IEICE TRANS. INF. & SYST., VOL.E97–D, NO.9 SEPTEMBER 2014

LETTER

Block Utilization-Aware Buffer Replacement Scheme for Mobile NAND Flash Storage[**]

Dong Hyun KANG[†], **Changwoo MIN**[†*], ***Nonmembers*****,** ***and*** **Young Ik EOM**[†a)], ***Member***

SUMMARY NAND flash storage devices, such as eMMCs and microSD cards, are now widely used in mobile devices. In this paper, we propose a novel buffer replacement scheme for mobile NAND flash storages. It efficiently improves write performance by evicting pages flash-friendly and maintains high cache hit ratios by managing pages in order of recency. Our experimental results show that the proposed scheme outperforms the best performing scheme in the recent literature, Sp.Clock, by 48%.

key words: *buffer replacement scheme, NAND flash storage, mobile devices*

1. Introduction

Mobile devices, such as smartphones and tablets, are becoming ever more popular. According to a Gartner forecast, the number of mobile applications targeting smartphones and tablets will surpass that of native PC applications by 2015 [1]. Moreover, mobile NAND flash storages, such as eMMCs and microSD cards, have now become the norm in the mobile devices to store applications and user data. This is because NAND flash storage has many characteristics that are suitable for mobile devices, such as small size, low power consumption, and shock resistance. However, a recent study shows that storage performance indeed affects the performance of commonly used applications in mobile devices [2]. Also, most I/O stacks in operating systems assume the performance characteristics of hard disk drives. Thus, optimizing I/O performance for mobile devices has been proposed in various OS layers. In particular, there has been extensive research in buffer replacement schemes for NAND flash storage, because the buffer replacement policy plays an important role in obtaining high performance: deciding which pages to keep in memory to improve cache hit ratio and which pages to evict to reduce I/O cost.

Several recent studies extend the traditional replacement schemes, such as LRU or Clock, to exploit the unique performance characteristics of NAND flash storage: asymmetric performance between reads and writes and performance disparity between sequential and random write patterns. CFLRU [3] exploits asymmetric latency between reads and writes. It prefers to evict clean pages rather than dirty pages to reduce write operations. However, CFLRU generates random writes and results in low performance, because it does not consider write patterns during eviction. To consider write patterns, FAB [4] selects an erase block including the largest number of dirty pages and it simultaneously evicts multiple pages in the block to reduce the garbage collection cost in NAND flash storage. (For brevity, we will use *block* as NAND flash erase block throughout this paper.) BPLRU [5] follows this direction more aggressively: for completely filled block-level eviction, it pads pages not in the block. Though the FAB and BPLRU can reduce write cost by considering write patterns, they have two limitations: first, they do not consider clean pages and thus cache hit ratio can deteriorate. Second, since the unit of eviction is not a page but a block, hot dirty pages can be evicted early and the early eviction can generate many unnecessary writes. Sp.Clock [6] is based on the Clock scheme and modifies it to keep the pages in order of sector number rather than recency. The Sp.Clock evicts pages in sector number order to produce the sorted write patterns and it leads to better performance compared to unsorted ones. However, the recency is limitedly reflected only by the reference bit, and the sorted writes exploit the write performance characteristics of NAND flash storage in a limited manner.

In this paper, we propose a novel buffer replacement scheme for mobile NAND flash storage to improve the write performance and maintain a high cache hit ratio simultaneously. We introduce unique write performance characteristics of NAND flash storage, which means that writing more pages per block leads to higher write performance by reducing fragmentation in the flash translation layer (FTL). We call this Maximizing Block Utilization (MBU) principle. We select the Clock scheme as a baseline for getting high cache hit ratio and extend it by exploiting the MBU principle for high write performance. The key idea of our scheme is to evict a page that belongs to the block with many dirty pages in it, i.e., high block utilization. Our experimental results on three real-world traces with two microSD cards show that our scheme outperforms the state-of-the-art scheme by up to 48%.

Manuscript received January 31, 2014.
Manuscript revised May 1, 2014.

[†]The authors are with College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea.
[*]Presently, with Samsung Electronics, Suwon, Korea.

[**]This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT&Future Planning (2010-0020730) and the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014 (H0301-14-1020)) supervised by the NIPA (National IT Industry Promotion Agency).

a) E-mail: yieom@skku.edu (Corresponding author)

DOI: 10.1587/transinf.2014EDL8021

2. Block Utilization-Aware Buffer Replacement Scheme

2.1 Maximizing Block Utilization (MBU) Principle

FTL maintains a mapping table between the logical address from the host and the physical address in NAND flash chips, and performs garbage collection that reclaims invalid pages and migrates valid pages to new locations. Many approaches have been proposed to consider write patterns because sequential write patterns reduce the garbage collection cost by mitigating internal fragmentation of the flash block [4]–[7]. A common approach to produce sequential write pattern is to write all pages in a block simultaneously [4], [5], [7] and to write pages in order of page number [6]. Previous studies show that block-sized random write performance approaches the maximum sequential write performance and page-sized random write performance approaches the minimum write performance [7]. We explore between the two extremes. Our hypothesis is that if we write $x\%$ pages in a block (i.e., the block utilization is $x\%$), the sustained write performance of such a write pattern will be a monotonic increasing function of x. We measured the throughput over different block utilizations on two commercial microSD cards, Patriot 16GB (10 Class) and Adata 16GB (6 Class), to verify our hypothesis. For measurement, we assume that the block size is 4 MB according to their product specifications. We performed 4 KB random writes according to different block utilizations, 25%, 50%, 75%, and 100%. In Fig. 1, we show the throughput of 1 GB writes over the four different block utilizations. As we expected, the write throughput increases as the block utilization is higher. It clearly shows that the maximizing block utilization (MBU) principle should be a key for optimizing write performance in NAND flash storage by minimizing fragmentation

2.2 Design of the Buffer Replacement Scheme

Our scheme is implemented based on Clock scheme to maintain high cache hit ratio and also exploits MBU principle to improve the write performance of NAND flash storage. In this section, we describe three main techniques used in our scheme. First, our scheme maintains a *reference count* instead of the reference bit of Clock scheme to reduce the number of expensive write operations and to keep cache hit ratio as high as possible. Second, our scheme divides dirty pages in the circular list of the Clock scheme into several lists based on their sector numbers and then sorts them in each list to produce sorted write patterns eventually. This is to optimize the write performance as mentioned in Sp.Clock [6]. Third, our scheme performs eviction at the granularity of page rather than at the granularity of block. By doing so, it can mitigate early eviction of hot pages and eliminate unnecessary write operations.

Figure 2 explains the details of our replacement scheme. It uses a *reference count* for each page and it is set whenever that page is accessed, similarly to Clock scheme (Line 12, 14 - 15). If a clean page is accessed, our scheme always sets its reference count to 1 (Line 12). Otherwise, its reference count is set in a predetermined manner according to the *block utilization* (Line 14 - 15). In our scheme, *block utilization* is calculated as the ratio of the number of dirty pages, whose reference count is zero, in a block (Line 14 - 15, 36). If a dirty page, which belongs to a block with low *block utilization* is accessed, our scheme sets its *reference count* to a large value for maintaining it longer in the cir-

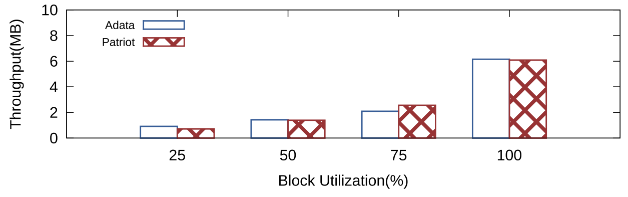

Fig. 1 Write throughputs for synthetic traces with different block utilizations on real microSD cards.

```
1  page *t−hand = null, *s−hand = null;
2  void Algorithm(page *p) {
3    if (p is not in the buffer cache) {
4      if (the buffer cache is full) {
5        page *v = choose_victim();
6        evict v;
7      }
8      if (t−hand is null) t−hand = p;
9      else insert p into the position of t−hand;
10     if (p.status is dirty) insert p into the sorted list of p.block;
11   }
12   if (p.status is clean) p.reference_count = 1;
13   else {
14     p.reference_count = ceil(
15       4 * p.block.zero / number of pages per block);
16   }
17 }
18 page *choose_victim() {
19   while (true) {
20     if (t−hand.reference_count is 0) {
21       if (t−hand.status is clean) return t−hand;
22       if (s−hand is null)
23         s−hand = the first page in the sorted list of t−hand.block;
24       while (true) {
25         page *s = s−hand;
26         if (s−hand is the last of the block)
27           s−hand = the first page
28              in the sorted list of t−hand.block;
29         else {
30           s−hand = the next page in the sorted list of the block;
31           return s;
32         }
33       }
34     }
35     t−hand.reference_count−−;
36     if (t−hand.reference_count is 0) p.block.zero++;
37     t−hand = the next page of t−hand in the circular list;
38   }
39 }
```

Fig. 2 The pseudo-code of the our replacement scheme.

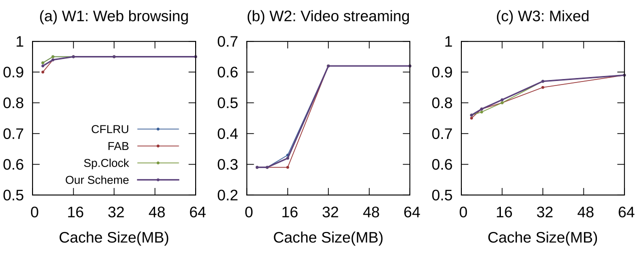

Fig. 3 Cache hit ratios.

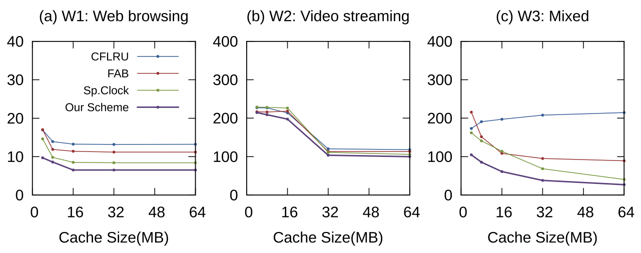

Fig. 4 Elapsed time of Patriot microSD card.

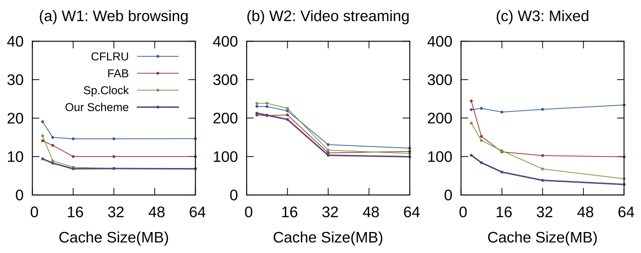

Fig. 5 Elapsed time of Adata microSD card.

cular list (For 25%, 50%, 75%, and 100% *block utilization*, our scheme linearly sets the reference count to 4, 3, 2, and 1, respectively, because the performance of linear approach is quite similar to that of non-linear approach). To shape evicted dirty pages to sequential write pattern, our scheme manages two hands, *t-hand* and *s-hand*, to select a victim. In order to select a victim page, our scheme firsts checks the reference count of each page using *t-hand*. If the reference count of the page pointed by *t-hand* is larger than zero, our scheme decreases its reference count by one and forwards *t-hand* to the next page (Line 35, 37). Otherwise, our scheme checks whether the page pointed by *t-hand* is clean or dirty. If the pointed page is clean, it is instantly evicted (Line 21). Otherwise, our scheme selects another dirty page using *s-hand*. *S-hand* scans dirty pages belonging to a block in order of sector number and then evicts the page pointed by *s-hand* instead of the page pointed by *t-hand* for flash-friendly write patterns (Line 29 - 31). After evicting the page pointed by *s-hand*, our scheme inserts a new page into the position of *t-hand* (Line 9). If *s-hand* is not set or points the last dirty page in the block, *s-hand* is set to the smallest sector number in the block which includes the page pointed to by *t-hand* (Line 22 - 23, 26 - 28).

3. Evaluation

We evaluated the performance of our scheme on a system with a dual-core Intel Atom CPU and 2 GB memory. Also, we used Linux Kernel 3.2.0 version and ext3 file system. We followed the testing methodology of the prior work [6] with three steps: (1) Obtaining before-cache trace which is a page cache access trace using it as an input to the cache simulator. (2) Cache simulation, which emulates the replacement schemes and generates evicted traces as a result of the simulation. (3) Replaying, which replays the evicted traces on real microSD cards with `O_DIRECT` option. We used the before-cache traces obtained from Sp.Clock [6] for comparison. These are composed of three real traces from an Android smartphone: W1 for web browsing, W2 for video streaming, and W3 for mixed applications execution. We implemented flash-aware replacement schemes such as CFLRU, FAB, and Sp.Clock to compare our scheme and performed the experiments on the microSD cards mentioned in Sect. 2.1. We set the block size to 4 MB and varied the cache size from 4 MB to 64 MB. Figure 3 shows cache hit ratios according to the cache size from 4MB to 64MB. In Fig. 3, our scheme shows comparable cache hit ratios to other three replacement schemes, even though pages in the block are evicted in order of sector number. This is because our scheme considers temporal locality effectively and mitigates early eviction. Figure 4 and Fig. 5 show the elapsed time of our scheme along with the comparison to other schemes. They clearly show that our replacement scheme improves performance of mobile NAND flash storage. Especially, it outperforms the state-of-the-art replacement scheme, Sp.Clock, by 33% for W1, 12% for W2, and 48% for W3. The reason is that our approach prefers to evict clean pages over dirty pages as well as it sequentially evicts dirty pages that belong to the block with high *block utilization* for minimizing fragmentation in FTL. As a result, our scheme leads to higher write performance with lower garbage collection overhead.

4. Conclusion

We proposed a block utilization-aware buffer replacement scheme for mobile devices. It improves write performance of mobile NAND flash storage by minimizing the fragmentation and it also maintains pages in recency order for high cache hit ratio. Our experimental results clearly show that the proposed scheme outperforms the state-of-the-art scheme by up to 48% on real microSD cards.

References

[1] "Gartner." http://www.gartner.com/newsroom/id/1862714
[2] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," Proc. USENIX FAST'12, pp.209–222, 2012.
[3] S.Y. Park, D. Jung, J.U. Kang, J.S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," Proc. ACM CASES'06, pp.234–241, 2006.
[4] H. Jo, J.U. Kang, S.Y. Park, J.S. Kim, and J. Lee, "FAB: Flash-aware buffer management policy for portable media players," IEEE Trans.

Consum. Electron., vol.52, no.2, pp.485–493, May 2006.
[5] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," Proc. USENIX FAST'08, pp.239–252, 2008.
[6] H. Kim, M. Ryu, and U. Ramachandran, "What is a good buffer cache replacement scheme for mobile flash storage?," Proc. ACM SIGMETRICS'12, pp.235–246, 2012.
[7] C. Min, K. Kim, H. Cho, S.W. Lee, and Y.I. Eom, "SFS: Random write considered harmful in solid state drives," Proc. USENIX FAST'12, pp.139–154, 2012.