온라인 트랜잭션 프로세싱 서버에서의
효율적인 플래시캐쉬 관리 방법

Efficient Flash Cache Management

in Online Transaction Processing Server

2015 년

서강대학교 대학원

컴퓨터공학과

김 진 표

온라인 트랜잭션 프로세싱 서버에서의
효율적인 플래시캐쉬 관리 방법

Efficient Flash Cache Management

in Online Transaction Processing Server

지도교수 김 주 호

이 논문을 공학석사 학위논문으로 제출함

2015 년 12 월

서강대학교 대학원

컴퓨터공학과

김 진 표

논 문 인 준 서

김진표의 공학석사 학위논문을 인준함.

2015 년 12 월

주심 임 종 석 (인)

부심 김 주 호 (인)

부심 이 혁 준 (인)

감사의 글

지난 2년간의 대학원 생활은 저에게 다양한 경험을 할 수 있는 시간이었고, 그로 인해서 제가 많이 성장할 수 있었습니다. 먼저 연구를 지도해주시고 논문을 쓰는 과정에 많이 신경 써주시고 졸업 이후의 삶의 방향도 제시해 주신 김주호 지도 교수님께 진심으로 감사 드립니다. 또한 제가 학부과정과 석사과정으로 학교를 다니는 동안 큰 가르침을 주시고, 바쁘신 와중에도 저의 논문을 심사해 주신 임종석 교수님과 이혁준 교수님께 감사 드립니다. 그리고 연구실에 적응할 수 있도록 많이 신경써주시고, 저를 늘 좋게 봐주신 덕근이형, 정원이형, 명우형에게 감사드립니다. 또한 연구실 동기로 함께 2년간 함께 생활하며, 같이 웃고, 또 힘든 시간들을 버틸 수 있게 도와준 성일이, 혜민이, 선화에게도 고맙다는 말을 전하고 싶습니다. 또한 제가 논문 쓰는 동안 연구실 일들을 잘 감당해준 무준이형, 민균이, 싸드에게도 감사를 전합니다.

마지막으로 항상 저를 위해 기도해주시고, 격려와 지지를 보내주신 사랑하는 부모님과 동생 그리고 한나에게 깊은 감사의 말을 전합니다.

2015 년 12 월

김 진 표 올림

Table of Contents

List of Figures

List of Tables

초 록

호스트사이드 플래시 캐시는 하이엔드 엔터프라이즈 서버 시스템에서 높은 I/O 성능과 낮은 지연을 위하여 컴퓨터 구조에 나타난 새로운 계층이다. 플래시 메모리가 저장장치로 쓰이는 기존의 방법과 달리, 플래시 메모리가 캐시로서 사용되는 데 몇 가지 방법이 사용된다. 그러나, 쓰기가 집중적인 온라인 트랜잭션 프로세싱 (OLTP) 서버에서 기존의 플래시 캐시 시스템은 낮은 성능을 보인다. 따라서, 이 논문에서는 플래시 캐시에 쓰기 버퍼의 관리에 초점을 두었다. 솔리드 스테이트 캐시 (SSC) 기반의 효율적인 플래시 캐시 관리 시스템을 제안한다. 제안된 시스템은 쓰기 버퍼에서의 접근 횟수를 사용하는 캐시 입장 정책을 활용하고, 이에 따라 적절한 유형의 플래시 메모리 블락을 할당한다. 이렇게 함으로써, 쓸모 있는 데이터가 캐시내에 유지되고, 플래시 메모리에 불필요한 쓰기가 감소한다. 제안한 방법을 검증하기 위해, 리눅스 서버에 TPC-C 벤치 마크 [13] 를 설치했고, I/O 트레이스들을 추출했다. 또한 시뮬레이터를 구현하여 시뮬레이션을 수행했다. FlashTier [3]와 비교한 실험 결과는 제안한 방법이 hit 비율을 10%이상 향상시키고, 일반화된 평균 응답시간을 0.9에서 0.76만큼 감소시켰다.

키워드 - 쓰기 버퍼, 솔리드 스테이트 캐시 (SSC), 플래시 캐시

ABSTRACT

Host-side flash cache is a new tier in computer architecture that can archive high I/O performance and low latency in enterprise server systems. In contrast to conventional methods that use flash memory for storage, some approaches employ flash memory as cache. However, existing flash cache systems show low hit rate and high latency in the write-intensive online transaction processing (OLTP) server. The main idea of this thesis is on the utilization of the write buffer in the flash cache. In this thesis, an efficient flash cache management system based on solid state cache (SSC) is proposed. The proposed system utilizes an admission policy that employs the access count in the write buffer and allocates the appropriate type of block in flash memory. It is thereby possible to retain valuable data and reduce unnecessary writing in flash memory. To validate the proposed system, TPC-C benchmark [13] was installed on a Linux server and the I/O traces were extracted. In addition, SSC simulator was implemented and simulation was performed on it. Experimental results show that the proposed system improves the hit rate by up to 10% and reduces the normalized average response time from 0.9 to 0.76 compared with FlashTier [3].

Keywords – write buffer, Solid state cache (SSC), Flash cache

Chapter 1

Introduction

Flash cache is primarily used in networked storage environments. In fact, a new tier has emerged between the main memory (DRAM) and storage area network (SAN). Flash memory has faster I/O performance than magnetic disk (HDD) and it costs less than main memory (DRAM). Flash cache is used in hyper computing servers, cloud servers, and data centers that are commercially used in Facebook and Google. Naturally, there exist various access patterns of workloads depending on the usage environment. In the online transaction processing (OLTP) server, the ratio of small size random write requests is greater than the ratio in other servers. Accordingly, flash cache is added to improve the endurance and hit rate by effectively managing small-size write requests in the OLTP Server. Figure 1.1 shows that relation between hit rate of flash cache and I/O performance of system with networked storage. Koller et al. [1] mentioned that cache miss degrades throughput of flash device. Solid state drive (SSD) has similar architecture to the flash cache and it is used as a fast storage device.

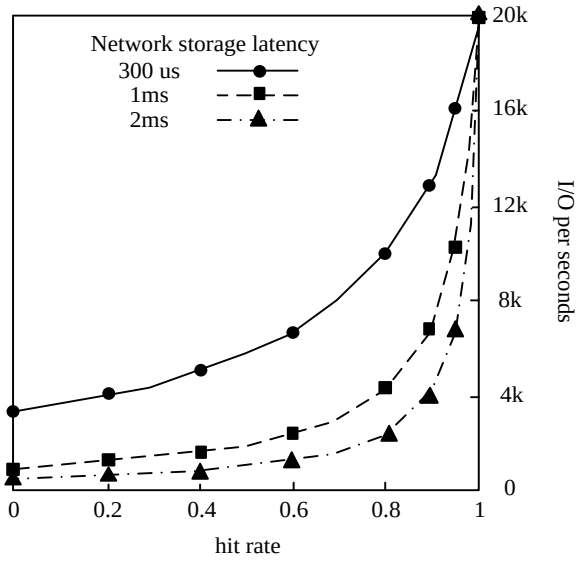

Figure 1.1. I/O performance impact of hit rate

Flash memory has two key characteristics. Firstly, it is characterized as 'erase-before-write' memory; i.e., it cannot overwrite without erasing the block containing the page. This feature of flash memory has recently drawn many researchers' attention on the out-of-place update and garbage collection (GC). Secondly, the flash memory cell has a limited number of erase counts. The maximum number of erases per block is 100,000 in single-level cell (SLC). It is even less in multi-level cell (MLC) (≤ 10,000) and in three-level cell (TLC) (≤ 3,000). In order to extend the lifespan of flash memory, many techniques have been developed and the wear-leveling is commonly accepted method among them.

Using flash memory as a cache has different behaviors that distinguish this application them from its conventional usage as storage. First, data in the cache may present elsewhere in the system. Thus, it has some

flexibility in terms of how it manages data. Second, a cache requires consistency. A cache must ensure that it never returns stale data; however, it can also return nothing if no data is present. Thus, writing data of logical block address (LBA) that have a few read requests and many write requests wastes space because the data have already been written somewhere. On the other hands, there are also LBAs that have a high potential of being read. This results in a lack of space for the latter data, as well as the generation of more unnecessary invalid pages by submitting continual write requests and invoking frequent garbage collection.

In recent years, there has been tremendous interest in exploring the flash cache system. Mercury [2] proposed host-side flash caching that provides a cache for various virtual machines over networked storage protocols. Their approach focuses on flash caching in a virtualized, shared storage data center. FlashTier [3] is a flash cache layer based on block-level caching. It uses a custom flash translation layer (FTL) that is optimized for caching. In particular, FlashTier decreases the amount of writing by replacing the native garbage collection with silent eviction. Additionally, HEC [4] adopted flash-layer write amplification (FLWA) and cache-layer write amplification (CLWA) as measurement units and experiments for endurance improvement. HEC strived to reduce the write amplification results from GC, cache admission policies, and the cache eviction policies.

The above methods provide block-level caching. In fact, they manage coarse-grained cached data. Block-level caching can write in block units when a cache miss occurs. It means that only one cache miss can load a whole block of many data that may not be used later. If only one page of a

block has a read request, and the other pages have no requests, the spaces occupied by the pages with no requests will be wasted. Consequently, it leads into a lack of space and calls garbage collection. Therefore, a new approach that manages fine-grained data in a page-level caching is proposed. Since most request size of write-intensive OLTP server workloads (TPC-C) [13] is 8KB, the proposed method writes in the size of corresponding request units. Furthermore, the proposed method is efficient in terms of space utilization in flash memory block.

In this thesis, an efficient write buffer management using the flash cache was proposed. In addition, a method of managing block in flash cache to retain data that have a high potential of reading was presented. In short, the main contributions of this thesis are outlined below.

- The write buffer was used for analyzing read/write counts for the LBA and mitigate pressure due to repetitive write requests in the workload.
- The proposed method manages flash memory by classifying blocks into two types for effective eviction. In this method, data that have a high possibility of reading are preserved in the flash cache.
- The proposed method utilizes capacity for retaining more data by page-level caching. As a result, it produces efficient garbage collection and improvement in high hit rates.

The remainder of this thesis is organized as follows. Chapter 2 describes the background and motivation of this research. Chapter 3 describes the proposed method. Chapter 4 describes experimental results. Finally, chapter 5 presents conclusions.

Chapter 2

Background

2.1 Host-side Flash Cache

Host-side flash cache resides in layers between SAN and DRAM. Deployment of host-side flash cache can improve I/O performance in enterprise server. In fact, flash cache fills the gap between HDD and DRAM with high I/O performance and low cost. In typical enterprise systems, the host server employs solid state drive (SSD) directly attached to cache data at the SAN backend. In other words, host-side flash cache removes confusion in the SAN backend and produces overall performance improvement by removing network latency. Inherent differences exist between the flash-based cache and DRAM-based cache. Firstly, the flash caches is located underneath the DRAM in the storage hierarchy. Therefore, it stores data that are less referenced. Secondly, flash cache is used instead of DRAM as the caching media. Although flash cache has a finite number of program/erase (P/E) cycles, DRAM-based cache does not have the same write cycle limitation. Thus, the write cycle limitation is relevant to the device lifetime.

The proposed method of flash cache has two main advantages. Because SAN requires immense access latency, data are returned when possible in flash cache or DRAM for obtaining high I/O performance of the application. Thus, the first advantage comes from high hit rate by retaining data that have a high potential of read in near future. The second advantage comes from reducing garbage collection that affects I/O performance. In general, flash cache predominantly runs at full capacity to achieve high hit rate. Naturally, it requires garbage collection for reallocating space; moreover, write amplification increases on account of valid page copy during garbage collection. Furthermore, the garbage collection overhead further degrades I/O performance as more valid page copying occurs. Therefore, the proposed method considers retaining data that have a high potential of read and reducing garbage collection.

2.2 Access Patterns in the TPC-C Benchmark

The TPC-C benchmark is the OLTP benchmark. It is more complex than the conventional OLTP benchmark because it is comprised of multiple transactions and more complex database. Its database consists of nine table types. TPC-C benchmark simulates a complete computing environment in which a group of users executes transactions against the database simultaneously. The workload involves a combination of five concurrent transactions of different types. This workload contains a large number of read/write requests and it calls small-size I/O operations (8 KB) that randomly access the data spread over a wide portion of the disk.

Table 2.1. TPC-C Benchmark

Size	Read	Write	Both types
8KB	65.70%	32.66%	98.36%
16KB	0.00%	1.49%	1.49%
Other sizes	0.00%	0.15%	0.15%
All sizes	65.71%	34.29%	100.00%

The ratio of read and write at various request size is described in Table 2.1. As shown in Table 2.1, the ratio between the read and write requests of the workload-extracted TPC-C benchmark is 1.9:1. As shown in Table 2.1, TPC-C benchmark has a property of having more number of small size write access than others.

2.3 FTL and Write Amplification

Over several decades, considerable research on FTL has been conducted for obtaining high performance. For hybrid mapping, various FTLs exist, such as FAST [5], BAST [6], SAST [7], and LAST [8]. For page-level mapping, DFTL [9] was proposed to cache the frequently used mapping table in the on-disk SRAM as a means of improving the address translation performance. In addition, μ-FTL [10] adopted the μ-tree on the mapping table to reduce the memory footprint.

FTL performs out-of-place updates, thereby it leaves invalid pages. When SSD has no free block, FTL operates garbage collection for setting aside free spaces. In normal operation, the garbage collection erases the blocks only after valid pages in the blocks are copied to another blocks. This

results in additional writes because the block of valid pages is copied.

In HEC, CLWA and FLWA are proposed measure of write amplification in the flash cache. First, CLWA is computed as the ratio between the cache-generated writes and the original workload writes. In other words, it measures the amount of write amplification on cache misses. CLWA can be significantly reduced by cache admission policies. Second, FLWA is the ratio between the GC-generated writes and the original workload writes. It can be affected by access pattern of workloads and GC policies.

2.4 Motivation

Recently, flash memory is often used as cache between main memory and storage. The flash cache can reduce tremendous amount of storage access with proper memory management. However, the native flash cache in write-intensive OLTP server may suffer from significant performance losses. Existing methods adopt an admission policy such as touch count and selective sequential rejection (SSEQR). Touch count is not effective in OLTP server because it allocates space for data only after a cache miss occurs several times. SSEQR does not admit a long sequential request. It can lead to an immense number of cache misses because unpermitted large-size request may contain hot data.

The existing methods utilize the access count or size of request for cache admission policy. However, those methods are inappropriate to determine which data is copied into the flash cache especially with write-intensive workload. The data that have a high potential of write and few potential of

read should not occupy the flash cache space. Even though these data occupy flash cache, this method provides no benefits in system. Rather than generating more invalid page, more garbage collection is required. In write-through policy, the present of data in flash cache does not affect the total time from write request to the completion because of writes in networked storage. Thus, the admission policy of the proposed method does not admit data that have a high write count in write buffer.

Garbage collection is another factor of serious performance loss in the flash cache. When capacity of the flash cache is full, it must clean invalid pages and erase the block or destage some blocks to set aside space through garbage collection. One of the commercially available FTL is the FAST [5] and it performs garbage collection with the sequential and random write log blocks. FlashTier proposed the silent eviction replacing garbage collection. Silent eviction selects the victim block with the least number of valid pages and then the chosen victim block is erased without copying the valid pages in the victim block. Accordingly, the method can reduce the time of setting aside free space. Nevertheless, the silent eviction causes frequent cache miss because it does not preserving valid pages that might be read soon. Naturally, increasing the cache miss results in increasing access on SAN. This causes declining IOPS (input/output operations per second) and significantly high response time. It motivated this thesis towards applying proper modification to the exiting silent eviction method.

Chapter 3

Flash cache management in OLTP server

A new flash cache management is described in this chapter. Figure 3.1 shows enterprise server system with the proposed flash cache. In Figure 3.1, the flash cache is located between main memory and networked storage in server system architecture. A flash cache system includes write buffer, cache controller, flash controller, and NAND flash device. The cache controller controls the cached data with admission and eviction policies. The flash controller not only manages NAND flash device but also performs GC algorithm. The operating system directly maps the flash cache page, and writes to the flash cache using the logical block address and offset. The flash cache internally maps those addresses to physical locations. The following six steps describes the life cycle of data in the proposed flash cache.

1. Write buffer gets allocated only if the read/write request trigger cache misses.
2. When the data in the write buffer are destaged, the data could be written in the flash cache block by the admission policy.

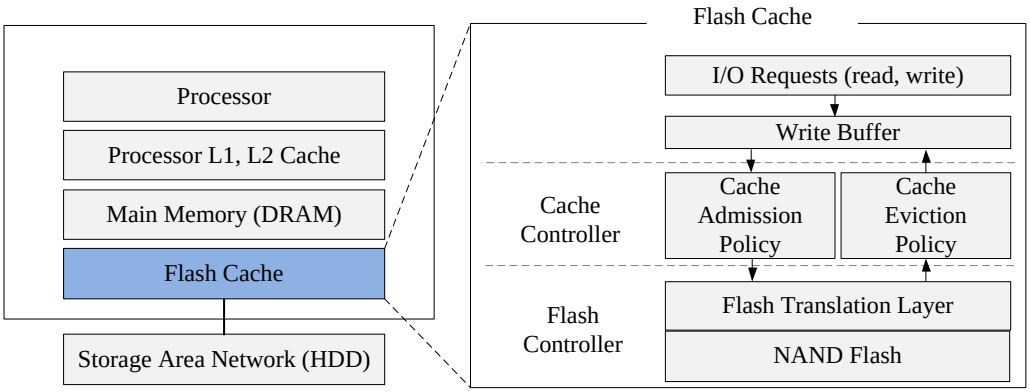

Figure 3.1. Flash cache architecture

3. Corresponding blocks are allocated depending on the type of data which is either write-frequent data (WFD) or read-frequent data (RFD),
4. The flash controller allocates pages in the block upon the size of the request.
5. The flash controller notifies the cache controller of full in NAND flash device, and then cache eviction policy chooses the victim blocks.
6. The cache controller performs efficient eviction with the write buffer if the victim block is read-frequent data block (RFDB), and then erases the victim blocks.

The admission policy

```
Input:  node pointer of each LBA in write buffer
If node-> write_count > Threshold_write && node -> read_count == 0 then
    invlalid_data(lba);
else if node->read_count > Threshold_read || node->write_count < Threshold_write then
    ppa=allocate_RFDB();
else
    ppa=allocate_WFDB();
end
flash_write(ppa, node->data, node->lba);
```

Figure 3.2. The admission policy

3.1 Admission Policy with read/write count in the Write Buffer

The cache admission policy admits data except that of frequent writes. In order to identify data of high potential writes, the flash cache records read/write count of each LBA in write buffer. The flash cache manages a list stored read/write count of each LBA in the write buffer. The list gets updated when the read/write request corresponding with the LBA are submitted. When the LBA are destaged on the write buffer, the admission policy admits the data of the LBA by checking the list. In addition, the least recently written (LRW) policy in write buffer is applied to retain WFD for a long time. Figure 3.2 shows the detailed admission policy. First of all, if the read count of the LBA is zero and the write count of the LBA is larger than the fixed value, the data of LBA does not write to flash cache block. Next, the admission policy classifies the LBA that could be written in flash cache

block into two types. The LBA is classified as RFD if the read count of the LBA is larger than the fixed value, or the write count of LBA is smaller than the fixed value. Lastly, the remaining data is considered as the WFD and the admission policy allocates write-frequent data block (WFDB).

3.2 Eviction Policy

The eviction policy performs eviction in different ways depending on type of blocks. The flash cache blocks are manages by splitting them into RFDBs and WFDBs. In Figure 3.3, evictions of two block type is described and two type of eviction is different. In order to retain data of high potential read, the eviction policy evicts WFDB first. In WFDB, the eviction policy selects the victim block with the largest number of invalid pages like silent eviction and then the victim block is erased without copying the valid page. When the eviction policy continuously evicts only WFDB blocks, there would not be any WFDBs and then only RFDBs remain. For this reason, the eviction policy maintains the number of WFDB that represents 10% of the whole block in the flash cache. Accordingly, the eviction policy performs eviction in RFDB if necessary. The efficient eviction chooses the victim block with the largest number of invalid pages in RFDBs. As shown in 3.3, valid pages contained in the victim block are written again in the write buffer and then, the victim block gets erased. After some time, the admission policy classifies the uploaded data into unpermitted data, or RFD or, WFD.

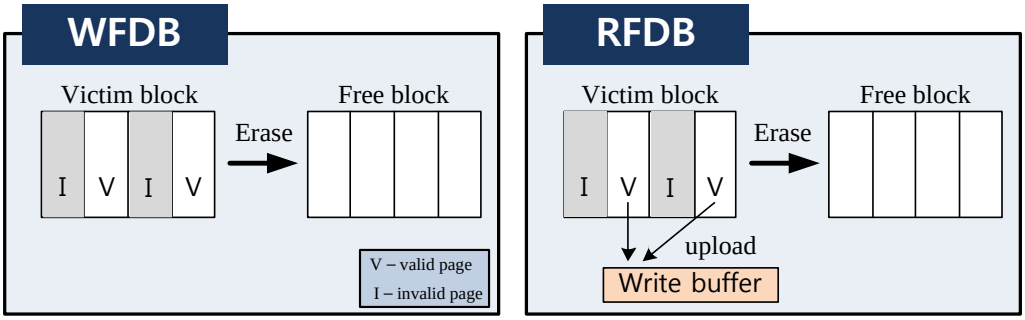

Figure 3.3. Method of eviction in WFDB and RFDB

3.3 Flash memory block management

The admission policy classifies destaging data into two types and notifies flash controller of the type of data, RFD and WFD. Figure 3.4 shows that the proposed method divides the flash memory block into two type of blocks, read-frequent and write-frequent data blocks. In RFDB, the flash controller uses hybrid mapping method with log block, and it is similar to the method of FAST. The flash controller uses log blocks for out-of-place update and performs log block merge in RFDB when the number of log block reaches up to its limit. On the other hand, the flash controller employs page-level mapping without log block in WFDB. The flash controller just retains lots of write-frequent data temporarily to prevent cache miss. Thus, there are no log block merge operations because of low importance. If the write request corresponding with the LBA that stored in WFDB are submitted, it would be written in the write buffer and flash controller would mark the page that are stored the data in WFDB as invalid.

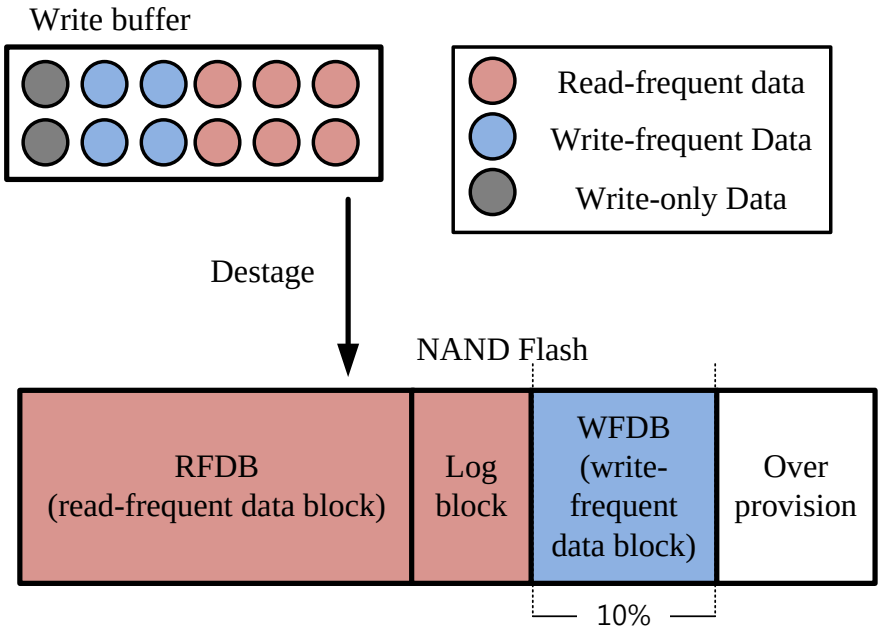

Figure 3.4. Management in flash memory block

3.4 Page level caching

Page level caching is applied to the proposed method. Figure 3.5 and 3.6 show block level caching and page level caching in server system. As shown in Figure 3.5, flash cache allocates space by the block. In contrary to block level caching, it can allocate flash memory space in request size unit. Page level caching can handle a lot of cached data.

In addition, page level caching is different from page level mapping. In other words, unit of caching method would be page if mapping method is block mapping. In this thesis, mapping table uses hah map. If write buffer

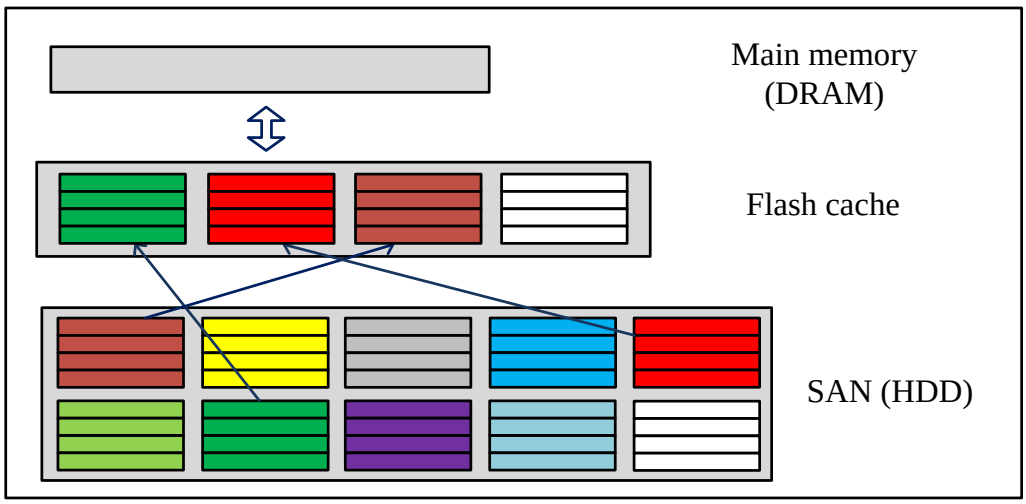

Figure 3.5. Block level caching

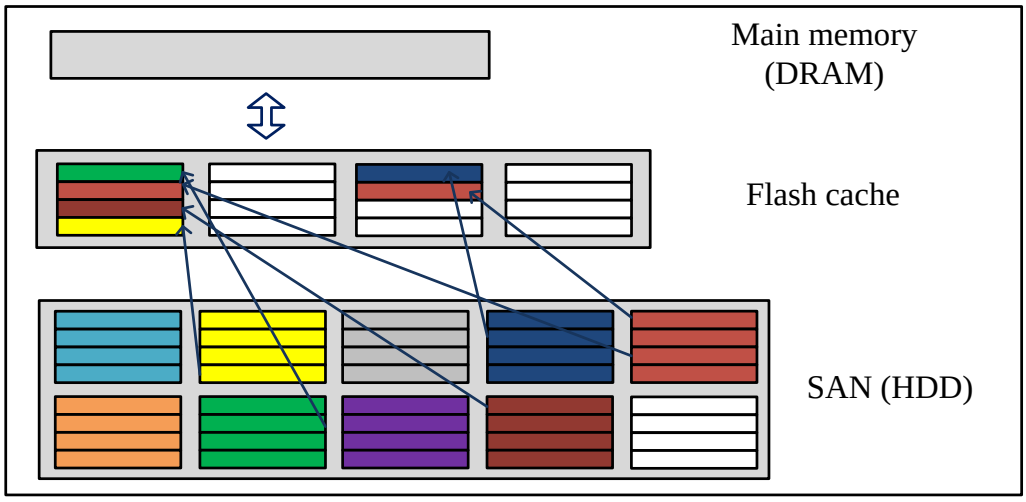

Figure 3.6. Page level caching

is full, flash cache destages data in write buffer and allocates a appreciate space and then each combination of physical block address and offset is independently mapped with combination of local block address and offset. Accordingly, mapping method of RFDB can be hybrid mapping.

Chapter 4

Experimental Results

4.1 Simulation Environments

The proposed method is evaluated by the simulator that is based on the solid state cache (SSC) simulator in FlashSim [11]. The simulator composes of a write buffer, a cache controller, and a flash controller. This simulator includes write buffer management schemes, admission policies, and eviction policies. An SSC simulator was emulated with the parameters [14] in Table 4.1.

4.1.1 Trace of TPC-C benchmark

TPC-C benchmark was installed on a Linux server. An 8 GB database of TPC-C benchmark was generated through Mysql, which uses InnoDB as a storage engine and default page size is 16 KB. An I/O trace of the TPC-C benchmark was extracted in four hours run with the configuration.

Table 4.1. Emulation parameter

Page read	65	Control delay	10
Page write	85	Page size	4096 bytes
Block erase	1000	Page / erase block	64
Seq. Read	585 MB/sec	Rand. Read	149,700 IOPS
Seq. Write	124 MB/sec	Rand. Write	15,300 IOPS

In addition, an I/O trace of the TPC-C benchmark was extracted by compressing the page size to 8 KB since most of read request size in the TPC-C benchmark is 8 KB.

4.1.2 Cache Controller and Flash Controller in simulator

This program simulates the behavior of cache controller depending on the write buffer scheme, the cache mode, the admission policy, and the eviction policy. The cache controller in the simulator mainly operates two functions. Firstly, it manages cached data with mapping table in memory. Secondly, it determines which data gets admitted or evicted. Moreover, the cache controller provides only a write-through policy. For that reason, dirty page information is not stored; it only writes the data. For comparison purpose, the simulator supports least recently used (LRU) policy of write buffer, various cache method, and eviction.

The flash controller simulates a NAND flash device that is similar to a SSD. It consists of a FTL, GC, and other capabilities to manage flash device. In order to reduce response time, the flash controller stores the maps of

valid, invalid, and erase counts in memory. In addition, the flash controller supports various GC method and mapping method. The flash controller reserves 10% of total capacity for the overprovisioned blocks and maintains 7% of their capacity for log blocks.

4.1.3 Memory Consumption

The flash cache stores the mapping table from the LBA to the physical address of the flash cache using a hash map. The hash map is similar to that of FlashTier; however, the values of key and physical address are different. The hash map has a higher speed and lower space overhead than the hash table of flashcache [12]. The flash cache commercially maintains the whole mapping table in its own memory (DRAM). The proposed system consumes approximately 8 bytes per key of cached page. Thus, it requires more memory than the method of block-level mapping.

In this simulation, the 4 GB flash cache required 1,048,576 entries. Because the size of each entry is 8 bytes, the total mapping table requires 7.5 MB. If the flash cache size is 512 GB, it would require 960 MB. The address map of native method [2] is implemented using an array of 4 byte entries. This results in a 768 MB address map in 512 GB cache device. A tradeoff exists between the hit rate and memory consumption in enterprise systems. Additionally, the proposed system manages the list of data stored in the write buffer. This list requires 16 KB when the write buffer size is 8 MB because the write buffer has 2,048 entries and each entry requires 8 bytes.

4.2 Simulation Analysis

The merits and overheads of the proposed method are compared to FlashTier bearing three key questions in mind;

1) How well does the proposed method retain read-frequent data?
2) How accurately does the proposed method evict unnecessary data?
3) To what extent does the proposed method reduce GC count?

To verify the effectiveness of the proposed method, trace-driven simulations were conducted. In this experiments, each flash cache block consists of 64 pages by default and the write buffer size ranges from 8M to 16M bytes. Simulation were conducted on the TPC-C benchmark suit in the following four cases.

1) write buffer size – 8 MB, database page size of workload – 16 KB
2) write buffer size – 16 MB, database page size of workload – 16 KB
3) write buffer size – 8 MB, database page size of workload – 8 KB
4) write buffer size – 16 MB, database page size of workload – 8 KB

The simulation results of flash cache on TPC-benchmark (16 KB and 8 KB) for FlashTier and the proposed method are in Table 5.1 and 5.2. The hit rate, GC count, CLWA, FLWA, total write amplification factor (TWAF), and normalized average response time are contained in Table 5.1 and Table 5.2.

The simulation results of hit rate in both FlashTier and the proposed method are illustrated in Figure 5.1. The hit rate of the proposed method outperforms FlashTier by up to 10%. The proposed method achieves high hit rate (up to 90%) in simulation results with the trace of compressed page size. Moreover, the hit rate of the proposed method increases up to 96% in the case of the 16 MB write buffer. The proposed method achieves low TWAFs in all of cases. In 16 KB size of trace, the proposed method reduces TWAF from 33.24 to 27.88. In addition, TWAF decreased from 15.65 to 11.83 in 8 KB size of traces. Moreover, CLWA and FLWA were compared. The proposed method significantly reduces CLWA. On the other hands, FLWA of the proposed method larger than that of FlahTier. The proposed method decreases GC count. In 16 KB traces, GC count of the proposed method is reduced by up to 51%. Additionally, GC count of the proposed method is reduced by 40% in trace with compressed page size. Comparison results of normalized average response time in both FlashTier and the proposed method are described in Figure 5.2. The normalized average response time of the proposed method is improved from 0.9 to 0.76 in Figure 5.2(a).

Table 5.1. Simulation results of TPC-C benchmark (16 KB)

Flash Cache	Write Buffer Size	Hit rate	GC count	CLWA	FLWA	TWAF	Normalized average response time
FlashTier	0 MB	85.0 %	199,975	35.00	13.33	48.34	1.0
	8 MB	84.7 %	89,225	28.91	5.42	34.34	0.904
	16 MB	84.93 %	87,750	28.53	4.70	33.24	0.896
Proposed Method	8 MB	94.6 %	42,587	25.46	2.55	28.01	0.765
	16 MB	94.58 %	42,125	25.34	2.53	27.88	0.764

Table 5.2. Simulation results of TPC-C benchmark (8 KB)

Flash Cache	Write Buffer Size	Hit rate	GC count	CLWA	FLWA	TWAF	Normalized average response time
FlashTier	0 MB	81.10%	316,888	4.94	24.34	29.29	1.0
	8 MB	85.79%	113,240	15.65	0.0003	15.65	0.84
	16 MB	85.69%	116,833	15.76	0.0003	15.76	0.84
Proposed Method	8 MB	91.25%	77,677	2.16	9.67	11.83	0.80
	16 MB	96.27%	70,725	2.16	10.55	12.72	0.77

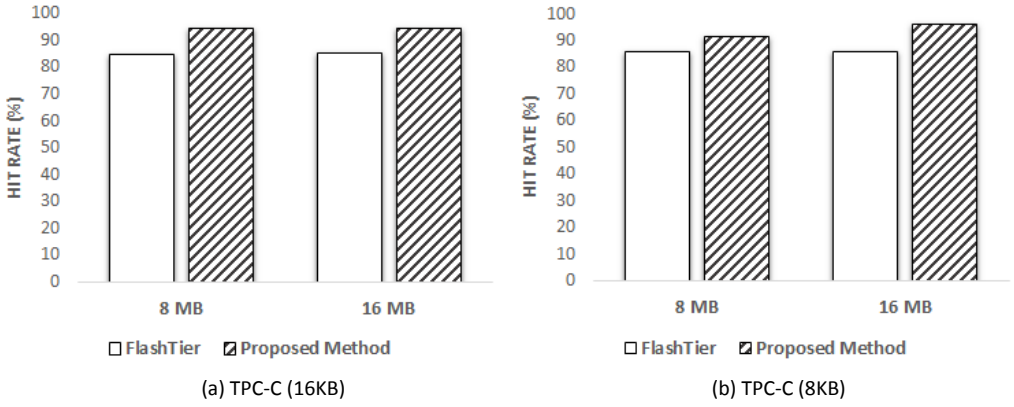

Figure 5.1. Hit rate

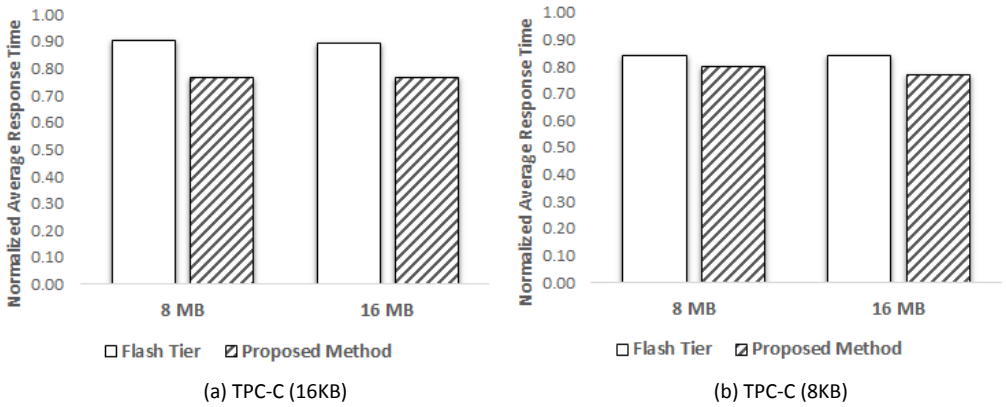

Figure 5.2. Normalized average response time

There are two major reasons for the improvement. Firstly, benefits of the proposed method come from identifying the read-frequent data. And then, the proposed method manages flash cache by classifying block into WFDB and RFDB. Thus, retaining data that are allocated in RFDB reduces cache misses. Accordingly, hit rate increases and a substantial amount of writes decreases. Secondly, the proposed method allocates space in accordance with request size. Therefore, write amplification factor and GC count decrease by utilizing the overall flash cache block. On the other hands, FlashTier uses block-level caching and evicts the cached data too early, as shown in the FLWA of FlashTier. The overhead of the proposed method includes the computation time of the admission policy and recording read/write count of write buffer. However, they are negligible.

Chapter 5

Conclusions

Existing flash caches are widely employed in enterprise server system for achieving high I/O performance. However, traditional flash caches incur performance degradation due to small-sized read/write requests. A new method of flash cache management improves overall performance especially in write-intensive OLTP server.

The key insight is that flash cache has some flexibility in terms of how it manages data. Contrary to conventional usage as storage, it does not require to preserve the data of frequent write. Thus, the proposed method classifies incoming data by using the read/write count during the write buffer, and it then writes on affordable space in flash cache block.

In this thesis, SSC simulator was implemented to validate the proposed method and it performed simulations several times. The experimental results showed a comparison of FlashTier and the proposed method. The proposed method improves the hit rate by 10% for flash cache. In addition, it obtains half of the GC count and reduces a substantial amount of writing on flash memory block. Lastly, it reduces normalized average response time from 0.9 to 0.76.

REFERENCE

[1] R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala and M. Zhao, "Write policies for host-side flash caches", in *Proc. 11th USENIX Conf. File Storage Technol*., pp.45 -58 2013

[2] S. Byan, J. Lentini, A. Madan, L. Pabon, M. Condict, J. Kimmel, S. Kleiman, C. Small, and M. Storer, "Mercury: Host-side flash caching for the data center," in *Proc. 28th IEEE MSST*, pp. 1–12, 2012.

[3] Saxena, Mohit, Michael M. Swift, and Yiying Zhang, "Flashtier: a lightweight, consistent and durable storage cache," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012.

[4] J. Yang, N. Plasson, G. Gillis, N. Talagala, S. Sundararaman, and R. Wood, "HEC: improving endurance of high performance flash-based cache devices," in *International Systems and Storage Conference*, p. 10, 2013.

[5] S.W. Lee, D.J. Park, T.S. Chung, D.H. Lee, S. Park and H.J. Song "A Log Buffer Based Flash Translation Layer Using Fully Associative Sector Translation", ACM Trans. Embedded Computing Systems, vol. 6, no. 3, 2007.

[6] J. Kim, J.M. Kim, S.H. Noh, S.L. Min and Y. Cho "A Space-Efficient Flash Translation Layer for Compact Flash Systems", IEEE Trans. Consumer Electronics, vol. 48, no. 2, pp.366 -375, 2002.

[7] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho and J. Kim, "A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND

Flash-Based Applications," ACM Trans. Embedded Computing Systems, vol. 7, no. 4, pp. 38:1-38:23, 2008.

[8] S. Lee, D. Shin, Y. Kim and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," ACM SIGOPS Operating Systems Rev., vol. 42, no. 6, pp. 36-42, 2008.

[9] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings,” In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, pages 229-240, March 2009.

[10] Y.G. Lee, D. Jung, D. Kang, and J.S. Kim, “µ-ftl:: a memory-efficient flash translation layer supporting multiple mapping granularities,” In *EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software*, pages 21-30, 2008.

[11] Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. “FlashSim: A simulator for NAND Flash-based solid-state drives,” In *Proceedings of the 2009 First International Conference on Advances in System Simulation*, SIMUL '09, pages 125-131, 2009.

[12] T. Kgil and T. Mudge "FlashCache: A NAND flash memory file cache for low power web servers", *Proc. of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES \'06)*, 2006

[13] TPC-C. http://www.tpc.org/tpcc. July of 1992.

[14] Intel Corp, Intel smart response technology, http://download.intel.com/design/flash/nand/325554.pdf, 2011.