Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 136246, 11 pages
http://dx.doi.org/10.1155/2014/136246

Research Article

Flash-Aware Page Replacement Algorithm

Guangxia Xu,[1] Lingling Ren,[1] and Yanbing Liu[2]

[1] *School of Software Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*
[2] *School of Computer Science, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*

Correspondence should be addressed to Guangxia Xu; 38742571@qq.com

Received 19 May 2014; Accepted 19 July 2014; Published 12 August 2014

Academic Editor: Massimo Scalia

Due to the limited main memory resource of consumer electronics equipped with NAND flash memory as storage device, an efficient page replacement algorithm called FAPRA is proposed for NAND flash memory in the light of its inherent characteristics. FAPRA introduces an efficient victim page selection scheme taking into account the benefit-to-cost ratio for evicting each victim page candidate and the combined recency and frequency value, as well as the erase count of the block to which each page belongs. Since the dirty victim page often contains clean data that exist in both the main memory and the NAND flash memory based storage device, FAPRA only writes the dirty data within the victim page back to the NAND flash memory based storage device in order to reduce the redundant write operations. We conduct a series of trace-driven simulations and experimental results show that our proposed FAPRA algorithm outperforms the state-of-the-art algorithms in terms of page hit ratio, the number of write operations, runtime, and the degree of wear leveling.

1. Introduction

Recently, more and more consumer electronics are equipped with NAND flash memory as storage media due to its attractive features such as fast data access speed, small size, shock resistance, low power consumption, increasing capacity, and decreasing cost [1]. Because the main memory resource of consumer electronics is very limited and it is not enough for running applications with large memory footprint requirements or multiple applications at the same time in the main memory, efficient page replacement algorithms should be designed for NAND flash memory based storage media to evict pages that will be least likely to be referenced in the near future and obtain free page frames in the main memory.

Traditional page replacement algorithms are designed for magnetic disk to maximize their page hit ratios under the assumption that the *I/O* operation costs of magnetic disk are equal and magnetic disk can be overwritten. However, the *I/O* operation costs of NAND flash memory are asymmetric and NAND flash memory only supports the out-of-place update scheme. Therefore, traditional page replacement algorithms are not available for use to NAND flash memory based storage media. In order to improve the performance of the NAND flash-based storage system, a number of page replacement algorithms have been designed for the NAND flash-based storage systems by modifying the original LRU algorithm to evict the clean pages preferentially. For example, CFLRU [2], DL-CFLRU/E [3], LRU-WSR [4], and CCF-LRU [5] are proposed to select the clean pages as the victim pages preferentially. Therefore, these page replacement algorithms designed for NAND flash-based storage systems show low page hit ratios.

In order to keep a high page hit ratio for NAND flash based storage systems and reduce the number of write operations to NAND flash based storage systems, this paper proposes an efficient page replacement algorithm for NAND flash memory in the light of its unique characteristics, which is called FAPRA. FAPRA focuses on preventing the serious degradation of page hit ratio, and reducing the number of write operations, as well as lowering the degree of wear leveling of NAND flash memory. The contributions of our paper can be summarized as follows.

(1) FAPRA introduces an efficient victim page selection scheme, which takes into account the benefit-to-cost ratio for evicting each victim page candidate, the combined recency

and frequency value, and the erase count of the block to which each page belongs.

(2) Since the dirty victim page often contains clean data that exist in both the main memory and NAND flash memory, FAPRA only writes the dirty data within the victim page back to reduce the redundant write operations in terms of clean data.

We conduct a series of trace-driven simulations with real traces and experimental results show that our proposed FAPRA algorithm outperforms existing page replacement algorithms designed for NAND flash memory.

The rest of this paper is organized as follows. Section 2 describes the overview of NAND flash memory. In Section 3, we briefly review the related work. In Section 4, we present the detailed design of our proposed FAPRA algorithm. In Section 5, we show performance evaluation. Finally, we conclude our work in Section 6.

2. Background

NAND flash memory consists of blocks and each block contains a fixed number of pages. NAND flash memory performs three basic operations, which are read, write, and erase operation. Read operation is responsible for reading data from a target page and write operation is used to write data to a target page. Erase operation is in charge of erasing a block. Therefore, the basic unit of read and write operation is a page, while the basic unit of erase operation is a block, as listed in Table 1.

NAND flash memory shows several inherent characteristics. Firstly, NAND flash memory presents the erase-before-write hardware constraint. Namely, once a page is written, it should be erased in advance before it is written again. NAND flash memory adopts the out-of-place update scheme to address this hardware constraint. Secondly, the erase count of each block is limited, typically 10,000~100,000 times. If the erase count of a block exceeds this threshold value, the block will be worn out and suffer from frequent write errors. Finally, the costs of write operation and read operation of NAND flash memory are asymmetric in terms of access time and energy consumption, as shown in Table 1 [6].

Table 1: The characteristics of NAND flash memory.

Basic operation	Access time	Energy consumption	Access granularity
Read	47.2 μs	679 nJ	Page (512 B)
Write	533 μs	7.66 μJ	Page (512 B)
Erase	3 ms	43.2 μJ	Block (16 KB)

3. Related Work

The least recently used algorithm (LRU) is widely used in most modern operating systems to maximize the page hit ratio. LRU works on the idea that pages that have been most heavily used in the past few instructions are most likely to be used heavily in the next few instructions too. Therefore, LRU keeps track of the recently referenced time of each page and evicts the least recently referenced page to improve the page hit ratio. LRU is customized for magnetic disk under the assumption that the magnetic disk supports in-place update scheme and its *I/O* costs for read and write operation are equal. But NAND flash memory shows different inherent characteristics, which are the out-of-place update scheme, as well as the asymmetric *I/O* costs for read, write, and erase operation. When LRU is directly implemented on the NAND flash memory based storage media, LRU will evict the least recently referenced page regardless of whether this page is dirty or not. If the victim page that is selected each time is dirty, writing the dirty victim page back to NAND flash memory based storage media will incur a number of write operations. In order to reduce the number of write operations, a number of page replacement algorithms have been studied for NAND flash memory based storage media by modifying the LRU and evicting the clean pages preferentially.

Park et al. proposed the first page replacement algorithm for NAND flash memory, which is called CFLRU [2, 7]. CFLRU divides the LRU list into two regions, which are the working region and clean-first region, respectively. The pages in the working region have high likelihood to be referenced in the near future and most of page hits are generated in this region, while the clean-first region contains the pages that are least recently referenced. In order to reduce the number of write operations, it preferentially evicts the clean pages within the clean-first region by the LRU order. The page hit ratio of CFLRU is influenced by the window size of the clean-first region. Based on CFLRU, Yoo et al. proposed three page replacement algorithms for NAND flash memory [3], which are the CFLRU/C, CFLRU/E, and DL-CFLRU/E. CFLRU/C enhances CFLRU by evicting the dirty page with the lowest access frequency in the clean-first region while CFLRU/E improves CFLRU by evicting the dirty page belonging to the block with the lowest erase count in the clean-first region. DL-CFLRU/E maintains two page lists, which are the clean page list and the dirty page list. It first evicts the pages within the clean page list by the LRU order. If the clean page list is empty, DL-CFLRU/E evicts the dirty page belonging to the block with the lowest erase count within the window of the dirty page list. Jung et al. also proposed a page replacement algorithm for NAND flash memory called LRU-WSR [4]. It enhances the LRU by delaying the eviction of not-cold dirty page from the buffer cache to NAND flash memory in order to reduce the number of write operations and prevent excessive degradation of the page hit ratio. Li et al. proposed an efficient buffer replacement algorithm called Cold-Clean-First LRU (CCF-LRU) for NAND flash memory based database systems [5], which also introduces a cold-detection algorithm to divide the victim page candidates into hot and cold ones. CCF-LRU preferentially evicts the page that is cold and clean. Shen et al. proposed an adaptive page replacement algorithm called APRA for NAND flash memory storages [8]. APRA maintains two LRU lists with the same size. The first LRU list stores all the victim page candidates and has a window on the LRU position. The second LRU list, also called a ghost list, is just used to store the metadata of the pages evicted from the first LRU list. APRA uses a learning rule to

adaptively and continually revise the size of the window in response to diverse workloads with different access patterns. Lin et al. proposed a page replacement algorithm called EPRA for different kinds of NAND flash memory with different cost ratios of write operation to read operation [9]. In EPRA, each victim page candidate is assigned a weighted value and the victim page candidate with the least weighted value is evicted. Ou et al. proposed an efficient buffer management algorithm called CFDC for NAND flash-based data systems [10]. CFDC maintains a page list and divides into two regions, which are the working region and the priority region, respectively. The working region is used to keep the hot pages that are frequently and recently revisited, while the priority region contains the victim page candidates that will be evicted in the near future. The priority region consists of a clean page list and a priority queue of dirty-page clusters. It first evicts the clean pages within the clean page list of the priority region. If the clean page list is empty, the priority queue is scanned and the cluster with the lowest priority is evicted. Lin et al. proposed a greedy page replacement algorithm called GDLRU for flash-aware swap system [11]. GDLRU introduces the subpaging technique to divide the dirty pages within the main memory into a fixed number of subpages. GDLRU also first evicts the clean pages within the page list. If there is no clean page within the page list, the dirty page with the least dirty subpages is selected as a victim page and flushed into the NAND flash-based storage device. Lin et al. also proposed an adaptive buffer replacement algorithm called HDC for NAND flash memory-based databases [12]. Each page is assigned with a replacing index. HDC selects a page with the smallest replacing index as a victim page instead of selecting the clean pages preferentially. Xu et al. proposed a new page replacement algorithm called CLRU for NAND flash-based consumer electronics [13]. CLRU reduces the number of write operations to NAND flash-based storage devices by delaying the eviction of cold dirty pages and improves the page hit ratio by first evicting cold pages.

4. Proposed Page Replacement Algorithm

This section presents an efficient page replacement algorithm called FAPRA for NAND flash memory. Our proposed FAPRA algorithm focuses on reducing the number of write operations and runtime, preventing the serious degradation of page hit ratio and lowering the degree of wear leveling of NAND flash memory to extend the lifetime of NAND flash memory. Before describing our proposed FAPRA itself, the architecture of our NAND flash memory storage system is explained in the first part of this section. Next, the design principles, basic idea, and related definitions of our proposed FAPRA algorithm are given in the second part of this section. Then, the third part of this section presents the victim page selection scheme and an example of our proposed FAPRA algorithm. Finally, the performance overhead of our proposed FAPRA algorithm is discussed in the fourth part of this section.

4.1. NAND Flash Memory Storage System Architecture. The architecture of our NAND flash memory storage system is shown in Figure 1. The page replacement algorithm is designed for Linux kernel in the light of the inherent characteristics of NAND flash memory. Our NAND flash memory storage system architecture mainly contains the flash translation layer (FTL), Linux operating system, and NAND flash memory. FTL is a software module between the traditional file system and the NAND flash memory based storage device. FTL [14–16], which provides address translator and garbage collector, is responsible for hiding the inherent characteristics of NAND flash memory and emulating NAND flash memory as a block device.

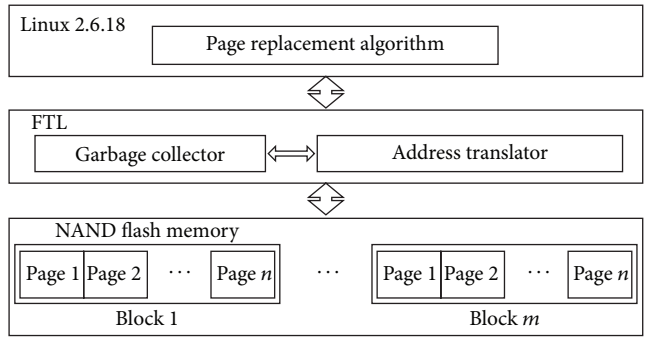

Figure 1: The architecture of our NAND flash memory storage system.

4.2. Proposed FAPRA Algorithm. Intensive dirty victim pages written back to NAND flash memory based storage media could not only generate a large number of write operations to NAND flash memory and result in using up the free space of NAND flash memory based storage media quickly, but also incur the garbage collection operation with high energy consumption frequently to reclaim garbage and obtain free space for NAND flash memory. Therefore, existing page replacement algorithms customized for NAND flash memory enhance the LRU by preferentially evicting the clean pages and delaying the eviction of the dirty pages to reduce the number of write operations.

Because access frequency and hot degree of page could also influence the page hit ratio of page replacement algorithm like access recency, CFLRU/C considers the access frequency of each dirty page with the window during the selection of victim page, while both LRU-WSR and CCF-LRU introduce cold-detection scheme to divide all the victim page candidates into hot and cold ones and preferentially evict the cold pages to improve the page hit ratio.

The erase count of the block can influence the lifetime of NAND flash memory. If a dirty page belonging to the block with a large erase count is evicted and written back to NAND flash memory, the corresponding flash pages within the block will be marked invalid and this block will be reclaimed by garbage collection operation for obtaining free space. In this case, the erase count of this block will increase and may exceed the threshold value.

Therefore, the principles for designing an efficient page replacement algorithm for NAND flash memory based storage media are reducing the number of write operations, as

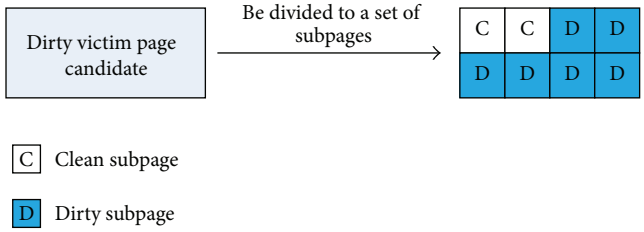

Figure 2: The process of divding the dirty main memory page.

well as considering the access recency, access frequency, and the erase count of the block to which each page belongs when selecting a victim page. In order to achieve the above design principles, this paper proposes an efficient page replacement algorithm called FAPRA for NAND flash memory in the light of the unique characteristics of NAND flash memory.

Existing page replacement algorithms designed for NAND flash memory write the dirty victim page back to NAND flash memory fully when a page fault happens [17]. The typical sizes of the main memory page and flash page are 4 KB and 512 B. Therefore, writing a full dirty victim page back to NAND flash memory will result in eight write operations. Li et al. reported that the dirty victim page often contains clean data [6]. In this case, some of these eight write operations are redundant and the redundant write operations should be eliminated because the write operation is costly and intensive write operations will use up the free space of NAND flash memory quickly.

In order to identify the clean data, our proposed FAPRA algorithm introduces the subpaging technique used by Li et al. and divides each dirty victim page candidate in the main memory ito a fixed number of subpages that are of the same size, as illustrated in Figure 2. The size of each subpage is equal to that of flash page and each subpage has an additional flag called dirty flag. If the subpage in the main memory is different from its copy in the NAND flash memory based storage device, its dirty flag is set and it is considered as dirty subpage. If the subpage in the main memory is the same as its copy, its dirty flag keeps unset and it is called clean subpage.

As illustrated in Figure 3, FAPRA maintains two page lists, which are the mixed page list and full dirty page list. The mixed page list contains all the clean pages and the partial dirty pages, while the full dirty page list links all the full dirty pages. The page in which all the subpages are dirty is considered as full dirty page, while the dirty page in which partial subpages are clean is referred to as partial dirty page.

We define the benefit-to-cost ratio for evicting a victim page candidate as the free space saved for NAND flash memory based storage device to the cost for reading the dirty subpages and writing them back to NAND flash memory based storage device. Then, the benefit-to-cost ratio for evicting the victim page candidate P is denoted by BCR(P) and expressed as follows:

$$\mathrm{BCR}(P) = \frac{1-u}{1+u} \tag{1}$$

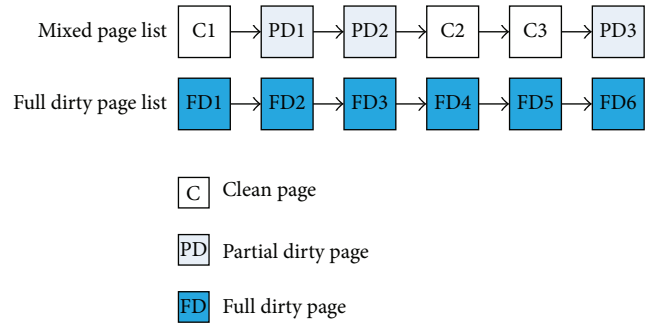

Figure 3: The example of FAPRA.

with

$$u = \frac{n}{m}, \tag{2}$$

where u is the percentage of dirty subpages within the victim page candidate P. m and n are the total number of subpages and the number of dirty subpages within the victim page candidate P, respectively. The terms $1-u$ and $1+u$ represent the free space of NAND flash memory that is saved and the cost for reading the whole victim page candidate in order to read dirty subpages and writing them back to NAND flash memory based storage device. The benefit-to-cost ratio for evicting a clean victim page candidate is 1 because it has no dirty subpages, while that for evicting a full victim page candidate is 0 because all the subpages within it are dirty. It can be seen that evicting a dirty page is much more costly than evicting a clean page.

The combined recency and frequency value of the victim page candidate P is denoted by CRFV(P) and is calculated as follows:

$$\mathrm{CRFV}(P) = \sum_{i=1}^{k} F\left(t_{c\_t} - t_{pi\_t}\right) \tag{3}$$

with

$$F(x) = \left(\frac{1}{\sqrt{2}}\right)^{x}, \tag{4}$$

where t_{c_t} is the current time and $\{t_{p1\_t}, t_{p2\_t}, \ldots, t_{pi\_t}, \ldots, t_{pk\_t}\}$ are the recently referenced times of the victim page candidate P. The system time is designed to be an integer value and is incremented by one on each page reference. The combined recency and frequency value of each page considers access recency and access frequency of each page to quantify the likelihood that the page will be referenced in the near future. Because of the combined recency and frequency value, the page hit ratio of page replacement algorithm can be improved.

4.3. The Victim Page Selection Scheme. Our proposed FAPRA algorithm introduces an efficient victim page selection scheme to select a victim page for replacing, which is called BCE and takes into account the benefit-to-cost ratio for evicting each victim page candidate, the combined recency

Algorithm VictimPageSelection

Input: Mixed page list; Full dirty page list
Output: Victim page
(1) if (the mixed page list is not empty)
(2) for (each page within the mixed page list)
(3) victim page = The page with the biggest selecting index value;
(4) end for
(5) end if
(6) else
(7) for (each page within the full dirty page list)
(8) victim page = the full dirty page with the biggest selecting index value;
(9) end for
(10) end else

Algorithm 1: The pseudocode for the victim page selection scheme.

and frequency value of each victim page candidate, and the erase count of the block to which each page belongs. BCE assigns a selecting index value to each victim page candidate within the mixed page list, which is calculated as follows:

$$\mathrm{SIV}(P) = \mathrm{BCR}(P) \times \frac{1}{\mathrm{CRFV}(P)} \times \frac{\varepsilon_{\max} - \varepsilon_{\min} + 1}{P_{\mathrm{erase\_count}} - \varepsilon_{\min} + 1}, \tag{5}$$

where $\mathrm{SIV}(P)$ is the selecting index value of the main memory page P and $P_{\mathrm{erase\_count}}$ is the erase count of the block to which the main memory page P belongs. $\varepsilon_{\max}$ is the maximum erase count of all blocks in the NAND flash memory, while $\varepsilon_{\min}$ is the minimum erase count of all blocks in the NAND flash memory.

Because the benefit-to-cost ratio for evicting each full dirty page is 0, the selecting index value of the full dirty page cannot be defined as (5). We define the selecting index value of the full dirty page P by not considering the benefit-to-cost ratio for evicting each full dirty page and it is calculated as follows:

$$\frac{1}{\mathrm{CRFV}(P)} \times \frac{\varepsilon_{\max} - \varepsilon_{\min} + 1}{P_{\mathrm{erase\_count}} - \varepsilon_{\min} + 1}. \tag{6}$$

When the number of free page frames in the main memory is less than the threshold value, our proposed FAPRA algorithm is triggered. FAPRA first scans the mixed page list and selects the page with the biggest selecting index value as victim. If the mixed page list is empty, FAPRA changes to scan the full dirty page list and evicts the full dirty page with the biggest selecting index value. The pseudocode for the victim page selection scheme in the proposed algorithm is shown in Algorithm 1.

A full dirty victim page written back to NAND flash memory based storage media often generates a large number of redundant write operations that write clean data to NAND flash memory. These redundant write operations could accelerate the speed of using up the free space of NAND flash memory and incur the garbage collection operation with high energy consumption frequently to reclaim garbage. Moreover, the write operation to NAND flash memory is costly. Therefore, FAPRA checks the status of the victim page after it is selected as victim page. If it is a clean page, FAPRA just erases it from main memory and makes the corresponding page frame free. If it is a dirty page, FAPRA continues to check the status of each subpage within the victim page and only writes the dirty subpages back to NAND flash memory based storage media. After that, FAPRA erases the whole dirty victim page from main memory and makes the corresponding page frame free. In order to lower the degree of wear leveling and extend the lifetime of NAND flash memory based storage device, FAPRA writes the dirty subpages back to the free block with the least erase count within the NAND flash memory.

For instance, a dirty victim page contains two clean subpages and six dirty subpages as shown in Figure 2. FAPRA is adopted to only write the six dirty subpages within the victim page back to the NAND flash memory based storage media; two write operations are eliminated.

4.4. Performance Overhead Discussion. The performance overhead of FAPRA is analyzed in this part. Li et al. tested five commonly used applications and they found that the dirty victim page in these applications often contains a significant amount of clean data. All the tested applications have one clean subpage at least except gqview [6]. It indicates that the average number of clean subpages within each dirty page in most applications is at least 1. The increasing time cost for setting and checking dirty flags is calculated as $(2 \times n \times C_a)$, where C_a is the average memory access time. The saved time cost for writing the clean subpages back to is calculated as $((m-n) \times C_w)$, where C_w is the time cost for one write operation to NAND flash memory. According to [6, 7], C_a and C_w are approximately equal to 2.56 μs and 533 μs, respectively. Since $1 \leq n \leq 7$, the maximum time cost for setting and checking dirty flags is 35.84 μs and the minimum saved time cost for writing the clean subpages back to NAND flash memory is 533 μs. Apparently, the saved time cost for writing the clean subpages back to NAND flash memory is much more than that for setting and checking the dirty flags and the performance overhead of our proposed FAPRA is negligible.

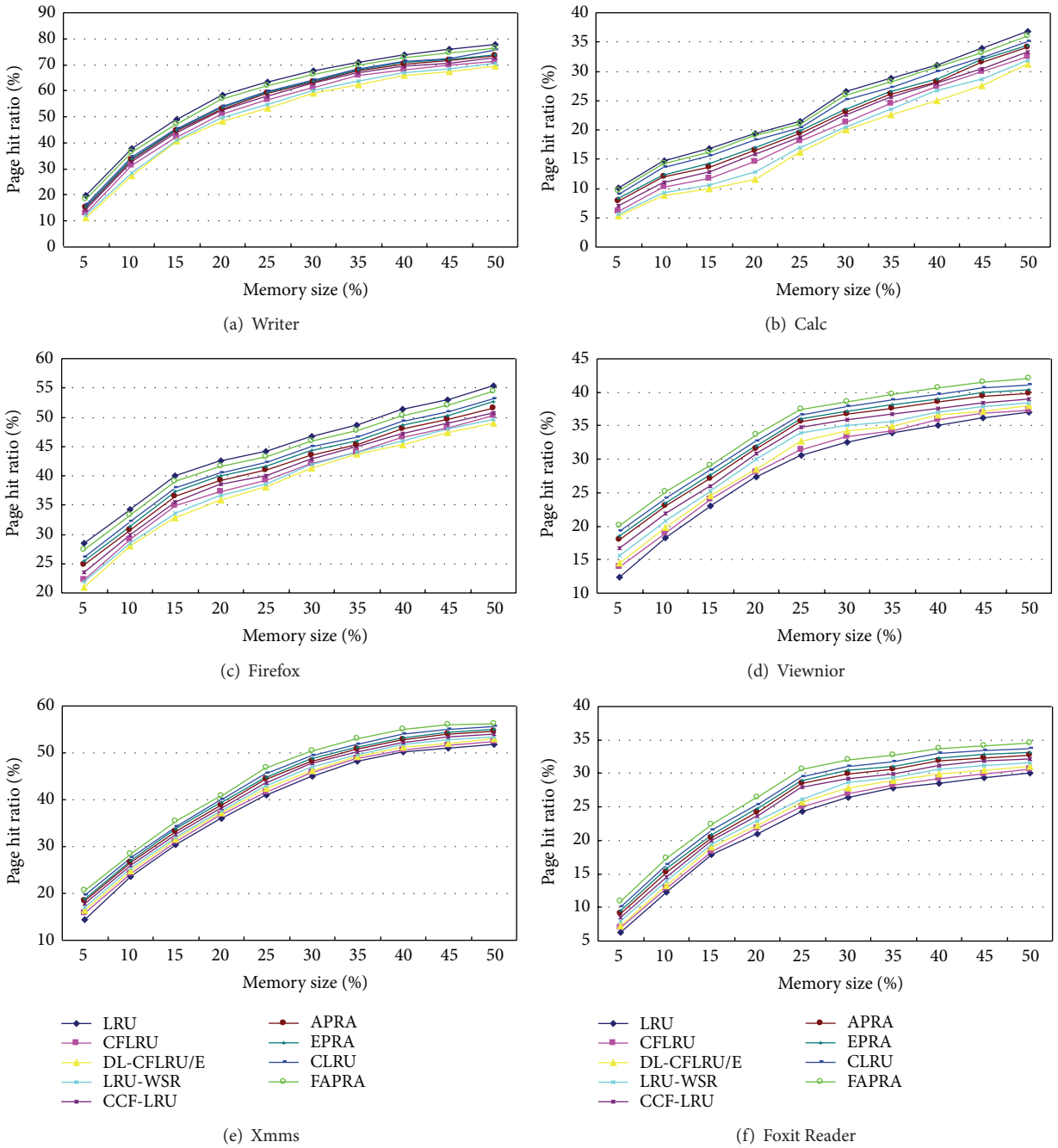

Figure 4: The page hit ratio for different traces.

5. Performance Evaluation

In this section, we compare the performance of our proposed FAPRA algorithm with those of the state-of-the-art algorithms, which are LRU, CFLRU, DL-CFLRU/E, LRU-WSR, CCF-LRU, APRA, EPRA, and CLRU by conducting trace-driven simulation with real traces. LRU is customized for magnetic disk and other algorithms are designed for NAND flash memory. The performance metrics that our simulation experiment is performed to test are the page hit ratio, the number of write operations, runtime, and the degree of wear leveling.

5.1. Experiment Setup. We conduct the trace-driven simulation with the real traces. Our simulator contains a demand paging system. The virtual memory reference traces are collected using the Valgrind toolset, which is a powerful memory profiling tool [18]. These traces are captured when executing six different applications on Linux 2.6.18 running on an x86-based PC. The characteristics of traces used in our simulation are also shown in Table 2.

In our simulation, we assume that the demand paging system has a 533 MHz microprocessor, 32 MB SDRAM, 2 MB NOR flash memory, and 64 MB NAND flash memory. The

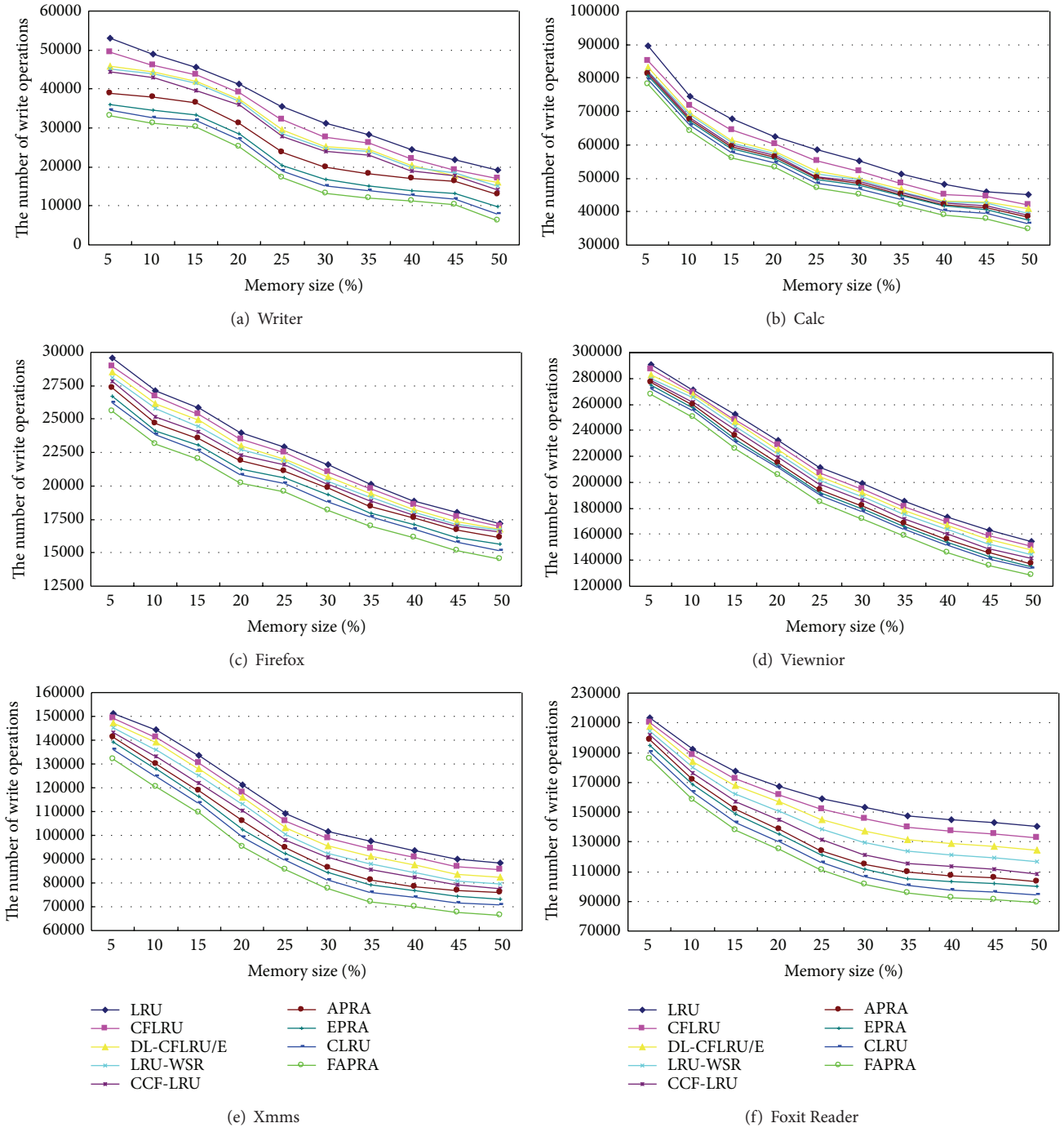

Figure 5: The number of write operations for different traces.

Table 2: Workloads used in our simulations and their characteristics.

Workload	Memory footprint	Ratio of read count to write count	Memory references			
			Total I/O requests	Instruction read	Data read	Data write
Writer	27.95 MB	4.22 : 1	4,540,417	973,956	2,696,020	870,441
Calc	38.61 MB	2.17 : 1	10,797,948	1,439,745	5,951,252	3,406,951
Firefox	37.42 MB	5.49 : 1	4,028,737	1,963,639	1,444,658	620,440
Viewnior	20.37 MB	1 : 1.35	27,204,428	842,574	10,751,502	15,610,352
Xmms	19.2 MB	1 : 5.13	2,793,764	155,465	300,301	2,337,998
Foxit Reader	35.42 MB	1 : 1.5	9,842,423	1,877,086	2,053,107	5,912,230

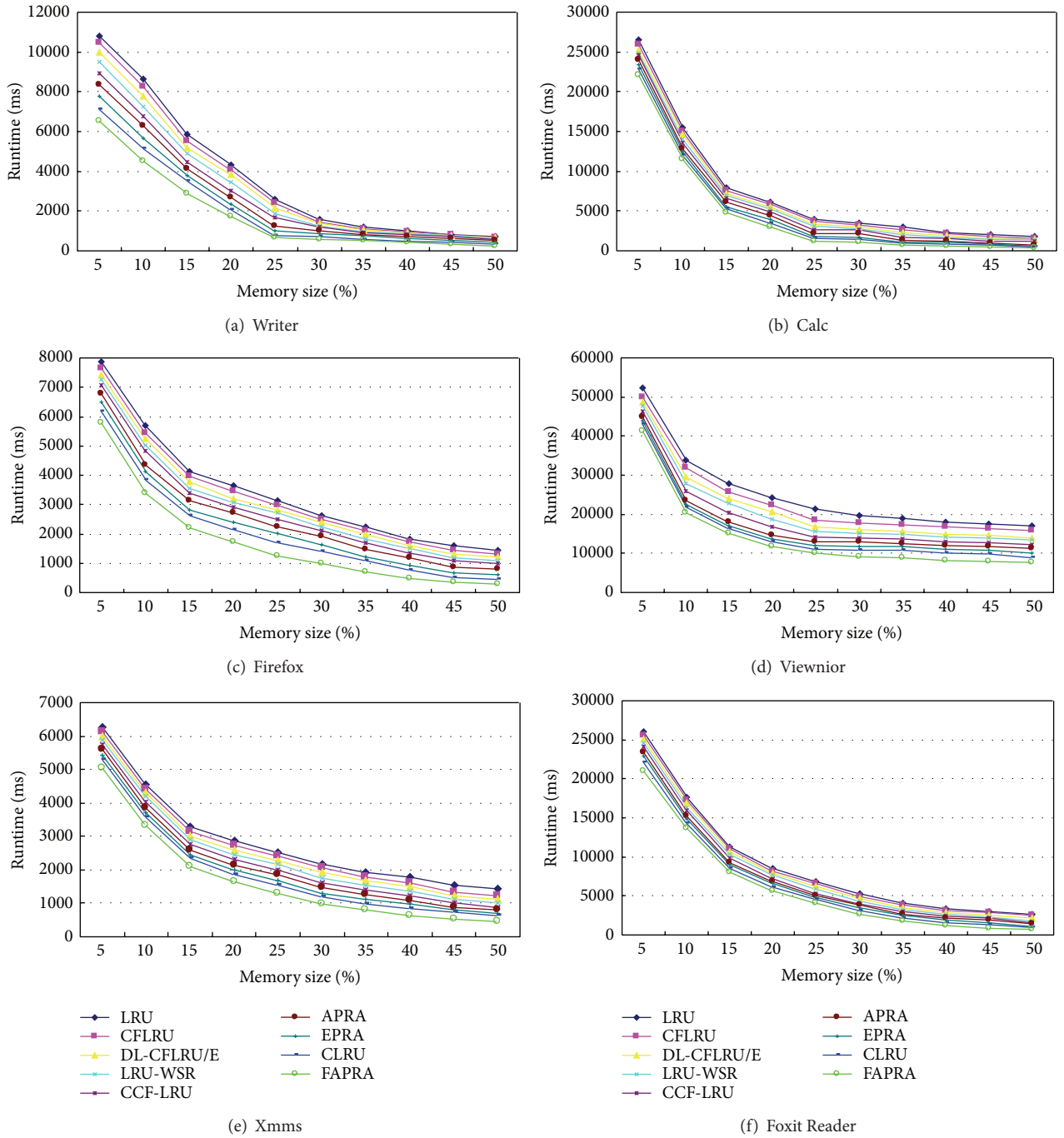

Figure 6: The runtime for different traces.

size of a virtual memory page is set to 4 KB, which is common to most modern operating systems including Linux. We also assume that the Linux operating system and the X Window System occupy about 16 MB main memory space in terms of SDRAM and only the remaining 16 MB SDRAM is available for use to each application. The FTL in our simulation is log-based NFTL [19] and it adopts the cost-benefit garbage collection policy [20] to reclaim garbage. For NAND flash memory, we determine the simulation parameters of NAND flash memory by referring to the Samsung K9F1208R0B flash chip [21] in our simulator. The page size and block size of this flash chip are 512 B and 16 KB, respectively. The limited number of erase cycles of this flash chip is 100,000. The other information of this flash chip is listed in Table 1. As the size of window has influence on the page hit ratios of CFLRU and DL-CFLRU/E, then we set $w_1 = 0.2$ for CFLRU and w_2 for DL-CFLRU/E.

5.2. Simulation Results. Figure 4 shows the page hit ratios of nine page replacement algorithms. Figure 4(a) to Figure 4(c)

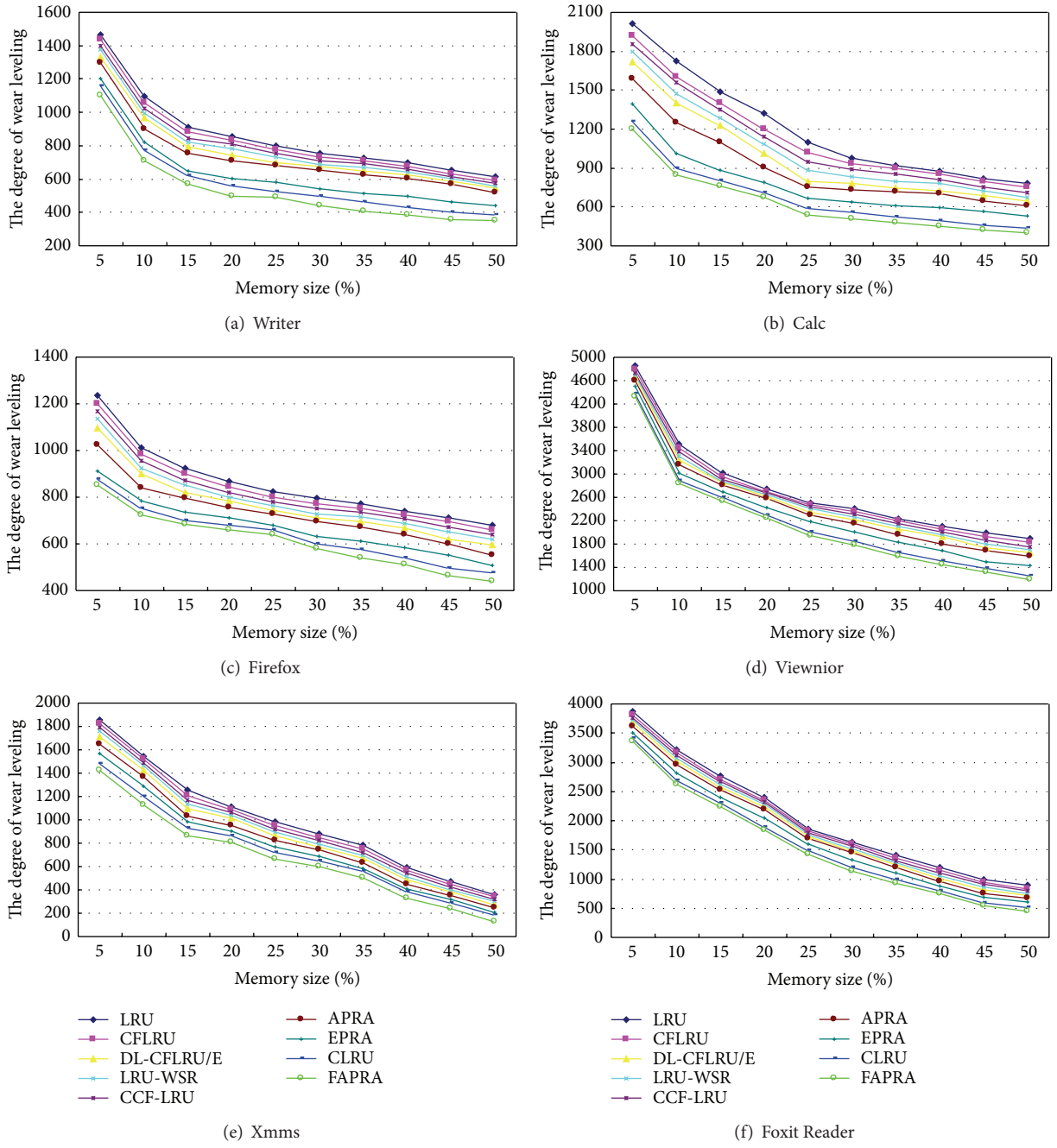

Figure 7: The degree of wear leveling for different traces.

show that the page hit ratio of LRU is higher than that of other algorithms including our proposed FAPRA algorithm, because Writer, Calc, and Firefox are read-intensive applications and all the page replacement algorithms preferentially evict the clean pages except LRU. However, the page hit ratio of our proposed FAPRA algorithm is higher than that of other algorithms designed for NAND flash memory because our proposed FAPRA algorithm considers the combined recency and frequency value of each page when selecting a victim page. Figure 4(d) to Figure 4(f) show that the page hit ratio of our proposed FAPRA algorithm is higher than that of other algorithms including the original LRU algorithm. The Viewnior, Xmms, and Foxit Reader are write-intensive applications. Existing page replacement algorithms proposed for NAND flash memory first evict the clean pages and delay the eviction of dirty pages, so existing page replacement algorithms outperform the original LRU algorithm. Our proposed FAPRA algorithm considers the access recency and frequency of each page and then it shows a higher page hit ratio than other page replacement algorithms designed for NAND flash memory.

Figure 5 shows the number of write operations generated by nine page replacement algorithms. Although existing page replacement algorithms designed for NAND flash memory first evict the clean pages, they do not find the existence of clean data within the dirty block and write the whole dirty block back to the NAND flash memory. However, the victim page selection scheme introduced by our proposed FAPRA algorithm considers the benefit-to-cost ratio for evicting each page. Moreover, FAPRA only writes the dirty subpages within the victim page back to NAND flash memory. Therefore, FAPRA incurs the least write operations to NAND flash memory in all cases.

Figure 6 shows the runtime for different traces. The runtime is highly influenced by the page hit ratio and the number of write operations. The higher page hit ratio and less write operations are, the less runtime consumed is. Because FAPRA has higher page hit ratio and less write operations than other algorithms designed for NAND flash memory, FAPRA takes the least runtime.

Figure 7 shows the degree of wear leveling for different traces. All the existing page replacement algorithms designed for NAND flash memory and the original LRU algorithm do not consider the erase count of the block to which each page belongs during the selection of a victim page. However, our proposed FAPRA algorithm considers the erase count of the block to which each page belongs during the selection of a victim page and writes the dirty subpages back to the free block with the least erase count, so it can be seen that FAPRA significantly improves the degree of wear leveling from Figure 7.

6. Conclusions

This paper presents an efficient page replacement algorithm called FAPRA for NAND flash memory based storage devices. Our proposed FAPRA algorithm divides each dirty victim page candidate in the main memory into subpages of the same size that is equal to that of flash page. The subpages are classified into dirty subapges and clean subpages. When the number of free page frames is lower than a threshold value, FAPRA introduces an efficient victim page selection scheme, which takes into consideration the benefit-to-cost for evicting each victim page candidate, the combined recency and frequency value of each victim page candidate, and the erase count of the block to which each page belongs when selecting a victim page. Since the dirty victim page often contains clean data, FAPRA does not write the clean subpages within the victim page back to NAND flash memory based storage media in order to reduce the redundant write operations. We conducted a series of trace-driven simulations and obtained encouraging results.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation (Grant no. 61309032), Project Supported by Program for Innovation Team Building at Institutions of Higher Education in Chongqing (Grant no. KJTD201310), the Natural Science Foundation of Chongqing (Grant no. cstc2012jjA40053), the Chongqing Medical Research Project (Grant no. 2013-1-049), and the Scientific Research Fund of Chongqing University of Posts and Telecommunications (Grant no. A2012-12).

References

[1] M. Lin, S. Chen, G. Live, and Z. Zhou, "Optimised Linux swap system for flash memory," *Electronics Letters*, vol. 47, no. 11, pp. 641–642, 2011.

[2] S. Y. Park, D. Jung, J. Kang, J. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 234–241, October 2006.

[3] Y. S. Yoo, H. Lee, Y. Ryu, and H. Bahn, "Page replacement algorithms for NAND flash memory storages," in *Computational Science and Its Applications-ICCSA 2007*, vol. 4705 of *Lecture Notes in Computer Science*, pp. 201–212, Springer, Berlin, Germany, 2007.

[4] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: integration of LRU and writes sequence reordering for flash memory," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1215–1223, 2008.

[5] Z. Li, P. Jin, X. Su, K. Cui, and L. Yue, "CCF-LRU: a new buffer replacement algorithm for flash memory," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1351–1359, 2009.

[6] H.-L. Li, C.-L. Yang, and H.-W. Tseng, "Energy-aware flash memory management in virtual memory system," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 8, pp. 952–964, 2008.

[7] C. Park, J. Kang, S. Park, and J. Kim, "Energy-aware demand paging on NAND flash-based embedded storages," in *Proceedings of the 2004 International Symposium on Lower Power Electronics and Design (ISLPED '04)*, pp. 338–343, August 2004.

[8] B. Shen, X. Jin, Y. H. Song, and S. S. Lee, "APRA: adaptive page replacement algorithm for NAND flash memory storages," in *Proceedings of the International Forum on Computer Science-Technology and Applications (IFCSTA '09)*, vol. 1, pp. 11–14, Chongqing, China, December 2009.

[9] M. W. Lin, S. Y. Chen, and Z. Zhou, "An efficient page replacement algorithm for NAND flash memory," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 779–785, 2013.

[10] Y. Ou, T. Härder, and P. Jin, "CFDC: a flash-aware buffer management algorithm for database systems," in *Advances in Databases and Information Systems*, vol. 6295 of *Lecture Notes in Computer Science*, pp. 435–449, Springer, Berlin, Germany, 2010.

[11] M. Lin, S. Chen, and G. Wang, "Greedy page replacement algorithm for flash-aware swap system," *IEEE Transactions on Consumer Electronics*, vol. 58, no. 2, pp. 435–440, 2012.

[12] M. Lin, S. Chen, G. Wang, and T. Wu, "HDC: an adaptive buffer replacement algorithm for NAND flash memory-based databases," *Optik*, vol. 125, no. 3, pp. 1167–1173, 2014.

[13] G. Xu, F. Lin, and Y. Xiao, "CLRU: a new page replacement algorithm for NAND flash-based consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 1, pp. 38–44, 2014.

[14] T. Z. Wang, D. Liu, Y. Wang, and Z. Shao, "FTL2: a hybrid flash translation layer with logging for write reduction in flash memory," *ACM SIGPLAN Notices*, vol. 48, no. 5, pp. 91–100, 2013.

[15] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "MNFTL: an efficient flash translation layer for MLC NAND flash memory storage systems," in *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC '11)*, pp. 17–22, June 2011.

[16] D. Liu, Y. Wang, Z. Qin, Z. Shao, and Y. Guan, "A space reuse strategy for flash translation layers in SLC NAND flash memory storage systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1094–1107, 2012.

[17] O. Kwon, H. Bahn, and K. Koh, "FARS: a page replacement algorithm for NAND flash memory based embedded systems," in *Proceedings of the IEEE 8th International Conference on Computer and Information Technology (CIT '08)*, pp. 218–223, July 2008.

[18] N. Nethercote and J. Seward, "Valgrind: a program supervision framework," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 2, pp. 47–69, 2003.

[19] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366–375, 2002.

[20] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," in *Proceedings of the USENIX Technical Conference*, pp. 155–164, 1995.

[21] Samsung Electronics Company, Datasheet of Samsung K9F1208R0B NAND Flash 2004.