Considering Block Popularity in Disk Cache Replacement for Enhancing Hit Ratio with Solid State Drive

Yonjoong Ryou, Byungjun Lee, SanghyunYoo, Hee Yong Youn
College of Information and Communication Engineering
Sungkyunkwan University
Suwon, Korea
e-mail: {yjryu, byungjun, shyunyoo, youn7147}@skku.edu

***Abstract*—The Solid State Drive (SSD) is now becoming a main stream in storage systems. It is widely deployed as cache for hard disk drive (HDD) to speed up the execution of data intensive applications. In this paper we propose a novel block replacement algorithm for flash-based disk cache, named Block Replacement based on Popularity(BRP). Using the frequency and recency of block access, it calculates the block popularity to select the block which will be evicted from SSD. This avoids cache pollution and keeps popular blocks in SSD cache, leading to high hit ratio. Meanwhile, the proposed scheme reduces block replacements, and thus incurs less write operations to SSD. As a result, BRP enhances the performance of storage and the lifetime of SSD. Computer simulation demonstrates that the proposed scheme consistently outperforms five existing cache replacement algorithms with two different kinds of traces.**

Keywords—Disk Cache; Solid State Disk; Cache Replacement; Block Popularity; Ghost Queue.

I. INTRODUCTION

NAND flash-based Solid State Drive (SSD) is now widely used in various computing environment such as laptop, desktop, and server storage system due to numerous merits including non-volatility, fast random access, shock resistance, small size, and low power consumption. SSD has attracted a lot of attentions recently as effective replacement of Hard Disk Drives (HDDs). However, the price of SSD is still higher than HDD although it is decreasing rapidly. Therefore, SSD is desirable to serve as cache storage of HDD for cost efficiency.

Meanwhile, SSD has a few undesirable features including limited erase/write cycles and drastically shortened lifetime with excessive operations. These are serious issues, and need to be taken care of by software approach. Efficient disk cache management is required at the HDD level to avoid unnecessary writes to SSD. For example, sequential access pattern is often observed in the scan operation handled with typical data analysis applications. If the LRU policy is applied to manage the HDD, all sequentially-accessed data blocks will be moved from HDD to SSD. Caching these data blocks in SSD inevitably requires a large number of write operations to SSD, but without any benefit of caching as these blocks are never reused again. On the other hand, cache pollution occurs as the spaces are wasted to cache useless data blocks. The traditional cache management is not suitable for SSD cache.

The existing cache management algorithms were designed primarily for buffer cache resident in RAM. They focused exclusively on the improvement of hit ratio to maximize the utilization of cache, and a variety of approaches have been proposed. The most widely used LRU algorithm exploits temporal locality by keeping the most recently used blocks for future use. More sophisticated algorithms such as LRU-K [2], MQ [3], ARC [4], LARC[5] were also proposed to overcome the weakness of LRU in coping with the access patterns of weak temporal locality. They try to improve hit ratio by identifying seldom accessed blocks and evicting them early from the cache. However, direct application of these algorithms is inappropriate for SSD-based disk cache because the endurance issue was ignored.

In this paper, thus, we propose a novel block replacement algorithm for SSD-based disk cache, named Block Popularity Replacement based on Popularity(BRP). BRP is different from other algorithms in considering the frequency and recency of block, trying to keep hot blocks in SSD cache and evict seldom accessed blocks out of SSD cache. BRP uses block popularity to identify potentially popular blocks. As a result, blocks in SSD cache tend to be more popular and kept longer, leading improved the hit ratio. Meanwhile, BRP reduces the number of block replacements and hence incurs less write operation to SSD. Computer simulation demonstrates that BRP outperforms and reduces write operation in SSD cache compared with five different cache replacement algorithms with two different kinds of workloads.

The rest of the paper is organized as follows. In Section 2 the basics of SSD and the issues with the existing cache management algorithms are discussed. Section 3 introduces the proposed scheme. Performance evaluation with trace-driven simulation is presented in Section 4, and the conclusion is given in Section 5.

978-1-4799-8676-7/15/$31.00 copyright 2015 IEEE
SNPD 2015, June 1-3 2015, Takamatsu, Japan

II. Background And Related Works

This section introduces the hardware characteristics and the operational mechanism of flash memory along with the software layer providing the block device interface.

A. SSD-based Cache

There are basically two usage models for SSD-based disk cache [6](Fig. 1). In the first model, SSD is used as an extension of system memory. In this model RAM and SSD are managed as a single unified cache tier for HDD. In the other model, it is used as extended disk lying beneath the standard block interface to serve as a second level cache. It is transparent to other components in the system.

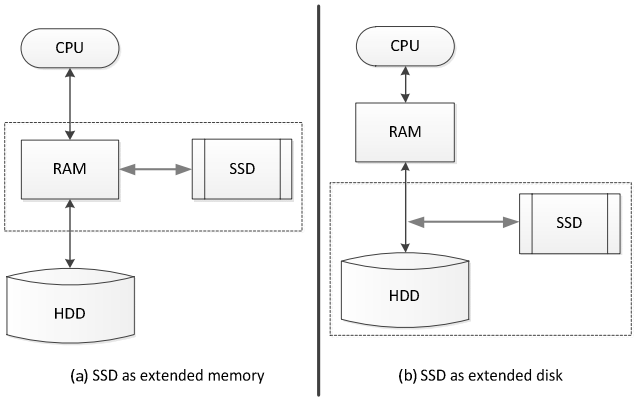


Fig. 1. Two usage models of SSD-based disk cache.

Due to the inclusive property of multi-level cache [7], the extended disk model is often inferior to the other one with respect to hit ratio. However, implementing the extended memory model usually involves the modification of the operating system or the application itself [8]. This is expensive or even impossible often. Consequently, the extended disk model is more common in the field, and it is adopted in this research.

B. Cache Replacement

The LRU replacement policy is widely used for the replacement of cache because of the simplicity, which replaces the least recently used page in the cache. It performs well in some applications where a burst of references to infrequently used blocks may cause replacement of commonly referenced blocks in the cache [3,13]. The LRU-K [2] replacement policy considers the time of the K^{th}-to-last reference to the page. It replaces the page of the least reference frequency and the least recency of the reference.

The multi-queue replacement policy (MQ) [3] uses m (typically, $m = 8$) LRU queues: Q_0 to Q_{m-1}, where Q_i contains the pages that have been accessed at least 2^i times but not more than $(2^{i+1} - 1)$ times. MQ also maintains a cache history queue, Q_{out}. The pages can be moved between Q_0 to Q_{m-1}. On a page hit, the frequency of the page is incremented. The page of high frequency but low recency is replaced in the queues, and the MQ algorithm uses both the recency and frequency of accesses of each page.

The adaptive Replacement Cache (ARC) [4] policy dynamically, adaptively, and continually balances between the recency and frequency in an online and self-tuning fashion in response to evolving and changing access patterns. It uses a learning rule to adaptively and continually revise its assumptions about the workload. The ARC scheme maintains two LRU lists of pages. One list, say L1, contains the pages that have been seen only once "recently" while the other list, say L2, contains pages that have been seen at least twice. Hence, L1 captures "recency" while L2 does "frequency". It maintains a cache directory that remembers twice as many pages as the ones in the locality of cache memory. While the total size of L1 and L2 is fixed, the size of L1 and L2 are varied using a learning rule.

The Lazy Adaptive Replacement Cache (LARC) [5] scheme identifies seldom accessed blocks and keeps them out of cache. It uses a Ghost cache, which is an LRU queue storing only the block identifiers. The first-time accessed blocks are stored in Ghost cache as the candidate of replacement. If it is accessed again it is regarded as popular one and thus moved to the physical cache on SSD. The physical cache is managed with an LRU queue. LARC first searches the queue for a request of a block. If the target block is found there, the request is redirected to the corresponding block on SSD and the block is moved to the end of the queue. Otherwise, the Ghost Queue is searched to determine whether cache replacement needs to be triggered. If the block is found from Ghost Queue, LARC replaces the LRU block in the queue with this block. If the block is absent from both the queue and Ghost Queue, the request is redirected to HDD and its identifier is inserted at the head of Ghost Queue. If Ghost Queue is full, the LRU block is removed.

The existing replacement algorithms for traditional hard disks were designed to minimize the page miss ratio assuming the cost of page read and write are equal. In the case when flash memory is used as storage medium, it is necessary to reflect the characteristics of flash memory in cache management. This paper distinguishes itself from the existing algorithms in considering the recency and frequency for block or page replacement. The residence time in SSD Queue and hit count of a block are used to decide the popularity.

III. The Proposed Scheme

In this section the proposed BRP scheme is presented, which exploits the popularity of blocks for block replacement with SSD cache. According to LARC, the accessed blocks are classified into two types, one time accessed blocks and blocks accessed more than one time. The second type block are most likely to be accessed again later, and thus need to be kept in the SSD cache. The BRP scheme calculates the popularity of blocks to keep the seldom accessed blocks out of SSD Cache. It will prevent cache pollution and lead to high hit ratio and less write operations.

A. The Basic Approach

The basic idea of BRP is to properly identify seldom accessed blocks and evict them out of the SSD cache. This is achieved by the notion of block popularity, which is managed

by *S_Queue*. It also adopts ghost queue as a filter used to reduce the SSD write operations. *S_Queue* is a queue whose entry indicates the value of popularity of the corresponding SSD block, with the block of highest (lowest) popularity at the head (tail). Ghost queue contains the metadata of the block referenced once, with the most (least) recently used one at the head (tail). Algorithm 1 shows the operation of the proposed scheme using ghost queue for block replacement.

Algorithm 1. Ghost Queue-based Replacement

Initialize : $Size_{GQ} = 0.1 * Size_{SQ}$
if requested *Block* B_n is in ***S_Queue*** **then**
  $Size_{GQ}$ = max(0.1 * $Size_{SQ}$, $Size_{GQ}$ - $Size_{SQ}$ / $Size_{SQ}$ - $Size_{GQ}$)
  Redirect the request to the corresponding SSD block
  return
end if
$Size_{GQ}$ = min(0.9 * $Size_{SQ}$, $Size_{GQ}$ + $Size_{SQ}$ / $Size_{GQ}$)
if *Block* B_n is in ***GhostQ*** **then**
  Remove *Block* B_n from ***GhostQ***
  Allocate the *Block* B_n to the ***S_Queue***
  if ***SSDQ*** > $Size_{SQ}$ **then**
    Remove the lowest popularity of *Block* in ***S_Queue***
    Allocate the *Block* B_n to ***S_Queue***
  else
    Allocate the *Block* B_n to free ***S_Queue*** *Block*
  end if
  else
   Allocate *Block* B_n to the MRU block in ***GhostQ***
   if ***GhostQ*** > $Size_{GQ}$ **then**
     Remove the LRU *Block* of ***GhostQ***
   end if
end if

*$Size_{SQ}$ **:** Size of SSD Queue, $Size_{GQ}$: Size of Ghost Queue,

The size of Ghost Queue ($Size_{GQ}$) indicates how long a block needs to be placed in Ghost Queue. Obviously, large size Ghost Queue leads to high hit ratio, but also increases write operations. Therefore, $Size_{GQ}$ is dynamically changed according to the hit ratio of SSD cache. The self-tuning policy will be effective for coping with varying block access patterns.

Fig. 2 shows the three cases of operations of block move,

- *Case* (1): When a request for a block arrives, SSD Queue is first checked if the requested block is in the SSD cache or not. If true, the access information on the block is updated.
- *Case* (2): If the requested block is not found, Ghost Queue is checked for its presence. Then the requested block is moved to the SSD cache by referring to the metadata of the block in Ghost Queue.
- *Case* (3): When the requested block is not in both SSD Queue and Ghost Queue, the metadata of the request block is put in the head of Ghost Queue.

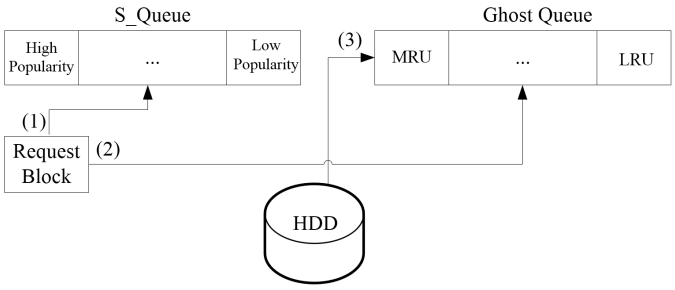


Fig. 2. The operation of the proposed BRP Scheme.

B. Block Popularity

To avoid cache pollution, a new metric called Block Popularity is adopted, which is decided by *residence time* and, *hit count* of the block. It is calculated using the variables summarized in Table I.

TABLE I.
VARIABLES USED TO CALCULATE BLOCK POPULARITY

Variable	Description
$P(n)$	Popularity of Block_n
$T_A(n)$	The last access time of Block_n
$T_R(n)$	Residence time of Block_n
$H(n)$	Hit count of Block_n

The residence time of a block, T_R, is defined as the time interval between the current time t, and the last access time of the block. It is thus

$$T_R(n) = t - T_A(n) \tag{1}$$

T_R of a block indicates how long the block has been staying in SSD cache since the last reference. The second information on a block is the hit count, defined as the number of read/write hits on the block while it is in the SSD cache. When a block is put in SSD cache, the block hit count is initialized to 1, and it is increased whenever the block in SSD Cache is requested.

Using the block hit count and residence time, the popularity of block n, $P(n)$, is obtained as follows.

$$P(n) = \frac{H(n)}{T_R(n)} = \frac{H(n)}{t - T_A(n)} \tag{2}$$

If $T_R(n)$ is 0, Block_n has just been put in SSD Queue. $P(n)$ is initialized to 1. After the block is requested again, the $P(n)$ value is changed. If the residence time of a block is not taken into account in deciding its popularity, the temporal locality is missed. Even with the same hit count, the blocks perhaps have different residence times, and consequently different probability of access in the future.

Compared to the existing schemes, the proposed scheme can effectively allocate the blocks to SSD cache based on the

block popularity. Fig. 3 shows an example of cache replacement with the popular LRU cache replacement policy.

Here the SSD cache is composed of eight blocks. B_{43} is the LRU block when SSD Queue is full, and thus it has the highest priority of eviction from the cache. On the other hand, B_{53} is MRU block, and thus it will stay long in the cache (also SSD Queue). B_{43} has only one hit after located in SSD Queue, it has not been requested again until t = 1000. However, when requested at t = 1000, it is moved to the head of the queue.

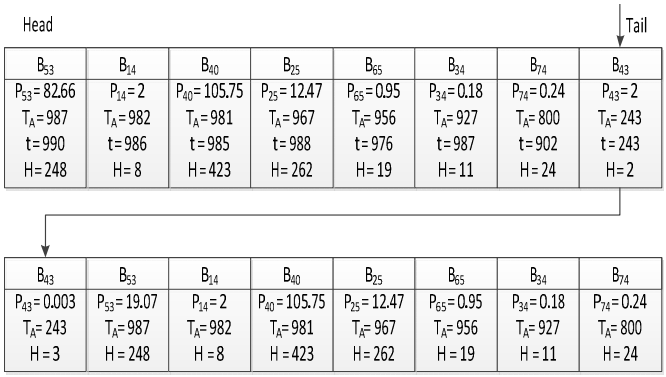


Fig. 3. An example of LRU algorithm.

4 shows the operation of the proposed BRP scheme for the same situation with Fig. 3. The first row of Fig. 4 is sorted by block popularity. Observe from the figure that B_{43} was first located in the middle of the SSD Queue. Recall that the P value is initialized to 1(P_{43} = 1) when the block is inserted in SSD Queue for the first time. At t = 1000, B_{43} is requested again as in Fig. 3, where it was located in the MRU position. Nevertheless, it is moved to the tail of the SSD Queue with the proposed scheme since B_{43} has the lowest popularity. In other words, B_{43} is "cold" block. If B_{43} is not requested again, it wastes the capacity of SSD cache until finally evicted from SSD Queue. Observe that the residence time of B_{43} in Fig. 3 is much longer than that of Fig. 4. In the long run, B_{43} will degrade the hit ratio. The policy employed in the proposed scheme can effectively prevent cache pollution and keep "hot" blocks in SSD Queue for higher hit ratio.

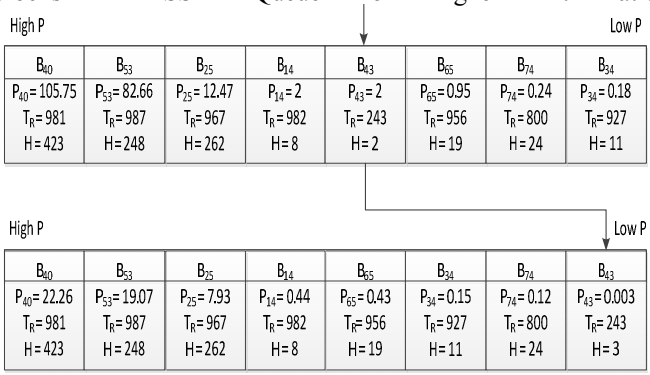


Fig. 4. An example of the operation of the proposed BRP scheme.

Algorithm 2. Cache Replacement with Block Popularity

```
1:  if requested Block B_n is in GhostQ
2:    Block B_n -> Hit Count = 1;
3:    Block B_n -> Block Popularity = 1;
4:    Block B_n -> Last accessed time = Current accessed time;
5:    Allocate the Block B_n in S_Queue
6:  end if
7:  if requested Block B_n is in S_Queue
8:    Block B_n -> Hit Count ++;
9:    Update the Block B_n -> Last accessed time;
10:   Block B_n BP = Hit Count /residence time;
11:   Update the Popularity of Block B_n
12: end if
13: else
14:   if S_Queue is Full then
15:     for each Block B_n -> BP in S_Queue do
16:       Calculate B_n -> BP = Hit Count / t – Last accessed time;
17:       Evict the lowest Popularity of Block B_n in S_Queue
18:       Allocate the requested Block B_n of GhostQ to S_Queue
19:     end for
20:   end if
```

Algorithm 2 presents the cache replacement with block popularity. The scanning process (line 14-20) computes popularity of all the blocks in the SSD Queue. Line 1-6 are for a block inserted in the SSD Queue first time. Line 7-12 are for updating the hit count when the requested block is hit. Also, the residence time in SSD Queue is calculated based on the requested time. Using the block information, BRP updates the popularity of the blocks. According to the calculated popularity of the block, the block of lowest popularity will be evicted from SSD Queue, and then a new block which is hit from the Ghost Queue is allocated.

IV. Performance Evaluation

In this section the proposed BRP algorithm is evaluated by trace driven simulation with real-world workloads. It is also compared with five previous cache replacement algorithms. Here hit ratio and number of write operations are compared with different workload traces commonly employed in the existing researches. Each of them has a different access pattern, which is summarized in Table II. These traces are available from UMass Trace Repository [9].

TABLE II.
THE CHARACTERISTICS OF I/O WORKLOAD TRACES

Trace	Read Request	Write Request	Total Request	Ratio of Read
Websearch	1055236	212	1055448	99.97%
Financial	3046112	653082	3699194	82.34%

The Websearch trace is random read intensive I/O workload obtained from a web search engine. It is unique in that 99% of the requests are reads while only 1% are write requests.

The Financial trace are collected from the online transaction processing(OLTP) applications running at two

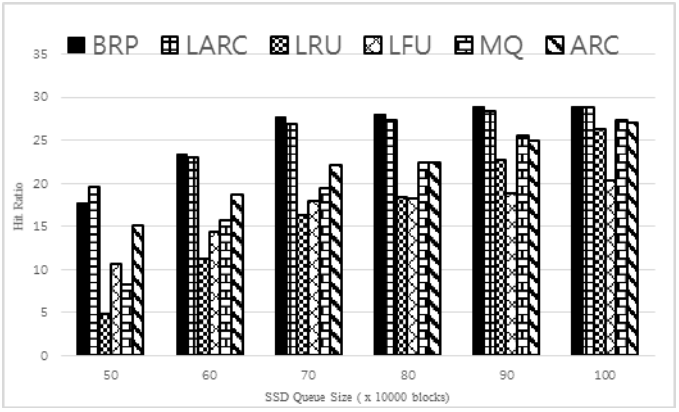
large financial institutions, which are random write intensive I/O workload.

The sizes of the blocks are different according to the characteristic of workload trace. The block size of *Websearch* is 4KB and 512Byte for *Financial*. For the performance evaluation, we implemented BRP using C language. For the comparison in terms of hit ratio and the number of write operation, five other algorithms are chosen including LRU, LFU, MQ [3], ARC [4] and LARC [5].

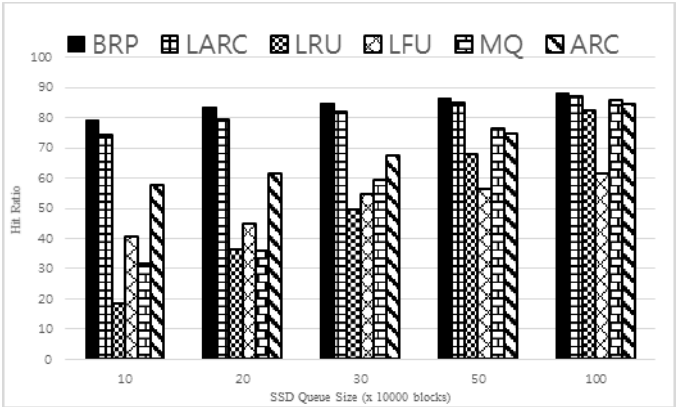
The main goal of BRP is to increase hit ratio in SSD cache, which is the ratio between the number of hits in the SSD Queue and the total number of requests.

Fig. 5 shows the comparison of hit ratios of six different cache replacement algorithms with two different traces as the size of SSD Queue increases.

Observe from the figure that the proposed scheme consistently allows higher hit ratio than the others. In addition, as the size of SSD Queue is increases, the hit ratio also slightly grows. The Websearch trace displays much smaller hit ratio than Financial trace due to the nature of the applications.



(a) *Websearch*



(b) *Financial*

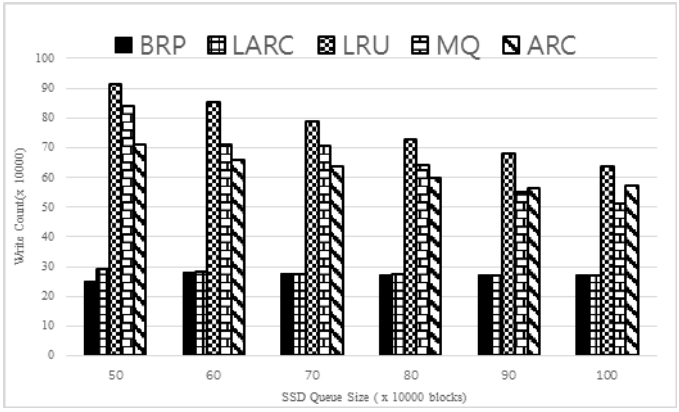
Fig. 5. The comparisons of hit ratio of different algorithms.

Notice that the Least Recently Used (LRU) policy displays lowest hit ratio due to its simplicity.

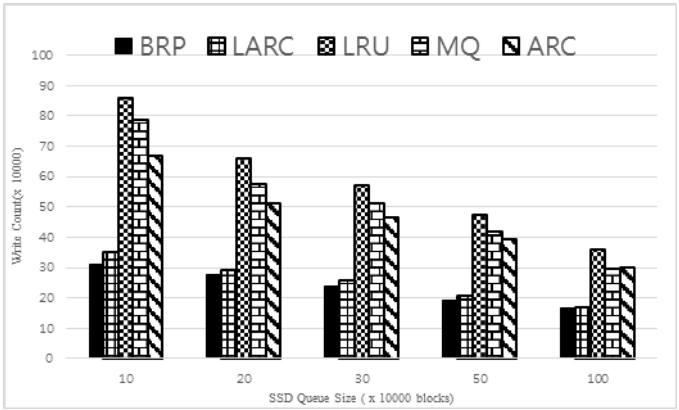
The second goal of BRP is to reduce write operations which allows extended lifetime of SSD. Various researches are thus focused on reducing write operations.

LARC [5] used Ghost Queue to prevent unnecessary block replacement, which affects the frequency of write operation. We also employ Ghost Queue to reduce write operation. When a block is written to SSD Queue and block replacement occurs, the write count increases. Until the SSD Queue become full, the write count is continuously increased. When the SSD Queue is full, BRP decides whether the block is kept in SSD cache or not.

Fig. 6 compares the number of write operations of five different cache replacement algorithms with two different kinds of traces as the size of SSD Queue increases. Considering the popularity of the block, BRP consistently reduces write operations regardless of the applications and SSD Queue size.



(a) *Websearch*



(b) *Financial*

Fig. 6. The comparisons of write counts of different algorithms.

V. CONCLUSION

In this paper we have proposed an enhanced cache replacement scheme for flash-based disk cache. It considers

both the frequency and recency of the access to the blocks to decide whether the blocks are kept in SSD or not. This avoids cache pollution and keeps popular blocks in SSD cache, leading to high hit ratio. Also, the proposed scheme reduces the frequency of block replacement, and thus incurs less write operations to SSD. Consequently, it enhances the performance of the storage and lifetime of the SSD. Computer simulation demonstrated that the proposed scheme consistently increases the hit ratio and reduces write operations in SSD cache, compared with five different cache replacement algorithms for two different kinds of traces.

As a future course of study, we plan to reduce the execution time of the proposed scheme. In this paper hash table was used for fast operation, but still needs improvement. Furthermore, the effectiveness of the replacement algorithm will be more comprehensively tested with different scenarios and operation conditions.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2012R1A1A2040257 and 2013R1A1A2060398), the second Brain Korea 21 PLUS project, ICT R&D program of MSIP/IITP (1391105003), and Samsung Electronics (S-2014-0700-000). Corresponding author: Hee Yong Youn.

REFERENCES

[1] SD. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A.Rowstron, "Migrating server storage to SSDs: analysis of tradeoffs," in Proceedings of the 4th ACM European conference on Computer systems, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 145–158. TheGoogle file system. SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[2] Y. Smaragdakis, S. Kaplan, and P. Wilson, "Eelru: simple and effective adaptive page replacement," in ACM SIGMETRICS Performance Evaluation Review, vol. 27, no. 1. ACM, 1999, pp. 122–133.

[3] Y. Zhou, J. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches," in Proceedings of the General Track: 2002 USENIX Annual Technical Conference, 2001, pp. 91–104.K. Elissa, "Title of paper if known," unpublished.

[4] N. Megiddo and D. Modha, "ARC: A self-tuning, low overhead replacement cache," in Proceedings of the 2nd USENIX Conference on File and Storage Technologies, 2003, pp. 115–130.

[5] S. Huang et al., "Improving Flash-based Dish Cache with Lazy Adaptive Replacement," in Proc. Mass Storage Systems and Technologies(MSST),2013 IEEE 29th Symposium on, 2013 May 6-10,pp. 1-10

[6] G. Graefe, "The five-minute rule twenty years later, and how flash memory changes the rules," in Proceedings of the 3rd international workshop on Data management on new hardware. ACM, 2007, p. 6.

[7] T. Wong and J. Wilkes, "My cache or yours? Making storage more exclusive," in In Proceedings of the 2002 USENIX Annual Technical Conference, 2002.

[8] B. Gill, "On multi-level exclusive caching: offline optimality and why promotions are better than demotions," in Proceedings of the 6th USENIX Conference on File and Storage Technologies. USENIX Association, 2008, p. 4.

[9] UMass Trace Repository. [Online]. Available:
http://traces.cs.umass.edu/index.php/Storage/Storage

[10] Q. Wei, L. Zeng, J. Chen, C. Chen, "A Popularity-Aware Buffer Management to Improve Buffer Hit Ratio and Sequentiality for Solid State Drive," in Proc. Magnetics, IEEE Transaction on, 2013 June, pp 2786-2793.

[11] Lee, D., et. al., LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies, IEEE Transaction on Computers, vol. 50, no. 12, pp. 1352-1361, 2001.

[12] B. Debnath, S. Subramanya, D. Du, and D. J. Lilja, "Large Block CLOCK (LB-CLOCK): A Write Caching Algorithm for Solid StateDisks," in MASCOTS, 2009.

[13] Q. Wei; C. Chen; J.Yang "CBM: A cooperative buffer management for SSD", Mass Storage Systems and Technologies (MSST), 2014 30th Symposium on, On page(s): 1 – 12

[14] Y.Chai, Z. Du, X.Qin, D. Bader, "WEC: Improving Durability of SSD Cache Drives by Caching Write-Efficient Data," in Computers, IEEE Transactions, 2015 Feb , Vol.1, issue 99.

[15] Liang Shi, Juanhua Li and Chun Jason Xue, "ExLRU: A Unified Write Buffer Cache Management for Flash Memory", In proceedings of the ACM EMSOFT'11, 2011 Oct 9-14, pp 339-348.