IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 21, 2019, accepted March 8, 2019, date of publication March 12, 2019, date of current version March 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2904639

An Advanced Adaptive Least Recently Used Buffer Management Algorithm for SSD

YINGBIAO YAO, XIAOCHONG KONG, JIE ZHOU, XIAORONG XU, WEI FENG, AND ZHAOTING LIU

School of Communication Engineering, Hangzhou Dianzi University, Hangzhou 310018, China

Corresponding authors: Jie Zhou (1395529361@qq.com) and Wei Feng (396843354@qq.com)

This work was supported in part by the China Postdoctoral Science Foundation under Grant 2017M621796, in part by the Zhejiang Province Science Foundation for Public Welfare under Grant LGG19F020014, and in part by the National Natural Science Foundation of China under Grant 61671192.

ABSTRACT The traditional solid-state drive buffer management algorithm generally adopts fixed structures and parameters, leading to their poor adaptability. For example, after the underlying flash translation layer (FTL) or the upper workload is changed, the traditional algorithm's performance fluctuates significantly. Focusing on this problem, based on the cross-layer aware method, we propose an Advanced Adaptive Least Recently Used buffer management algorithm (AALRU). The core idea of the AALRU is that by sensing the characteristics of the upper workload and the status of the underlying FTL, the AALRU adaptively adjusts its structure, parameters, and write-back strategy to optimize the buffer's performance. First, the AALRU divides the buffer into two parts: read buffer and write buffer, and their proportion is adjusted by sensing the read-write characteristics of the workload and the underlying read-write latency. Second, the AALRU employs different granularities to manage the buffer. On one hand, for data loading and migrating, the AALRU adopts page-level granularity, which can avoid the problem of hot and cold data page entanglement in block management, and thus improve the buffer hit ratio. On the other hand, for data writing back to the FTL, the AALRU adopts block-level granularity, which can enhance the continuity of write requests and thus reduce the underlying FTL's garbage collection overhead. Finally, when the clustered data are written back, by sensing the underlying FTL's garbage collection status, the AALRU adaptively adjusts the page-padding trigger threshold to reconstruct the continuity of the write-back data, which can mitigate the underlying FTL's garbage collection overhead. The experimental results show that the AALRU has the best adaptability to different FTLs and test workloads, and it can achieve optimal or near-optimal results.

INDEX TERMS Solid-state drive, buffer management, adaptive algorithm, cross-layer aware.

I. INTRODUCTION

The performance of processors has increased rapidly in accordance with Moore's Law over the past few decades. However, the performance improvement of a storage system based on a hard disk drive (HDD) has been slow, which has led to an increasing performance gap between computing and storage [1]. Compared with the traditional HDD, the solid-state drive (SSD) based on the NAND flash memory has many excellent features such as its low power consumption, fast reading and writing speed, light weight, small size, shock resistance, and noise-free performance [2], [3]. Therefore, SSD is widely used in consumer electronics and enterprise-level computer systems.

Despite the SSD's performance advantages, due to the characteristics of the underlying NAND flash memory, it retains many disadvantages [4]–[6]. For example, the in-place update is not allowed because NAND flash memory has the erase-before-write physical restriction, i.e., once a storage unit is written, it cannot be programmed until it is erased. Injecting electrons into the floating gate always takes longer than sensing its status, resulting in SSD's asymmetric performance between read, write and erase. Moreover, NAND flash memory has limited program/erase (P/E) times, that is to say, when the erase counts exceed the maximum value, the dedicated storage unit cannot store data reliably.

The associate editor coordinating the review of this manuscript and approving it for publication was Guan Gui.

To solve or alleviate the above problems caused by the characteristics of underlying NAND flash memory in the SSD, two methods are mainly used: 1) Add a flash translation layer (FTL) to the SSD to hide the characteristics of the underlying flash memory [7], [8] so that the SSD can work like a traditional HDD. 2) Add a buffer layer to the SSD and design a corresponding buffer management algorithm (BMA), so that most of the requests can be served in the buffer. In this paper, we focus on the latter.

Generally, the size of the buffer is limited and much smaller than the storage space. To maximize the usage of buffer space, the BMA should load the frequent-access data into the buffer and evict the infrequent-access data when the buffer is full. Due to the physical restriction of NAND flash memory, i.e., the read, write and erase performance of NAND flash are asymmetrical and the cost of write is significantly higher than that of read [9]–[11], the traditional BMA developed for the HDD is not applicable on SSD. Therefore, many BMAs are proposed for SSD's buffer management. However, the existing BMAs for SSDs generally adopt fixed structures and parameters, leading to their poor adaptability. After the underlying FTL or the upper workload is changed, their performance may degrade dramatically.

In this paper, based on the cross-layer design idea, we propose an advanced adaptive least recently used (AALRU) buffer management algorithm for SSDs. The key idea is to adaptively adjust the buffer structure, parameters, and write-back strategy by sensing the characteristics of the upper workload and the status of the underlying FTL, thereby increasing the buffer hit rate, reducing the actual flash writes and erase counts, and improving the SSD's performance. The experimental results show that under different FTLs and different test workloads, AALRU has the best adaptability and can achieve optimal or near-optimal results. Specifically, compared with the classic BPLRU in [12], FAB in [13], CFLRU in [14], and ADLRU in [15], AALRU improves the buffer hit rate, block erasure counts, and average response time, as shown in Table 1.

TABLE 1. Experimental results.

	BPLRU	FAB	CFLRU	ADLRU
Buffer Hit Rate	96.9%	144.2%	3.0%	30.9%
Block Erase Counts	21.3%	27.5%	17.7%	23.5%
Average Response Time	12.4%	24.7%	14.1%	21.6%

The rest of this paper is organized as follows. In Section 2, we present related work, including introduction of flash memory, various existing FTLs and BMAs. The proposed BMA is described in Section 3. The experimental results are explained in Section 4. The conclusions are presented in Section 5.

II. RELATED WORK AND MOTIVATIONS

A. FLASH MEMORY

There are two types of flash memories: NOR and NAND. NOR flash memory supports random accesses in bytes and it is mainly used for storing code. NAND flash memory is designed for storing data with denser capacity and only allows access in pages. Nowadays, Most SSDs available on the market are based on NAND flash memories.

NAND flash memory can be classified into three categories: single-level cell (SLC), multi-level cell (MLC) and triple-level cell (TLC). The SLC, MLC and TLC flash memory cells store one bit, two bits, three bits or even more, respectively. For all types of NAND flash memory, a package is composed of one or more dies (chips). Each die within a package contains multiple planes. A typical plane consists of thousands (e.g., 2,048) of blocks. Each block in turn consists of 64 to 128 pages. Each page has a data area, for storing user data, and an out-of-band area, for storing other data (e.g., identification, page state, and error correcting code).

Flash memory supports three major operations: read, write, and erase. Read and write are done in units of pages while erase is done in units of blocks. Flash blocks must be erased before they can be reused, known as the erase-before-write property. In addition, each block has a finite number of P/E times. The P/E cycles are usually around 1,000 to 100,000, depending on the type of the flash type. For example, the SLC flash has about 100,000 erase cycles and the TLC has only about 1,000 erase cycles.

B. FLASH TRANSLATION LAYER

As an essential firmware of SSD, FTL plays a key role in providing address mapping, garbage collection and wear leveling, which allows hosts to access flash memory in the same way as conventional HDDs. Address mapping is the core functions, because the flash does not support in-place updating and requires erase before write, so an address mapping scheme is required to store the updated data in other free pages of the flash and then invalidate the pages that previously stored the data.

According to the size of the address mapping granularity, address mapping can be divided into page-level, block-level, and hybrid mapping. Page-level mapping needs to maintain the mapping relationship between the physical page and the logical page. Block-level mapping needs to maintain the mapping relationship between the logical block and the physical block. Since a block of flash memory usually contains multiple (64 or 128) pages, the block-level mapping has a much smaller table size than the page-level mapping. However, the consistency of the intra-block data page offset address in block-level mapping results in low efficiency. To overcome the shortcomings of these two mapping methods, hybrid mapping uses a dedicated log block to record updated data, a page-level mapping scheme is used in the log block, and other storage spaces use a block-level mapping scheme, effectively reducing the occupation of the page-level mapping table and reducing the erase counts of the block-level mapping scheme.

Representative hybrid address mapping algorithms are FAST [16], Superblock [17], LAST [18], etc. However, hybrid mapping generally has the problem of full merge caused inefficient garbage collection. The cost of full merge is

too high, which seriously affects the read-write performance and increases the wear of the SSD. Therefore, many new page-level address mapping algorithms have been proposed, such as DFTL [19], OAFTL [20], K-CPM [21], DAC [22], DVPFTL [23], MVFTL [24], etc. Currently, hybrid mapping or new page-level mapping are the commonly used address mapping algorithms.

C. BUFFER MANAGEMENT ALGORITHM

The buffer is an optional module of SSDs, which is located between the host interface layer and the FTL and mainly used to improve the read-write performance and extend the life of SSDs. In the BMA design for SSDs, it is necessary to cache the write request of high-frequency access as much as possible, reconstruct the random request as the sequential request, and reduce the FTL's write amount and garbage collection overhead. According to the management granularities, the BMA can be divided into page-level BMA and block-level BMA.

The page-level BMA manages the buffer with the granularity of a page; the representative algorithms include CFLRU, LRU -WSR [25], CCF-LRU [26], CLRU [27], ADLRU, etc. CFLRU takes into account the different read-write delays and costs in the flash, and prioritizes replacing clean pages in the buffer based on the LRU strategy. LRU-WSR divides the dirty pages in the buffer into cold and non-cold by means of the ''cold detection'' scheme based on the secondary opportunity and preferentially evicts the cold dirty pages when the dirty pages are removed. CCF-LRU further recognizes the clean pages of the buffer as cold and hot, and the buffer evicting order is cold clean pages, cold dirty pages, hot clean pages, and hot dirty pages. Based on CCF-LRU, CLRU recognizes cold and hot data pages with relative access spacing and dynamically distinguishes between cold and hot of clean and dirty queues. However, in the actual system operation process of LRU-WSR, CCF-LRU, and CLRU, a problem is that the hot clean page cannot be recognized because the cold region is gradually reduced. For this problem, ADLRU sets the minimum cold region lower limit (***min_lc***). When the cold region size reaches the ***min_lc***, the hot region is selected for eviction to ensure that the data in the cold region can gradually become hot. In the processing of hot page and cold page recognition, page-level algorithms, such as CARF [28] and PR-LRU [29], recognize hot data based on the page access history information. CARF converts the corresponding page information into page weights and manages the data pages by the weights. PR-LRU converts the corresponding page information into the probability that the data page will be accessed again, and selects the data page with the lowest probability as the victim page. Generally, because of the page-level granularity, the page-level BMA can better identify hot and cold of the data page, thereby increasing the hit rate of the buffer. However, when the data page is evicted from the FTL, since the evicting size is a page, clustered write-back data pages cannot be implemented, which may increase the overhead of FTL's garbage collection.

Unlike the page-level BMA, the block-level BMA manages the buffer with the granularity of a block (i.e., the data pages of the same logical block). Thus, it can reconstruct random requests into sequential requests and cluster the data pages of the same logical block to writeback, which reduces the FTL's garbage collection overhead. In the specific implementation, the block-level BMA only caches write requests generally, and the representative algorithms are FAB, BPLRU, CLC [30], etc. FAB sorts the clusters in the buffer according to the cluster size. When the buffer is full, the largest block is found for eviction. Based on the LRU strategy, BPLRU adopts the LRU compensation scheme when continuously writing and uses the page padding scheme when evicting, which can improve the block-level BMA's performance. CLC divides the buffer block queue into high-time and low-time local characteristic block queues. The LRU strategy is used to manage high-time local characteristics block queues, and FAB is used to manage low-time local characteristics block queues. The ratio of the number of blocks contained in the two queues is approximately α. When the buffer is full, the largest block in the block queue with the low-time local characteristics is selected for eviction. Based on the CFLRU, CFDC [31] adopts hybrid management granularity. In the priority replacement region of CDFC, the dirty pages are clustered into blocks to write-back, which reduces the number of writeback compared with CFLRU. Overall, the block-level BMAs has better write-back performance, but the buffer hit rate is lower. The comparison of mentioned BMAs are summarized in table 2.

D. MOTIVATIONS

The above-mentioned BMAs have designed well the SSD's buffer management from the page-level and the block-level perspectives. However, they have failed to solve the following three problems at the same time:

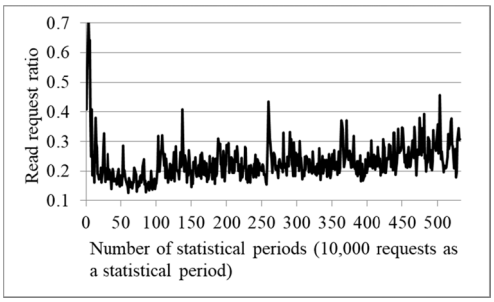
1) NON-ADAPTIVE STRUCTURE AND PARAMETERS

The structure of the existing BMA is mostly fixed, and many parameters are also specified in advance, which makes the existing BMA not suitable for various workloads generally.

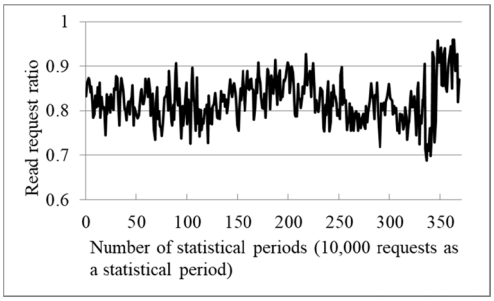
In the actual application, many types of SSD workloads exist, and their characteristics are difficult to predict. Even the same workload exhibits different characteristics in different statistical periods. Figure 1 shows the read requests ratio of Fin1 and Fin2 [32]. In Figure 1, the horizontal axis represents the number of periods and 10,000 requests of the workload are regarded as one period. Among them, Fin1 is the write-intensive type, and Fin2 is the read-intensive type. As illustrated in Figure 1, both Fin1 and Fin2 have sharp changes in their read ratios. So the read-write ratio of workloads changes always. However, the existing BMAs have failed to be specifically designed for the workloads characteristics.

TABLE 2. Comparison of algorithms.

Algorithm	Granularity	Adaptability	Replacement Strategy
CFLRU	Page	No	Clean pages preferentially.
LRU-WSR	Page	No	Cold dirty pages preferentially.
CCF-LRU	Page	No	Evicting order is cold clean pages, cold dirty pages, hot clean pages, and hot dirty pages.
CLRU	Page	No	The page with least probability of being referenced again.
ADLRU	Page	Yes	Evicting the clean page first from the cold LRU queue when the cold region size does not reach the *min_lc*.
CARF	Page	No	The page with the least weight based on frequency and recency.
PR-LRU	Page	No	The minimum probability page is inserted into victim LRU list and the clean page in victim LRU list is selected for eviction.
FAB	Block	No	The largest block.
BPLRU	Block	No	The least recent block with page padding and LRU compensate scheme for a sequential write pattern.
CLC	Block	No	The largest block in low-time local characteristic blocks or the least recent block in high-time local characteristic blocks.
CFDC	Hybrid	No	Clean pages are first considered for replacement and dirty pages are clustered into blocks to write back.



(a)



(b)

FIGURE 1. Read ratio of workloads from OLTP [32] application. (a) Read ratio of Fin1. (b) Read ratio of Fin2.

2) UNBALANCED BUFFER HIT RATE AND CLUSTERED WRITEBACK

As mentioned, the page-level BMA has a high buffer hit rate, and the block-level BMA has a good clustered write-back performance, but the existing BMA cannot consider both at the same time. Despite the page-level BMA's high buffer hit rate, there is no clustered writeback, which may cause a large FTL garbage collection overhead. Although the block-level BMA has a good clustered write-back performance, due to the extremely uneven hot and cold data pages in the hot blocks, a large number of cold pages in the hot blocks occupy the buffer space, thereby reducing the buffer's hit rate. Figure 2 shows the hot and cold statistical ratio of the data pages in the hottest 24 blocks of Fin1. The data pages with access counts below 100 are recorded as cold data pages, the access counts higher than 10,000 are recorded as hot pages, and the others are recorded as non-cold pages. The statistical results in Figure 2 show that only 1-3 hot data pages and about 50-60 cold data pages in the hot block. Therefore, the hot data blocks are filled with a lot of cold data pages.

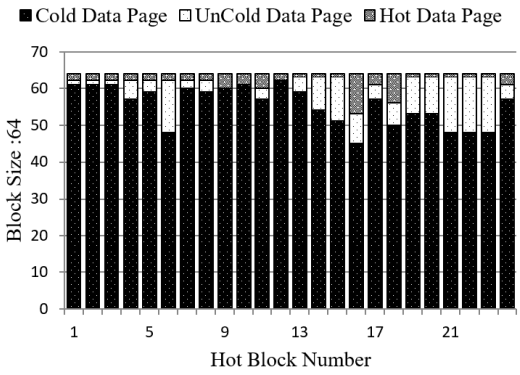


FIGURE 2. Hot and cold ratio of the data pages in the hottest 24 blocks of Fin1.

3) UNCONSIDERED UNDERLYING GARBAGE COLLECTION

In the above-mentioned BMAs, the underlying FTL's garbage collection status is rarely considered. Only BPLRU considers that the excessively fragmented data will cause the increasing overhead of the underlying garbage collection. Therefore, when the data pages in the buffer write back, the page padding scheme is used to improve the continuity of the write-back data. However, BPLRU employs page padding in all write-back blocks, which may sometimes lead to excessive overhead in writing back.

Figure 3(a) shows the page padding counts distribution of BPLRU when the workload is Fin1, and Figure 3(b) shows the number of write caused by page padding and the actual clustered write counts of BPLRU. Based on Figure 3, it can be concluded that BPLRU's page padding scheme can easily cause an excessive amount of data writes. Therefore it is very important to choose the right time to trigger the page padding scheme according to the underlying garbage collection status.

To solve the cited problems, this paper proposes a cross-layer aware AALRU for the SSD's buffer management. Its main contributions are listed as follows:

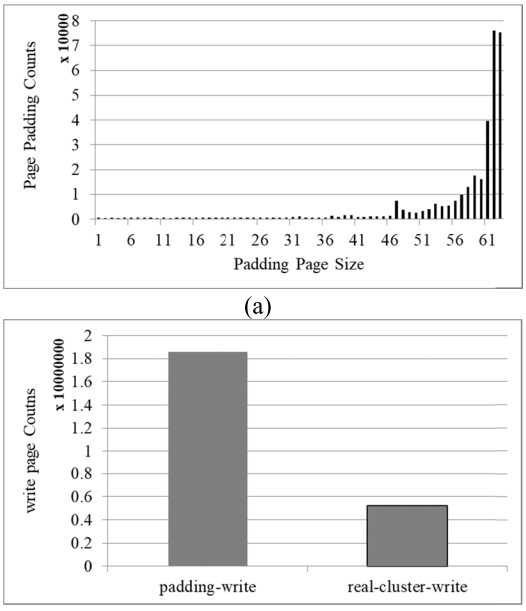


FIGURE 3. Page padding effect of BPLRU under Fin1. (a) Page padding counts distribution of BPLRU. (b) Page padding caused write counts and actual clustered write-back counts of BPLRU.

1) *Adaptively Adjusted Read-Write Buffer Ratio:* The buffer is divided into two buffers for reading and writing, and the ratio can be adaptively adjusted according to the read-write ratio of the upper workload and the read-write cost of the underlying flash memory.

2) *Hybrid Management Granularity:* Data loading and migration of the read-write buffers are managed in page units according to the LRU strategy. The victim buffer is adaptively selected according to the current actual ratio and the expected ratio of the read-write buffers. When the buffers' data pages are written back, they are clustered back to the FTL in block units.

3) *Adaptively Adjusted Page Padding Trigger Threshold:* To reduce the underlying FTL's garbage collection overhead, AALRU can sense the FTL's garbage collection pressure by means of the approximate flash write amplification factor, and the page padding trigger threshold is adaptively adjusted according to the FTL's garbage collection pressure.

III. PROPOSED AALRU

A. STRUCTURE OF AALRU

Figure 4 shows the overall structure of AALRU. It is divided into two parts: the data page buffer and the adaptive threshold adjustment scheme. The data page buffer is further divided into a read buffer (RB) and a write buffer (WB). The WB caches dirty pages (data pages corresponding to write requests), and the RB caches clean pages (data pages corresponding to read requests). The granularity of data loading and migration in the RB and the WB is page, and the management strategy is the LRU. The dirty data pages of the same logical block are clustered to write back. The adaptive threshold adjustment scheme includes the read-write buffer ratio threshold τ and the page padding trigger threshold Th. By sensing the read-write ratio of the upper layer workload and the status of the underlying FTL, AALRU can adaptively adjust the thresholds τ and Th.

B. ADAPTIVE UPDATING OF READ-WRITE BUFFER RATIO THRESHOLD

AALRU periodically counts the read-write ratio of the workload and the read-write performance of the buffer to determine the appropriate read-write buffer ratio threshold τ for the next period. Specifically, AALRU processes the number of T access requests as a period. Next, AALRU counts the RB read hit times RRH and the RB write hit times RWH, the WB read hit times WRH and the WB write hit times WWHin the period, and counts the average read delay D_r and write delay D_w of the single data page in the period. It then calculates the WB unit benefit WR and the RB unit benefit RR, as follows.

$$WR = \frac{WWH \times D_w + WRH \times D_r}{BS - \tau'} \tag{1}$$

$$RR = \frac{RWH \times D_w + RRH \times D_r}{\tau'} \tag{2}$$

where BS is the size of the total buffer and τ' is the threshold in the last period. According to WR and RR, the threshold τ of the next period is calculated as follows.

$$\tau = \frac{RR}{WR + RR} \times BS \tag{3}$$

where τ is the expected RB normalized size. When AALRU needs to evict the data page from the buffer, it compares the current RB size RL with the expected τ. If $RL \geq \tau$, the data page in the RB is evicted, and otherwise evicted from the WB.

C. ADAPTIVE UPDATING OF THE PAGE PADDING TRIGGER THRESHOLD

Different address mapping schemes used in FTL may cause different garbage collection performance. Regardless of the FTL's address mapping scheme, its write amplification factor, Wf,can reflect its garbage collection performance. The variable Wf is the ratio of the actual flash write counts to the buffer write-back counts Bf_w. The actual flash write counts includes Bf_w and the write counts GC_w caused by garbage collection. The Wf can be calculated as follows.

$$Wf = \frac{GC_w + Bf_w}{Bf_w} \tag{4}$$

Calculating Wf by Equ. (4) requires the FTL to report GC_w, but sometimes, it is difficult to obtain GC_w directly from the FTL. Equ. (4) can be rewritten as:

$$Wf = \frac{(GC_w + Bf_w) \times F_w}{Bf_w \times F_w} = \frac{\left(\frac{GC_w}{Bf_w} + 1\right) \times F_w}{F_w} \simeq \frac{D_w}{F_w} \tag{5}$$

where F_w is the standard writing delay of the flash page, and D_w denotes the average delay of the actual page write-back request of the buffer, which contains the standard writing

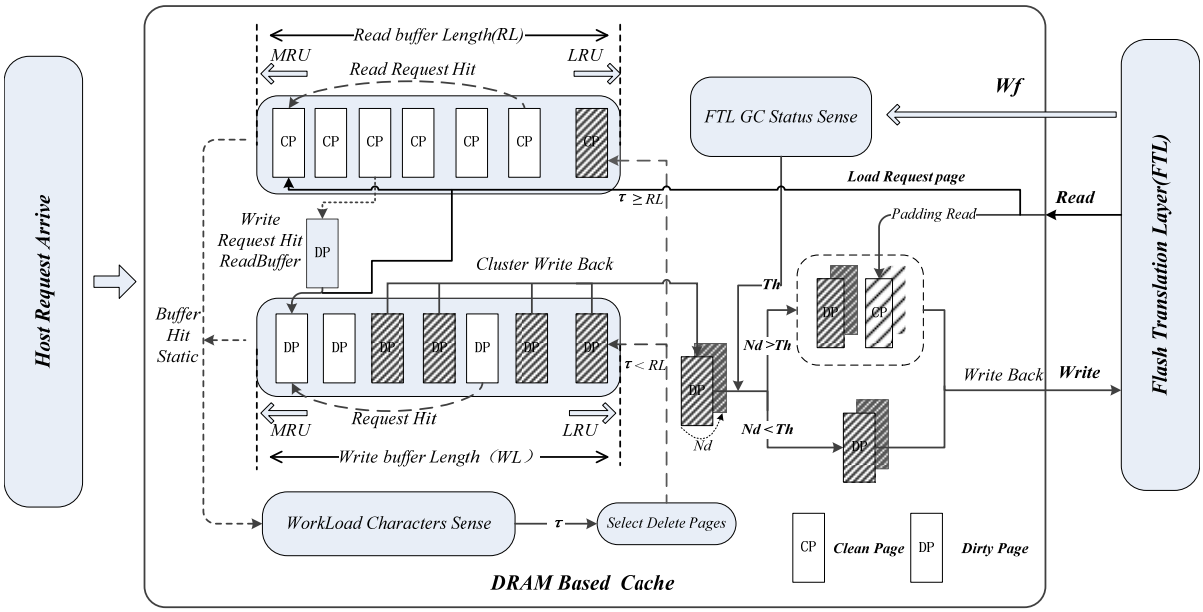


FIGURE 4. Structure of AALRU.

delay and the average write delay caused by garbage collection. The D_w can be obtained by counting the response completion time and arrival time of each request in the buffer management layer, and the F_w can be obtained according to the parameters of the flash chip. When *Wf* increases, it indicates that the garbage collection overhead of the current FTL is large. It is necessary to reduce the randomness of the write-back request when the data page in the buffer is evicted into the flash memory.

In this paper, two methods are used to reduce the randomness of the write-back data. 1) Using a clustered write-back strategy, that is, all dirty data pages belonging to the same logical block are written back to the flash memory at the same time. 2) Using an adaptive page padding strategy. AALRU introduces a page padding threshold *Th*, and then compares N_D (the number of dirty pages of the current victim logical block) with *Th*.If $N_D \geq Th$, the page padding scheme is adopted; Otherwise, there is no page padding. The *Th* value is adaptively adjusted as the *Wf* changes. AALRU models the relationship between *Wf* and *Th* as linear, as shown in Figure 5.

When $Wf = 1$, it means that there is no garbage collection, so *Th* is set to *BMS* (the number of pages per block) and page padding is prohibited. When $Wf > \beta$ (β is the write amplification factor for unconditional page padding), it means that the garbage collection overhead is very large; at this time *Th* is set to 0, that is, page padding is used in all write-back pages. When *Wf* is in the interval $[1, \beta]$, *Th* exhibits a linear relationship with *Wf*. According to Figure 5, the relationship between *Th* and *Wf*can be obtained

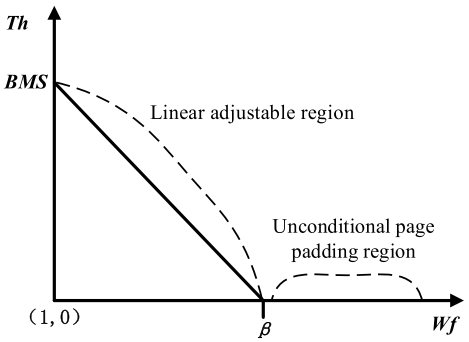


FIGURE 5. Linear relationship between threshold *Th* and *Wf*.

as follows.

$$Th = -\frac{BMS}{\beta} Wf + BMS \tag{6}$$

Equ. (6) shows that in the threshold adjustment region, when *Wf* increases, it indicates an increasing pressure of the FTL garbage collection, at this time, *Th*decreases, and then the continuity of the write-back requests increases. When *Wf* decreases, it indicates a decreasing pressure of the FTL garbage collection, so *Th* increases to avoid unnecessary page padding.

D. HYBRID MANAGEMENT GRANULARITY

To realize hybrid management granularity, AALRU sets a cache block record table in the buffer. The cache block record table records the position pointer of different data pages of the

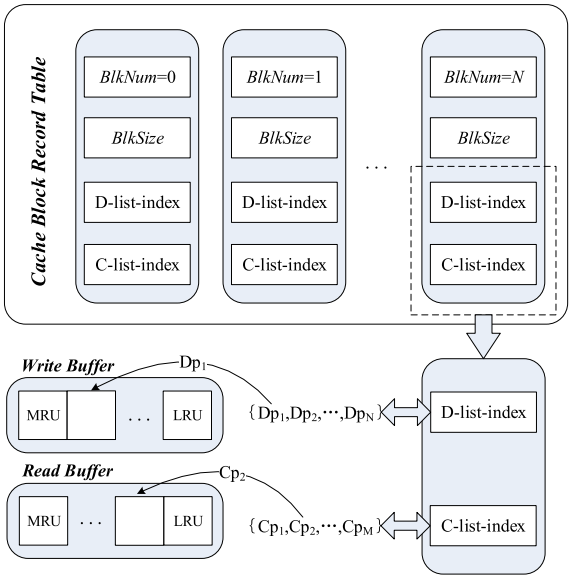


FIGURE 6. Structure of cache block record table.

same logical block in the buffer and is used to assist in implementing dirty page clustered writeback and the recognizing hot and cold data pages.

As shown in Figure 6, an entry in the cache block record table consists of the logical block number *BlkNum*, the number of data pages *BlkSize*, the clean pages position pointer list *C-index-list* and the dirty pages position pointer list *D-index-list*. The pointers in *C-index-list* and *D-index-list* point to data page positions in RB and WB, respectively. Through the cache block record table, AALRU can realize the precise operation of each page like the page-level BMA, and realize the clustered management like the block-level BMA. More importantly, with the cache block record table, AALRU can effectively solve the problem of hot and cold data page entanglement in the same block when clustered writeback.

When the data page of the buffer is updated, the cache block record table must update accordingly. The cache block record table's updating operations include migration updating, replacement updating, and load updating. 1) In migration updating, when the data page is hit in the buffer, the position pointer in the corresponding entry needs to be moved to the MRU position. 2) In replacement updating, when the data page is replaced, the corresponding position pointer is deleted. 3) In load updating, when the new data page is loaded into the buffer, a new position pointer is allocated and inserted into the MRU position of pointer list.

The following example describes the overhead of the cache block record table. Assuming that the read-write buffer is 8 MB, the data page is 2 KB, the data block is 128 KB, and the upper file system is 32 bits, the overhead of the cache block record table is as follows: the *BlkNum* is 32 bits, and the *BlkSize* is 5 bits. The 8 MB buffer can store 4,096 pages, and the corresponding position pointer requires 12 bits. Additionally, 1 bit is required to indicate whether the page is dirty or clean. Therefore, the worst situation is that all data pages in the buffer belong to different blocks, and the overhead of the cache block record table is $(32 + 5 + 12 + 1) * 4096 = 204,800$ bits (25 KB), which is 0.3% compared with the 8 MB buffer. The best situation is that all data pages in the buffer are organized in complete blocks, and the overhead is $(32 + 5 + 12 * 64 + 1 * 64) * 64 = 55616$ bits (about 7KB), which is 0.08% compared to the 8 MB buffer.

E. SPECIFIC PROCEDURE OF AALRU

Algorithm 1 and 2 show how AALRU works and they manage the data in the buffer and output nothing. Algorithm 1 shows AALRU processing an access request. The inputs are the request logical page address, the request size, and the request type. The pseudo-code of Algorithm 1 is briefly explained as follows: Lines 2-7 show the processing flow of the hit request, that is, the read request hit in the RB is moved to the MRU position of the RB queue. The write request hit in the RB or read/write request hit in the WB is moved to the MRU position of the WB queue. When a request is hit, AALRU also updates *RRH*, *RWH*, *WRH*, and *WWH* for subsequent threshold τ updating calculations. Line 10-18 show the processing flow of the missed request. If the buffer is full, AALRU calls Algorithm 2 to evict the data page, and then loads the request data page to the corresponding buffer MRU position according to the request type. Line 19 responds to the request by returning the data to the host or writing the data to the buffer. Lines 20-22 periodically update the thresholds τ and *Th* according to Equ. (3) and Equ. (6).

Algorithm 2 shows how to evict the data pages when the buffer is full. When $RL > \tau$, the page in the LRU position of the RB queue is selected for eviction; otherwise, the page in LRU position of the WB queue is selected as the candidate victim page, and the victim logical block number of the clustered writeback is determined according to the logical page number of the candidate victim page. Then, the data pages, belonging to the victim logical block, are selected to form a writeback block. If $N_D > Th$, AALRU writes the victim block back with page padding; otherwise, it writes the victim block back without page padding. Lastly, AALRU deletes all pages of the selected victim block in the WB.

IV. PERFORMANCE ANALYSIS

A. EXPERIMENTAL SETTINGS

To evaluate the performance of AALRU, we have added the buffer management module into FlashSim [33]. In the experiment, the buffer size is fixed at 8 MB and the storage space is 4 GB. The underlying FTL is DFTL or FAST, which represent the page-level and the hybrid FTL, respectively. Table 3 lists the key parameters of the underlying flash memory.

Four workloads are used in the experiment, including the enterprise-level workloads Fin1 and Fin2, and workloads synthesized with DiskSim [34], T1 and T2. Table 4 presents the characteristics of the four workloads.

The buffer hit ratio, the average response time of requests and the block erase counts are selected to evaluate the

Algorithm 1 Procedure of AALRU

Input: request logical page address Req_{lpn}, request size Req_{size}, request type Req_{type},
Output: NULL
Abbreviation description: read buffer (RB), read buffer size RL, write buffer (WB), processing request number $SysCount$, threshold updating period T, buffer ratio threshold τ, page padding trigger threshold Th

```
1.  WHILE Req_size ≠ 0 DO
2.    IF Req_lpn hits in Buffer THEN
3.      IF Req_type is Read and Req_lpn in RB THEN
4.        Move Req_lpn to the MRU position of RB
5.      ELSE
        /* RB write hit and WB read and write hit */
6.        Move Req_lpn to the MRU position of WB
7.      ENDIF
        /* Count each buffer request hit information */
8.      Update Buffer Request Hit Static
9.    ELSE
10.     IF Buffer is full THEN
        /* Call algorithm 2 to evict data page from
             buffer */
11.       DeleteVictimPages(RL, τ, Th)
12.     ENDIF
13.     IF Req_type is read THEN
         /* Load the data page of read request to RB */
14.       Load Req_lpn to the MRU position of RB
15.     ELSE
         /* Load the data page of write request to WB */
16.       Load Req_lpn to the MRU position of WB
17.     ENDIF
18.   ENDIF
19.   Service Request
20.   IF mod(SysCount, T) == 0 THEN
21.     Update Th and τ/* Update threshold
                                  periodically */
22.   ENDIF
23.   SysCount++
24.   Req_lpn ++
25.   Req_size−
26. ENDWHILE
```

TABLE 3. Experimental flash configuration.

Parameter	Value
Page Size/KB	2
Block Size/KB	128
Page Read/us	32.7
Page Write/us	101.5
Block Erase/ms	1.5
Over-provisioning coefficient	0.2

performance, which can reflect the buffer's ability to retain hot data, SSD's read-write performance, and SSD's wear degree (the SSD's lifetime), respectively. The higher buffer hit ratio, the lower average response time and the lower block erase counts are expected. Compared with other algorithms, the performance improvement ΔP is measured as follows:

$$P = \frac{\left|P_{proposed} - P_{compared}\right|}{P_{compared}} \times 100\% \tag{7}$$

The selected comparison algorithms include two classic page-level BMAs (CFLRU and ADLRU) and two classic block-level BMAs (BPLRU and FAB). Additionally, according to [14], the priority replacement window ratio of CFLRU is set to 0.4; according to [15], the minimum cold region lower limit ratio of ADLRU is 0.2. Moreover, in this paper the write amplification factor of AALRU for unconditional page padding is set to 5. The statistical period T is 8192 access requests.

Algorithm 2 Evicting the Victim Pages of AALRU

Input: read buffer size RL, buffer ratio threshold τ, page padding trigger threshold Th
Output: NULL

```
1.  IF RL≥ τ THEN        /* Select RB as the
                                victim buffer */
2.    Delete the LRU page in RB
3.  ELSE      /* Select WB as the victim buffer */
4.    Select the LRU page in WB as the victim page
5.    Select the victim block according
            to the victim page
6.    IF N_D > Th THEN    /* Enable page padding*/
7.      Write the victim block back with page padding
8.    ELSE       /* Do not enable page padding*/
9.      Write the victim block
            back without page padding
10.   ENDIF
11.   Delete all pages of the selected
            victim block in WB
12. ENDIF
```

TABLE 4. Workload characteristics in the experiment.

Traces	Number of Request	Write Ratio %	Average Request Size/KB
Fin1	5344983	76.84	4.92
Fin2	3698864	17.65	3.89
T1	750000	63.98	12.78
T2	750000	29.73	13.32

B. EXPERIMENTAL RESULTS

1) COMPARISON OF BUFFER HIT RATIO

Since the data evicting scheme of the compared BMAs is independent of the underlying FTL, their buffer hit ratios are the same under DFTL and FAST. AALRU considers the FTL's write amplification, which can make it different buffer hit ratios under DFTL and FAST in theory. However, the actual results show that AALRU's buffer hit ratio has

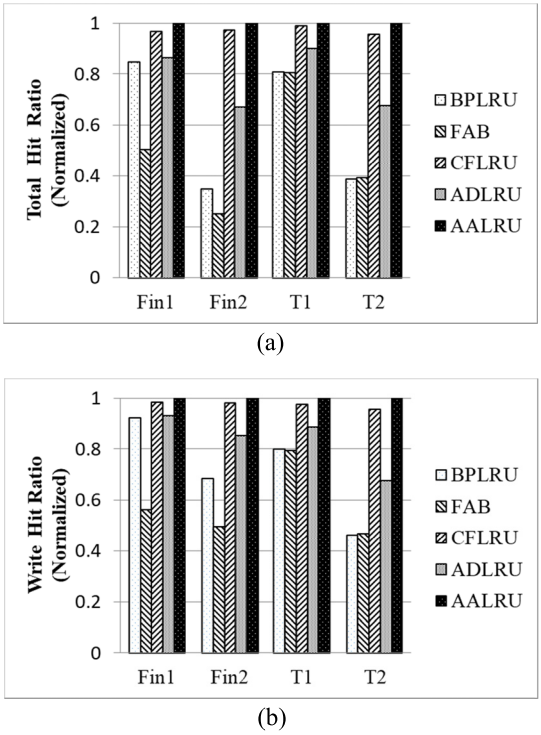


FIGURE 7. Comparison of buffer hit ratio and write hit ratio under FAST and different workloads. (a) Total buffer hit ratio comparison. (b) Write hit ratio comparison.

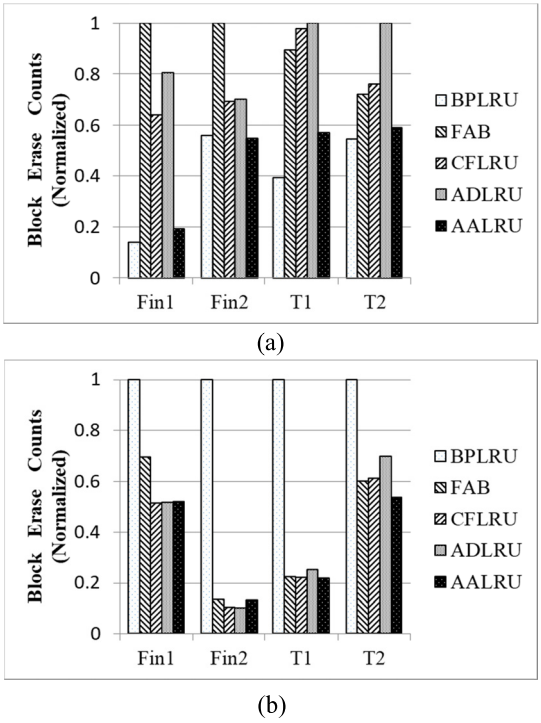


FIGURE 8. Comparison of the block erase counts under different workloads. (a) Under FAST. (b) Under DFTL.

a small difference under DFTL and FAST, so Figure 7 only shows the AALRU's buffer hit ratio under FAST. The buffer hit ratio presented in Figure 7 is normalized using AALRU's results, where Figure 7(a) gives the total hit ratio, and Figure 7(b) gives the write hit ratio.

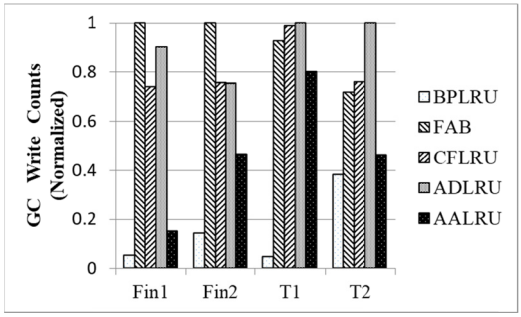
As illustrated in Figure 7, AALRU's total buffer hit ratio and write hit ratio are the best because the granularity is page-level when loading and migrating data pages and the granularity is block-level when writing back data pages. Specifically, AALRU has 96.9%, 144.2%, 3.0%, and 30.9% buffer hit ratio improvement over BPLRU, FAB, CFLRU, and ADLRU on average, respectively. Therefore, AALRU's ability to retain hot data is better than those of the compared BMAs. The buffer hit ratio of the traditional block-level BMAs is lower than that of page-level BMAs. The explanations are that the block-level BMA uses a block as the management unit and cannot distinguish between the hot and cold status of data pages in the same block, which makes the cold data pages in the hot block occupy valuable buffer space and therefore results in the reduction of the total buffer hit ratio and the write hit ratio.

2) COMPARISON OF BLOCK ERASE COUNTS

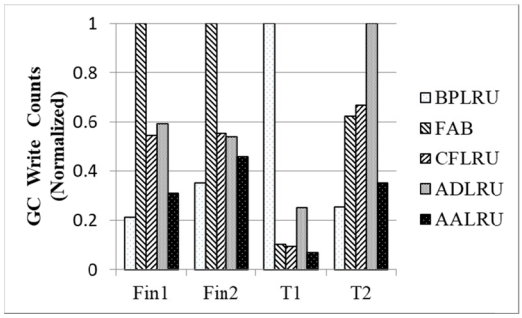
Figure 8 shows the block erase counts results of different BMAs, which are normalized using the worst experimental results for each workload. The small block erase counts in Figure 8 indicates that the light wear degree of SSD.

The experimental results in Figure 8 show that under different FTLs and workload conditions, the block erase counts performances of FAB and BPLRU vary greatly, and those of ADLRU and CFLRU vary little. The most obvious fluctuation among these BMAs is that of BPLRU, which has the least or the second-least block erase counts under FAST, and the most block erase counts under DFTL. The explanation for BPLRU's block erase counts fluctuation is its fixed page padding scheme, which only works well under FAST. In contrast, AALRU's adaptive page padding scheme works well under FAST and DFTL. Therefore, AALRU's block erase counts performance remains stable under different test conditions. Specifically, AALRU reduces the REC by 21.3%, 27.5%, 17.7%, and 23.5% over BPLRU, FAB, CFLRU, and ADLRU respectively, which indicates that AALRU has the best ability to reduce SSD's wear.

Figure 9 shows the comparison of the number of writes caused by garbage collection in various BMAs. The experimental results in the figure are normalized by the worst case, and a fewer number of writes is expected. The results in Figure 9 show that BPLRU has the lowest garbage collection overhead. The only exception occurs when the workload is T1 in Figure 9(b), and its garbage collection overhead is the worst. The experimental results show that the actual number of dirty pages in the victim block selected by BPLRU is very small under workload T1, causing a large amount of unnecessary page padding overheads. Furthermore, these large page padding overheads can result in a large number of translation page updating in DFTL, which makes BPLRU's garbage collection performance worse under DFTL and workload T1. Under all test conditions, AALRU

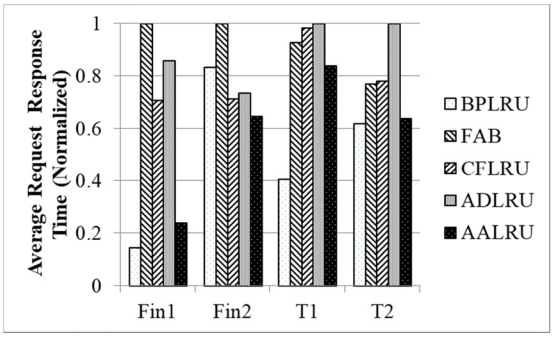


(a)

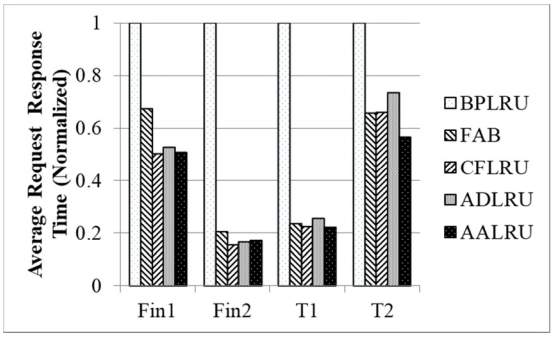


(b)

FIGURE 9. Comparison of the number of writes caused by garbage collection under different workloads. (a) Under FAST. (b) Under DFTL.



(a)



(b)

FIGURE 10. Comparison of the average response time under different workloads. (a) Under FAST. (b) Under DFTL.

has a stable garbage collection overhead, which is the least or second-least. The low overhead of AALRU's garbage collection is mainly due to its clustered writeback and adaptive page padding scheme. In the case of the clustered writeback, the FTL status is sensed by the write amplification factor, and the page padding trigger threshold is adaptively adjusted, which substantially reduces the garbage collection overhead. The comparisons of the block erase counts in Figure 8 and the average response time in Figure 10 show that while AALRU reduces garbage collection overhead, its page padding scheme does not increase excessive data page reads and writes and cause other performance degradation.

3) COMPARISON OF AVERAGE RESPONSE TIME

Figure 10 shows the average response time results of different BMAs, which are normalized by the worst case for each workload. A smaller average response time is expected, which means that the read-write time performance is better. Figure10 illustrated that the BPLRU's average response time is the best under FAST, but is far worse than those of other algorithms under DFTL, indicating BPLRU's unstable performance. The buffer hit ratio of ADLRU, CFLRU, FAB, and AALRU is relatively stable under all test conditions. Additionally, AALRU's average response time is optimal or near-optimal under all conditions. AALRU reduces the average response time by 12.4%, 24.7%, 14.1%, and 21.6% on average over BPLRU, FAB, CFLRU, and ADLRU, respectively, indicating that AALRU can improve the read-write time performance of SSD.

V. CONCLUSION

Based on the cross-layer aware method, in this paper, we have proposed an advanced adaptive SSD buffer management algorithm, AALRU, to solve the problem of poor adaptability of existing BMAs. AALRU can adaptively adjust the buffer structure, parameters, and write-back strategy by sensing the read-write characteristics of the upper workloads and the garbage collection overhead of the underlying FTL to achieve the goal of optimizing buffer performance. Specifically, AALRU first abandons the previous BMAs' fixed buffer structures and parameters, considers the actual workload characteristics and flash read-write delays, and adaptively adjusts the read-write buffer ratio to respond to read and write requests in time. Second, AALRU manages the buffer in page units when loading or migrating data and in logical block units when clustered writeback, which balances the write continuity and buffer hit ratio. Finally, AALRU senses the garbage collection overhead of the underlying FTL in real time by means of the writing amplification factor and adaptively adjusts the page padding trigger threshold, achieving the optimal adaptation of write-back request continuity and FTL status, thereby reducing garbage collection overhead. Compared with the other BMAs, under different FTLs and various types of workloads, the experimental results show that AALRU has stable adaptability and optimal or near-optimal performance.

REFERENCES

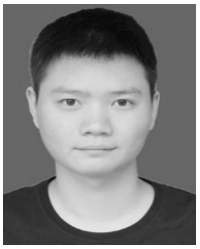
[1] B. Mao, S. Wu, and L. Duan, "Improving the SSD performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Des. Integr.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.

[2] N. R. Mielke, R. E. Frickey, I. Kalastirsky, M. Quan, D. Ustinov, and V. J. Vasudevan, "Reliability of solid-state drives based on NAND flash memory," *Proc. IEEE*, vol. 105, no. 1, pp. 1725–1750, Sep. 2017.
[3] S. Kim, H. Eom, and Y. Son, "Improving spatial locality in virtual machine for flash storage," *IEEE Access*, vol. 7, pp. 1668–1676, 2019.
[4] S. Park, Y. Kim, B. Urgaonkar, J. Lee, and E. Seo, "A comprehensive study of energy efficiency and performance of flash-based SSD," *J. Syst. Archit.*, vol. 57, pp. 354–365, Apr. 2011.
[5] D. Ma, J. Feng, and G. Li, "A survey of address translation technologies for flash memories," *ACM Comput. Surv.*, vol. 46, pp. 36-1–36-39, Jan. 2014.
[6] A. Fukami, S. Ghose, Y. Luo, Y. Cai, and O. Mutlu, "Improving the reliability of chip-off forensic analysis of NAND flash memory devices," *Digit. Invest.*, vol. 20S, pp. S1–S11, Mar. 2017.
[7] S. Lee, J. Kim, and A, Mithal, "Refactored design of I/O architecture for flash storage," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 70–74, Jan./Jun. 2015.
[8] Q. Xia and W. Xiao, "High-performance and endurable cache management for flash-based read caching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3518–3531, Dec. 2016.
[9] X. Chen, Y. Li, and T. Zhang, "Reducing flash memory write traffic by exploiting a few MBs of capacitor-powered write buffer inside solid-state drives (SSDs)," *IEEE Trans. Comput.*, vol. 68, no. 3, pp. 426–439, Mar. 2019.
[10] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1537–1550, May 2016.
[11] X. Liu, Y. Lu, J. Yu, and Y. Lu, "Optimizing read and write performance based on deep understanding of SSD," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 2607–2616.
[12] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," in *Proc. FAST*, Berkeley, CA, USA, 2008, pp. 16:1–16:14.
[13] 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.
[14] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst.*, Seoul, South Korea, 2006, pp. 234–241.
[15] P. Jin, Y. Ou, T. Härder, and Z. Li, "AD-LRU: An efficient buffer replacement algorithm for flash-based databases," *Data Knowl. Eng.*, vol. 72, pp. 83–102, Feb. 2012.
[16] S.-W. Lee, D.-J. Park, T.-S. Chung, D. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 3, pp. 1–27, Jul. 2007.
[17] D. Jung, J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "Superblock FTL: A superblock-based flash translation layer with a hybrid address translation scheme," *ACM Trans. Embedded Comput. Syst.*, vol. 9, pp. 1–41, Mar. 2010.
[18] S. Lee, D. Shin, Y. J. Kim, and J. Kim, "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, pp. 36–42, 2008.
[19] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 229–240, 2009.
[20] X. Y. Qi, X. Tang, Z. C. Liang, and X. Meng, "OAFTL: An efficient flash translation layer for enterprise application," *J. Comput. Res. Develop.*, vol. 48, no. 10, pp. 1918–1926, 2011.
[21] H. Kim and D. Shin, "Clustered page-level mapping for flash memory-based storage devices," *IEEE Trans. Consum. Electron.*, vol. 61, no. 1, pp. 47–55, Feb. 2015.
[22] R. Chen, Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "On-demand block-level address mapping in large-scale NAND flash storage systems," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 1729–1741, Jan. 2015.
[23] Q. Luo, R. C. C. Cheung, and Y. Sun, "Dynamic virtual page-based flash translation layer with novel hot data identification and adaptive parallelism management," *IEEE Access*, vol. 6, pp. 56200–56213, 2018.
[24] D. Lee, M. Shin, W. Choi, H. Roh, and S. Park, "MV-FTL: An FTL that provides page-level multi-version management," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 1, pp. 87–100, Jan. 2018.
[25] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and writes sequence reordering for flash memory," *IEEE Trans. Consum. Electron.*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.
[26] Z. Li, P. Jin, X. Su, K. Cui, and L. Yue, "CCF-LRU: A new buffer replacement algorithm for flash memory," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1351–1359, Aug. 2009.
[27] G. Xu, F. Lin, and Y. Xiao, "CLRU: A new page replacement algorithm for NAND flash-based consumer electronics," *IEEE Trans. Consum. Electron.*, vol. 60, no. 1, pp. 38–44, Feb. 2014.
[28] Z. Jiang, Y. Zhang, J. Wang, and C. Xing, "A cost-aware buffer management policy for flash-based storage devices," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, Cham, Switzerland, 2015, pp. 175–190.
[29] Y. Yuan, Y. Shen, W. Li, D. Yu, L. Yan, and Y. Wang, "PR-LRU: A novel buffer replacement algorithm based on the probability of reference for flash memory," *IEEE Access*, vol. 5, pp. 12626–12634, 2017.
[30] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha, "Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices," *IEEE Trans. Comput.*, vol. 58, no. 6, pp. 744–758, Jun. 2009.
[31] Y. Ou, T. Härder, and P. Jin, "CFDC: A flash-aware buffer management algorithm for database systems," *Advances in Databases and Information Systems*, vol. 6295. Berlin, Germany: Springer, 2010, pp. 435–449.
[32] UMASS. (2015). *OLTP Trace From UMass Trace Repository*. [Online]. Available: http://traces.cs.umass.edu/index.php/storage/storage
[33] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simulation*, Porto, Portugal, 2009, pp. 125–131.
[34] J. S. Bucy and G. R. Ganger, "The DiskSim simulation environment version 3.0 reference manual," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-03-102, 2003.

YINGBIAO YAO received the Ph.D. degree in information and communication engineering from Zhejiang University, Hangzhou, China, in 2006.

From 2008 to 2016, he was an Associate Professor with the School of Communication Engineering, Hangzhou Dianzi University, China. In 2011, he was a Visiting Scholar with RPI University, Troy, NY, USA. Since 2017, he has been a Professor with the School of Communication Engineering, Hangzhou Dianzi University. He is the author of three books, and over 50 articles and 30 inventions. His research interests include storage system design, wireless sensor networks, indoor localization, and media signal processing.

Dr. Yao was a recipient of the third level of Zhejiang Provincial "151" Talent for Excellence, in 2015, the First Class Prize of the Zhejiang Provincial Scientific and Technological Progress Award, in 2016, and the Zhejiang Provincial young and middle-aged academic leaders, in 2017.



XIAOCHONG KONG received the B.E. degree in information countermeasure techniques from Hangzhou Dianzi University, China, in 2018, where he is currently pursuing the M.E. degree. His current research interest includes file and storage systems with solid-state drive.

JIE ZHOU received the B.E. degree in communication engineering from Hangzhou Dianzi University, China, in 2016, where he is currently pursuing the M.E. degree. His current research interest includes file and storage systems with solid-state drive.

XIAORONG XU received the Ph.D. degree in signal and information processing from the Nanjing University of Posts and Telecommunications, Nanjing, China, in 2010. From 2011 to 2013, he was a Postdoctoral Researcher with the Institute of Information and Communication Engineering, Zhejiang University, Hangzhou, China. From 2013 to 2014, he was a Research Scholar with the Electrical and Computer Engineering Department, Stevens Institute of Technology, Hoboken, NJ, USA. He is currently an Associate Professor with the School of Communication Engineering, Hangzhou Dianzi University, Hangzhou, China, where he is also an Excellent Backbone Teacher. His research interests emphasize on wireless and mobile communications, signal processing, and storage system design.

WEI FENG received the B.E. degree in electronics and information engineering from Hubei Engineering University, China, in 2005, and the M.E. and Ph.D. degrees in communication and information systems from the South China University of Technology, China, in 2009 and 2014, respectively. Previously, from 2005 to 2006, she was an FAE with LITE-ON Technology Cooperation, Guangzhou, China. She is currently a Lecturer with Hangzhou Dianzi University. Her research interests emphasize on energy efficiency and physical layer security in future wireless communications.

ZHAOTING LIU received the M.S. degree in communication engineering from Dalian Maritime University, Dalian, China, in 2005, and the Ph.D. degree in electrical engineering from the Nanjing University of Science and Technology, Nanjing, China, in 2011. From 2005 to 2014, he was with Shaoxing University, Shaoxing. Since 2015, he has been with the School of Communication Engineering, Hangzhou Dianzi University, Hangzhou. His current research interests include radar signal processing, wireless sensor networks, and solid-state drives.