TS-CLOCK: Temporal and Spatial Locality Aware Buffer Replacement Algorithm for NAND Flash Storages [*]

Dong Hyun Kang
Sungkyunkwan University
Suwon, Korea
kkangsu@skku.edu

Changwoo Min
Sungkyunkwan University
Suwon, Korea
multics69@skku.edu

Young Ik Eom
Sungkyunkwan University
Suwon, Korea
yieom@skku.edu

ABSTRACT

NAND flash storage is widely adopted in all classes of computing devices. However, random write performance and lifetime issues remain to be addressed. In this paper, we propose a novel buffer replacement algorithm called TS-CLOCK that effectively resolves the remaining problems. Our experimental results show that TS-CLOCK outperforms state-of-the-art algorithms in terms of performance and lifetime.

Categories and Subject Descriptors

D.4.2 [**Software**]: Operating Systems—*Storage Management*

Keywords

NAND flash storage, buffer replacement, locality

1. INTRODUCTION

In recent years, flash-aware buffer replacement algorithm has been actively studied because traditional algorithms, such as LRU, CLOCK, and Linux2Q, are not suitable for NAND flash storages. Most previous research focused on two performance characteristics of NAND flash storages. One is *asymmetric read/write cost* that the write operation is slower than the read operation [6], and the other is *performance variability in write patterns* that sequential writes are significantly faster than random writes [3, 1, 4, 5].

CFLRU [6] is a representative flash-aware buffer replacement algorithm that evicts clean pages over dirty pages to reduce the number of expensive write operations. However, CFLRU generates a large number of hidden writes inside NAND flash storage since it does not consider the write patterns of the evicted pages. In contrast, FAB [3], LB-CLOCK [1], and Sp.Clock [4] focuses on write patterns to reduce hidden writes. FAB and LB-CLOCK perform eviction at the granularity of a block rather than a page to generate sequential write patterns. However, they negatively impact cache hit ratio and performance due to the *early eviction problem* [7]. Among the existing flash-aware buffer replacement algorithms, Sp.Clock is most efficient. It manages page frames in order of logical sector number rather than recency order, for the sorted eviction scheme. However, Sp.Clock suffers from high garbage collection (GC) cost with wide I/O ranges such as server workloads, since the evicted patterns become random. The major drawback of existing algorithms is that they ignore lifetime of NAND flash storage. However, it is important to extend the lifetime because increasing flash capacity by storing additional bits per cell reduces the flash chip's lifetime by orders of magnitude [2].

[*]This work was supported by the IT R&D program of MKE/KEIT. [10041244, SmartTV 2.0 Software Platform]

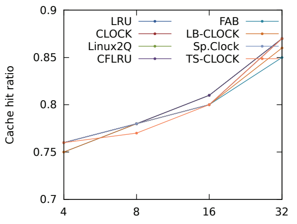
SIGMETRICS'14, June 16–20, 2014, Austin, Texas, USA.
ACM 978-1-4503-2789-3/14/06.
http://dx.doi.org/10.1145/2591971.2592028.

2. TS-CLOCK ALGORITHM

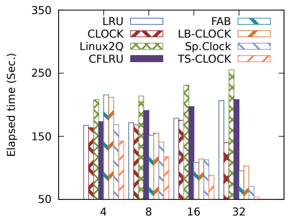
We propose *Temporal and Spatial locality aware CLOCK* (TS-CLOCK) replacement algorithm. The goal of our algorithm is to improve the performance and lifetime of NAND flash storage. To achieve it, TS-CLOCK extends traditional CLOCK algorithm for temporal locality, and exploits spatial locality by using the reference count and two hands.

Algorithm 1: Victim selection algorithm

```
Input: head of circular list t-hand, head of sorted list
       s-hand, new page np
while true do
    if t-hand.ref_count is zero then
        if t-hand.status is clean then
            evict t-hand;
        else
            if s-hand is null then
                s-hand = the first page in the sorted list of
                t-hand.block;
            while s-hand.ref_count is not zero do
                if s-hand is the last of the block then
                    s-hand = the first page in the sorted list
                    of t-hand.block;
                else
                    s-hand = the next page in the sorted list
                    of the block;
            evict s-hand;
        insert np into the position of t-hand;
        if np.status is dirty then
            insert np into the sorted list of np.block;
        break;
    else
        t-hand.ref_count = t-hand.ref_count - 1;
        t-hand = the next page of in the circular list;
```

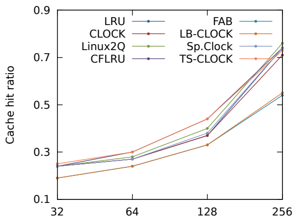


(a) Cache hit ratio

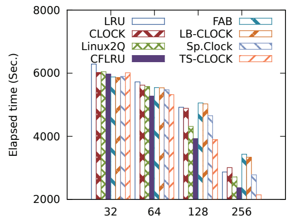


(b) Elapsed time

Figure 1: Cache hit ratio and elapsed time for mobile workload on the Patriot microSD card



(a) Cache hit ratio



(b) Elapsed time

Figure 2: Cache hit ratio and elapsed time for server workload on the Samsung SSD

Reference count. TS-CLOCK maintains a reference count per page to evict clean pages preferentially. A reference count of every clean page is always set to 1 when the page is accessed. On the other hand, that of every dirty page is differently set depending on *update likelihood* of the block that it belongs to. The update likelihood of a block is calculated as the ratio of dirty pages (i.e., the number of pages whose reference count is not 0 in a block / the number of pages per block). In our scheme, we set the reference count to 1, 2, 3, and 4 for 0-25%, 25-50%, 50-75%, and 75-100% update likelihoods, respectively.

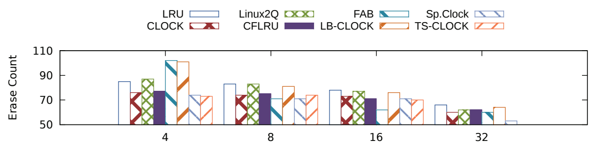
Two hands. TS-CLOCK manages *t-hand* and *s-hand* to shape evicted dirty pages into flash-friendly sequential pattern. Similarly to CLOCK, *t-hand* scans pages in the circular list and decrements its reference count by 1 for selecting a victim page. If a selected page whose reference count is 0 is clean, it is immediately evicted. Otherwise, *s-hand* scans dirty pages, which belong to the block, by their sector numbers until a page with reference count of 0 is found. Even though *s-hand* does not decrement the reference count of each page during scanning, it guarantees to find a victim page. This is because *s-hand* is set to the first page of the block, which includes the page pointed by *t-hand*, when *s-hand* reaches the last page in a block. If *s-hand* finds a page with reference count of 0 in a block, it is evicted. After evicting the page, TS-CLOCK inserts a new page into the position of the *t-hand* to grant at least one full life to the page. In this way, TS-CLOCK minimizes hidden writes by generating flash-friendly sequential write patterns. Algorithm 1 shows our victim selection algorithm.

3. EVALUATION

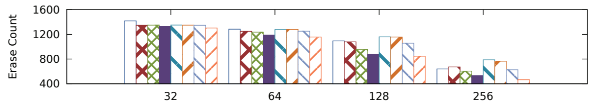
We evaluated TS-CLOCK and compared it with well-known replacement algorithms by trace-driven simulation and real NAND flash storage based experimentation. We also used two workloads: a mobile workload, which is a mixed application from Sp.Clock [4], and a server workload which is acquired from Dbench benchmark.

Experiment on the microSD card. We first compared the performance of eight replacement algorithms on the microSD card. Figure 1 shows the hit ratio and elapsed time of each algorithm while varying buffer cache size from 4MB to 32MB. As shown in Figure 1, TS-CLOCK significantly outperforms the other algorithms by up to 90%. The main reason is that TS-CLOCK reduces the hidden writes by exploiting spatial locality.

Experiment on the SSD. Figure 2 presents our experimental results on the triple-level cell (TLC) based SSD in terms of cache hit ratio and performance. We set cache



(a) FAST FTL with mobile workload



(b) page-level FTL with server workload

Figure 3: Comparison of erase count

sizes from 32MB to 256MB, which are more realistic ranges for high performance SSD. Figure 2 clearly shows that TS-CLOCK significantly outperforms the state-of-the-art algorithm, Sp.Clock, by up to 22.7%. To understand the results, we analyzed the amount of evicted I/O requests. As a result, we found that TS-CLOCK generates the smallest write requests due to clean-first eviction. Furthermore, TS-CLOCK minimizes the number of hidden writes by generating flash-friendly patterns.

Effect on Lifetime. We also measured the lifetime of NAND flash storage by implementing a trace-driven flash translation layer (FTL) simulator, which supports FAST FTL and page-level FTL. Figure 3 clearly shows that TS-CLOCK extends the lifetime by up to 29% compared to CFLRU using FAST FTL and by up to 40% compared to FAB using page-level FTL.

4. REFERENCES

[1] B. Debnath, S. Subramanya, D. Du, and D. J. Lilja. Large Block CLOCK (LB-CLOCK): A write caching algorithm for solid state disks. In *Proc. IEEE MASCOTS*, 2009.

[2] L. M. Grupp, J. D. Davis, and S. Swanson. The bleak future of NAND flash memory. In *Proc. USENIX FAST*, 2012.

[3] 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.*, 52(2):485–493, May 2006.

[4] H. Kim, M. Ryu, and U. Ramachandran. What is a good buffer cache replacement scheme for mobile flash storage? In *Proc. ACM SIGMETRICS*, 2012.

[5] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom. SFS: Random write considered harmful in solid state drives. In *Proc. USENIX FAST*, 2012.

[6] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee. CFLRU: A replacement algorithm for flash memory. In *Proc. ACM CASES*, 2006.

[7] G. Wu, X. He, and B. Eckart. An adaptive write buffer management scheme for flash-based SSDs. *ACM Trans. Storage*, 8(1):1, Feb. 2012.