JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.17, NO.1, FEBRUARY, 2017 ISSN(Print) 1598-1657
https://doi.org/10.5573/JSTS.2017.17.1.015 ISSN(Online) 2233-4866

Divided Disk Cache and SSD FTL for Improving Performance in Storage

Jung Kyu Park, Jun-yong Lee, and Sam H. Noh*

***Abstract*—Although there are many efficient techniques to minimize the speed gap between processor and the memory, it remains a bottleneck for various commercial implementations. Since secondary memory technologies are much slower than main memory, it is challenging to match memory speed to the processor. Usually, hard disk drives include semiconductor caches to improve their performance. A hit in the disk cache eliminates the mechanical seek time and rotational latency. To further improve performance a divided disk cache, subdivided between metadata and data, has been proposed previously. We propose a new algorithm to apply the SSD that is flash memory-based solid state drive by applying FTL. First, this paper evaluates the performance of such a disk cache via simulations using DiskSim. Then, we perform an experiment to evaluate the performance of the proposed algorithm.**

***Index Terms*—Cache, hard disk, SSD, simulation, DiskSim**

I. INTRODUCTION

As computer processor speeds continue to increase, the challenge to ensure timely data supply also continues to increase. Processor speed increases at approximately 60% every year, whereas memory speed is growing at approximately 10% [9]. One major bottleneck in meeting the increased processor demand is the speed data and instructions are supplied from the storage devices. Although high speed cache systems and various other techniques have improved to fill the performance gap, secondary storage access speed remains a concern, especially for big data.

Access times for secondary devices are significantly higher than main memory, as shown in Table 1. Hard disk access is 10^5 slower than the main memory. As the gap increases toward 6 orders of magnitude, further optimization techniques are required. Even a small disk cache improvement could significantly reduce response time [11], substantially improving overall response time. Hospodor provides the total access time when the required data is not present in the disk cache and the physical disk must be accessed [6].

For simplicity, we consider cache transfer time to be equal that for a magnetic disk, although the latter is much higher as it involves rotating the physical disk to read from the sectors [7]. Access time is reduced by a factor of 100 when there is a cache hit. Many mechanical techniques have improved actual response over the last decade via reducing seek time by approximately 8% and increasing rotational speed by approximately 9% annually. Data area density improvements by 40% annually have also assisted, reducing response time by 8% every year. Overall, there has been a 15% yearly improvement in seek times from improved disk technology [1, 4].

The disk cache is a buffer in the disk system that holds recently accessed portions of disk memory. The file system/database cache represents the logical cache and the disk cache represents the physical cache. When the processor makes a request to the disk drive, the OS first checks the logical cache. A miss at the logical level

Manuscript received Oct. 16, 2016; accepted Feb. 3, 2017
Sam H. Noh is with the school of Electrical and Computer Engineering, UNIST
E-mail : samhnoh@unist.ac.kr

cache results in an I/O request to the physical cache. If a miss occurs at the physical level cache, then the physical drive is accessed.

This paper focuses on the physical cache [1], i.e., the disk cache of the disk drive system. Every disk cache hit results in substantially reduced (1–4 ms) I/O than would be required to access the disk itself (10–100 ms) [16]. We evaluate the performance of a disk cache divided into data and metadata regions, where metadata represents only a small portion of the memory, but is accessed very frequently [2, 5, 6].

Section 2 presents related works. Section 3 explains split percentage design and Section 4 discusses the implementation and evaluation methodology. Section 5 presents the simulation results. Section 6 concludes the paper and discusses future work.

II. RELATED WORK

We assess related research in the area of disk cache systems, and since metadata is a key aspect of the proposed system, we also discuss some optimizations in the field of metadata access.

1. Optimization Techniques for Hard Disk Cache

Yang and Hu proposed a novel disk storage technique called disk caching disk (DCD) to improve disk cache I/O performance [18]. They used a small log disk, or cache-disk, as a secondary disk cache to optimize write performance. This exploited access speed differences between the normal and cache disks. The latter has faster access even though both have the same physical characteristics because of the different data units used and differences in the way the data was accessed. Data transfer rate in units of tracks is almost eight times faster than in unit of blocks. Therefore, the log buffer was used as an extension to the RAM buffer to cache file changes and destage this to the data disk when the system was idle. All the small and random writes were first buffered into the RAM cache, and then written in a single data transfer to the cache disk when it was idle. Hence, the RAM buffer was cleared for more data transfer. When the disk was idle, the destage operation between the cache to disk and data disk was performed. Experiment tests with three traces (hplajw, cello, and snake), showed the DCD technique could improve write performance at the secondary storage level by one to two orders of magnitude. This technique involves the use of an additional hardware element to obtain the performance improvement.

2. Optimization Techniques for Metadata Access

Pen Gu et al. argued that existing data prefetching algorithms did not consider group prefetching and have higher computational complexity [5]. These techniques do not work with metadata access. Hence, they proposed an accurate and distributed metadata oriented prefetching algorithm. They proposed a weighted graph based technique for prefetching. Experiments showed the proposed algorithm provided considerable improvements for metadata access on the client side, reducing response time by 67% on average compared to LRU and other prefetching algorithms.

Hong evaluated the performance impact from using micro electromechanical systems (MEMS) as metadata storage and disk cache [6]. MEMS have seek-times 10–20 times faster than hard drives, storage density 10 times higher, and also lower power consumption. Simulations for MEMS used as dedicated metadata storage show a potential improvement of 28–46% in system performance for user workload, depending on how much metadata traffic is incorporated in the workload. He also discussed how using MEMS as a disk write buffer could improve system performance by a factor of 3.3–8.2, and provide better consistency on system performance than a disk system by a factor of 2.4 to 5.7.

Scott et al. discussed the importance of efficient metadata management in large distributed storage systems [2]. They argued that subtree partitioning and pure hashing were common techniques for managing metadata in such systems, but have a bottleneck of high concurrent access rates. They proposed a Lazy Hybrid (LH) technique for metadata management that combined the advantages of the two approaches, while avoiding the disadvantages.

One common conclusion from these researchers was that metadata is relatively small, but accessed very frequently. This forms the basis for our design.

3. Divided Disk Cache into Data and Metadata

Baskiyar et al. discussed split cache, or divided disk cache (DDC), architecture, where the data region of the cache is split into data and metadata [1]. In this theoretical paper, DDC was analyzed with the aim to improve the read miss ratio. They claimed the technique would reduce interference between data and metadata, and the effective read miss ratio could be improved by 20%, which would improve response time by 16%. The major points of their paper were:

Metadata accounts for only a small portion of the data but is accessed very frequently.

They used the Linux EXT2 file system to calculate metadata size, and provided data and metadata relationships.

Metadata account for 4.66% of an 8 kB file and 3.156% of a 12 kB file.

4. SSD

Flash memory-based solid state drive (SSD) has excellent I/O performance with low power consumption. Due to the limitation of NAND flash, the Flash Translation Layer (FTL) is implemented in the SSD controller to emulate block device such as hard drive. The FTL performs a garbage collection (GC) process to reclaim free space [19]. Servers use SSD as a storage device to store persistent data in big data environment. However, many devices in big data and Internet of Things (IoT) environment send a data frequently to server and makes a lot of small writes on the SSD. As a result, GC process performs internal data movement between the NAND flash block so that I/O performance is decreased [18].

For this reason, in this paper, we propose new FTL scheme that is based on page mapping FTL. Our experimental results show that our proposed method has 11% better results than the conventional page mapping FTL.

III. OPTIMUM DIVIDED CACHE DESIGN

We consider various research techniques in the area of disk cache. Also, since metadata and its access patterns are important for our research, we discuss optimization of metadata access.

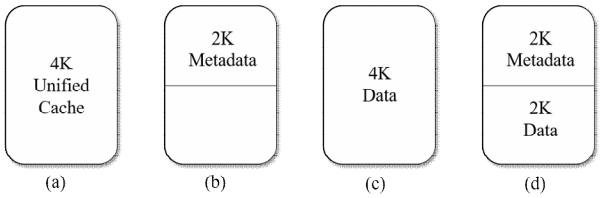


Fig. 1. Miss rates and composite derivatives for two processes with conventional cache.

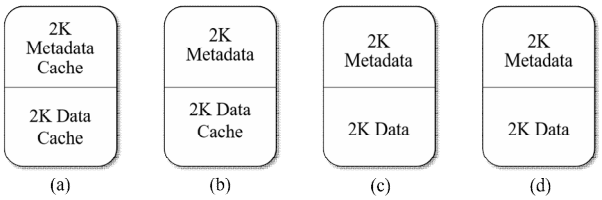


Fig. 2. Miss rates and composite derivatives for two processes with divided cache.

The divided cache design used here is similar to [1]. From Hsu and Smith [9, 10], the cache system becomes effective when the cache size reaches 1% of the external storage space. For example, for a 1 TB hard disk, the disk cache should be 5 GB when the disk is half full.

However, it is generally impractical to provide a cache of this magnitude. Therefore, we split the cache into data and metadata regions, so that metadata cache amounts to at least 1% of the metadata in the disk drive. Consider the case where we have a cache of 4 kB, as shown in Fig. 1(a). The disk is accessed twice to retrieve a 4 kB file and 2 kB metadata file. Metadata is first retrieved from the disk and stored in the cache Fig. 1(b), and then the data is retrieved from the disk and brought into the data cache. Since the data is 4 kB, it replaces the metadata in the cache Fig. 1(c).

When the same file is requested again, the metadata for the file is not present in the cache, and hence there is a cache miss. Metadata is now retrieved from the disk and stored in the metadata cache, replacing 2 kB data Fig. 1(d). The cache now contains 2 kB data and 2 kB metadata, resulting in a hit for half the data.

However, If the cache is divided into data and metadata regions as shown in Fig. 2(a), each of size 2 kB, then the metadata does not get a miss in the second request. This reduces interference between data and metadata and increases the overall hit ratio.

Some space is removed from the data cache region, so the number of data hits may reduce. However, since

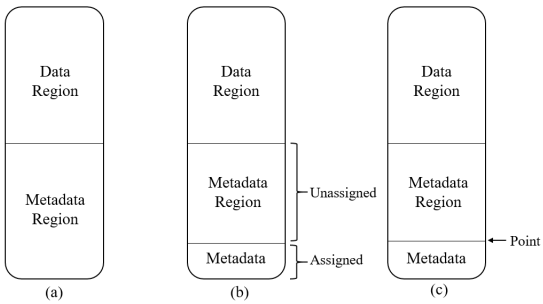


Fig. 3. Data and metadata regions in cache.

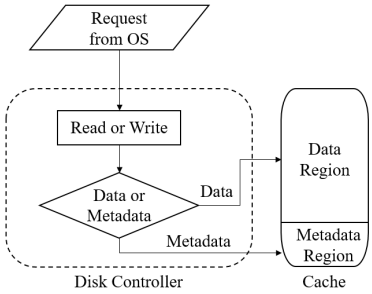


Fig. 4. Metadata detection method.

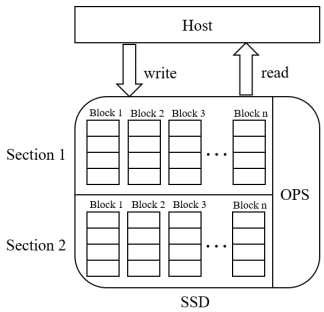


Fig. 5. Architecture of Section based page mapping FTL.

metadata is accessed more frequently, the number of metadata hits increases significantly, compensating for the reduced data hits [2, 5, 6]. On the other hand, metadata is relatively small, and metadata cache space must be allocated carefully. Allocating more space than required for metadata would reduce performance significantly, as this increased metadata cache space is at the expense of data cache space.

Fig. 3 explains the motivation behind varying the split percentage. Fig. 3(a) shows the cache divided in the middle, i.e., 50% for data and 50% for metadata cache regions. But metadata is smaller than data, and so there is lot of unassigned space within the metadata cache region. Since this space is taken from the data cache region, there is less space for data in the cache, which increases data misses, and degrades the system. Fig. 3(b) is a more reasonable split ratio, where metadata cache occupies only a small portion of the available cache space. It is important to identify the optimum split point where metadata hits exceed the loss in data hits to provide optimal performance gain. This paper addresses this issue by evaluating outcomes for a range of relative metadata and data cache region sizes.

As discussed in [1], requests to appropriate regions (data requests to the data cache region and metadata requests to the metadata cache region), are modified at the OS level, which sends one additional bit of information. This bit can be read by the disk controller and the request directed appropriately.

There are a number of tasks required to achieve the proposed design, as shown in Fig. 4:

Identify the I/O request as read or a write.

Once the I/O type is identified, differentiate between a data and metadata read.

Once the request type is completely identified, the request is satisfied by searching for the block numbers in their respective regions, i.e., if it is a data read request, then search in the data cache region; whereas if it is a metadata read request, search in the metadata cache region.

IV. SECTION BASED PAGE MAPPING FTL

Our goal is to reduce a GC process to improve the I/O performance of SSD while writing operation is performed. The write operation in IoT environment has small size of data that consists of metadata and data that can be scattered across multiple blocks in an SSD. If invalidated pages are scattered across multiple blocks when performing the garbage collection in order to get a new block in the SSD, additional page copies and block erases cause a high GC overhead.

To decrease the problem, we propose a section based page mapping based FTL that divides the SSD into two sections to store metadata and user data on different sections. Fig. 5 shows an architecture of proposed FTL scheme. This method separates metadata and user data so that data are separately stored in each section based on block number of data so that GC task can be reduced.

V. EVALUATION

The DDC and Section-based page mapping FTL discussed above are implemented in the well-known DiskSim 4.0 simulator [3]. DiskSim is written in *C*, and is widely accepted for storage system simulation. Several experiments were performed, along with benchmark traces.

1. DiskSim Simulator

DiskSim 4.0 is a well-known simulator developed to support research in storage subsystems. Disksim includes modules to simulate disks, intermediate controllers, buses, device drivers, request schedulers, disk block caches and disk array data organization. DiskSim has been successfully validated against various commercial disk drives with exceptional results [3].

2. Evaluation Traces and Workloads

Experiments were conducted with the hplajw and cello 1999 benchmarks from HP-UX. These traces have I/O events that include data and metadata access [5]. In hplajw, write requests (67%) dominate for hplajw, whereas read requests (63%) dominate for cello benchmark [18]. Overall, metadata read requests account for 40% and 10% of all read requests in cello and hplajw, respectively [12]. We used two full day traces of cello and a single day trace from hplajw, and evaluated the system performance with DDC compared to unified cache.

Various other synthetic traces were generated to evaluate DDC behavior. There are two important metadata factors consider in determining the split effectiveness: the percentage of metadata requests, and the access pattern. Therefore, we generated synthetic traces of valid trace format and assigned a few read and write events as metadata events. Since metadata is smaller, we assigned only read events that retrieved small numbers of blocks as metadata reads. From the average metadata size, block size in the cache, and relative data and metadata block sizes, we randomly marked records where access size was less than or equal to 2 blocks as metadata. For a given cache size, the experiment was conducted with three different access patterns and the

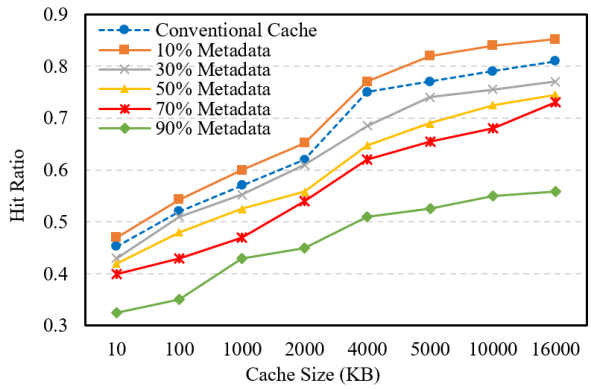


Fig. 6. Hit ratio with varying cache size for hplajw.

average value calculated, as shown in Fig. 6. For example, if there were 10000 records in the trace file, we randomly selected 4000 records as metadata reads to provide 40% metadata reads. We created trace files with metadata content at 20%, 40%, and 60%.

Experiments were conducted with nine cache sizes (10–16 MB), different read percentages (30–90%), different metadata percentages (20–60%, as discussed above), and different metadata-data split percentages ranging from 10:90 to 90:10. All experiments were conducted three times, and the average calculated. Thus, there were $9 \times 7 \times 3 \times 5 \times 3 = 2835$ data points covering all the combinations.

For a particular trace file, all the block numbers with ≤ 2 blocks requested were collected initially. For a particular random number, the request at that line in the trace file was checked to see if the block number had been collected previously, and if so it was considered to be a metadata access. This process was conducted three times with the same trace files but different random seeds. Finally, the average hit ratio was calculated. Thus, we accounted for metadata percentages in the trace file and also different access patterns. This experiment was intended to mimic metadata access patterns similar to the various test benchmarks. The entire trace file was spanned to obtain the particular percentage of metadata access.

VI. RESULTS

DDC evaluation was performed as discussed above, and in general, as the metadata region increased, the hit ratio reduced, as expected (discussed above). The read

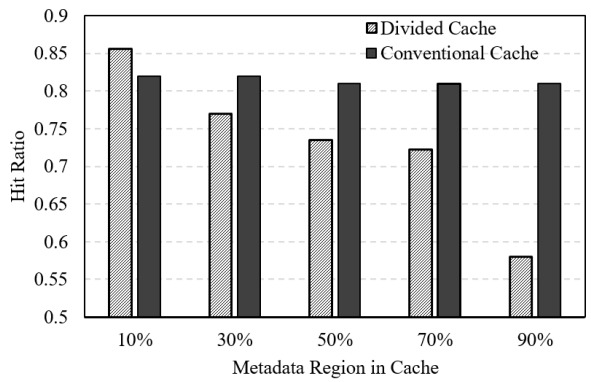


Fig. 7. Hit ratio with varying metadata cache regions for hplajw.

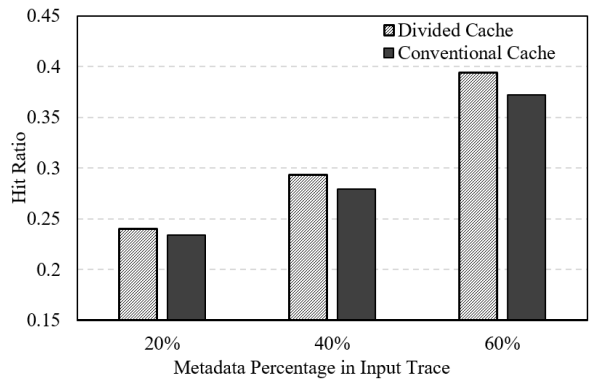


Fig. 8. Average hit ratio with varying metadata percentage in input for Synthetic trace with Validate trace format and random access.

ratio also plays a critical role in optimizing the hit ratio. DDC provides improvement only when the percentage of reads in the trace file was above 70%. For all other cases, there was very little or no optimization. Hence DDC improves hit ratio for read intensive applications, and when the data cache region is above 50%.

1. Standard Benchmarks

Fig. 6 and 7 show Cello-1990 traces (predominantly read access), which were taken over two days. Compared to conventional cache, the hit ratio improvement rises to 6% with increasing metadata cache size. Note that the maximum improvement (6%) occurs at approximately 16000 kB (0.015 GB), or 0.17% of the disk size (9.1 GB). If the disk were considered half full, this is approximately 1/3% (0.33%) of the used disk space. Thus, DDC performance with moderate size caches is more beneficial and cost-effective than a 1% unified cache.

2. Synthetic Trace Files and Random Access Methods

Fig. 7 shows that DDC will produce almost same hit ratio as conventional cache when the data to metadata region ratio is 80:20. But when the metadata region is reduced, there is an improvement in overall hit ratio over conventional cache. Fig. 8 shows the DDC is effective when the number of metadata requests in the input trace is high. As the number of metadata accesses increase, the hit ratio for metadata also increases. Fig. 9, shows significant benefit in hit ratio for DDC when the input trace read requests exceed 60%. Furthermore, DDC does

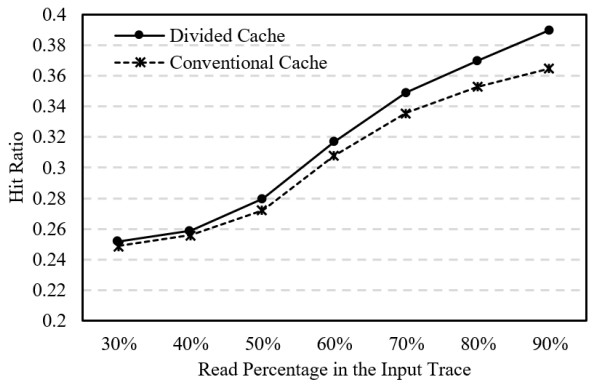


Fig. 9. Average Hit Ratio with varying read percentage in input for Synthetic trace with Validate trace format and random access.

not hinder performance for write intensive applications, with read percentages over 30%.

3. Evaluation of Section based Page Mapping FTL

We utilize the DiskSim simulation environment that is integrated with the FTL simulator for experiments. A few modifications were done in DiskSim source code to implement the section based FTL design. Major code changes were done in *flash.c*, *pagemap.c* and *ssd_interface.c* [17]. To evaluate the section based page mapping FTL, we used 2 traces that are the MSR Cambridge traces from SNIA and real block trace taken from IoT server HDD. First trace has 80% read and 15% write and last trace has 15% read and 85% write. Fig. 10 shows that block erase is 1.4% less than the page mapping FTL method. And Fig. 11 shows that the section based FTL reduced GC counts approximately 11% compared to conventional page mapping FTL.

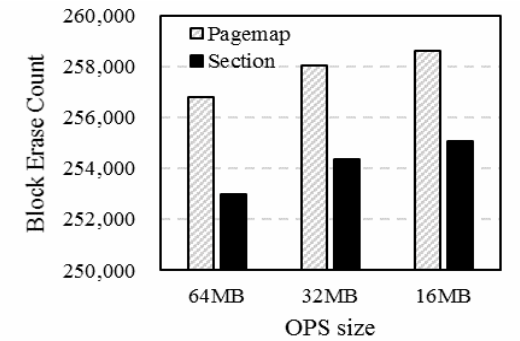


Fig. 10. Block Erase Count.

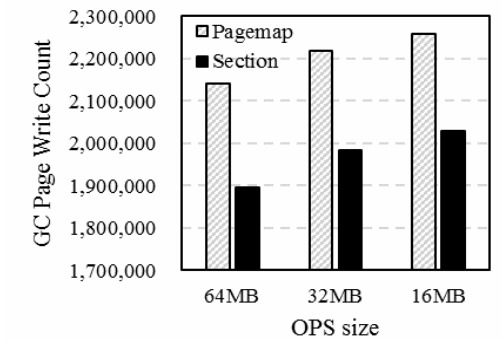


Fig. 11. GC Page Write Count.

VII. CONCLUSIONS

Significant improvements can be achieved for DDC when the percentage of reads in an application exceeds 60%, with metadata reads being at least 40% of the overall reads. For this condition, there is an optimum point where the number of total hits in DDC is more than for unified cache. This reduces traffic to the physical disk and hence provides greater response time and higher processing speeds when the external storage is involved in I/O operations. Thus, the processor can be kept busier and the bottleneck between processor and secondary storage system is significantly reduced.

We conclude that dividing the disk cache into data and metadata regions can yield positive results when the split ratio is between 70:30 to 90:10, with the latter being preferred, as this would not significantly reduce the hit ratio for data reads, but provides higher overall hit ratio.

Finally, we proposed the section based page mapping FTL scheme that is an appropriate solution for big data and IoT environment that have small data writing. This project is a work in progress. We plan to consider further dividing the SSD into several section. We also plan to conduct experiments using large scale and real workloads.

ACKNOWLEDGMENTS

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0190-15-2012, High Performance Big Data Analytics Platform Performance Acceleration Technologies Development).

REFERENCES

[1] S. Baskiyar and C. Wang, “Split disk-cache architecture to reduce read miss ratio,” *in Proc. of the 9th International PDCN*, pp.249-254, 2010.

[2] S. A. Brandt, E. L Miller, D. D. Long, and L. Xue, “Efficient metadata management in large distributed storage systems,” *in Proc. of MSST*, pp.290-298. 2003.

[3] J. S Bucy, J. Schindler, S. W Schlosser, and G. R Ganger, “The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101),” *Parallel Data Laboratory*, 2008.

[4] P. Gu, J. Wang, Y. Zhu, H. Jiang, and P. Shang, “A novel weighted graph based grouping algorithm for metadata prefetching,” *IEEE Transaction computers.*, Vol.59, No.1, pp.1-15, 2010.

[5] B. Hong, “Exploring the usage of mems-based storage as metadata storage and disk cache in storage hierarchy,” *Technical Reports, University of California at Santa Cruz*, 2003.

[6] A. Hospodor, “Hit ratio of caching disk buffers,” *in Proc. of 37th IEEE Computer Society Int. Conf.*, pp.427-432, 1992.

[7] W. W. Hsu and A. J. Smith, “Characteristics of i/o traffic in personal computer and server workloads,” *IBM Journal of Research and Development*, Vol.42m No.2, pp.347-372, 2004.

[8] W. W. Hsu and A. J. Smith, “The performance impact of i/o optimizations and disk improvements,” *IBM Journal of Research and Development*, Vol.48, No.2, pp.255-289, 2004.

[9] W. W. Hsu and A. J. Smith, “The real effect of I/O optimizations and disk improvements,” *Technical Reports*, *Computer Science Division, University of California*, 2003.

[10] C. Ruemmler and J. Wilkes, “Unix disk access patterns,” In Proc. *USENIX Technical Conference*, pp.405-420, 1993,

[11] A. J. Smith, "Disk cache miss ratio analysis and design considerations," *ACM TOCS*, Vol.3, No.3, pp.161-203. 1985.

[12] H. S. Stone, J. Turek, and J. L. Wolf, "Optimal partitioning of cache memory," *IEEE TC*, Vol.41, No.9, pp.1054-1068, 1992.

[13] A. S Tanenbaum, J. N. Herder and H. Bos, "File size distribution on unix systems-then and now," *Operating Systems Review*, Vol.40, No.1, pp.100-104, 2006.

[14] D. Thiebaut, H. S. Stone and J. L. Wolf, "Improving disk cache hit-ratios through cache partitioning," *IEEE TC*, Vol.41, No.6, pp.665-676, 1992.

[15] Q. Yang and Y. Hu, "DCD-disk caching disk: A new approach for boosting i/o performance," *23rd Annual International Symposium on Computer Architecture*, pp.169, 1996.

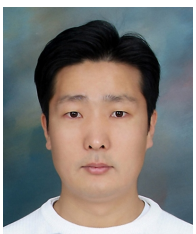
[16] Y. Zhu and Y. Hu, "Disk built-in caches: evaluation on system performance," *In Proc. MASCOTS*, pp.306-313. 2003.

[17] A simulator for various FTL schemes. http://csl.cse.psu.edu/?q=node/322, 2008.

[18] J. Lee, S. Park, M. Ryu and S. Kang, "Performance Evaluation of the SSD-Based Swap System for Big Data Processing," *In Proc. TrustCom*, 2014.

[19] A. Gupta, Y. Kim and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of pagelevel address mappings," *In Proc. ASPLOS*, 2009.

[20] J. Kang, J. Hyun, H. Maeng and S. Cho, "The Multi-streamed Solid-State Drive," *In Proc. HotStorage*, 2014.



Jung Kyu Park received the M.S. and Ph.D. degrees in computer engineering from Hongik University in 2002 and 2013, respectively. He has been a research professor at the Dankook University since 2014. In 2016, he joined the Research scientist of School of Electrical and Computer Engineering at the UNIST. His research interests include operating system, new memory, embedded system and robotics theory and its application.

Jun-yong Lee was born in Seoul, Korea. He received the B.E. degree in computer engineering from Seoul National University, Seoul, Korea in 1986. He received the M.E and Ph.D. degrees in computer engineering from the University of Minnesota in 1988 and 1996, respectively. After working as a research staff member in IBM (from 1996), he has been a professor at the Department of Computer Engineering, Hongik University, Seoul, Korea. He also has been a visiting professor at the Towson University in Maryland in 2004. His research interest includes computer architecture, embedded systems, computer security and others. He is a member of KISS, KSCI, IEEK, and KSTC. His interests include computer system architecture, logic synthesis, high-speed memory system and so on.

Sam H. Noh received the BS degree in computer engineering from the Seoul National University, Seoul, Korea, in 1986, and the PhD degree from the Department of Computer Science, University of Maryland, College Park, MD, in 1993. He held a visiting faculty position at the George Washington University, Washington, DC, from 1993 to 1994 before joining Hongik University, Seoul, Korea, where he is now a professor in the School of Computer and Information Engineering. From August 2001 to August 2002, he was also a visiting associate professor with the University of Maryland Institute of Advanced Computer Studies (UMIACS), College Park, MD. He has served as General Chair, Program Chair, and Program Committee Member on a number of technical conferences and workshops including the ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES), IEEE International Conference on Parallel and Distributed Systems (ICPADS), USENIX Conference on File and Storage Technologies (FAST), and International World Wide Web (WWW) Conference. He also serves as Associate Editor of the ACM Transactions on Storage. His current research interests include operating system issues pertaining to embedded/computer systems. He is a member of the ACM, IEEE, USENIX, and KIISE.