LETTER

LAB-LRU: A Life-Aware Buffer Management Algorithm for NAND Flash Memory*

Liyu WANG[†a)], ***Member***, **Lan CHEN**[†b)], ***and*** **Xiaoran HAO**[†], ***Nonmembers***

SUMMARY NAND flash memory has been widely used in storage systems. Aiming to design an efficient buffer policy for NAND flash memory, a life-aware buffer management algorithm named LAB-LRU is proposed, which manages the buffer by three LRU lists. A life value is defined for every page and the active pages with higher life value can stay longer in the buffer. The definition of life value considers the effect of access frequency, recency and the cost of flash read and write operations. A series of trace-driven simulations are carried out and the experimental results show that the proposed LAB-LRU algorithm outperforms the previous best-known algorithms significantly in terms of the buffer hit ratio, the numbers of flash write and read operations and overall runtime.

key words: *NAND flash memory, storage system, buffer management algorithm, life value*

1. Introduction

NAND flash memory possesses the advantages of high speed, low power consumption and shock resistance [1]. Therefore, it has been widely applied in storage systems [2]. Whereas, flash memory also has some inherent defects, such as asymmetric I/O latencies of read and write operations, erase-before-write and limited erase cycles [3], which are different from disk. Thus, it is necessary to redesign the buffer management algorithm for flash memory.

Although existing buffer management algorithms for flash memory are capable of distinguishing between write and read operations, they still have their own drawbacks. The main ideas along with the drawbacks of them will be given in next section. Besides these, the common drawback is that they seldom consider different access frequency and recency in hot region, where they treat pages equally no matter how many times and how recently pages have been accessed.

In this paper, we propose a life-aware buffer management algorithm named LAB-LRU. It combines the access frequency, recency and the cost of flash operation to define a life value for every page, which decides how long the page can stay in the buffer. The experimental results show that LAB-LRU can clearly improve hit ratio, reduce flash write and read operations, and shorten runtime.

Manuscript received March 15, 2016.
Manuscript revised May 26, 2016.
Manuscript publicized June 21, 2016.

[†]The authors are with the Institute of Microelectronics, Chinese Academy of Sciences, Beijing, China.

*This work was supported in part by the National Science and Technology Major Project of China under grant 2013ZX03001008-003.

a) E-mail: wangliyu@ime.ac.cn
b) E-mail: chenlan@ime.ac.cn

DOI: 10.1587/transinf.2016EDL8062

2. Related Work

LRU (Least Recently Used) is the basic buffer algorithm which gives priority to evict the least recently used pages. Actually, it is very difficult to record and judge the least recently used pages in practical scenarios. Traditional LRU algorithm links all buffer pages into one list depending on their recency. The newly accessed pages will be linked to the tail of the list, and the eviction begins from the head of the list. However, it does not consider the access frequency and distinguish between the write and read operations for flash memory.

CFLRU [4] is the first buffer management algorithm designed for flash memory, which considers the asymmetric I/O latencies of flash read and write operations. The buffer is divided into the working region and clean-first region. CFLRU always evicts clean pages preferentially, then evicts dirty pages when there are no pages in the clean-first region. Therefore, it can reduce flash write and erase operations. Nevertheless, CFLRU has some obvious shortcomings: (1) The preferential eviction of clean pages leads to evict newly accessed clean pages too early and let cold dirty pages stay too long in the buffer. (2) It has to scan the region to find clean pages, which increases system time.

LRU-WSR [5] considers the access frequency of dirty pages compared to CFLRU. It prefers to evict clean pages and cold dirty pages to prevent cold dirty pages from staying too long in the buffer. Hot dirty pages get the second chance by setting to cold. LRU-WSR performs better than CFLRU, but it does not consider the access frequency of clean pages, this also causes hot clean pages to be evicted too early lowering the hit ratio.

CCF-LRU [6] differentiates cold and hot pages among clean pages based on the above algorithms. It splits the buffer into Cold Clean LRU List (CCL) and Mixed LRU List (ML). CCL stores cold clean pages which will be evicted preferentially. Then hot clean and dirty pages in ML will be selected as victims when CCL is empty. However, CCF-LRU cannot control the size of CCL. With the eviction of cold clean pages, CCL will shrink and even become empty. Then when a newly read page is added to CCL, it will be evicted very early. Which in turns lowers the hit ratio and system performance.

AD-LRU [7] tries to solve the problem of CCF-LRU by adjusting the size of cold list. It divides the buffer into cold

LRU list and hot LRU list. Cold list holds the clean and dirty pages which have been accessed only once, and hot list stores the pages which have been accessed more than once. AD-LRU sets a minimal size threshold *min_lc* for the cold list. Cold clean pages in the cold list are evicted firstly. When there are no cold clean pages, if the size of cold list is more than *min_lc*, the cold dirty pages from the cold list will be selected as victims. Otherwise, the pages in the hot list will be evicted. Unfortunately, giving priority to evict cold clean pages will gradually let cold dirty pages occupy the cold list, which causes the same problem like CCF-LRU.

APB-LRU [8] splits the cold list into cold clean list and cold dirty list. It adopts a fixed probability based on the cost of flash write and read operations to choose from cold clean list or cold dirty list. It has two disadvantages: (1) The fixed eviction probability is not flexible for various applications with different read/write ratios. A dirty page may be evicted even if there are still many cold clean pages. (2) Pages moving from the hot list to cold list all get the same treatment without considering the access frequency and recency.

PT-LRU [9] has three LRU lists like APB-LRU. It first evicts cold clean pages. When the cold clean pages are not enough, the eviction will choose between the cold dirty pages and hot clean pages according to a certain probability. PT-LRU also has some problems: (1) It does not solve the second problem of APB-LRU; (2) The best probability is obtained only through four synthesized traces which cannot represent all application scenarios.

3. The LAB-LRU Algorithm

3.1 The Structure of LAB-LRU Algorithm

In order to index pages quickly, a hashtable which supports the insert, query and delete operations is used to manage the buffer. Unlike CFLRU and LRU-WSR, LAB-LRU considers the access frequency, and differentiates cold pages and hot pages. Furthermore, to reduce the time overhead of scanning the cold region to get a cold clean page, the cold region is divided into two parts depending on clean and dirty pages. Therefore, LAB-LRU splits the buffer into three LRU lists: an inactive clean list, an inactive dirty list and an active mixed list. As an example in Fig. 1, data A is first written by an application and added to the inactive dirty list. Data B is newly read from flash and stored in the inactive clean list. When A or B is accessed again, it will be relived to the active mixed list (like A' or B'). By contrast, the pages with low life value in active mixed list will be dispatched to inactive lists.

3.2 The Eviction Strategy

When the usage of buffer size exceeds the threshold *TV* which is estimated as seven eighths of buffer size, the dead pages will be evicted in the background. The eviction process is controlled by double thresholds *INV_MAX* and *INV_MIN* which are estimated as three fourths and one

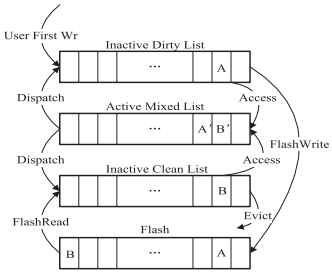

Fig. 1 The flow diagram of data in three lists.

fourth of inactive list sizes. The reason using two thresholds for each inactive list is to balance the eviction of cold clean and cold dirty pages. This can prevent the newly read clean pages from being evicted too soon, and avoid cold dirty pages staying too long. The eviction process is described as follows:

(1) If the size of inactive clean list is beyond *INV_MAX*, the inactive clean pages from the head of inactive clean list will be evicted and discarded directly.

(2) If the size of inactive clean list is less than *INV_MAX* and the buffer size is still more than *TV*. If the size of inactive dirty list is beyond *INV_MAX*, the inactive dirty pages will be evicted and written back to flash memory.

(3) If the size of inactive dirty list is less than *INV_MAX* and the buffer size is still more than *TV*. If the size of inactive clean list is beyond *INV_MIN*, the clean pages in the inactive clean list will be evicted again until its size meets *INV_MIN*. Otherwise, if the size of inactive dirty list is more than *INV_MIN*, the inactive dirty pages will be evicted in the same way.

(4) If both of the inactive lists are less than *INV_MIN* and the buffer size is still more than *TV*, the eviction strategy will evict directly from the head of the active mixed list. It picks out the pages with low life value like the dispatch strategy. When buffer size is down below *TV*, the eviction process will be stopped and the dispatch process will be started to supply the inactive region.

3.3 The Adjustment of the Sizes of Different Lists

(1) Adjusting the sizes of the inactive clean and dirty lists.
During the eviction process, the evicted pages from inactive clean and dirty lists are respectively counted as *evict_clean_cnt* and *evict_dirty_cnt*. Assuming that flash read and write latencies are Cr and Cw. When the ratio *evict_clean_cnt*/*evict_dirty_cnt* exceeds $(Cr + Cw)/Cr$, this means too many clean pages have been evicted. Thus, the inactive clean list should be extended. On the contrary, if the eviction ratio is less than $(Cr + Cw)/Cr$, the inactive dirty list will be extended.

(2) Adjusting the sizes of the active and inactive lists.
When both of the sizes of inactive clean and dirty lists are less than *INV_MAX*, or when the size of active mixed list is beyond its upper limit *ACTV*, the dead pages in active list will be dispatched to the inactive lists.

3.4 The Dispatch and Relive Strategy

When one of the above situations is met, the dispatch process will be activated to calculate the life value of the page pointed by *dispatch_hand*. The definition of life value follows the principle of LRU considering the effect of different access frequency, recency and even the cost of flash write and read operations.
(1) The frequency F: It is defined as the average access counts between the first access time of the page and the system current access time which are recorded as *first_t* and *current_t*. The access counts of the page is counted as *access_count*. Therefore, F is computed as follows:

$$F = \frac{access\_count}{current\_t - first\_t + 1} \tag{1}$$

(2) The recency R: The more recently the page is accessed, the more likely it is to be accessed again. So the recent access time *recent_t* and the first access time *first_t* of the page decides the recency R:

$$R = recent\_t - first\_t + 1 \tag{2}$$

(3) The weight of flash operation θ: it depends on the costs of flash write and read operations. Assuming that the cost of flash write operation *wr* is Cw and read operation *rd* is Cr. Hence, the weight θ is given as:

$$\theta = \begin{cases} \dfrac{Cr + Cw}{Cr}, & wr \\ 1, & rd \end{cases} \tag{3}$$

(4) The life value *Life*: the definition of life value combines the effect of access frequency, recency and the weight of flash operation. Therefore, the life value is defined as follows:

$$\begin{aligned} Life &= F \times R \times \theta = \frac{access\_count}{current\_t - first\_t + 1} \\ &\quad \times (recent\_t - first\_t + 1) \times \theta \end{aligned} \tag{4}$$

(5) The dispatch process: If the life value of one page in the active mixed list is less than 1 or the access count is 1, the page will be dispatched depending on its dirty flag. Clean pages will be moved to the inactive clean list, and dirty pages will be linked to the inactive dirty list. Otherwise, if the life value of the page is beyond 1, the page will be skipped and its access count will be decreased by 1.
(6) The relive strategy: when one page located in the inactive lists is accessed again, it will be relived and added to the tail of the active mixed list.

3.5 The Adoption of Multithreading

The existing algorithms begin to evict when buffer pages are used up, which leads to block users' requests. In order to avoid this situation, LAB_LRU adopts the multithreading technology *pthread* to monitor the usage of buffer size and execute above processes in parallel.

4. Performance Evaluation

This section first introduces the flash memory simulator. Then, LAB-LRU is evaluated and compared with some already existing algorithms such as LRU, CFLRU, CCF-LRU, APB-LRU and PT-LRU in aspects of buffer hit ratio, flash write and read counts, and overall runtime.

4.1 Experiment Setup

To compare with other algorithms accurately, the same simulation system Flash-DBSim [10] is adopted, which is an efficient, reusable and configurable flash memory simulator. The specific parameters are given in Table 1.

The performance of the compared algorithms is related to their parameter settings. The window size w of CFLRU is set to 0.5. The lower bound of hot region of APB-LRU is 0.8, and its probability to choose cold clean or cold dirty pages is equal to the cost ratio of flash operations $(write + erase)/read$. The parameter *pro* of evicting between cold dirty and hot clean pages in PT-LRU is set to 0.8 depending on its optimal performance.

The experiment is evaluated by four synthetic traces which are in accordance with the Zipf distribution. The information of test cases is listed in Table 2. The locality "*x%/y%*" means that *x%* of total requests are located in the *y%* of pages.

4.2 Experiment Results and Analysis

Hit ratio is one of the most important evaluation standards for buffer algorithms. Figure 2 shows the hit ratios of above algorithms with different test cases and buffer sizes. According to the results, the hit ratio of LAB-LRU is higher than other algorithms in most cases. Because LAB-LRU adopts double thresholds to control the eviction process. This avoids pages which are newly read into the buffer being evicted immediately, and prevents cold dirty pages from staying too long in the buffer. Moreover, LAB-LRU considers access frequency and recency when calculating the life

Table 1 Parameter configurations of the flash memory.

Parameters	Value
Page size	2048 B
Block size	64 pages
Block nums	1024 blocks
Read latency	25 us/page
Write latency	200 us/page
Erase latency	2.5 ms/block
Erase endurance	100000 cycles

Table 2 Four synthesized test traces.

Test Cases	Request Num	Read/Write Ratio	Locality
T1	3000000	90%/10%	60%/40%
T2	3000000	30%/70%	70%/30%
T3	3000000	60%/40%	60%/40%
T4	3000000	80%/20%	80%/20%

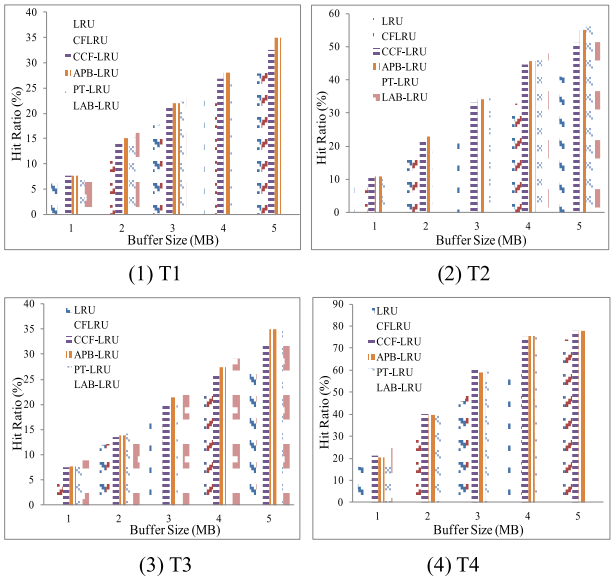

Fig. 2 Hit ratios of four traces under various buffer sizes.

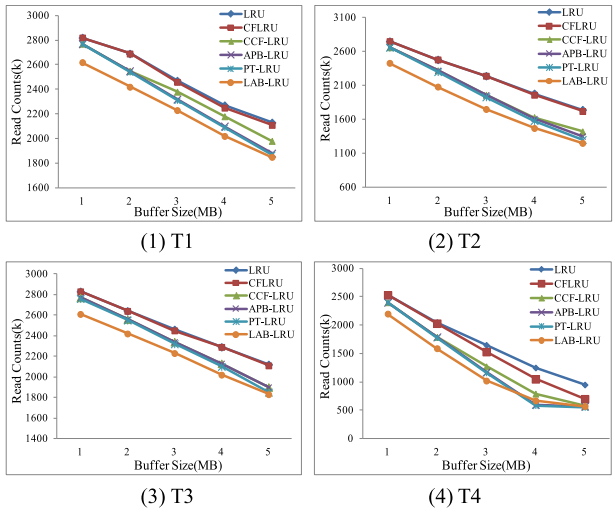

Fig. 3 Physical read counts of four traces under various buffer sizes.

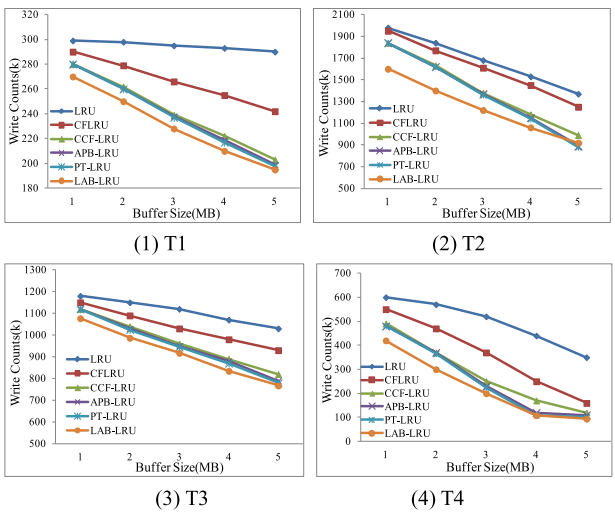

Fig. 4 Physical write counts of four traces under various buffer sizes.

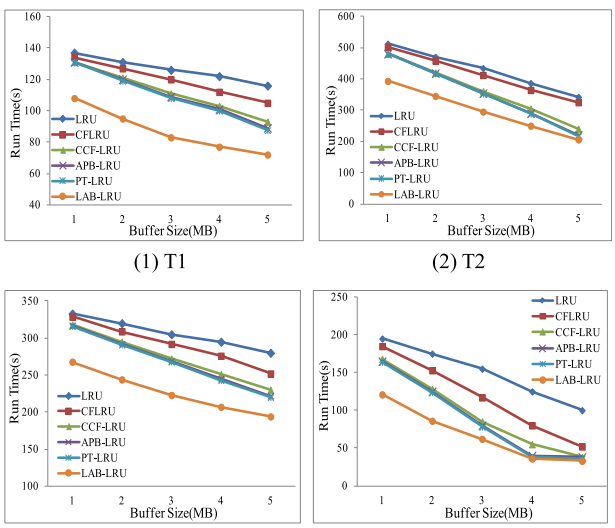

Fig. 5 Runtime of four traces under various buffer sizes.

value of pages in active mixed list. As a result, active pages with higher frequency and recency can stay longer in the buffer.

The counts of flash read and write operations determine the I/O latency and system performance. Figure 3 gives the physical read counts of flash of above algorithms. It can be seen that LAB-LRU reduces read counts of flash obviously. CF-LRU evicts clean pages preferentially and CCF-LRU gives priority to cold clean pages. APB-LRU uses a fixed probability to evict cold clean or cold dirty pages, and PT-LRU chooses cold dirty or hot clean pages by a parameter *pro*. To some extent, above algorithms all lead to early eviction of clean pages even if there are still many cold dirty pages, which increase the read counts of flash. Instead of fixed ways, LAB-LRU evicts clean and dirty pages depending on their access frequency and recency, which is closer to the principle of LRU.

Since the flash write latency is much longer than the read latency, and long erase operations are needed before write operations, it's very significant to reduce write counts of flash. Figure 4 shows the physical write counts of flash of above algorithms. Under the same conditions, the write counts of LAB-LRU are fewer than other algorithms in most cases. Because the calculation of life value not only considers the effect of frequency and recency, but also the different costs of flash write and read operations. It gives a greater weight to the hot dirty pages to enable them stay longer in the buffer.

Figure 5 shows the runtime of different algorithms under various buffer sizes and traces. The runtime is mainly decided by the costs of write, read and erase operations of flash. Although the calculation of life value is bound to bring additional time overhead, LAB-LRU decreases the runtime significantly. This should be attributed to the adoption of multithreading technology, which enables to monitor

the use of buffer size in parallel and evict dead pages in advance. So the runtime is shortened along with improving the system efficiency.

5. Conclusion

The efficiency of a buffer management algorithm for flash memory plays an important role in improving system performance. To overcome the drawbacks of the existing algorithms, we proposed a life-aware buffer management algorithm named LAB-LRU which defined a life value for every page to decide how long the page can stay in the buffer. The definition of life value considered the effect of access frequency, recency and the cost of flash operations. Therefore, active pages with higher access frequency or recency can stay longer in the buffer. Additionally, the adoption of multithreading technology shortened system execution time and improved system efficiency. Four synthesized traces have been carried out, and the experimental results showed that the proposed LAB-LRU algorithm achieves better performance than other existing algorithms in terms of the buffer hit ratio, counts of flash operations and overall runtime.

References

[1] G. Lawton, "Improved flash memory grows in popularity," Computer, vol.39, no.1, pp.16–18, Jan. 2006.

[2] H. Kim, K.Y. Lee, J. Jung, and K. Bahng, "A new transactional flash translation layer for embedded database systems based on MLC NAND flash memory," Proc. International Conference on Consumer Electronics, pp.1–2, Las Vegas, NV, 2008.

[3] M. Lin, S. Chen, and Z. Zhou, "An Efficient Page Replacement Algorithm for NAND Flash Memory," IEEE Trans. Consum. Electron., vol.59, no.4, pp.779–785, 2013.

[4] S.-Y. Park, J. Dawoon, and J.-U. Kang, "CFLRU: A replacement algorithm for flash memory," Proc. CASES'06, pp.234–241, Seoul, Korea, 2006.

[5] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration for LRU and writes sequence reordering for flash memory," IEEE Trans. Consum. Electron., vol.54, no.3, pp.1215–1223, Aug. 2008.

[6] Z. Li, P.Q. Jin, X. Su, K. Cui, and L.H. Yue, "CCF-LRU: A new buffer replacement algorithm for flash memory," IEEE Trans. Consum. Electron., vol.55, no.3, pp.1351–1359, Aug. 2009.

[7] P.Q. Jin, Y. Ou, T. Härder, and Z. Li, "AD-LRU: An efficient buffer replacement algorithm for flash-based databases," Data & Knowledge Engineering, vol.72, pp.83–102, Feb. 2012.

[8] Z.Y. Lin, M.X. Lai, Q. Zou, Y.S. Xue, and S.Y. Yang, "Probability-Based Buffer Replacement Algorithm for Flash-Based Database," Chinese Journal of Computers, vol.36, no.8, pp.1568–1581, Aug. 2013.

[9] J. Cui, W. Wu, Y. Wang, and Z. Duan, "PT-LRU: A Probabilistic Page Replacement Algorithm for NAND Flash-based Consumer Electronics," IEEE Trans. Consum. Electron., vol.60, no.4, Nov. 2014.

[10] P.Q. Jin, X. Su, Z. Li, and L.H. Yue, "A flexible simulation environment for flash-aware algorithms," in Proc. 18th ACM conference on Information and knowledge management, pp.2093–2094, Hong Kong, China, Nov. 2009.