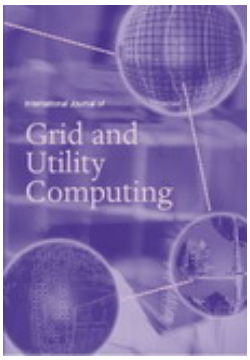



**International Journal of Grid and Utility Computing**

ISSN online: 1741-8488 - ISSN print: 1741-847X
https://www.inderscience.com/ijguc

# A page weight-based replacement algorithm to enhance the performance of buffer management in flash memory

Shweta, P.K. Singh

**DOI:** 10.1504/IJGUC.2023.10060590

**Article History:**

| | |
|---|---|
| Received: | 13 January 2023 |
| Last revised: | 02 April 2023 |
| Accepted: | 27 April 2023 |
| Published online: | 19 February 2024 |

# A page weight-based replacement algorithm to enhance the performance of buffer management in flash memory

Shweta and P.K. Singh

Computer Science and Engineering Department,
Madan Mohan Malaviya University of Technology,
Gorakhpur, Uttar Pradesh, India
Email: shweta20989@gmail.com
Email: topksingh@gmail.com
*Corresponding author

**Abstract:** Flash memory is used in various electronic handheld devices such as laptops and PDAs as secondary storage because of its excellent performance, low energy consumption, compact size, high-access speed and resistance to shock with growing density and lowering prices. However, the intrinsic properties, such as no in-place update and asymmetric I/O operations, provide challenges to designing buffer replacement strategies. This paper suggests an improved buffer management strategy called the Page Weight Buffer Replacement (PWBR) algorithm for flash memory, which considers buffers' page weight. An eviction approach is applied which tries to minimise the number of write counts and maintain a higher buffer hit rate by integrating recency, operational cost and temporal locality. Our finding shows PWBR is superior to existing buffers' management policies in terms of increasing the hit ratio of LRU-WSR, CF-LRU, CCF-LRU and AD-LRU by 9.3%, 6.4%, 3.7% and 2.5% higher, respectively.

**Keywords:** buffer replacement algorithm; frequency; recency; page migration.

**Reference** to this paper should be made as follows: Shweta and Singh, P.K. (2024) 'A page weight-based replacement algorithm to enhance the performance of buffer management in flash memory', *Int. J. Grid and Utility Computing*, Vol. 15, No. 1, pp.75–83.

**Biographical notes:** Shweta received her BTech degree in Computer Science and Engineering from the Purvanchal University Jaunpur, India and MTech degree in Computer Science and Engineering from Kamla Nehru Institute of Technology, Sultanpur, India. Currently, she is working as Research Scholar with Department of Computer Science and Engineering, Madan Mohan Malaviya University of Technology, Gorakhpur, India. Her current research interests include flash memory and multicore architectures.

P.K. Singh received his BE degree in Computer Science from the DDU University of Gorakhpur, Gorakhpur, India, MTech degree in Computer Science and Technology from University of Roorkee, Roorkee (now IITR) and PhD degree in Computer Science and Engineering from the DDU University of Gorakhpur. Presently, he is a Professor at Madan Mohan Malaviya University of Technology in Gorakhpur, India. His current research interests include memory and parallelism optimisation for storage systems, multi-core architectures and compiler design.

## 1 Introduction

SSDs based on NAND flash memory have low-power consumption, faster reading and writing speed, compact size, resistance to shock and noise-free performance which makes it suitable for storage systems (Du et al., 2019; Lawton, 2006). NAND flash memory capacity and cost are growing due to which an increasing number of consumer gadgets are now outfitted with flash memory as a secondary storage device (Mielke et al., 2017). The buffer management strategy can influence the sequence of access to the flash device, which affects the system performance. Many ways have been developed in the study on successful buffer replacement systems (Jiang et al., 2015; Butt et al., 2007; Javaid et al., 2017).

Many buffer management methods were presented by integrating recency and frequency which will aim to maximise buffer hit rates. Traditional buffer replacement techniques are intended for hard disc memory, attempting to improve hit ratio and hence I/O speed (Yao et al., 2019; Jia et al., 2018; Yuan et al., 2017). Operating systems employ buffer cache to store portions of disc blocks to decrease I/O requests. Because the cache buffer is smaller than the disc, I/O speed can be improved by buffer replacement schemes.

In recent decade, processor performance has improved quickly due to Moore's Law. The delayed performance increase of HDD-based storage systems has led to a growing performance gap between computation and storage. Compared to typical HDDs, flash memory features asymmetric I/O latencies, erase-before-write and restricted erase cycles, unlike hard disks (Wang and Wong, 2015). It's required to modify the flash memory buffer management algorithm. Flash memory differs from magnetic discs in numerous ways, including asymmetric I/O performance, write once and block erase count. Each flash memory block has a predetermined number of pages (Kim et al., 2018). Read/write operations takes place on the granularity of page, while erase operations is performed to block. Write/erase is slower than read. Write latency is substantially greater than read latency, while erase operation is much slower than write. Flash memory has durability issues since blocks are limited to being erased 10,000 to 100,000 times (Kobayashi et al., 2017). In other words, a block will get worn out and unreliable after receiving the fixed amount of erase operations. The buffer may support various storage layers' asymmetric I/O speeds. By integrating the buffer with disk drives, a hard drive or flash memory could be able to handle upper-layer demands more effectively and quickly. Buffer improves storage I/O performance. Traditional algorithms are developed for magnetic discs which cannot be used for flash memory (Liu et al., 2017). These techniques try to enhance buffer hit ratio while ignoring flash memory's asymmetric I/O performance. They don't work well with NAND flash memory storage systems. Traditional buffer management strategies are ideal for HDDs.

For flash memory buffer management system there is need of some modifications to cope up with above mentioned constraints.However, several limitations still limit SSDs performance. The first is its 'erase-before-write' operating technique (Wu et al., 2017; Chen et al., 2018). An SSD unit often has several blocks, each storing many pages. Page is the unit of write and read, whereas block is the unit of erase. Rewriting a single page requires erasing the whole block comprising it. Because block erasing takes time, it reduces SSD performance.

The second problem is SSDs endurance as the program/erase cycles are restricted to about 1,000,000 depending on NAND flash type. The concerns have been addressed extensively. FTL stands for Flash Translation Layer, which is used as an interface so that it can be easily used as secondary storage (Mittal and Vetter, 2015; Liu et al., 2017). Many buffer management techniques have constraints in successfully managing buffer space holding clean data since the eviction unit size does not correspond to the I/O request data size.

We suggest the Page Weight-based Buffer Replacement (PWBR) flash-based buffer management system for managing the buffer efficiently. The basic idea of PWBR is to evaluate the weight of page in buffer. The victim page from the buffer is selected on the basis of weight calculated. The access frequency, recency and operational cost of page is taken into consideration. PWBR page migration approach separates the buffer into two list based on access mode of page and frequency. The eviction method is modified to lower the write count and keep the buffer hit ratio high by integrating real-time eviction cost, page recency and access frequency.

The rest of this paper is structured as follows. In Section 2, we examine the characteristics of flash memory and existing literature on buffer management. A novel buffer management strategy for flash memory is presented in Section 3. In Section 4, we evaluate how effectively the proposed policy has worked. Section 5 concludes the research in its last paragraph.

## 2 Background

### *2.1 Related work*

DRAM is employed as the SSD buffer to reduce write operations. For maximum efficiency, the victim page selection from the buffer must be done efficiently. SSD buffer management techniques vary. Before rewriting, the target area must be erased. Read operations are faster than write/erase. LRU allows easy operation and a good hit ratio. Existing approaches concentrate on reducing write operations.Several techniques reduce write operations to improve I/O speed. Many algorithms avoid evicting dirty pages from buffer.The LRU page is selected as victim for eviction when the system requires more space. Flash memory writes cost more than reads, hence buffer replacement methods are used to decrease write operations.

To decrease flash write operations, the CFLRU (Park et al., 2006) used a clean-first eviction policy, which selects a clean page over a dirty page as the eviction victim. Many buffer management techniques use clean first evictions. To enhance buffer hit ratio, they often adopt a clean-first/cold-first eviction strategy. Several techniques reduce write operations to improve I/O speed. In addition to the wear-leveling degree and the number of erasure operations (Yoo et al., 2007) extends the original CFLRU method by incorporating asymmetric read/write costs into the algorithm design. In particular, CFLRU/C and CFLRU/E take into account the frequency of access and the quantity of block erasure operations, respectively. DL-CFLRU/E considerably increases the wear-leveling level of flash memory while decreasing the amount of erase operations.

LRU-WSR (Jung et al., 2008) intended to overcome CF-LRU drawbacks, it evaluates page hotness in buffer and retains hot dirty pages. It contains tag bit so that cold and clean pages are removed first. To decrease write requests to flash memory. Cold detection technique in LRU–WSR is designed to identify cold dirty pages. Each buffer page has a cold flag added. A dirty page that was selected as a target has the cold flag activated. If it isn't set, the page is relocated to the buffer's MRU position and then a new page is selected from the LRU. If activated, the page is saved to flash memory as the victim. No matter whether a page seems clean or not, it will be victimised. On visit, the buffer list dirty page is transferred to the MRU place and its cold tag is deleted.

Li et al. (2009) introduced a novel page replacement technique dubbed CCF-LRU. CCF-LRU maintains both the mixed and cold clean LRU page lists. Cold clean pages are the only ones on the mixed LRU page list. CCF-LRU first checks into the cold clean LRU page list, then evicts the pages in LRU order after that it scans the mixed LRU page list, identifying the most frequently referred page. It will activate the cold flag and shift the page to MRU inside the mixed list. In contrast to CFLRU, LRU-WSR not only postpones flushing dirty pages but also takes into account how frequently they are accessed. This enhances the overall I/O efficiency of the flash memory-based storage system and, in most instances, secures against serious deterioration of the buffer hit ratio. It buffers clean and dirty pages using two LRU lists.

AD-LRU (Jin et al., 2012) employs dynamic cold and hot areas to prevent evicting buffered clean pages. Pages on the cold list are removed whenever it reaches a certain size. Present flash-based approaches are far better than conventional buffer management policies, which don't use access pattern and page locality. Flash-based buffer management strategies may be optimised.

CLRU (Xu et al., 2014) manages clean and dirty page listings by LRU order. Both have cold-first and hot-first regions. By favourably removing cold pages from cold-first regions and taking into account their frequency of access, CLRU maximises page hit ratio. It also lowers write operations by postponing the removal of cold dirty pages effectively.

HDC (Lin et al., 2014) offers a replacement index to evict pages. This replacement index examines each page's hotness and the operation cost of target page to flash memory. It provides effective way to handle partial update in which migrates the dirty data of victim page to secondary flash memory. This reduces the cost of write operations while simultaneously improving the dependability of flash memory.

**Table 1** Comparison of buffer replacement algorithms

| *Algorithm* | *Granularity* | *Replacement technique* |
|---|---|---|
| CFLRU | Page | It evicts preferentially clean pages. |
| LRU-WSR | Page | It firstly evicts cold dirty pages. |
| CCF-LRU | Page | Pages are evicted in order of cold clean, hot clean and hot dirty, respectively. |
| CLRU | Page | There is high probability of evicting the least-referenced page. |
| ADLRU | Page | LRU queue in situations where the size of the cold area does not meet the minimum required for lc. |
| FAB | Block | Block of large size. |
| BPLRU | Block | BPLRU handles write request for buffer memory. |
| | | It redirects reads to the FTL. |
| CFDC | Block | Clean pages are prioritised for replacement, while blocks of dirty pages are used to write back. |

PT-LRU (Cui et al., 2014) preferentially evicts cold clean pages over hot clean pages. In case the list is empty, pages which are cold dirty are more likely to be evicted from the buffer than hot clean pages. By emphasising the eviction of the cold clean pages, PT-LRU postpones the removal of hot clean pages. Cold-page detection is used to prevent permanently retaining dirty pages with in buffer.

## 3 Proposed architecture of PWBR approach

### *3.1 Proposed approach*

The PWBR is designed to enhance the performance of buffer by delaying the eviction of pages based on their weight which tends to improve the buffer hit ratio. In PWBR pages are classified into four categories: hot clean $(C_H)$, cold dirty $(D_C)$ cold clean $(C_C)$ and hot dirty $(D_H)$. A page is referred to as a hot page if it is accessed two or more times; otherwise, it is known as a cold page. If the pages operation mode is read it is clean in nature otherwise it is dirty. Pages in the buffer are differentiated by PWBR based on their weight, which is determined by the frequency, recency and cost of operation.

Figure 1 depicts the buffer management of the proposed PWBR approach, which includes the working list and hold list. Working list and hold list sizes are L1 and L2, respectively, while the total buffer size. L is the sum of L1 and L2. Figure 1 shows the process of hit and miss occurrence in PWBR when a request arrives, it is structured and placed in a First-In-First-Out (FIFO) queue. The following steps takes place when request is arrived:

- It is ensured that the requests are processed in the order that they were received.
- The request is sent through the hash function to see if there is a match in hash bucket.
- If there is a hit, the buffer will be updated to reflect the changes made to the pages in either the working list or the hold list.
- Upon miss, the page which is requested is taken from memory and the buffer is updated.
- At last, the evicted buffer page is relocated to flash memory.

### *3.2 Page weight (PW) algorithm and page migration policy*

The page migration in buffer is done within working list and hold list and victim is selected is depicted in Figure 2. Hold list follows the LRU policy whereas working list obeys page migration according to PW algorithm. Suppose the refrence of page is denoted by *R* the sequence *Q* can be represented as $Q = R_1, R_2, ..., R_x, ..., R_u \quad 1 < x < u$ where $x$ is page sequence number and *u* represents total refrenced amount.

**Figure 1** Buffer management in PWBR

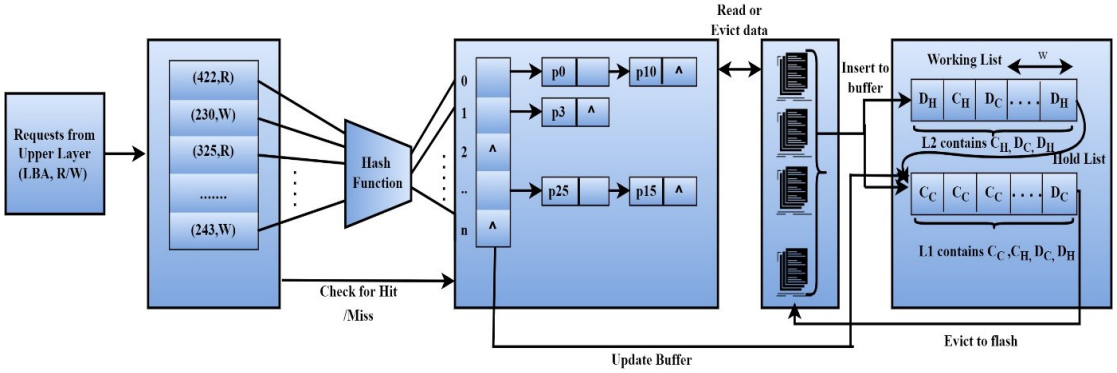
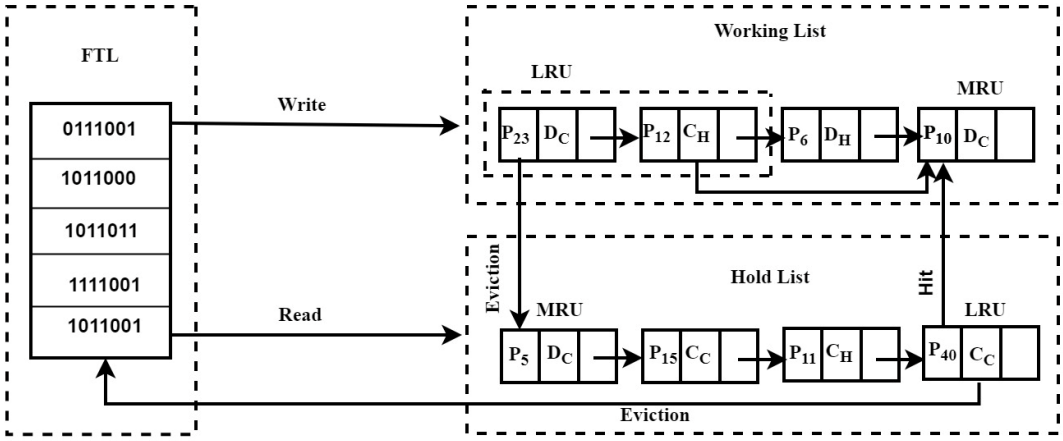



**Figure 2** Page migration in PWBR




To determine the weight of the page, we will compute it based on the following four definitions. The pages access frequency is provided in Definition 1. Definition 2 defines recency of page. Definition 3 compute the cost of operation performed on page. In Definition 4 the weight of page is calculated by integration of above definitions.

Definition 1: *Page Frequency is determined as number of time page is accessed in certain time period. The access frequency is calculated by considering hit count and age count. The time difference between the present time and the first time page was accessed indicates the age of a page. If page i's initial access time is ti and the current time is tc, then access frequency of page is evaluated as*

$$APi = tc - ti$$

The second required information is the page's write hit count in the buffer.The hit count starts at 1 when a page is allocated, rises by 1 with each write hit and is cleared when flash memory is flushed. Table 2 shows the Hit and Age count of pages in buffer at different instances.

**Table 2** Hit count(Hc) and age count(Ac) of pages in buffer at different instances

| | *T1* | *T2* | *T3* | *T4* | *T5* |
|---|---|---|---|---|---|
| *Page* | *P* | *P Q* | *P Q R* | *Q R P* | *Q R S T* |
| $A_c$ | 1 | 2 1 | 3 2 1 | 3 2 1 | 4 3 2 1 |
| $H_c$ | 1 | 1 1 | 1 1 1 | 1 1 2 | 1 1 2 1 |

When a new page comes to buffer, the value of age-count is initialised to 1 and for all existing pages in buffer the age counts is incremented by 1. When a page is changed, its age-count is refresh to 1, and for others that has a lower age count than the updated page is raised by 1. Other pages with high value of age-count remains unaltered. Any update to a page with the value of Age-count 1 will leave all buffer pages' age-count values unaffected. When a page gets evicted from buffer, pages with a greater age-count will decremented by 1 whereas remaining pages age count remains unaffected. For example, as shown in Table 2 hit count and age count at various time instances, i.e., *Tx* represents the state of buffer at time *x*. At T1 there is only one page P in buffer, at T2 Q is added so it's age count becomes 1 whereas the age count of P is incremented by 1, i.e., 2. At T4, page P's age count is reinitialised to 1 and Q and R's are increased by 1, i.e., 3 and 2, respectively. Therefore, access frequency is calculated using:

$$AF(x) = \frac{H_c(x)}{A_c(x)} \tag{1}$$

Definition 2: *The term $R_c$ (Recency) is described as the number of distinct pages that have been referred between lastest access of page $R_{LT}$ and the last page access $R_L$. A page has a higher chance of being re-visited the more recently it has been seen, the recency is calculated by recent access and current access time of page.Pages with a lower*

*freshness should be discarded first because recency indicates how recent page is*.

$$R_c = \frac{R_{LT} - R_L}{R_{LT}}, R_L \le R_{LT} \tag{2}$$

Definition 3: *The cost of operation* $\theta$ *defines the operation of write and read from memory. Suppose* $R^l = \{R_1^l, R_2^l, \ldots, R_f^l, \ldots, R_g^l\}$ *and* $W^l = \{W_1^l, W_2^l, \ldots, W_y^l, \ldots, W_g^l\}$ *are the current* $g$ *latencies of read and write operations in flash memory*.

$$\theta = \begin{cases} \dfrac{\sum_{f=1}^{g} R_f^1}{\sum_{f=1}^{g} R_f^1 + \sum_{g=1}^{g} W_y^1}, & pages \in \{C_C, H_H\} \\ \dfrac{\sum_{g=1}^{g} W_y^1}{\sum_{f=1}^{g} R_f^1 + \sum_{y=1}^{g} W_y^1}, & pages \in \{D_C, D_H\} \end{cases} \tag{3}$$

Unlike other LRU-based algorithms, $\theta$ is adaptable based on the write/read cost ratio of a particular memory device. Utilising $\theta$, the overall latency may be decreased while taking into account the various features of flash memory.

Definition 4: *The weight of page is evaluated using three previously mentioned arguments*:

$$weightofpage = \frac{AF * \theta}{R_c} \tag{4}$$

The aforementioned formula is used to compare pages within the working list's window. The ultimate eviction decision is based on page weight, and the page having least weight is evicted from working to hold list.

### *3.3 Workflow of PWBR and description of algorithm*

The workflow of PWBR algorithm is described in algorithm1. If there is a request for page access, the hash function determines if that page exists in the buffer. Hit occurs if that page is already in buffer; otherwise, it is miss.The page is moved to the working list's Most Recently Used (MRU) position if hit happens in the working list.Whenever hit take place in hold list or there is miss the subsequent step is to find whether hold or working list is full. The Page Weight (PW) algorithm is applied if the required list is full.

After removing the victim page, the requested page is added to the specified list. Depending on the circumstances, the eviction may go somewhere different. If hit happens in the hold list, the victim page is relocated to the MRU position of the hold list (lines 11–13). The victim page is shifted to the MRU of the hold list whenever there is a miss in the working list, and the hold list's LRU is moved to flash memory (lines 29–32). Whenever there is miss in hold list, eviction of victim page is done to secondary flash memory (lines 21–23). The eviction algorithm's operation is illustrated by Algorithm 2. The algorithm determines the window's page weight (lines 8–11). The mentioned three definitions are used to calculate weight of page (line 9). A weight map is used to keep each page's weight, and the page having least weight is selected as the victim page as hown in (line 12–17).

**Algorithm 1:** PWBR Algorithm

1: **Initialise** $P_R$ : Requested page, $P_M$ : Access mode of page, WL: Working List, HL: Hold List;
2: **Output** Reference of $P_R$ ;
3: **if** $P_R$ belongs to buffer, then;
4: **if** $P_R \in$ WL **then**
5: Migrate $P_R$ to MRU of WL
6: **else**
7: **if** $WL_{Size} < WL_{Size} - Max$ **then**
8: Add $P_R$ to MRU of WL
9: **else**
10: $V_P \leftarrow$ Evict(WL)
11: Add $P_R$ into MRU of WL
12: Add $V_P$ to MRU of HL
13: **end if**
14: **end if**
15: **else**
16: **if** $P_M == R$ **then**
17: **if** $HL_{Size} < HL_{Size} - Max$ **then**
18: Add $P_R$ into MRU of HL
19: **else**
20: $V_P \leftarrow$ Evict(HL)
21: Add $P_R$ to MRU of HL
22: Remove the $V_P$ to secondary memory
23: **end if**
24: **else**
25: **if** $WL_{Size} < WL_{Size} - Max$ **then**
26: Add $P_R$ to MRU of WL;
27: **else**
28: $V_P \leftarrow$ Evict(WL)
29: Add $P_R$ to MRU of WL
30: Add $V_P$ to MRU of HL
31: Remove LRU of HL to secondary storage
32: **end if**
33: **end if**
34: **end if**
35: **return** $P_R$ .

### *3.4 Complexity of PWBR*

The space and time complexity of the PWBR algorithm is described in this subsection. The eviction algorithm is the main cause of time complexity (see in Algorithm 2). The page weight must be determined for pages in eviction window. In the hold zone, the eviction window's size is fixed, hence the eviction method has O(1) time complexity. As a result, the PWBR algorithm's overall time complexity is O(1). To keep track of each page's recency and temporal locality interval PWBR employs extra data structures that

reduce space complexity. As this extra space is constant so for every page belonging to buffer the complexity is O(1).

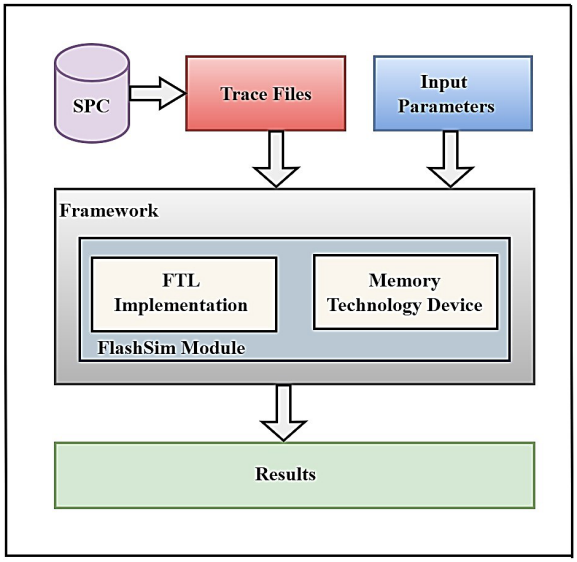
**Algorithm 2:** Algorithm for eviction

1: **Initialise** $P$: page, $S_R$: Selected region, WL: Working List, HL: Hold List, $W_P$: page weight: $M_W$: Weight map ;
2: **Output** Reference of $V_P$ ;
3: **if** $S_R$ belongs to HL, **then**;
4: Apply LRU in HL of LRU list;
5: **return** LRU reference of HL;
6: **else**
7: Execute algorithm1 in WL;
8: **for** $P \in [0, w]$ of WL **do**
9: $W_P \leftarrow \text{PW}(P)$;
10: $M_W$ .put $\left(P, W_p\right)$;
11: **end for**
12: **for** P $M_W$ **do**
13: **if** $P <$ $W_{\text{minimum}}$ **then**
14: $V_P \leftarrow$ P
15: $W_{\text{minimum}} \leftarrow M_W$ .get($P$);
16: **end if**
17 **return** $V_P$
18: **end if**

## 4 Performance evaluation

The tests are carried out with the assistance of the simulation platform known as Flash-DBSim (Su et al., 2009). It is one of the most widely used open-source simulator for flash memory. FlashDBSim's architecture incorporates modules such as Flash translation layer and Memory technology device. Each module's content may be modified to meet research requirements. The framework for simulator is shown in Figure 3.

**Figure 3** Framework of simulator (see online version for colours)




The experimental parameters used in simulator are shown in Table 3.

**Table 3** Parameters

| *Parameter* | *Description* |
|---|---|
| Size of Page (KB) | 2 |
| Size of Block (KB) | 64 |
| Page Read (µs) | 20 |
| Block Erase (µs) | 1500 |
| Page Write (µs) | 200 |

Three metrics are analysed in this simulation which are runtime, hit ratio and write count. Table 4 provides a detailed description of the two synthetic traces which are gathered by Zipfian distribution, with the read-and-write proportion showing the percentage of total read and write requests .The locality indicates the number of operations carried out on a significant number of pages.From read-intensive to write-intensive, there are two synthetic traces, each with 300,000 requests. A storage system's workload may be inferred from varying ratios of all activities, and access patterns of distinct upper-layer applications can be seen in combination at different locations.The reference localisation '80%/40%' indicates that 80% of the references are densely done in 20% of the pages.

**Table 4** I/O synthetic trace characteristics

| *Trace* | *Total Request* | *Reference Locality (%)* | *Read or Write* |
|---|---|---|---|
| T1 | 300,000 | 80/20 | 70%/30% |
| T2 | 300,000 | 80/20 | 90%/10% |

The second category of traces includes the real-world traces Financial1 shown in Table 5. that was generated by recording OLTP application requests at a major financial institution which were made accessible by Storage Networking Industry Association (Umass.edu, 2018). These traces T1 & T2 are often used for performance evaluation of storage systems. Analysis and comparison of the simulation's outcomes with those from various buffer replacement methods, which are referred to as LRU-WSR, CCF-LRU, CFLRU and AD-LRU correspondingly.

**Table 5** I/O real world trace characteristics

| *Trace* | *Total Request* | *Read Ratio (%)* | *Write Ratio (%)* |
|---|---|---|---|
| Financial 1 | 5,334,987 | 23.2 | 76.8 |

Figures 4, 5 and 6 show hit rates for five buffer replacement algorithms. The hit ratio increases with a larger buffer, but the growth rate slows. Among other four policies PWBR has better hit rate because it takes into consideration of weight of page which is evaluated using frequency and recency. Hence, the active pages can be stayed longer into buffer showing highest hit ratio compared to others policies. The number of

hit ratio for trace T1 at buffer size 3 MB is calculated to compare it with those of CF-LRU, LRU-WSR, CCF-LRU and AD-LRU and it was found that PWBR performs better than the existing approaches and increases the hit rate by 9.3%, 6.4%, 3.7% and 2.5% higher, respectively.

**Figure 4** Hit ratio for T1 (see online version for colours)

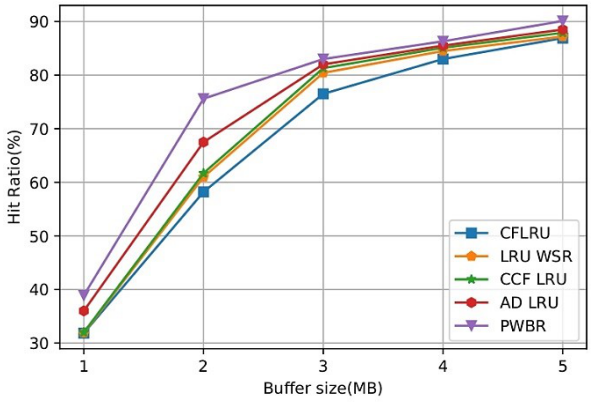



**Figure 5** Hit ratio for T2 (see online version for colours)

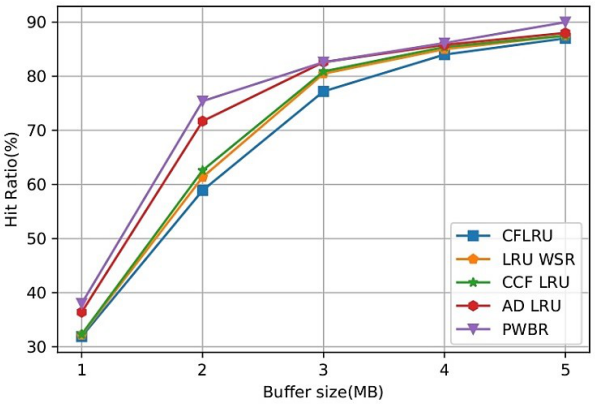
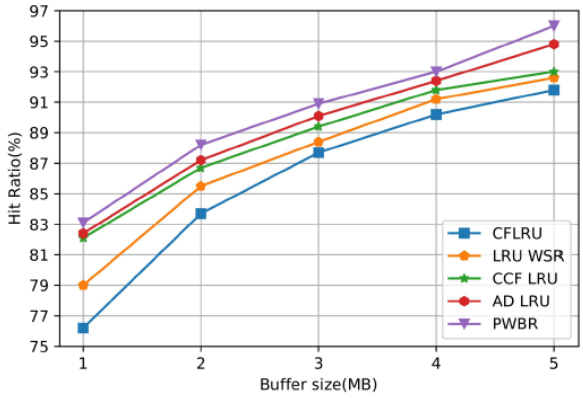



**Figure 6** Hit ratio for Financial 1 (see online version for colours)




Figures 7, 8 and 9 represent algorithm runtimes of five buffer replacement policies for various buffer sizes PWBR has minimal runtime. Asymmetric I/O latencies cause small discrepancies in flash memory write operations. It makes dirty pages more significant and prolongs their persistence. Therefore, buffer replacement methods are designed to reduce writes. Runtime falls slower than memory time when buffer capacity is big. In comparison to AD-LRU, LRU-WSR, CCF-LRU and CFLRU the proposed PWBR decreases the runtime of T1 for buffer size 3 MB by 50.2%, 41.8%, 38.2% and 14.1%.

**Figure 7** Runtime for T1 (see online version for colours)

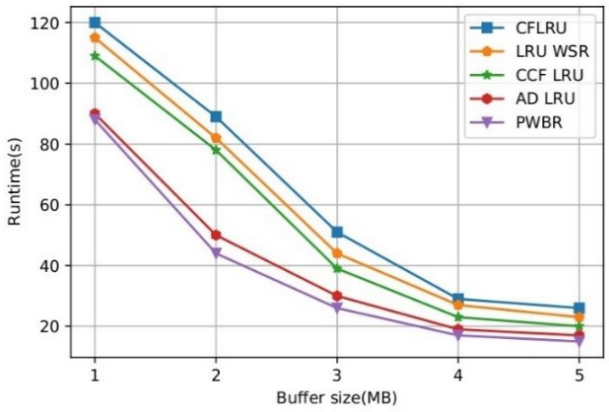



**Figure 8** Runtime for T2 (see online version for colours)

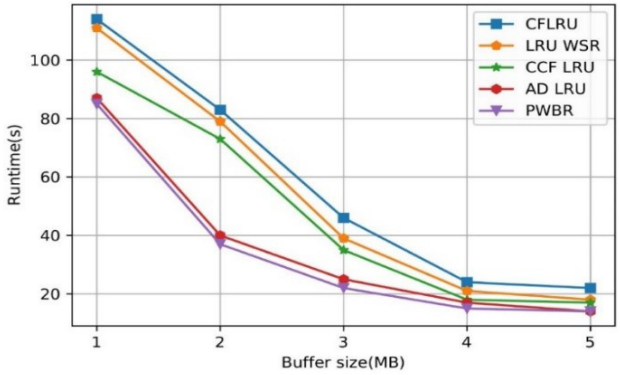
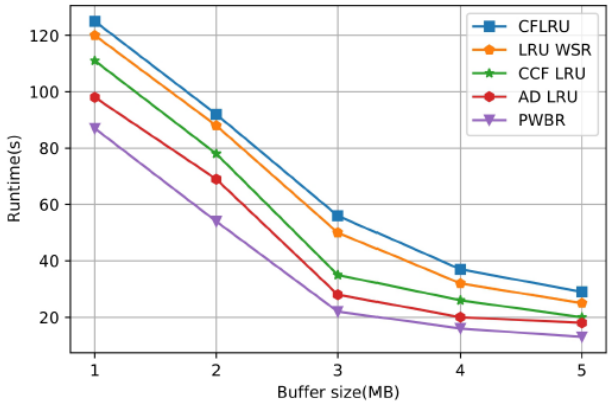



**Figure 9** Runtime for Financial 1 (see online version for colours)




Figures 10, 11 and 12 demonstrate how buffer size affects algorithm write counts. Since the pages having frequency and recency are stayed longer in buffer it can be seen, PWBR only requires a small amount of flash write operation to flash storage system. As the write operations decreases the lifetime of flash memory increases. Since it keeps dirty pages longer in buffer so there is probability of high-hit ratio and lesser

amount of write operation. For PWBR the write count for trace T1 at buffer size 3 MB is reduced by 48.4%, 42.2%, 31.5% and 10.7% compared by LRU-WSR, CFLRU, CCF-LRU and AD-LRU, respectively.

**Figure 10** Write count for T1 (see online version for colours)

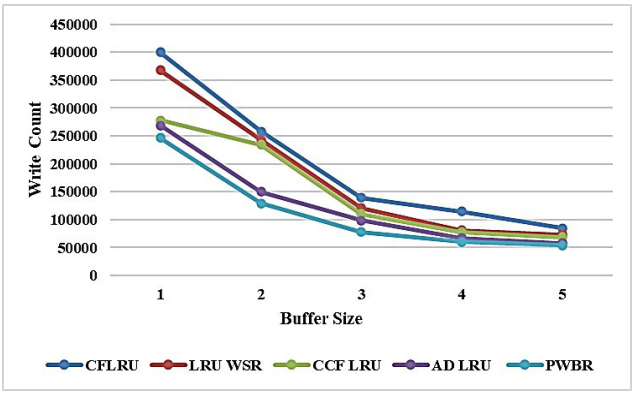



**Figure 11** Write count for T2 (see online version for colours)

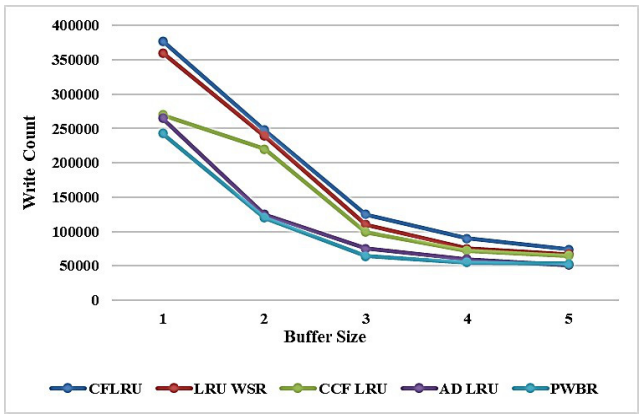
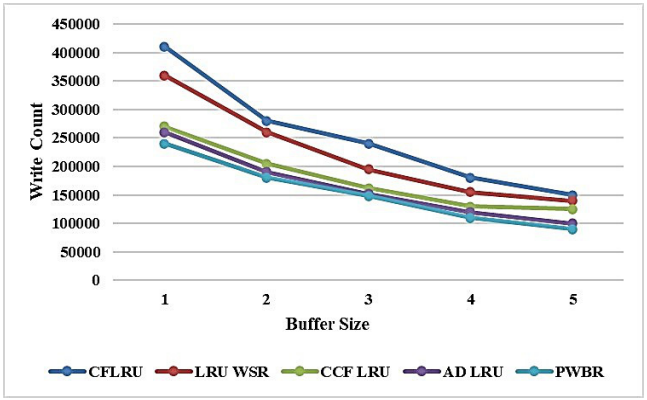



**Figure 12** Write count for Financial 1 (see online version for colours)




## 5 Conclusions

A flash memory buffer's efficiency improves system performance. To tackle the shortcomings of current algorithms, we presented PWBR, a page weight buffer replacement algorithm that uses a page's weight to determine how long it may remain in the buffer.Buffer is partitioned into working and hold list which guarantee the efficiency and accuracy of buffer. Meantime, pages in the buffer are classified into different kinds based on their access method and frequency, which have a significant impact on the hit ratio. PWBR introduces a evolutionary page migration mechanism to improve buffer efficiency over existing LRU-based replacement methods. An eviction algorithm is designed for reducing the latency by integrating recency, temporal locality and operational cost of page. Two synthetic traces and one real world trace were taken for simulation, and tests results revealed that the suggested PWBR method provides performance superior to that of current techniques in terms of total runtime, flash operations and buffer hit ratio. We simulated PWBR and other algorithms using the Flash-DBSim simulator. PWBR is more efficient than other buffer management strategies in context of hit rate, write count and overall runtime. The number of hit ratio for trace T1 at buffer size 3 MB is calculated to compare it with LRU-WSR, CF-LRU, CCF-LRU and AD-LRU and it was found that PWBR performs better than the existing approaches and increases the hit rate by 6.4%,9.3%, 3.7% and 2.5% higher, respectively.

Our ongoing research will concentrate on the use of the PWBR algorithm in many contexts. In particular, a database engine and DBMS will be connected with the PWBR technique. Our future study will also encompass the PWBR algorithm's implementation in big data analysis in various IoT applications.

## References

Butt, A.R., Gniady, C. and Hu, Y.C. (2007) 'The performance impact of kernel prefetching on buffer cache replacement algorithms', *IEEE Transactions on Computers*, Vol. 56, No. 7, pp.889–908.

Chen, X., Li, Y. and Zhang, T. (2018) 'Reducing flash memory write traffic by exploiting a few mbs of capacitor-powered write buffer inside solid-state drives (ssds)', *IEEE Transactions on Computers*, Vol. 68, No. 3, pp.426–439.

Cui, J., Wu, W., Wang, Y. and Duan, Z. (2014) 'Pt-Lru: a probabilistic page replacement algorithm for nand flash-based consumer electronics', *IEEE Transactions on Consumer Electronics*, Vol. 60, No. 4, pp.614–622.

Du, C., Yao, Y., Zhou, J. and Xu, X. (2019) 'Vbbms: a novel buffer management strategy for nand flash storage devices', *IEEE Transactions on Consumer Electronics*, Vol. 65, No. 2, pp.134–141.

Javaid, Q., Zafar, A., Awais, M. and Shah, M.A. (2017) 'Cache memory: an analysis on replacement algorithms and optimization techniques', *Mehran University Research Journal of Engineering and Technology*, Vol. 36, No. 4, pp.831–840.

Jia, G., Han, G., Wang, H. and Wang, F. (2018) 'Cost aware cache replacement policy in shared last-level cache for hybrid memory based fog computing', *Enterprise Information Systems*, Vol. 12, No. 4, pp.435–451.

Jiang, Z., Zhang, Y., Wang, J. and Xing, C. (2015) 'A cost-aware buffer management policy for flash-based storage devices', *Proceedings of the 20th International Conference Database Systems for Advanced Applications*, 20–23 April, Springer, Hanoi, Vietnam, pp.175–190.

Jin, P., Ou, Y., Härder, T. and Li, Z. (2012) 'Ad-Lru: an efficient buffer replacement algorithm for flash-based databases', *Data and Knowledge Engineering*, Vol. 72, pp.83–102.

Jung, H., Shim, H., Park, S., Kang, S. and Cha, J. (2008) 'Lru-Wsr: integration of lru and writes sequence reordering for flash memory', *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 3, pp.1215–1223.

Kim, S., Eom, H. and Son, Y. (2018) 'Improving spatial locality in virtual machine for flash storage', *IEEE Access*, Vol.7, pp.1668–1676.

Kobayashi, K.M., Miyazaki, S. and Okabe, Y. (2017) 'Competitive buffer management for multi-queue switches in qos networks using packet buffering algorithms', *Theoretical Computer Science*, Vol. 675, pp.27–42, 2017.

Lawton, G. (2006) 'Improved flash memory grows in popularity', *Computer*, Vol. 39, No. 1, pp.16–18.

Li, Z., Jin, P., Su, X., Cui, K. and Yue, L. (2009) 'Ccf-Lru: a new buffer replacement algorithm for flash memory', *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 3, pp.1351–1359.

Lin, M., Chen, S., Wang, G. and Wu, T. (2014) 'Hdc: an adaptive buffer replacement algorithm for nand flash memory-based databases', *Optik*, Vol. 125, No. 3, pp.1167–1173.

Liu, D., Yao, L., Long, L., Shao, Z. and Guan, Y. (2017) 'A workload-aware flash translation layer enhancing performance and lifespan of tlc/slc dual-mode flash memory in embedded systems', *Microprocessors and Microsystems*, Vol. 52, pp.343–354.

Liu, X., Lu, Y., Yu, J. and Lu, Y. (2017) 'Optimizing read and write performance based on deep understanding of SSD', *Proceedings of the 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp.2607–2616.

Mielke, N.R., Frickey, R.E., Kalastirsky, I., Quan, M., Ustinov, D. and Vasudevan, V.J. (2017) 'Reliability of solid-state drives based on nand flash memory', *Proceedings of the IEEE*, Vol. 105, No. 9, pp.1725–1750.

Mittal, S. and Vetter, J.S. (2015) 'A survey of software techniques for using non-volatile memories for storage and main memory systems', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, No. 5, pp.1537–1550.

Park, S-Y., Jung, D., Kang, J-U., Kim, J-S. and Lee, J. (2006) 'Cflru: a replacement algorithm for flash memory', *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp.234–241.

Su, X., Jin, P., Xiang, X., Cui, K. and Yue, L. (2009) 'Flash-dbsim: a simulation tool for evaluating flash-based database algorithms', *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology*, IEEE, pp.185–189.

Umass.edu (2018) *I/O Traces From UMass Trace Repository*, Storage Networking Industry Association. Available online at: http://traces.cs.umass.edu/index.php/Storage/Storage (accessed on 15 March 2018).

Wang, C. and Wong, W-F. (2015) 'Treeftl: an efficient workload-adaptive algorithm for ram buffer management of nand flash-based devices', *IEEE Transactions on Computers*, Vol. 65, No. 8, pp.2618–2630.

Wu, C-H., Wu, D-Y., Chou, H-M. and Cheng, C-A. (2017) 'Rethink the design of flash translation layers in a component-based view', *IEEE Access*, Vol. 5, pp.12895–12912.

Xu, G., Lin, F. and Xiao, Y. (2014) 'Clru: a new page replacement algorithm for nand flash-based consumer electronics', *IEEE Transactions on Consumer Electronics*, Vol. 60, No. 1, pp.38–44.

Yao, Y., Kong, X., Zhou, J., Xu, X., Feng, W. and Liu, Z. (2019) 'An advanced adaptive least recently used buffer management algorithm for ssd', *IEEE Access*, Vol. 7, pp.33494–33505.

Yoo, Y-S., Lee, H., Ryu, Y. and Bahn, H. (2007) 'Page replacement algorithms for nand flash memory storages', *Proceedings of the International Conference on Computational Science and its Applications*, Springer, pp.201–212.

Yuan, Y., Shen, Y., Li, W., Yu, D., Yan, L. and Wang, Y. (2017) 'Pr-lru: a novel buffer replacement algorithm based on the probability of reference for flash memory', *IEEE Access*, Vol. 5, pp.12626–12634.