical block across multiple channels. Although it can increase I/O parallelism, it has high garbage collection overhead. This is because many blocks are associated with garbage collection, thus making many erase and valid page copy operations.

Figure 2 shows garbage collection overhead of a multi-channel FTL scheme compared to that of a 1-channel FTL scheme for describing the motivation of the CAVE scheme. In Figure 2, LBN means a logical block number which consists of multiple logical pages, PBN means a physical block number, and Ch means a channel in SSD. We assume that SSD has four channels and each block has four pages. Each page has three types which are valid, invalid and free.

The right side of Figure 2 shows a garbage collection operation of a multi-channel FTL scheme. Initially, four logical blocks which are from LBN 0 to LBN 3 are mapped to physical blocks which are PBN 0 in multiple channels and page 4 and page 12 are invalid pages. When a garbage collection is invoked for reclaiming these invalid pages, we should copy 12 valid pages and erase 4 blocks in this case. This is because a multi-channel FTL scheme maps pages in one logical block to multiple channels for increasing I/O parallelism.

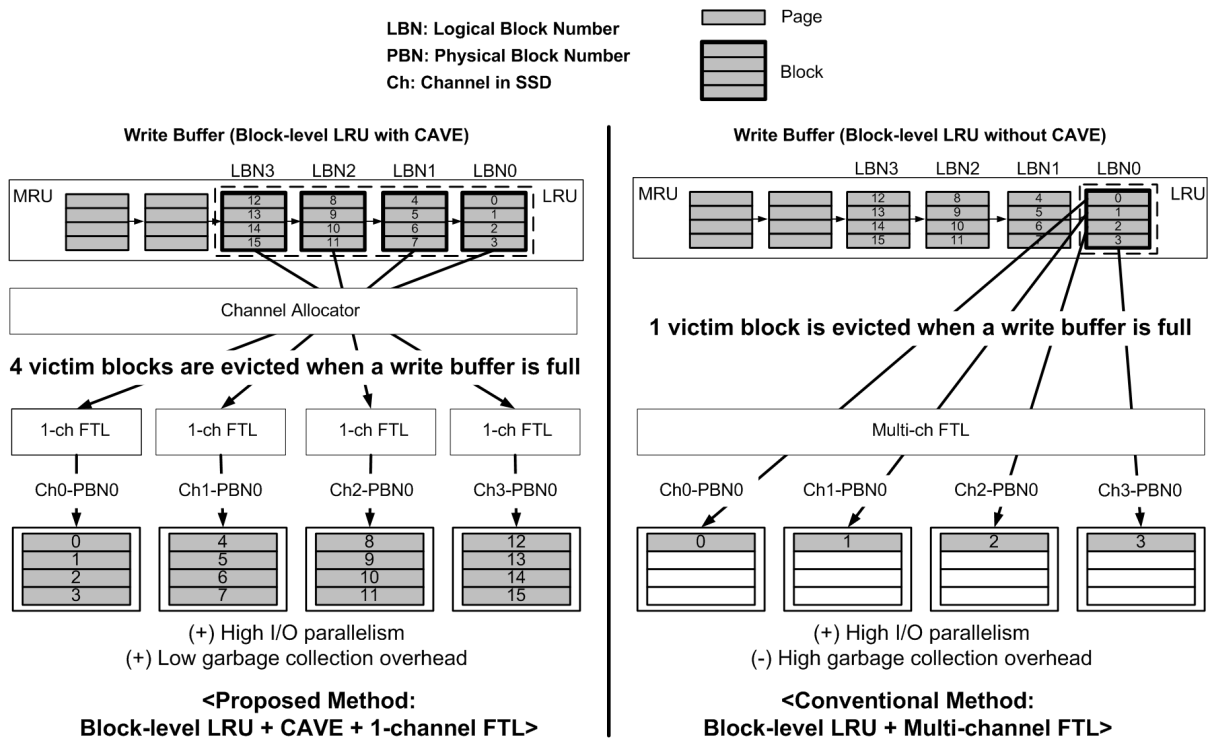


Figure 3: Proposed Method (with CAVE) compared to Conventional Method (without CAVE)

In contrast with a multi-channel FTL scheme, a 1-channel FTL scheme maps pages in one logical block to one channel. For example, LBN 0 is mapped to PBN 0 in channel 0 as shown in the left side of Figure 2. In Figure 2, page 1 and page 3 are invalid pages of PBN 0 in channel 0. In this case, we copy only 2 valid pages and erase only 1 block when a garbage collection operation is invoked.

We show that a multi-channel FTL scheme has higher garbage collection overhead than a 1-channel FTL scheme although it can increase I/O parallelism. We are motivated by this high garbage collection overhead when using a multi-channel FTL scheme. In order to overcome this problem, we assign a write buffer to exploit a multi-channel architecture instead of FTL. For a write buffer, we propose the Channel-Aware Victim Eviction scheme, called CAVE, which increases I/O parallelism and reduces garbage collection overhead, thus finally improving performance of SSD.

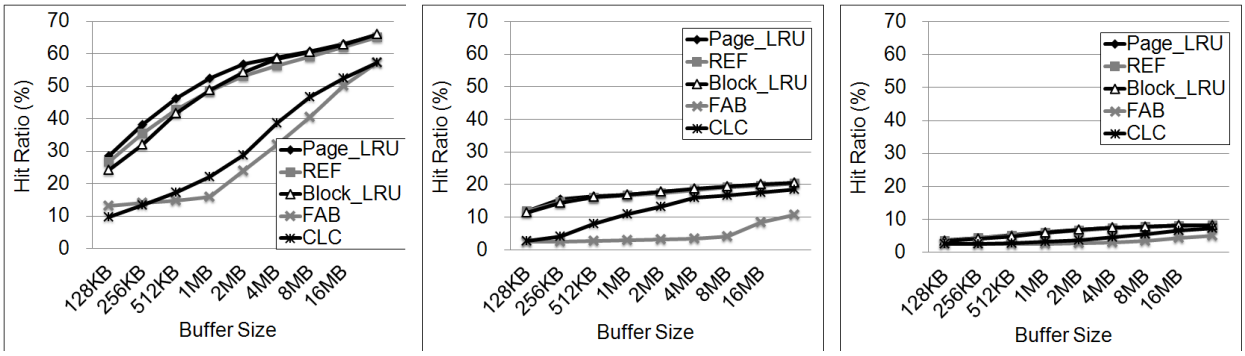
4.2 Algorithm Description

The key idea of the CAVE scheme is to exploit a multi-channel architecture of SSD in a write buffer. To achieve this, we evict multiple victims when a write buffer is full. The number of victims selected at a time is equal to the number of NAND flash memories in SSD for achieving full I/O parallelism. For example, we evict 32 victims from a write buffer if SSD has 8-channel 4-way architecture which contains 32 NAND flash memories. Therefore, we can increase I/O parallelism without a multi-channel FTL scheme. We can also reduce garbage collection overhead by using a 1-channel FTL scheme because a write buffer takes a role in exploiting a multi-channel architecture. The CAVE scheme considers the eviction rate for increasing I/O parallelism. Thus, it is possible to combine the CAVE scheme with previous buffer management scheme which consider the eviction order. By combining these schemes, we can finally achieve both strengths in the CAVE scheme and in previous buffer management scheme, which are the high hit ratio, high I/O parallelism, and low garbage collection overhead.

Figure 3 describes an example of the proposed method that a write buffer exploits a multi-channel architecture compared to the conventional method that a multi-channel FTL scheme exploits it. In this example, a write buffer is managed by a block-level LRU scheme [8], thus a victim unit is a block which consists of four pages.

In the conventional method as shown in the right side of Figure 3, one victim block is evicted when a write buffer is full. The pages in this victim block are distributed to four channels by a multi-channel FTL scheme. Page 0 is allocated to channel 0, page 1 to channel 1, page 2 to channel 2, and page 3 to channel 3. Therefore, it can achieve high I/O parallelism, but having high garbage collection overhead due to the distribution of pages in a logical block. If a victim unit is a page, this method cannot achieve high I/O parallelism either. A victim is too insufficient to exploit a multi-channel architecture.

In contrast with the conventional method, the CAVE scheme evicts four victim blocks when a write buffer is full as shown in the left side of Figure 3. These four victim blocks are allocated to each channel by a channel allocator and each victim block is mapped to a physical block by a 1-channel FTL scheme in each channel. Because one logical block is assigned to one physical block, it has low garbage collection overhead. It can also achieve high I/O parallelism that a write buffer handles to exploit a multi-channel architecture of SSD. If a victim unit is a page, this method can achieve high I/O parallelism unlike the conventional method.



(a) OLTP (b) GENERAL (c) SYSMARK

Figure 4: Hit Ratio: Page-level LRU, REF, Block-level LRU, FAB, and CLC schemes

It is important to assign victims from a write buffer to each channel. In the CAVE scheme, we use a round-robin scheme for allocating the channel number to a block in a channel allocator. It can distribute victim blocks to all channels evenly. This scheme has a problem that some valid pages of LBN 0 are assigned to channel 0, but other valid pages of LBN 0 are assigned to channel 1. In this case, we should move those pages to one channel, thus making an overhead. However, we assume this case does not frequently occur because frequently updated data are absorbed in a write buffer.

Figure 3 seems like the CAVE scheme has some overheads which are the high total write number and the low hit ratio compared to the conventional method because the CAVE scheme evicts more victim blocks at a time. However, the total write number and the hit ratio are similar to the conventional method. This is because the CAVE scheme evicts infrequently updated victim blocks which are located in LRU positions and makes more free spaces to store new requests. When a write buffer is small, the total number of write operations and the hit ratio with the CAVE scheme, of course, is slightly higher. However, its difference is being reduced as increasing the size of a write buffer.

As exploiting a multi-channel architecture in a write buffer, we can increase I/O parallelism compared to the exploitation using a multi-channel FTL scheme. Also, a 1-channel FTL scheme has lower garbage collection overhead than a multi-channel FTL scheme. Therefore, we conclude that the CAVE scheme can improve performance by increasing I/O parallelism and reducing garbage collection overhead.

Although there are many advantages in the CAVE scheme, it has some disadvantages in read operation overhead and disturbance of other requests when evicting victims. In the CAVE scheme, it cannot exploit I/O parallelism in read operations because one logical block is assigned to one channel. Therefore, it has a drawback in sequential read operations. However, we insist that the write speed improvement is much better than the read speed degradation because a read speed is over 10x faster than a write speed in NAND flash memory. The second disadvantage is the disturbance of other requests when evicting victims. This is because we evict victims only when a write buffer is full. In this case, a new request is blocked until victims are fully written to SSD. As the further work, we'll consider the eviction timing like linear-threshold eviction [4].

Table 1: Simulator Specification

Parameters	Values
Channel and Way	8-channel 4-way
Host Interface Bandwidth	190.735 MB/s
Transfer Time	63.37 us

Table 2: NAND Flash Memory Specification

SAMSUNG 1GB SLC NAND	Value
Page Size	2KB
Block Size	128KB
Read Time	25us
Program Time	200us
Erase Time	1500us

5. EXPERIMENT

From now on, we explain the experimental setup such as a simulator organization, comparison, performance metrics, and trace characteristics. We then show the performance results which are the hit ratio and execution time compared with the conventional method.

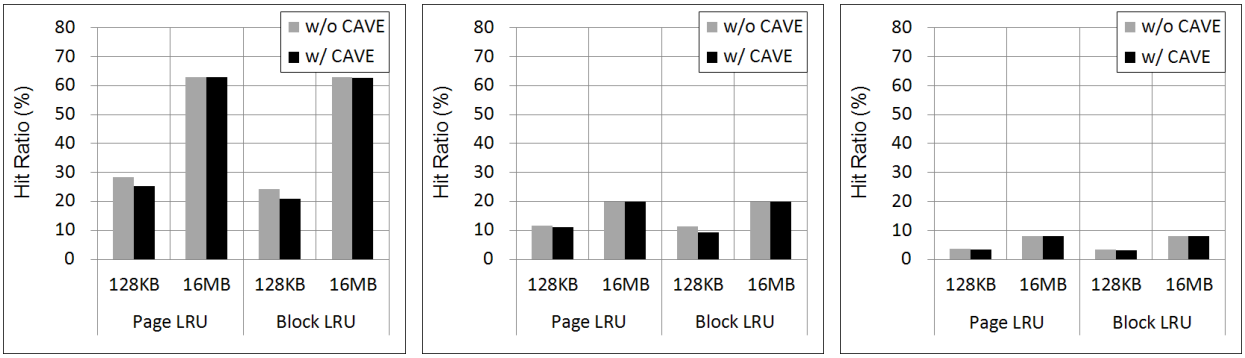
5.1 Experimental Setup

We implement a trace-driven simulator and evaluate the performance of the CAVE scheme using it. The architecture is 8-channel 4-way architecture which has 32 NAND flash memories. This is because current commercial SSDs usually use 8-channel architecture. The hardware specification we use for evaluation is shown in Table 1. We also use specifications of SLC NAND flash memory [1]. Table 2 shows the specifications of NAND flash memory.

In order to evaluate the performance of the CAVE scheme, we have to choose buffer management schemes about the eviction order for combining with the CAVE scheme and a 1-channel FTL scheme. As the comparison, we also choose the buffer management schemes and a multi-channel FTL scheme. For choosing buffer management schemes about the eviction order, we first evaluate the hit ratio of page-level LRU, REF, block-level LRU, FAB, and CLC. Figure 4 shows the hit ratio as increasing a write buffer size. In this result, FAB and CLC schemes show lower hit ratio than other

Table 3: Characteristics of Traces

Traces	OLTP	GENERAL	SYSMARK
Storage	12.5GB	32GB	32GB
Random Pattern	98.7%	69.0%	47.0%
Average Request Size	5.3KB	24.9KB	47.1KB
Write Number	11,639,536	7,933,879	2,131,747
Applications	OLTP applications	Office work, download, web surfing, and installation	E-learning, office work, video creation, and 3D modeling



(a) OLTP (b) GENERAL (c) SYSMARK

Figure 5: Hit Ratio of Proposed Method (with CAVE) and Conventional Method (without CAVE)

previous buffer management schemes because these schemes focus on sequential requests. Among page-level LRU, REF, and block-level LRU, we finally choose page-level LRU and block-level LRU for the eviction order because we have to see both characteristics when a victim unit is a page and a block. As a 1-channel FTL scheme, we use the BAST scheme because it is one of the most famous 1-channel FTL schemes. The MCSplit scheme is selected as a multi-channel FTL scheme. As the conventional method, we finally use both page-level LRU and block-level LRU in the MCSplit scheme. As the proposed method, we use page-level LRU and block-level LRU with the CAVE scheme in the BAST scheme. In further experiments, we'll compare the proposed method with the conventional method.

We use three workload traces for performance evaluation as described in Table 3. The first trace is extracted from OLTP applications running at a large financial institution. This trace is made available courtesy of Ken Bates from HP, Bruce McNutt from IBM, and the Storage Performance Council [2]. The second and third traces are obtained from a Microsoft Windows-based laptop computer [13]. These are GENERAL and SYSMARK which represent the workload of typical PC usage scenarios. GENERAL trace is obtained from a 5-day long general PC usage and SYSMARK trace is collected from SYSmark 2007 Preview.

The main performance metrics are the hit ratio and execution time. As estimation of buffer management scheme, the hit ratio is the most well-known factor. Therefore, we evaluate the hit ratio of previous buffer management scheme with and without CAVE. For estimating write performance of SSD, the execution is also an important factor. It contains the effects of the exploitation of a multi-channel architecture and reduction of garbage collection overhead.

5.2 Result

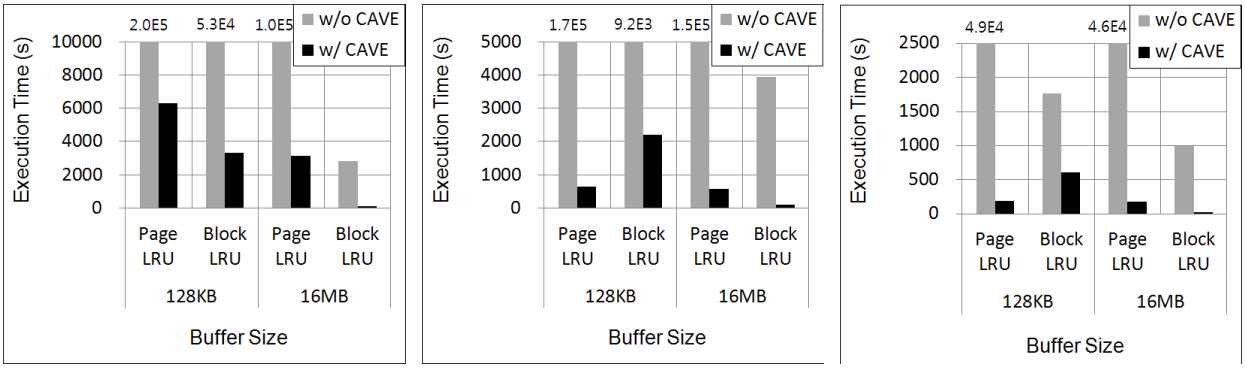
We give experimental results in this section. We show the hit ratio and execution time in 128KB buffer size and 16MB buffer size. By comparing previous buffer management scheme without CAVE in a multi-channel FTL scheme, we show that CAVE can improve performance of SSD.

5.2.1 Hit Ratio

As the first experiment, we compare the hit ratio of the proposed method which is combination of page-level LRU or block-level LRU with the CAVE scheme in a 1-channel FTL scheme to the conventional method which is combination of page-level LRU or block-level LRU with a multi-channel scheme as shown in Figure 5. When a buffer size is small (128KB), the difference between the proposed method and the conventional method is large. The hit ratio of the conventional is higher than the proposed method. The reason is that the CAVE scheme evicts more victims at a time. Therefore, the probability for updating in a write buffer is reduced, thus reducing the hit ratio.

However, Figure 5 shows that the hit ratio of the proposed method and the conventional method is similar when a write buffer size is large (16MB). This result shows that the CAVE scheme has no overhead about the hit ratio. This is because most victims in a large write buffer are located in LRU positions which represent low probability for updating. It also means that the total write number of the proposed method is similar to that of the conventional method. This is because it also makes more free spaces.

Even though the CAVE scheme evicts more victims than the conventional method, this cannot critically affect the hit ratio of a write buffer.



(a) OLTP (b) GENERAL (c) SYSMARK

Figure 6: Overall Performance (Execution Time: Write Time + Garbage Collection Time)

5.2.2 *Execution Time*

As the second experiment, we evaluate the execution time which contains the page write time and garbage collection time. We do not consider buffer write time because the write speed of DRAM is much faster than that of NAND flash memory. From this experiment, we can identify how much performance can be improved by the CAVE scheme.

Figure 6 shows the result of the execution time. The result shows that the execution time of the proposed method is always lower than the conventional method. I/O parallelism of the CAVE scheme is similar with or rather higher than that of a multi-channel FTL scheme. The main reason for the different execution time is garbage collection overhead. A multi-channel FTL scheme spreads out pages in a logical block across multiple channels, but a 1-channel FTL writes pages in a logical block to one channel. It makes a difference of the performance between the proposed method and the conventional method. A page-level LRU shows much higher garbage collection overhead than a block-level LRU due to page-level LRU scheme does not consider the characteristics of SSD like erase-before-write. In contrast with a page-level LRU, a block-level LRU shows relatively low garbage collection overhead. Also, the execution time in 16MB is much better than that in 128KB because a large size buffer can reduce the total number of write operations to SSD.

Figure 7 shows the portion of garbage collection time in the overall performance. The result shows that garbage collection time is a critical factor of the overall performance. Using a multi-channel FTL scheme, it cannot reduce garbage collection overhead, thus showing the high execution time. In small size buffer, the portion of garbage collection overhead is higher than that in large size buffer. The reason is that the total write number in small size buffer is higher than that in large size buffer. The high total write number is also one reason for high garbage collection overhead. Thus, in order to reduce garbage collection overhead, we also have an effort to reduce the write number by increasing the hit ratio of a write buffer.

From the experiment for the execution time, we insist that the CAVE scheme can reduce the execution time as the overall performance by 55.5% - 97.4% in block-level LRU using SYSMARK compared to conventional method which uses a multi-channel FTL scheme by increasing I/O parallelism and reducing garbage collection overhead.

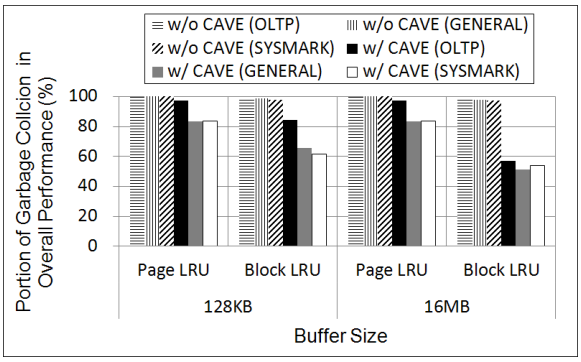


Figure 7: Portion of Garbage Collection Overhead in Overall Performance

6. CONCLUSIONS

We propose a new buffer management scheme, called Channel-Aware Victim Eviction (CAVE) which evicts multiple victims whose number is the same with the number of NAND flash memories. By using the CAVE scheme, we can increase I/O parallelism. We can use a 1-channel FTL scheme, thus reducing garbage collection overhead because a write buffer exploits a multi-channel architecture of SSD instead of a multi-channel FTL scheme. The experimental results show that the CAVE scheme can reduce it by 55.5% - 97.4% in block-level LRU using SYSMARK compared to the conventional method.

Even though the CAVE scheme can improve performance of SSD, there are still some overheads. First, we focus on exploiting full I/O parallelism, but it has a disadvantage in latency because we evict victims only when a write buffer is full. As the further work, we'll have an effort to control the eviction rate according to a buffer occupancy like a linear-threshold scheme. In the linear-threshold scheme, we have to find out an appropriate point between throughput and latency. Second, we'll consider an overhead of read operation. In order to overcome this problem, we adaptively regulate a level of the channel number by examining a relationship between I/O parallelism and garbage collection. Combining multi-channel FTL schemes which can adaptively control a level of the channel number with the CAVE scheme is one of the solutions for this method.

7. ACKNOWLEDGEMENTS

The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.

8. REFERENCES

[1] Datasheet: K9XXG08UXA (NAND Flash Memory), *http://www.samsung.com*.

[2] UMass TraceRepository, *http://traces.cs.umass.edu/index.php/Storage/Storage*.

[3] N. Agrawal, V.Prabhakaran, and T. Wobber. Design Tradeoffs for SSD Performance. In *proc. ATC'08*, pages 57–70, 2008.

[4] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini. STOW: A Spatially and Temporally Optimized Write Caching Algorithm. In *proc. ATC'09*, pages 327–340, 2009.

[5] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee. FAB: Flash-Aware Buffer Management Policy for Portable Media Players. *IEEE Transactions on Consumer Electronics*, 52(2):485–493, 2006.

[6] J. Kang, J. Kim, C. Park, H. Park, and J. Lee. A Multi-Channel Architecture for High-Performance NAND Flash-Based Storage System. *Journal of Systems Architecture*, 53(9):644–658, 2007.

[7] 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*, 58(6):744–758, 2009.

[8] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *proc. FAST'08*, pages 1–14, 2008.

[9] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho. A Space-Efficient Flash Translation Layer for Compactflash Systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, 2002.

[10] J. H. Kim, S. H. Jung, and Y. H. Song. Cost and Performance Analysis of NAND Mapping Algorithms in a Shared-bus Multi-chip Configuration. In *proc. IWSSPS'08*, 2008.

[11] 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. *ACM Transactions on Embedded Computing Systems*, 6(3):17–43, 2007.

[12] S. Lee, D. Shin, Y. Kim, and J. Kim. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems. In *proc. SPEED'08*, pages 36–42, 2008.

[13] Y. Lee, D. Jung, D. Kang, and J. Kim. u-FTL: A Memory-Efficient Flash Translation Layer Supporting Multiple Mapping Granularities. In *proc. EMSOFT'08*, pages 21–30, 2008.

[14] J. Park, G. Park, C. Weems, and S. Kim. Sub-grouped Superblock Management for High-performance Flash Storages. *IEICE Electronics Express*, 6(6):297–303, 2009.

[15] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee. CFLRU: A Replacement Algorithm for Flash Memory. In *proc. CASES'06*, pages 234–241, 2006.

[16] D. Seo and D. Shin. Recently-Evicted-First Buffer Replacement Policy for Flash Storage Devices. *IEEE Transactions on Consumer Electronics*, 54(3):1228–1235, 2008.

[17] J. Seol, H. Shim, J. Kim, and S. Maeng. A Buffer Replacement Algorithm Exploiting Multi-Chip Parallelism in Solid State Disks. In *proc. CASES'09*, pages 137–146, 2009.

[18] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J. Choi, S. Lee, Y. H. Bae, J. Lee, Y. Cho, and S. L. Min. Hydra: A Block-Mapped Parallel Flash Memory Solid-State Disk Architecture. *IEEE Transactions on Computers*, 59(7):905–921, 2010.