JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.15, NO.5, OCTOBER, 2015 ISSN(Print) 1598-1657
http://dx.doi.org/10.5573/JSTS.2015.15.5.546 ISSN(Online) 2233-4866

Preventing Fast Wear-out of Flash Cache with An Admission Control Policy

Eunji Lee and Hyokyung Bahn

Abstract—**Recently, flash cache is widely adopted as the performance accelerator of legacy storage systems. Unlike other cache media, flash cache should be carefully managed as it has peculiar characteristics such as long write latency and limited P/E cycles. In particular, we make two prominent observations that can be utilized in managing flash cache. First, a serious worn-out problem happens when the working-set of a system is beyond the capacity of flash cache due to excessively frequent cache replacement. Second, more than 50% of data has no hit in flash cache as it is a second level cache. Based on these observations, we propose a cache admission control policy that does not cache data when it is first accessed, and inserts it into the cache only after its second access occurs within a certain time window. This allows the filtering of data disruptive to flash cache in terms of endurance and performance. With this policy, we prolong the lifetime of flash cache 2.3 times without any performance degradations.**

Index Terms—**Flash cache, admission control, replacement policy, flash memory, LRU**

I. INTRODUCTION

For decades, the wide speed gap between main memory and secondary storage has been a primary source of performance degradations in computer systems. Due to the mechanical moving part of hard disks, the access time of secondary storage has been limited to tens of milliseconds, which is five or six orders of magnitude slower than DRAM access time. To relieve this problem, buffer caching mechanisms have been studied extensively in database [1-3] and operating systems [4-6]. A buffer caching system stores requested data in a certain part of DRAM memory called *buffer cache*, thereby servicing subsequent requests directly without accessing slow disk storage. However, as the size of storage data grows even faster than that of DRAM, the effectiveness of buffer caching is increasingly limited.

Recently, flash cache is widely adopted as the performance accelerator of legacy storage systems. Due to the rapid improvements in micro-fabrication processes and MLC (multi-level cell) technologies, the cost per gigabyte of flash memory has been reduced by 50% every year, and its capacity seems to be growing just as fast [7]. This makes flash memory increasingly useful for the storage cache of enterprise-scale server systems.

Flash cache serves as a second level cache, i.e., a request that has been missed from the buffer cache is sent to the flash cache. Unlike other cache media, flash cache should be carefully managed as it has peculiar characteristics such as long write latency and limited program/erase (P/E) cycles. In particular, the number of P/E cycles allowed for MLC flash memory is as small as 10^4, which is 12-15 orders of magnitude less than that of DRAM. Thus, frequent cache replacement is not desirable in flash caches.

We analyze various storage access traces, and make two prominent observations that can be exploited in managing flash cache. The first observation is that a serious worn-out problem of flash cache is encountered

Manuscript received Apr. 22, 2015; accepted Jun. 8, 2015
A part of this work was presented in Korean Conference on Semiconductors, Seoul in Korea, Feb. 2015.
Ewha Womans University
E-mail : bahn@ewha.ac.kr

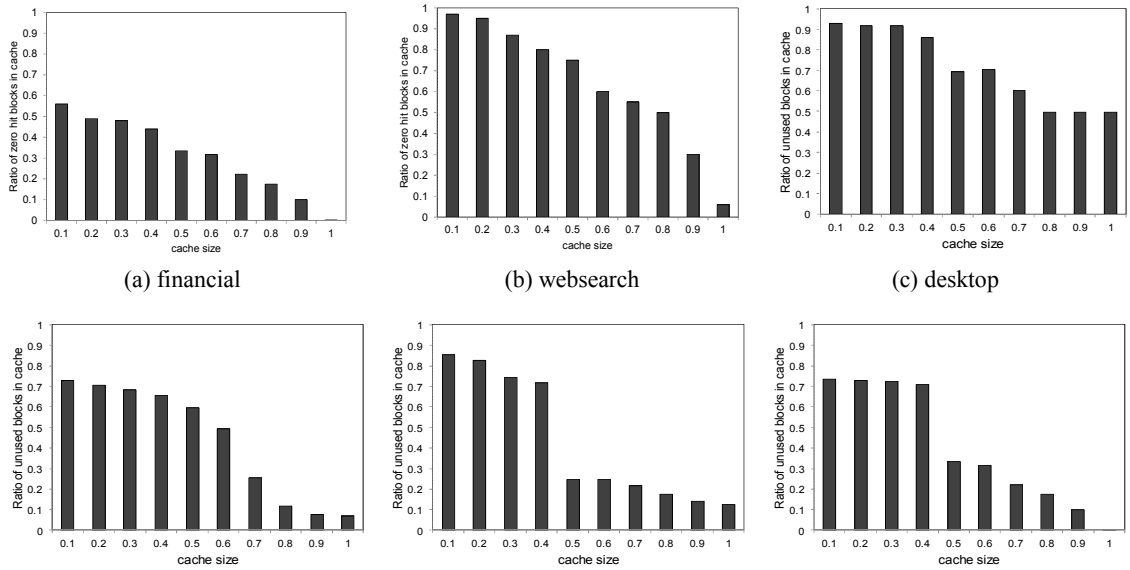


Fig. 1. The ratio of non-accessed data during the cache residence

when the working-set of a system becomes larger than the flash cache size. Specifically, with respect to the lifetime of flash memory, using multiple flash chips simultaneously as a large cache is more effective than adopting one small flash chip first and another one after the chip is worn out. This is because a small cache incurs excessively frequent replacement because hot data are replaced and inserted repeatedly due to the caching of non-reusable data if the cache size is not enough to hold data.

Our second observation is that a large portion of data are never used after entering the flash cache, which would degrade the storage performance and reduce the lifetime of flash cache if we cache them. We observe that the ratio of non-accessed data during the cache residence is more than 50%. As flash cache is a second level cache, the locality of accesses is weaker than the buffer cache, which is the main reason of such situations.

Based on the aforementioned observations, we present a new cache management scheme appropriately designed for flash cache. Specifically, we propose a cache admission control (AC) policy that detects data unlikely to be re-referenced soon and prevents them from being loaded into the flash cache.

Before storing data into the flash cache, our policy estimates whether it will be accessed again. Specifically, we do not cache data when it is first accessed, and insert it into the flash cache only after its second access occurs within a certain time window. This allows the filtering of data disruptive to flash cache in terms of endurance and performance. With this policy, we prolong the lifetime of flash cache 2.3 times compared to original LRU, without any performance degradations.

The remainder of this paper is organized as follows. Section II describes the motivation of this research. Section III describes a new cache management scheme for flash cache. In Section IV, we show the performance evaluation results to assess the effectiveness of the proposed scheme. Section V summarizes related works of this research. Finally, we conclude this paper in Section VI.

II. MOTIVATIONS

In this section, we first analyze various file access traces to investigate the access count distribution of cache data. Fig. 1 shows the ratio of cached data that are not accessed again before evicted from the cache, varying the cache size from 0.1 to 1.0 relative to the total access sizes. In reality, the cache size of 1.0 is identical to the infinite cache capacity where a complete data accesses in the trace can be cached at the same time. This is an unrealistic condition but we present it to show the complete trend of access count distributions as a function

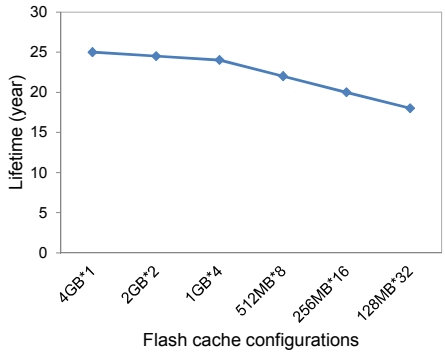
of the cache size. In practical aspects, the cache size smaller than 0.5 represents most of real system situations. As shown in Fig. 1, the ratio of data accessed only once (i.e., no access while in the cache) accounts for a large portion of cached data in all practical cases. Specifically, all workloads exhibit more than 40% of single-accessed data for practical cache sizes. This large portion of single-accessed data is not helpful to the performance improvement as they are not re-used at all before evicted from the cache. This happens as flash cache is a second level cache which receives requests only when a DRAM cache miss occurs.

Another important observation is that the large portion of single-accessed data reduces the lifetime of flash cache significantly if the cache size is relatively small. In particular, when the working-set of a system is beyond the capacity of flash cache, it encounters a serious worn-out problem caused by excessively frequent cache replacement. To quantify this effect, we investigate the lifetime of different flash cache configurations under the same cost. Specifically, we set the initial cache size to 4GB and investigate the lifetime of this cache first. Then, we split another 4GB cache into two 2GB caches and use one 2GB first, and the other 2GB after the first one is worn out. Similarly, we measure the lifetime of subsequent cache configurations reducing the cache size in half. As shown in Fig. 2, the lifetime of using many small caches repeatedly is not longer than the lifetime of adopting a large cache once. This is because a small cache incurs much more frequent replacement.

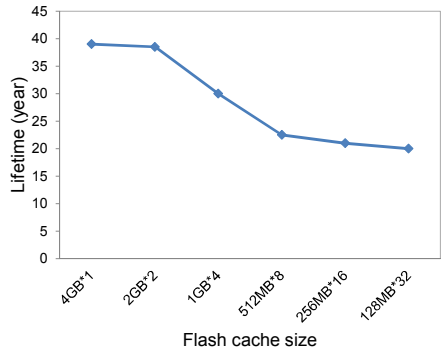
Our observations indicate that a cache admission control policy is necessary in order to detect data unlikely to be re-referenced and prevents them from being loaded into the flash cache.

III. The Flash Cache Admission Policy

The principle of caching is to retrieve data from slow storage and maintain it in the cache even after servicing the current request assuming the data to be requested again in the near future. Usually, we can expect the performance gain by caching all requested data although we do not know whether the data will be subsequently requested or not. However, as mentioned in Section II, this is not the case for flash cache, which receives a large portion of single-accessed requests and allows limited



(a) Financial workload



(b) Websearch workload

Fig. 2. Flash cache lifetime under different configurations

P/E cycles.

In this section, we present a new cache management scheme appropriately designed for flash cache. Specifically, we propose a cache admission control (AC) policy that estimates data unlikely to be re-referenced and bypasses those requests from the flash cache. In particular, we do not cache data when it is first accessed, and insert it into the flash cache only after its second access occurs within a certain time window. This allows the filtering of data disruptive to flash cache in terms of endurance and performance.

To maintain the time window, we use a small amount of history buffer that does not store the contents of actual data, but maintains the information that the data were accessed recently. Note that this history buffer consists of BPRAM (byte-addressable persistent random access memory) such as PCM (phase change memory) or STT-MRAM (spin torque transfer magnetic RAM) rather than flash cache itself. The optimal size of the history buffer varies depending not only on the workload characteristics but also on the actual flash cache size, and thus it can be a control parameter to be tuned. As a basic configuration,

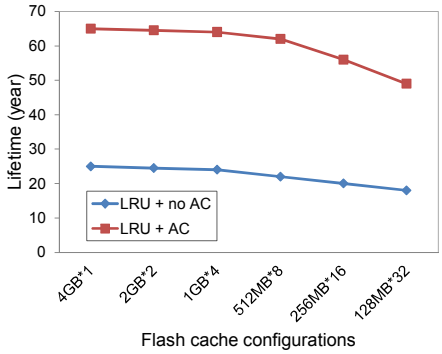
we set the number of history buffer entries to the same number of flash cache blocks. This is reasonable because a bypassed data itself is not cached but its history information is maintained as if it is cached until it is evicted from the history buffer whose size is identical to the actual cache size. Note that maintaining this size of history buffer has very low overhead because it only contains a small size of metadata (less than 20 bytes for each data block) whereas an actual data block contains 4KB of data [2].

Now, let us return to the description of the cache admission control policy. The motivation of this policy is already explained in the previous section where the access count distribution of cached data exhibits a large portion of single accesses. Thus, the second access within a short time duration is a good indicator of whether a data block will be effective or not in the near future. Therefore, bypassing flash cache on the first access is effective in discriminating non-profitable data.

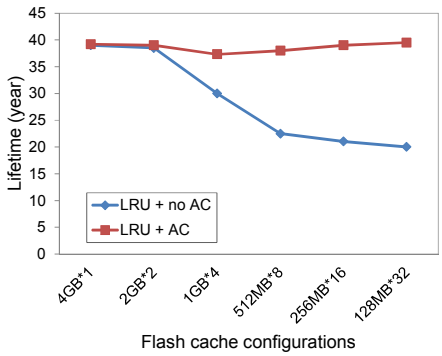
The benefit of our cache admission control policy can be observed in terms of two aspects. First, the write cost of storing non-profitable data blocks into the flash cache can be saved. In addition, our policy can protect expensive cache space from being polluted by non-profitable data blocks when the cache capacity is relatively smaller than current working set. The saved cache space can be utilized for maintaining more profitable data blocks, which also has the effect of preventing the cache from thrashing.

IV. PERFORMANCE EVALUATION

In this section, we present the performance evaluation results to assess the effectiveness of the proposed cache admission control policy. We collected file access traces through the modified strace 4.6 utility. As the trace collection was conducted at the system call layer, the traces contain all file access requests including those of cache hits as well as actual storage accesses. Note also that the traces contain the physical time of each request, and thus we replay them at the exact time of each request. These experiments provide more fair comparison than direct execution of real workloads each time. This is because workloads cannot be executed with the same user interaction and/or system status (e.g. cache state) for each workload run, making fair comparison difficult.



(a) Financial workload



(b) Websearch workload

Fig. 3. Flash cache lifetime under different configurations

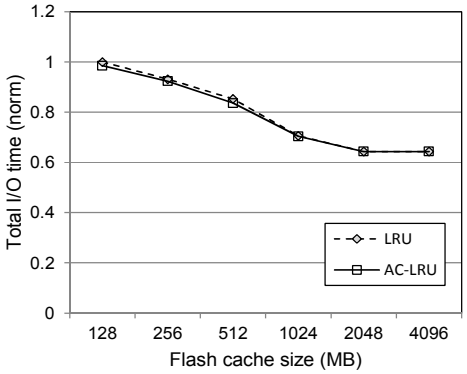
Fig. 3 shows the lifetime of flash cache for each workload as the cache configuration is varied. As shown in the figure, our cache admission control policy performs better than the conventional no-admission control policy for all cases. Specifically, the extended lifetime of flash cache is 230% on average. In this experiment, we use LRU (least recently used) as the cache replacement policy.

Our cache admission control policy extends the lifetime of flash cache significantly for all cache configurations when the financial workload is used. The extended lifetime is in the range of 160-182%. This implies that the financial workload has a lot of single-accessed data and the workload is relatively heavy to hold all requested data without the admission control policy.

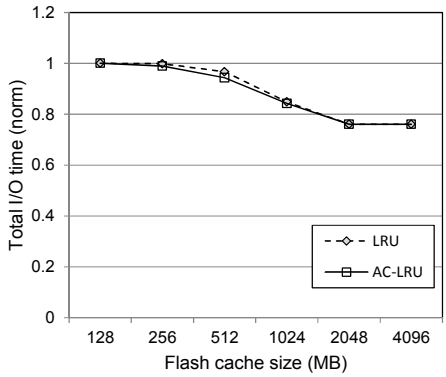
In contrast, when the websearch workload is used, the effectiveness of the admission control policy is contrasted as the cache configuration changes. In a small size cache, since the time duration that data reside in the cache is short, data are more likely to be evicted from the cache before accessed. In this situation, our admission control policy

performs well by filtering the large portion of non-profitable data blocks. It can save substantial amount of additional cost required to store them into the flash cache. In addition, the admission control policy has an effect of increasing the effective cache space. This gain becomes large when the cache size is small. Note that the marginal performance gain per increased cache size is large when the workload suffers from a small cache capacity. On the other hand, as the cache size becomes large, the gap of the two policies becomes smaller and finally their results merge to a single point. The reason is that most requests can be accommodated to a large cache irrespective of cache management policies in this case. The extended lifetime by adopting the admission control policy in websearch workload is 5-98%.

Now, let us compare the effectiveness of the admission control policy with respect to the I/O performances. As the proposed admission control policy does not cache data upon the first access, it incurs additional cache misses when the data is accessed again. Fig. 4 shows the total I/O time of the workloads as the admission control policy is adopted in comparison with those that do not use it. As shown in the figure, the results of the two policies are almost same and thus no performance degradation is observed even though we filter out a certain amount of requests. This is because single-accessing data are responsible for a large portion of such filtering target and admission control can also improve the performance due to the extension of effective cache space.



(a) Financial workload



(b) Websearch workload

Fig. 4. Performance results under different configurations

V. RELATED WORK

To alleviate the speed gap in computer systems, buffer caching algorithms have been studied extensively for decades. Section V-1 describes the goal of buffer caching for hard disks and summarizes algorithms for hard disks. As flash memory becomes increasingly popular, caching algorithms for flash memory storage are also being studied actively. Section V-2 summarizes the distinct characteristics of flash memory storage and the caching algorithms for flash memory.

1. Buffer Caching Algorithms for Hard Disks

Most operating systems, including Linux, have been optimized under the assumption that storage devices are an order of magnitude slower than main memory. Buffer caching algorithms therefore focus on maximizing the hit ratio by replacing the block least likely to be referenced again.

The LRU (least recently used) algorithm does this by exploiting the recency of the last reference time. LRU evicts the block that has the furthest last reference time if free space is needed. This is based on the temporal locality of block references, which implies that a more recently referenced block is more likely to be referenced again in the near future. The LFU (least frequently used) algorithm uses the frequency of block references instead of recency to identify blocks likely to be referenced again. LFU maintains the reference count of each block in the cache, and evicts the least frequently referenced block if needed.

There have been studies that aim to combine the advantages of LRU and LFU. O'Neil et al. present the LRU-*k* replacement algorithm to address the problem of LRU that cannot consider the reference frequency of blocks [3]. LRU-*k* decides blocks to be replaced based on

the time of the kth-to-last reference. A larger k can discriminate better between frequently and infrequently referenced blocks. However, LRU-k ignores the recency of the k–1 references, and thus it does not work adaptively to changing workloads when k is large.

Johnson and Shasha propose a block replacement policy called 2Q [2]. This algorithm divides the LRU cache list into two queues, namely A_1 and A_m, and places blocks initially in the A_1 queue. If a block in the A_1 queue is re-referenced, it is promoted to the A_m queue. The replacement occurs in A_1 according to LRU. In this algorithm, blocks referenced only once are quickly removed from the cache, while blocks that are repeatedly referenced can remain in the buffer cache for an extended period of time.

Lee et al. propose LRFU (least recently/frequently used) algorithm that subsumes the LRU and LFU algorithms [5]. In LRFU, each cached block is associated with a CRF (combined recency and frequency) value that estimates the re-reference likelihood of the block in terms of both recency and frequency. All past references to a block during its residence in the cache are reflected in CRF and a reference's contribution decreases as time progresses.

ARC (adaptive replacement cache) is another algorithm that adaptively considers the recency and frequency of references [8]. ARC maintains two LRU lists for cache directory, T_1 to capture recency and T_2 to capture frequency, and adaptively adjusts their sizes according to the evolution of workloads.

Some algorithms detect reference patterns and allocate separate cache space to each detected pattern. UBM (unified buffer management) is a representative algorithm in this class [4]. UBM classifies referenced blocks into three patterns, sequential, looping, and other references, and then allocates cache space to each patterns based on their marginal gain.

LIRS (low inter-reference recency set replacement) uses the concept of inter reference recency to accurately estimate future block references [6]. LIRS divides blocks into two sets: HIR (high inter-reference recency) and LIR (low inter-reference recency) block sets. LIRS gives higher caching priorities to the LIR block set as it contains frequently accessed blocks. When there is a reference to an HIR block, it can be promoted to the LIR block set if its inter-reference recency is shorter than the recency of an LIR block. If free space is needed, LIRS evicts blocks in the HIR block set.

DULO (dual locality) exploits both temporal locality and spatial locality in selecting victim blocks [9]. Because hard disks have relatively large seek time compared to transfer time, both a randomly accessed block and a certain number of sequentially accessed blocks have almost the same cost to read. DULO considers this by managing the LRU stack according to the recency and the size of block sequences.

2. Caching Algorithms for Flash Memory

Recently, as NAND flash memory is widely adopted as the secondary storage of mobile systems, there have been extensive studies on buffer caching algorithms for NAND flash memory. CFLRU (clean-first LRU) is a cache replacement algorithm for flash memory that considers the hit ratio as well as the physical characteristics of NAND flash memory, in which reading and writing have different I/O costs [10]. CFLRU can accommodate the different eviction costs of a clean block, which can simply be discarded, and a dirty block, which should be written back to flash memory. CFLRU maintains a certain cache area called window and delays the eviction of dirty blocks in the window as long as a clean block is available for eviction.

LRU-WSR (LRU with write sequence reordering) is another replacement algorithm that favors dirty blocks [11]. Basically, LRU-WSR also manages blocks using the LRU list. Instead of setting the window area, LRU-WSR gives one more chance to a dirty block when it reaches the LRU position in the list. In this way, LRU-WSR considers asymmetric operation costs of reads and writes in the flash memory.

FAB (flash-aware buffer management) is proposed as a buffer replacement algorithm in flash-based PMP systems [12]. PMP systems commonly have long sequential accesses for media data and some short accesses for metadata at the same time. One problem with this situation is that short write accesses cannot be buffered for a long time because they are pushed away by a large amount of sequential data. To cope with this problem, FAB manages buffered data from the same NAND flash memory block as a group, and replaces them together. Specifically, FAB replaces the group with

the largest number of buffers first, which is usually large sequential data.

BPLRU (block padding least recently used) is a write buffer management algorithm to improve the random write performance of flash storage [13]. Similar to FAB, BPLRU groups buffers from the same NAND flash memory block, and replaces them together. When a buffer is accessed by a write operation, buffers in the same group are moved together to the MRU position of the list. BPLRU selects buffers in the LRU position as a victim, and flushes all data in the group. This block-level flushing reduces the random write cost of NAND flash memory.

CLC (cold and largest cluster) is another write buffer replacement algorithm for NAND flash memory [14]. CLC uses BPRAM as its write buffer. Similar to FAB and BPLRU, CLC manages blocks from the same NAND flash memory blocks together. When replacement is needed, CLC selects a NAND block group with the largest number of blocks among groups that have not been referenced recently.

VI. CONCLUSION

Flash cache is widely adopted in modern high performance storage systems. In this paper, we made two prominent observations that can be utilized in managing flash cache efficiently. The first one is that a serious worn-out problem of flash cache happens when the working-set of a system is beyond the capacity of flash cache due to excessively frequent cache replacement. Secondly, we observed that more than 50% of data has no hit in flash cache as it is a second level cache. Based on these observations, we proposed a cache admission control policy that does not insert data into the flash cache when it is first accessed, and allows it to be cached only after its second access occurs within a certain time window. This allows the filtering of data disruptive to flash cache in terms of endurance and performance. With this policy, we showed that the lifetime of flash cache can be extended by 230% without any performance degradations.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation (NRF) grant funded by the Korea government (MEST) (No. 2011-0028825) and the Ministry of Science, ICT & Future Planning (No. NRF-2014R1A1A3053505).

REFERENCES

[1] Faloutsos, C., Ng, R., & Sellis, T., Flexible and adaptable buffer management techniques for database management systems, IEEE Transactions on Computers, vol. 44, no. 4, pp. 546–560, 1995.

[2] Johnson, T. & Shasha, D. 2Q: a low overhead high performance buffer management replacement algorithm. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), pp. 439–450, 1994.

[3] O'Neil, E.J., O'Neil, P.E., & Weikum, G. The LRU-K page replacement algorithm for database disk buffering. In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 297–306, 1993,.

[4] Kim, J.M., Choi, J., Kim, J., Noh, S.H., Min, S.L., Cho, Y., & Kim, C.S. A low-overhead, high-performance unified buffer management scheme that exploits sequential and looping references, In Proceedings of USENIX Symposium on Operating System Design and Implementation (OSDI), pp. 119–134, 2000.

[5] Lee, D., Choi, J., Kim, J.H., Noh, S.H., Min, S.L., Cho, Y., & Kim, C.S. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Transactions on Computers, vol. 50, no. 12, pp. 1352–1361, 2001.

[6] Jiang, S. & Zhang, X. Making LRU friendly to weak locality workloads: a novel replacement algorithm to improve buffer cache performance. IEEE Transactions on Computers, vol. 5, no. 8, pp. 939–952, 2005.

[7] Leventhal, A. Flash Storage Memory. Communications of the ACM, vol. 51, no. 7, pp. 47–51, 2008.

[8] Megiddo, N. & ModhaHA, D.S. ARC: a self-tuning,

low overhead replacement cache. In Proceedings of the 2nd USENIX Conference on File and Storage Technologies, pp. 115–130, 2003.

[9] Jiang, S. & Zhang, X. Making LRU friendly to weak locality workloads: a novel replacement algorithm to improve buffer cache performance. IEEE Transactions on Computers, vol. 5, no. 8, pp. 939–952, 2005.

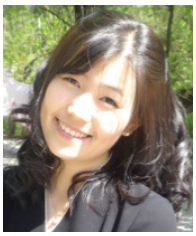
[10] Park, S., Jung, D., Kang, J., Kim, J., & Lee, J. CFLRU: replacement algorithm for flash memory. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES), pp. 234–241, 2006.

[11] Jung, H., Shim, H., Park, S., Kang, S., & Cha, J. LRU-WSR: integration of LRU and writes sequence reordering for flash memory. IEEE Trans. Consumer Electron., vol. 54, no. 3, pp. 1215–1223, 2008.

[12] Jo, H., Kang, J., Park, S., Kim, J., & Lee, J. FAB: flash-aware buffer management policy for portable media players. IEEE Trans. Consumer Electron., vol. 52, no. 2, pp. 485–493, 2006.

[13] Kim, H. & Ahn, S. BPLRU: a buffer management scheme for improving random writes in flash storage. In Proceedings of the 6th USENIX Conference File and Storage Technologies (FAST), 2008.

[14] Kang, S., Park, S., Jung, H., Shim, H., & Cha, J. Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices. IEEE Trans. Consumer Electron., vol. 58, no. 6, pp. 744–758, 2009.



Eunji Lee received the PhD degree in computer engineering from Seoul National University in 2012. She was a visiting scholar at the Department of EECS, the University of Michigan, Ann Arbor, and a senior engineer at the Samsung Electronics, Co., Ltd. She is currently an assistant professor in the software department, Chungbuk National University, Korea. Her research interests include operating systems, embedded systems, and storage systems. She has published more than 40 papers in leading conferences and journals in these fields, including IEEE Trans. Computers, IEEE Trans. Knowledge & Data Engineering, and ACM Trans. Storage. She also received the Best Paper Awards at USENIX FAST in 2013.

Hyokyung Bahn received the BS, MS, and PhD degrees in computer science from Seoul National University, in 1997, 1999, and 2002, respectively. He is currently a full professor of computer engineering at Ewha University, Korea. His research interests include operating systems, storage systems, embedded systems, and real-time systems. He received the Best Paper Awards at the USENIX Conference on File and Storage Technologies in 2013.