Chapter-One

Background and Introduction

1.1Background

1.1.1 Flash Memory

Flash memory is a type of EEPROM, which was invented by Intel and Toshiba in 1980s. Unlike magnetic disks, flash memory does not support update in-place, i.e., previous data must be first erased before a write can be initiated to the same place. As another important property of flash memory, three types of operations can be executed: read, write, and erase[1]. In contrast, magnetic disks only support read and write operations. Moreover, all granularities and latencies of read and write operations differ for both device types.

Compared to magnetic disks, flash memory has the following special properties:

(1) It has no mechanical latency, i.e., seek time and rotational delay are not present.

(2) It uses an out-of-place update mechanism[2], because update in-place as used for magnetic disks would be too costly.

(3) Read/write/erase operations on flash memory have different latencies. While reads are fastest, erase operations are slowest.

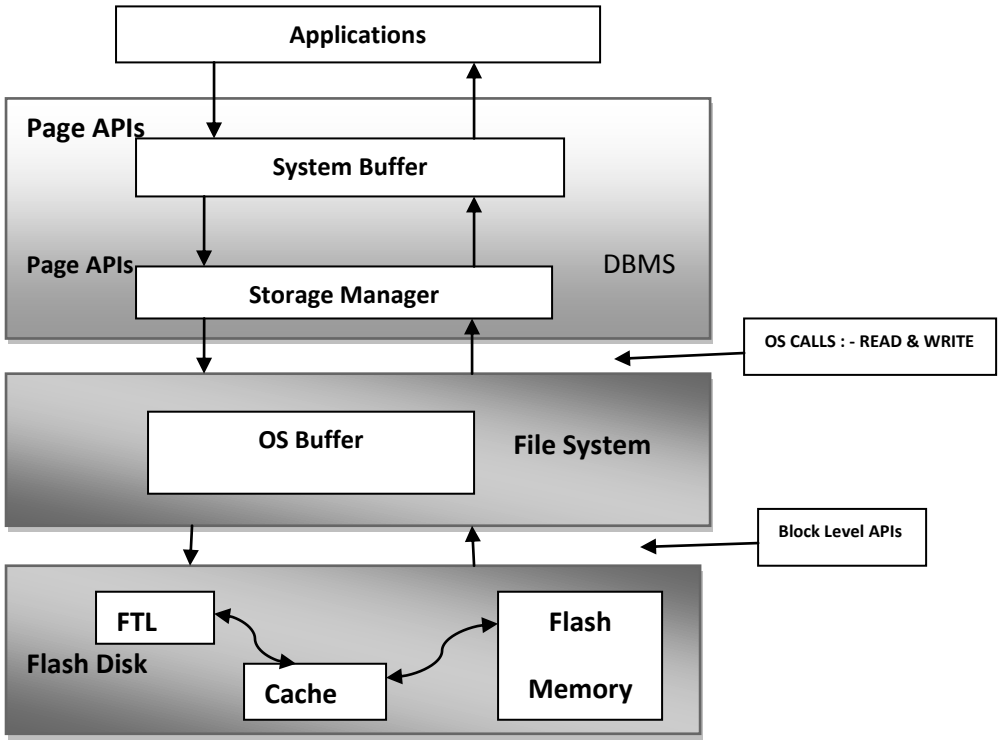


Fig.1.1 A simple architecture of Flash based storage systems

Flash memory is a complex technology, and many factors impact its overall performance, reliability, and suitability for a particular application.Flash memories store data as charge trapped on a floating gate between the control gate and the channel of a CMOS transistor. Each gate can store one or more bits of information depending on whether it is a single-level cell (SLC) or a multi-level cell (MLC). Commercially available devices store between 1 and 4 bits per cell [3].

The ever increasing requirement for high performance and high capacity memories of emerging handheld devices or applications has led to the widespread adoption of DRAM and NAND type flash memories[4], respectively. As the handheld devices are becoming multifunctional day to day and the size & the number of applications is rapidly increasing, so they demand more hardware resources but their production cost is accordingly becoming higher.

Flash memory has characteristics of out-of-place update and asymmetric I/O latencies for read write and erase operations. Thus, the buffering policy for flash based databases has to improve the overall performance. In recent years, flash memory greatly gained acceptance in various embedded computing systems and portable devices such as PDA's, HPC's, PMP's and mobile phones because of low cost, volumetric capacity and low power consumption [5]. There are generally two methods to execute applications on those types of devices: eXecute-In-Place on NOR flash memory (NOR-XIP) and second method is known as shadowing. But, these two methods have the common problem of high production cost; a Demand Paging Scheme is the better choice. Since demand paging scheme[2] stores the applications code to a cheap secondary memory (OneNAND) and loads the required pages to the main memory (DRAM) on demand as shown in fig.1.2 below:

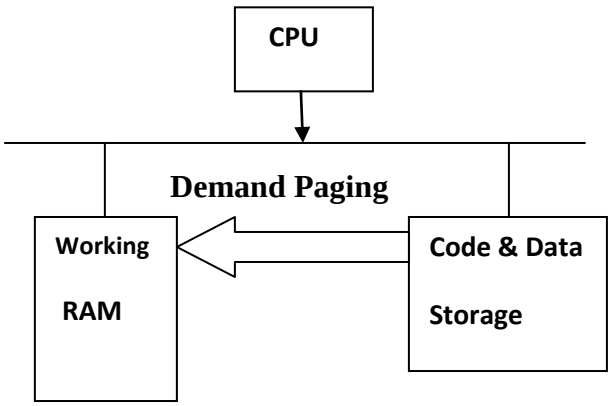


Fig.1.2 Demand Paging using OneNAND

Flash memory usually consists of many blocks and each block contains a fixed set of pages [3]. Read/write operations are performed on page granularity, whereas erase operations are relatively slow compared to read operation. Typically, write operations are about ten times slower than read operations and erase operations are about ten times slower than write operations [6].

Since flash memory has become a serious disk alternative, and since traditional;(magnetic-disk-based) buffering algorithms do not consider the differing I/O latencies of flash memory. So their straight adoption would result in poor buffering performance and would demote the development of flash based systems. The use of flash memory requires new buffer replacement policies 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. As a consequence a replacement policy should minimize the number of write/erase operations on flash memory and at the same time increase the hit ratio. Most of the traditional buffer replacement algorithms focused on the hit ratio improvement alone, but not the write costs caused by the replacement process.

1.1.2 Performance Matrices

Offline performance of buffer replacement algorithm is measured in terms of page fault count, write counts, hit rate and hit ratio, miss rate and miss ratio etc as follows:

1.1.2.1 Page Fault Count

An efficient page replacement algorithm always computes less number of page faults. It can be computed by counting occurrences of number of page faults between some intervals of references.

1.1.2.2 Write counts

Write count[5] is the number of pages propagated to flash memory which can be calculated by counting the number of physical page 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.

1.1.2.3 Hit and Miss rates

Hit rate can be calculated by using formula: **hr = 100 - mr,** where hr is the hit rate and mr is the miss rate. Hit rate is the percentage calculation of the fraction hit ratio. Hit ratio can be calculated by subtracting miss ratio from 1.

Miss rate (mr) can be calculated by using formula:

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

Where #pf is the number of page faults, #distinct is the number of distinct pages referenced and #refs is the total number of pages referenced. Miss Ratio is the fraction number of page fault and reference ignoring the distinct references.

1.1.3 Program Behavior

There are several factors that influence performance of buffer replacement algorithms. The performance of buffer replacement algorithm relies on pattern of pages that are referenced. Behavior of program depends upon the access pattern it references memory which is further depends upon working set, locality of reference and write or read mode natures of the reference pages.

1.1.3.1 Locality of Reference

During the course of execution of program memory references tend to cluster forming certain locality. Locality varies on the basis of time and space. Temporal locality is based on time, it assumes that memory location referenced just now is likely to be reference again in near future. Looping, subroutines, stacks, variable used for counting & totaling etc supports this assumption. Spatial locality is based on space, is assumes that once a memory is referenced there is high chance of nearby memory location to be referenced again. Array traversal, sequential code execution, related variable declaration nearby in source code supports this assumption. Hints of locality are followed in any type memory reference sequence.

1.1.3.2 Memory Reference Patterns

Altogether three types of standard synthetic traces i.e. random traces, readmost traces, and writemost traces are used in this dissertation.

Traces:- Traces are the page references that are supplied to the algorithm as input for testing and the output generated is then analyzed in this dissertation three standard input traces are taken which are described below.

i. **Random Traces**

The page references having random read and write nature of pages are called random traces.

ii. **WriteMost Traces**

The page references having most of the pages with write mode nature are called writemost traces.

iii. **Readmost Traces**

The page references having most of the pages with read mode nature are called readmost traces.

In order to obtain a good approximation, there are total 1,00,000 page references in each of the three traces, which are restricted to a set of pages whose numbers range from 1 to 49,999.

1.2Introduction

1.2.1Problem Statement

Since, in most operating systems which are customized for disk-based storage system, the replacement algorithm concerns only the number of memory hits. However, flash memory has asymmetric (different) read and write/erase cost 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. The replacement cost of dirty page is higher than that of clean page with regard to both access time and energy consumption [7].

There are many buffer replacement algorithms developed for flash based systems. Hence the quantitative evaluation of these buffer replacement algorithms for flash based systems is required in terms of write counts and hit rate. This dissertation will mainly focus on comparative evaluation of three algorithms:LRU[8] that only considers recency of workloads, CFLRU: that considers recency & cleanliness and AD-LRU: that considers recency, cleanliness & frequency.

For this :- the optimal value for the **window size (w)** of CFLRU repalcement algorithm will be determined by varying its size from 0 to 0.9 for different workloads and Finally, the obtained optimal value of window size (w) is used for CFLRU and the three algorithms LRU,CFLRU and AD-LRU are compared and evaluated.

1.2.2Objectives

The main objectives of this dissertation work are:

i. To evaluate the optimal values of window size **w** of CFLRU algorithm.

ii. To evaluate the performances of LRU, CFLRU & ADLRU algorithms and to perform the comparative study of LRU, CFLRU and AD-LRU buffer replacement algorithms for flash based systems

1.2.3Motivation

Since, the use of flash memory requires buffer replacement policies 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. As a consequence a replacement policy should minimize the number of write/erase operations on flash memory and at the same time increase the hit ratio.

The three algorithms taken has different characteristics with different nature hence their comparision is necessary. LRU only considers the recency of memory reference while CFLRU considers recency & cleanliness properties of the reference pages but AD-LRU not only considers recency & Cleanliness but also the frequency of the reference patterns which is shown in fig below:-

	LRU	CFLRU	AD-LRU
Recency	YES	YES	YES
Cleanliness	NO	YES	YES
Frequency	NO	NO	YES

Table.1.1 Motivation of the Dissertation

1.2.4 Dissertation Organization

Background part of this dissertation work focuses on buffer replacement algorithm and the related basic terms and terminologies which are already mentioned above along with an introduction to AD-LRU. Some more chapters are remaining which clarifies the topics fulfilling the objectives of this dissertation work. Chapter 2 consists of literature review which briefly reviews the related topics. Literature review includes details of several page replacement algorithms etc. within their category. This chapter also contains the research methodology part which shows the flow of my research. Chapter 3 consists of program development steps of our simulation. It includes the algorithm description and algorithm tracing for certain manual input traces. Chapter 4 includes detail design of the program. Also it includes details about the data structures and programming language used to build the simulation. Chapter 5 consists of data collection and analysis part which includes details about the traces taken in this dissertation, output results with several analyzing graphs. Chapter 6 consists of conclusion of this whole dissertation work and the future work which shows guidelines for further research work.

Chapter – Two

Literature Review and Methodology

2.1 Literature Review

2.1.1 Buffer Replacement Algorithms

Buffer management is one of the key issues in operating systems. Typically; system uses two-level storage systems: 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. And hence those algorithms evaluate their performances based on not only the hit and miss rates but also considering the number of read, write and erase operations that occur during the replacement process. The different categories are discussed below.

2.1.2 Traditional (Disk based) Buffer Replacement Algorithms

The best-known traditional (Disk Based) buffer replacement algorithms are:-

2.1.2.1 OPT Page Replacement al**gorithm**

The best possible page replacement algorithm is easy to describe but impossible to implement. It goes like this. At the moment that a page fault occurs, some set of pages is in memory. One of these pages will be referenced on the very next instruction (the page containing that instruction). Other pages may not be referenced until 10, 100, or perhaps 1000 instructions later. Each page can be labeled with the number of instructions that will be executed before that page is first referenced.The only problem with this algorithm is that it is unrealizable. At the time of the page fault, the operating system has no way of knowing when each of the pages will be referenced next. Still, by running a program on a simulator and keeping track of

all page references, it is possible to implement optimal page replacement on the second run by using the page reference information collected during the first run.

From the past experiences and research papers the researches on the page replacement algorithms are categorized into LRU based replacement algorithms and CLOCK[9] based replacement algorithms.

2.1.2.2 LRU Based Page Replacement Algorithms

2.1.2.2.1 General LRU Page Replacement Algorithm

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in the last few instructions will probably be heavily used again in the next few. Conversely, pages that have not been used for ages will probably remain unused for a long time. This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been unused for the longest time. This strategy is called LRU (Least Recently Used) paging [8].

Although LRU is theoretically realizable, it is not cheap. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operation, even in hardware (assuming that such hardware could be built).

However, there are other ways to implement LRU with special hardware. Let us consider the simplest way first. This method requires equipping the hardware with a 64-bit counter, *C*, that is automatically incremented after each instruction. Furthermore, each page table entry must also have a field large enough to contain the counter. After each memory reference, the current value of *C* is stored in the page table entry for the page just referenced. When a page fault occurs, the operating system examines all the counters in the page table to find the lowest one. That page is the least recently used.

2.1.2.2.2 NRU Page Replacement Algorithm

The not recently used (NRU), sometimes known as the Least Recently Used (LRU), page replacement algorithm is an algorithm that favors keeping pages in memory that have been

recently used. This algorithm works on the following principle: when a page is referenced, a referenced bit is set for that page, marking it as referenced. Similarly, when a page is modified, a modified bit is set. The setting of the bits is usually done by the hardware, although it is possible to do so on the software level as well.

When a page needs to be replaced, the operating system divides the pages into four classes:

3. referenced, modified

2. referenced, not modified

1. not referenced, modified

0. not referenced, not modified

Although it does not seem possible for a page to be not referenced yet modified, this happens when a class 3 page has its referenced bit cleared by the clock interrupt. The NRU algorithm picks a random page from the lowest category for removal. So out of the above four pages, the NRU algorithm will replace the not referenced, not modified [2].

2.1.2.2.3 MRU Page Replacement Algorithm

Most Recently Used (MRU) algorithm [10, 11] works on the basis of recency factor as in LRU. It violates LRU principle and works totally in opposite manner. LRU evicts unused page following locality of principle but MRU evicts recently used page as victim. MRU is only suitable when there weak locality of reference, which is worst case of LRU. MRU can be implemented in similar way as LRU by maintaining recency stack. But here front one is removed and bottom one is stored for future use. Hence MRU is only suitable in case of worst locality of reference where LRU could not deal with this effect.

2.1.2.2.4 LFU Page Replacement Algorithm

Often confused with LRU, Least Frequently Used (LFU) [12] selects a page for replacement if it has not been used often in the past. Instead of using a single age as in the case of LRU, LFU defines a frequency of use associated with each page. This frequency is calculated throughout the reference stream, and its value can be calculated in a variety of ways. The most common frequency implementation begins at the beginning of the page reference stream, and continues to calculate the frequency over an ever-increasing interval. Although this is the most accurate representation of the actual frequency of use, it does have some serious drawbacks. Primarily,

reactions to locality changes will be extremely slow[5]. Assuming that a program either changes its set of active pages, or terminates and is replaced by a completely different program, the frequency count will cause pages in the new locality to be immediately replaced since their frequency is much less than the pages associated with the previous program. Since the context has changed, and the pages swapped out will most likely be needed again soon (due to the new program's principal of locality), a period of thrashing will likely occur. If the beginning of the reference stream is used, initialization code of a program can also have a profound influence, as described by [10]. The pages associated with initial code can influence the page replacement policy long after the main body of the program has begun execution.

2.1.2.2.5 LRFU Page Replacement Algorithm

Having analyzed the advantages and disadvantages of LRU and LFU, A new algorithm LRFU[12] is proposed by combining them through weighing block recency and frequency factors. The performance of the LRFU algorithm largely relies on a parameter called (Lambda), which determines the relative weight of LRU or LFU and has to be adjusted according to the system configurations, even according to different workloads.

2.1.2.2.6 LRU-K Page Replacement Algorithm

LRU - K [11] evicts the page that is the one whose backward K-distance is the maximum of all pages in buffer. Backward K-distance bt(p,K) can be defined as the distance backward to the Kth most recent reference to page p where reference string known up to time t (r1, r2, …,rt). The value of parameter K can be taken as 1, 2 or 3. If K=1, it works as simple LRU algorithm. Highly increasing value of K the overall performance of algorithm reduces. LRU-K can discriminate better between frequently referenced and infrequently referenced pages. Unlike the approach of manually tuning the assignment of page pools to multiple buffer pools, LRU-K does not depend on any external hints. Unlike LFU and its variants, our algorithm copes well with temporally clustered patterns.

2.1.2.2.7 2Q Page Replacement Algorithm

The LRU-2 makes its replacement decision based on the time of the second to last reference to the block and evicts the oldest resident block. The 2Q [13] quickly removes from the buffer

cache, sequentially-referenced blocks and looping-referenced blocks with long loop periods by using a special buffer called the A1in queue in which all missed blocks are initially placed and from which the blocks are replaced in the FIFO order short residence. This algorithm uses special buffer queue A1in of size Kin, ghost buffer queue A1out of size Kout and the main buffer Am. Special buffer contains all missed that is first time referenced block. Ghost buffer contains replaced blocks from special buffer. Frequently accessed block are available in main buffer. Hence victim blocks are always from special buffer and main buffer.

2.1.2.2.8 LIRS Page Replacement Algorithm

LIRS[14] is one of the important replacement algorithms especially for weak locality or references. Here pages are categorized into two groups: High Inter-reference Recency (HIR) block set and Low Inter-reference Recency (LIR) block set. Each block with history information in cache has a status {either LIR or HIR. Some HIR blocks may not reside in the cache, but have entries in the cache recording their status as HIR of non-residence. Divide the cache, whose size in blocks is L, into a major part and a minor part in terms of the size. The major part with the size of Llirs is used to store LIR blocks, and the minor part with the size of Lhirs is used to store blocks from HIR block set, where Llirs + Lhirs = L. When a miss occurs and a free block is needed for replacement, we choose an HIR block that is resident in the cache. LIR block set always resides in the cache and there are no misses for the references to LIR blocks. However, a reference to an HIR block would likely to encounter a miss, because Lhirs is very small (its practical size can be as small as 1% of the cache size).

The main objective of LIRS is to minimizing the deficiencies presented by LRU using an additional criterion named IRR (Inter- Reference Recency) that represents the number of different pages accessed between the last two consecutive accesses to the same page. The algorithm assumes the existence of some behavior inertia and, according to the collected IRRs, replaces the page that will take more time to be referenced again. This means that LIRS does not replace the page that has not been referenced for the longest time.

2.1.2.2.9 ARC Page Replacement Algorithm

Adaptive Replacement Cache (ARC) [15] improves the basic LRU strategy by splitting the cache directory into two lists, T1 and T2, for recently and frequently referenced entries. In

turn, each of these is extended with a ghost list (B1 or B2), which is attached to the bottom of the two lists. These ghost lists act as scorecards by keeping track of the history of recently evicted cache entries, and the algorithm uses ghost hits to adapt to recent change in resource usage. Note that the ghost lists only contain metadata (keys for the entries) and not the resource data itself, i.e. as an entry is evicted into a ghost list its data is discarded. The combined cache directory is organized in four LRU lists:

1. T1, for recent cache entries.

2. T2, for frequent entries, referenced at least twice.

3. B1, ghost entries recently evicted from the T1 cache, but are still tracked.

4. B2, similar ghost entries, but evicted from T2.

T1 and B1 together are referred to as L1, a combined history of recent single references. Similarly, L2 is the combination of T2 and B2.

2.1.2.3 ClOCK based Page Replacement Algorithms

2.1.2.3.1 CLOCK Page Replacement Algorithm

Research and experience have shown that CLOCK [15] is close approximation of LRU, and its performance characteristics are very similar to those of LRU. So all the performance disadvantages about LRU are also applied to CLOCK. In CLOCK, the memory spaces holding the pages can be regarded as a circular buffer. Here each page is associated with a bit, called reference bit, which is set by hardware whenever the page is accessed. When it is necessary to replace a page to service a page fault, the page pointed to by the hand is checked. If its reference bit is unset, the page is replaced. Otherwise, the algorithm resets its reference bit and keeps moving the hand to the next page.

2.1.2.3.2 GCLOCK Page Replacement Algorithm

In generalized CLOCK page replacement algorithm each page frame in memory associate a count field and arrange these count fields in a circular list [15]. Whenever a page is referenced, the associated count field is set to i. When a page fault occurs, a pointer that circles around this circular list of page frames is observed. If the count field pointed to is zero,

then the page is removed and the new page is placed in that frame. Otherwise, the count is decremented by 1, the pointer is advanced to the next count field, and the process is repeated. When a new page is placed in the page frame, the count field is set to i if the page is to be referenced (demand fetch) and it is set to j if the page has been pre-paged and is not immediately referenced. This algorithm abbreviated by writing CLOCKP (j, i). The "P" indicates that this is a pre-paging algorithm (the pre-paging strategy has not been specified).

2.1.2.3.3 CAR Page Replacement Algorithm

Another CLOCK based algorithm is CAR[16] (CLOCK with adaptive replacement), this algorithm uses two clocks T1 & T2 and two lists B1 & B2. T1 and T2 contain cold pages and hot pages i.e. contain pages in the cache, while B1 & B2 maintain history information about the recently evicted pages from B1 & B2 respectively.

2.1.2.3.4 CART Page Replacement Algorithm

A limitation of ARC and CAR is that two consecutive hits are used as a test to promote a page from "recency" or "short-term utility" to "frequency" or "long-term utility". At upper level of memory hierarchy, we often observe two or more successive references to the same page fairly quickly. Such quick successive hits are known as "correlated references" [12] and are typically not a guarantee of long-term utility of a page, and, hence, such pages can cause cache pollution–thus reducing performance. The motivation behind CART is to create a temporal filter that imposes a more stringent test for promotion from "short-term utility" to "long-term utility". The basic idea is to maintain a temporal locality window such that pages that are re-requested within the window are of short-term utility whereas pages that are re-requested outside the window are of long-term utility. Furthermore, the temporal locality window is itself an adaptable parameter of the algorithm.

2.1.2.3.5 CLOCK-Pro Page Replacement Algorithm

Another important algorithm is CLOCK-Pro whose objective is to minimize the fault rate in weak locality of references[17] and also increases the performance of a computer because it does not need to movement of pages in case of page hit. But normally such case not takes place in other replacement algorithms.

2.1.3.6 Adaptive CLOCK-Pro Page Replacement Algorithm

Its objective is to minimize the fault rate in weak locality of references like CLOCK-Pro only difference is that here cold page size is varying dynamically.

2.1.3 Buffer Replacement Algorithms for Flash-based Systems

2.1.3.1 CFLRU

Since 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 page 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. For this purpose CFLRU [18] (Clean-First LRU), which is modified from the LRU algorithm, can be the best solution. CFLRU divides the LRU list into two regions to find a minimal cost point.

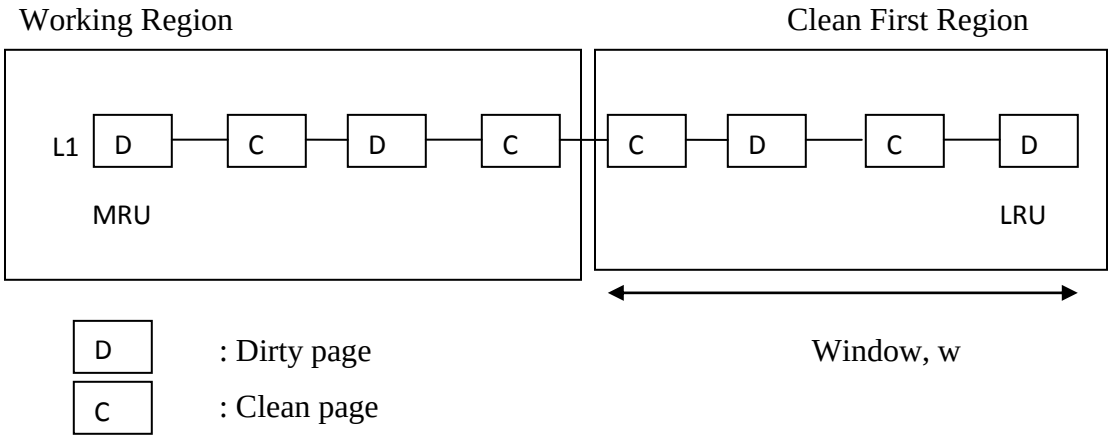


Fig.2.1 Example of CFLRU algorithm

2.1.3.2 ARC

ARC[19] maintains two LRU pages lists: L1 and L2. L1 maintains pages that have been seen only once, recently, while L2 maintains pages that have been seen at least twice, recently. The

algorithm actually caches only a fraction of the pages on these lists. The pages that have been seen twice within a short time may be thought of as having high frequency or as having longer term reuse potential. Hence, we say that L1 captures recency, while L2 captures frequency. If the cache can hold c pages, we strive to keep these two lists to roughly the same size, c. Together, the two lists comprise a cache directory that holds at most 2c pages. ARC caches a variable number of most recent pages from both L1 and L2 such that the total number of cached pages is c. ARC continually adapts the precise number of pages from each list that are cached.

2.1.3.3 CFDC (Clean First Dirty Clustered)

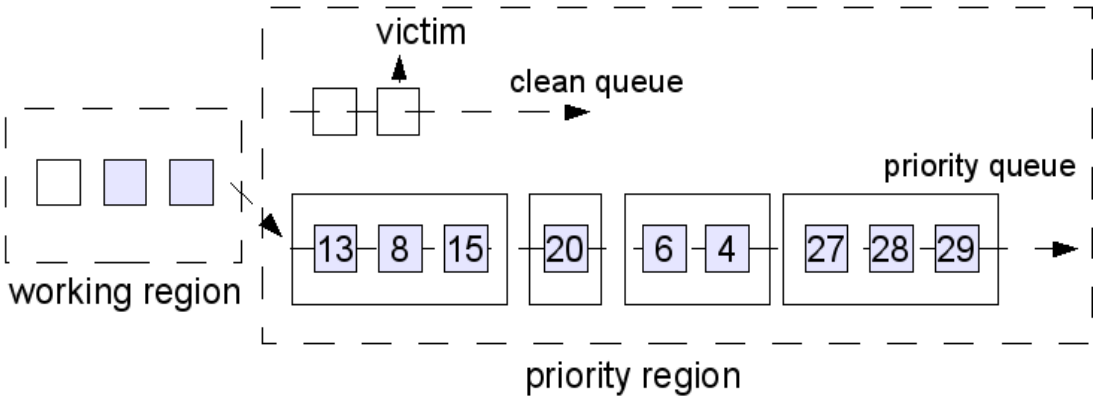


Fig. 2.2 Example of CFDC

CFDC[20] manages the buffer in two regions: the working region 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 $\lambda.b$ pages and the remaining $(1-\lambda)\cdot b$ pages are managed in W.CFDC algorithm is a generlized two-region scheme, independent of LRU in which the parameter priority window is the size of the priority region.This algorithm performs the seperation of clean and dirty pages where dirty pages are grouped in clusters and clusters are ordered by priority.

Since CFDC is the improvement of the CFLRU algorithm and CFDC improves the effciency of the buffer manager by using pages in a clustered fashion, based on the oberservation that flash writes with strong spatial locality can be served by flash disks more efficiently than random writes.

2.1.3.4 CASA (Cost Aware Self Adaptive)

The CASA[20] algorithm uses the notion of cost ratio to refer to the extent of R/W asymmetry of the underlying storage device, defined as the ratio of the long-term cost of physical reads to that of physical writes.

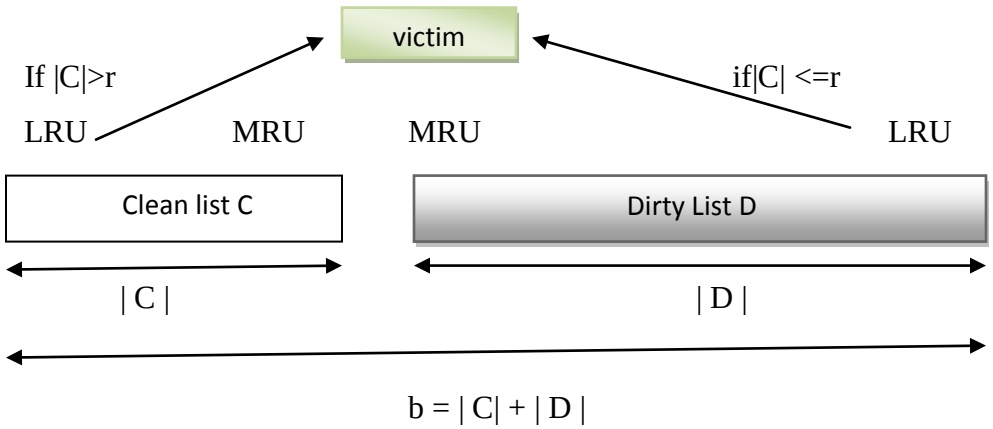


Figure 2.3 CASA dynamically adjusts the size of the clean list and the dirty list

CASA manages the buffer pool B of b pages using two dynamic lists: the clean list C for keeping clean pages that are not modified since being read from secondary storage, and the dirty list D accommodating dirty pages that are modified at least once in the buffer. Pages in either list are ordered by reference recency. Both lists are initially empty while, in the stable state (no empty buffer frames available).

2.1.3.5 LRU-WSR

LRUWSR[5] 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 search from the LRU end of the list. If a clean page is visited, it will be returned immediately (LRU and clean-first strategy). If a dirty page is visited and is marked “cold”, it will be returned; otherwise, it will be marked “cold” (Second Chance) and the search continues.

2.1.3.6 LIRS-WSR

The LIRS-WSR algorithm[21] is an improvement of LIRS[14] so that it can suit the requirements of flash-based systems. However, LIRS-WSR has the same limitation as CFLRU and LRU-WSR, because it is not self-tuning, too, and hardly considers the reference frequency. Frequently referenced pages may be evicted before a cold dirty page, because a dirty page is always put on the top of the LIRS stack, irrespective of its reference frequency. Moreover, LIRS-WSR needs additional buffer space, because it has to maintain historical reference information for those pages that were referenced previously, but are currently not in the buffer.

2.1.3.7 AD-LRU

AD-LRU is also one of the efficient buffer replacement policy developed for flash based systems, which also focuses on reducing the number of write/erase operations as well as maintain high buffer hit ratio. This algorithm not only considers frequency but also the recency of page references and cleanliness of pages.

The AD-LRU[22] algorithm is summarized as follows:

(1) Two LRU queues are used to capture both the recency and frequency of page references, among which one cold LRU queue stores the pages referenced only once and the hot LRU queue maintains the pages referenced at least twice.

(2) The sizes of the double LRU queues are dynamically adjusted according to the changes in the reference patterns. The size of the hot LRU queue is increased and the size of the cold one when a page in the cold queue is re-referenced is decreased. The hot queue shrinks when a page is selected as victim and moved from there to the cold queue.

(3) During the eviction procedure, at first least-recently-used clean page from the cold LRU queue as the victim is selected, for which a specific pointer FC is used. If clean pages do not exist in the cold LRU queue, second-chance policy[5] can be used to select a dirty page as the victim. For this reason, each page in the double LRU queues is marked by a referenced bit, which is always set to 1, when the page is referenced. Hence, the second-chance policy ensures that dirty pages in the cold LRU queue will not be kept in the buffer for an overly long period.

2.1.3.8 CCF-LRU

The CCF-LRU[23] further refine the idea of LRUWSR by distinguishing between cold-clean and hot-clean pages. Cold pages are distinguished from hot pages using the Second Chance algorithm. This algorithm defined four types of eviction costs: cold-clean, cold-dirty, hot-clean, and hot-dirty, with increasing priority, thus cold-clean pages are first considered for eviction, then cold-dirty, and so on. Although LRUWSR and CCF-LRU don't require parameter tuning, their clean-first strategy is carried out only based on the coarse assumption of R/W cost asymmetry and hot-cold detection using the Second Chance algorithm, which, in turn, only approximates LRU. As a consequence, it is difficult to reason, when a cold-dirty page should be first considered for eviction over a hot-clean page, and vice versa.

2.2 Research Methodology

The main purpose of research is to discover answers to the questions through the applications of scientific procedures. So, the main aim of research is to find out the truth which is hidden and which has not been discovered yet [24]. Out of different types of research methodologies, this dissertation is based on analytical research in which the simulation approach of quantitative research strategy is used. So, after the problem is formulated based on the collected input data, output information is analyzed and finally the information is generalized. Hence, the main exploration of this dissertation also flows in the same way and focused on the quantitative evaluation of prominent three buffer replacement algorithms for flash based systems.

Chapter – Three

Buffer Replacement Algorithms for Flash Memory based systems in this Dissertation

3.1 LRU Buffer Replacement

Using the recent past as an approximation of the near future, then the page that has not been used for the longest period of time is replaced. This approach is the least-recently-used (LRU) algorithm.LRU[8] replacement associates with each page the time of that page's last use.When a page must be replaced, LRU chooses that page that has not been used for the longest period of time. This strategy is the optimal page-replacement algorithm looking backward in time, rather than forward.

3.1.1 LRU Buffer Replacement Algorithm Implementation Issues

Two implementations are feasible for LRU:

Counters:-

In the simplest case, in each page-table entry a time-of-use field is associated, and a logical clock or counteris added to the CPU. The clock is incremented for every memory reference. Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page. In this way, the "time" of the last reference to each page is noticed. The page with the smallest time value is replaced. This scheme requires a search of the page table to find the LRU page, and a write to memory for each memory access. The times must also be maintained when page tables are changed. Overflow of the clock must be considered.

Stack:-

Another approach to implementing LRU replacement is to keep a stack of page numbers. Whenever a page is referenced, it is removed from the stack and put on the top. In this way, the top of the stack is always the most recently used page and the bottom is the LRU page . Because entries must be removed from the middle of the stack, **it is best implemented by a doubly linked list[8]** with a head and tail pointer. Removing a page and putting it on the top of the stack then requires changing six pointers at worst. Each update is a little more expensive, but there is

no search for a replacement; the tail pointer points to the bottom of the stack, which is the LRU page.

3.1.2 LRU Buffer Replacement Algorithm Tracing

Input Reference String:-

0,3

1,1

0,4

0,2

1,5

0,2

1,1

0,9

1,3

0,6

1,12

0,11

0,10 **Where 0 → page fetched to read i.e. clean page**

1 → page fetched to write i.e. dirty page

memory size = 8

V_Memsize=100

Step 1:- page fetchd= 0,3

MRU **LRU**

3							
0							

Step 2:- page fetchd= 1,1

MRU **LRU**

1	3						
1	0						

Step 3:- page fetchd= 0,4

MRU **LRU**

4	1	3					
0	1	0					

Step 4:- page fetchd= 0,2

MRU **LRU**

2	4	1	3				
0	0	1	0				

Step 5:- page fetchd= 1,5

MRU **LRU**

5	2	4	1	3			
1	0	0	1	0			

Step 6:- page fetchd= 0,2

Since the page 0,2 is already in the cache. So cache hit occurs. And now the page referenced i.e.0,2 becomes the MRU page.

MRU **LRU**

2	5	4	1	3			
0	1	0	1	0			

Step 7:- page fetchd= 1,1

Since the page 1,1 is already in the cache. So cache hit occurs. And now the page referenced becomes the MRU page.

MRU **LRU**

1	2	5	4	3			
1	0	1	0	0			

Step 8:- page fetchd= 0,9

MRU **LRU**

9	1	2	5	4	3		
0	1	0	1	0	0		

Step 9:- page fetchd= 1,3

MRU **LRU**

3	9	1	2	5	4	3	
1	0	1	0	1	0	0	

Step 10:- page fetchd= 0,6

MRU **LRU**

6	3	9	1	2	5	4	3
0	1	0	1	0	1	0	0

Step 10:- page fetchd= 1,12

Since the buffer is full and the new page reference is fetched which is not in the cache.Hence a cache miss occurs.

Now the page replacement occurs. According to LRU page replacement policy Least Recently Used page from the list is the victim.

Therefore, the victim page is 0,3.

MRU **LRU** **victim page**

6	3	9	1	2	5	4	3
0	1	0	1	0	1	0	0

Hence after 0,3 page is replaced and the fetched page 1,12 is maintained at MRU position of the list.

MRU **LRU**

12	6	3	9	1	2	5	4
1	0	1	0	1	0	1	0

replaced page

Step:-11 page fetched= 0,11

Victim page = LRU page= 0,4

MRU **LRU** **victim page**

12	6	3	9	1	2	5	4
1	0	1	0	1	0	1	0

Hence after 0,4 page is replaced and the fetched page 0,11 is maintained at MRU position of the list. **MRU** **LRU**

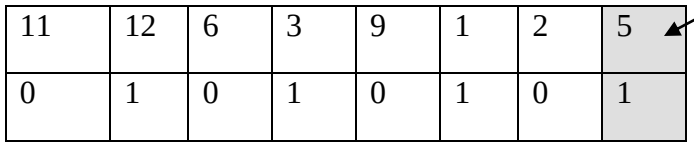
11	12	6	3	9	1	2	5
0	1	0	1	0	1	0	1

replaced page

Step:-11 page fetched= 0,10

Now, According to LRU policy victim page=1,5

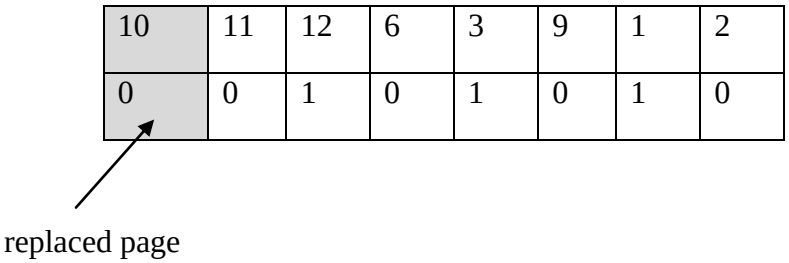
MRU **LRU** **victim page**



11	12	6	3	9	1	2	5
0	1	0	1	0	1	0	1

Hence after 1,5 page is replaced and the fetched page 0,10 is maintained at MRU position of the list.

MRU **LRU**



10	11	12	6	3	9	1	2
0	0	1	0	1	0	1	0

replaced page

Hence the order of victim pages is 0,2 | 1,1 and so on according to the LRU algorithm if in each buffer miss occurs since LRU policy have no concern about cleanliness of the pages.

Stack Implementation

input reference c a d b e b a b c d

lru page

stack

c	a	d	b	e	b	a	b	c	d
	c	a	d	b	e	b	a	b	c
		c	a	d	d	e	e	a	b
			c	a	a	d	d	e	a

page to be replaced (lru) c d e

3.2 CFLRU Buffer Replacement

Since 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. For this purpose CFLRU[18] (Clean-First LRU), which is modified from the LRU algorithm, can be the best solution. CFLRU divides the LRU list into two regions to find a minimal cost point.

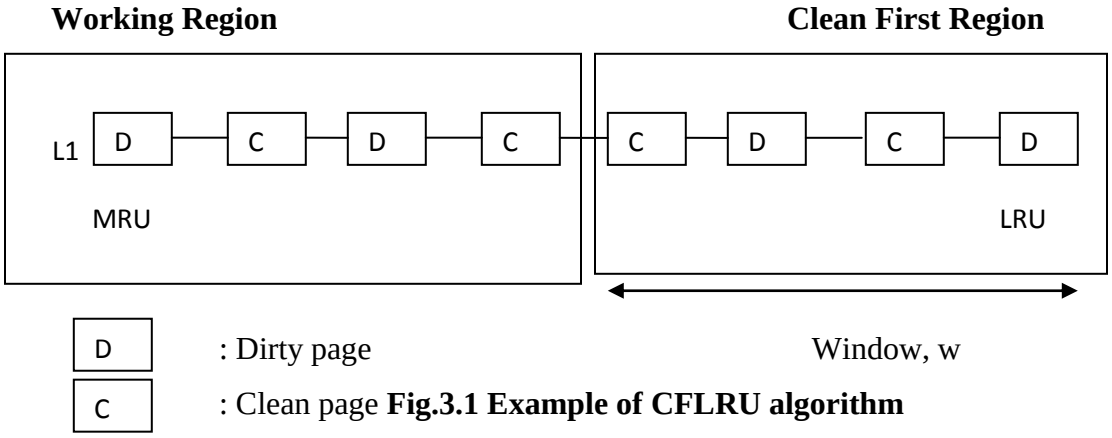


Fig.3.1 Example of CFLRU algorithm

3.2.1 CFLRU Buffer Replacement Algorithm

Algorithm:- 1

Input :- LRU queue L and different workloads

Result (Outputs) :- 1. reference to the victim page p

2. number of page faults

3. number of write counts

1. BEGIN
2. Set the Buffer Size and window size (w) such that

    2.1 w contains the LRU pages and

    2.2 remining part contains the MRU pages
3. Fetch the page along with its mode (R or W)
4. If buffer is empty and fetched page is not found in the buffer then page fault occurs

    4.1 Increment page fault count

    4.2 Insert fetched page at the MRU position of queue

5. Else // if buffer is full

5.1 If the fetched page is found in the buffer then page hit occurs

Adjust the queue by inserting the fetched page at MRU position by shifting other pages in the queue towads LRU position

5.2 Else // fetched page not in the buffer and buffer is full

5.2.1 if LRU clean page is found in w

5.2.1.1 Victim page= LRU clean page

5.2.1.2 Return the referece to the victim page

5.2.1.3 Insert fetched page at MRU position by shifting other pages towards LRU position

5.2.2 else // if LRU clean page is not found

5.2.2.1 Victim page = LRU Dirty page from w

5.2.2.2 Increment the write counts

5.2.2.3 Return the referece to the victim page

5.2.2.4 Insert fetched page at MRU position by shifting other pages towards LRU position

6. END

3.2.2 CFLRU Buffer Replacement Algorithm Tracing

Input Reference String:-

0,3

1,1

0,4

0,2

1,5

0,2

1,1

0,9

1,3

0,6

1,12

0,11

0,10 **Where 0 → page fetched to read i.e. clean page**

1 → page fetched to write i.e. dirty page

memory size = 8

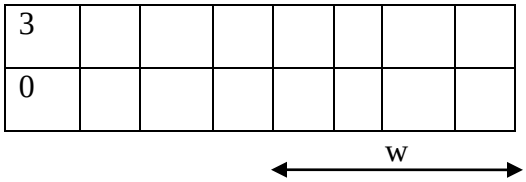
V_Memsize=100

window size (w) = 0.5

Hence actual window_size = 0.5*8 =4

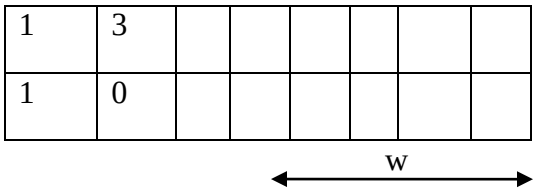
Step 1:- page fetchd= 0,3

MRU **LRU**



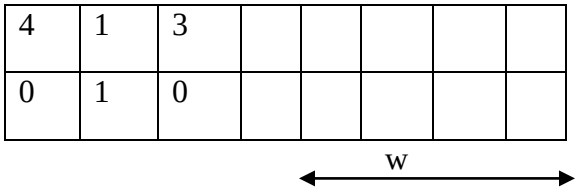
Step 2:- page fetchd= 1,1

MRU **LRU**



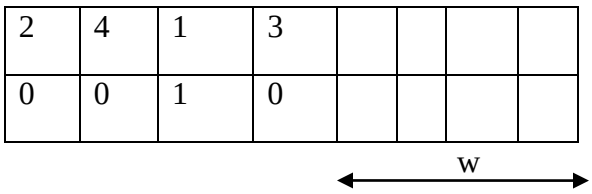
Step 3:- page fetchd= 0,4

MRU **LRU**



Step 4:- page fetchd= 0,2

MRU **LRU**



Step 5:- page fetchd= 1,5

MRU **LRU**

5	2	4	1	3			
1	0	0	1	0			

w

Step 6:- page fetchd= 0,2

Since the page 0,2 is already in the cache. So cache hit occurs. And now the page referenced i.e.0,2 becomes the MRU page.

MRU **LRU**

2	5	4	1	3			
0	1	0	1	0			

w

Step 7:- page fetchd= 1,1

Since the page 1,1 is already in the cache. So cache hit occurs. And now the page referenced becomes the MRU page.

MRU **LRU**

1	2	5	4	3			
1	0	1	0	0			

w

Step 8:- page fetchd= 0,9

MRU **LRU**

9	1	2	5	4	3		
0	1	0	1	0	0		

w

Step 9:- page fetchd= 1,3

MRU **LRU**

3	9	1	2	5	4	3	
1	0	1	0	1	0	0	

w

Step 10:- page fetchd= 0,6

MRU **LRU**

6	3	9	1	2	5	4	3
0	1	0	1	0	1	0	0

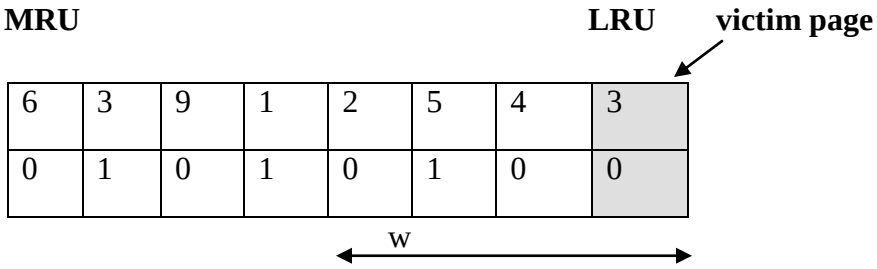
w

Step 11:- page fetchd= 1,12

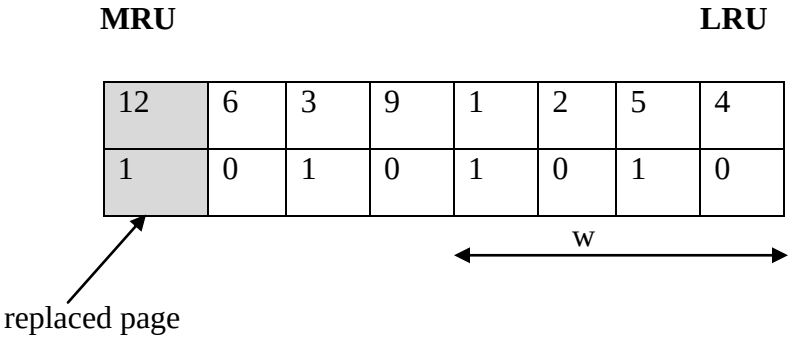
Since the buffer is full and the new page reference is fetched which is not in the cache.Hence a cache miss occurs.

Now the page replacement occurs. According to CFLRU page replacement policy Least Recently Clean page from the window (w) is the victim.

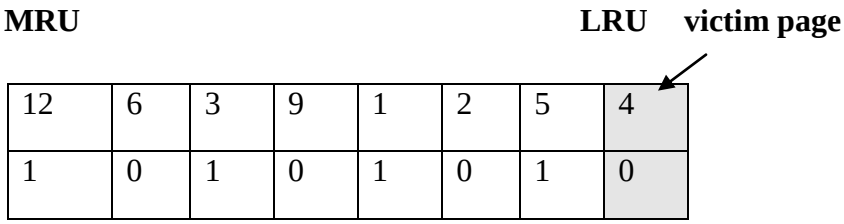
Therefore, the victim page is 0,3.



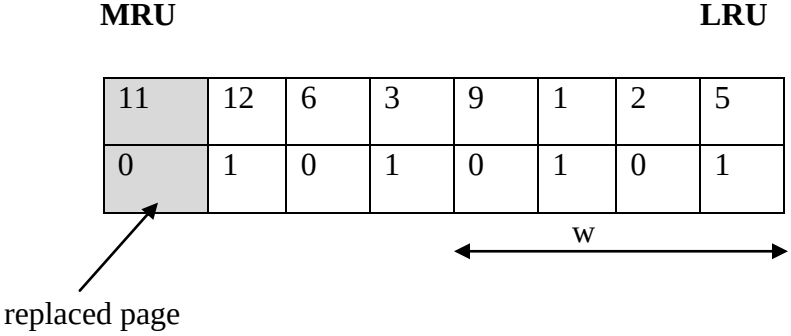
Hence after 0,3 page is evicted and the fetched page 1,12 is maintained at MRU position of the list.



Step:-12 page fetched= 0,11

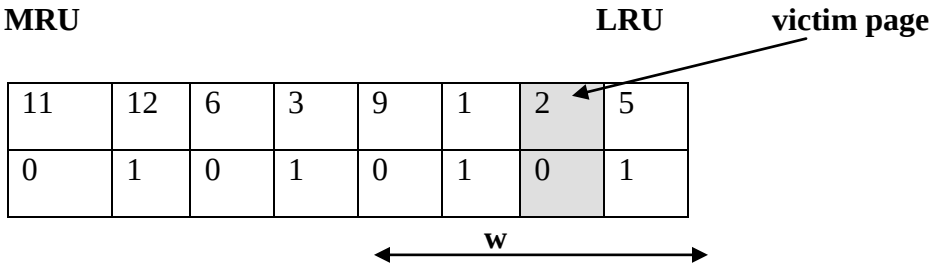


Hence after 0,4 page is evicted and the fetched page 0,11 is maintained at MRU position of the list.

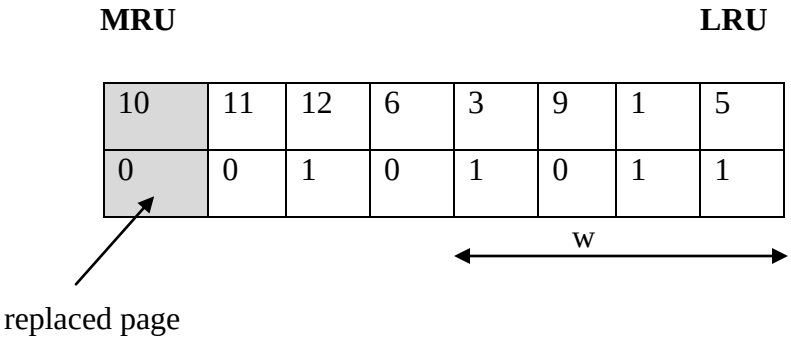


Step:-13 page fetched= 0,10

Now, According to CFLRU policy victim page=0,2



Hence after 0,2 page is evicted and the fetched page 0,10 is maintained at MRU position of the list.



Hence the order of victim pages is 0,4 | 0,2 | 0,9 and so on according to the CFLRU algorithm if in each case miss occurs since CFLRU policy considers recency as well as cleanliness of the referenced pages but have no concern of the frequency of the pages.

3.3 AD-LRU Buffer Replacement

For buffer replacent algorithms of flash-based systems, not only the hit ratio but also the write counts are considered. However the traditional algorithm like LRU not always evict the clean pages and the flash aware algorithms like CFLRU evict the hot clean pages from the buffer, because it uses clean-first strategy and donot take the page reference frequency into account and hence the hit ratios may degrade.

So, ADLRU algorithm enhanced the traditional LRU policy by frequency considerations and first evict least-recently and least-frequently used clean pages to reduce the write counts during the page replacement.

AD-LRU is also one of the efficient buffer replacement policy developed for flash based systems, which also focuses on reducing the number of write/erase operations as well as maintain high buffer hit ratio. This algorithm not only considers frequency but also the recency of page references and cleanliness of pages.

The AD-LRU algorithm is summarized as follows:

(1) Two LRU queues are used to capture both the recency and frequency of page references, among which one cold LRU queue stores the pages referenced only once and the hot LRU queue maintains the pages referenced at least twice.

(2) The sizes of the double LRU queues are dynamically adjusted according to the changes in the reference patterns. The size of the hot LRU queue is increased and the size of the cold one when a page in the cold queue is re-referenced is decreased. The hot queue shrinks when a page is selected as victim and moved from there to the cold queue.

(3) During the eviction procedure, at first least-recently-used clean page from the cold LRU queue as the victim is selected, for which a specific pointer FC is used. If clean pages do not exist in the cold LRU queue, second-chance policy[5] can be used to select a dirty page as the victim. For this reason, each page in the double LRU queues is marked by a referenced bit, which is always set to 1, when the page is referenced. Hence, the second-chance policy ensures that dirty pages in the cold LRU queue will not be kept in the buffer for an overly long period.

3.3.1 AD-LRU Buffer Replacement Algorithm

Algorithm 1:- AD-LRU_fetch

data :- L_c → the cold queue containing clean (c) pages and L_h → the hot queue containing hot(hot) pages, intially c=0, h=0

result :- return a reference to the requested page (p)

1. Begin
2. If p is in L_h then // p is in the hot queue

Move p to the MRU position in L_h

Adjust FC in L_h to let FC point to LRU clean page in L_h

Ref(p)=1

Return a reference to p in L_h

2.1 Else if p is in L_c then // p is in the cold queue

h++; // increase the size of hot queue

c--; // decrease the size of cold queue

Move p to the MRU position in L_h

Adjust FC in L_c to let FC point to LRU clean page in L_c

Ref(p)=1

Adjust FC in L_h to let FC point to LRU clean page in L_h

Return a reference to p in L_h

3. Else // p is not in the buffer

3.1 If

there is free space in the buffer then

c++;

put p in L_c

adjust L_c by putting p into the MRU position

adjust FC in L_c to let FC point to the LRU clean page in L_c

ref(p)=1

return a reference to p in L_c

3.2 else

victim = SelectVictim(L_c);

else // cpld queue is too small so replace L_h

victim = SelectVictim(L_h);

h - - ; // adjust the cold and hot queues

c + +;

3.3 else if victim is dirty then

WriteDirty(p); // write to flash

4. Put p into a free frame In L_c
5. Adjust L_c by putting p into MRU position
6. Ref(p)=1
7. Return a reference to p in L_c
8. End

Algorithm 2:- Select_Victim

data :- LRU queue L

result :- return a reference to the victim page

1. Begin
2. If FC of L is not NULL then // select the first clean page
   Remove the FC page from L
   Adjust FC position in L
   return a reference to the FC page
   // select a dirty page using the second chance policy starting from the LRU position
3. victim = L.first;
4. while ref(victim) =1 do
   move victim to the MRU position in L
   ref(victim) = 0;
   victim= L.first;
5. remove victim from L
6. return a reference to the victim
7. End

3.3.2 AD-LRU Buffer Replacement Algorithm Tracing

Input Reference String:-

0,3

1,1

0,4

0,2

1,5

0,2

1,1

0,9

1,3

0,6

1,12

0,11

0,10 **Where 0 → page fetched to read i.e. clean page**

1 → page fetched to write i.e. dirty page

memory size = 8

V_Memsize=100

Step 1:- page fetched= 0,3

MRU **LRU**

Pg no	3							
mode	0							
frequency	1							

Cold region

Step 2:- page fetched= 1,1

MRU **LRU**

Pg no	1	3							
mode	1	0							
frequency	1	1							

Cold region

Step 3:- page fetched= 0,4

MRU **LRU**

Pg no	4	1	3					
mode	0	1	0					
frequency	1	1	1					

Cold region

Step 4:- page fetched= 0,2

MRU **LRU**

Pg no	2	4	1	3				
mode	0	0	1	0				
frequency	1	1	1	1				

Cold region

Step 5:- page fetched= 1,5

MRU **LRU**

Pg no	5	2	4	1	3			
mode	1	0	0	1	0			
frequency	1	1	1	1	1			

Cold region

Step 6:- page fetched= 0,2 which is re-referenced

So it is moved to MRU position of hot LRU queue and hence c- - and h + +

MRU **LRU**

Pg no	5	4	1	3	2			
mode	1	0	1	0	0			
frequency	1	1	1	1	2			

Cold region Hot region

Step 7:- page fetchd= 1,1 which is re-referenced

So it is moved to MRU position of hot LRU queue and hence c- - and h + +

MRU **LRU**

Pg no	5	4	3	1	2			
mode	1	0	0	1	0			
frequency	1	1	1	2	2			

Cold region Hot region

Step 8:- page fetchd= 0,9

MRU **LRU**

Pg no	9	5	4	3	1	2		
Mode	0	1	0	0	1	0		
frequency	1	1	1	1	2	2		

Cold region Hot region

Step 9:- page fetchd= 1,3

MRU **LRU**

Pg no	3	9	5	4	3	1	2	
Mode	1	0	1	0	0	1	0	
frequency	1	1	1	1	1	2	2	

Cold region Hot region

Step 10:- page fetchd= 0,6

MRU **LRU**

Pg no	6	3	9	5	4	3	1	2
mode	0	1	0	1	0	0	1	0
frequency	1	1	1	1	1	1	2	2

Cold region Hot region

Step 11:- page fetchd= 1,12

Buffer is full hence Victimpage= LRU Cold Clean page = 0,3

MRU **victim page** **LRU**

Pg no	6	3	9	5	4	3	1	2
mode	0	1	0	1	0	0	1	0
frequency	1	1	1	1	1	1	2	2

Cold region Hot region

After 0,3 page is evicted new page 1,12 is maintainted at MRU position of Cold region as follows.

MRU **LRU**

Pg no	12	6	3	9	5	4	1	2
Mode	1	0	1	0	1	0	1	0
Frequency	1	1	1	1	1	1	2	2

Cold region Hot region

Step 12:- page fetched= 0,11

Buffer is full hence Victimpage= LRU Cold Clean page = 0,4

MRU **victim page** **LRU**

Pg no	12	6	3	9	5	4	1	2
mode	1	0	1	0	1	0	1	0
frequency	1	1	1	1	1	1	2	2

Cold region Hot region

After 0,4 page is evicted new page 0,11 is maintainted at MRU position of Cold region as follows.

MRU **LRU**

Pg no	11	12	6	3	9	5	1	2
mode	0	1	0	1	0	1	1	0
frequency	1	1	1	1	1	1	2	2

Cold region Hot region

Step 13:- page fetched= 0,10

Buffer is full hence Victimpage= LRU Cold Clean page = 0,9

MRU **victimpage** **LRU**

Pg no	11	12	6	3	9	5	1	2
mode	0	1	0	1	0	1	1	0
frequency	1	1	1	1	1	1	2	2

Cold region Hot region

After 0,9 page is evicted new page 0,10 is maintainted at MRU position of Cold region as follows.

MRU **LRU**

Pg no	10	11	12	6	3	5	1	2
Mode	0	0	1	0	1	1	1	0
Frequency	1	1	1	1	1	1	2	2

Cold region Hot region

Since 0,2 page is referenced earlier than the page 0,9 but the frequency count of 0,9 is less than of page 0,2 hence according to the ADLRU policy 0,9 is evicted first for page replacement which clearifies that ADLRU considers recency, frequency and cleanliness of the referenced pages.

3.4 Determination of optimal window size (w) for CFLRU

Finding the optimal window size of the clean-first region is important to minimize the replacement cost. Since a large window size may increase cache miss rate and a small window size may increase the number of evictedd dirty pages[18,21] i.e. the number of write counts. Therefore, the window size of the clean-first region needs to be decided properly in order to minimize the overall replacement cost.

Hence, in this dissertation work, the optimal window size is determined with help of experimental/ simulation study since there is no exact formula or theory for finding the optimal owindow size.

For this the values of w are varied from 0 to 0.9 and tested for minimal number of write counts with high hit rates, taking different workloads i.e. input traces.

The impact of size of w in the write counts and hit rate for different workloads with different patterns for CFLRU are studied by changing the value of w from 0 to 0.9.

Chapter – Four

Implementation and Testing

4.1 Tools Used

4.1.1 Programming Language

Java programming language is used for the implementation of the algorithms in this dissertaion. Java is a high level programming language developed by Sun Microsystems which was originally called OAK. Java is a object oriented programming language similar to C++ but simplified to eliminate language features that cause common programming errors. Java source code files are compiled into a format called bytecodes which can then be executed by a Java interpreter. Compiled Java code can run on most computers because Java interpreters and runtime environments, known as Java Virtual Machines(JVM) exist for most Operating System. Bytecode can also be converted directly into machine language instructions by a Just in Time(JIT).

Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such asNetscape Navigator or Microsoft Internet Explorer

Java language is popular programming language due to some of the following main characterstics:-

1. It is simple, object-oriented and familiar.
2. It is robust and secure.
3. It is architecture-neutral and portable.
4. It executes with high performance.
5. It is interpreted, threaded, and dynamic.

4.1.2 NetBeans IDE

NetBeans is an open source integrated development environment (IDE) for developing primarily with Java, but also with other languages. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. The NetBeans project consists of a full-featured open source IDE written in the Java programming language and a rich client application platform, which can be used as a generic framework to build any kind of applications.

4.2 Data Structures Used

4.2.1 Doubly Linked List (DLL)

A linked list in which all nodes are linked together by multiple number of links i.e. each node contains three fields (two pointer fields and one data field i.e. prev, next & info respectively) rather than two fields is called Doubly Linked List[8] (DLL). It provides bidirectional traversal.

To implement the three algorithms in this dissertation, doubly linked list is used since it provides bidirectional traversal of nodes so that it is easy to adjust the page references during replacement phenomenon.

The two node links allow traversal of the list in either direction. While adding or removing a node in a doubly-linked list requires changing more links than the same operations on a singly linked list, the operations are simpler and potentially more efficient (for nodes other than first nodes) because there is no need to keep track of the previous node during traversal or no need to traverse the list to find the previous node, so that its link can be modified.

The first and last nodes of a doubly-linked list are immediately accessible (i.e., accessible without traversal, and usually called head and tail) and therefore allow traversal of the list from the beginning or end of the list, respectively: e.g., traversing the list from beginning to end, or from end to beginning, in a search of the list for a node with specific data value. Any node of a

doubly-linked list, once obtained, can be used to begin a new traversal of the list, in either direction (towards beginning or end), from the given node.

But some tradeoffs are there. Doubly linked list occupy more space and often more operations are required for the similar tasks as compared to singly linked lists.

4.2.2 Structure of LRU, CFLRU and ADLRU nodes:-

Structure of LRU node

```
package bufferreplacement_simulation;

public class LRU
{
   private PageNode head;
   private int npf;
   private int np;
   private PageNode[] pt;
   private int mem_size;
   private int VM_SIZE;
   private int free_mem_size;
   LRU()
   {
      head=null;
      npf=np=0;
      free_mem_size=mem_size;
      pt=new PageNode[VM_SIZE+1];
      for(int i=1;i<=VM_SIZE;i++)
```

```
        {
            pt[i]=new PageNode();
            pt[i].pn=i;
        }
        trace();
    }
```

Structure of CFLRU node

```
package bufferreplacement_simulation;
class PageNode
{
    int pn;
    boolean isclean;
    boolean isresident;
    PageNode next;
    PageNode prev;
    public PageNode()
    {
        isresident=false;
        next=prev=null;
    }
}
class CFLRU
{
```

```
private PageNode head=new PageNode();
private int npf;
private int np;
private int wc;
private PageNode[] pt;
private int mem_size;
private int VM_SIZE;
private int free_mem_size;
private double w;
CFLRU()
{
   head=null;
   npf=np=0;
   wc=0;
   mem_size=1024;
   VM_SIZE=50000;
   w=(0.5*mem_size);
   free_mem_size=mem_size;
   pt=new PageNode[VM_SIZE+1];
   for(int i=1;i<=VM_SIZE;i++)
   {
      pt[i]=new PageNode();
       pt[i].pn=i;
```

```
        }
    }
```

Structure of ADLRU node

```
package bufferreplacement_simulation;
class PageNode
{
    int pn;
    boolean isclean;
    boolean iscold;
    boolean isresident;
    PageNode next;
    PageNode prev;
    int frequency;
    public PageNode()
    {
        frequency=0;
        iscold=(this.frequency<2);
        isresident=false;
        next=prev=null;
    }
}
class ADLRU
{
```

```
private PageNode head;
private int npf;
private int np;
private int wc;
private PageNode[] pt;
private int mem_size;
private int VM_SIZE;
private int free_mem_size;
ADLRU()
{
   head=null;
   npf=np=0;
   wc=0;
   mem_size=32;
   VM_SIZE=50000;
   free_mem_size=mem_size;
   pt=new PageNode[VM_SIZE+1];
   for(int i=1;i<=VM_SIZE;i++)
   {
      pt[i]=new PageNode();
      pt[i].pn=i;
   }
}
```

4.3 Flowcharts of the Algorithms taken:-

4.3.1 LRU

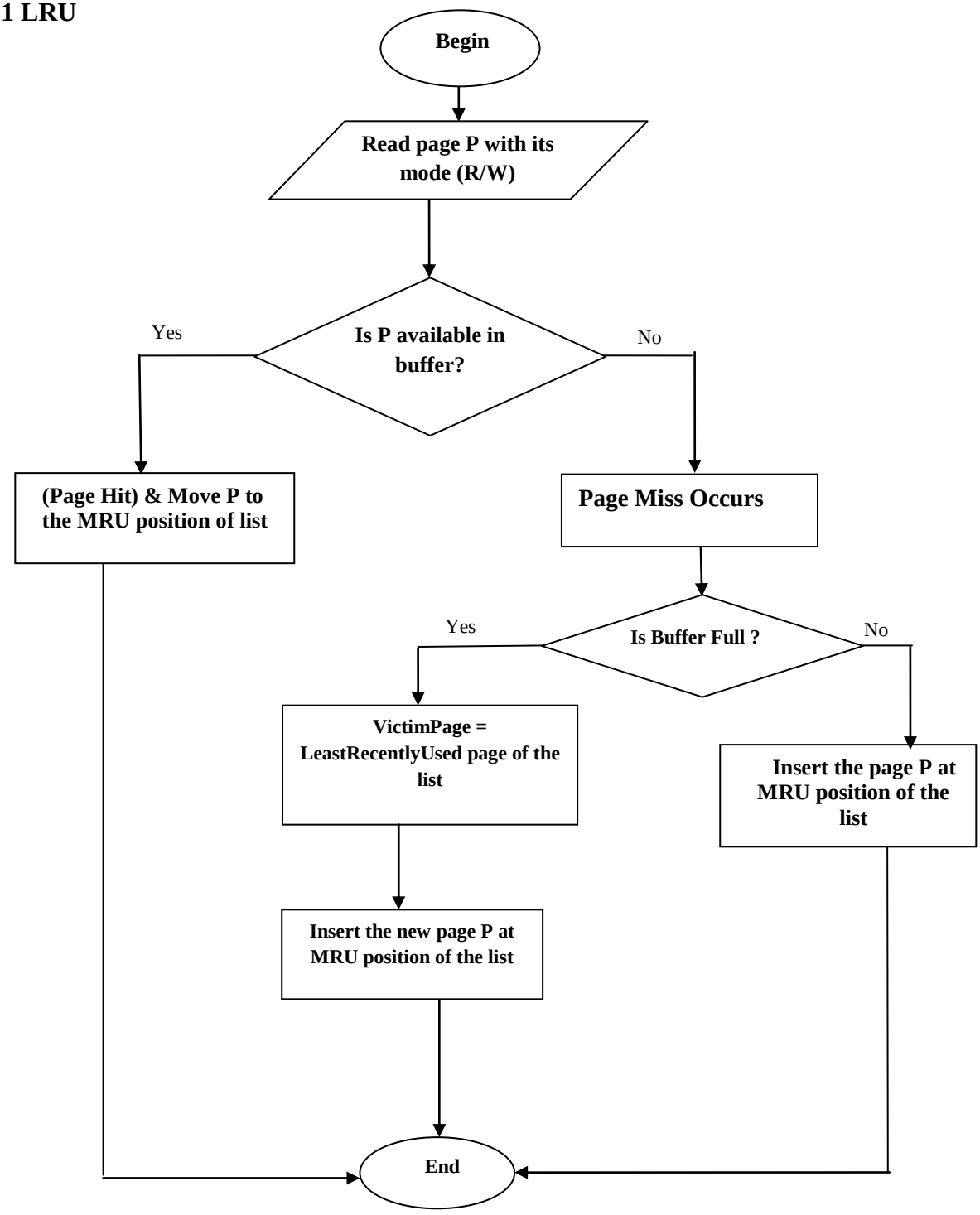


Fig.4.1 Flowchart of LRU Algorithm

4.3.2 CFLRU

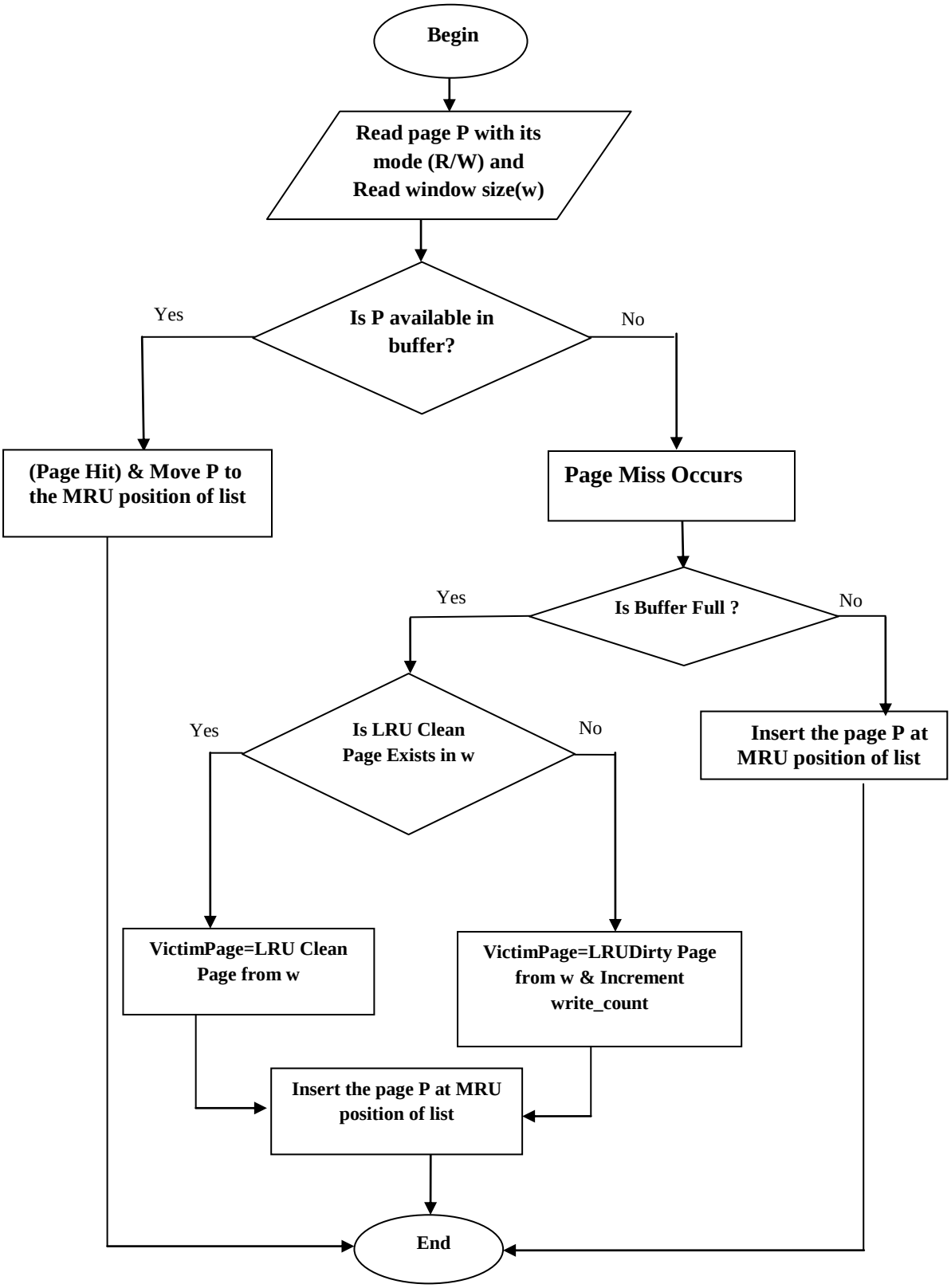


Fig.4.2 Flowchart of CFLRU Algorithm

4.3.3 ADLRU

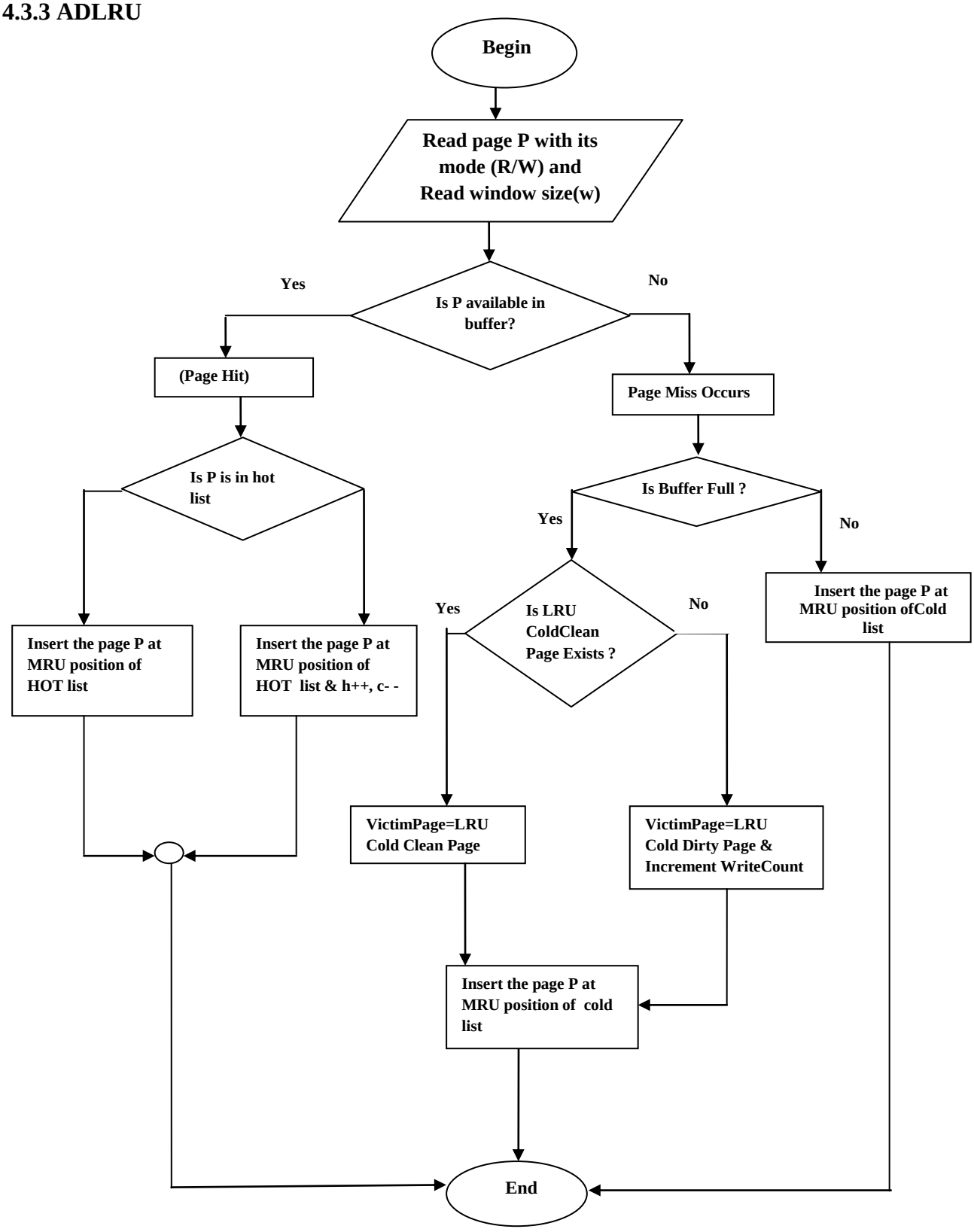


Fig.4.3 Flowchart of ADLRU Algorithm

4.4 Sample Test Cases

4.4.1 Random Data References

Input data format = (r/w mode, page_reference) , where r i.e. 0 → read and w i.e. 1 → write

1,8575 0,17754 0,33289 0,3838 0,19942 1,25113 1,35145 1,1939 0,40780
0,12831 0,31724 1,37162 1,861 1,35912 0,39216 1,10863 0,15454 0,32425
0,42141 1,34769 0,29923 1,3050 0,4043 0,39113 1,11686 1,25837 0,4941
0,7882 0,39262 1,32631 0,36490 1,11934 1,8851 1,16962 0,37665 1,23980
0,41727 0,15074 1,19029 1,1750 0,49554 1,18797 1,6747 0,31276 1,786
1,42798 0,30971 0,42594 1,49503 0,23075 1,8717 1,13521 1,988 0,22467
1,12586 1,45284 1,39329 0,45058 0,14795 0,21120 0,7786 1,43211 0,47655
0,42213 0,919 0,603 0,4844 0,44923 1,29324 0,26292 1,31526 1,38097
1,39819 0,30117 1,14208 1,27844 1,8361 1,16455 1,5699 0,10670 1,1066
0,9039 1,6477 1,41170 0,23504 1,32354 0,14280 1,36795 1,8732 1,46002
1,4880 1,5637 1,21680 1,3496 1,3220 0,13282 1,42670 0,11669 0,2716
1,49749 0,12437 0,42550 0,27038 0,26790 1,44095 1,25674 0,4498 1,32206
0,33123 0,9846 0,46190 0,20089 0,14060 1,28875 1,16434 1,12575 1,47687
1,41433 0,16610 0,3411 1,23633 1,17429 0,49681 1,25625 1,34155 0,33804
0,21089 0,16647 1,3104 0,3843 1,7142 0,30193 0,12695 1,28453 0,9115
0,25532 0,47722 1,47868 1,49752 0,6476 1,41825 1,7631 0,14127 0,29127
1,12805 0,48855 1,33911 0,41079 1,25483 1,39430 0,1037 0,3297 0,16599
1,36036 0,15578 0,10091 1,25578 0,23037 0,24073 1,16386 0,15490 1,1048
0,19682 0,8798 0,26493 0,48889 1,7791 1,35987 1,16638 1,45825 1,38057
0,30566 0,48228 0,38949 1,47502 0,26137 1,22920 1,32430 1,7944 1,35589
1,40867 1,47773 0,46838 1,44616 1,39286 1,39175 0,42242 1,20480
1,22293 0,20389 0,23900 0,18555 0,46427 0,8516 1,49886 0,10679 0,9400
1,24467 0,3709 0,411 0,25540 0,22153 1,3954 0,23179 0,9759 1,33020
0,2711 1,42697 1,34063 1,22716 1,23599 0,25436 1,22036 0,34470 1,12097
0,16505 0,30238 0,37133 1,47578 1,43832 1,12285 1,43630 1,25872
1,22922 0,47801 0,33166 0,30809 0,22288 1,15530 1,18379 1,26444

0,15686 1,47213 0,23218 0,17078 0,9358 1,22390 1,12973 1,12756 0,14487
1,14736 0,17512 0,16192 1,20303 0,13516 0,7694 0,46578 1,14630 0,44937
0,8268 1,27634 1,42340 1,5782 0,18033 0,36288 1,20348 1,25705 0,13909
0,48059 1,39900 1,42855 1,22621 0,19304 1,34024 1,12876 1,24241
1,44899 0,44903 0,47548 0,2683 1,15201 1,40903 0,49098 0,11108 1,8735
1,23818 0,9948 0,2917 0,17513 1,26798 0,37204 1,48172 0,41628 0,21196
0,16265 1,33034 0,49456 1,20361 0,46366 0,49157 0,22078 1,21714
0,36970 1,8646 1,9469 0,36225 1,46020 1,1702 1,2979 1,21191 1,46458
0,48207 1,29311 1,33363 0,18666 1,9636 1,25704 1,43329 1,32357 0,34652
1,29324 0,12938 1,36894 0,43868 1,47460 1,47637 0,24355 1,12176
0,12286 1,7957 0,41794 0,49251 1,29601 0,45568 0,43907 1,10770 0,27169
1,45435 0,9439 1,46069 0,14691 1,2992 0,20988 1,23961 0,42924 0,16206
1,17346 1,17766 0,19820 1,17517 0,1382 0,41472 1,21713 0,47315 1,3598
0,22383 0,54 0,28589 0,39334 1,5327 1,31446 0,11905 0,2578 0,42675
1,36464 0,10865 1,38028 1,10859 0,21603 0,31835 0,47015 0,11141
1,37753 0,41532 0,5012 1,42335 1,19108 1,6948 0,3639 0,37524 0,14929
1,30811 0,39972 1,17926 1,20116 1,24335 0,49921 0,42809 1,20340 1,3331
0,46753 1,20246 1,2500 1,30976 1,18790 1,35884 1,8436 1,8547 0,40654
0,37234 1,4720 0,14709 0,49005 1,120 1,49345 1,21382 0,5659 1,8975
1,35557 0,10747 0,22871 0,40127 1,8215 0,32476 1,5622 0,11686 1,7083
0,47138 0,8859 1,49704 1,26240 0,36223 1,4757 1,46740 1,43596 1,44175
1,11825 1,10458 0,30016 1,43645 0,28919 1,6579 1,47380 0,4275 0,47727
1,21962 1,32498 0,15530 1,22466 1,45797 0,44519 0,47839 0,34722
0,29879 0,7742 1,35170 0,8487 0,32399 0,48678 0,33246 1,46767 0,889
1,15261 1,45658 0,14995 1,18321 0,7303 1,1460 1,2724 1,14771 1,49585
0,2935 1,3603 0,47776 0,48439 0,36931 1,27169 1,48598 0,10737 1,23017
1,46089 0,32556 0,1153 1,12834 1,8672 1,33281 0,8337 0,16815 0,14078
0,23123 1,38627 1,3974 1,39029 0,24143 0,45127 1,6153 0,21868 1,3032
0,49348 0,27357 0,2787 1,41676 0,45740 1,12125 1,35064 1,6677 0,36840
0,23619 0,28071 0,4706 0,33306 0,45229 0,44371 1,41742 0,29463 1,35470
1,20690 1,21422 0,2804 0,9359 1,23231 0,21345 0,1137 1,49613 0,17119

Chapter – Five

Test Results and Analysis

5.1 Data Collection

Data are raw facts or the sources of information which are used as input to the replacement algorithms, that produces meaningful results after processisng[23]. So, all the data's (i.e. the page references) with (mode, page number) format in this dissertation work are taken from the standard synthetic traces generated by the simulaions. In this study, altogether three types of synthetic traces (i.e. random, readmost, and writemost traces) are used and tested. These three different traces with different nature are used as workloads for the studied buffer replacement algorithms. The output generated by the algorithms are then used to calculate hit rates in percentage and the total number ofwrite counts. Each of the workloads taken in this dissertation contains 1,00,000 page references and the page numbers ranges from 0 to 49,999. The workloads taken are categorized as Workload 1 (random traces), Workload 2 (readmost traces), and Workload 3 (writemost traces). The sample of each of the workloads are given in Appendices A,B, and C respectively. The input is (mode, page number), where mode is either read or write i.e 0 represents read mode and 1 represents write mode of the page and the page number is the value of the page reference or fetched page number.

5.2 Testing

These three workloads are separately tested in the simulator in this dissertation. Each workload is tested in LRU, CFLRU and AD-LRU simulator by varying the cache size from 4 to 1024.In the case of CFLRU one of the paramenter i.e. the size of window(w) is varied from 0.1 to 0.9 and the optimal window size is taken for each workloads. The optimal value of window size is taken on the basis of high hit rate and at the same time the less number of write counts (wc).So for each workload the optimal value of w is taken and then the algorithms are compared for quantitative evaluation.

5.2.1 Test result of Workload 1[random traces]

5.2.1.1 Test result for CFLRU for finding optimal value of window size(w) for Workload 1

Total number of page referenced (np)=100000 total number of distinct pages (tdp) =43247 buffer_size =1024

Window size (w)	CFLRU			
	Npf	mr (%)	hr (%)	write count(wc)
0.1	99396	98.93	1.07	99106
0.25	98680	97.67	2.33	98023
0.4	98026	96.52	3.48	97125
0.5	97947	96.38	3.62	96923
0.6	97123	94.93	5.07	97009
0.75	96056	93.05	6.95	97314
0.9	94135	89.66	10.34	97191

Table 5.1 Test result for CFLRU for finding optimal value of window size for workload 1

Since the hit rate of CFLRU may be affected by w. So from above experiment it is clear that the optimal value of window size is 0.5 since the write counts is least at that value of w . Though the hit rate of CFLRU is maximum at 0.75 but it has greater number of write counts. Hence the parameter w is taken 0.5 which means half of the buffer is used as clean first window.

5.2.1.2 Test result for three algorithms with varying buffer size

Total number of pages referenced =1,00,000 Total number of distinct pages = 43247 Window size= 0.5

Buffer Size	LRU				CFLRU				ADLRU			
	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count
4	99993	99.98	0.02	NO	99986	99.97	0.03	99982	99531	99.17	0.83	99486
8	99989	99.97	0.03		99981	99.96	0.04	99973	99016	98.26	1.74	98953

16	99982	99.96	0.04	PROVISION	99971	99.42	0.58	99955	97921	96.33	3.67	94120
32	99971	99.42	0.58		99939	99.34	0.66	99907	96867	94.47	5.53	93028
64	99947	99.37	0.63		99884	99.29	0.71	99820	96003	92.95	7.05	92371
128	99897	99.31	0.69		99775	99.26	0.74	99647	95246	91.62	8.38	91369
256	99783	99.28	0.72		99531	99.22	0.78	99275	93486	88.52	11.48	89567
512	99602	99.23	0.77		99013	98.26	1.74	98501	92228	86.30	13.70	87430
1024	98107	96.66	3.34		97947	96.38	3.62	96923	87712	78.35	21.65	73591

Table 5.2 Test results of Workload 1 with varying buffer size

5.2.2 Test result of Workload 2[readmost traces]

5.2.2.1 Test result for CFLRU for finding optimal value of window size(w) for Workload 2

np=100000 tdp=43212 buffer_size =1024

Window size (w)	CFLRU			
	Npf	mr (%)	hr (%)	Wc
0.1	97689	5.93	4.07	81921
0.25	91557	85.13	14.87	76508
0.4	84962	73.51	26.49	42472
0.5	84601	72.88	27.12	42721
0.6	83367	70.71	29.29	43059
0.75	85698	74.81	25.19	42904
0.9	85147	74.84	25.16	46595

Table 5.3 Test result for CFLRU for finding optimal value of window size for workload 2

So from above experiment the optimal value of window_size(w) =0.4

5.2.2.2 Test result for three algorithms with varying buffer size

Total number of pages referenced =1,00,000 Total number of distinct pages = 43212 Window size= 0.4

buffer size	LRU				CFLRU				ADLRU			
	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count
4	98185	96.8	3.2	NO PROVISION	98121	96.6	3.4	86305	97677	95.9	4.1	85304
8	98036	96.5	3.5		98000	96.47	3.5	84391	96433	93.72	6.28	82163
16	97769	96.07	3.03		97689	95.93	4.07	81921	94891	91	9	78001
32	96981	94.52	5.48		96314	93.5	6.5	80751	92997	87.66	12.34	71112
64	95289	91.7	8.3		95128	91.42	8.58	78648	91388	84.84	15.16	62586
128	93505	88.56	11.44		92893	87.48	12.52	61826	90624	83.45	16.55	54792
256	91007	84.16	15.84		90657	83.54	16.46	58122	87035	77.17	22.83	46399
512	88679	80.06	19.94		88251	79.31	20.69	54036	79293	63.54	36.46	42106
1024	85147	73.84	26.16		84962	73.51	26.49	42472	71111	49.19	50.81	39248

Table 5.4 Test results of Workload 2 with varying buffer size

5.2.3 Test result of Workload 3[writemost traces]

5.2.3.1 Test result for CFLRU for finding optimal value of window size(w) for Workload 3

Total number of pages referenced =1,00,000 Total number of distinct pages = 43182 buffer_size= 1024

Window size (w)	CFLRU			
	Npf	mr (%)	hr (%)	Wc
0.1	97938	96.37	3.63	97598
0.25	95854	92.7	7.3	95007
0.4	86369	76.01	23.9	91102

0.5	81004	66.57	33.43	89224
0.6	73377	53.42	46.58	86722
0.75	72825	52.17	47.83	88050
0.9	71205	49.32	50.68	87983

Table 5.5Test result for CFLRU for finding optimal value of window size(w) for Workload 3

So from above experiment the optimal value of window_size(w) =0.6

5.2.3.2Test result for three algorithms with varying buffer size

Total number of pages referenced =1,00,000 Total number of distinct pages = 43182 Window size= 0.6

buffer size	LRU				CFLRU				ADLRU			
	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count	Page faults	Miss rate(%)	Hit rate(%)	Write count
4	99586	99.79	0.21	NO PROVISION	97993	96.98	3.02	97868	97020	95.26	4.74	96997
8	98122	97.2	0.8		97089	95.38	4.62	96771	96988	95.2	4.8	96123
16	97315	95.78	4.22		96854	94.96	5.04	96844	95024	91.73	8.27	94556
32	95020	91.72	8.28		93125	88.37	11.63	95026	91165	84.89	15.11	91085
64	93169	88.44	11.56		91625	85.71	14.29	94119	87050	77.62	22.38	93838
128	91377	85.27	14.73		87000	77.53	22.47	94006	81819	68.36	31.64	91641
256	89884	82.63	17.37		84555	73.2	26.8	91125	80056	65.24	34.76	90037
512	85090	74.14	25.86		81092	67.07	32.93	88912	72521	51.91	48.09	87881
1024	78141	61.85	32.15		73377	53.42	46.58	86722	63559	36.05	63.95	85581

Table 5.6 Test results of Workload 3 with varying buffer size

5.3 Anlysis

All the collected data's are then analyzed by drawing different graphs. Hit rates in percentage and the total number of write counts of algorithms are used as criteria for analyzing their performances.

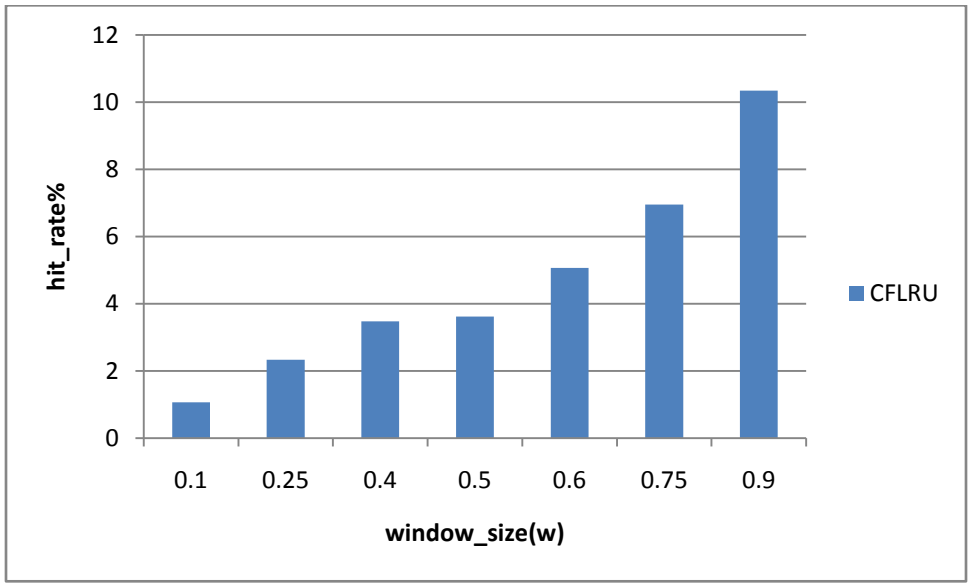


Figure 5.1 Graph for Table 5.1 showing hit rate for CFLRU with varying window_size

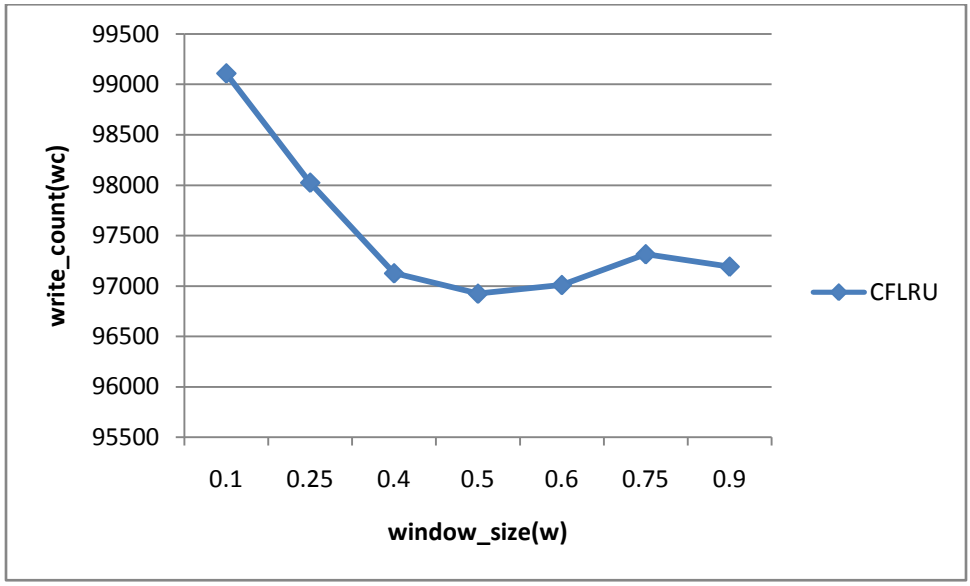


Figure 5.2 Graph for Table 5.1 showing write_counts for CFLRU with varying window_size

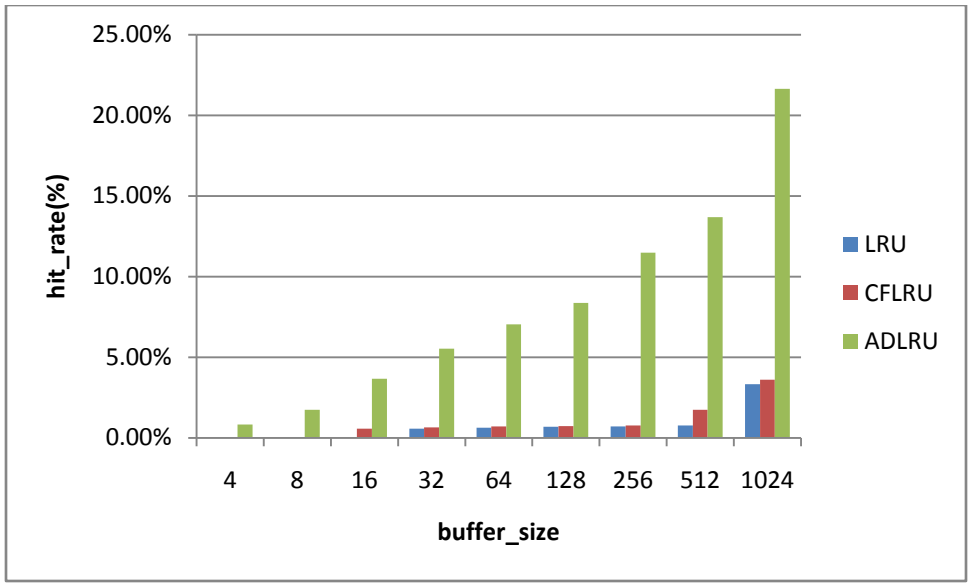


Figure 5.3 Graph for Table 5.2 showing hit rate

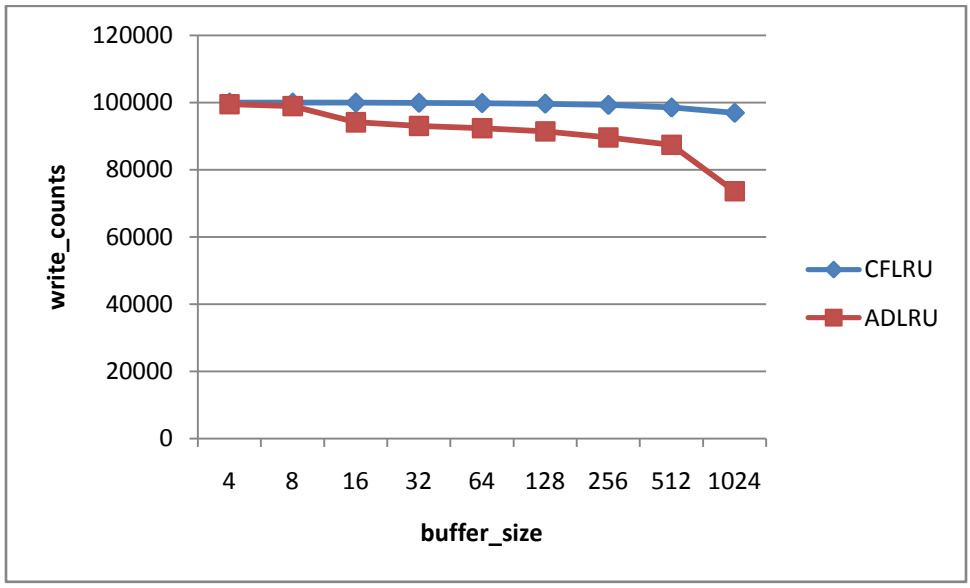


Figure 5.4 Graph for Table 5.2 showing write counts

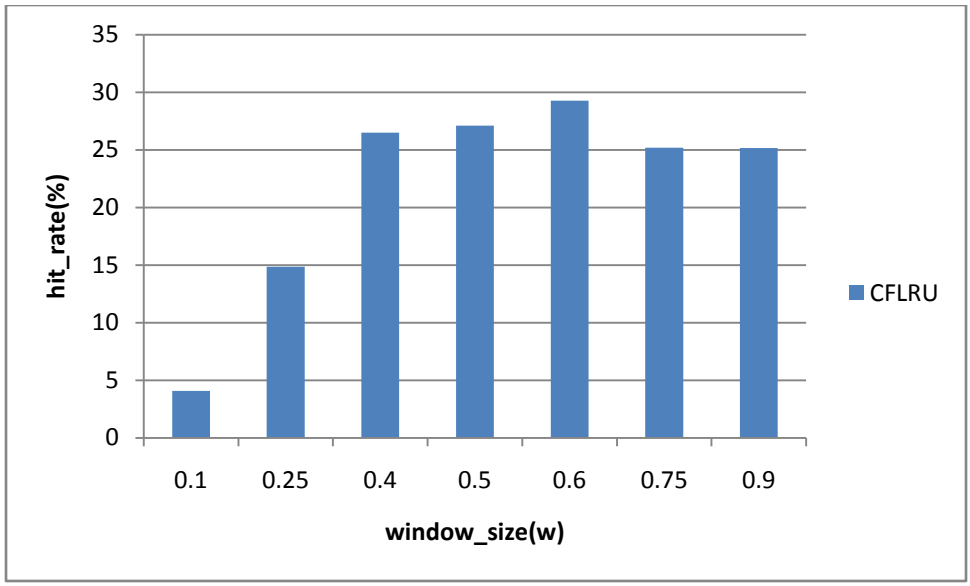


Figure 5.5 Graph for Table 5.3 showing hit rate for CFLRU with varying window_size

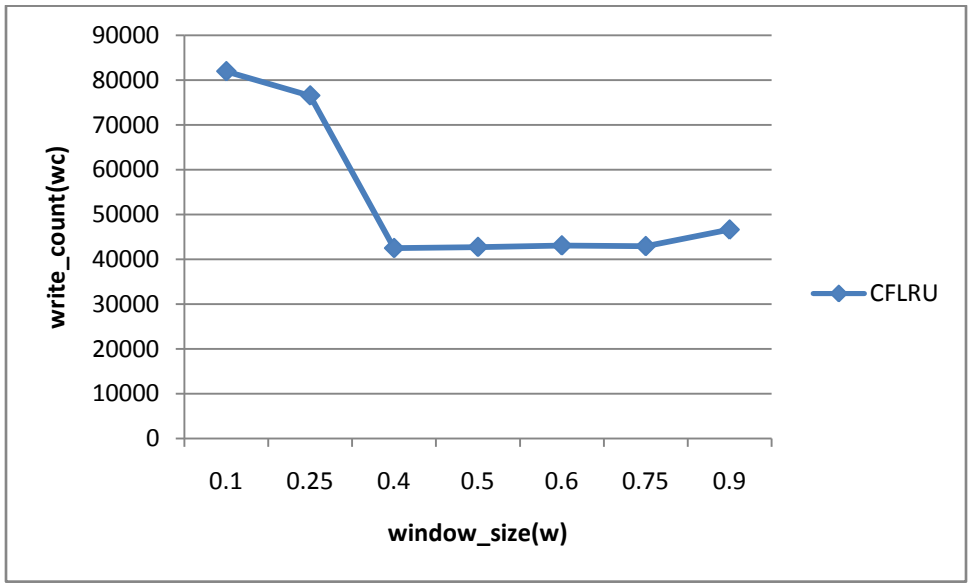


Figure 5.6 Graph for Table 5.3 showing write_count for CFLRU with varying window_size

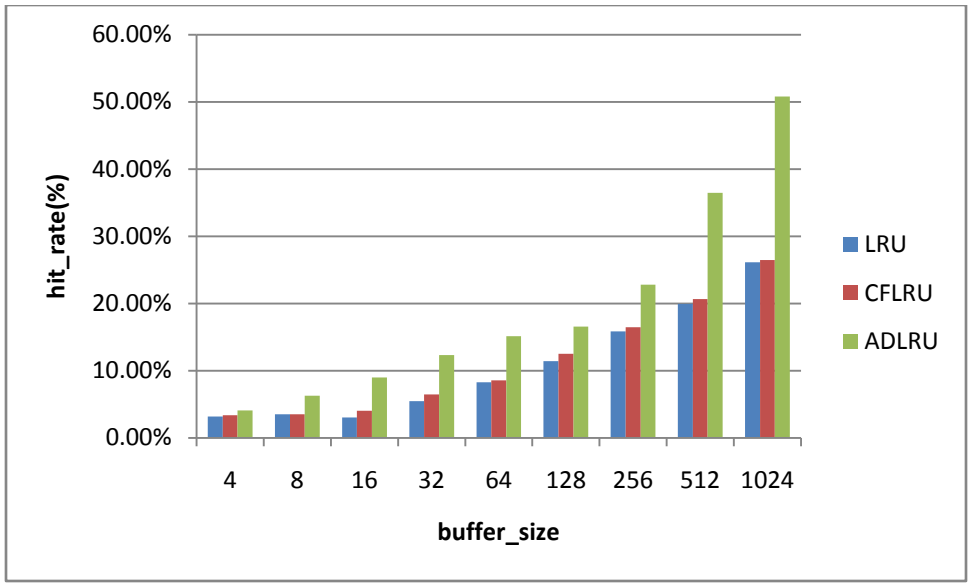


Figure 5.7 Graph for Table 5.4 showing hit rates

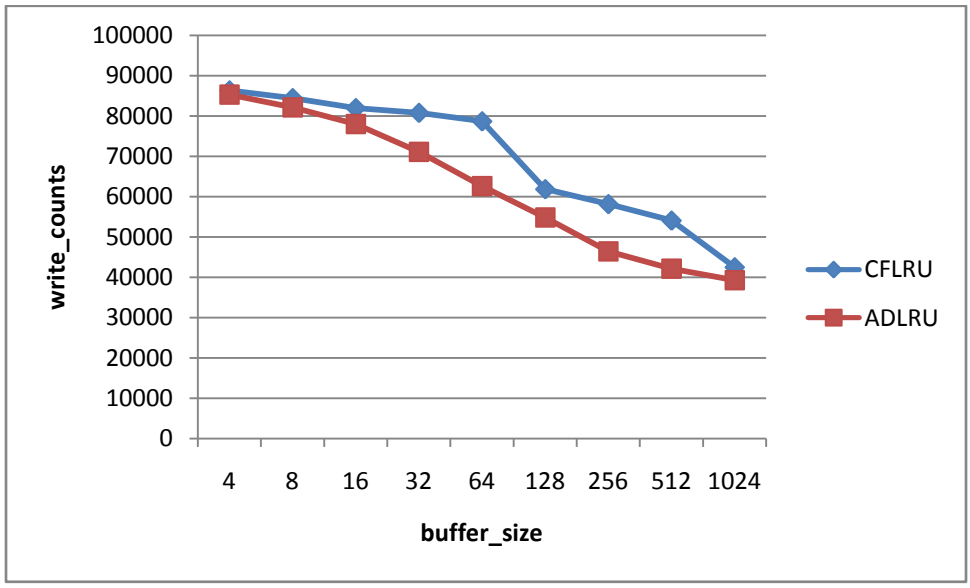


Figure 5.8 Graph for Table 5.4 showing write counts

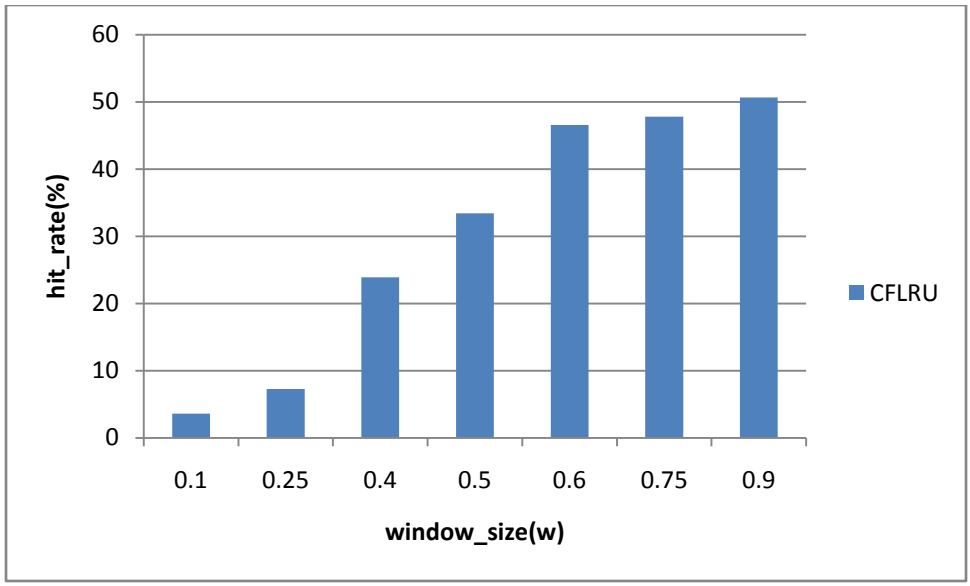


Figure 5.9 Graph for Table 5.5 showing hit rate for CFLRU with varying window_size

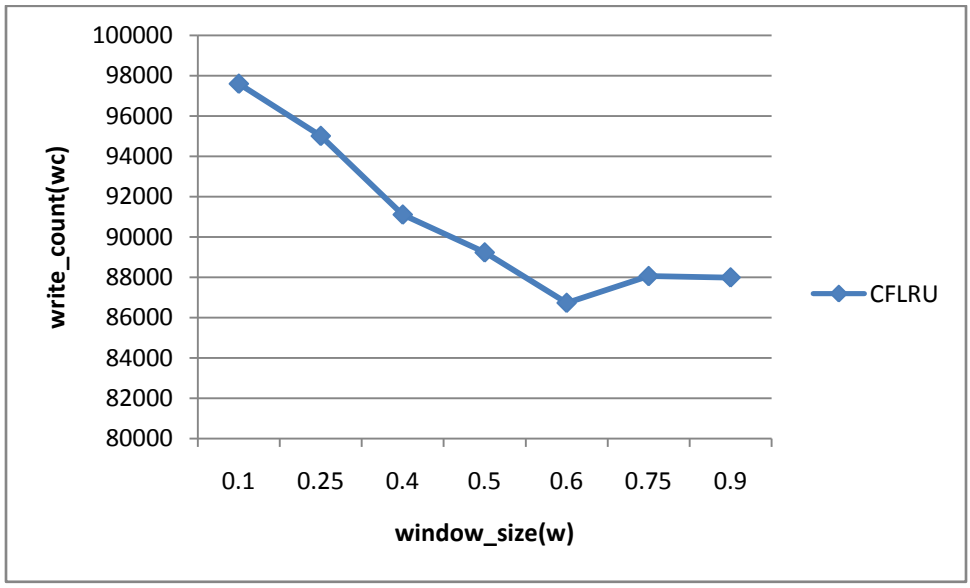


Figure 5.10 Graph for Table 5.5 showing write_count for CFLRU with varying window_size

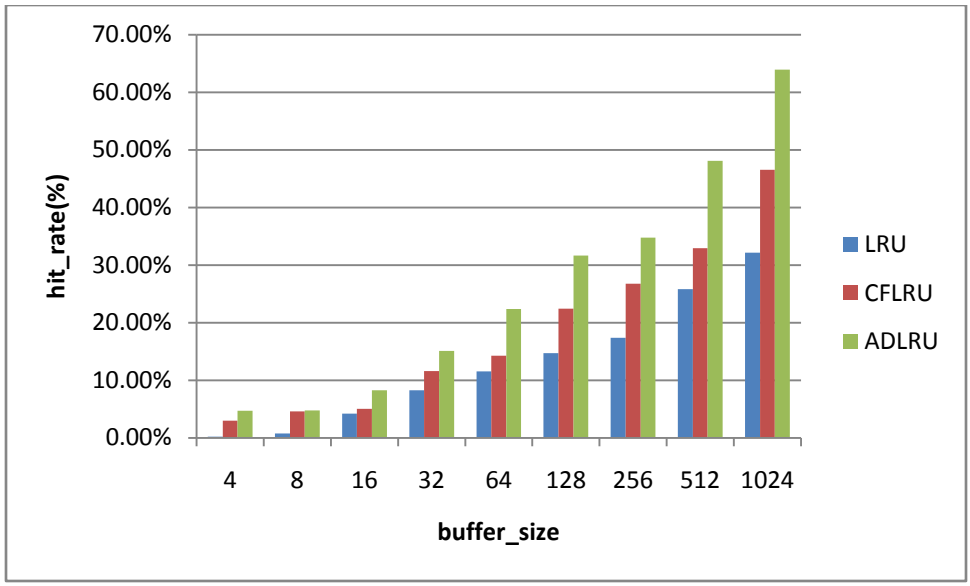


Figure 5.11 Graph for Table 5.6 showing hit rates

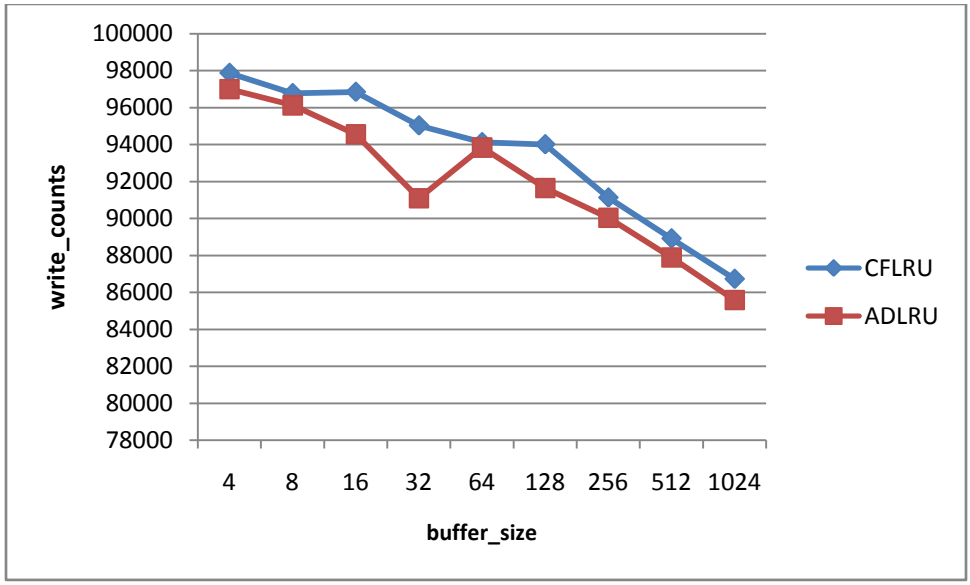


Figure 5.12 Graph for Table 5.6 showing write counts

Figure 5.1 shows that increasing in window size gradually from 0.1 to 0.9 also increase in the hit rate. But hit rate only is not the factor for evaluating the performance of buffer replacement algorithms for flash memory based systems. For this another factor write counts also must be taken. Hence, Figure 5.2 shows that the minimum write count of CFLRU algorithm is at window size 0.5 for random traces i.e. Workload 1. Similarly Figure 5.5, Figure 5.6 are used to calculate the optimal window size for Workload 2 and Figure 5.9, Figure 5.10 are also used to calculate

the optimal value of window size for Workload 3. After finding the optimal value of window size for each of the three Workloads taken, that value is taken for CFLRU for window size and then the algorithms LRU, CFLRU and ADLRU are compared which is shown clearly in graphs of Figure 5.3, Figure 5.4, Figure 5.7, Figure 5.8, Figure 5.11 and Figure 5.12.

All the column graphs of Figure 5.3, Figure 5.7 and Figure 5.11 are for comparing the algorithms on the basis of hit rates. And similarly all the line graphs of Figure 5.4, Figure 5.8 and Figure 5.12 are for comparing the algorithms on the basis of write counts. Hence from all those different graphs it is clear that for all the Workloads the hit rate of ADLRU outperforms the LRU and CFLRU. Similarly the write counts of ADLRU and CFLRU for Workload 1 i.e. for random traces, they are comparative though less number of write counts of ADLRU. For Workload 2 i.e for readmost traces, for smaller sized buffer the write counts of both algorithms is same but as the buffer size also increases the write counts of ADLRU drastically decreases. But in case of Workload 3, the write counts of ADLRU is less than CFLRU upto buffer size 32. But, for buffer size 64, again the write count of both CFLRU and ADLRU algorithms becomes equal. Again, by increasing the buffer size further upto 1024, the write count also decreases for both algorithms.

Hence from the experiments performed, the consequence is that in most of the cases the ADLRU outperforms its competitive algorithm CFLRU buffer replacement algorithm and the traditional disk based algorithm LRU, in both hit rates and the write counts since ADLRU captures both the frequency and recency of page references. It also uses the adaptive mechanism to make the sizes of the two lists cold and hot LRU suitable for different reference patterns.

Chapter – Six

Conclusion and Recommendation

6.1 Conclusion

Since, the use of flash memory requires buffer replacement policies 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 i.e . write counts and erase counts. As a consequence a replacement policy should minimize the number of write/erase operations on flash memory and at the same time increase the hit ratio. Since LRU only considers the recency of page references while CFLRU considers recency & cleanliness properties of the reference pages but AD-LRU not only considers recency & Cleanliness but also the frequency[25] of page refereces, the LRU algorithm has the worst performance for each of the workloads which clearly indicates that the traditional disk based replacement algorithms will not work well in flash memory based systems since it do not have any provisions regarding the write or erase counts. CFLRU clearly indicates that it is a good choice to first evict clean pages from the buffer while replacement for flash based systems. Similarly, ADLRU clearly indicates that it is a good choice to first evict cold clean page i.e. a clean page with frequency less than 1. Hence from the simulations and experiments performed so far, ADLRU algorithm has a lower number of write counts and at the same time high hit ratio which clearly indicates that ADLRU exhibits superior performance behavior than the others.

For buffer size 1024, the performance of CFLRU has a little bit higher than LRU about 0.28% but at the same time, the performance of ADLRU has 17.73% higher than CFLRU and 18.01% highr than LRU for Workload 1 i.e. for random type of page refernces. The performance gain by ADLRU for Workload 2 i.e. for readmost type of page refernces is about 24.32% higher than the other algorithms. Similarly, the performance gain of ADLRU for Workload 3 i.e. for writemost ype of page refernces, it is about 31.8% hogher than LRU and 17.7% higher than CFLRU. Thus from above results, the dissertation concludes that, ADLRU is superior than CFLRU and LRU in most of the cases for flash memory based systems, since it considers recency, cleanliness as well as frequency of the page references

and it uses an adaptive mechanism to make the size of hot and cold LRU lists suitable for different page reference patterns.

6.2 Recommendation

The algorithms in this dissertation can also be implemented using other standard benchmarks and additional real input traces for further performance evaluations. Another future work many be to use two or more than two queues to organize the buffer pages so that different types of frequencies can be supported. But the additional overhead may arise using more queues but it may be helpful to improve the hit rates as well as to reduce the write counts.

References

[1] L.M.Grupp, A.M. Caulfield, J.Cobum, “Characterizing Flash Memory: Anomalies, Observations and Applications”, MICRO, Dec. 2009.

[2] J. In,I. Shin, H. Kim, “SWL- A Search While-Load Demand Paging Scheme with NAND Flash Memory”, Samsung Electronics Co. Ltd., ACM-2007.

[3] L.Grippa, R. Micheloni, I.Motta, & M. Sangalli, “NonVolatile Memories: NOR Vs. NAND Architecture”, Springer-Verlag Berlin Heidelberg, 2008.

[4] Y. Yoo, H. Lee, Y. Ryu, H. Bahn, “Page replacement algorithms for NAND ash memory storages”, in: computational Science and Its Applications (ICCSA 07), 2007.

[5] Y. Ou. T. Harder, “Clean First or Dirty First: a Cost-Aware Self-Adaptive Buffer Replacement Policy”, in: Proc. of the 14th International Database Engineering & Applications Symposium, ACM, 2010.

[6] 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).

[7] Y.Joo, Y.Choi, J.Park, “Energy and Performance Optimization of Demand Paging with OneNAND Flash Memory”, IEEE,2008.

[8] Silberschatz, A., Galvin, P. B., & Gagne, G., “Operating system concepts”, 7th Edition, 2004

[9] S. Jing, F. Chen, X. Zhang, “CLOCK-Pro – An Effective Improvement of the CLOCK Repalcement”, USENIX Annual Technical Conference, 2005.

[10] Amit S. Charan, Kartik R. Nayak, “A Comparasion of Page Repalcement ALgorithms”, IACSIT International Journal of Engineering and Technology, Vol. 3, April 2011.

[11] E. O'neil, P. O'neil, G. Weikum, “The LRU-K Page Replacement Algorithm for Database Disk Buffering”, 1993.

[12] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho, C. Kim, “LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies”, IEEE Trans. on Computers (2001).

[13] T. Johnson, S. D., “2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm”, 1994.

[14] S. Jiang, X. Zhang, "LIRS: An Efficient Low Inter-Reference Recency Set Replacement Policy to Improve Buffer Cache Performance", ACM New York, NY, USA, 2002.

[15] X. Tang, X. Meng,"ACR: an Adaptive Cost Aware Buffer replacement Algorithm for Flash Storage Devices", IEEE-2010

[16] S. Bansal and D.Modha, "CAR: Clock with Adaptive Replacement",Proceedings of the 3nd USENIX Symposium on File and Storage Technologies, 2004.

[17] A.S. Tanenbaum, Modern Operating Systems (Prentice Hall Second Edition), 2007.

[18]S.Y. Park, D. Jung, J. Kang, "CFLRU-A Replacement Algorithm for Flash Memory", Korean Advanced institute of Science & Technology, ACM-2007

[19] N. Megiddo, D. S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache", 2003.

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

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

[22] P. Jin, Yi Ob, T. Harder, Z. Li, "The AD-LRU: An Efficient Buffer Replacement Algorithm for Flash Based Systems", Department of Computer Science and Information Technology, University of Science & Technology of China, September 2011.

[23] Z. Li, P. Jin, X.Su, "CCF-LRU: A new Buffer Replacement Algorithm for Flash Memory", Transaction on Consumer Electronics,2009.

[24] M.L.Singh, "Understanding Research Methodology", Scientific Method and Research.

[25] L.P. Chang, T.W. Kuo, "Efficient Management for Large Scale Flash Memory Storage Systems with Resource Conservation", ACM Transaction 2005.

APPENDIX A : A sample trace of Workload 1(random traces)

1,8575 0,17754 0,33289 0,3838 0,19942 1,25113 1,35145 1,1939
0,40780 0,12831 0,31724 1,37162 1,861 1,35912 0,39216 1,10863
0,15454 0,32425 0,42141 1,34769 0,29923 1,3050 0,4043 0,39113
1,11686 1,25837 0,4941 0,7882 0,39262 1,32631 0,36490 1,11934
1,8851 1,16962 0,37665 1,23980 0,41727 0,15074 1,19029 1,1750
0,49554 1,18797 1,6747 0,31276 1,786 1,42798 0,30971 0,42594
1,49503 0,23075 1,8717 1,13521 1,988 0,22467 1,12586 1,45284
1,39329 0,45058 0,14795 0,21120 0,7786 1,43211 0,47655 0,42213
0,919 0,603 0,4844 0,44923 1,29324 0,26292 1,31526 1,38097
1,39819 0,30117 1,14208 1,27844 1,8361 1,16455 1,5699 0,10670
1,1066 0,9039 1,6477 1,41170 0,23504 1,32354 0,14280 1,36795
1,8732 1,46002 1,4880 1,5637 1,21680 1,3496 1,3220 0,13282
1,42670 0,11669 0,2716 1,49749 0,12437 0,42550 0,27038 0,26790
1,44095 1,25674 0,4498 1,32206 0,33123 0,9846 0,46190 0,20089
0,14060 1,28875 1,16434 1,12575 1,47687 1,41433 0,16610 0,3411
1,23633 1,17429 0,49681 1,25625 1,34155 0,33804 0,21089 0,16647
1,3104 0,3843 1,7142 0,30193 0,12695 1,28453 0,9115 0,25532
0,47722 1,47868 1,49752 0,6476 1,41825 1,7631 0,14127 0,29127
1,12805 0,48855 1,33911 0,41079 1,25483 1,39430 0,1037 0,3297
0,16599 1,36036 0,15578 0,10091 1,25578 0,23037 0,24073 1,16386
0,15490 1,1048 0,19682 0,8798 0,26493 0,48889 1,7791 1,35987
1,16638 1,45825 1,38057 0,30566 0,48228 0,38949 1,47502 0,26137
1,22920 1,32430 1,7944 1,35589 1,40867 1,47773 0,46838 1,44616
1,39286 1,39175 0,42242 1,20480 1,22293 0,20389 0,23900 0,18555
0,46427 0,8516 1,49886 0,10679 0,9400 1,24467 0,3709 0,411
0,25540 0,22153 1,3954 0,23179 0,9759 1,33020 0,2711 1,42697
1,34063 1,22716 1,23599 0,25436 1,22036 0,34470 1,12097 0,16505
0,30238 0,37133 1,47578 1,43832 1,12285 1,43630 1,25872 1,22922
0,47801 0,33166 0,30809 0,22288 1,15530 1,18379 1,26444 0,15686
1,47213 0,23218 0,17078 0,9358 1,22390 1,12973 1,12756 0,14487
1,14736 0,17512 0,16192 1,20303 0,13516 0,7694 0,46578 1,14630
0,44937 0,8268 1,27634 1,42340 1,5782 0,18033 0,36288 1,20348
1,25705 0,13909 0,48059 1,39900 1,42855 1,22621 0,19304 1,34024
1,12876 1,24241 1,44899 0,44903 0,47548 0,2683 1,15201 1,40903
0,49098 0,11108 1,8735 1,23818 0,9948 0,2917 0,17513 1,26798
0,37204 1,48172 0,41628 0,21196 0,16265 1,33034 0,49456 1,20361
0,46366 0,49157 0,22078 1,21714 0,36970 1,8646 1,9469 0,36225
1,46020 1,1702 1,2979 1,21191 1,46458 0,48207 1,29311 1,33363
0,18666 1,9636 1,25704 1,43329 1,32357 0,34652 1,29324 0,12938

1,36894 0,43868 1,47460 1,47637 0,24355 1,12176 0,12286 1,7957
0,41794 0,49251 1,29601 0,45568 0,43907 1,10770 0,27169 1,45435
0,9439 1,46069 0,14691 1,2992 0,20988 1,23961 0,42924 0,16206
1,17346 1,17766 0,19820 1,17517 0,1382 0,41472 1,21713 0,47315
1,3598 0,22383 0,54 0,28589 0,39334 1,5327 1,31446 0,11905
0,2578 0,42675 1,36464 0,10865 1,38028 1,10859 0,21603 0,31835
0,47015 0,11141 1,37753 0,41532 0,5012 1,42335 1,19108 1,6948
0,3639 0,37524 0,14929 1,30811 0,39972 1,17926 1,20116 1,24335
0,49921 0,42809 1,20340 1,3331 0,46753 1,20246 1,2500 1,30976
1,18790 1,35884 1,8436 1,8547 0,40654 0,37234 1,4720 0,14709
0,49005 1,120 1,49345 1,21382 0,5659 1,8975 1,35557 0,10747
0,22871 0,40127 1,8215 0,32476 1,5622 0,11686 1,7083 0,47138
0,8859 1,49704 1,26240 0,36223 1,4757 1,46740 1,43596 1,44175
1,11825 1,10458 0,30016 1,43645 0,28919 1,6579 1,47380 0,4275
0,47727 1,21962 1,32498 0,15530 1,22466 1,45797 0,44519 0,47839
0,34722 0,29879 0,7742 1,35170 0,8487 0,32399 0,48678 0,33246
1,46767 0,889 1,15261 1,45658 0,14995 1,18321 0,7303 1,1460
1,2724 1,14771 1,49585 0,2935 1,3603 0,47776 0,48439 0,36931
1,27169 1,48598 0,10737 1,23017 1,46089 0,32556 0,1153 1,12834
1,8672 1,33281 0,8337 0,16815 0,14078 0,23123 1,38627 1,3974
1,39029 0,24143 0,45127 1,6153 0,21868 1,3032 0,49348 0,27357
0,2787 1,41676 0,45740 1,12125 1,35064 1,6677 0,36840 0,23619
0,28071 0,4706 0,33306 0,45229 0,44371 1,41742 0,29463 1,35470
1,20690 1,21422 0,2804 0,9359 1,23231 0,21345 0,1137 1,49613
0,17119 0,33336 1,32844 1,3105 0,13328 1,27617 0,21679 0,23756
0,1447 1,16627 0,10501 1,6270 1,48493 0,16937 0,3405 1,35778
0,40294 1,19994 0,19317 1,21849 0,16686 1,10258 0,37921 0,37620
1,34475 0,39281 1,7300 0,44227 1,13074 0,24621 1,14952 0,21515
0,19190 1,3207 1,37054 0,45755 1,793 0,15683 0,13356 1,15584
1,41297 0,13732 0,3287 0,35325 0,45480 0,9126 0,6162 1,45840
0,16771 0,34900 0,8499 0,44231 1,23597 1,30814 1,5684 0,32808
1,39345 0,6659 1,9489 1,13594 1,30040 0,18013 1,27989 0,38871
1,44336 0,30589 0,27927 0,11027 1,2111 1,24641 0,27437 1,13143
0,7526 1,13994 1,15792 0,39187 0,21976 0,12247 1,44256 1,14579
0,27233 0,35737 0,49200 0,5252 1,1753 1,39824 1,39857 1,13701
0,4107 1,23362 0,49430 0,40719 1,37172 0,20166 1,4892 1,30819
1,41838 0,38020 0,33332 1,23075 1,24285 1,49272 1,19458 0,20474
0,15879 1,11600 0,48591 0,48481 1,29903 1,11074 0,30888 1,49938
0,16396 0,6674 1,43204 1,43781 1,26073 1,41564 1,3268 1,34426
0,19360 0,41111 1,29064 0,16136 1,38420 1,2721 0,46284 0,49374
0,11741 0,49831 1,11262 1,36635 0,25753 1,30251 0,13295 1,9735

APPENDIX B : A sample trace of Workload 2(readmost traces)

0,47138 0,8885 0,46509 0,30725 1,15160 0,2460 0,9807 0,46791
1,5087 0,11237 0,22932 0,37902 0,6713 0,34922 0,4119 0,42689
0,25737 0,39402 0,9355 0,10606 0,641 0,27320 0,38193 0,21972
0,42518 0,10783 0,28314 1,1900 0,13867 0,39219 0,46605 1,38017
0,46494 0,23527 0,38630 1,21176 0,293 0,12907 0,39277 0,40610
0,7266 1,41366 0,30769 1,8749 1,10029 0,1320 0,46614 0,41918
0,26128 0,41673 0,19547 0,48693 0,37972 0,38947 0,15954 0,3438
0,18472 0,16481 0,6566 0,9291 0,43502 1,33032 0,3183 0,19948
0,6053 1,38512 0,46694 0,33131 0,29974 0,19584 0,49468 1,24278
0,17376 0,46130 0,4161 0,3133 0,45468 0,35567 0,36470 0,24196
1,34021 0,39449 0,18771 0,19982 0,26021 0,17350 0,44669 0,11232
0,2877 0,14913 0,26197 0,37578 0,44932 0,27057 0,8577 0,21545
1,19614 0,26010 0,31719 0,21978 0,9246 0,32690 0,35125 0,29523
0,34981 1,3135 1,2971 0,1054 0,15836 0,29720 0,39483 0,42668
0,23341 1,7058 1,37083 0,5836 0,39234 0,30664 0,47423 0,48384
0,49832 0,47732 0,6181 0,28049 0,20673 0,14815 1,16584 0,35416
0,15178 1,22743 0,37824 0,20809 0,43815 0,7992 1,22767 1,981
0,6349 0,22302 0,1909 0,37810 0,24271 0,27349 0,21940 0,11289
0,3186 0,14000 0,38546 1,20359 0,34039 0,3939 0,3492 0,44098
0,2151 0,17422 0,30562 1,24662 0,23074 0,26344 1,31895 0,6416
0,48410 0,15522 0,14390 0,34163 0,13073 0,19750 0,985 0,48011
1,18012 0,11608 0,14481 0,34997 0,22648 0,26672 0,15980 0,49335
1,34079 0,11814 0,31534 0,20259 0,11874 0,45185 1,20792 0,39186
0,18681 0,24097 0,8582 0,26107 1,11335 0,33248 0,31662 0,47539
0,2856 0,41237 0,19933 0,10902 1,6574 0,14599 0,39656 0,15879
0,9645 0,32760 0,11311 0,14258 0,38921 0,47086 0,24615 0,36799
1,23373 0,30556 0,6997 0,5647 0,22385 0,14890 0,5537 0,11311
0,18829 0,9608 0,44776 0,35106 0,21597 0,18245 0,25921 0,19819
0,41022 0,2924 0,33953 0,9818 0,21029 0,1955 0,26130 0,48683
0,16144 0,42243 0,1071 0,48155 0,17289 0,24699 1,19033 0,18424
0,39192 0,45975 0,949 0,25811 0,35775 0,28294 0,7946 0,2748
0,36907 0,5078 0,27022 1,14669 0,37419 0,12382 0,8955 0,43073
0,4139 0,37292 0,31386 0,15131 0,44501 0,40518 0,6139 0,49892
0,22521 0,9057 1,43638 0,45879 0,30391 0,14690 0,25367 0,10125
0,24894 0,41810 0,49555 0,38776 0,16140 0,49637 0,36102 0,13534
0,4838 1,33623 0,19639 0,33611 1,38969 0,34042 0,32887 0,13925
0,12100 0,10997 0,8528 0,11794 0,23601 0,15213 0,29736 0,47737
0,15336 0,16109 0,10809 0,36945 0,49102 0,40775 1,2132 0,12292
1,28002 0,29787 0,12657 0,27496 0,11586 0,35950 0,19189 0,7309

0,16707 0,45708 0,43469 0,32897 0,21864 0,18648 0,36112 0,29233
0,18148 0,37425 0,21023 0,8947 0,30022 0,43937 0,18352 0,32213
0,26617 0,6472 0,9465 0,1001 0,33444 0,38882 0,20992 0,19267
0,31603 0,47080 0,36330 0,27784 0,9610 0,2085 0,47824 0,8152
0,2037 0,7399 0,2509 0,15924 0,21722 0,22456 0,1344 0,31460
0,26839 0,13079 0,9131 0,4649 0,42108 0,8980 0,31326 0,1164
0,2116 1,33931 0,19819 0,21947 0,43479 0,12174 0,34616 0,18643
0,9172 0,15023 0,16227 0,33152 0,25781 1,32700 0,27830 0,33616
0,43438 0,24311 0,33625 0,27599 0,5211 0,12664 0,41573 0,46198
0,49994 1,31239 0,47727 0,646 1,37050 0,12074 0,32477 0,42313
0,47770 0,46694 0,20214 0,26103 0,5700 0,16571 1,16484 0,27397
0,42946 0,31008 0,29316 0,7067 0,26433 0,28105 1,13643 0,21567
0,41547 0,43496 0,27556 0,13613 0,43995 0,6948 0,13336 0,14920
1,37646 1,17696 0,28939 0,25819 0,41267 0,8789 0,49076 0,43432
0,34655 0,18554 1,2156 0,29334 0,23288 0,20318 0,23510 0,29543
0,20736 0,25775 0,41235 0,20547 0,48593 0,14735 0,15425 0,37705
0,20563 0,5176 0,23816 0,4212 0,73 0,1533 0,31879 0,35956
0,33468 0,18525 0,16509 0,21445 0,49441 1,27666 0,48467 0,18078
0,20598 0,2954 0,3779 1,42299 0,41212 0,49422 0,5285 0,34482
0,2121 0,12208 0,16843 0,43890 0,5114 0,29368 0,18048 0,46923
0,8388 0,38472 0,26362 0,22877 0,47095 0,16052 0,25752 1,28985
0,18284 0,20430 0,19205 0,6473 1,6948 0,3957 0,4795 0,41872
0,17352 1,2385 0,44678 0,36418 0,40141 0,39450 0,11828 0,18181
0,28935 0,26468 0,32854 0,3130 0,39175 0,37648 0,6903 0,33788
0,25156 0,10791 0,20301 0,38533 1,33099 0,22855 0,14319 0,32810
0,13657 0,18918 0,10063 1,45405 0,41789 0,19834 0,8878 0,42519
0,22301 0,31632 0,34843 0,9710 0,29224 0,45669 0,33202 0,28723
0,28469 0,19276 1,48889 0,32789 0,22921 0,37976 1,14042 0,40266
0,10572 0,2371 0,1856 0,27864 0,48851 0,12320 0,8046 0,36058
0,26917 0,45898 0,13012 0,33703 0,37801 0,35456 0,46517 1,39406
0,42940 0,34131 0,25288 0,10775 0,20746 0,18061 0,382 0,28769
0,4438 0,45124 0,14697 1,39087 0,15722 0,27004 0,20125 0,34195
0,43287 0,49173 0,36883 0,16981 0,29870 0,31391 0,32459 0,20107
1,30190 0,23806 1,19013 0,28304 0,49550 0,6669 0,4493 0,14581
0,8351 0,11261 0,20004 0,41954 0,24849 0,62 0,28875 1,3940 0,429
1,34690 1,4804 0,24203 0,46097 0,22611 0,31691 0,34656 0,9863
0,46018 0,18755 0,47633 0,3477 0,32363 0,13419 0,33842 0,36158
1,25152 0,49592 1,1617 0,26068 0,35534 0,48057 0,1741 0,11145
0,47406 0,17319 0,13342 0,32230 0,9724 0,7757 0,17773 0,3723
0,26021 0,9208 0,10427 0,27148 0,37419 0,42750 0,43348 1,41872
0,11259 0,23443 0,41763 0,49494 0,18709 0,34267 0,14046 0,27289

APPENDIX C : A sample trace of Workload 3(writemost traces)

1,12527 1,1216 1,698 1,35286 1,39722 1,25887 1,45028 1,47558
1,44966 1,10018 1,41052 0,8011 1,42731 1,9714 1,39263 1,40196
1,6269 1,39623 1,33031 1,1853 1,29107 1,5242 1,1010 1,28122
1,35606 1,27792 1,19845 1,24155 1,20899 1,37819 1,27592 1,1272
1,2536 1,35733 1,33645 1,37360 1,13287 1,35073 1,24973 1,31865
1,7424 1,5993 1,8751 1,2237 1,39556 1,8440 1,35811 1,25015
1,42880 1,12603 1,8230 1,45262 1,10924 0,40802 1,24112 1,38237
1,31304 1,5412 1,43801 1,29898 1,10638 1,47683 1,4487 1,44810
1,8571 1,9911 1,33896 0,35169 1,35950 1,9344 1,2859 1,32483
1,2158 1,46525 1,32777 1,20380 0,25035 1,5188 1,6797 0,24879
1,8889 1,19975 1,8644 1,8494 1,17945 1,5175 1,29078 1,36322
1,46605 0,3722 1,23254 1,35573 1,44707 1,16353 1,23944 1,24724
0,40235 1,9453 1,33001 1,23185 1,19468 1,4818 1,18662 1,14189
0,1378 1,16011 1,18092 1,36090 1,37183 1,4364 1,33538 1,41008
1,19253 1,34763 1,21453 1,5052 1,38178 1,39783 1,33887 1,46310
1,2396 1,41563 1,18490 1,18554 1,46076 1,3812 1,46712 1,22442
0,15937 1,38230 1,45473 1,6945 1,24479 1,9632 1,21724 1,12421
1,20451 1,35388 0,980 1,4486 1,47436 1,44968 1,42560 1,34505
1,42484 1,8868 1,13237 1,45460 1,40381 1,46871 1,18937 1,1389
1,22092 1,20688 1,30869 0,45818 1,47306 1,3497 1,1803 1,6096
1,24012 1,43783 1,7630 1,24744 1,47367 1,42187 1,43951 1,21302
1,26076 1,12092 0,38106 1,21666 1,45645 1,12638 1,5712 0,14779
1,33647 1,29306 1,20191 1,33315 1,26443 1,11996 1,28139 1,18374
1,24340 1,26206 1,6606 1,1590 1,16723 1,48509 1,29078 1,36414
1,5498 0,24528 1,43092 1,11633 1,27217 1,10035 1,5380 1,2269
1,41075 1,7928 1,8105 1,3437 1,22547 1,45582 0,8817 1,38670
1,20172 1,30414 1,47214 1,19627 1,26446 1,40787 1,39687 1,3454
1,37369 1,30931 1,33101 1,18169 1,22790 1,11904 0,47052 1,3672
1,42585 1,9384 1,5275 1,13720 1,19348 1,49136 1,20843 1,19068
1,25883 1,16481 1,27189 0,29307 1,16008 1,45273 0,9839 1,38955
1,48500 1,48560 1,47897 1,37830 1,39217 1,9133 1,18904 0,10499
1,48972 0,42043 1,45152 1,1636 1,12524 1,39143 1,37057 1,9006
1,47238 1,45840 1,5534 1,45368 0,28865 1,6060 1,41228 1,31789
1,48175 1,22391 1,23196 1,34069 0,27033 1,11358 1,21846 1,38558
1,36046 1,4791 1,26938 1,20824 1,4823 1,48716 1,44135 1,28505
1,49252 1,44939 1,36081 1,29232 0,30656 0,47723 1,48222 1,35146
1,878 1,18288 1,8098 1,31077 1,8318 0,21097 0,7152 0,13565
1,46677 1,1957 1,31401 1,39787 1,27588 1,17227 1,31164 1,47753
1,12432 1,2839 1,47863 1,26882 1,6630 1,21134 1,19651 1,27453

1,14355 1,10102 1,29343 0,7942 1,1493 1,28572 0,38982 1,9057
1,15971 1,890 1,41953 1,49738 1,23491 1,31693 1,33812 1,32832
1,9872 0,9447 1,3797 0,32651 1,40169 1,10428 1,46901 1,21121
0,43432 1,31932 1,31178 1,48543 0,22614 1,46575 1,48943 1,39342
1,48404 1,1240 1,43650 1,49269 0,9064 0,21974 1,45434 1,46811
1,48624 1,49290 1,11505 1,448 1,11224 1,11304 1,47535 1,15242
1,36174 1,12490 1,4666 1,42262 1,41583 1,17152 1,12715 0,18951
1,21008 1,16825 1,25648 1,7780 0,32453 1,31306 1,31417 1,8739
1,26541 1,29079 1,20598 0,32708 1,20379 1,7102 1,47144 1,6410
1,34135 1,46739 1,30138 1,31747 1,31348 0,6005 1,22079 1,27629
1,27350 1,14363 1,32636 1,6512 1,38772 1,31748 1,40623 1,46464
0,7840 1,38240 1,41170 1,48485 0,9147 1,13603 1,21382 1,23347
1,32522 1,18992 1,48222 1,38319 1,45065 1,35093 1,29744 1,16979
0,12561 1,4371 1,11740 1,9723 1,4997 0,6338 1,9511 1,1668 1,6783
1,13377 1,2671 1,16732 1,25981 1,27165 1,81 1,35553 1,31878
1,22785 1,17572 1,9548 1,21927 1,48014 1,17879 1,17783 1,47687
1,49421 1,24950 1,45829 1,22042 1,36591 0,20713 1,26414 1,30873
1,41931 1,32470 1,17745 1,14697 1,35251 1,44569 1,43227 1,17229
1,17520 1,49316 1,14564 1,33594 1,7837 1,1316 1,35393 1,36273
1,34393 1,31050 1,17156 1,83 1,3210 1,9195 1,41171 1,39217
1,16562 1,1466 1,41723 1,36963 0,47959 1,19432 1,29369 1,15640
1,15839 1,13620 1,7344 1,16065 1,39560 1,11159 1,40807 1,26421
1,5383 0,43262 1,2821 1,16617 1,6729 1,26508 0,23680 1,9374
1,35984 1,20078 1,22442 1,12644 1,48135 0,39378 1,25798 0,14005
1,23302 1,13471 1,14359 1,21971 1,20334 1,31838 1,49621 1,28811
1,3897 1,43040 1,27843 1,32634 1,27524 1,16779 1,43505 1,25441
1,44214 1,2239 1,21654 1,40382 1,27730 0,25933 0,23356 1,32532
1,5623 0,2742 1,14485 1,21326 1,36952 1,29226 1,15461 0,16820
1,22017 0,9035 1,25265 1,11212 1,33356 1,27032 1,14152 1,15124
1,47528 1,7678 0,23761 1,48844 1,29099 0,3500 1,26389 1,2172
1,11354 0,6325 1,42822 1,26389 1,5102 1,34110 0,21757 1,20043
1,43395 1,14065 1,16337 1,43311 1,34617 1,29231 0,5727 1,27015
0,34690 1,40669 1,19798 1,28686 1,20363 1,34354 1,44407 1,24634
1,16787 1,38452 1,42688 1,40239 1,1854 0,4638 1,42761 1,26164
1,16350 1,22821 0,28702 1,6451 1,32037 1,47186 1,18653 1,45641
1,16091 1,29841 1,10516 1,7979 1,168 1,30352 1,3196 1,46214
1,30409 1,43742 1,48075 1,26852 1,35557 1,29788 1,47754 1,17588
1,17838 1,42179 1,38931 1,32941 1,47935 1,48613 1,42451 1,13432
1,25523 1,16903 1,30117 1,34689 1,32151 1,49445 1,20582 1,27493
1,41441 1,37480 1,32555 1,43946 1,5870 1,49112 0,11214 1,36303
1,18266 1,9420 1,33141 0,31688 1,32068 1,40259 1,16869 1,35169