ExLRU: A Unified Write Buffer Cache Management for Flash Memory

Liang Shi1,2, Jianhua Li1,2, Chun Jason Xue1, Chengmo Yang3 and Xuehai Zhou2

1Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong and USTC-CityU Joint Advanced Research Centre, Suzhou, P.R.China
2Department of Computer Science, University of Science and Technology of China, Hefei, China and USTC-CityU Joint Advanced Research Centre, Suzhou, P.R.China
3Department of Electrical & Computer Engineering, University of Delaware

{shili0704, jianhual}@mail.ustc.edu.cn, jasonxue@cityu.edu.hk, chengmo@udel.edu, xhzhou@ustc.edu.cn

ABSTRACT

NAND flash memory has been widely adopted in embedded systems as secondary storage. Yet the further development of flash memory strongly hinges on the tackling of its inherent implausible characteristics, including read and write speed asymmetry, inability of in-place update, and performance harmful erase operations. While Write Buffer Cache (WBC) has been proposed to enhance the performance of write operations, the development of a unified WBC management scheme that is effective for diverse types of access patterns is still a challenging task. In this paper, a novel WBC management scheme named Expectation-based LRU (ExLRU) is proposed to improve the performance of write operations while at the same time reducing the number of erase operations on flash memory. ExLRU accurately maintains access history information in WBC, based on which a new cost model is constructed to select the data with minimum write cost to be written to flash memory. An efficient ExLRU implementation with negligible hardware overhead is further developed. Simulation results show that ExLRU outperforms state-of-art WBC management schemes under various workloads.

Categories and Subject Descriptors
D.4.2 [Operating System]: Storage Management—Secondary storage; B.3.3 [Performance Analysis and Design Aids]: Simulation

General Terms
Design, Performance, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EMSOFT ’11, October 9-14, 2011, Taipei, Taiwan
Copyright 2011 ACM 978-1-4503-0714-7/11/10 ...$10.00.

Keywords
Write Buffer Cache, Flash Memory, ExLRU

1. INTRODUCTION

NAND flash memory is widely adopted in embedded systems given its well-identified advantages [1, 3] such as low power consumption, lightweight form factor and shock resistance. However, flash memory also has several implausible features, including asymmetric speeds of read and write, inability of in-place update and limited endurance. The speed of a write operation on flash memory is about 8 times slower than that of a read operation, thus making write operations the bottleneck that dominates the performance of flash memory. In addition, flash memory is unable to conduct in-place update; data update in flash memory always needs to be coupled with a prior erase operation [7], which is not only time consuming but also lifetime harmful. The inability of performing in-place update further weakens the write performance of flash memory. Due to these characteristics, it is necessary to improve the write performance and reduce the number of erase operations in future deployment of flash memory in various systems. In this paper, we propose an approach to reduce the number of write and erase operations on flash memory.

Researchers have studied the use of Write Buffer Caches (WBCs) [4, 9, 11, 12, 17] that exploit write access patterns as well as characteristics of flash memory to reduce the number of write and erase operations on flash memory. Two representative WBC designs are FAB [9] and BPLRU [12], with the former aiming at improving sequential write performance of flash memory (especially for Portable Media Players applications) and the latter aiming at improving random write performance. Unfortunately, these two WBC designs are not workload adaptive; FAB delivers very limited performance advantage for workloads dominated by random write accesses, while BPLRU shows very limited improvement when applied to workloads dominated by sequential write accesses.

To overcome the aforementioned limitations of existing WBC designs, the work presented in this paper focuses on proposing a WBC management scheme for flash memory, capable of adaptively delivering appreciable performance im-
provement for diverse types of write patterns. In particular, we propose a novel Expectation-based LRU (ExLRU) scheme that accurately maintains the information of write accesses in WBC with negligible hardware overhead. Based on such accurate access information, a unified write cost model is constructed with considerations of both the characteristics of flash memory and the types of write patterns. Using this cost model, WBC is organized in a way that upon a block eviction, the block with the minimum write cost can be selected to be written back to flash memory earlier. An efficient implementation of ExLRU is further proposed to identify the most suitable eviction targets at negligible cost. Simulation results confirm that ExLRU consistently delivers a sizable reduction of the number of write and erase operations on different workloads, thus outperforming the state-of-art WBC management schemes.

The rest of the paper is organized as follows. Section 2 presents a set of background knowledge of flash memory and related work. Section 3 describes the fundamental principles of the ExLRU scheme, while Section 4 provides an efficient implementation of ExLRU. Experiments are presented in Section 5. Finally, Section 6 summarizes the paper.

2. BACKGROUND AND RELATED WORK

In this section, we briefly introduce the characteristics of flash memory as well as the previous work in improving write performance and reducing the number of erase operations on flash memory.

2.1 Characteristics of Flash Memory

In general, there are two types of flash memories: NOR flash memory and NAND flash memory. NOR flash memory, as it delivers high access speed, is popular for code storage. In comparison, NAND flash memory offers higher integration density and hence is widely used for data storage. When used for data storage, flash memory has a number of advantages over traditional storage media such as hard disk, including high access bandwidth, random accessibility and shock resistance.

There are three types of operations in flash memory: read, write and erase. Read and write operations are performed in the granularity of pages, while erase operations are performed in the granularity of blocks, with a block composed of multiple pages. The typical characteristics of these three types of operations are listed in Table 1 [1]. As can be seen, write operations are slower than read operations by 8 times.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Access Time</th>
<th>Operation Unit</th>
<th>Typical Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>25μs</td>
<td>Page</td>
<td>2KB</td>
</tr>
<tr>
<td>Write</td>
<td>200μs</td>
<td>Page</td>
<td>2KB</td>
</tr>
<tr>
<td>Erase</td>
<td>2ms</td>
<td>Block</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Table 1: The Parameters and Operations of Flash Memory

Given such peculiarities of flash memory, various types of work have been proposed to reduce the number of write and erase operations [18, 19], including Flash Translation Layers (FTLs) and write buffer caches (WBCs). In this work, we propose a new WBC management scheme for flash memory using a hybrid FTL. The differences among the various FTLs are briefly introduced in the subsequent section.

2.2 Flash Translation Layer

FTL has been proposed to improve write performance and reduce the number of erase operations in flash memory. FTL provides a mimic block storage interface that hides the internal complexities of flash memory to operating systems (OSs), thus enabling the OS to read/write flash memory in the same way as reading/writing the hard disk. The various FTLs proposed in prior work can be classified into three types according to the mapping granularity: block, page and hybrid approaches. Block-level FTL maintains a coarse-grained, page-level mapping between logical and physical pages, while within each block a logical page can be only mapped to a single physical page. This approach imposes very limited storage overhead for mapping tables yet delivers poor write performance; an update to a dirty page would require the entire block to be erased. In comparison, page-level FTL [2, 6] maintains a fine-grained, page-level mapping wherein a logical page can be mapped to any physical page within the flash memory. Such a flexible mapping strategy delivers good write performance when a page needs to be frequently rewritten, yet hinges on the existence of huge mapping tables.

To overcome the limitations of pure block-level and pure page-level FTLs, current work focuses on the development of hybrid FTLs [13, 15, 20, 10, 14] to combine the advantages of these two types of approaches. Hybrid FTL partitions blocks of flash memory into data blocks and log blocks, and respectively maintains block-level and page-level mappings for these two types of blocks. A number of different FTL organizations have been proposed, including BAST [13] and FAST [15]. The former attaches one log block to one data block, while the latter employs an on-demand allocation strategy for log blocks. If log blocks are fully utilized, one log block can be merged with its attached data blocks, using one of the three merge operations proposed in [15], namely, full merge, partial merge, or switch merge. Lee et al. [14] further propose to exploit the locality of write accesses in the design of FTL, and show that sequential write patterns always induce switch merge operations while random write patterns tend to induce full merge operations.

Taking into consideration the correlation between write patterns and merge operations in hybrid FTL, the goal of the proposed ExLRU is to organize the WBC in a way that produces suitable write patterns for FTL to increase the possibility of switch merge operations and reduce the possibility of full merge operations.

2.3 Write Buffer Cache Management

Write Buffer Caches (WBCs) have been adopted in flash memory to reduce the number of write and erase operations. A WBC is typically organized in blocks composed of pages of the same logical block, with the size of each page equal to the size of flash memory page. When the WBC receives a write request, it searches for the corresponding logical block. If the block exists in the WBC, the page is further indexed within the block. If the page exists, it is a write hit and the data will be written in the page. If the page does not exist, a free page needs to be allocated in the WBC to accommodate the incoming data and the page is added to the corresponding block. In case the WBC is full, a block needs to be evicted to flash memory.

Various approaches have been proposed to manage the WBC, with FAB [9], BPLRU [12], and CLC [11] being the
most representative strategies. FAB [9] aims at improving the performance of sequential write patterns on flash memory. Blocks in FAB are sorted according to their sizes, i.e., the number of pages in each block. Blocks of the same size are ordered according to the least-recently-used (LRU) information. This organization is depicted in Figure 1(a), with $B_x$ denoting the blocks and $P_i$ the pages. Upon a block eviction request, the algorithm selects, among the group of largest blocks, the block in the LRU position to be written to flash memory. Accordingly, in Figure 1(a) the block $B_2$ will be evicted to the flash memory. Using this strategy, FAB is able to significantly improve the performance of sequential write operations. Yet for workloads dominated by random write patterns, FAB delivers very limited performance improvement as it falls short of exploiting the temporal locality associated with random write patterns.

BPLRU [12] aims at improving the performance of random write patterns on flash memory. Similar to FAB, BPLRU organizes the WBC in blocks of logical pages. However, regardless of their sizes, all the blocks in BPLRU are organized in the LRU order: a write to a page will result in the corresponding block being moved to the most-recently-used (MRU) position. This organization is depicted in Figure 1(b). Upon an eviction request, $B_3$, as it is in the LRU position, will be evicted to flash memory. Meanwhile, BPLRU additionally proposes to pad the evicting block to its current mapping in flash memory, and then writing the evicting block together with these valid pages to a free block in flash memory. Using this approach, BPLRU is able to significantly improve the performance of random writes operations on flash memory.

CLC [11] is a hybrid approach that aims at exploiting the advantages of both FAB and BPLRU, through partitioning the WBC into two disjoint regions that exclusively employ either FAB or BPLRU.

### 3. Motivation of ExLRU

As flash memory displays asymmetric operation granularity for write and erase operations, the latest work [9, 12, 11] integrates pages from the same logical block into a single block and employs block-level temporal/spatial locality management for the WBC. By organizing pages into blocks, pages in the same block can be written to the same physical block of flash memory at the same time, thus effectively reducing the number of write operations.

While block-based organizations can effectively improve write performance and reduce the number of erase operations on flash memory, they fall short of accurately capturing the access information of individual pages. In the ideal case, upon an eviction request, the block in the WBC that will be accessed least frequently in the future should be evicted to flash memory. However, previous WBC management schemes maintain the locality information at the block level, while updating the LRU sequence upon every page access. The lack of page-level access information tends to result in inappropriate eviction decisions, listed as follows:

- **Slow retirement of large cold blocks:** In the case of BPLRU, a block can be evicted only when it is retired to the LRU position, while a block is moved to the MRU position whenever a page in the block is accessed. As a result, a block that contains a large number of pages will be moved to the MRU position more frequently and hence will be retired slowly, even if each individual page in the block may be cold.

- **Early eviction of small hot blocks:** In contrast to the large blocks, a small block may be evicted too early even if its individual pages are hot. In order for a block to be kept in the WBC, its page needs to be accessed more frequently. For example, given a large block containing 32 pages each of which is written once, a small block with a single page needs to be updated for at least 32 times in order to compete with the large block for the MRU position.

- **Cold page retention in heat-imbalanced blocks:** A heat-asymmetric block is defined as a block that contains a few hot pages together with a large number of cold pages. Whenever a hot page is written, the entire block will be moved to the MRU position, resulting in the cold pages being retained in the WBC for a long time.

The issues listed above directly derive from the lack of page-level access information in the block-based WBC organizations. In light of these observations, we propose in this work a unified model to evaluate the eviction cost of various types of blocks in the WBC. This model takes into consideration both the page-level access information and the characteristics of flash memory, thus enabling the development of a WBC management approach capable of accurately differentiating the eviction cost of various types of blocks.
### 3.2 Cost Model of ExLRU

The proposed ExLRU scheme manages the WBC for flash memory based storage systems. Same as previous work [9, 12, 11], the WBC is organized in blocks of logical pages, with the size of each page equal to the size of flash memory page. This organization is shown in Figure 2, wherein $B_x$ indicates a block with logical block number $x$ and $P_i$ indicates a page with page number $i$.

![Figure 2: Organization of the proposed ExLRU](image)

Figure 2: Organization of the proposed ExLRU: each page is extended to contain age and count information and each block is extended to contain Pcnt.

The most significant advantage of the proposed ExLRU scheme over the state-of-art WBC management schemes is its ability in utilizing the accurately maintained page-level access information to determine the block evictions. To do so, a number of extra fields have been added to each page in the WBC. The extended fields are described in the follow parts.

The first piece of access information is the age of pages, defined as the time interval between the current time and the first access time of the page. Assuming that the first access time of page $i$ is $t_i$ and the current time is $t_c$, the age of page $i$ is defined as $AfP_i = t_c - t_i$. In this work, we use the number of distinct operations between its earliest access time and the current time to represent the age of a page. To record the age of a page, the request number of the first write access to the page needs to be recorded. As shown in Figure 2, in addition to the virtual page number (VPN) and the data, each page also includes a Request_number field that records the request number of the first access to the page. Age of a page expires when the page is flushed to flash memory.

The second piece of access information is the access count of pages, defined as the number of write hits on the page when it is in the WBC. Every page maintains a Hit_count field to record its access count, as shown in Figure 2. Hit_count is set to be 1 when the page is allocated, incremented by 1 upon every write hit on the page, and cleared when the page is flushed to flash memory.

Utilizing the aforementioned page age and access count information, we model the hotness (i.e., the access frequency) of a page, represented as the Average Frequency of Pages (AFP).

**Definition 1.** $AfP$ of page $i$ is defined as:

$$ AfP(i) = \frac{n_i}{r_c - r_i}, $$

with $i$ denoting the page number, $n_i$ the hit count, $r_c$ the current request number, and $r_i$ the request number of the first access to page $i$. If $r_c$ is equal to $r_i$, indicating that page $i$ is just brought into the WBC, $AfP(i)$ is set to 1. $AfP(i)$ effectively represents the hotness of a page: a larger $AfP(i)$ implies that page $i$ is hotter, as it has a shorter access interval on average.

Based on the averaged frequency of pages, the Averaged Frequency of a Block (AFB) can be calculated.

**Definition 2.** $AfB$ of block $x$ is defined as:

$$ AFB(x) = \sum_{i=1}^{N} AfP(i) \cdot \frac{1}{N} \sum_{i=1}^{N} \frac{n_i}{r_c - r_i}; $$

with $x$ denoting the logical block number and $N$ the number of pages in the block. To record $N$, each block in the WBC is extended to contain a $Pcnt$ field, as shown in Figure 2. Here, $LBN$ is the logical block number used for block indexing.

Compared to the BPLRU scheme, $AfB(x)$ effectively captures the average access frequency of all the pages in block $x$. As a concrete example, Figure 3 shows a WBC composed of 7 pages that are divided to 3 blocks. If BPLRU is applied, $B_1$ will be evicted from the WBC as it currently is the least-recently-used block. However, it turns out that evicting $B_0$ would be a superior decision, as the single page in $B_1$ has been accessed 4 times while each of the four pages in $B_0$ are accessed only one time. In other words, $B_1$ is “hotter” than $B_0$ and hence should not be evicted. Such information can be accurately captured in the proposed $AfB(x)$ measurement. In particular, assuming that $r_i = 11$, we can obtain $AfB(0) = 0.213$, $AfB(1) = 0.667$ and $AfB(2) = 1.063$ for the three blocks in Figure 3. These values clearly indicate that $B_1$ is hotter than $B_0$, thus avoiding the eviction of a small hot block in the proposed ExLRU.

![Figure 3: WBC is composed of 7 pages that are divided into three blocks: B0 contains 4 pages, B1 one page, and B2 two pages.](image)

While the proposed $AfB(x)$ measurement effectively captures the “hotness” of a block through calculating the average hotness of all the pages within the block, it does not take into consideration the block size. To give large blocks a higher priority to be evicted to flash memory, we additionally divide this measurement by the block size and define a unified cost model as follows:

**Definition 3.** Unified eviction cost of block $x$ is defined as:

$$ UC(x) = \frac{AFB(x)}{N}; $$

Using the unified cost model, blocks with various access patterns can be effectively differentiated and prioritized for eviction purposes. In general, blocks in the WBC can be classified into five types according to their sizes and hot/cold degrees: large cold blocks, small cold blocks, large hot blocks,
small hot blocks, and cold/hot interleaving blocks. The descriptions of these 5 types of blocks and their corresponding eviction priorities are summarized in Table 2.

Table 2: The 5 types of blocks in WBC based on the size and hot/cold of blocks

<table>
<thead>
<tr>
<th>Block Types</th>
<th>Descriptions</th>
<th>Eviction priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large cold blocks</td>
<td>Pages in a block are sequentially written for one or two times. This type of write pattern is typical in large file downloading and data copying.</td>
<td>Highest eviction priority.</td>
</tr>
<tr>
<td>Small cold blocks</td>
<td>In a single block, only a few pages are written for one or two times. This type of write pattern is typical in web browsing and small file copying.</td>
<td>Preferable for eviction.</td>
</tr>
<tr>
<td>Large hot blocks</td>
<td>The block size is large, and all pages in a block are frequently updated. This type of write pattern is rare.</td>
<td>Preferably to be kept in the WBC until they become cold.</td>
</tr>
<tr>
<td>Small hot blocks</td>
<td>In a single block, only a few pages are accessed and frequently updated. This type of write pattern is typical in file system metadata accessing and in database systems.</td>
<td>Preferably to be kept in the WBC until they become cold.</td>
</tr>
<tr>
<td>Cold/hot interleaving blocks</td>
<td>Accesses to the pages in a single block are imbalanced. This type of write pattern comes from the cold and hot blocks whose addresses do not align to the block addresses in flash memory block.</td>
<td>Eviction priority depends on the ratio of hot v.s. cold pages and block size.</td>
</tr>
</tbody>
</table>

The proposed unified cost model can effectively prioritize these blocks for eviction. Definition 3 clearly confirms that 1) for blocks of the same size, colder blocks will be given a larger chance to be evicted earlier, and 2) for blocks of the same hotness, larger blocks will be given a larger chance to be evicted earlier. As a result, the three inappropriate situations identified in Section 3.1 can be avoided using this model. For the three blocks in Figure 3, their eviction costs are $UC(0) = 0.053$, $UC(1) = 0.667$ and $UC(2) = 0.531$. Upon an eviction request, $B_0$, the block with the smallest eviction cost, will be selected. Compared to BPLRU that evicts $B_1$, the eviction of $B_0$ is a superior decision since $B_0$ is both larger and colder than $B_1$. Additionally, under this unified cost model, $B_2$, although it is the most frequently used block, has a smaller eviction cost than $B_0$. This is because $B_2$ has a cold page, thus making it a hot/cold interleaving block. For such blocks, their $UC$s are decided based on the ratio of hot and cold pages in them. When the hot pages become cold, these blocks will be evicted quickly.

3.3 Management of ExLRU

Overall, the organization of WBC is shown in Figure 2. The page-level access information is recorded in a structure named `frame`, with each entry in the frame extended to include `{Hit_count, Request_number}`. `Hit_count` records the number of hits to the corresponding page (before eviction), while `Request_number` records the request number of the first access to the page. In addition, at the block level, `Pcnt` records the number of cached pages within the block. It needs to be noted that the hardware overhead for recording these extra values is very low. For a typical WBC design with a block size of 64-page and a page size of 2KB, we use 10 bits for recording `Hit_count`, 20 bits for `Request_number`, and 6 bits for `Pcnt`. The introduced hardware overhead is 0.18% of the total size of the WBC.

With the help of these fields, the proposed WBC management and block eviction scheme is shown in Algorithm 1. Upon a page hit, the corresponding `Hit_count` field is incremented by 1 (lines 2-3). In contrast, upon a page miss, a free page should be allocated to accommodate the incoming data (line 19). In case the WBC is full, a victim block in the WBC is selected for eviction (lines 5-18). As can be seen, ExLRU chooses the block with the smallest $UC$, by scanning all the blocks in the WBC (lines 7-16). Finally, the missing page is added to WBC, while the `Request_number`, the `Hit_count`, and the `Pcnt` are updated accordingly (lines 20-23).

Algorithm 1 ExLRU: WBC Management Scheme

Input:

$P$ - The page under written;
$LBN$ - The logical block number;
$VPN$ - The virtual page number;
$r$ - The current request number;

1: Index $LBN$ and $VPN$;
2: if page hit then
3: $P \rightarrow Hit_count + 1$;
4: else
5: if WBC is Full then
6: $UC_{\min} = \infty$;
7: for each block $B_i$ in the WBC do
8: $n_i = P_i \rightarrow Hit_count$;
9: $r_i = P_i \rightarrow Request_number$;
10: $N = B_z \rightarrow Pcnt$;
11: $UC(x) = (\sum_{i=1}^{N} \frac{n_i}{r_i})/N^2$;
12: if $UC_{\min} > UC(x)$ then
13: Victim_block = $B_i$;
14: $UC_{\min} = UC(x)$;
15: end if
16: end for
17: Evict Victim_block;
18: end if
19: Allocate a new page to accommodate $P$;
20: Add $P$ to block $B_{LBN}$;
21: $P \rightarrow Request_number = r$;
22: $P \rightarrow Hit_count = 1$;
23: $B_{LBN} \rightarrow Pcnt + 1$;
24: end if
25: Move $B_{LBN}$ to MRU Position;

Algorithm 1 presents an ideal case wherein the block with the lowest $UC$ is selected for eviction. As the scanning process (lines 7-16) computes all the pages in each block of the WBC, the time complexity of the algorithm is $O(n)$, assuming that the WBC contains a total number of $n$ pages. In many cases, even though there may exist plenty of idle time between requests, this high runtime computation complexity is unacceptable. To overcome this limitation, we furthermore propose an efficient implementation of ExLRU within tightly bounded approximation and computation overhead, as discussed in the next section.

4. EFFICIENT EXLRU IMPLEMENTATION

The high computation overhead of ExLRU stems from the fact that in order to identify the block with the lowest eviction cost, the entire WBC needs to be scanned. To reduce
such overhead, we propose an approximate ExLRU scheme that pre-identifies, during the idle time between two write requests, a number of WBC blocks whose UC values are low enough. In this way, upon an eviction request, the victim block can be quickly selected among this pre-identified group of blocks. In the remaining parts of this section, we first provide an analysis regarding the idle time between two consecutive write requests to the WBC, and then discuss how such idle time can be exploited to pre-identify blocks with low UCs.

Given the ever increasing memory sizes, the I/O requests to secondary storage are performed in a low frequency [5]. As a concrete example, Figure 4 shows the distributions of idle time between flash-access requests for two applications, with X-axis showing the request indices and Y-axis the time interval between two consecutive write requests. As can be seen, for the workload Financial, for 60% of the case, the idle time between two consecutive write requests is larger than 2ms. For PC trace, it is 37.6%.

![Figure 4: Idle Time Distribution between 2 Consecutive Flash-Access Requests](image)

To mitigate the computation overhead in the block eviction process, we propose to partition the WBC into two regions: a Work Region (WR) and an Eviction Region (ER). During the I/O idle time, the ExLRU scheme can be activated to scan WR to find the blocks whose UC values are smaller than a threshold $T_{UC}$. These blocks are moved to ER. During the eviction process, a block in ER is to be selected for eviction. This WBC organization is shown in Figure 5, with both WR and ER organized in LRU lists. Clearly, the sizes of WR and ER depend on the threshold value $T_{UC}$; a larger $T_{UC}$ would cause more blocks to be added to ER.

![Figure 5: The Partitioned WBC with $T_{UC}$ as Threshold. The WBC has two regions: Work Region and Eviction Region.](image)

The two-region organization of the WBC enables a decomposition of the WBC scanning and victim block selection processes, as detailed below.

**WBC Scanning:** Upon finishing serving a write request in the WBC, if ER has very few or no blocks (i.e., the number of blocks in ER is smaller than a threshold $N_{min}$), WR will be scanned to select blocks with $UC(x) \leq T_{UC}$. Typically there will be plenty of idle time between two consecutive write requests, implying that the scan process usually can find enough blocks and move them to ER before the next write request arrives at the WBC. Assuming that the idle time between requests $i$ and $i + 1$ is $t_i$, there are $n$ blocks in WR, and the average time for scanning a block is $t$. If $n \times t_i \leq t$, the entire WR can be scanned during the idle time. In contrast, if $n \times t_i > t$, the entire scanning process cannot be finished before the subsequent write request. To ensure that write requests are served as early as possible, the scanning process can be pre-empted and then re-activated once the write request is done.

$T_{UC}$ is a critical parameter in the block scanning process. A large $T_{UC}$ would increase the chances of finding blocks to be moved to ER during the scanning process while degrading the eviction performance. On the other hand, a small $T_{UC}$ guarantees the quality of eviction decisions, yet increases the number of scanning processes needed, as each scan identifies fewer blocks for ER. To select the most appropriate $T_{UC}$, we have experimented both a static $T_{UC}$ and a dynamic $T_{UC}$. The static $T_{UC}$ is simple for implementation, yet may cause a situation that the scanning process cannot find a block whose eviction cost is lower than $T_{UC}$. In this case, the LRU block in WR will be selected for block eviction. In comparison, the dynamic approach adaptively adjusts the $T_{UC}$ value; if there are too many blocks selected in the scanning process, $T_{UC}$ will be reduced and vice versa.

In order to avoid the inappropriate setting of $T_{UC}$ that may make the scanning process select too many blocks to ER, a threshold $T_{scan}$ is also set to limit the maximal number of blocks that can be selected in one scanning process. The comprehensive studies of $N_{min}$, $T_{UC}$ and $T_{scan}$ will be shown in the next section.

**Eviction Region Management:** The scanning process ensures that all the blocks in ER have a low eviction cost. To simplify the block selection during the eviction process, blocks in ER are organized in the LRU order. When a block is added to ER, it is placed at the MRU position. Upon a block eviction request, the block in the LRU position of ER will be selected to be evicted to flash memory.

As a write request always varies UC values, its impact to the blocks in ER needs to be examined, as detailed below.

- If a write request does not hit any blocks in ER, the UC values of these blocks will decrease, thus consistently fulfilling the UC threshold requirement. As a result, no block movement is needed in ER.
- If a write request hits a page in ER, the UC value of the corresponding block will be recomputed to check whether the block should be moved back to WR.
- If a write request hits a block in ER yet the required page is missing, a new page will be added to the block. As the block grows larger, its UC value decreases, thus fulfilling the UC threshold requirement. This block will be moved to the MRU position of ER, thus allowing it to stay in the WBC for a while to capture future write hits, if any.

The implementation of this efficient two-region WBC management scheme is shown in Algorithm 2. Upon a write miss,
if the WBC is full, the block at the LRU position of \( ER \) is selected to be evicted to flash memory (lines 5-8). For the accessed block, if it is in \( WR \) or it is in \( ER \) and its updated \( UC \) value is larger than \( T_{UC} \), the block is added to MRU position of \( WR \) (lines 15-16). Otherwise, the block stays in \( ER \) (line 18). Note that a scanning process is activated after the current write request if the number of blocks in \( ER \) is smaller than a threshold \( N_{min} \) (line 20). The scanning process selects the blocks with \( UC(x) \leq T_{UC} \) and adds these blocks to \( ER \) (lines 21-30). During the scanning process, if a write request is sent to the WBC, the scanning process is preempted (lines 32-34). As the scanning process is only performed during the idle time and it can be preempted by the subsequent write request, it would not affect the system performance. In addition, if a block is accessed and the number of pages in the block is 64 (block size of flash memory), the block is directly moved to the LRU position of \( ER \) so that it can be evicted as soon as possible.

**Algorithm 2 Efficient Implementation of ExLRU**

**Input:**
- \( P \) - The page under written;
- \( LBN \) - The logical block number;
- \( VPN \) - The virtual page number;
- \( r_e \) - The current request number;
- \( T_{scan} \) - The maximal number of blocks in one scan;
- Index \( LBN \) and \( VPN \);

1. if page hit then
   - \( P \rightarrow \text{Hit} \_\text{count} + 1 \);
2. else
   - if WBC is Full then
     - \( \text{Victim block} \leftarrow \text{the LRU Block of ER} \);
     - Evict \( \text{Victim block} \);
   - end if
   - Allocate a new page to accommodate \( P \);
   - Add \( P \) to block \( B_{LRN} \); (*assume \( P \in \text{Block} B_{LRN} \))
   - \( P \rightarrow \text{Request number} = r_e \);
   - \( P \rightarrow \text{Hit count} = 1 \);
   - \( B_{LRN} \rightarrow \text{Pen} \_t + 1 \);
3. end if
4. if \( (B_{LRN} \in \text{WR}) \) \( \land \) \( (B_{LRN} \in \text{ER} \land \text{UC(LBN)} > T_{UC}) \) then
   - \( \text{Move} \_\text{LRU} \rightarrow \text{MRU Position of WR} \);
   - else
     - \( \text{Move} \_\text{LRU} \rightarrow \text{MRU Position of ER} \);
   - end if
6. if \( | ER | \leq N_{min} \) then
   - \( B_s \leftarrow \text{the LRU Block of WR} \);
21. while \( B_s \rightarrow \text{NULL} \land \text{scan} < T_{scan} \) do
23. \( n_t = P \rightarrow \text{Hit count} \);
26. \( UC(x) = (\sum_{i=1}^{N} \frac{n_t}{r_e})/N^2 \);
27. end if
28. \( \text{Add} B_s \rightarrow \text{MRU Position of ER} \);
29. \( \text{scan} + 1 \);
30. end if
31. \( B_s \leftarrow \text{the next Block of WR from LRU} \);
32. if Write request then
33. Execute the write request;
34. end if
35. end while
36. end if

5. EXPERIMENT AND ANALYSIS

In this section, we compare the proposed ExLRU against the currently best-known WBC management schemes. Specifically, we show the comparison of the number and the average size of WBC blocks evicted to flash memory, as well as the reduction in the number of write and erase operations on flash memory. Before presenting the experimental results, we first introduce our experimental methodology.

### 5.1 Experimental Methodology

We adopt a trace-driven simulator for experiments. The simulator contains a write buffer cache, an FTL and a storage model with NAND flash memory. The size of NAND flash memory used in the experiments is 8GB, with a page size of 2KB and a block size of 64 pages. The parameters of flash memory are listed in Table 1. We adopt a widely researched hybrid FTL, the fully-associative sector translation layer (FAST) [15], as the FTL scheme of flash memory in our experiments. FAST performance is known to be better than the traditional log block scheme [13], as it maps data blocks with log blocks in fully associative style, thus maximizing the utilization of log blocks and improving random write performance significantly. In our experiments, the number of log blocks is set to be 8. The size of a sector is 512 bytes and a page can hold 4 sectors.

The workload we use in this study consists of one TPC-C benchmark from [16] and a PC trace [8] collected from a notebook. The TPC-C benchmark is widely used in prior work to model disk behaviors on enterprise level applications, such as database servers, whose write requests are mainly random requests with small data request size. The PC trace includes receiving and sending emails, web surfing, editing documents, listening music and so on.

#### 5.2 Experimental Results

In this section, we first present the reduction in the number of blocks evicted to flash memory and the increase in the average size of evicted blocks. Subsequently, the effectiveness of ExLRU in reducing the number of write and erase operations is examined. Finally, a comprehensive study of parameter selection is performed.

##### 5.2.1 Average Size and Number of Evicted Blocks

The number and the average size of evicted blocks are evaluated to show the effectiveness of ExLRU. Increasing the size of evicted blocks is beneficial for not only reducing the number of erase operations on flash memory, but also improving the performance of FTL. This is because hybrid FTLs, such as FAST, are sensitive to the sizes of evicted blocks. For small blocks, FAST would have a high degree of association with log blocks, thus in turn inducing the time consuming full merge operations in FAST. In order to show the effectiveness of ExLRU, we vary the size of the WBC from 1MB to 64MB in our experiments.

Figures 6 and 7 respectively show the reduction in the number of evicted blocks and the increase in the average size of evicted blocks. As can be seen, the proposed ExLRU scheme is compared against FAB, BPLRU and CLC. Furthermore, as \( T_{UC} \) is a critical parameter in the block scanning process, the proposed ExLRU is evaluated using both a static \( T_{UC} \) (ExLRU\_S) and a dynamic \( T_{UC} \) (ExLRU\_D). \( T_{UC} \) is set to be \( 0.1 \times 10^{-5} \) for the static approach. For the dynamic \( T_{UC} \), if a scanning process moves \( T_{exp} \) blocks from \( WR \) to \( ER \), \( T_{UC} \) is subtracted by \( 0.1 \times 10^{-5} \).
reduced by 19.7% for the 2MB WBC. The average size of evicted blocks has been increased by 19.1% compared with CLC as shown in Figure 7. ExLRU delivers better performance when the WBC is small.

5.2.2 Reduction of the number of Write and Erase Operations

Write and erase operations are critical to the performance of flash memory. Figures 8 and 9 present the reduction in the number of write and erase operations on flash memory when the proposed ExLRU scheme is compared against FAB, BPLRU and CLC schemes. As shown in Figures 8(a) and 9(a), ExLRU_D can reduce the number of write activities on flash memory by 3% on average compared with CLC. The best case occurs when a 2MB WBC is used; ExLRU_D can reduce the number of write activities for the Financial benchmark by 10.4%. Note that Figures 8 and 9 partition the write activities on flash memory into two categories, WBC writes and garbage collection (GC) writes. WBC writes represent the write operations performed when WBC blocks are evicted to flash memory. GC writes are performed to collect the invalid pages that are created by FAST as a result of the out-of-place updates. With flash memory erase operations taking place in the unit of blocks, the collection of the invalid pages in a block is always coupled with the reading of the valid pages from the corresponding block and the writing of these pages to a free block. These write operations are denoted as GC writes. Note that there are no GC writes in the BPLRU scheme, as BPLRU uses the block padding approach to read out the valid pages from flash memory, pad them to the block in the WBC, and then write the entire block back to a free block in flash memory. In comparison, while ExLRU cannot fully eliminate GC writes, it takes block size into consideration and tends to evict larger blocks early, thus significantly reducing the average number of valid pages in an erasing block.

Figures 8 and 9 also show the reductions in the number of erase operations on flash memory. It can be seen that ExLRU_D is superior to the other WBC management schemes in reducing the number of erase operations in most cases. Overall, the results in Figures 8 and 9 confirm that ExLRU_D consistently reduces the number of write and erase operations on flash memory for both workloads. This significant advantage derives from the fact that ExLRU computes the eviction cost based on the fine-grained, page-level access information and differentiates the cost of different types of blocks. When comparing the two ExLRU approaches, it can be noted that ExLRU_S does not deliver a consistent reduction on the number of write and erase operations. The reason is that the static, inflexible T_UC is not sufficient in fulfilling the requirements of WBCs with diverse sizes. For a large size WBC, the average UC should be much smaller than that of a small size WBC and vice-versa. The detailed studies on static T_UC will be provided in the next part.

5.2.3 Parameter Sensitive Studies

We perform detailed studies on three parameters of ExLRU on the Financial workload, as it is widely used in prior work. The first parameter we examine is \( N_{\text{min}} \), which is used as the lower bound of the number of blocks in \( ER \). A large \( N_{\text{min}} \) would induce more scanning processes and degrade the quality of eviction decisions. We study the impact of \( N_{\text{min}} \) on the number of scanning processes and erase operations on flash memory to find the best \( N_{\text{min}} \) for ExLRU. The second parameter we examine is \( T_{\text{UC}} \), which is used during the scanning process as the threshold \( UC \) to select blocks to be put in \( ER \). A large \( T_{\text{UC}} \) would result in too many blocks to be added to \( ER \), while these blocks display large UCs, thus degrading the eviction decisions. On the other hand, a small \( T_{\text{UC}} \) would induce too many scanning processes as each scanning process may only find a very limited number of blocks to be added to \( ER \). We also study the relationship between the scanning process and the number of erase operations by varying \( T_{\text{UC}} \) from \( 0.1 \times 10^{-5} \) to \( 2.0 \times 10^{-5} \). The third parameter we examine is \( T_{\text{ex}} \), which is used to avoid the inappropriate setting of \( T_{\text{UC}} \) in the dynamic approach.

Impact of \( N_{\text{min}} \): Figure 10 depicts the relationship between the number of erase operations and the scanning processes

Figure 10: The impact of \( N_{\text{min}} \) on the number of erase operations and the scanning processes
in Figure 10, as $N_{\text{min}}$ increases, both the number of erase operations on flash memory and the number of scanning processes on the WBC increase. When $N_{\text{min}} = 1$, the number of erase operations reaches the minimum, while the number of scanning processes reaches the minimum when $N_{\text{min}} = 0$. A smaller $N_{\text{min}}$ not only allows the selection of more blocks upon each scanning process, but furthermore tightly bounds the eviction cost of the blocks in ER and hence reduces the number of erase operations. As a result, in this work, we set the minimal number of blocks in ER to 1.

However, we find that a large $T_{UC}$ would also result in a significant increase in the number of erase operations. To balance the frequency of scanning and the number of erase operations, in this work, we set $T_{UC}$ to $0.1 \times 10^{-5}$ for the static approach. Yet it needs to be noted that the best static $T_{UC}$ varies from workload to workload; our study on the PC trace benchmark indicates that the best $T_{UC}$ for that benchmark is $0.2 \times 10^{-5}$. On the other hand, for the dynamic approach, we find that it consistently delivers the minimal number of erase operations within a reasonable number of scanning processes.

**Impact of $T_{UC}$**: We apply and compare the static and the dynamic $T_{UC}$ setting approaches. We vary $T_{UC}$ of static approach from $0.1 \times 10^{-5}$ to $2.0 \times 10^{-5}$. Figure 11 shows the results regarding the impact of $T_{UC}$ on the number of scanning processes and erase operations. The two dot lines represent the static $T_{UC}$ case while the two dash lines represent the dynamic $T_{UC}$ case. As shown in Figure 11, the number of scanning processes is reduced as $T_{UC}$ increases.

**Impact of $T_{scan}$**: In order to avoid the inappropriate setting of the dynamic $T_{UC}$, $T_{scan}$ is used to bound the maximal number of blocks that can be selected in a single scanning process. We vary $T_{scan}$ from 1% of the total number of blocks in WBC to 20% to find the relationship between the
number of scanning processes and the erase operations. Figure 12 shows such a relationship using workload Financial. It can be seen that varying $T_{scan}$ has little impact on the number of erase operations, while a large $T_{scan}$ tends to reduce the number of scanning processes. In this work, we set $T_{scan}$ to 5% of the total number of blocks.

6. CONCLUSIONS

We have proposed in this paper a new WBC management scheme, ExLRU, for flash memory based storage systems. This scheme is designed to improve the write performance and reduce the number of erase operations on flash memory. Unlike previous work that focuses only on a single type of access patterns of workloads, ExLRU takes diverse types of access patterns into consideration and selects the lowest cost blocks for eviction. This is accomplished through the development of a unified cost model that utilizes the fine-grained, page-level access information as well as the block size to determine the eviction cost of blocks. Furthermore, an efficient implementation of ExLRU with negligible overhead is proposed to effectively approximate the optimal ExLRU. Simulation results show that ExLRU outperforms previous WBC management schemes through simultaneously reducing the number of write and erase operations by 10.4% and 11%, respectively.

7. REFERENCES


