Tribhuvan University
Institute of Science and Technology

A Comparative Evaluation of Buffer Replacement Algorithms LIRS-WSR and CCF-LRU for Flash Memory Based Systems

Dissertation Proposal
Submitted to

Central Department of Computer Science & Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements
for the Masters Degree in Computer Science & Information Technology

By
Mahesh Kumar Yadav
Date: February 13, 2017

Supervisor
Prof. Dr. Subarna Shakya, PhD

Co-Supervisor
Mr Arjun Singh Saud

TABLE OF CONTENTS

1. INTRODUCTION

Flash memory is non-volatile, shock resistant, and power economic. With recent technology breakthroughs in both capacity and reliability, flash-memory storage systems are much more affordable than ever. As a result, flash memory is now among the top choices for storage media in embedded systems [1].

There are two major types of flash memory in the current market: NAND and NOR flash memory. NAND flash memory is mainly designed for data storage, and NOR flash memory is for EEPROM replacement [1]. Flash memory has become a powerful and cost-effective solid-state storage technology widely used in mobile electronics devices and other consumer applications. NAND Flash, which was designed with a very small cell size to enable a low cost-per-bit of stored data, has been used primarily as a high-density data storage medium for consumer devices such as digital still cameras and USB solid-state disk drives. NOR Flash has typically been used for code storage and direct execution in portable electronics devices, such as cellular phones and PDAs [2]. However, several hardware limitations exist in a flash memory. Firstly, a data unit of erase operations is a block that is the set of fixed number of contiguous pages even if a data unit of read/write operations is a page. Secondly, it is impossible to re-write the page in-place in a flash memory. So, in order to update data of the page, a system should perform only one of the following:

1. Writing these data to a newly allocated page and invalidating the original page; or
2. Writing these data to the original page only after erasing the block containing that page.

Thirdly, the lifetime of a flash memory is shorter than the lifetime of a hard disk and a DRAM. In other words, only a limited number of erase operations can be performed safely to each memory cell, typically, between 100,000 and 1,000,000 cycles. Finally, there exist differences among I/O latencies according to the kinds of I/O operations, i.e., read, write, and erase. The write operation is about 10 times slower than the read operation, and the erase operation is about 20 times slower than the write operation [3, 4, 5].

Flash caching is needed for reducing flash I/O latencies. The traditional magnetic-disk-based buffering algorithms LRU [6], LIRS [7], ARC [8] etc. focus on hit-ratio improvement alone, but not on write costs caused by the replacement process. So, their straight adoption would result in poor buffering performance and would demote the development of flash-based systems. The replacement policy should minimize the number of writes and erases operations on flash memory and at the same time prevent the degradation of the hit ratio. Recently, CF-LRU [9], LIRS-WSR [3] and AD-LRU [10] were proposed as new buffering algorithms for flash-based systems. These new flash based buffer replacement policies consider not only buffer hit ratios but also replacement costs incurring when a dirty page has to be propagated to flash memory to make room for a requested page currently not in the buffer. These algorithms try to obtain the optimal I/O sequence from the given I/O sequence by discriminatively selecting the accesses according to the type of I/O operations. These algorithms favor to first evict clean pages from the buffer so that the number of writes incurring for replacements can be reduced.

1.1 LIRS-WSR

LIRS-WSR (Low Inter-reference Recency Set Write Sequence Reordering) [3] algorithm is designed for a buffer cache of the flash memory based storage system. The objective of LIRS-WSR is reducing the number of flushes of dirty pages from the buffer into flash memory when page replacement occurs, To achieve this objective, it uses the strategy: *delaying eviction of the page which is dirty and has high access frequency as possible*.

It enhances an existing LIRS buffer replacement algorithm with add-on buffer replacement strategy, namely Write Sequence Reordering (WSR). WSR reorders writing not-cold dirty pages from the buffer cache to the disk to reduce the number of write operations while preventing excessive degradation of the hit ratio. The LIRS [7] algorithm uses history information of data accesses in the form of two metrics - the Inter-Reference Recency (IRR) and the Recency. The IRR of a data block refers to the number of other distinct blocks accessed between the last two consecutive accesses of the data block in question while recency refers to the number of other distinct blocks accessed between the last reference to the current time.

LIRS algorithm uses two sets of pages based on IRR. Set of pages with low IRR value is taken as a hot block and called low inter-reference recency set (LIRS). Set of pages with high IRR value is taken as a cold block and called high inter-reference recency set (HIRS). Blocks that can be most probably used in future are taken as hot blocks whereas blocks that may not be used in near future are taken as cold blocks. Hence, HIR blocks are always replaced and LIR blocks are never replaced. LIRS always selects HIR page with the largest recency as a victim for replacement.

Write Sequence Reordering (WSR) policy is developed to adapt LIRS with flash memory [3]. The Basic scheme of WSR is following:

1. Use cold-detection algorithm to judge whether the page is cold or not; and
2. Delays flushing dirty pages which are not regarded as cold.

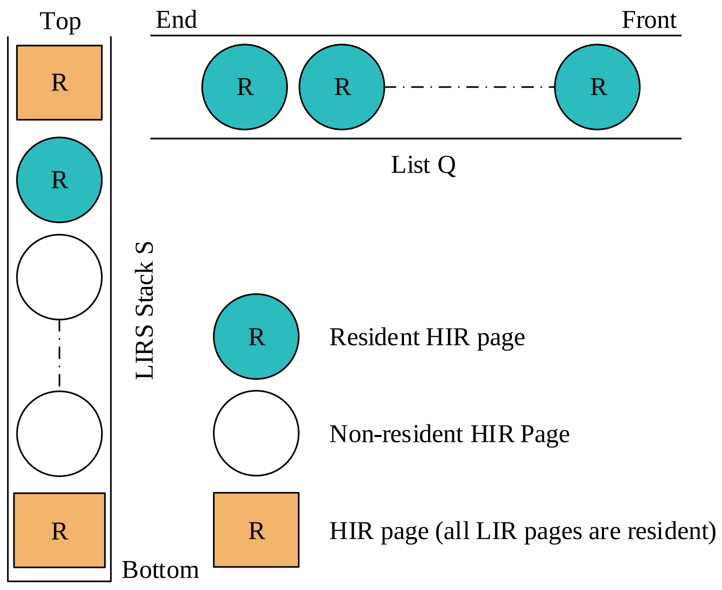

Figure 1.1: Two Lists of the LIRS algorithm [7]

LIRS-WSR is implemented using 2 lists as shown in Figure 1.1: LIR stack S which stores all LIR pages, as well as HIR pages, regardless of the residence status some of them are resident and others are not (actually, only their metadata are stored in the list) and HIR list Q that stores HIR resident pages. The operations on these two data structures are same that of LIRS. Every page has additional status either cold or not-cold. Initially, all pages are cold, this cold flag is cleared if the pages are referenced again when they are in stack S or queue Q. If a page is introduced to the buffer for a write request for the first time, it becomes a dirty page and enters the top of the stack S as an LIR page. Every time when Stack's bottom is moved to HIR Q, WSR policy is applied. That is, if bottom LIR page is dirty and not cold, then it's cold flag is set and moved to the head of Stack, otherwise, it is moved to the head of HIR Q. All other operations like pruning, switching between LIR and HIR pages are same as that of LIRS.

1.2 CCF-LRU

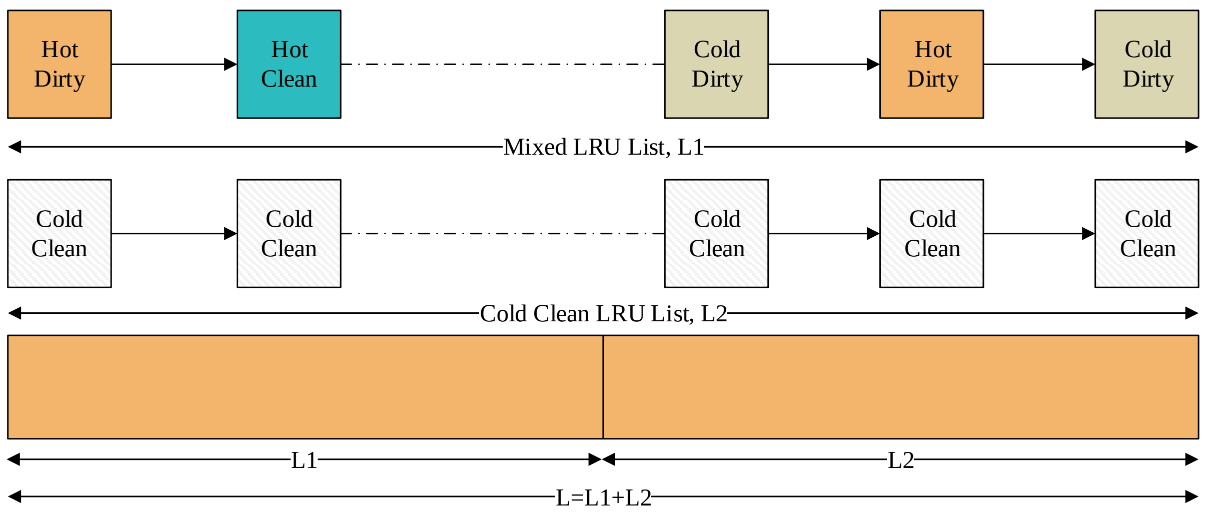

Figure 1.2: Mixed LRU list and cold clean LRU list in CCF-LRU [11]

An efficient buffer replacement algorithm for flash memory based storage systems called CCF-LRU [11] (Cold-Clean-First LRU). The goal of CCF-LRU is to improve the overall I/O performance by focusing on reducing the write count incurring in the replacement process. In order to accomplish this goal, it tries to first evict clean pages with low access frequencies. If there are no such clean pages, it will evict the dirty pages with low access frequencies instead of the clean pages with high access frequencies. Using the cold-detection mechanism of LRU-WSR, pages in the buffer list can be classified into the following four groups, namely cold clean page, hot clean page, cold dirty page and hot dirty page. A cold flag attached with each page is used to distinguish cold pages from hot pages. The CCF-LRU algorithm maintains two LRU lists, which are called mixed LRU list and cold clean LRU list. The mixed LRU list contains L1 pages and is used to maintain hot clean pages and dirty pages regardless of the status of its cold flag and the cold clean LRU list with the size of L2 is only for cold clean pages. If the buffer contains a total of L pages, the sizes of the two LRU lists are both from 0 to L. Moreover, the sum of L1 and L2 is L.

The first referenced pages are regarded as cold by default, each of which is inserted into the cold clean LRU list with a cold flag. When the page in the cold clean LRU list is referenced again or becomes dirty, it will be moved from the cold clean LRU list to the MRU position in the mixed LRU list. When the page in the mixed LRU list is referenced, it will be moved to the MRU position of the mixed LRU list. The CCF-LRU selects a victim page by the following rules in order:

1. If the cold clean LRU list is not empty, the LRU page in the cold clean LRU list is selected as the victim; and
2. If the cold clean LRU list is empty, the LRU page in the mixed LRU list is chosen as the victim candidate. If the candidate is a cold dirty page, it is selected as the victim. If the candidate is a hot dirty page, it is labeled as cold and moved to the MRU position of the mixed LRU list. If the candidate is a hot clean page, it is set to cold and moved from the mixed LRU list to the MRU position of the cold clean LRU list, and we continue to check the LRU position in the mixed LRU list. If there is no victim found after traversing the mixed LRU list, it needs to call the CCF-LRU algorithm one more time to select a victim.

2. PROBLEM DEFINITION

Since the use of flash memory requires buffer replacement policies for flash memory considering not only buffer hit ratios or miss ratios but also replacement costs incurring when a dirty page has to be propagated to flash memory, not in the buffer. There are many buffer replacement algorithms developed for flash-based systems. The traditional magnetic-disk-based buffering algorithms LRU [6], LIRS [7], ARC [8] etc. focus on hit-ratio improvement alone, but not on write costs caused by the replacement process. However, Flash memory has characteristics of out-of-place update and asymmetric I/O latencies for read-write and erase operations in the aspects of time and energy. So, the replacement algorithm with flash memory should consider not only the hit count but also the replacement cost caused by selecting dirty victim pages. There are many buffer replacement algorithms developed for flash-based systems. The evaluation of these buffer replacement algorithms for flash-based systems in terms of hit rate and write counts is required to rate their performance. This dissertation work will mainly focus on the comparative evaluation of two algorithms: LIRS-WSR and CCF-LRU.

3. OBJECTIVES

The main objectives of this dissertation work are:

- To perform a comparative study of LIRS-WSR and CCF-LRU buffer replacement algorithms for flash based systems in terms of hit rate and write count; and
- To evaluate the performance of LIRS-WSR and CCF-LRU.

4. LITERATURE REVIEW

4.1 Flash Memory

Flash memory has emerged as a high-performing and viable alternative to magnetic disks for data-intensive applications. As the density of a flash memory chip increases and the price continues to drop, the flash memory is being adopted in more diverse storage applications. For example, flash memory solid-state disks (SSDs) that provide the same interface as hard disk drives (HDDs) are replacing HDDs in mobile and general-purpose computers. Flash memory has characteristics that are different from conventional storage devices such as HDDs. Thus, specialized hardware and software are required to use flash memory as a storage device. The role of flash memory software is particularly important because it has to deal with the peculiarities of flash memory.

Table 4.1: Characteristics of flash memory [12]

Device	**Current** (mA)		**Access time** (4 kB)		
	Idle	Active	Read	Write	Erase
NOR	0.03	32	20 μs	28 ms	1.2 sec
NAND	0.01	10	25 μs	250 μs	2 ms

Unfortunately, flash memory has two critical drawbacks: First, blocks of memory need to be erased before they can be rewritten. This is because flash memory technology only allows the toggling of individual bits or bytes in one way for writes. The erase operation resets the memory cells with either all ones or all zeros; this needs more time than read or write operation (0.6-0.8 seconds). The second drawback is that the number of rewrite operations allowed to each memory cell is limited.

Compare to the magnetic disk, flash memory has special properties. Firstly, flash memory has no latency associated with the mechanical head movement to locate the proper position to read or write data. Secondly, flash memory has asymmetric read and write operation characteristic in terms of performance and energy consumption. Table 4.1 compares the access time and the energy consumption in flash memory when 4 kB data is read, written, or erased [12]. Thirdly, 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. If the erase operations are involved, the cost imbalance would be even worse. Finally, blocks of flash memory are worn out after the specified number of write/erase operations.

4.2 Traditional Buffer Replacement Algorithms

Buffer management is one of the key issues in memory management. Typically; the system has a two-level storage system: main memory and external (secondary) storage. Both of them are logically organized into a set of pages, where a page is the only interchanging unit between the two levels. When a page is requested from modules of upper layers, the buffer manager has to read it from secondary storage if it is not already contained in the buffer. If no free buffer frames are available, some page has to be selected for replacement. In such a scheme, the quality of buffer replacement decisions contributes as the most important factor to buffer management performance. Traditional replacement algorithms primarily focus on the hit ratio [8], because a high hit ratio will result in a better buffering performance. Many algorithms have been proposed so far, either based on the recency or frequency property of page references. Among them, the best-known ones are LRU [6], CLOCK [13], LRU-2 [14], LIRS [5], Clock-Pro [15], ARC [7,8] etc.

LRU always evicts the least-recently-used page from an LRU queue used to organize the buffer pages ordered by time of their last reference. It always replaces the page found at the LRU position. An important advantage of LRU is its constant runtime complexity. Furthermore, LRU is known for its good performance in case of reference patterns having high temporal locality, i.e., currently referenced pages have a high re-reference probability in the near future. But LRU also has severe disadvantages. First, it only considers the recency of page references and does not exploit the frequency of references. Second, it is not scan-resistant, i.e. a scan operation pollutes the buffer with one-time referenced pages and possibly evicts pages with higher re-reference probability.

CLOCK uses a reference bit which is set to 1 whenever the page is referenced [13]. Furthermore, it organizes the buffer pages as a circle, which guides the page inspection when a victim is searched. When a page currently inspected has a 1 in the referenced bit, it is reset to 0, but not replaced. The first page found having referenced bit 0 is used as the victim. Obviously, CLOCK does not consider reference frequency. Moreover, it is not scan-resistant, too.

The LRU policy takes into account the recency information while evicting pages, without considering the frequency. To consider the frequency information, LRU-K [14] was proposed which evicts pages with the largest backward K-distance. Backward K-distance of a page p is the distance backward from the current time to the Kth most recent reference to the page p. Since this policy considers Kth most recent reference to a page, it favors pages which are accessed frequently within a short time. Experimental results indicate that LRU-2 performs better than LRU [14]; while higher K does not result in an appreciable increase in the performance, but has high implementation overhead.

The LIRS (low inter-reference recency set) is an enhanced buffer replacement algorithm which captures both recency and frequency. LIRS maintains variable size LRU stack which classifies pages into LIR pages and HIR pages. LIR pages are those who have been accessed again while staying in the stack and HIR pages are those who were not in the stack (as a real page or metadata) when they were accessed. LIRS always selects the HIR page with the largest recency value among all HIR pages as a victim. LIRS algorithm usually outperforms LRU algorithm because it works well for looping pattern, for which LRU shows worst performance. However, it sometimes shows worse performance than LRU algorithm when the buffer cache size is larger than working set size. Also, since metadata of already evicted pages remain in the LIR stack, LIRS usually require more memory space than other buffer replacement algorithm.

CLOCK-Pro takes the same principle as that of LIRS-it uses reuse distance (called IRR in

LIRS) rather than recency in its replacement decision. When a page is accessed, the reuse distance is the period of time in terms of the number of other distinct pages accessed since its last access. A page is categorized as a cold page if it has a large reuse distance or as a hot page if it has a small reuse distance. Although there is a reuse distance between any two consecutive references to a page, only the most current distance is relevant in the replacement decision.

The ARC (Adaptive Replacement Cache) algorithm is another buffer replacement algorithm that outperforms the LRU algorithm. ARC maintains two variable sized LRU lists holding not only the pages in the cache but also the traces of replaced pages. The first LRU list contains cold pages which were referenced only once, recently and the second LRU list contains hot pages accessed at least twice, recently. The cache spaces allocated to the pages in these lists changes depending on the number of page misses occurred in each list: when a page miss occurs in a list then the size of the list decreases by 1 while that of the other list increases by 1.

The ARC algorithm is a low-overhead and scan-resident algorithm. And it is adaptive to the change of access pattern. However, in case that the size of buffer cache is a bit smaller than working set size, burst page misses occurs because hot pages not used anymore still reside in the buffer cache. There are other replacement algorithms that consider both recency and frequency such as LRFU [16], 2Q [17], and FBR [18]. However, they have similar shortcoming as LRU-2, i.e., they all are not self-tuning for different buffer sizes and workloads. All the traditional algorithms mentioned above do not consider the asymmetric I/O properties of flash memory. Yet, reducing the write/erase count for buffer management of flash-based systems is not only useful to improve the runtime but also helpful to extend the life cycle of flash memory. Hence, buffer replacement algorithms focusing on the avoidance of write operations have been investigated in recent years.

4.3 Buffer Replacement Algorithms for Flash-Based Systems

Existing buffer replacement algorithms are designed to maximize the page hit ratio. These algorithms treat the costs of page reads and writes as equal. However, because the write cost for evicting a dirty or modified page is much higher than the read cost in flash memory, existing algorithms may not maximize flash I/O performance. In [19], a buffer replacement algorithm called CF-LRU (Clean First LRU) was proposed. CF-LRU is a flash memory-aware page replacement algorithm that considers the different execution times for reading and writing.

Suppose pages were recently accessed in the order E, D, C, B, A, as illustrated in Figure 4.1 (so that A is the most recently used clean page and E is the least recently used dirty page). Under the LRU page replacement algorithm, the sequence of victim pages is E, D, C, B, always evicting the least recently used page first. When using NAND flash memory for storing victim page data, however, it may be advantageous to first evict the clean page D to reduce the number of flashes write operations, even though the page was more recently accessed than the dirty page E. As the page fault ratio may increase if the recently used clean page is evicted, only the clean pages within a predetermined window size (w) become candidate victims in CF-LRU.

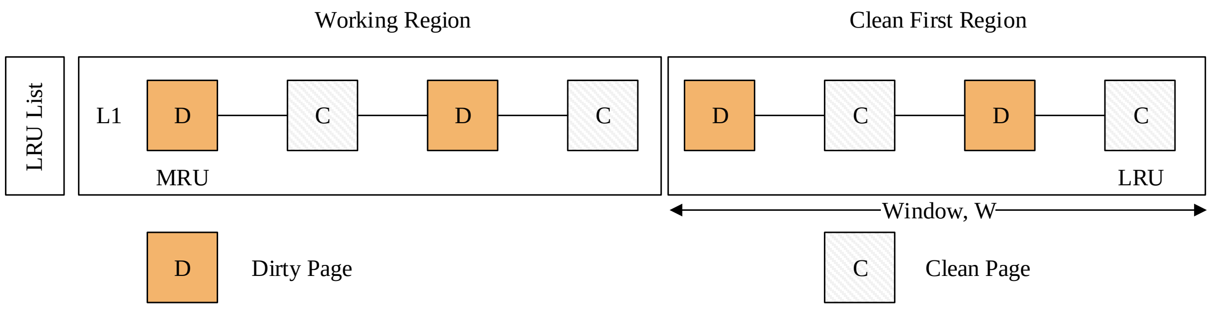

Figure 4.1: CF-LRU page replacement example [19]

If the algorithm does not find a clean page within the window, it defaults to the normal LRU algorithm, in which the least recently used page becomes the victim whether the page is dirty or not [19]. Despite that the hit ratio of CF-LRU may be lower than that of normal LRU, in many cases it reduces the numbers of write and erase operations more effectively. However, CF-LRU needs to determine w and thus is difficult to adapt to tasks with various workloads. CF-LRU also has a search overhead, as it should determine whether each page in the window is dirty. Above all things, it keeps both cold- and hot-write data; it sometimes performs more read operations than normal LRU, reducing performance. In particular, it needs an adaptive on-line algorithm to determine window size and should apply hot-cold identification to avoid keeping a cold-write page in the buffer.

5. RESEARCH METHODOLOGY

The research is to discover answers to the questions through the applications of scientific procedures. So, the main aim of the research is to find out the truth which is hidden and which has not been discovered yet. Out of different types of research methodologies, this dissertation is based on the trace-driven simulation approach. All the data collected are primary data, which are traces of page references. Output information gathered is analyzed in a quantitative approach. Finally, the conclusion is drawn with the help of analyzed data.

5.1 Data Collection and Analysis

Real Memory traces with different access pattern will be given as input to simulate algorithms and the output traces generated by the algorithms will be analyzed to calculate different performance metrics such page faults, hit rates, miss rates and write counts. The analyzed results will be shown by different tables and graphs to draw conclusions.

5.2 Program Development

For quantitative evaluation of the algorithms, the simulator program will be developed. The development of simulator program will be done in java programming language.

5.3 Data Structure Used

A doubly linked list and list will be used to implement both the buffer replacement algorithms because of its benefits over singly linked list. Each node of linked list will contain different necessary data and pointers as fields.

5.4 Performance Metrics

The off-line performance of buffer replacement algorithm is measured in terms of page fault count, hit rate and hit ratio, miss rate and miss ratio and write count. When an accessed block of memory is currently mapped to the physical memory then hit occurs. If it doesn't map them miss occurs. The Higher hit rate of the algorithm exhibits higher performance. In the case of flash based system, a minimum number of write count is a measure for optimal cost algorithm.

5.4.1 Page Fault Counts

Page Fault is an interrupt generated when the processor references a page that is neither in cache nor in main memory. An efficient page replacement algorithm always produces less number of page faults. It can be computed by counting the occurrences of a number of page faults between some intervals of references.

5.4.2 Hit/Miss Rate

When the processor needs to read or write a location in main memory, it first checks whether that memory location is in the cache. This is accomplished by comparing the address of the memory location to all tags in the cache that might contain that address. If the processor finds that the memory location is in the cache, we say that a **cache hit** has occurred; otherwise, we speak of a **cache miss**.

Miss rate can be calculated by using the formula:

miss rate = 1-hit rate.

Hit ratio is calculated by subtracting miss ratio from 1.

Miss ratio (mr) is calculated by using the formula:

mr = 100 * ((#pf - #distinct) / (#refs - #distinct))

where #pf is a number of page faults, #distinct is the number of distinct pages referenced and #refs is the total number of referenced pages [20].

Hit ratio is also calculated by dividing a total number of hit counts by a total number of reference counts.

Hit ratio = Total number of Hit Counts/Total number of Reference Counts

To represent it as a percentage: Hit% = Hit ratio * 100

5.4.3 Write Counts

Write count is a number of pages propagated to flash memory which can be calculated by counting the number of physical pages writes to flash memory and at the end of each test the dirty pages in the buffer are flushed to the flash memory to get exact write counts.

6. EXPECTED RESULT

Since buffer replacement algorithms for flash memory requires to consider not only buffer hit ratios or miss ratios but also replacement costs incurring when a dirty page has to be propagated to flash memory from the buffer, a replacement policy should minimize the number of write operations on flash memory and at the same time increase the hit ratio. As CCF-LRU tries to integrate the properties of frequency, and cleanness into the buffer replacement policy whereas LIRS-WSR considers the recency and cleanliness of page references, it is expected that CCF-LRU will outperform LIRS-WSR in terms of both hit rate and write counts.

7. TENTATIVE SCHEDULE

Tentative schedule of this dissertation work has shown in Table 7.1.

Table 7.1: Tentative schedule

Activities	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w	1w
Paper Study and Analysis	▨	▨	▨	7w	▨	▨	▨						
Proposal Preparation						▨	3w	▨					
Implementation							▨	3w	▨				
Testing and Data Collection								▨	3w	▨			
Documentation							▨	▨	6w	▨	▨	▨	
Review													1w
Presentation													*

References

[1] L.-P. Chang and T.-W. Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation," *ACM Transactions on Storage (TOS)*, vol. 1, no. 4, pp. 381–418, 2005.

[2] N. Toshiba, "vs. nor flash memory technology overview," tech. rep., Technical Report, 2006.

[3] H. Jung, K. Yoon, H. Shim, S. Park, S. Kang, and J. Cha, "Lirs-wsr: Integration of lirs and writes sequence reordering for flash memory," in *International Conference on Computational Science and Its Applications*, pp. 224–237, Springer, 2007.

[4] H.-j. Kim and S.-g. Lee, "A new flash memory management for flash storage system," in *Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International*, pp. 284–289, IEEE, 1999.

[5] E. Gal and S. Toledo, "Mapping structures for flash memories: techniques and open problems," in *IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'05)*, pp. 83–92, IEEE, 2005.

[6] A. Silberschatz, P. B. Galvin, G. Gagne, *et al.*, "Memory management strategies," *Operating System Concept, 8th ed. Wiley Student Edition*, pp. 315–417.

[7] S. Jiang and X. Zhang, "Lirs: an efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[8] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache.," in *FAST*, vol. 3, pp. 115–130, 2003.

[9] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, “Cflru: a replacement algorithm for flash memory,” in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pp. 234–241, ACM, 2006.

[10] P. 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, 2012.

[11] 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.

[12] M.-L. Chiang, P. C. Lee, and R.-C. Chang, “Managing flash memory in personal communication devices,” in *Consumer Electronics, 1997. ISCE’97., Proceedings of 1997 IEEE International Symposium on*, pp. 177–182, IEEE, 1997.

[13] F. Corbato, “Festschrift: In honor of pm morse, chapter a paging experiment with the multics system, pages 217–228,” 1969.

[14] E. J. O’neil, P. E. O’neil, and G. Weikum, “The lru-k page replacement algorithm for database disk buffering,” *ACM SIGMOD Record*, vol. 22, no. 2, pp. 297–306, 1993.

[15] S. Jiang, F. Chen, and X. Zhang, “Clock-pro: An effective improvement of the clock replacement.,” in *USENIX Annual Technical Conference, General Track*, pp. 323–336, 2005.

[16] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, “Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies,” *IEEE transactions on Computers*, vol. 50, no. 12, pp. 1352–1361, 2001.

[17] D. Shasha and T. Johnson, “2q: A low overhead high performance buffer management replacement algoritm,” in *Proceedings of the Twentieth International Conference on Very Large Databases, Santiago, Chile*, pp. 439–450, 1994.

[18] J. T. Robinson and M. V. Devarakonda, *Data cache management using frequency-based replacement*, vol. 18. ACM, 1990.

[19] C. Park, J.-U. Kang, S.-Y. Park, and J.-S. Kim, “Energy-aware demand paging on nand flash-based embedded storages,” in *Proceedings of the 2004 international symposium on Low power electronics and design*, pp. 338–343, ACM, 2004.

[20] H. Paajanen, “Page replacement in operating system memory management,” 2007.