Tribhuvan University
Institute of Science and Technology

Sensitivity Analysis of Cache Partition in CLOCK-Pro Page Replacement and its Comparison with Adaptive CLOCK-Pro

Dissertation

Submitted to

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

In partial fulfillment of the requirements
for the Master's Degree in Computer Science & Information Technology

By
Bhupendra Singh Saud
Date: April, 2013

Supervisor
Prof. Dr. Shashidhar Ram Joshi

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science & Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

... ...
Mr. Bhupendra Singh Saud
Date: 9th April, 2013

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my supervision by Mr. **Bhupendra Singh Saud** entitled "**Sensitivity Analysis of Cache Partition in CLOCK-Pro Page Replacement and its Comparison with Adaptive CLOCK-Pro**" in partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Information Technology be processed for the evaluation.

... ...
Prof. Dr. Shashidhar Ram Joshi
Department of Electronics & Computer Engineering,
Institute of Engineering,
Pulchowk, Nepal
Date: 9th April, 2013

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science & Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Evaluation Committee

Asst. Prof. Nawaraj Poudel	**Prof. Dr. Shashidhar Ram Joshi**
Acting Head of Department	Department of Electronics
Central Department of Computer Science	& Computer Engineering
& Information Technology	Institute of Engineering,
(External Examiner)	**(Internal Examiner)**

Date: 15th April 2013

Acknowledgement

I would like to express my gratitude to all the people who supported and accompanied me during the preparation of this dissertation **"Sensitivity Analysis of Cache Partition in CLOCK-Pro Page Replacement and its Comparison with Adaptive CLOCK-Pro".** This research work has been performed under Central Department of Computer Science and Information Technology (***Tribhuwan University***), Kirtipur. I am very grateful to my department for giving me an enthusiastic support.

First, I would like to express my gratitude to my supervisor **Professor Dr. Shashidhar Ram Joshi,** head of the Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk. This research would not have been possible without his advices and patience.

I deeply extends my heartily acknowledgement to **Mr. Arjun Singh Saud,** lecturer CDCSIT, who gave me an enthusiastic support of the preparation of this dissertation. He is the one who listened to all my problems I faced during this thesis and showed me the way to overcome them.

Most importantly I would like to thank to respected Head of Department of Central Department of Computer Science and Information Technology, Assoc. Prof. Dr.Tanka Nath Dhamala**,** respected teachers Prof. Sudarsan Karanjit, Ast. Prof. Dr. Subarna Sakya, Mr. Min Bahadur Khati, Mr. Bishnu Gautam, Mr. Dinesh Bajracharya, Mr. Navaraj Poudel, Mr. Jagdish Bhatta, Mr Dheeraj Kedar Pandey, Mr.Sarbin Sayami, Mr. Tej Bahadur Shahi & Mrs. Lalita Sthapit of CDCSIT, TU, for providing me such a broad knowledge and inspirations.

Special thanks to my family and members of educational organizations that I have been working, for their endless motivation, constant mental support and love which have been influential in whatever I have achieved so far.

I wish to thank to all my colleagues and friends especially Mr. Deepak Bhatt, Mr. Upendra Raj Joshi, and Mr. Dipak Bhatt for supporting me directly and indirectly in this research work.

I have done my best to complete this research work. Suggestions from the readers are always welcomed, which will improve this work.

Abstract

With the ever-growing performance gap between memory systems and disks, and rapidly improving CPU performance, virtual memory (VM) management becomes increasingly important for overall system performance. However, one of its critical components, the page replacement policy, is still dominated by CLOCK, a replacement policy developed almost 40 years ago. While pure LRU has an unaffordable cost in VM, CLOCK simulates the LRU replacement algorithm with a low cost acceptable in VM management. Over the last three decades, the inability of LRU as well as CLOCK to handle weak locality accesses has become increasingly serious, and an effective fix becomes increasingly desirable. This dissertation work is focused on an improved CLOCK replacement policy, called CLOCK-Pro. By additionally keeping track of a limited number of replaced pages, CLOCK-Pro works in a similar fashion as CLOCK with a VM-affordable cost. CLOCK-Pro improves weaknesses of CLOCK especially in weak locality of references by using reuse distance also known as IRR.

For weak locality workloads CLOCK-Pro and adaptive CLOCK-Pro always performs better than CLOCK page replacement algorithm. CLOCK-Pro page replacement policy increases hit rate up to 40% and adaptive CLOCK-Pro increases hit rate up to 43%. Performance gain is more in case of purely weak locality workloads (such as looping pattern) and performances are comparable in case of strong locality workloads (such as temporally clustered workloads). Again, if the number of distinct pages is nearly equal to cache size CLOCK algorithm also performs better for weak locality workloads.

Keywords: Cache memory, CLOCK page replacement algorithm, CLOCK-Pro page replacement algorithm, Adaptive CLOCK-Pro page replacement algorithm, Page faults, Cache partition, History pages or Non-resident Cold Pages, Cold Pages, Hot Pages.

Table of Contents

CHAPTER 1

Background & Introduction

CHAPTER 2

Literature Review & Methodology

CHAPTER 3

Page Replacement Algorithms carried out in Dissertation work

CHAPTER 4

Implementation

CHAPTER 5

Data Collection & Analysis

CHAPTER 6

Conclusion and Future Study

List of Figures

List of Tables

List of Abbreviations

RAM	-	Random Access Memory
SRAM	-	Static RAM
DRAM	-	Dynamic RAM
VM	-	Virtual Memory
LRU	-	Least Recently Used
LFU	-	Least Frequently Used
2Q	-	Two Queues
ARC	-	Adaptive Replacement Cache
CAR	-	Clock with Adaptive Replacement
CART	-	CAR with Temporal Filtering
CLOCK Pro	-	Clock with Pro
CPU	-	Central Processing Unit
LIRS	-	Low Inter-reference Recency Set
LRFU	-	Least Recently Frequently Used
MRU	-	Most Recently Used
NRU	-	Not Recently Used
OPT	-	Optimum
PFF	-	Page Fault Frequency
MR	-	Miss Rate
HR	-	Hit Rate
NPF	-	Number of Page Fault
NDP	-	Number of Distinct Pages
IRR	-	Inter- Reference Recency
LIR	-	Low Inter-reference Recency
HIRS	-	High Inter-reference Recency Set
HIR	-	High Inter-reference Recency
GCLOCK	-	Generalized CLOCK
JRE	-	Java Runtime Environment
IDE	-	Integrated Development Environment
CDLL	-	Circular Doubly Linked List

CHAPTER 1

Background and introduction

1.1 Background

1.1.1 Memory Hierarchy

The term memory hierarchy is used in computer architecture when discussing performance issues in computer architectural design, algorithm predictions, and the lower level programming constructs such as involving locality of reference. A "memory hierarchy" in computer storage distinguishes each level in the "hierarchy" by response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by the controlling technology. Even though varieties of memory devices which vary on response time, cost, reliability, memory capacity etc. are available in today's market, the computer system has limited memory. Memory Hierarchy is the ranking of memory devices so as to achieve higher performance with in the limited storage capacity. Memory Hierarchy consists of different levels of memory that are faster one over other but faster memory is costlier and has low storage capacity compared to slower memory.

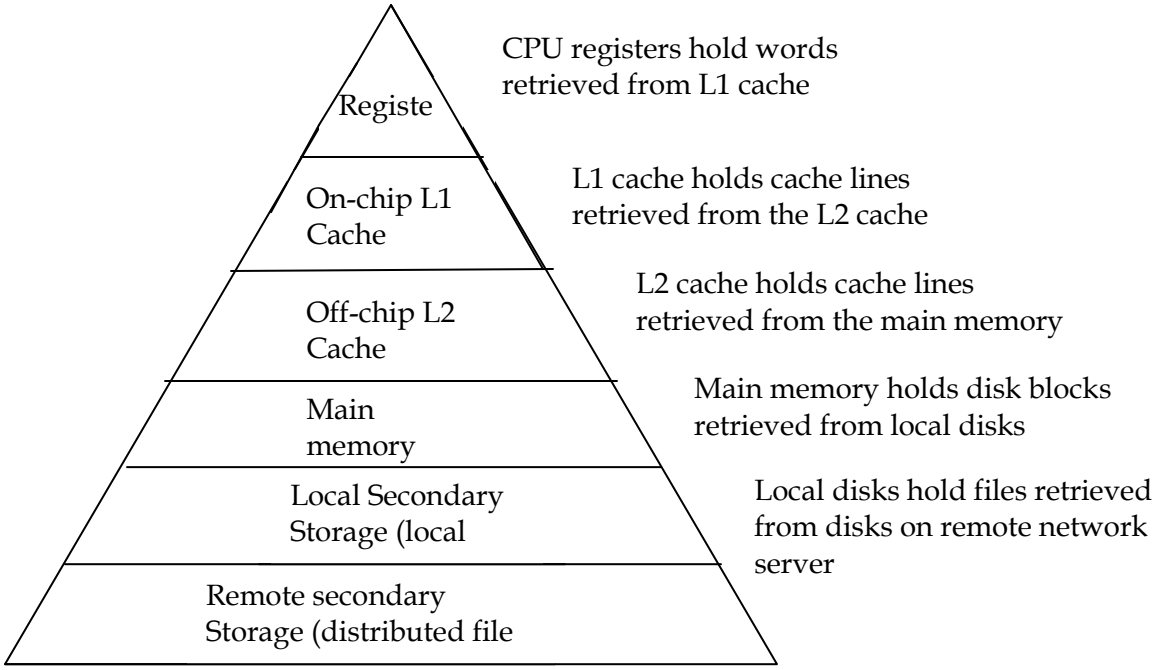


Figure 1.1: Computer Memory Hierarchy [27]

Figure1.1Computer Memory Hierarchy shows the hierarchy of memories used in a computer system with their speed and memory capacity. The arrangement of memory devices in a computer system is such that faster memory is at top level and slower memory is at the bottom. Overall performance of computer system depends upon management and organization of such memories. All the memory management policies are automatically handled by OS and devices are arranged according as the principles followed by it. Different types of memories available up to now can be categorized into two major groups. They are primary memory and secondary memory which can be taken as real memory. Besides real memory OS uses virtual memory to speed up the overall performance of the computer system.

1.1.2 Virtual Memory and Paging

Virtual memory acts as a cache between main memory and secondary memory. Data is fetched in advance from the secondary memory (hard disk) into the main memory so that data is already available in the main memory when needed. The benefit is that the large access delays in reading data from hard disk are avoided. Fotheringham 1961 [1], devised a concept of virtual memory which is associated with ability to address a memory space much larger than the available real memory. Virtual memory [16, 20] is a service provided by an OS that allows execution of programs larger than available physical memory. Virtual memory plays vital role to overcome limited primary memory. Handling virtual memory is one of the important issues of today's computer system.

Virtual memory systems use a technique called paging [9, 17] in which each program is divided into a number of blocks called pages and, the main memory is also divided into a number of block called frames. Generally, page and frame sizes are equal. Program execution begins after loading only a few pages, including zero, in memory. We say that page fault has occurred, if the execution of a program references a page that is not currently in main memory. The program or data page that is not currently in main memory is required to be brought into memory as needed and, some other page should be removed from memory to allocate the space for incoming page. Generally memory is fully allocated to increase degree of multiprogramming.

A major reason for success of virtual memory is that it works silently and automatically, without any intervention from the application programmer. In order to manage memory more efficiently and with fewer errors, modern systems provide an abstraction of main memory known as virtual memory (VM). Virtual memory is an elegant interaction of hardware exceptions, hardware address translation, main memory, disk files, and kernel software that provides each process with a large, uniform, and private address space. With one clean mechanism, virtual memory provides three important capabilities.

a. It uses main memory efficiently by treating it as a cache for an address space stored on disk, keeping only the active areas in main memory, and transferring data back and forth between disk and memory as needed.
b. It simplifies memory management by providing each process with a uniform address space.
c. It protects the address space of each process from corruption by other processes.

Generally memory is fully allocated to increase degree of multiprogramming.

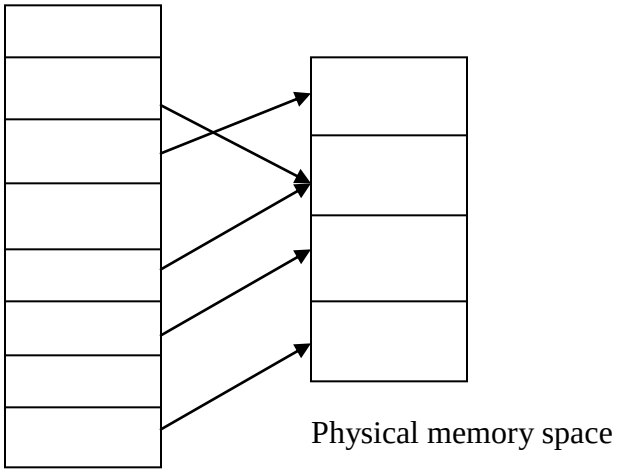


Fig 1.2 Mapping virtual memory and physical memory

The process of choosing a page frame to replace, when a page fault occurs, is called page replacement and, the page frame chosen for the replacement is called victim frame. The

algorithm used by the operating system to find the victim frame is called page replacement algorithm.

1.1.3 Page Replacement Algorithms

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. If the page to be removed is modified while in memory, it must be rewritten to the disk so that the disk copy remains up to date. If, however, the page has not been changed (e.g., it contains program text), the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted. While it would be possible to pick a random page to evict at each page fault, system performance is much better if a page that is not heavily used is chosen. If a heavily used page is removed, it will probably have to be brought back in quickly, resulting in extra overhead. Much work has been done on the subject of page replacement algorithms [17], both theoretical and experimental.

There are many algorithms devised for page replacement. Because the secondary storage, where the remaining pages are stored, has a low speed as compared to the speed of the main memory; the operating system in the least recently used (LRU) algorithm, tries to replace those blocks that have low probability of being referenced again and, it tries to retain those blocks which have high probability of being referenced in near future.

"Locality of reference" [4] is one such property of page reference pattern, which is used by many algorithms to predict about the future references. We say that a workload (sequence of page references) consists of locality of reference if many memory references are accesses to neighboring page of the page referenced just before it. A good approximation to the optimal algorithm is based on the observation that pages that have been used heavily in the recent past would probably be used again in the near future. Conversely, those pages which have not been used heavily would probably remain unused in next few references. This idea can be directly exploited in a simple algorithm: when a page fault occurs, take out the page that has been unused for the longest time. This strategy is called LRU (Least Recently Used) algorithm. In other words, LRU predicts about the future by looking

at the past behaviors. LRU, thus, uses the "recency" factor which means that most recent pages are eligible to remain in memory and least recent page is "thrown out". Clearly, LRU uses the "locality of reference" as a guiding principle.

If a program has a good locality of reference, LRU performs as an excellent algorithm. But if a program has a weak locality, it is worse than other simple algorithms. For such workloads (i.e. having weak locality) LRU shows some anomalous behaviors. The typical cases where weak locality of page references occur include file scanning, regular accesses over more block than the memory size, accesses on blocks with distinct frequency.

1.1.3.1 Local vs Global Page Replacement

Replacement algorithm can be local or global [20]: when a process incurs a page fault, local page replacement algorithm selects for replacement some page that belongs to that process (or a group of processes sharing memory partition), whereas global replacement algorithm is free to select any page in memory. Local page replacement assumes some form of memory partitioning that determines how many pages are to be assigned to a given process of a group of processes. Most popular forms of partitioning are fixed partitioning and balance set algorithm based on the working set model. Advantage of local page replacement is its scalability: each process can handle its page faults independently without contending for some shared global data structure.

Global page replacement involves competition between processes as there are a limited number of frames. This results in more page faults. In local page replacement, there is the principal of locality where the pages are based on the most frequently used, least frequently used, or some other 'prediction' policies which imply that certain pages are more likely to be used than others.

1.1.3.2 Static vs Dynamic Page Replacement Algorithm

Static page replacement algorithms [14] share frames equally among processes. It splits m frames to n users such that each user gets m/n frames. For example, if we have 100 frames and 5 processes then each process will get 200 frames. But, some applications require more frames than others. Compare a database program of 127k and a small student process of 10k. One solution to this problem is to decide the number of frames at initial load time according to program size, or priority.

Dynamic paging algorithms share frames according to needs rather than equally. Some processes need more frames than others and sometimes a process needs more frames than other times. Change of locality when change of function. Some localities require more pages and Change of localities requires more pages. These issues can be addressed with dynamic page replacement algorithm. Although it is apparent that Dynamic Algorithms are more versatile in their ability to deal with locality changes and the natural occurrence of working page set changes, their complexity makes them a reality only for large-scale systems. When working with smaller systems, approximations of Static Algorithms such as Least Recently Used (LRU) or Least Frequently Used (LFU) tend to yield the best performance while dealing with limited hardware support for additional functions [10].

1.1.3.3 Stack Algorithms

One would naturally expect the behavior of static paging algorithms to be linear; after all, they are static in nature. Instinct tells us that by increasing the available physical memory for storing pages, and thus decreasing the needed number of page replacements, that the performance of the algorithm would increase. However, with most simple algorithms this is not necessarily the case. In fact, by increasing the available physical memory, some algorithms such as FIFO can decrease in page fault performance seemingly at random, as evidenced by [1]. This occurrence is referred to as Belady's Anomaly [22], and is a primary factor in considering the practicality of any static algorithm [16, 25]. The predictable change in performance with an increase in physical memory is obviously not something to be taken for granted. It can be proven, however, that if any algorithm with allocation of size m has pages that are guaranteed to be a subset of the allocation m + 1, it will not be subject to Belady's Anomaly; this is what is referred to as the inclusion property. Static algorithms that meet this requirement are called Stack Algorithms, named rightly so for the process of stacking subsets of pages as available allocations increase. Not only are Stack Algorithms more useful, since they are guaranteed not to degrade in performance as available resources increase, their page faulting behavior is also easy to predict. Examples of Stack Algorithms include LRU and LFU, which are among the minority of algorithms not subject to Belady's Anomaly [20].

1.1.4 Performance Metrics

If the requested block is available then hit occurs. If it doesn't then page fault occurs which can be taken as occurrence of miss. Performance gain can be achieved due to more hit rather than miss. For each miss OS has to pay miss penalty which is time consuming and need more resource. Offline performance of the page replacement algorithm is measured in terms of page fault count, hit ratio, miss ratio etc [4].

1.1.4.1 Page Fault Count

A successful page replacement algorithm always computes less number of page faults. Page fault count can be measured by counting occurrence of number of page faults between some intervals of reference, which is also known as page fault frequency (PFF).

1.1.4.2 Miss Rate & Miss Ratio

Miss rate (MR) can be calculated by using formula

$$MR = 100 \times ((NPF - NDP) / (NR - NDP))$$

Where NPF is the number of page faults, NDP is the number of distinct pages referenced and NR is the total number of pages referenced [17]. Miss Ratio is the fraction number of page fault and reference ignoring the distinct references.

1.1.4.3 Hit Rate & Hit Ratio

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.

1.1.5 Program Behavior

There are several factors that influence performance of page replacement algorithm. The performance of page 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 and locality of reference.

1.1.5.1 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. But some follows strongly and some follows weakly [6].

1.1.5.2 Memory Reference Pattern

Memory locations that are referenced repeatedly in a same order can be viewed as cyclic pattern. Loop generates cyclic pattern. For example if P1,P2,P3 be the memory blocks used then cyclic pattern can be taken as P1,P2,P3, P1,P2,P3, P1,P2,P3, P1,P2,P3, P1,P2,P3 when loop executes five times.

Access of memory location at particular place then repeated after some duration, such memory reference pattern can be viewed as correlated pattern. For example if p1,p2,p3 be the memory blocks frequently used then correlated pattern can be taken as if p1,p2,p3, p4,p5,p6,p7,p8,p9p10,p1,p2,p3, p11,p12,p13,p14 when two times correlated access is performed.

When particular memory block has a stationary reference probability and all other blocks are accessed independently with the associated probabilities, such memory reference pattern can be viewed as probabilistic pattern [17]. For example if p1,p2 be the memory blocks frequently used then probabilistic pattern can be taken as p1,p2, p3,p4,p5,p6,p1, p7, p2, p8,p9,p10,p11, p1, p12,p13,p2, p14,p15, p1.

A temporally clustered reference pattern has the property that a block referenced more recently will be referenced sooner in the future. For example temporally clustered pattern can be taken as p1, p2, p1, p3, p2, p4, p3, p1, p2, p5, p6, p5, p6, p9.

Mixed pattern is formed by the combination of cyclic pattern, correlated pattern, temporally clustered pattern and probabilistic pattern. For example of mixed pattern can be taken as p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11, p1,p2,p3,p4,p5,p1, p8, p2, p12,p13, p1, p14,p15,p10, p1, p2, p3, p4.

1.2 Introduction

Among variety of page replacement algorithm Least Recently Used (LRU) algorithm is simple, flexible and has low overhead. LRU replaces page that is not accessed for longest time. LRU adapts faster during change in working set with workloads having good locality of reference. But LRU makes bold assumption on recency factor only which made LRU miss behave with weak locality workloads. Research and experience have shown that CLOCK [3] 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. In CLOCK 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.

LIRS page replacement algorithm is one of the modified versions of LRU. This algorithm identifies and eradicates the misbehaviors of LRU on weak locality of references 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.

CLOCK-Pro [18] takes the same principle as that of LIRS [28] - (it uses the reuse distance (called IRR) rather than recency in its replacement decision) based on CLOCK infrastructure. Generally in various replacements algorithms even in LIRS the movement of pages needed even the page hit occur but in CLOCK-Pro in this situation movement of pages never take place. Here pages categorized into two groups: cold pages and hot pages based on their reuse distances (or IRR). When a page is accessed, the reuse distance is the

period of time in terms of the number of other distinct pages accessed since its last access. Although there is a reuse distance between any two consecutive references to a page, only the most current distance is relevant in the replacement decision. This algorithm uses the reuse distance of a page at the time of its access to categorize it either as a cold page if it has a large reuse distance, or as a hot page if it has a small reuse distance. Then mark its status as being cold or hot. Also place all the accessed pages, either hot or cold, into one single list in the order of their accesses.

In the list, the pages with small recencies are at the list head, and the pages with large recencies are at the list tail. To give the cold pages a chance to compete with the hot pages and to ensure their cold/hot statuses accurately reflect their current access behavior, CLOCK-Pro grant a cold page a test period once it is accepted into the list. Then, if it is re-accessed during its test period, the cold page turns into a hot page. If the cold page passes the test period without a re-access, it will leave the list. Note that the cold page in its test period can be replaced out of memory; however, its page metadata remains in the list for the test purpose until the end of the test period or being re-accessed. When it is necessary to generate a free space, this algorithm replaces a resident cold page. The key question here is how to set the time of the test period. When a cold page is in the list and there is still at least one hot page after it (i.e., with a larger recency), it should turn into a hot page if it is accessed, because it has a new reuse distance smaller than the hot page(s) after it. Accordingly, the hot page with the largest recency should turn into a cold page. So the test period should be set as the largest recency of the hot pages. If we make sure that the hot page with the largest recency is always at the list tail, and all the cold pages that pass this hot page terminate their test periods, then the test period of a cold page is equal to the time before it passes the tail of the list. So all the non-resident cold pages can be removed from the list right after they reach the tail of the list.

There are three hands: Hand-hot for hot pages, Hand-cold for cold pages and Hand-test for running a reuse distance test for a block. The allocation of memory pages between hot pages (Mhot) and cold pages (Mcold) are adaptively adjusted. (M=Mhot+Mcold). Here all hot pages are resident; some cold pages are also resident and also keep track of recently replaced pages.

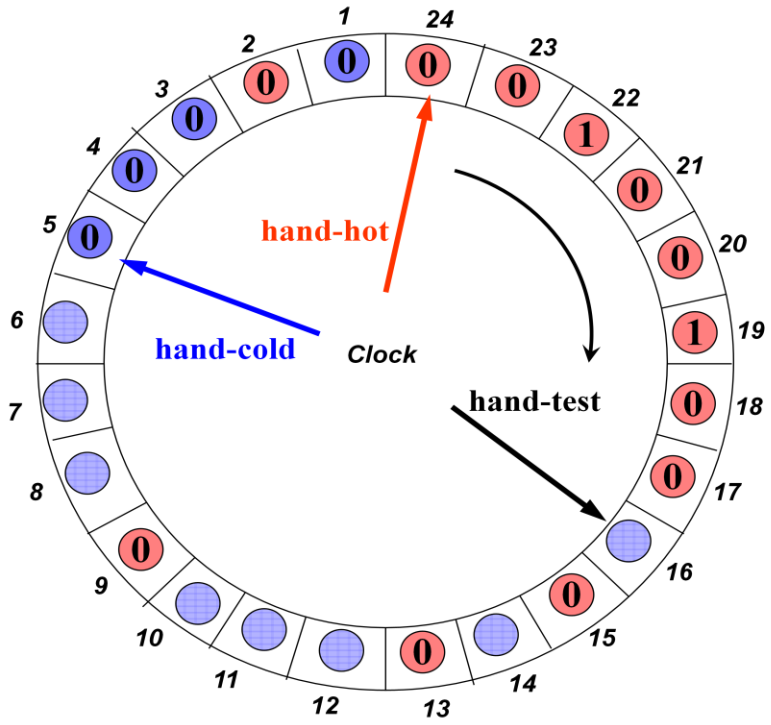


Fig 1.3 Three types of pages in CLOCK-Pro [18]

In CLOCK-Pro if memory size for cold pages (m_c) large then it behaves more like CLOCK. Also if m_c is small then the cold page would have to be replaced out of memory soon after its being loaded in. Adaptive CLOCK-Pro makes the memory allocation for hot and cold pages dynamic.

1.2.1 Problem Statement

CLOCK-Pro replacement algorithm divides memory into two logical partitions: memory for hot pages block (m_h) and memory for cold pages block (m_c). Deciding the size of m_h and m_c is one of the crucial issues of CLOCK-Pro. In addition there are some attempts to make the size of m_h and m_c dynamic (i.e. Adaptive CLOCK-Pro). This Dissertation work focus on analyzing best cache partition ratio for CLOCK-Pro algorithm and also studying the impact of Adaptive CLOCK-Pro on hit ratio.

1.2.2 Objectives

The objectives of this thesis work are

a. To analyze CLOCK-Pro by using different sized resident cold pages blocks.
b. To study performance impact of “Adaptive CLOCK-Pro” over CLOCK-Pro with Fixed hot and cold size.
c. To compare CLOCK, CLOCK-Pro and Adaptive Clock-Pro” page replacement algorithms.

1.3 Motivation

Memory management is not only the burden of today's computing devices. It has been researched for decades. Whatever variety of storage devices found in today's market is the great achievement of computer science. But still computer memory is the limited source which directly hampers the performance of computing system. Performance gain can be achieved by increasing the capacity of primary storage. Expectation of customer is to decrease cost price with sufficient working memory. Hence to fulfill this demand for manufacturing such device fewer materials are used and size of memory is being decreased. But rather than this technical view, it is not possible to gain performance without managing memory logically for its usability. Varieties of techniques had been tried for this achievement. Among such techniques paging is the successful one. Page replacement algorithm is the main part of paging technique because deciding the victim page is a very tough job. Optimal page replacement algorithm is the best one. But it can be only simulated since references should be known earlier, which is not possible in most of the real systems. Many near-optimal replacement schemes have been found, but their complexity and various practical considerations tend to limit the effectiveness of the algorithms implemented in real systems.

Implementing LRU is a successful idea due to its simplicity, flexibility and performance gain. But still LRU shows anomalous behavior with weak locality workloads. It is better if an algorithm could work as LRU comparatively equivalent to computational complexity as well as it could solve the problem on weak locality workloads. Research and experience have shown that CLOCK [3] 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. Reading related research papers it

is found that CLOCK-Pro can fulfill these criteria. It is successfully implemented in different fields [18]. It is better if CLOCK-Pro could store deeper history information. CLOCK-Pro can be implemented in a different approach based on its principle.

1.4 Thesis Organization

Background part of this dissertation work focuses on page replacement algorithm and the related basic terms which are already mentioned above along with an introduction to CLOCK-Pro. Some more chapters are remaining which clarifies the topics CLOCK-Pro 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 like Optimal, LRU, MRU, LRFU, 2Q, CLOCK, CAR, CART, etc within their category. This chapter also contains the research methodology part which shows the flow of our research. Chapter 3 consists of program development steps of our simulation. 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 generating traces of memory references that shows trace driven input, output results with several analyzing graphs which are only tested for weak locality workloads. Chapter 6 consists of conclusion of this whole dissertation work and the future work which shows guidelines for further research.

CHAPTER 2

Literature Review and Methodology

2.1 Literature Review

2.1.1 OPT Page Replacement Algorithm

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 based replacement algorithms.

2.1.2 LRU Based Page Replacement Algorithms

2.1.2.1 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 [7].

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 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].

2.1.2.3 MRU Page Replacement Algorithm

Most Recently Used (MRU) algorithm [10, 15] 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.

2.1.2.4 LFU Page Replacement Algorithm

Often confused with LRU, Least Frequently Used (LFU) [10] 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 [1, 15]. 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.

2.1.2.5 LRFU Page Replacement Algorithm

Having analyzed the advantages and disadvantages of LRU and LFU, A new algorithm LRFU is proposed by combining them through weighing block recency and frequency factors. The performance of the LRFU algorithm largely relies on a parameter called (λ), which determines the relative weight of LRU or LFU and has to be adjusted according to the system configurations, even according to different workloads [15].

2.1.2.6 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 K^{th} 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.

2.1.2.7 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 [19] 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 K_{out} 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.

2.1.2.8 LIRS Page Replacement Algorithm

LIRS 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 L_{lirs} is used to store LIR blocks, and the minor part with the size of L_{hirs} is used to store blocks from HIR block set, where $L_{lirs} + L_{hirs} = L$. 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 L_{hirs} 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, but it uses the access recency information to predict which pages have more probability to be accessed in a near future [21, 28].

2.1.2.9 ARC Page Replacement Algorithm

Adaptive Replacement Cache (ARC) [23] 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.

2.1.3 CLOCK Based Page Replacement Algorithms

2.1.3.1 CLOCK Page Replacement Algorithm

Research and experience have shown that CLOCK [18] 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.

2.1.3.2 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 [18]. 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). When no pre-paging is involved, the algorithm is abbreviated CLOCK (i). The algorithm used in MULTICS and CP-67 is CLOCK (1). So, CLOCK is GCLOCK (1).

2.1.3.3 CAR Page Replacement Algorithm

Another CLOCK based algorithm is CAR [26] (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.

2.1.3.4 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 [26] 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.

2.1.3.5 CLOCK-Pro Page Replacement Algorithm

Another important algorithm is CLOCK-Pro which is already described in section 1.2. Its objective is to minimize the fault rate in weak locality of references 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

This algorithm is already described in section 1.2. 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.

2.2 Research Methodology

The system of collecting data for research projects is known as **research methodology**. The data may be collected for either theoretical or practical research for example management research may be strategically conceptualized along with operational planning methods and change management [24]. Some important factors in **research methodology** include validity of research data.

The topics memory management and design has been studied from the early generation of computer. Page replacement algorithm is one of the major strategies to manage memory efficiently. All data collected are primary data, which are traces of page references. This dissertation work is based on trace driven simulation. Output information gathered is analyzed in a quantitative approach. Finally conclusion is drawn with the help of analyzed data which is not the generalized form. This work is only specialized for weak locality as well as not purely weak locality of workloads having probabilistic pattern, cyclic patterns, temporally clustered pattern and mixed patterns.

CHAPTER 3

Page Replacement Algorithms carried out in Dissertation work

3.1. CLOCK Page Replacement

In CLOCK, the memory spaces holding the pages can be regarded as a circular buffer. In CLOCK 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.

When a page fault occurs, the page being pointed to by the hand is inspected. If its *R* bit is 0, the page is evicted, the new page is inserted into the clock in its place, and the hand is advanced one position. If *R* is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with *R* = 0. Not surprisingly, this algorithm is called **clock**.

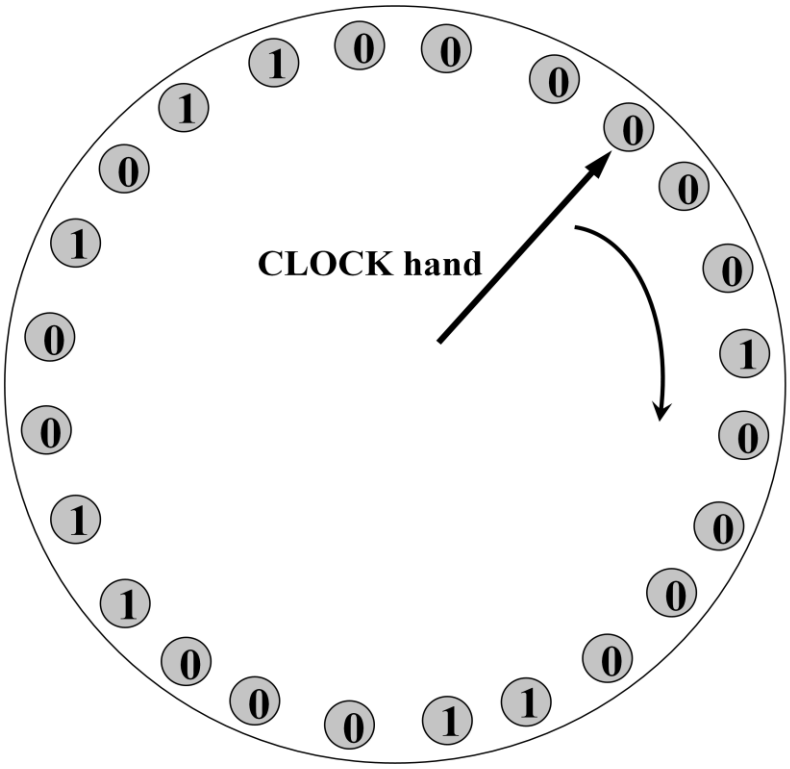


Fig 3.1 General CLOCK

3.1.1. CLOCK Algorithm

1. Begin
2. Read new page, say P
3. If P is available in CLOCK (Circular Linked List).
   3.1.Page hit occurs. then
   3.2.Turn reference bit to 1 and do nothing else.
4. else
   3.1. Page miss occurs then
   3.2. If its *R* bit is 0, the page is evicted, the new page is inserted into the clock in its place, and the hand is advanced one position.
   3.3. If *R* is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with $R = 0$.
5. Stop

3.1.2. CLOCK Tracing

Input References: 1 2 3 4 3 1 2 1 5 4

Size of Memory: 3

Total Number of References: 10

Number of Distinct References: 5

Upon accessing 1:

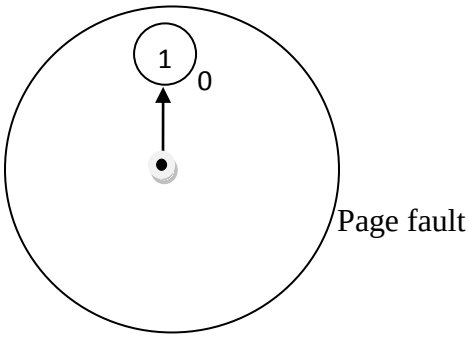


Fig 3.2 pages in CLOCK at virtual time 1

Upon accessing 2:

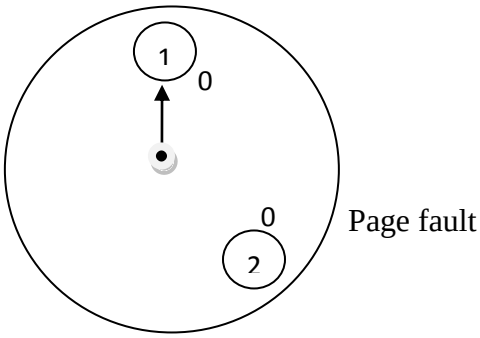
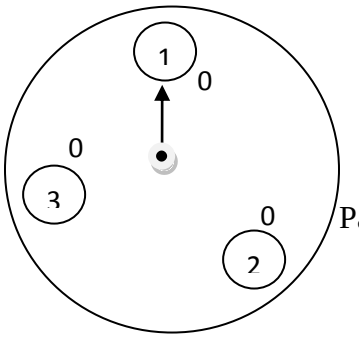


Fig 3.3 pages in CLOCK at virtual time 2

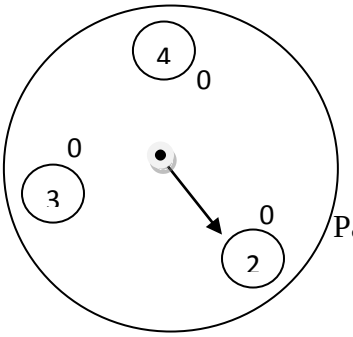
Upon accessing 3:



Page fault

Fig 3.4 pages in CLOCK at virtual time 3

Upon accessing 4:



Page fault

Fig3.5 pages in CLOCK at virtual time 4

Upon accessing 3:

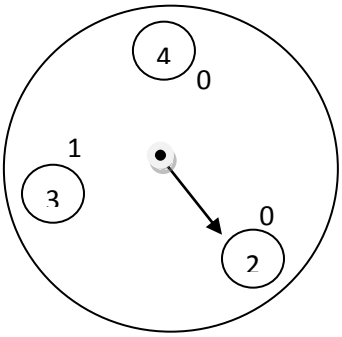
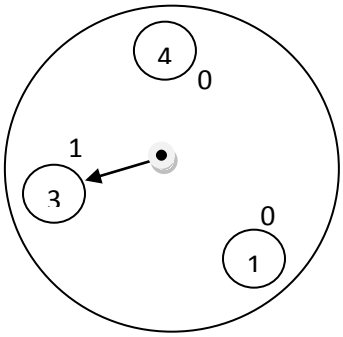


Fig 3.6 pages in CLOCK at virtual time 5

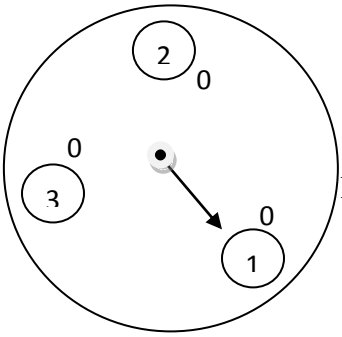
Upon accessing 1:



Page fault

Fig 3.7 pages in CLOCK at virtual time 6

Upon accessing 2:



Page fault

Fig 3.8 pages in CLOCK at virtual time 7

Upon accessing 1:

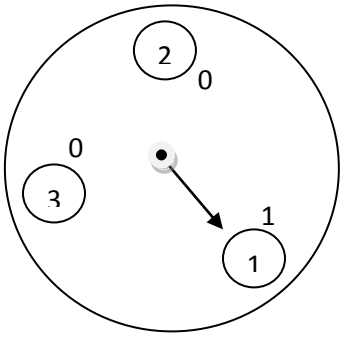


Fig 3.9 pages in CLOCK at virtual time 8

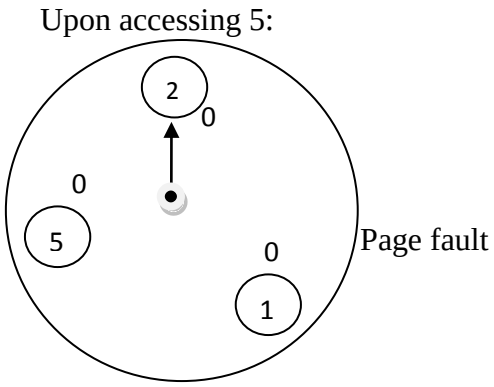


Fig 3.10 pages in CLOCK at virtual time 9

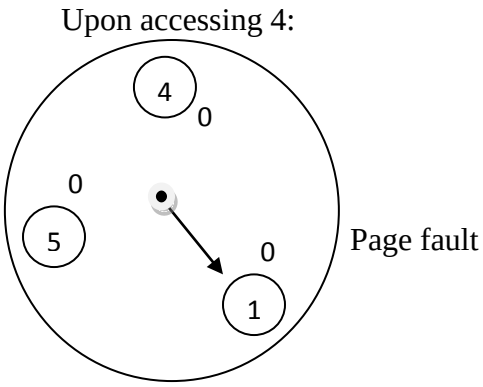


Fig 3.11 pages in CLOCK at virtual time 10

Total number of page faults: 8

3.2. CLOCK-Pro Page Replacement

CLOCK-Pro takes the same principle as that of LIRS [28]- (it uses the reuse distance (called IRR) rather than recency in its replacement decision) based on CLOCK infrastructure. Generally in various replacements algorithms even in LIRS the movement of pages needed even the page hit occur but in CLOCK-Pro in this situation movement of pages never take place. Here pages categorized into two groups: cold pages and hot pages based on their reuse distances (or IRR). When a page is accessed, the reuse distance is the period of time in terms of the number of other distinct pages accessed since its last access. Although there is a reuse distance between any two consecutive references to a page, only the most current distance is relevant in the replacement decision. This algorithm uses the reuse distance of a page at the time of its access to categorize it either as a cold page if it has a large reuse distance, or as a hot page if it has a small reuse distance. Then mark its status as being cold or hot. Also place all the accessed pages, either hot or cold, into one single list in the order of their accesses.

In the list, the pages with small recencies are at the list head, and the pages with large recencies are at the list tail. To give the cold pages a chance to compete with the hot pages and to ensure their cold/hot statuses accurately reflect their current access behavior, CLOCK-Pro grant a cold

page a test period once it is accepted into the list. Then, if it is re-accessed during its test period, the cold page turns into a hot page. If the cold page passes the test period without a re-access, it will leave the list. Note that the cold page in its test period can be replaced out of memory; however, its page metadata remains in the list for the test purpose until the end of the test period or being re-accessed. When it is necessary to generate a free space, this algorithm replaces a resident cold page. The key question here is how to set the time of the test period. When a cold page is in the list and there is still at least one hot page after it (i.e., with a larger recency), it should turn into a hot page if it is accessed, because it has a new reuse distance smaller than the hot page(s) after it. Accordingly, the hot page with the largest recency should turn into a cold page. So the test period should be set as the largest recency of the hot pages. If we make sure that the hot page with the largest recency is always at the list tail, and all the cold pages that pass this hot page terminate their test periods, then the test period of a cold page is equal to the time before it passes the tail of the list. So all the non-resident cold pages can be removed from the list right after they reach the tail of the list.

There are three hands: Hand-hot for hot pages, Hand-cold for cold pages and Hand-test for running a reuse distance test for a block. The allocation of memory pages between hot pages (Mhot) and cold pages (Mcold) are adaptively adjusted. (M=Mhot+Mcold). Here all hot pages are resident; some cold pages are also resident and also keep track of recently replaced pages.

3.2.1. CLOCK-Pro algorithm

1. Begin
2. Read new page, say P
3. If P is already in the buffer cache then,
   3.1.Page hit and turn reference bit to 1 and do nothing else
4. else
   4.1. If P is not in Test period (not in non-resident cold page list)
   4.1.1. If hot page size is not full then,
   -Set page P to hot page list
   4.1.2. Else if hot page size is full
   4.1.2.1 If cold page size is not full then
   - Set page P to cold page list

4.1.2.2 Else if cold page size is full then

4.1.2.2.1 If non-resident cold page size is not full

4.1.2.2.1.1 If Hand_cold.R==0 then

-Remove Hand_cold.page from cold page list and set this page to non-residen cold page list.

-Set page P into cold page list with R=0

4.1.2.2.1.2 Else if Hand_cold.R==1 then

-Promote page to hot page list

-insert page P into cold page list.

4.1.2.2.2 Else if non-resident page size is full then

-kick out one page from history information and go to step 4.1.2.2.1.1

4.2 Else if page P is in test period

-Then insert this page P into hot page list by demote some pages from hot page list to cold page list and also demote some pages from cold block list to history page list and kick out necessary pages from history page list.

5. Stop

3.2.2. CLOCK-Pro Tracing

Input References: 1 2 3 4 3 1 2 1 5 4

Total Number of References: 10 Number of Distinct References: 5

Size of Memory: 3 Hot_block_size: 2

Cold_block_size: 1 Cold_non-resident_block_size=3

Total Meta Data =2M=2*3=6

Upon accessing 1:

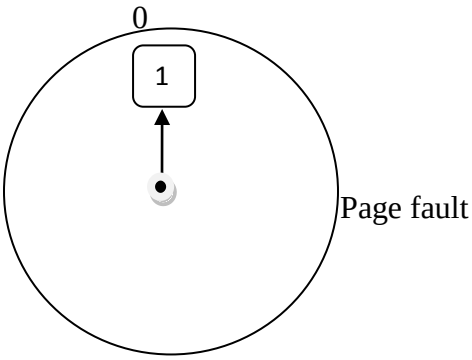


Fig3.12 pages in CLOCK-Pro at virtual time 1

Upon accessing 2:

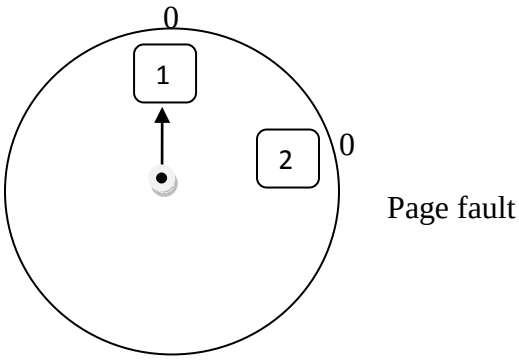


Fig3.13pages in CLOCK-Pro at virtual time 2

Upon accessing 3:

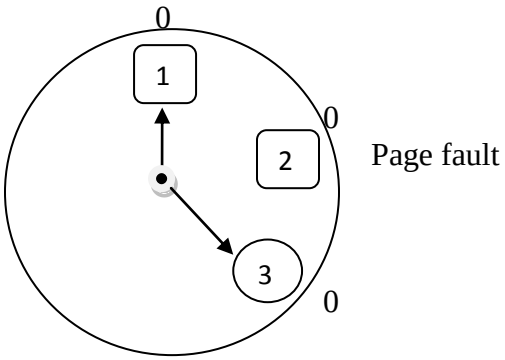


Fig3.14 pages in CLOCK-Pro at virtual time 3

Upon accessing 4:

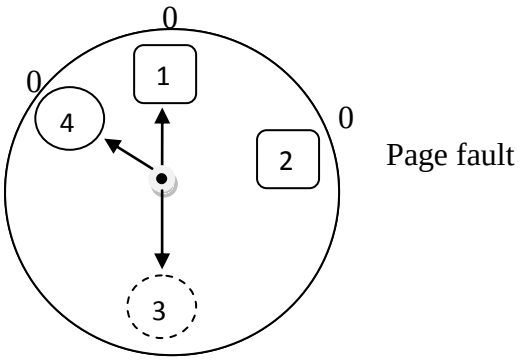


Fig3.15 pages in CLOCK-Pro at virtual time 4

Upon accessing 3:

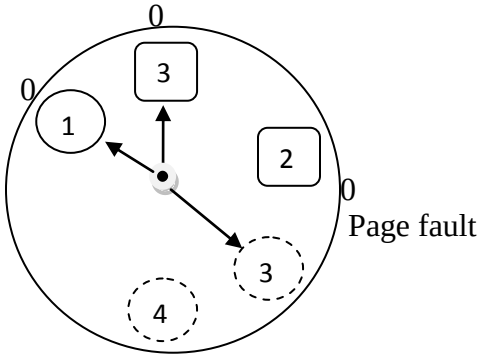


Fig3.16 pages in CLOCK-Pro at virtual time 5

Upon accessing 1:

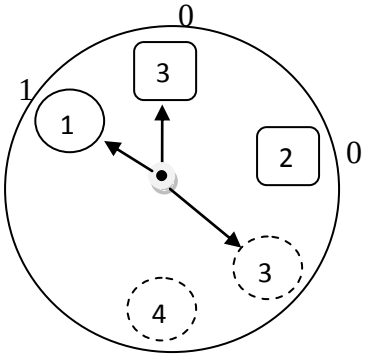


Fig3.17 pages in CLOCK-Pro at virtual time 6

Upon accessing 2:

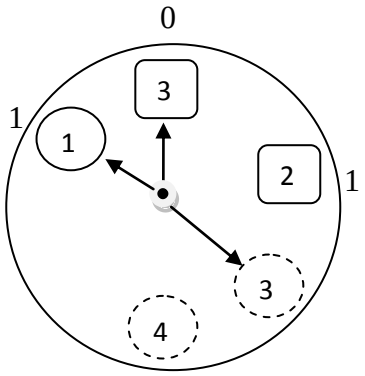


Fig3.18 pages in CLOCK-Pro at virtual time 7

Upon accessing 1:

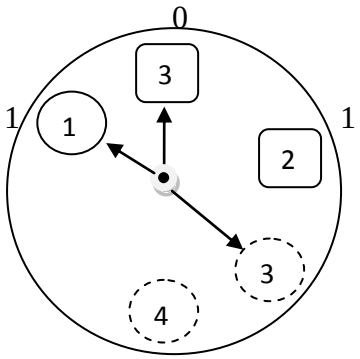
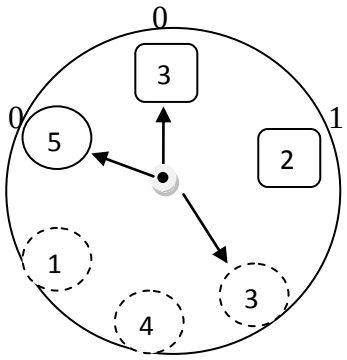


Fig3.19 pages in CLOCK-Pro at virtual time 8

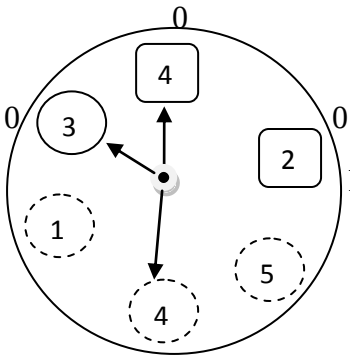
Upon accessing 5:



Page fault

Fig3.20 pages in CLOCK-Pro at virtual time 9

Upon accessing 4:



Page fault

Fig3.21pages in CLOCK-Pro at virtual time 10

Total number of page faults: 7

3.3. Adaptive CLOCK-Pro Page Replacement

Until now, in CLOCK and CLOCL-Pro the memory allocations for the hot and cold pages are fixed. In CLOCK-Pro, resident cold pages are actually managed in the same way as in CLOCK. **HAND$_{cold}$** behaves the same as what the clock hand in CLOCK does: sweeping across the pages while sparing the page with a reference bit of 1 and replacing the page with a reference bit of 0. So increasing the size of the allocation for cold pages, makes CLOCK-Pro behave more like

CLOCK. Let us see the performance implication of changing memory allocation in CLOCK-Pro. To overcome the CLOCK performance disadvantages with weak access patterns such as scan and loop, a small m_c value means a quick eviction of cold pages just faulted in and the strong protection of hot pages from the interference of cold pages. However, for a strong locality access stream, almost all the accessed pages have relatively small reuse distance. But, some of the pages have to be categorized as cold pages. With a small m_c, a cold page would have to be replaced out of memory soon after its being loaded in.

Due to its small reuse distance, the page is probably faulted in the memory again soon after its eviction and treated as a hot page because it is in its test period this time. This actually generates unnecessary misses for the pages with small reuse distances. Increasing m_c would allow these pages to be cached for a longer period of time and make it more possible for them to be re-accessed and to turn into hot pages without being replaced. Thus, they can save additional page faults. For a given reuse distance of an accessed cold page, m_c decides the probability of a page being re-accessed before its being replaced from the memory. For a cold page with its reuse distance larger than its test period, retaining the page in memory with a large m_c is a waste of buffer spaces. On the other hand, for a page with a small reuse distance, retaining the page in memory for a longer period of time with a large m_c would save an additional page fault. In the adaptive CLOCK-Pro [13, 18], adjust current reuse distance distribution dynamically. If a cold page is accessed during its test period, increment cold block size by 1. If a cold page passes its test period without a re-access, then decrement the size of cold page size by 1.

3.3.1 Adaptive CLOCK-Pro algorithm

1 Begin
2 Read new page, say P
3 If P is already in the buffer cache then,
  3.2.Page hit and turn reference bit to 1 and do nothing else
4 else
  4.1. If P is not in Test period (not in non-resident cold page list)
    Then cold_page_size= cold_page_size-1;
  4.1.1. If hot page size is not full then,
    -Set page P to hot page list

4.1.2. Else if hot page size is full

4.1.2.1 If cold page size is not full then

- Set page P to cold page list

4.1.2.2 Else if cold page size is full then

4.1.2.2.1 If non-resident cold page size is not full

4.1.2.2.1.1 If Hand_cold.R==0 then

-Remove Hand_cold.page from cold page list and set this page to non-residen cold page list.

-Set page P into cold page list with R=0

4.1.2.2.1.2 Else if Hand_cold.R==1 then

-Promote page to hot page list

-insert page P into cold page list.

4.1.2.2.2 Else if non-resident page size is full then

-kick out one page from history information and go to step 4.1.2.2.1.1

4.2 Else if page P is in test period

-Then cold_page_size= cold_page_size+1;

-insert this page P into hot page list by demote some pages from hot page list to cold page list and also demote some pages from cold block list to history page list and kick out necessary pages from history page list.

5. Stop

3.3.2 Adaptive CLOCK-Pro Tracing

Input References: 1 2 3 4 3 1 2 1 5 4

Total Number of References: 10 Number of Distinct References: 5

Size of Memory: 3 Hot_block_size: 2

Cold_block_size: 1 Cold_non-resident_block_size=3

Total Meta data =6

Upon accessing 1:

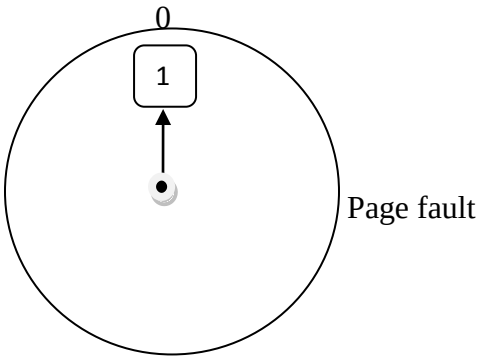


Fig3.22 pages in Adaptive CLOCK-Pro at Virtual time 1

Upon accessing 2:

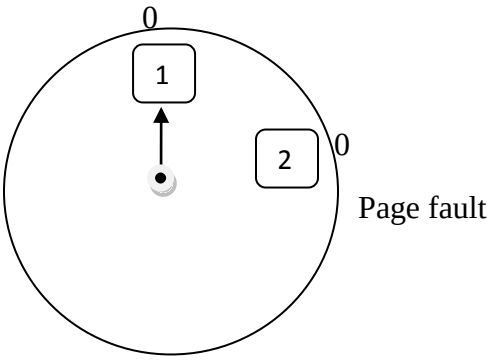


Fig3.23 pages in Adaptive CLOCK-Pro at virtual time 2

Upon accessing 3:

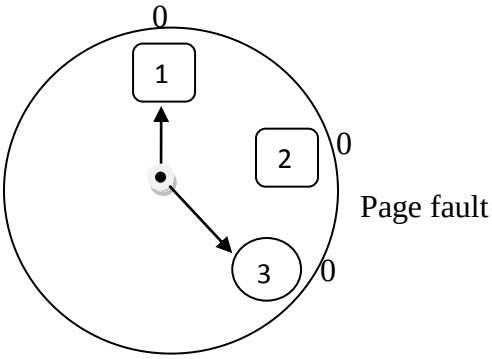


Fig3.24 pages in Adaptive CLOCK-Pro at Virtual time 3

Upon accessing 4:

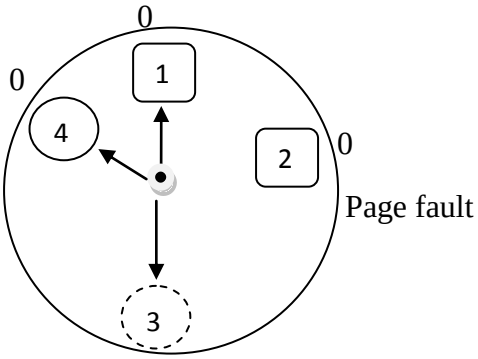
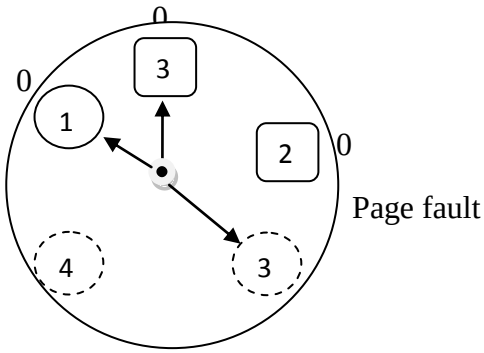
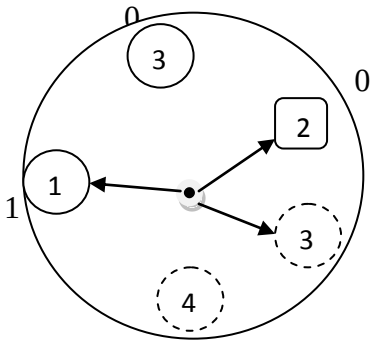


Fig3.25 pages in Adaptive CLOCK-Pro at Virtual time 3

Upon accessing 3:



Upon accessing 1:

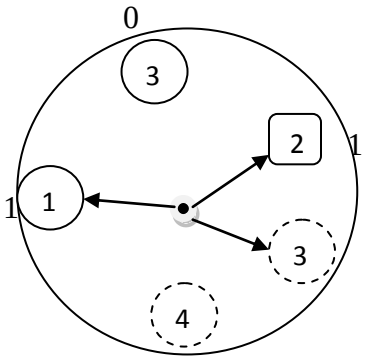


(Since 3 is in test period thus
Cold_block_size= Cold_block_size+1)

Fig3.26 pages in Adaptive CLOCK-Pro at virtual at virtual time 5

Fig3.27 pages in Adaptive CLOCK-Pro at virtual time 6

Upon accessing 2:



Upon accessing 1:

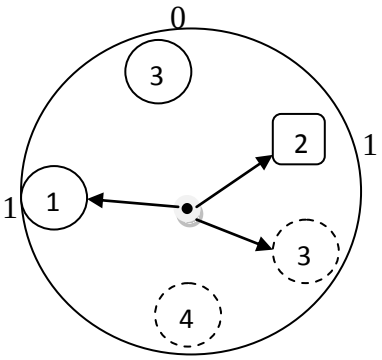
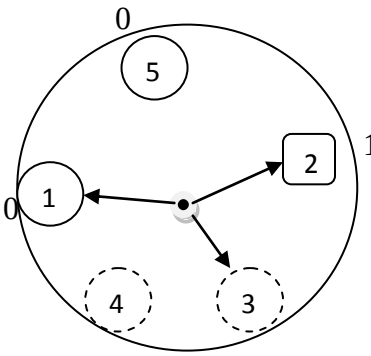


Fig3.28 pages in Adaptive CLOCK-Pro at virtual time 7

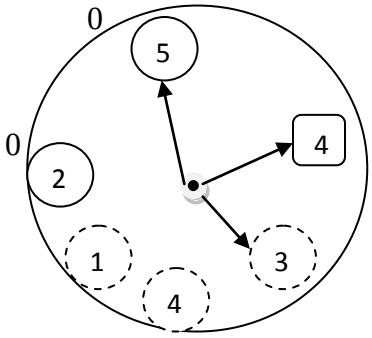
Fig3.29 pages in Adaptive CLOCK-Pro at virtual time 8

Upon accessing 5:



Page fault

Upon accessing 4:



Page fault

Fig 3.30 pages in Adaptive CLOCK-Pro at Virtual time 9

(Cold_block_size= Cold_block_size+1)

Fig3.31 pages in Adaptive CLOCK-Pro at virtual time 10

Total number of page faults=7

CHAPTER 4

Implementation

4.1. Tools used

4.1.1. Programming Language

For the implementation of proposed algorithm Java Programming Language is used. **Java is** a general-purpose, concurrent, class-based, object-oriented computer programming language that is specifically designed to have as few implementation dependencies as possible. One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to platform-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be interpreted by a virtual machine (VM) written specifically for the host hardware. End-users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a Web browser for Java applets.

Java is a robust language. It provides many safeguards to ensure reliable code. It has strict compile time and run time checking for data types. It is designed as a garbage-collected language ease the programmers virtually all memory management problems. Java also incorporates the concepts of exception handling which captures series errors and eliminates any risk of crashing the system.

4.1.2. NetBeans IDE

NetBeans is an 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 application.

4.2. Data Structure

4.2.1. Circular Doubly Linked List (CDLL)

The advantages of both doubly linked list and circular linked list are incorporated into a third list structure known as circular doubly linked list and it is known to be the best of its kind.

A circular list is one in which the last node in the list points to the first node. Circular lists are useful in certain applications where we want to repeatedly go around the list. For example, when multiple applications are running on a PC, it is common for the operating system to put the running applications on a list and then to cycle through them, giving each of them a slice of time to execute, and then making them wait while the CPU is given to another application. It is convenient for the operating system to use a circular list so that when it reaches the end of the list it can cycle around to the front of the list. Circular linked lists also make our implementation easier, because they eliminate the boundary conditions associated with the beginning and end of the list, thus eliminating the special case code required to handle these boundary conditions.

Because in CLOCK, CLOCK-Pro and Adaptive CLOCK-Pro all pages are organized in circular order thus to implement these algorithms efficiently the circular doubly linked list can be used.

Structure of CLOCK Node:

```
package clock_replacement;
public class Clock_node
{
        int R;   //referenc
        int pn;  //page number
        Clock_node next,prev;
        boolean isresident;
        public Clock_node()
        {
```

```
                R=0;
                pn=0;
                isresident=false;
                next=prev=null;
        } }
```

Structure of CLOCK-Pro and Adaptive CLOCK-Pro Node:

```
package CLOCK_Pr_Simulation;
enum PageStatus{non_resident,cold,hot};
public class CLOCK_Pr_Node
{
        private int prt; //reference times
        private int pft; //page fault times
        int pn;//page  number
        boolean isresident;
         boolean isinclock;
        PageStatus status;
        int R;   //referenc
        CLOCK_Pr_Node prev;
        CLOCK_Pr_Node next;
         public CLOCK_Pr_Node()
        {       prt=0;
                pft=0;
                R=0;
                status=PageStatus.non_resident;
                isresident=false;  isinclock=false;
                prev=next=null;  }}
```

4.3 Flowcharts

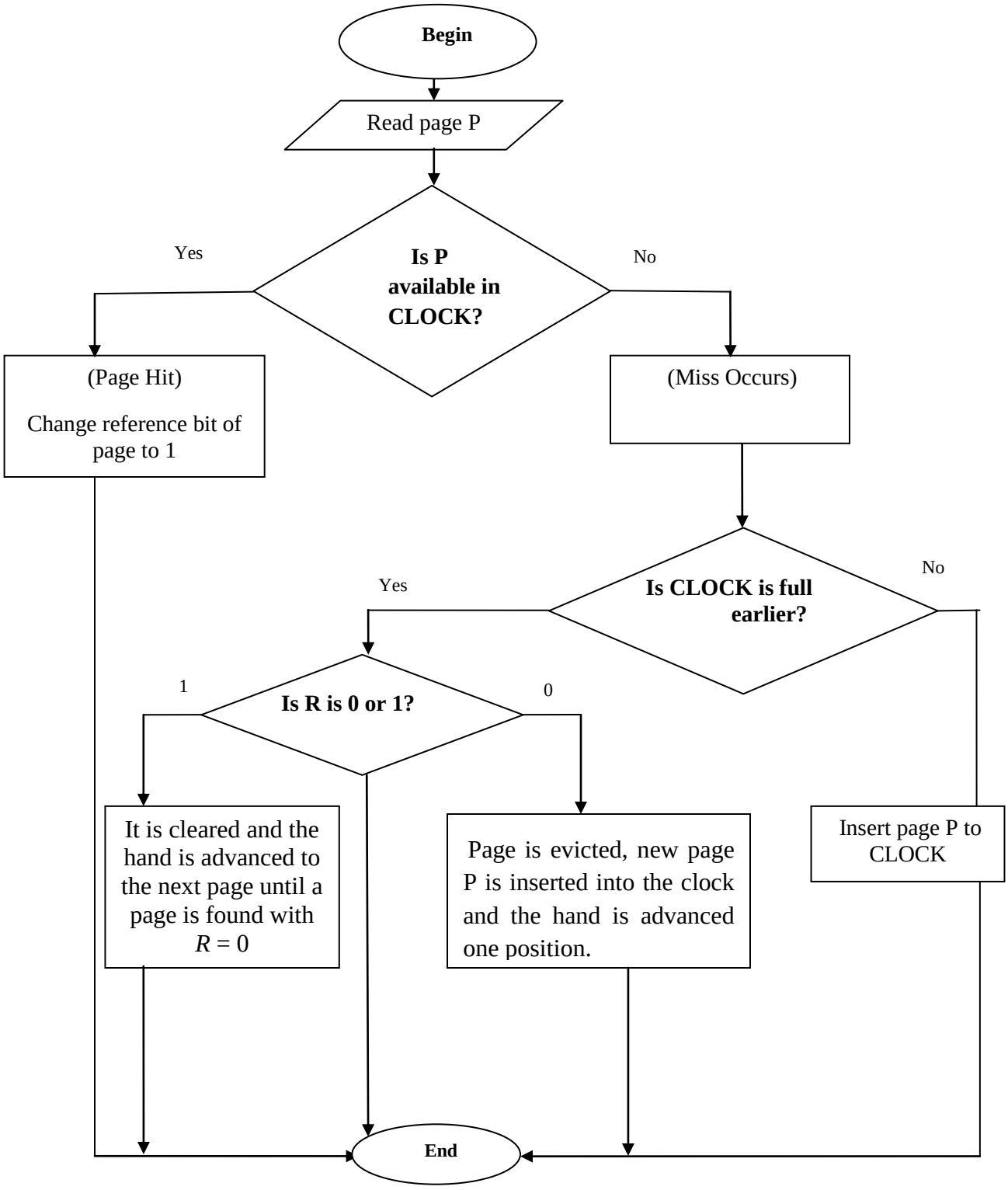


Fig 4.1 Flowchart of CLOCK Algorithm

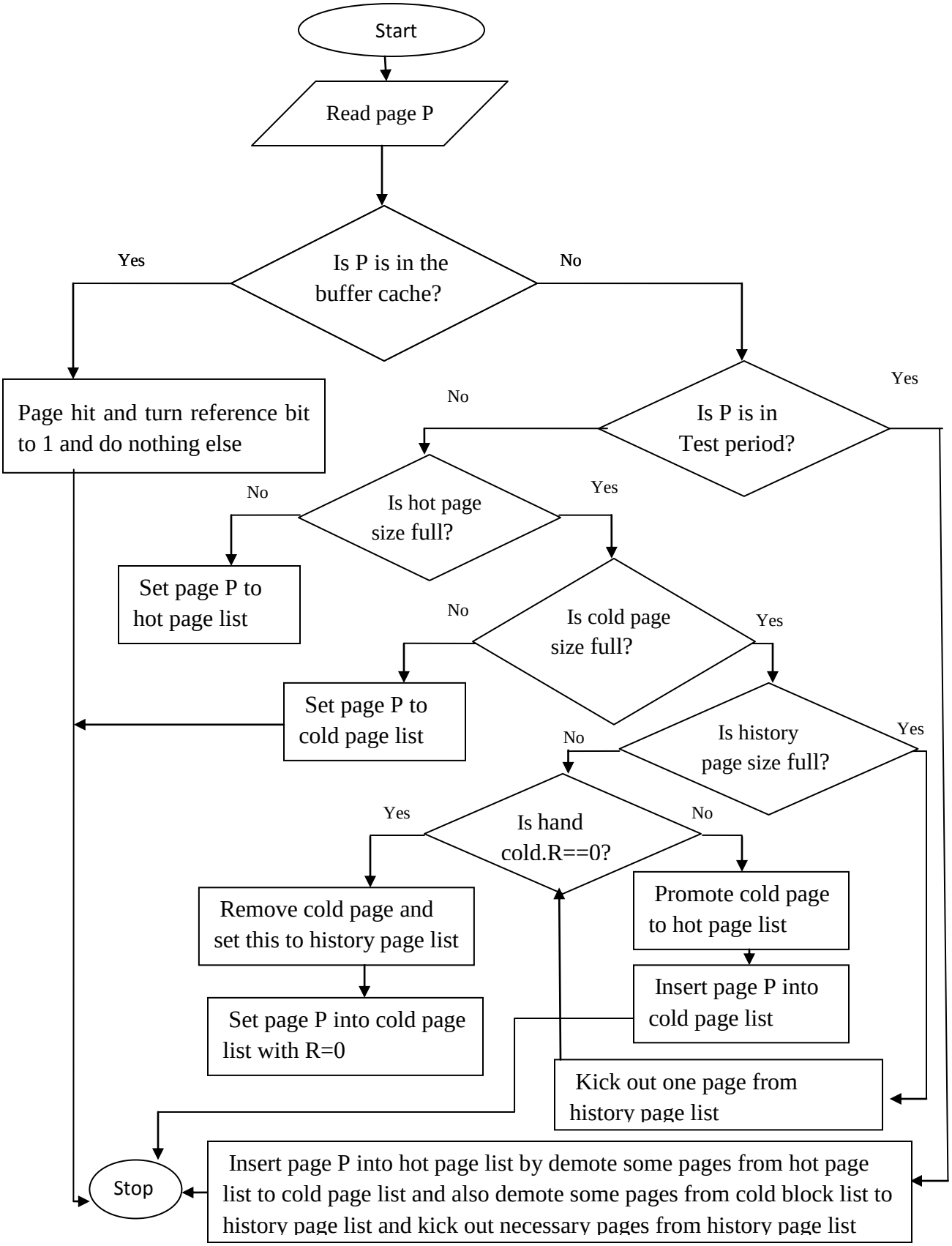


Fig 4.2 Flowchart of CLOCK-Pro Algorithm

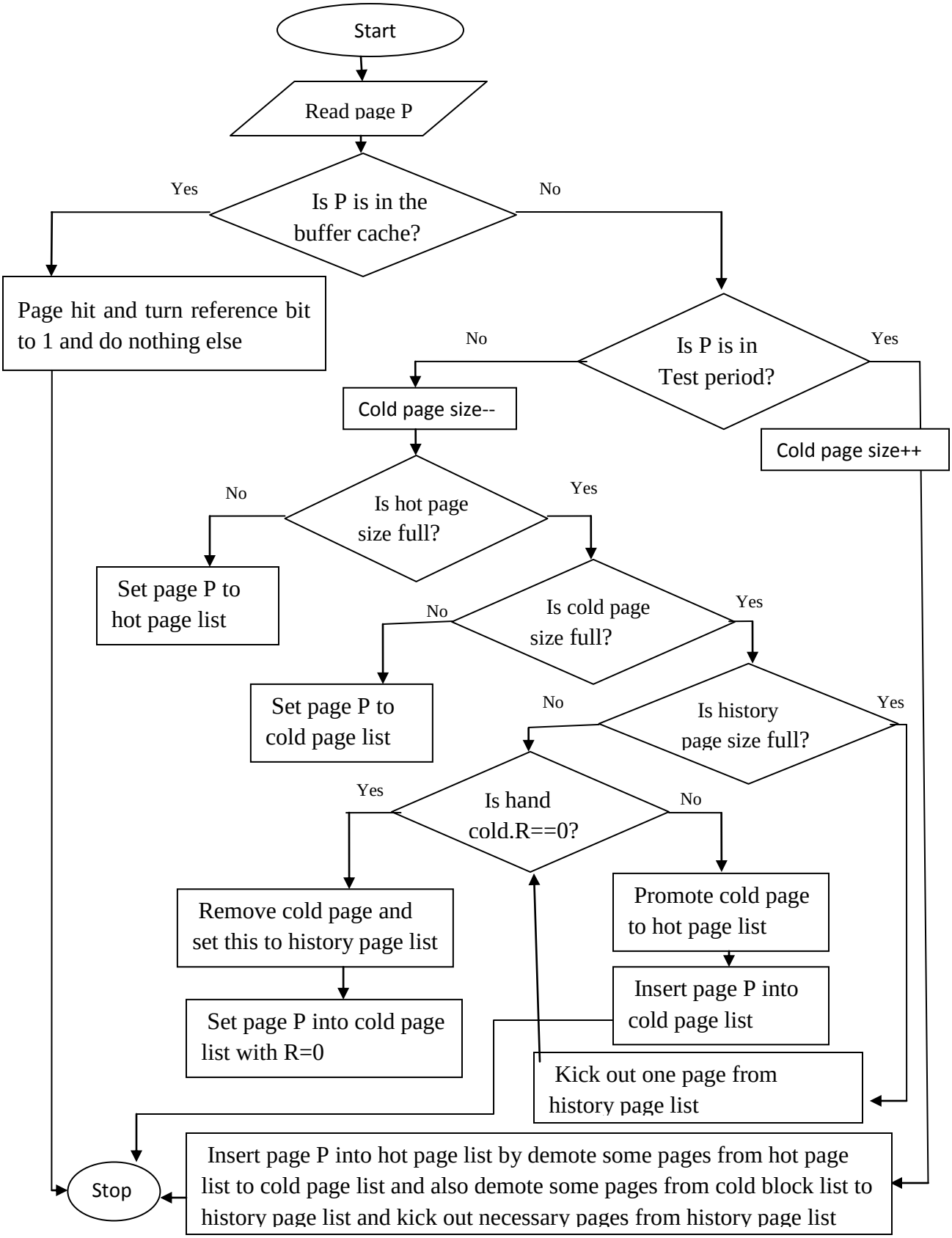


Fig 4.3 Flowchart of Adaptive CLOCK-Pro Algorithm

4.4 Sample Test Case

Temporary-clustered pattern:

0 1 2 3 4 4 5 6 7 8 9 10 11 12 13 14 9 15 16 17 9 18 9 19 9 20 21
22 23 24 25 26 15 16 4 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 5 86 87 88 89 2 3 90 91 9 93 94 95 96 97 98 99 100 8 101 54 59
4 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 102 103 104
105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
121 122 123 124 125 126 127 128 82 83 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 129 129 130 131 132 133 133 133 133 134 134 134 134 135
135 136 136 136 136 136 136 137 137 137 138 138 138 139 139 139
139 139 139 139 139 139 139 140 140 140 140 140 140 141 141 141
141 141 142 142 142 142 143 143 144 144 145 145 145 145 145 146
146 146 146 146 146 146 146 146 146 147 147 147 147 147 148 148
148 149 149 149 149 149 149 149 149 149 150 150 150 150 150 150
151 151 151 152 152 152 153 153 153 154 154 154 154 154 155 155
155 133 133 133 135 135 135 135 135 135 135 135 135 156 156 156
156 156 156 156 156 156 156 156 156 156 157 157 157 158 158 158
158 158 158 158 158 158 159 159 159 159 159 159 159 159 159 160
160 160 160 160 160 161 161 161 161 161 161 161 161 161 161 161
136 136 136 136 162 162 162 163 164 164 164 164 164 164 165 165
165 166 166 166 167 167 167 168 168 168 138 138 138 138 138 138
138 138 138 139 139 139 142 142 142 144 144 144 144 144 169 169
169 145 145 145 170 170 170 171 171 171 172 151 151 173 174 174
159 159 159 160 160 160 136 136 136 170 170 170 171 171 171 175
175 175 160 160 160 134 134 135 134 136 136 136 137 137 138 138
139 139 139 139 139 139 140 140 140 140 141 141 142 142 143 142
143 144 143 144 145 145 145 145 146 146 146 146 146 146 147 146
147 147 148 148 149 149 149 150 150 150 151 151 152 152 153 153
154 154 154 154 155 155 133 135 135 135 135 135 135 156 156 156
156 156 156 156 157 157 158 158 158 158 158 159 159 159 160 159
160 160 160 161 161 161 161 161 161 136 136 163 164 164 165 165

166 166 167 167 168 168 138 138 138 138 138 139 142 144 144 169
169 145 145 170 170 171 171 151 172 174 173 174 159 136 136 170
170 171 171 175 175 133 134 134 135 136 136 138 138 139 139 139
139 140 140 141 142 143 144 145 145 146 146 146 146 147 147 148
149 149 149 150 150 150 151 152 153 154 154 155 133 158 158 160
160 160 161 161 161 162 163 164 164 164 165 166 167 168 138 138
138 139 142 144 144 144 169 145 170 171 151 172 151 174 136 170
171 175 133 134 135 136 136 137 138 139 139 139 139 140 140 141
141 142 143 144 145 145 146 146 146 147 147 148 149 149 149 150
150 151 152 153 154 154 155 133 135 135 135 156 156 156 156 157
158 158 158 159 159 159 160 160 161 161 161 161 136 136 162 164
164 164 165 166 167 168 138 138 138 138 139 142 144 144 169 145
170 171 151 174 159 136 170 171 175 97 98 94 95 84 85 92 93 88 89
90 91 176 177 178 179 48 49 78 79 2 3 86 87 180 180 5 6 7 8 9 10
11 12 13 14 9 15 16 17 9 18 9 19 9 20 21 22 23 24 25 26 15 16 180
181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197 198 199 180 181 182 183 184 185 186 187 188 189 190 91 192
193 194 195 196 197 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 200 201
202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217
218 219 220 221 222 223 224 129 139 139 140 140 140 140 140 140
141 141 141 141 141 142 142 142 142 143 143 144 144 145 145 145
145 145 146 146 146 146 146 146 146 146 146 146 147 147 147 147
147 148 148 148 149 149 149 149 149 149 149 149 149 150 150 150
150 150 150 151 151 151 152 152 152 153 153 153 154 154 154 154
154 155 155 155 133 133 133 135 135 135 135 135 135 135 135 135
156 156 156 156 156 156 156 156 156 156 156 156 156 157 157 157
158 158 158 158 158 158 158 158 158 159 159 159 159 159 159 159
159 159 160 160 160 160 160 160 161 161 161 161 161 161 161 161
161 161 161 136 136 136 136 162 162 162 163 164 164 164 164 164
164 165 165 165 166 166 166 167 167 167 168 168 168 138 138 138
138 138 138 138 138 138 138 139 139 139 142 142 142 144 144 144 144
144 169 169 169 145 145 145 170 170 170 171 171 171 172 151 151
173 174 174 159 159 159 160 160 160 136 136 136 170 170 170 171
171 171 175 175 175 160 160 160 134 134 135 134 136 136 136 137

CHAPTER 5

Data Collection & Analysis

5.1 Data Collection

Data are raw facts or the sources of information. Hence data should be collected very carefully. All the data are collected by means of primary sources. All the data collected in this dissertation work are primary data and are collected from traces generated by the simulated page replacement algorithms. In this study, main focus is given in weak locality workload and taking various workloads eg. Probabilistic patterns, cyclic patterns, temporally clustered patterns and mixed patterns. Therefore first of all memory traces having these four patterns are generated and then those traces are used as workloads for page replacement algorithms. Traces generated by page replacement algorithms are then used to calculate their hit rates. Here workloads can be categorized into reference of loop which is larger than cache size as Workload 1 (flimpse), reference of temporally clustered as Workload 2 (Sprite), reference of probabilistic pattern as Workload 3 (cpp) and reference of mixed pattern as Workload 4 (multi2) [18]. Each category contains minimum of ten thousand memory references. Sample of Workload 1, Workload 2, Workload 3 and Workload 4 is in appendix A, appendix B, appendix C and appendix D respectively.

5.2 Testing

These four workloads are separately tested in our simulator. Each workload is tested in CLOCK, CLOCK-Pro and Adaptive CLOCK-Pro simulator by varying the cache size from 4 to 1024. In case of CLOCK-Pro algorithms cold page list and hot page list partition is maintained as 25% and 75% of cache size. But in case of Adaptive CLOCK-Pro initially take size of cold page list and hot page list 25% and 75% of cache size respectively and the size of cold size is dynamically changed by testing whether the page is in test case or not.

5.2.1 Test Result of Workload 1

5.2.1.1 Test Result for these three algorithms with varying cache size

No. of References = 10000

No. of Distinct Pages=2412

Cache Size	CLOCK			CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	9914	98.87%	1.13%	9894	98.60%	1.40%	9885	98.48%	1.52%
8	9907	98.77%	1.23%	9846	97.97%	2.03%	9835	97.83%	2.17%
16	9907	98.77%	1.23%	9804	97.42%	2.58%	9790	97.23%	2.77%
32	9905	98.75%	1.25%	9720	96.31%	3.69%	9695	95.98%	4.02%
64	9905	98.75%	1.25%	9531	93.82%	6.18%	9502	93.44%	6.56%
128	9905	98.75%	1.25%	9147	88.76%	11.24%	9106	88.22%	11.78%
256	9902	98.71%	1.29%	8379	78.64%	21.36%	8289	77.45%	22.55%
512	9874	98.34%	1.66%	6842	58.38%	41.62%	6701	56.52%	43.48%
1024	7656	69.11%	30.89%	4632	29.26%	70.74%	4489	27.37%	72.63%

Table 5.1 Test Result of Workload 1 with varying cache size

5.2.1.2 Test Result for CLOCK-Pro and Adaptive CLOCK-Pro with varying cold block size:

No. of References = 10000

Cache Size =512 No of Distinct Pages=2412

Cold block size	CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
50%	7866	71.88%	28.12%	7751	70.36%	29.64%
45%	7658	69.14%	30.86%	7521	67.33%	32.67%
40%	7450	66.39%	33.61%	7386	65.55%	34.45%
35%	7250	63.76%	36.24%	7098	61.76%	38.24%
30%	7042	61.02%	38.98%	6988	60.31%	39.69%
25%	6842	58.38%	41.62%	6751	57.18%	42.82%
20%	6634	55.64%	44.36%	6544	54.45%	45.55%
15%	6426	52.90%	47.10%	6298	51.21%	48.79%
10%	6226	50.26%	49.74%	6109	48.72%	51.28%
05%	6018	47.52%	52.48%	5998	47.26%	52.74%
04%	5978	47.00%	53.00%	5990	47.15%	52.85%
03%	5938	46.47%	53.53%	5988	47.13%	52.87%
02%	5898	45.94%	54.06%	5978	47.00%	53.00%
01%	5859	45.43%	54.57%	5977	46.98%	53.02%

Table 5.2 Test Result of Workload 1 with varying cold block size

5.2.2 Test Result of Workload 2

5.2.2.1 Test Result for these three algorithms with varying cache size

No. of References = 10000 No of Distinct Pages=2652

Cache Size	CLOCK			CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	9179	88.83%	11.17%	9179	88.83%	11.17%	9151	88.45%	11.55%
8	9124	88.08%	11.92%	9124	88.08%	11.92%	9109	87.87%	12.13%
16	9007	86.49%	13.51%	9049	87.06%	12.94%	9005	86.46%	13.54%
32	8659	81.75%	18.25%	8722	82.61%	17.39%	8689	82.16%	17.84%
64	8222	75.80%	24.20%	8165	75.03%	24.97%	8101	74.16%	25.84%
128	7670	68.29%	31.71%	7071	60.14%	39.86%	7021	59.46%	40.54%
256	6371	50.61%	49.39%	5826	43.20%	56.80%	5793	42.75%	57.25%
512	4325	22.77%	77.23%	4181	20.81%	79.19%	4133	20.16%	79.84%
1024	3144	6.70%	93.30%	3218	7.70%	92.30%	3189	7.31%	92.69%

Table 5.3 Test Result of Workload 2 with varying cache size

5.2.2.2 Test Result for CLOCK-Pro and Adaptive CLOCK-Pro with varying cold block size

No. of References = 10000

Cache Size =512 No of Distinct pages=2652

Cold block size	CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
50%	4618	26.76%	73.24%	4591	26.39%	73.61%
45%	4247	21.71%	78.29%	4201	21.08%	78.92%
40%	4185	20.86%	79.14%	4107	19.80%	80.20%
35%	4153	20.43%	79.57%	4098	19.68%	80.32%
30%	4174	20.71%	79.29%	4103	19.75%	80.25%
25%	4181	20.81%	79.19%	4111	19.86%	80.14%
20%	4190	20.93%	79.07%	4119	19.96%	80.04%
15%	4195	21.00%	79.00%	4123	20.02%	79.98%
10%	4224	21.39%	78.61%	4199	21.05%	78.95%
05%	4252	21.77%	78.23%	4189	20.92%	79.08%
04%	4256	21.83%	78.17%	4197	21.03%	78.97%
03%	4275	22.09%	77.91%	4201	21.08%	78.92%
02%	4315	22.63%	77.37%	4279	22.14%	77.86%
01%	4354	23.16%	76.84%	4295	22.36%	77.64%

Table 5.4 Test Result of Workload 2 with varying cold block size

5.2.3 Test Result of Workload 3

5.2.3.1 Test Result for these three algorithms with varying cache size

No. of References = 30241 No of Distinct Pages=7454

Cache Size	CLOCK			CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	30241	100.00%	0.00%	29243	95.62%	4.38%	29195	95.41%	4.59%
8	29741	97.81%	2.19%	28804	93.69%	6.31%	28769	93.54%	6.46%
16	29395	96.29%	3.71%	28322	91.57%	8.42%	28281	91.40%	8.60%
32	29000	94.55%	5.45%	27018	85.86%	14.14%	26979	85.68%	14.32%
64	28716	93.31%	6.69%	24556	75.05%	24.95%	24478	74.71%	25.29%
128	27696	88.83%	11.17%	21710	62.56%	37.44%	21611	62.13%	37.87%
256	23787	71.68%	28.32%	19783	54.11%	45.89%	18611	48.96%	51.04%
512	20021	55.15%	44.85%	16805	41.04%	58.96%	16632	40.28%	59.72%
1024	18603	48.93%	51.07%	15719	36.27%	63.73%	15419	34.95%	65.05%

Table 5.5 Test Result of Workload 3 with varying cache size

5.2.3.2 Test Result for CLOCK-Pro and Adaptive CLOCK-Pro with varying cold block size:

No. of References = 30241

Cache Size =512 No of Distinct Pages=7454

Cold block size	CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
50%	18132	46.86%	53.14%	18101	46.72%	53.28%
45%	17897	45.83%	54.17%	17813	45.46%	54.54%
40%	17303	43.22%	56.78%	17283	43.13%	56.87%
35%	17038	42.06%	57.94%	17003	41.91%	58.09%
30%	16905	41.48%	58.52%	16876	41.35%	58.65%
25%	16805	41.04%	58.96%	16781	40.93%	59.07%
20%	16785	40.95%	59.05%	16725	40.69%	59.31%
15%	16721	40.67%	59.33%	16700	40.58%	59.42%
10%	16702	40.58%	59.42%	16689	40.53%	59.47%
05%	16694	40.55%	59.45%	16671	40.45%	59.55%
04%	16696	40.56%	59.44%	16674	40.46%	59.54%
03%	16703	40.59%	59.41%	16691	40.54%	59.46%
02%	16711	40.62%	59.38%	16689	40.53%	59.47%
01%	16750	40.80%	59.20%	16711	40.62%	59.38%

Table 5.6 Test Result of Workload 3 with varying cold block

5.2.4 Test Result of Workload 4

5.2.4.1 Test Result for these three algorithms with varying cache size

No. of References = 10000 No of Distinct Pages=3353

Cache Size	CLOCK			CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	9882	98.22%	1.78%	9807	97.10%	2.90%	9781	96.71%	3.29%
8	9866	97.98%	2.02%	9643	94.63%	5.37%	9602	94.01%	5.99%
16	9838	97.56%	2.44%	9332	89.95%	10.05%	9299	89.45%	10.55%
32	9629	94.42%	5.58%	8830	82.40%	17.60%	8786	81.74%	18.26%
64	9550	93.23%	6.77%	7833	67.40%	32.60%	7797	66.86%	33.14%
128	9279	89.15%	10.85%	6817	52.11%	47.89%	6747	51.06%	48.94%
256	7122	56.70%	43.30%	6158	42.20%	57.80%	6092	41.21%	58.79%
512	6401	45.86%	54.14%	5303	29.34%	70.66%	5212	27.97%	72.03%
1024	5128	26.70%	73.30%	4430	16.20%	83.80%	4309	14.38%	85.62%

Table 5.7 Test Result of Workload 4 with varying cache size

5.2.4.2 Test Result for CLOCK-Pro and Adaptive CLOCK-Pro with varying cold block size:

No. of References = 10000

Cache Size =512 No of Distinct Pages=3353

Cold block size	CLOCK-Pro			Adaptive CLOCK-Pro		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
50%	5676	34.95%	65.05%	5595	33.73%	66.27%
45%	5543	32.95%	67.05%	5489	32.13%	67.87%
40%	5423	31.14%	68.86%	5401	30.81%	69.19%
35%	5344	29.95%	70.05%	5311	29.46%	70.54%
30%	5322	29.62%	70.38%	5301	29.31%	70.69%
25%	5303	29.34%	70.66%	5279	28.98%	71.02%
20%	5252	28.57%	71.43%	5203	27.83%	72.17%
15%	5190	27.64%	72.36%	5122	26.61%	73.39%
10%	5127	26.69%	73.31%	5095	26.21%	73.79%
05%	5071	25.85%	74.15%	5999	39.81%	60.19%
04%	5061	25.70%	74.30%	5077	25.94%	74.06%
03%	5052	25.56%	74.44%	5087	26.09%	73.91%
02%	5043	25.43%	74.57%	5115	26.51%	73.49%
01%	5035	25.30%	74.70%	5112	26.46%	73.54%

Table 5.8 Test Result of Workload 4 with varying cold block size

4.3 Analysis

All the collected data is analyzed by drawing different graphs. Hit rate of algorithms is used as criteria for analyzing their goodness.

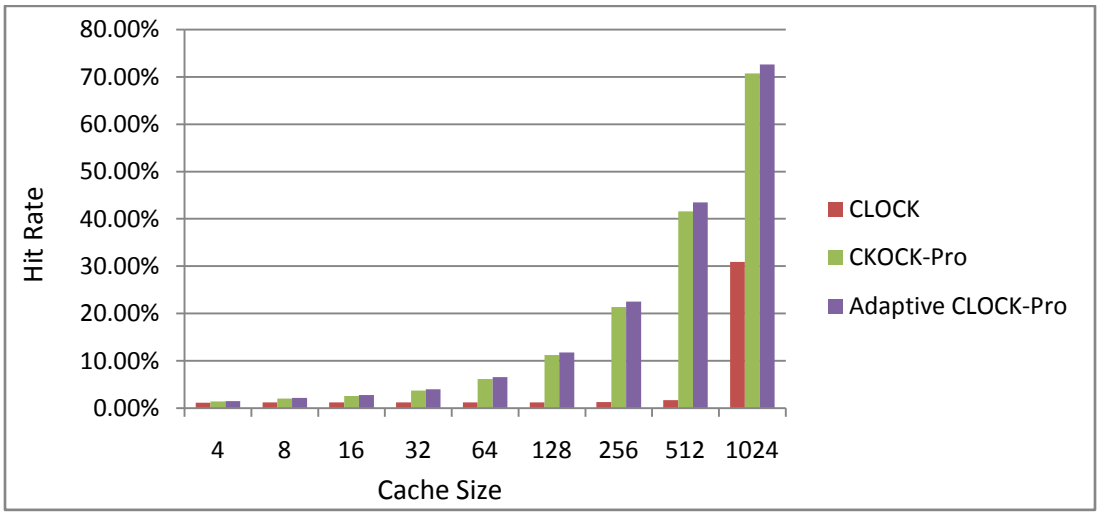


Figure 5.1: Graph for Table 5.1

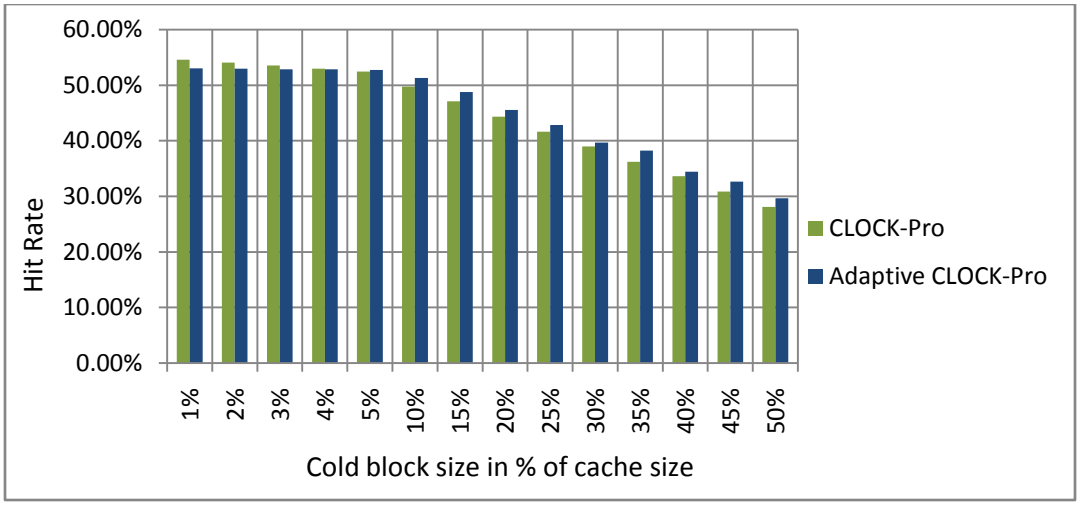


Figure 5.2: Graph for Table 5.2

The graphs of Figure 5.1 show that the Adaptive CLOCK-Pro algorithm is better than CLOCK and CLOCK-Pro algorithms. Since the workloads used in this work represent weak locality of memory references, the performance of CLOCK is worst in this case. After increasing the cache size to 1024, CLOCK performances drastically changes than CLOCK-Pro and Adaptive CLOCK-Pro algorithms. This is because loop size is nearly equal to 1024. Figure 5.2 show that Clock-pro algorithm gives best performance when

cold size is 1% of total cache size. When cold size is more than 5% of total memory size, Adaptive Clock-Pro seems more effective than Clock-Pro. But as the cold size is less or equal to 5% of total memory size, CLOCK-Pro seems to be more effective than Adaptive CLOCK-Pro.

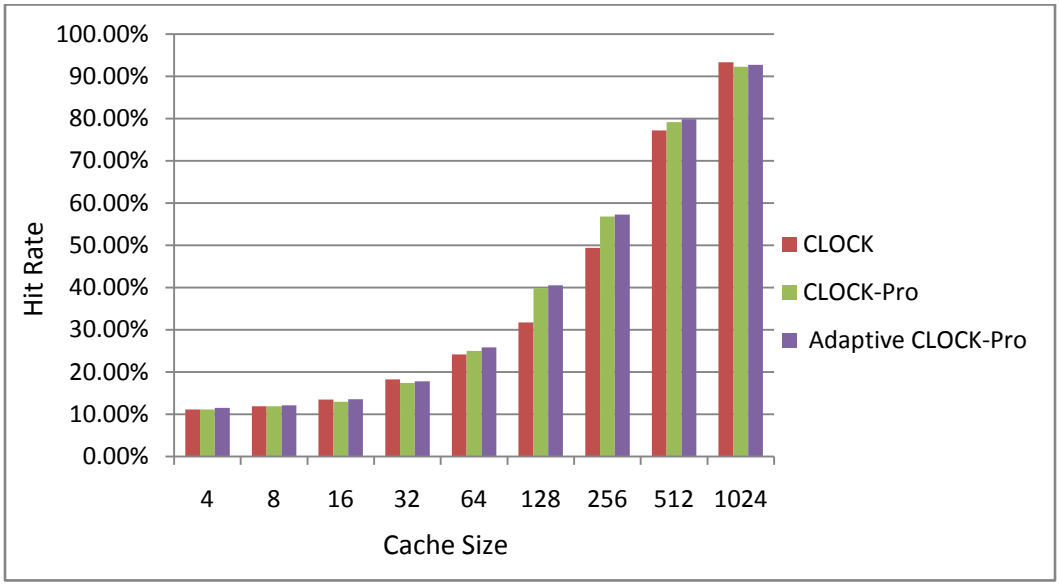


Figure 5.3: Graph for Table 5.3

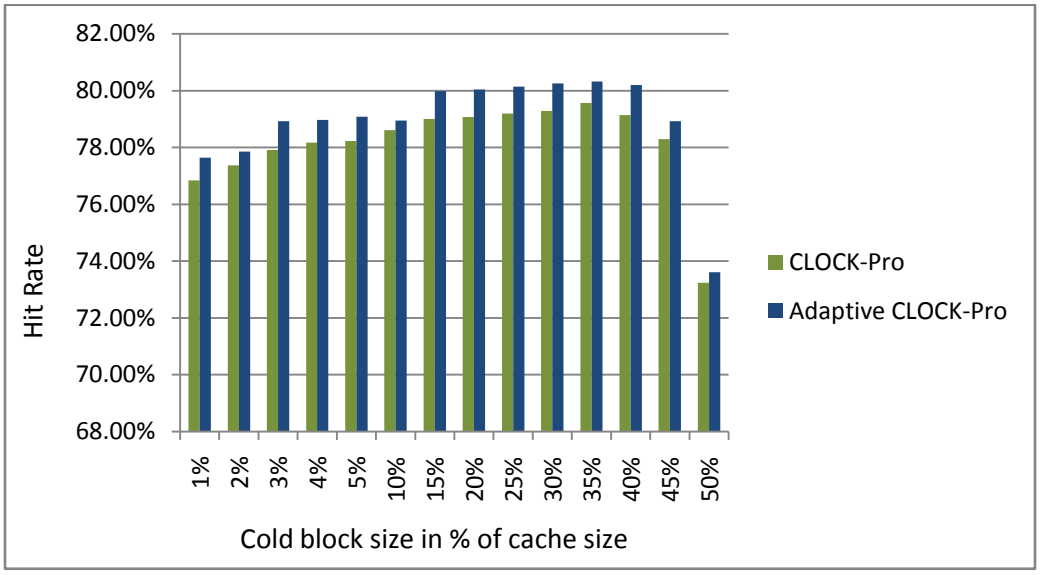


Figure 5.4: Graph for Table 5.4

Figure 5.3 shows that performances of CLOCK, CLOCK-Pro, and adaptive CLOCK-Pro is Comparable. This is because temporally clustered pattern is strong locality workload.

Thus CLOCK-Pro and adaptive CLOCK-Pro page replacement policies do not hurt performance in case of strong locality workloads. In this workload the best cache partition ratio is observed at 35% of cold size. Adaptive CLOCK-Pro is always superior than CLOCK-Pro.

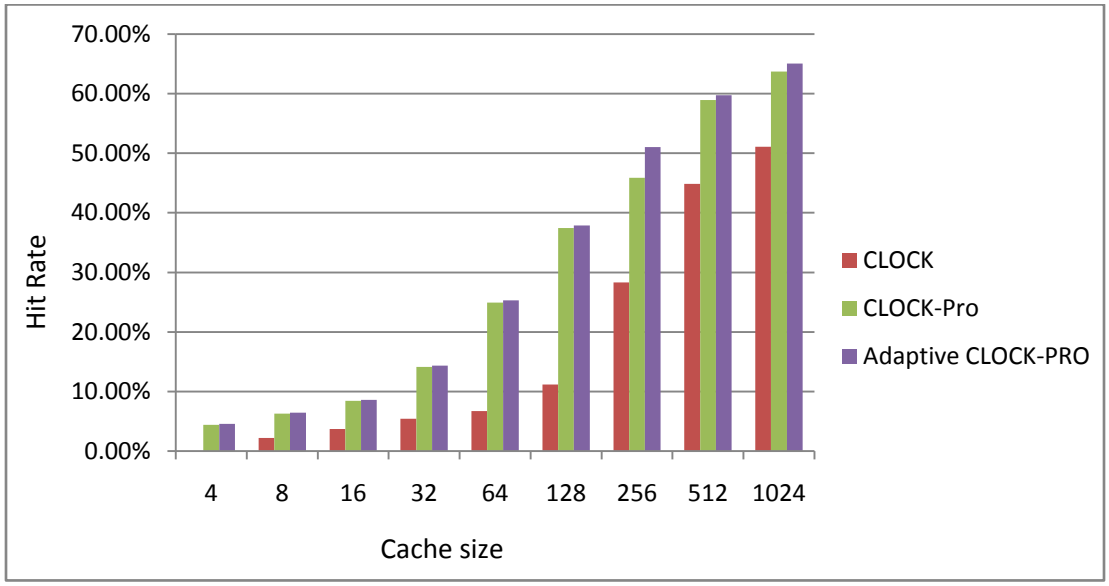


Figure 5.5: Graph for Table 5.5

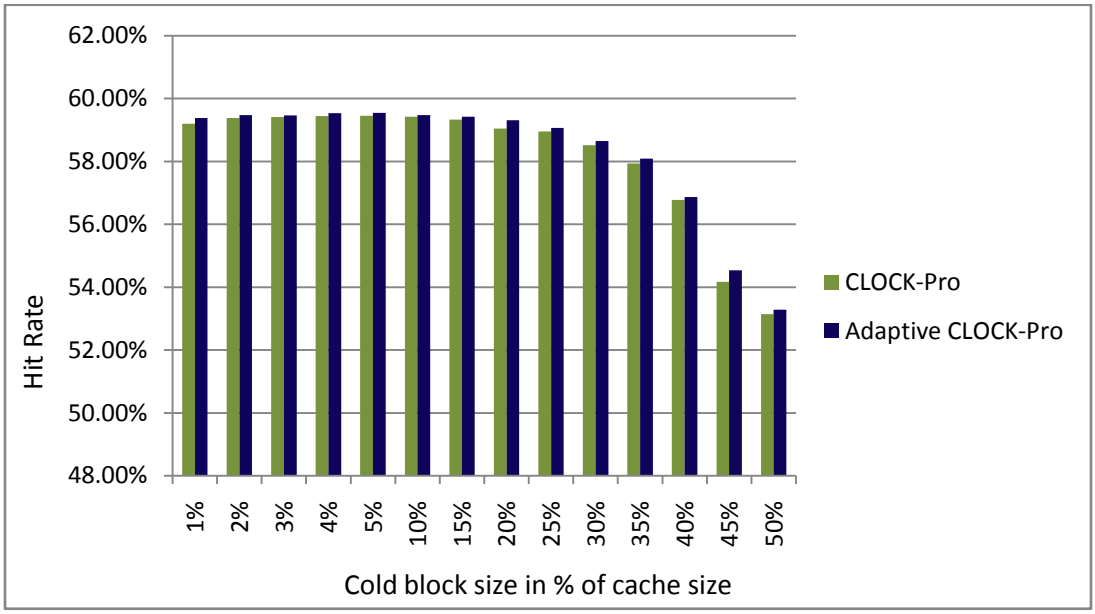


Figure 5.6: Graph for Table 5.6

Figure 5.5 show the performance of CLOCK-Pro and Adaptive CLOCK-Pro is better than CLOCK algorithm. And for this workload best cache partition ratio is observed at

5% cold size. Here again adaptive CLOCK-Pro is always superior than CLOCK-Pro but their performances are almost similar when cold size is below 5% of total cache size.

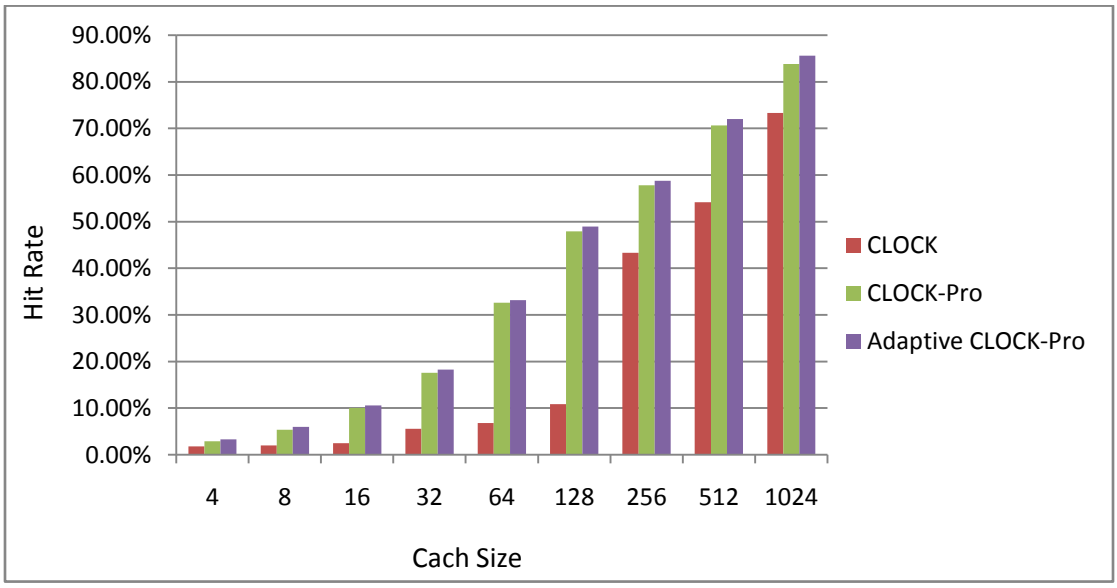


Figure 5.7: Graph for Table 5.7

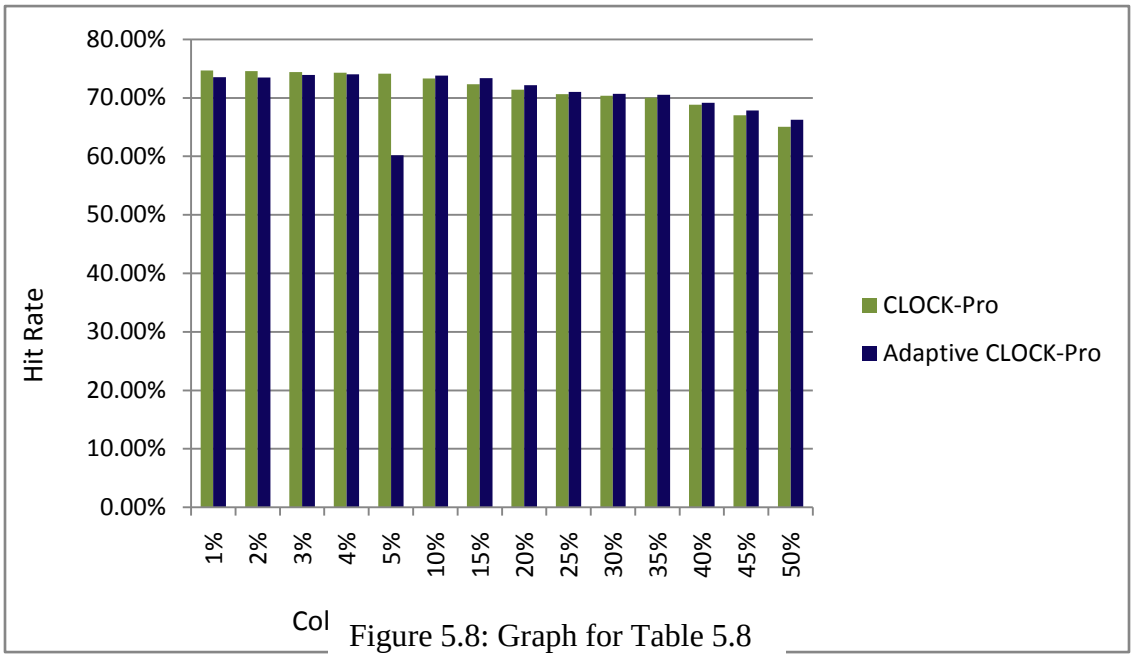


Figure 5.8: Graph for Table 5.8

Figure 5.7 show the performance of CLOCK-Pro and Adaptive CLOCK-Pro is better than CLOCK algorithm. For this workload best cache partition ratio is observed at 5%. When cold size is equal or less than 5% of total cache size, CLOCK-Pro seems to little bit more superior to adaptive CLOCK-Pro.

CHAPTER 6

Conclusion and Future Study

6.1 Conclusion

Replacement algorithms are valuable components of operating system design and can affect system performance significantly. CLOCK-Pro can solve problems regarding weak locality of reference by tracing and utilizing history information. The failure of CLOCK is due to the bold assumption on recency. Negative effects caused by taking only recency value are removed by considering hot pages, cold pages, and non-resident cold pages as history information. The algorithm successfully handles weak locality of reference.

For weak locality workloads CLOCK-Pro and adaptive CLOCK-Pro always performs better than CLOCK page replacement algorithm. CLOCK-Pro page replacement policy increases hit rate up to 40% and adaptive CLOCK-Pro increases hit rate up to 43%. Performance gain is more in case of purely weak locality workloads (such as looping pattern) and performances are comparable in case of strong locality workloads (such as temporally clustered workloads). Again, if the number of distinct pages is nearly equal to cache size CLOCK algorithm also performs better for weak locality workloads.

The performance comparison between CLOCK-Pro and Adaptive CLOCK-Pro algorithms is done by changing cold block size from 1% to 50% of cache size. For weak locality workloads, cold size below 5% of total cache size is seems to be better but for strong locality workload cold size around 35% of total cache size seems to be more beneficial. Again when cold size is below 5% for weak locality workloads, performance of CLOCK-Pro is little bit higher than adaptive CLOCK-Pro and when cold size is more than 5% adaptive CLOCK-Pro outperforms CLOCK-Pro. But, for strong locality workloads adaptive CLOCK-Pro always outperforms CLOCK-Pro. Thus summing up this it can be concluded that best cache partition ratio is different for different workloads and therefore it is very difficult to declare it. Thus adaptive CLOCK-Pro is Superior than CLOCK-Pro because it can easily adopt cold and hot size for strong locality as well as weak locality workloads.

6.2 Future Work

CLOCK-Pro algorithm consist the hot pages, cold pages and metadata information about the non-resident cold pages. In this dissertation work we analyze sensitivity analysis of cache partition in CLOCK-Pro. Also compare the performances of CLOCK, CLOCK-Pro and Adaptive CLOCK-Pro. In this dissertation work only M sized non-resident cold pages are taken when memory of size M. Changing the size of non-resident blocks and evaluating impact is interesting for further research.

References

[1] A.S. Tanenbaum, Modern Operating Systems (Prentice Hall Second Edition), pp 201-232. 2007.

[2] Bagchi, S., Nygaard, M.A Fuzzy Adaptive Algorithm for Fine Grained Cache Paging. 8th International Workshop (SCOPES'04), Netherlands, pp 200-213. 2004.

[3] Bansal, S. and Modha, D. S. CAR: Clock with Adaptive Replacement, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'04), San Francisco, pp 187-200 (2004)

[4] B. Subedi, An Evaluation of Page Replacement Algorithm Based on Low Inter Reference Recency Set Scheme on Weak Locality Workloads, Master's Thesis, Tribhuvan University, Central Department of Computer Science and Information Technology.

[5] BP-Wrapper: A System Framework Making Any Replacement Algorithms (Almost) Lock Contention Free Xiaoning Ding, Song Jiang, Xiaodong Zhang, pp 370.

[6] C. Ding and Y. Zhong, "Predicting Whole-Program Locality through Reuse-Distance Analysis", *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2003.

[7] Choi, J. An Implementation Study of a Detection-Based Adaptive Block Replacement Scheme, USENIX Annual Technical Conference, 239-252. (1999)

[8] Choi, J. Towards application/file-level characterization of block references: a case for fine-grained buffer management. In: Proceeding of the 25th International Conference on Measurement and Modeling of Computer Systems, Santa Clara, CA. (SIGMETRICS'00), and pp 286-295. (2000)

[9] Corbató, F. J.A paging experiment with the Multics system. In Honor of P. M. Morse, pp 217–228, MIT Press, 1969. Also as MIT Project MAC Report MAC-M-384.(1968)

[10] D. Lee, J. Choi, J. Kim, S. Noh, S. Min, Y. Cho and C. Kim, "On the Existence of a Spectrum of Policies that Subsumes the Least Recently Used (LRU) and Least Frequently Used (LFU) Policies", *Proceeding of 1999 ACMSIGMETRICS Conference*, 1999.

[11] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum, The LRU-K Page Replacement Algorithm for Database Disk Buffering, ACM SIGMOD, Washington D.C., pp 297-306.(1993)

[12] G. Glass, "Adaptive Page Replacement". Master's Thesis, University of Wisconsin, 1997.

[13] Glass, G. and Cao, P. Adaptive Page Replacement Based on Memory Reference Behavior, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97), pp 115-126.(1997)

[14] G. Nutt, Operating Systems A Modern Perspective(Addison Wesley Longman, Second Edition) 2000

[15] G. Prakash Joshi, Calculation Of Control Parameter λ That Results Into Optimal Performance In Terms Of Page Fault Rate In The Algorithm Least Recently Frequently Used(LRFU) For Page Replacement, Master's Thesis, Tribhuvan University, Central Department of Computer Science and Information Technology.

[16] H.M. Deitel, Operating Systems, Chap.9 Virtual Storage Management (Pearson Education, Second Edition).

[17] H. Paajanen, Page Replacement In Operating System Memory Management, Master's Thesis in Information Technology, University of Jyvaskyla, Department of Mathematical Information Technology (2007).

[18] Jiang, S., Chen, F., Zhang, X. CLOCK-Pro: An effective improvement of the CLOCK replacement. In Proceedings of the 10th Annual USENIX Technical 2005.

[19] Johnson and Shasha, *2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm*, Proceedings of the 20th International Conference on VLDB, pp 439-450. 1994

[20] John Kubiatowicz, Operating System and System Programming, Page Allocation and replacement , *http://inst.eec.berkeley.edu/~CS162*

[21] Kim, J.M. et al. A low-overhead high-performance unified buffer management scheme that exploit sequential and looping references. In Symposium on Operating System Design and Implementation, San Diego. OSDI' 2000 USENIX, pp 119-134.(2000)

[22] McMaster, Sambasivam, & Anderson, How Anomalous Is Belady's Anomaly?, Issues in Informing Science and Information Technology VOL 6, pp827-836(2009)

[23] Megiddo, N. and Modha, D. S.ARC: A Self-Tuning, Low Overhead Replacement Cache, In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'03), San Francisco, pp 115-130.(2003)

[24] M.L. Singh, Understanding Research Methodology, Chap.1Scientific Method and Research, pp 4.

[25] Sabeghil, M. and Yaghmaee, M. H Using fuzzy logic to improve cache replacement decisions. IJCSNS International Journal of Computer Science and Network. 2006.

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

[27] Silberschatz, A., Galvin, P. B., & Gagne, G., Operating system concepts (7th Edition). Wiley.Stuart, 2004

[28] S. Jiang and X. Zhang, "LIRS: An Efficient Low Inter reference Recency Set Replacement Policy to Improve Buffer Cache Performance", *In Proceeding of 2002 ACM SIGMETRICS*, pp. 31-42.(2002)

Appendix A: Sample Trace of loop pattern, Workload 1

0 1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 28 28 2 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67
68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90
91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143
144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177
178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194
195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211
212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228
229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245
246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262
263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279
280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296
297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313
314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330
331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347
348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364
365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381
382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398
399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415
416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432
433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449
450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466
467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483
484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500
501 502 503 504 505 506 507 508 509 510 0 511 512 513 514 515 516
517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533
534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550
551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567
568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584
585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601
602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618

619 620 621 622 623 624 625 626 627 628 628 628 28 628 629 630 631
632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648
649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665
666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682
683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699
700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716
717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733
734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750
751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767
768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784
785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801
802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818
819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835
836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852
853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869
870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886
887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903
904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920
921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937
938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954
955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971
972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988
989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004
1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017
1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030
1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043
1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056
1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069
1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082
1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095
1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108
1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121
1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134
1135 1136 1137 28 28 28 28 0 0 1 1 2 3 28 28 28 2 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391
392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408
409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476
477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493
494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510
511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527
528 529 530 531 532 533 534 535 536 537 538 0 539 540 541 542 543
544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560
561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577
578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594
595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611
612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628
628 628 28 628 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147
1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160
1161 1162 1163

Appendix B: Sample Trace of Probabilistic Pattern, Workload 2

0 1 2 3 4 5 6 7 8 8 9 9 10 11 12 13 14 15 16 16 17 18 17 18 18 18 19
20 1 2 3 4 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
86 87 88 89 90 19 20 1 2 3 4 91 92 93 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 70 94 95 96 97 98 62 63
99 100 101 56 57 64 65 53 54 55 58 59 60 102 66 67 68 69 49 50 51 52
103 104 105 106 71 72 73 74 75 76 77 81 107 82 83 85 86 84 87 88 108
89 109 110 111 112 113 110 19 20 1 2 3 4 114 115 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 94 95 97 116 62 63 96 66 67
68 69 56 57 70 99 43 44 45 46 47 48 71 72 73 74 75 76 77 117 81 82
83 108 84 85 86 87 88 107 89 19 20 1 2 3 4 118 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 70 94 95 97 62 63 19 20 1 2 3 4 119 120 121 122
123 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 117 43
44 45 46 47 48 96 64 65 19 20 1 2 3 4 124 125 126 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 61 62 63 105 106 66 67 68 69
56 57 70 104 43 44 45 46 47 48 71 72 73 74 75 76 77 99 127 49 50 51
52 53 54 55 58 59 60 128 129 130 131 132 133 81 82 83 108 84 85 86
87 88 134 135 136 137 19 20 1 2 3 4 138 139 140 26 27 28 29 24 25 30
31 32 33 34 35 36 37 38 39 40 41 42 53 54 55 56 57 58 59 60 99 49 50
51 52 19 20 1 2 3 4 141 26 27 28 29 24 25 30 31 32 33 34 35 36 37 38
39 40 41 42 53 54 55 56 57 58 59 60 99 49 50 51 52 103 104 109 19 20
1 2 3 4 142 143 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 66 67 68
69 71 72 73 74 75 76 77 100 101 144 62 63 105 106 102 78 70 145 146
147 148 149 150 151 152 153 154 155 156 157 19 20 1 2 3 4 158 159
160 161 162 163 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48 164 165 62 63 66 67 68 69 56 57 70 104 166
167 103 81 82 83 108 84 85 86 87 88 105 106 129 130 131 132 133 168
169 170 19 20 1 2 3 4 171 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 172 173 174 175 71 72 56 57 73 74
75 76 77 19 20 1 2 3 4 176 177 178 179 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
57 58 59 60 172 173 174 175 61 62 63 105 106 71 72 73 74 75 76 77 66

Appendix C: Sample Trace of Temporally-clustered Pattern, Workload 3

0 1 2 3 4 4 5 6 7 8 9 10 11 12 13 14 9 15 16 17 9 18 9 19 9 20 21 22
23 24 25 26 15 16 4 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 5 86 87
88 89 2 3 90 91 92 93 94 95 96 97 98 99 100 8 101 54 59 4 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 102 103 104 105 106 107
108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125 126 127 128 82 83 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 129 129 130
131 132 133 133 133 133 134 134 134 134 135 135 136 136 136 136 136
136 137 137 137 138 138 138 139 139 139 139 139 139 139 139 139 139
140 140 140 140 140 140 141 141 141 141 141 142 142 142 142 143 143
144 144 145 145 145 145 145 146 146 146 146 146 146 146 146 146 146
147 147 147 147 147 148 148 148 149 149 149 149 149 149 149 149 149
150 150 150 150 150 150 151 151 151 152 152 152 153 153 153 154 154
154 154 154 155 155 155 133 133 133 135 135 135 135 135 135 135 135
135 156 156 156 156 156 156 156 156 156 156 156 156 156 156 157 157 157
158 158 158 158 158 158 158 158 158 159 159 159 159 159 159 159 159
159 160 160 160 160 160 160 161 161 161 161 161 161 161 161 161 161
161 136 136 136 136 162 162 162 163 164 164 164 164 164 164 165 165
165 166 166 166 167 167 167 168 168 168 138 138 138 138 138 138 138
138 138 139 139 139 142 142 142 144 144 144 144 144 169 169 169 145
145 145 170 170 170 171 171 171 172 151 151 173 174 174 159 159 159
160 160 160 136 136 136 170 170 170 171 171 171 175 175 175 160 160
160 134 134 135 134 136 136 136 137 137 138 138 139 139 139 139 139
139 140 140 140 140 141 141 142 142 143 142 143 144 143 144 145 145
145 145 146 146 146 146 146 146 147 146 147 147 148 148 149 149 149
150 150 150 151 151 152 152 153 153 154 154 154 154 155 155 133 135
135 135 135 135 135 156 156 156 156 156 156 156 157 157 158 158 158
158 158 159 159 159 160 159 160 160 160 161 161 161 161 161 161 136
136 163 164 164 165 165 166 166 167 167 168 168 138 138 138 138 138
139 142 144 144 169 169 145 145 170 170 171 171 151 172 174 173 174

Appendix D: Sample Trace of Mixed Pattern, Workload 4

0 1 2 3 4 5 6 7 7 8 8 9 10 11 12 13 14 15 16 0 1 2 17 1 18 1 19 1 20
1 1 21 1 1 3 22 23 24 25 26 27 28 29 29 29 29 29 29 29 29 29 29 29
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
29 29 30 31 27 28 28 28 32 33 34 35 0 1 2 3 36 37 38 39 40 41 42 43
44 45 45 45 45 46 47 31 35 36 38 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 35 40 64 65 66 67 68 69 70 71 72 31 33 35 40 73 74 75 76
74 77 78 77 79 80 31 35 69 64 81 74 82 83 31 35 64 40 76 84 85 86 87
88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107
108 109 110 50 31 33 35 66 36 38 69 71 78 83 86 91 93 96 98 88 111
112 113 114 115 116 117 103 106 118 119 120 53 121 37 37 39 39 122
123 124 57 61 31 33 35 66 36 38 69 71 78 83 86 91 93 96 98 88 125
126 127 128 129 116 91 93 103 106 121 57 61 130 131 132 133 134 135
35 55 91 93 136 137 138 31 33 35 66 36 38 69 71 78 83 86 91 93 96 98
88 139 140 141 142 143 144 145 146 147 148 111 113 116 149 150 151
152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168
169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185
186 187 188 189 190 191 192 193 194 159 195 162 196 197 166 198 173
199 200 201 202 203 177 204 205 206 207 208 209 210 211 212 213 214
215 216 217 218 219 220 221 159 222 223 162 224 225 166 226 173 227
228 177 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243
244 159 245 246 162 247 248 166 249 173 250 251 177 252 253 254 252
255 256 252 252 257 258 259 255 260 261 262 263 264 265 266 267 268
269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285
286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302
303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319
320 321 322 323 32 34 0 1 2 3 324 325 326 327 328 44 46 329 330 331
332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348
349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365
51 366 367 58 368 369 63 370 371 68 372 373 75 374 375 80 376 377 85
378 379 90 380 381 95 382 383 100 384 385 105 386 387 110 388 389
115 390 391 120 392 393 124 394 395 129 396 397 134 398 399 138 400
401 143 146 402 403 151 404 405 155 406 407 160 408 409 232 410 411
412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428
429 430 431 432 433 434 435 436 205 208 437 438 439 440 441 442 443
444 445 446 447 448 449 450 451 452 453 185 188 454 455 213 216 456

Appendix E: Source Code for CLOCK, CLOCK-Pro and Adaptive CLOCK-Pro algorithms respectively as below:

Clock_Node.java

```
package clock_replacement;
public class Clock_node
{
   int R;//referenc
   int pn;//page number
   Clock_node next,prev;
   boolean isresident;
   public Clock_node()
   {
      R=0;
      pn=0;
      isresident=false;
      next=prev=null;
   }}
```

ClockMain.java

```
package clock_replacement;
import java.io.FileNotFoundException;
class SizeInfo
{
   static int vir_mem;
   static int mem_size;
   SizeInfo()
   {
```

```
        vir_mem=8;
        mem_size=3;
    }}
public class Clockmain
{
    public static void main(String[] args) throws FileNotFoundException
    {
      new Clock();
    }}
```

Clock.java

```
package clock_replacement;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
public class Clock extends SizeInfo
{
    private Clock_node tail, hand;
    private int npf;//total number of page faults
    private int np;//total number of pages
    private Clock_node[] pt;
    private int free_mem_size;
    //boolean isresident;
    Clock()
    {
        tail=hand=null;
        npf=np=0;
```

```
        free_mem_size=3;
        pt=new Clock_node[vir_mem+1];
        for(int i=1;i<vir_mem+1;i++)
        {
            pt[i]=new Clock_node();
            pt[i].pn=i;
        }
        trace();
    }

void trace()
    {
       Scanner in=null;
       int refpage=0;
        try {
            in = new Scanner(new File("clock.txt"));
        } catch (FileNotFoundException ex) {
            System.out.println("input file not found"+ex);
        }
        while(in.hasNext())
        {
            refpage=in.nextInt();
            np++;
            if(pt[refpage].isresident==true)
            {
                //System.out.print("page in stack"+refpage);
               if(pt[refpage].R==1)
```

```
  {
     //do nothing
  }
  else
  {
     pt[refpage].R=1;
  } }
else  //if page is not in clock
{
   //System.out.print("page in not stack"+refpage);
   npf++;
   if(free_mem_size>0)
   {
      if(hand==null)
      {
         hand=tail=pt[refpage];
         pt[refpage].next=hand;
         pt[refpage].prev=pt[refpage].next;
         pt[refpage].R=0;
         pt[refpage].isresident=true;
         free_mem_size=free_mem_size-1;
      }
      else
      {
         tail.next=pt[refpage];
         pt[refpage].prev=tail;
         pt[refpage].next=hand;
```

```
            tail=tail.next;
            hand.prev=tail;
            pt[refpage].R=0;
            pt[refpage].isresident=true;
            free_mem_size=free_mem_size-1;
          }}
        else  //MEMORY IS FULL
        {
            while(hand.R!=0)
            {
              hand.R=0;
             hand=hand.next;
            }
            hand.isresident=false;
            hand.prev.next=pt[refpage];
            pt[refpage].prev=hand.prev;
            pt[refpage].next=hand.next;
            hand.next.prev=pt[refpage];
            hand=pt[refpage];
            hand=hand.next;
            pt[refpage].isresident=true;
            pt[refpage].R=0;
        }}
      showStatus();
     }
}
```

```
 void showStatus()
    {
        System.out.println("------------Clock------------------:");
        System.out.println();
        System.out.println("Total number of pages:"+np);
        System.out.println("Total number of page faults:"+npf);
    }
}
```

Clock_pr_Node.java

```
package CLOCK_Pr_Simulation;
enum PageStatus{non_resident,cold,hot};
public class CLOCK_Pr_Node
{
    private int prt;//reference times
    private int pft;//page fault times
    int pn;//page  number
    boolean isresident;
    boolean isinclock;
    PageStatus status;
    int R;          //referenc
    CLOCK_Pr_Node prev;
    CLOCK_Pr_Node next;
    public CLOCK_Pr_Node()
    {
        prt=0;
        pft=0;
```

```
        R=0;
        status=PageStatus.non_resident;
        isresident=false;
        isinclock=false;
        prev=next=null;
    }}
```

Clock_pr_Main.java

```
package CLOCK_Pr_Simulation;
import java.io.FileNotFoundException;
class SizeInfo
{
    static int VM_SIZE;
    static int mem_size;
    SizeInfo()
    {VM_SIZE=100;
        mem_size=5;
    }}
public class CLOCK_Pr_Main
{
    public static void main(String[] args) throws FileNotFoundException
    {
      new CLOCK_Pr();
    }}
```

Clock_pr.java

```
package CLOCK_Pr_Simulation;
import java.io.File;
```

```
import java.io.FileNotFoundException;
import java.util.Scanner;
class CLOCK_Pr extends SizeInfo
{
    private CLOCK_Pr_Node shead,stail,qhead,qtail;
   private int npf;//total number of page faults
    private int np;//total number of pages
    private CLOCK_Pr_Node head;
    private CLOCK_Pr_Node tail;
    private CLOCK_Pr_Node[] pt;
    CLOCK_Pr_Node hand_hot;
    CLOCK_Pr_Node hand_cold;
    CLOCK_Pr_Node hand_test;
    private int free_mem_size;
    private int hot_pages_size;
    private int cold_pages_size;
    private int non_resident_coldpages_size;
    CLOCK_Pr() throws FileNotFoundException
    {head=tail=null;
        free_mem_size=SizeInfo.mem_size;
        if(SizeInfo.mem_size<4)
        {
            cold_pages_size=1;
        }
        else
        {
            cold_pages_size=(int) (SizeInfo.mem_size * 0.25);
```

```
    }
    hot_pages_size=SizeInfo.mem_size-cold_pages_size;
    non_resident_coldpages_size=SizeInfo.mem_size/2;
    npf=np=0;
    pt=new CLOCK_Pr_Node[VM_SIZE+1];
    for(int i=1;i<=VM_SIZE;i++)
    {
        pt[i]=new CLOCK_Pr_Node();
        pt[i].pn=i;
    }
    trace();
}

void trace()
{
   Scanner in=null;
   CLOCK_Pr_Node temp;
   CLOCK_Pr_Node temp1;
   int refpage=0;
    try {
        in = new Scanner(new File("clock_pro.txt"));
    } catch (FileNotFoundException ex) {
        System.out.println("input file not found"+ex);
    }
    while(in.hasNext())
    {
        refpage=in.nextInt();
```

```
np++;
if(pt[refpage].isresident==false) //if page is not in memory
{
   npf++;
   if(pt[refpage].isinclock==false)
   {   if(hot_pages_size>0) //if hot pages list is not full
       {
          if(head==null) //if linked list is empty
          {  head=tail=hand_hot=pt[refpage];
             pt[refpage].status=PageStatus.hot;
             pt[refpage].next=head;
             pt[refpage].prev=pt[refpage].next;
             pt[refpage].R=0;
             pt[refpage].isresident=true;
             hot_pages_size=hot_pages_size-1;
          }
          else
          {   pt[refpage].status=PageStatus.hot;
             tail.next=pt[refpage];
             pt[refpage].prev=tail;
             pt[refpage].next=head;
             tail=tail.next;
             head.prev=tail;
             pt[refpage].R=0;
             pt[refpage].isresident=true;
             hot_pages_size=hot_pages_size-1;
          }}
```

```
else  //if hot block is full
{
   if(cold_pages_size>0) //if cold pages list is not full
   {
      if(cold_pages_size==2) //if clock has no any cold pages
      {
         pt[refpage].status=PageStatus.cold;
         tail.next=hand_cold=pt[refpage];
         pt[refpage].prev=tail;
         pt[refpage].next=head;
         tail=tail.next;
         head.prev=tail;
         pt[refpage].R=0;
         pt[refpage].isresident=true;
         cold_pages_size=cold_pages_size-1;
      }
      else //if clock contains at least one cold page
      {
         pt[refpage].status=PageStatus.cold;
         tail.next=pt[refpage];
         pt[refpage].prev=tail;
         pt[refpage].next=head;
         tail=tail.next;
         head.prev=tail;
         pt[refpage].R=0;
         pt[refpage].isresident=true;
         cold_pages_size=cold_pages_size-1;
```

```
    } }
else // if cold page list is full
{
    if(hand_cold.R==0)
    {
        // then replace block from cold page list to non resident page list
        temp=hand_cold;
        temp.isresident=false;
        temp.isinclock=true;
        temp.status=PageStatus.non_resident;
        non_resident_coldpages_size=non_resident_coldpages_size+1;
        hand_cold=hand_cold.next;
        hand_cold.prev=temp;
        temp.next=hand_cold;
        temp.prev.next=temp;
        hand_cold.prev.prev=temp.prev;
        while(hand_cold.status!=PageStatus.cold)
            hand_cold=hand_cold.next;
        if(non_resident_coldpages_size>0) //if non resident block is not full
        {
            if(non_resident_coldpages_size==SizeInfo.mem_size/2)
            {
                hand_test=temp;
                tail.next=temp;
                temp.prev=tail;
                tail=tail.next;
                head.prev=tail;
```

```
            tail.next=head;
            temp.isinclock=true;
            non_resident_coldpages_size=non_resident_coldpages_size-1;
        }
        else
        {
            tail.next=temp;
            temp.prev=tail;
            tail=tail.next;
            head.prev=tail;
            tail.next=head;
            temp.isinclock=true;
            non_resident_coldpages_size=non_resident_coldpages_size-1;
        }
    }
    else //non resident block is full
    {
        temp1=hand_test;
        temp1.isinclock=false;
        temp1.prev.next=temp1.next;
        temp1.next.prev=temp1.prev;
        non_resident_coldpages_size=non_resident_coldpages_size-1;
        hand_test=hand_test.next;
        while(hand_test.status!=PageStatus.non_resident)
            hand_test=hand_test.next;
        tail.next=temp;
        temp.prev=tail;
```

```
            tail=tail.next;
            head.prev=tail;
            tail.next=head;
            temp.isinclock=true;
            cold_pages_size=cold_pages_size-1;
        }
        tail.next=pt[refpage];
        pt[refpage].prev=tail;
        tail=tail.next;
        head.prev=tail;
        tail.next=head;
        pt[refpage].isresident=true;
        pt[refpage].R=0;
        pt[refpage].status=PageStatus.cold;
        cold_pages_size=cold_pages_size+1;
    }
    else if(hand_cold.R==1)
    {
        if(hot_pages_size>0)
        {
            temp=hand_cold;
            temp.R=0;
            temp.prev.next=temp.next;
            temp.next.prev=temp.prev;
            hand_cold=hand_cold.next;
            while(hand_cold.status!=PageStatus.cold)
                hand_cold=hand_cold.next;
```

```
    tail.next=temp;
    temp.prev=tail;
    tail=tail.next;
    tail.next=head;
    head.prev=tail;
    temp.status=PageStatus.hot;
    cold_pages_size=cold_pages_size-1;
    hot_pages_size=hot_pages_size+1;
}
else //if hot page list is full
{
    temp=hand_cold;
    hand_cold=hand_cold.next;
    temp.status=PageStatus.hot;
    while(hand_cold.status!=PageStatus.cold)
        hand_cold=hand_cold.next;
    temp1=hand_hot;
    hand_hot=hand_hot.next;
    while(hand_hot.status!=PageStatus.hot)
        hand_hot=hand_hot.next;
     temp1.status=PageStatus.cold;
     tail.next=temp1;
     temp1.prev=tail;
     tail=tail.next;
     tail.next=head;
     head.prev=tail;
```

```
tail.next=temp;
temp.prev=tail;
tail=tail.next;
tail.next=head;
head.prev=tail;
if(hand_cold.R==0)
{
   temp=hand_cold;
   temp.status=PageStatus.non_resident;
   temp.isresident=false;
   hand_cold=hand_cold.next;
   non_resident_coldpages_size=non_resident_coldpages_size+1;
  //hand_cold.prev.next=hand_cold.next;
  //hand_cold.next.prev=hand_cold.prev;
  hand_cold=hand_cold.next;
  hand_cold.prev=temp;
  temp.next=hand_cold;
  temp.prev.next=temp;
  hand_cold.prev.prev=temp.prev;
  while(hand_cold.status!=PageStatus.cold)
     hand_cold=hand_cold.next;
if(non_resident_coldpages_size>0) //if non resident block is not full
  {  tail.next=temp;
     temp.prev=tail;
     tail=tail.next;
     head.prev=tail;
     tail.next=head;
```

```
                    temp.isinclock=true;
                    non_resident_coldpages_size=non_resident_coldpages_size-1;
                   }
                   else //non resident block is full
                   {
                    temp1=hand_test;
                    //temp1.status=PageStatus.non_resident;
                    temp1.isinclock=false;
                    temp1.prev.next=temp1.next;
                    temp1.next.prev=temp1.prev;

non_resident_coldpages_size=non_resident_coldpages_size+1;
                    hand_test=hand_test.next;
                    while(hand_test.status!=PageStatus.non_resident)
                       hand_test=hand_test.next;
                    tail.next=temp;
                    temp.prev=tail;
                    tail=tail.next;
                    head.prev=tail;
                    tail.next=head;
                    temp.isinclock=true;
                    non_resident_coldpages_size=non_resident_coldpages_size-1;
                   }}}
               tail.next=pt[refpage];
               pt[refpage].prev=tail;
               tail=tail.next;
               head.prev=tail;
               tail.next=head;
```

```
                    pt[refpage].isresident=true;
                    pt[refpage].R=0;
                    pt[refpage].status=PageStatus.cold;
                    cold_pages_size=cold_pages_size+1;
                }}
    }//isinclock
}// end of isresident==false
 else if(pt[refpage].isresident==true) // if accessed page is in memory
{
   if(pt[refpage].status==PageStatus.hot)
   {
      if(pt[refpage].R==1)
      {
         //do nothing
      }
      else
      {
         pt[refpage].R=1;
      }}
   if(pt[refpage].status==PageStatus.cold)
   {
      if(pt[refpage].R==1)
      {
         //do nothing
      }
      else
      {
```

```
                    pt[refpage].R=1;
                }}}}
        System.out.println("--------------------------------");
        System.out.println("The page is:"+refpage);
        showStatus();
        System.out.println("--------------------------------");
        System.out.println();
      } //end of while
   } // end of trace
   void showStatus()
   {
      System.out.println("Total number of pages:"+np);
      System.out.println("Total number of page faults:"+npf);
   }
 }//end of class


// Code for Adaptive CLOCK-Pro omitted here……
```