International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 1 | ISSN : 2456-3307

Empirical Evaluation of Effect of Window size on Static Window-Sized CFLRU Page Replacement Policy for Flash Based Systems

Arjun Singh Saud
Central Department of Computer Science and IT, TU, Nepal

ABSTRACT

Flash memory has characteristics of asymmetric I/O latencies for read, write and erase operations and out-of-place update. Thus, buffering policy for flash based systems has to consider these properties to improve the overall performance. CFLRU is one of the flash aware buffer replacement policies. It divides set of pages into two regions, working region and clean-first region, and then tries to replace clean pages from clean first region. Size of clean-first region is called window size (W). This research evaluates impacts of W in performance of CFLRU page replacement policy and concluded that using larger window sizes provides better performance. Hence, window having size more that 70% of total buffer size is recommended.

Keywords : Flash Memory Systems, Buffer Replacement Algorithm, Clean-First LRU, Write Count, Page Fault

I. INTRODUCTION

Flash memory is an electronic non-volatile computer storage medium that can be electrically erased and reprogrammed. Flash memory has been gaining popularity in mobile embedded systems as non-volatile storage due to its characteristics such as small and lightweight form factor, solid-state reliability, and low power consumption. The emergence of single flash memory chip with several gigabytes capacity makes a strong tendency to replace magnetic disk with flash memory for the secondary storage of mobile computing devices such as tablet PCs, PDAs, and smart phones [1,2].

The characteristics of flash memory are significantly different from magnetic disks. First, flash memory has no latency associated with the mechanical head movement to locate the proper position to read or write data. Second, flash memory has asymmetric read and write operation characteristic in terms of performance and energy consumption. Third, flash memory does not support in-place update; the write to the same page cannot be done before the page is erased. Thus, as the number of write operations increases so does the number of erase operations. Erase operations are slow and power-wasting that usually decreases system performance. Finally, blocks of flash memory are worn out after the specified number of write/erase operations. Therefore, erase operations should be avoided for better performance and longer flash memory lifetimes. To avoid wearing specific segments out which would affect the usefulness of the whole flash memory, data should be written evenly to all segments. This is called even wearing or wear-leveling [3,4].

Buffer replacement algorithms used in an OS or a DBMS in general assume that the speed of the read and write operations are about the same, which is true in the case of hard disks. The different characteristics of flash memory make it infeasible for system developed for hard disks as secondary storage to readily be used for the flash memory, and therefore force a reexamination of many key parts of the system architecture. Traditional performance metric such as 'buffer hit ratio' is not sufficient as performance indicator for flash based system. One naïve guide for this scheme may be stated as follows: "Try to reduce the number of writes/erases at the expense of the read operations." A new performance metric such as write count is also needed, in addition to the 'buffer hit ratio' [5].

II. OVERVIEW OF CFLRU POLICY

When cache replacement occurs, two kinds of replacement costs are involved. One is generated when a requested page is fetched from secondary storage to the page cache in RAM. Using Belady's MIN algorithm this cost can be minimized by selecting a victim that has the largest forward distance in the future references. Among online algorithms, LRU has been commonly used for replacement algorithm because it exploits the property of locality in references. Another cost is generated when a page is evicted from the page cache to secondary storage, that is, flash memory. This cost can be minimized by selecting a clean page for eviction. A clean page contains the same copy of the original data in flash memory thus the clean page can be just dropped from the page cache when it is evicted by the replacement policy. Satisfying only one kind of replacement cost would benefit from its advantage, but for a long term, it would affect the other kind of replacement cost. For example, a replacement policy might decide to keep dirty pages in cache as many as possible to save the write cost on flash memory. However, by doing this, the cache will run out of space, and consequently the number of cache misses will be increased dramatically, which, in turn, will increase the replacement cost of reading requested from flash memory. On the other hand, a replacement policy that focuses mainly on increasing the cache hit count will evict dirty pages, which will increase the replacement cost of writing evicted pages into flash memory. Thus, a sophisticated scheme to compromise both sides of efforts is needed to minimize the total cost [6].

CFLRU (Clean-First LRU) page replacement policy is used for this purpose, which is modified from the LRU algorithm. CFLRU divides the LRU list into two regions to find a minimal cost point, as shown in figure below. The working region consists of recently used pages and most of cache hits are generated in this region. The clean-first region consists of pages which are candidates for eviction. CFLRU selects a clean page to evict in the clean-first region first to save flash write cost. If there is no clean page in this region, a dirty page at the end of the LRU list is evicted [6]. For example, under the LRU replacement algorithm, the last page in the LRU list is always evicted first. Thus, the priority for being a victim page is in the order of P8, P7, P6, and P5. However, under the CFLRU replacement algorithm, it is in the order of P7, P5, P8, and P6.

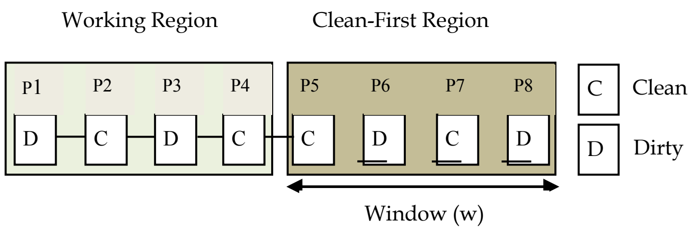


Figure 1. CFLRU Queue

The size of the clean-first region is called a window size, w. A large window size will increase the cache miss rate and a small window size will increase the number of evicted dirty pages, that is, the number of flash write operations. Therefore it is important to adjust the window size properly. This research has evaluated static window sized CFLRU page replacement policy empirically by taking different window sizes and suggested optimal window size for the policy.

III. RELATED WORK

Clean First Dirty Clustered (CFDC) [7] manages the buffer in two regions: the working region W for keeping hot pages that are frequently and recently revisited, and the priority region P responsible for optimizing replacement costs by assigning varying priorities to page clusters. A parameter λ, called priority window, determines the size ratio of P relative to the total buffer. Therefore, if the buffer has B pages, then P contains λ pages and the remaining (1-λ)*B pages are managed in W. Various conventional replacement policies can be used to maintain high hit ratios in W and, therefore, prevent hot pages from entering P. CFDC improves the efficiency of buffer manager by flushing pages in clustered fashion based on the observation that flash writes with strong spatial locality can be served by flash disks more efficiently than random writes. In paper [7] CFDC has been compared with LRU and CFLRU for different four workloads in database engine. The results show CFDC outperforms both competing policies, with a performance gain between 14% and 41% over CFLRU., in turn, is only slightly better than LRU with a maximum performance gain of 6%.

LRU-WSR [8] is a flash-aware algorithm based on LRU and Second Chance , using only a single list as auxiliary data structure. The idea is to evict clean and cold-dirty pages and keep the hot-dirty pages in buffer as long as possible. When a victim page is needed, it starts searching from the LRU end of the list. If a clean page is found, it will be returned immediately (LRU and clean-first strategy). If a dirty page marked as "cold" is found, it will also be returned; otherwise, it will be marked "cold" (Second Chance), moved to the MRU (most-recently used) end of the list, and the search continues. Although LRU-WSR considers the hot/cold property of dirty pages, which is not tackled by CFLRU, it has high dependency on the write locality of workloads. It shows low performance in case of low write locality, which may cause dirty pages to be quickly evicted. In paper [5], LRU-WSR has been compared with LRU, CFLRU algorithms for different workloads collected from PostgreSQL, GCC, Viewperf and Cscope. LRU-WSR has been found 1.4 times faster than LRU. In most of the cases, LRU-WSR has higher hit ratio and lower write count than others.

The authors of CCF-LRU [9] further refine the idea of LRU-WSR by distinguishing between cold-clean and hot clean pages. It maintains two LRU queues, a cold clean queue and a mixed queue to maintain buffer pages. The cold clean queue stores cold clean pages (first referenced pages) while mixed queue stores dirty pages or hot clean pages. It always selects victim from cold clean queue and if cold clean queue is empty then employs same policy as that of LRU-WSR to select dirty page from mixed queue. This algorithm focuses on reference frequency of clean pages and has little consideration on reference frequency of dirty pages. Besides, the CCF-LRU has no mechanism to control length of cold clean queue, which will lead to frequent eviction of recently read pages in the cold clean queue and lower the hit ratio. In paper [6] CCF- LRU has been compared with LRU, CFLRU and LRU-WSR with different four workloads. The results show that CCF-LRU performs better than LRU, CFLRU, and LRU-WSR with respect to hit rate, write count and run time.

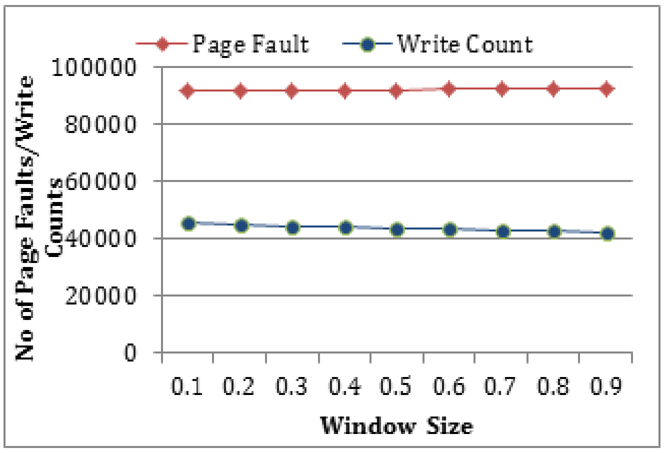
LIRS-WSR is an improvement of LIRS so that it can suit the requirements of flash-based systems. It integrates write sequence reordering (WSR) technique to original LIRS algorithm to reduce the number of page writes to flash memory. In paper [10] LIRS-WSR has been compared with LRU, CFLRU, LIRS and ARC for four different workloads: PostgreSQL, gcc, Viewperf and Cscope. LIRS-WSR has hit ratio very close approximate to LIRS and has higher hit ratio than other algorithms. LIRS-WSR has minimum write count than all other algorithms. In case of run time, LIRS-WSR is 2 times faster than LRU and 1.25 times faster than LIRS algorithm.

Adaptive double-LRU (AD-LRU) takes into account both reference frequency of clean pages and dirty pages and has a new mechanism to control the length of cold queue to avoid drop in hit ratio. In paper [11] AD-LRU has been compared with LRU, CFLRU, LRU-WSR, CCF- LRU algorithms for different four workloads: random, read-most, write-most and zipf traces. AD- LRU has been found better than other algorithms in terms of hit rate, write count and run time. In specific, AD-LRU reduced write count under zipf trace by 23%, 17% and 21% compared to LRU, CFLRU and LRU-WSR respectively.

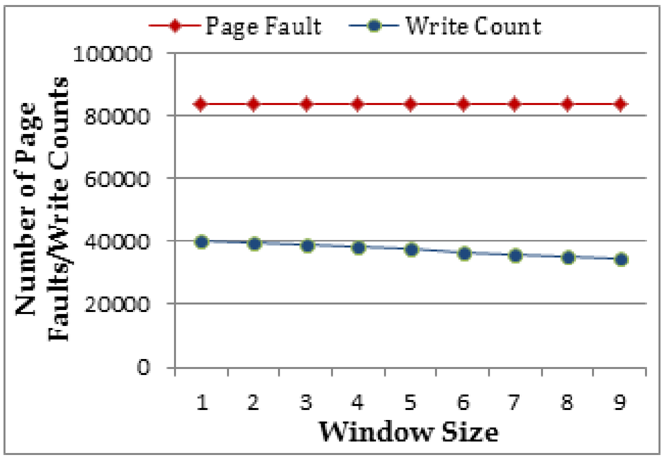
IV. DATA COLLECTION

In this research work four types of synthetic traces [11] have been used as input to simulated algorithm i.e., random trace, read- most trace (e.g., of decision support systems), write-most trace (e.g., of OLTP systems), and Zipf trace as Workload1, Workload 2, Workload 3 and Workload 4 respectively. These data are real memory traces. Workload represents different locality of memory reference pattern that are generated during execution of process in real OS. There are total 100,000 page references in each of the first three traces, which are restricted to a set of pages whose numbers range from 0 to 49,999. The total number of page references in the Zipf trace is set to 500,000 in order to obtain a good approximation, while the page numbers still fall in [0, 49999]. Zipf trace has a referential locality “20/80” meaning that eighty percent of the references deal with the most active twenty percent of the pages. Random, Read-most and write-most traces have a referential locality 50/50, 90/10, and 10/90 respectively.

V. RESULT ANALYSIS

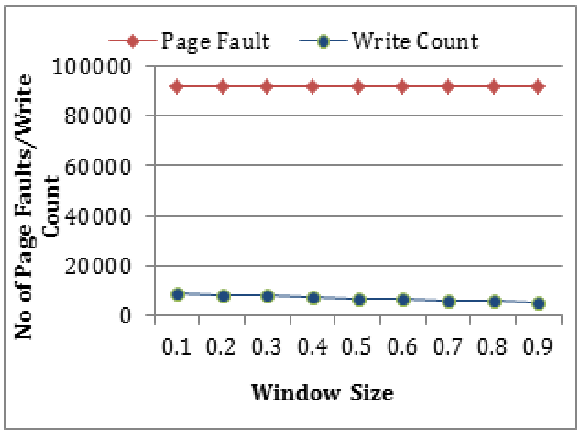


(a)

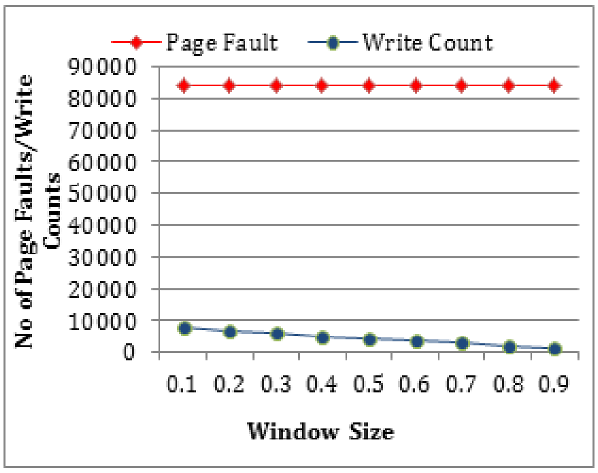


(b)

Figure 2. Graph for Random Trace (a) For Cache Size=4096 (b) For Cache Size=8192

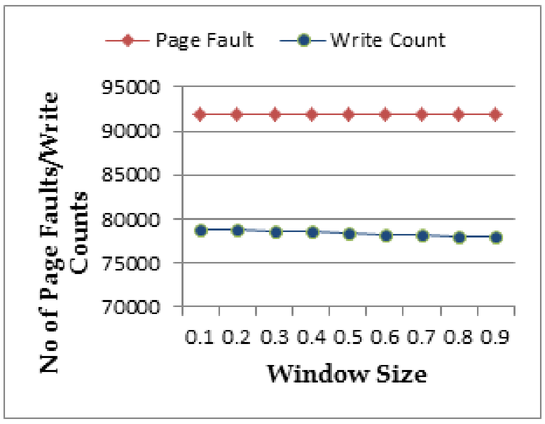


(a)

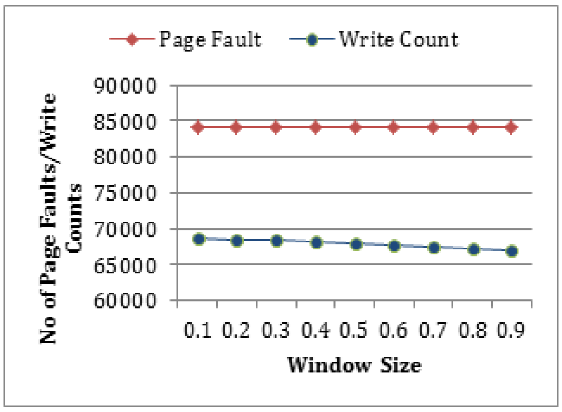


(b)

Figure 3. Graph for Read-most Trace (a) For Cache Size=4096 (b) For Cache Size=8192

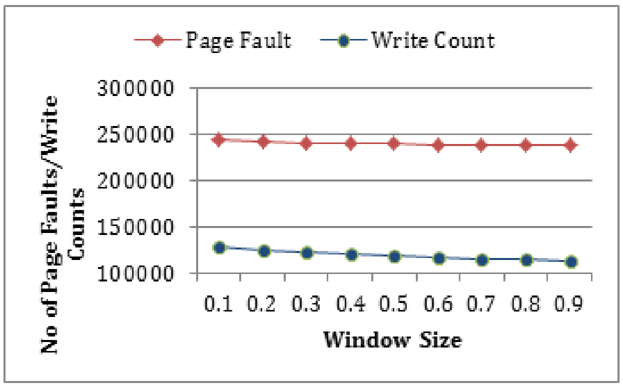


(a)

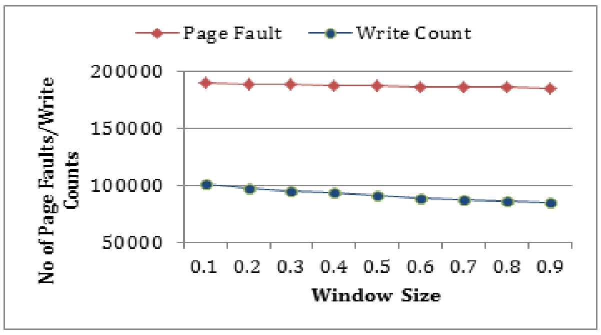


(b)

Figure 4. Graph for Write-most Trace (a) For Cache Size=4096 (b) For Cache Size=8192



(a)



(b)

Figure 5. Graph for Zipf Trace (a) For Cache Size=4096 (b) For Cache Size=8192

From above graphs we can observe that in case of graphs for workloads having uniform distribution (Figure 2 to Figure 4) of page references, number of page faults remains almost stagnant when we increase window size but number of write counts decreases significantly for increased window sizes. On the other hand, if we look at the graphs of workloads having high reference locality (Figure 5), we can observe that number of page faults decreases slightly along with increased window size but number of write counts again decreases significantly.

Further if we compare graph of figure 4 (graph for write-most trace) with other graphs, significant difference cannot be viewed. From this it can be observed that increased window size reduces numbers of write counts independently to the number of write operation contained in memory traces.

VI. CONCLUSION

Flash memory has become an alternative to the magnetic disks, which brings new challenges to traditional disk based system. To efficiently support the characteristics of flash storage devices, traditional buffering approaches need to be revised to take into account the imbalanced I/O property of flash memory. CFLRU prefers lest recently used clean pages while selecting victim page. This is because replacement of dirty pages requires writing them to flash memory, which is costly operation compared to reading a page in case of flash memories. To achieve this objective CFLRU divides set of pages into two regions, working region and clean-first region, and then tries to replace clean pages from clean first region. If no clean page is found in the region it simply uses LRU policy. Size of clean-first region is called window size (W). This research evaluates impacts of W in performance of CFLRU page replacement policy.

From the analysis it can be viewed that in case of memory traces having uniform distribution of pages, using higher window sizes decreases number of write counts without increasing number of page faults. At the same time we can see that that in case of memory traces having high locality of reference of pages, using higher window sizes decreases number of write counts significantly as well as decreases number of page faults slightly. Besides this, we have observed that ratio of write operations contained in memory traces does not affect performance of CFLRU policy with increased window sizes.

Thus it can be concluded that using larger window sizes provides better performance in all types of memory traces. Hence, window having size more that 70% of total buffer size is recommended for CFLRU policy.

VII. REFERENCES

[1]. L. P. Chang, T. W. Kuo, Efficient Management for Large-Scale Flash-Memory Storage Systems with Resource Conservation, ACM TOS 1 (4), 2005.

[2]. M. L. Chiang, C.H. Paul, R.C. Chang, Manage Flash Memory in Personal Communicate Devices. In: Proc. of IEEE Intl. Symposium on Consumer Electronics, IEEE Computer Society Press, Los Alamitos, 1997.

[3]. Jihyun In, Ilhoon Shin, Hyojun Kim, "SWL: A Search-While-Load Demand Paging Scheme with NAND Flash Memory", ACM, 2007

[4]. C. Park, J.U. Kang, S.Y. Park and J.S. Kim, "Energy aware demand paging on NAND flash-based embedded storages", Proc. of the international symposium on Low power electronics and design, 2004.

[5]. J. Kim, J. M. Kim, S. H. Noh, S. L. Min, Y. Cho, A Space-Efficient Flash Translation Layer for Compact Flash Systems, IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, May 2002.

[6]. S.Y. Park, D. Jung, J.U. Kang, J.S. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in Proc. Int. Conf. Compilers, Arch. Synthesis Embedded System, 2006

[7]. Y. Ou, T. Harder, P. Jin, "CFDC: a Flash-Aware Replacement Policy for Database Buffer Management", Proc. of the 5th International Workshop on Data Management on New Hardware, ACM, 2009.

[8]. H. Jung, H. Shim, S. Park, S. Kang, J. Cha, "LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory", IEEE Trans. On Consumer Electronics, 2008.

[9]. Z. Li, P. Jin, X. Su, K. Cui, L. Yue, "CCF-LRU: A New Buffer Replacement Algorithm for Flash Memory", Trans. on Cons. Electr, 2009.

[10]. H. Jung, K. Yoon, H. Shim, S. Park, S. Kang, J. Cha, "LIRS-WSR: Integration of LIRS and Writes Sequence Reordering for Flash Memory", ICCSA of LNCS, 2007.

[11]. P. Jin, Y. Ou, T. Harder, Z. Li, AD-LRU: An Efficient Buffer Replacement Algorithm for Flash-Based Databases, Data Knowledge Engineering, 2012.