



**An Evaluation of Page Replacement Algorithm  
Based on Low Inter-Reference Recency Set (LIRS) Scheme  
on Weak Locality Workloads**

**A Dissertation**

**Submitted To  
Central Department of Computer Science & Information  
Technology  
Tribhuvan University  
Kirtipur, Nepal**

**In Partial Fulfillment of the Requirements for the Master's  
Degree of Science  
in  
Computer Science & Information Technology**

**Submitted By  
Bijeta Subedi  
CDCSIT, TU  
(20<sup>th</sup> March, 2012)**



**An Evaluation of Page Replacement Algorithm  
Based on Low Inter-Reference Recency Set (LIRS) Scheme  
on Weak Locality Workloads**

**A Dissertation**

**Submitted To  
Central Department of Computer Science & Information  
Technology  
Tribhuvan University  
Kirtipur, Nepal**

**In Partial Fulfillment of the Requirements for the Master's  
Degree of Science  
in  
Computer Science & Information Technology**

**Submitted By  
Bijeta Subedi  
CDCSIT, TU**

**Supervisor:**

**Prof. Dr. Shashidhar Ram Joshi**

**Co-Supervisor:**

**Mr. Arjun Singh Saud**



**An Evaluation of Page Replacement Algorithm  
Based on Low Inter-Reference Recency Set (LIRS) Scheme  
on Weak Locality Workloads**

**Student's Declaration**

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

**Miss Bijeta Subedi**

**Supervisor's Recommendation**

I hereby recommend that the dissertation prepared under my supervision by **Miss. Bijeta Subedi** entitled "**An Evaluation of Page Replacement Algorithm Based on Low Inter-Reference Recency Set (LIRS) Scheme on Weak Locality Workloads**" be accepted as in fulfilling partial requirement for the completion of Master's Degree of Science in Computer Science & Information Technology.

**Prof. Dr. Shashidhar Ram Joshi**

Head of Department

Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk,  
Lalitpur, Nepal.

**(Supervisor)**



An Evaluation of Page Replacement Algorithms  
Based on Low Inter-Kernel Residency Set (LIRS) Scheme  
on Weak Locality Workloads

Student's Declaration

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

*[Signature]*  
Date: \_\_\_\_\_

Supervisor's Recommendation

I hereby recommend that the dissertation required within my supervision by Anna University entitled "An Evaluation of Page Replacement Algorithms Based on Low Inter-Kernel Residency Set (LIRS) Scheme on Weak Locality Workloads" be accepted as in fulfilling partial requirement for the completion of Master's Degree of Science in Computer Science & Information Technology.

*[Signature]*

Prof. Dr. \_\_\_\_\_  
Head of Department  
Department of Electronics and Computer Engineering, Institute of Anna University,  
Chennai, Tamil Nadu

CDACS17 - Thesis  
Reg. No. 610  
2008-010 (batch)  
2069-02-02

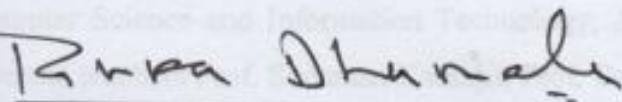




**Tribhuvan University**  
**Institute of Science & Technology**  
**Central Department of Computer Science & Information Technology**

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

**Evaluation Committee**



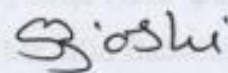
**Assoc. Prof. Dr. Tanka Nath Dhamala**

**Head of Department**

Central Department of Computer Science

& Information Technology

Tribhuvan University, Kirtipur, Nepal



**Prof. Dr. Shashidhar Ram Joshi**

**Head of Department**

Department of Electronics & Computer

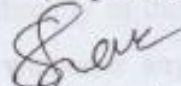
Engineering

IOE, Pulchowk, Nepal



**(External Examiner)**

**(Supervisor)**

  
Prof. Dr. Subarna Shaky

**(Internal Examiner)**

Date:-

## Acknowledgement

I am very happy to complete this thesis "**An Evaluation of Page Replacement Algorithm Based on Low Inter-Reference Recency Set (LIRS) Scheme on Weak Locality Workloads**", which has been performed under Central Department of Computer Science and Information Technology (*Tribhuvan University*), Kirtipur. I am very grateful to my department for assigning me such a laborious work.

First of all I would like to express my gratitude to my supervisor **Prof. 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 extend my heartily acknowledgement to my co-supervisor **Mr. Arjun Singh Saud** who gave me an enthusiastic support from the beginning to the end 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.

I would like to express heartfelt thanks to **Prof. Dr. Onkar Sharma**, Marist College, USA for his inspiration and support for addressing the problems at the beginning.

I would like to thank to our respected Head of Department of Central Department of Computer Science and Information Technology, **Assoc. Prof. Dr. Tanka Nath Dhamala**, respected teachers Prof. Sudarsan Karanjit, Prof. Dr. Subarna Sakya, Mr. Min Bahadur Khati, Mr. Bishnu Gautam, Mr. Dinesh Bajracharya, Mr. Navaraj Poudel, Mr. Jagdish Bhatta, Mr. Yog Raj Joshi, Mr Bikash Balami & Mr. Ashim Ghising of CDCSIT, TU, for providing me such a broad knowledge and inspirations within the period of my study.

Special thanks to my family and members of different 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. Nabin Ghimire, Mr. Sandeep Aryal, Mr. Bishal Gnyawali, Mr. Tej Bahadur Shahi and Mr. Ganga Ram Rimal for supporting me directly and indirectly in this research work. I have done my best to complete this research work. I welcome any type of suggestions, which will improve this research work.

## **Abstract**

The performance of page replacement algorithms used by cache management of OS is very much important. This situation further more complicates due to limitations of faster memory and I/O system. Among various page replacement algorithms LRU is simple and flexible. But the low overhead LRU misbehaves with weak locality of reference. Mainly weak locality workloads can be categorized into sequential pattern, loop with larger than cache size and probabilistic pattern. This weakness of LRU is only due to the bold assumption on recency factor. Recency factor is only not sufficient because frequency factor also plays important role according as the program behavior. Many modifications on LRU have done such as LRU-K, EELRU, LRFU etc. But unlike others LIRS improved the weaknesses of LRU by considering IRR factor, which is logically a combination of recency and frequency factor. IRR factor is also known as reuse distance and can be achieved by using recency value which is equal to number of distinct references between recent correlated access of a particular block. LIRS can be implemented by different approaches based on its principle. One by focusing on its principle called basic LIRS and another LIRS simulated through data structure which focuses on computational complexity. Both of them are evaluated by using variety of weak locality workloads which represents the memory reference pattern during the execution of program.

# Table of Contents

Details	Pages
<b>CHAPTER 1</b>	
<b>Background &amp; Introduction</b>	
1.1 Background	1-11
1.1.1 Memory Hierarchy	1-3
1.1.1.1 Primary Memory	2
1.1.1.1.1 Cache	2
1.1.1.1.2 Register	2
1.1.1.2 Secondary Memory	2
1.1.1.3 Virtual Memory	3
1.1.2 Memory Management	3-6
1.1.2.1 Overlays	3
1.1.2.2 Swapping	3
1.1.2.3 Paging	4
1.1.2.4 Segmentation	5
1.1.3 Paging Algorithm	6-7
1.1.3.1 Fetch Algorithm	6
1.1.3.2 Placement Algorithm	6
1.1.3.3 Replacement Algorithm	6
1.1.4 Performance Metrics	7
1.1.4.1 Page Fault Count	7
1.1.4.2 Hit Rate & Hit Ratio	7
1.1.4.3 Miss Rate & Miss Ratio	7
1.1.5 Memory Design	8-10
1.1.5.1 Sources of Miss	8-9
1.1.5.1.1 Compulsory Miss	8
1.1.5.1.2 Capacity Miss	9
1.1.5.1.3 Conflict Miss	9
1.1.5.1.4 Policy Miss	9
1.1.5.2 Reduction of Miss Ratio	9
1.1.6 Program Behavior	10-11
1.1.6.1 Locality of Reference	10



1.1.6.2 Memory Reference Pattern	10-11
1.1.6.2.1 Cyclic Pattern	10
1.1.6.2.2 Correlated Access Pattern	11
1.1.6.2.3 Temporally Clustered Pattern	11
1.1.6.2.4 Probabilistic Access Pattern	11
1.1.6.2.5 Mixed Pattern	11
1.1.6.3 Working Set	11
1.2 Introduction	12 - 14
1.2.1 Problem Statement	13
1.2.2 Objectives	13
1.3 Motivation	13
1.4 Thesis Organization	14

## **CHAPTER 2**

### **Literature Review & Methodology**

2.1 Literature Review	15-20
2.1.1 OPT or MIN Page Replacement Algorithm	15
2.1.2 Random Page Replacement Algorithm	15
2.1.3 FIFO Page Replacement Algorithm	15
2.1.4 FINUFO Page Replacement Algorithm	16
2.1.5 LRU Page Replacement Algorithm	16
2.1.6 NRU Page Replacement Algorithm	16
2.1.7 MRU Page Replacement Algorithm	17
2.1.8 LFU Page Replacement Algorithm	17
2.1.9 SEQ Page Replacement Algorithm	17
2.1.10 EELRU Page Replacement Algorithm	18
2.1.11 LRFU Page Replacement Algorithm	18
2.1.12 LRU-K Page Replacement Algorithm	18
2.1.13 2Q Page Replacement Algorithm	18
2.1.14 LIRS Page Replacement Algorithm	19
2.1.15 Clock Based Page Replacement Algorithms	19
2.1.16 Various Page Replacement Algorithms	19
2.2 Research Methodology	20

## CHAPTER 3

### Program Development

3.1 Development Methodology & Tools	21-35
3.2 LRU	21-23
3.2.1 Data Structure	21
3.2.2 Algorithm	21
3.2.3 Flowchart	22
3.2.4 Tracing	23
3.3 LIRS Simulated Through Data Structure	24-30
3.3.1 Data Structure	25
3.3.2 Major Function	25
3.3.3 Algorithm	25
3.3.4 Flowchart	27
3.3.5 Tracing	28
3.4 Basic LIRS	30-35
3.4.1 Data Structure	31
3.4.2 Algorithm	31
3.4.3 Flowchart	32
3.4.4 Tracing	33

## CHAPTER 4

### Data Collection & Analysis

4.1 Data Collection	36-39
4.2 Testing	36-37
4.2.1 Test Result of Workload 1	36
4.2.2 Test Result of Workload 2	37
4.2.3 Test Result of Workload 3	37
4.3 Analysis	38

## CHAPTER 5

### Conclusion & Recommendation

5.1 Conclusion	40
5.2 Recommendation	40
<b>References</b>	41-42
<b>Appendices</b>	43-45

## List of Figures

<b>Fig. No.</b>	<b>Caption</b>	<b>Pages</b>
Fig 1	- Computer Memory Hierarchy	1
Fig 3.1	- Flowchart of LRU Algorithm	22
Fig 3.2	- LRU Queue at Virtual Time 1-10	23
Fig 3.3	- General LIR vs. HIR Transition Diagram	24
Fig 3.4.1	- LIR vs. Resident HIR Transition Diagram	24
Fig 3.4.2	- LIR vs. Non-Resident HIR Transition Diagram	24
Fig3.5	- Flowchart of LIRS Simulated Through Data Structure	27
Fig3.6	- State at Virtual Time 1-10	28-30
Fig3.7	- Flowchart of Basic LIRS Algorithm	32
Fig3.8	- Stack S at Virtual Time 1-10	33-35
Fig4.1	- Graph for Workload1	38
Fig4.2	- Graph for Workload2	38
Fig4.3	- Graph for Workload3	39

## List of Tables

<b>Table No.</b>		<b>Caption</b>	<b>Pages</b>
Table 3.1	-	IRR Calculation for Virtual Time 2-11	33-35
Table 4.1	-	Test Result of Workload 1	36
Table 4.2	-	Test Result of Workload 2	37
Table 4.3	-	Test Result of Workload 3	37

## List of Abbreviations

2Q	-	Two Queue
ARC	-	Adaptive Replacement Cache
AFC	-	Application/File-level Characterization
AFPR	-	Adaptive Fuzzy Page Replacement
CAR	-	Clock with Adaptive Replacement
CLOCK Pro	-	Clock with improvement
CPU	-	Central Processing Unit
DEAR	-	DEtection based Adaptive Replacement
EELRU	-	Early Eviction Least Recently Used
FINUFO	-	First In Not Used First Out
FIFO	-	First In First Out
FPR	-	Fuzzy Page Replacement
HIR	-	High Inter-reference Recency
HIRS	-	High Inter-reference Recency Set
I/O	-	Input Output
IRG	-	Inter- Reference Gap
IRR	-	Inter- Reference Recency
LFU	-	Least Frequently Used
LIR	-	Low Inter-reference Recency
LIRS	-	Low Inter-reference Recency Set
LRFU	-	Least Recently Frequently Used
LRU	-	Least Recently Used
MMU	-	Main Memory Unit
MRU	-	Most Recently Used
NRU	-	Not Recently Used
OPT or MIN	-	OPTimum or MINimum
OS	-	Operating System
PPF	-	Page Fault Frequency
RAM	-	Random Access Memory
ROM	-	Read Only Memory
SEQ	-	SEQunetial
TLB	-	Translation Look-aside Buffer

# Chapter 1

## BACKGROUND & INTRODUCTION

### 1.1 Background

#### 1.1.1 Memory Hierarchy

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.

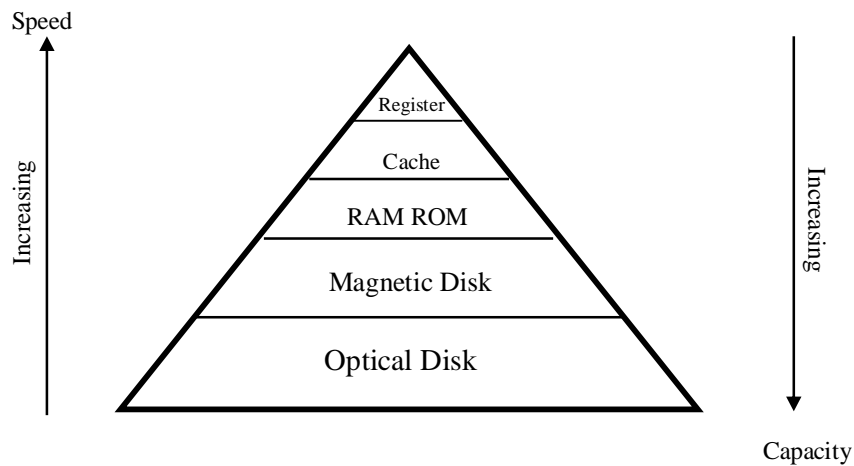


Fig 1.1 Computer Memory Hierarchy

Figure 1.1 Computer 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.

### **1.1.1.1 Primary Memory**

Primary memory is only the memory which can be referenced directly. It is also known as internal or main memory. It is made up of semiconductor material and can be accessed randomly. For example register, RAM, ROM, cache etc. It is faster and expensive memory that lies at the top most level of memory hierarchy. Since primary memory is volatile in nature computer system backups the data into secondary storage.

#### **1.1.1.1.1 Cache**

The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. Cache acts as bridge between processor and RAM since speed of processor is still faster compared to speed of RAM. Generally computer system consists of different levels of cache that are L1 cache and L2 cache. L1 cache is internal cache nearby register and L2 cache is external cache nearby RAM. L1 cache is faster than L2 cache. If L3 cache is available then it acts as earlier L2 cache. Hence L2 works as intermediate cache between L1 cache and L3 cache. Increasing the level of cache doesn't always increase the overall performance. Up to limited cache level the performance gain can be achieved. If there are more levels of cache, access time will increase due to swapping the blocks back and forth. Hence after crossing certain limitation of cache level overall performance slows down instead of increasing.

#### **1.1.1.1.2 Register**

Register is the fastest memory in which processing is actually performed. It is inbuilt inside CPU. In general, registers are temporary storage in the CPU that holds the data the processor is currently working on, while cache holds the program instructions and the data the program requires. Finally, there are generally only a few numbers of registers available on a processor. For example Intel chips have 6 general purpose registers and several specialized registers including a base register, stack register, flags register, program counter, and some addressing registers.

### **1.1.1.2 Secondary Memory**

Secondary memory is taken as the backup memory. It consists of massive volume of data. Comparatively it is cheaper, slower and less reliable. Secondary memory is external memory such as hard disk, optical disk, pen drive, flash cards etc.

### **1.1.1.3 Virtual Memory**

Fotheringham 1961[2], 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 is a service provided by an OS that allows execution of programs larger than available physical memory. Virtual memory plays a vital role to overcome limited primary memory. Handling virtual memory is one of the important issues of today's computer system.

### **1.1.2 Memory Management**

Memory management and organization has been one of the most important factors that influence performance of OS. It has been studied for many years. Memory management systems are of two classes one which move processes back and forth between main memory and disk during execution and other which does not.

Actually memory management is done by memory manager or memory management unit, which is handled by OS to manage memory hierarchy. The main job of memory management unit is to keep track of processes currently being executed. It keeps track which part of memory is currently in use and which does not. It also allocates memory for a process when required and deallocates memory when work is temporarily finished. It manages memory for a process to load and also manages extra memory that is virtual memory if it is too small to hold for the required process.

#### **1.1.2.1 Overlays**

Early age, too big programs that couldn't fit into available memory are usually split into pieces called overlays, which should be done manually. These overlays are swapped in for execution of programs and swapped out after execution. This makes the programmer to perform a tedious job which was time consuming and boring. Similar concept is used now also but besides partitioning manually, OS keeps track of part of program currently in use in main memory and rest on backing storage.

#### **1.1.2.2 Swapping**

A process must be in memory to be executed however it can be swapped out temporarily to a backing store and then brought back into memory for continued execution. Swapping is a technique in which memory blocks are partitioned into variable size. Here blocks are swapped back and forth which is done by OS. The memory block should fit in available free



space or hole for execution. There are several strategies to fit the hole like first fit, best fit, worst fit etc.

Using variable partition method or swapping leads to fragmentation. Fragments are the small holes which is not suitable for any process to fit. Even though there is free memory space available, it is wasted. Internal fragmentation occurs due to creation of hole while allocating memory to a process slightly larger than required memory. Swapping leads to number of small holes after a long duration of execution of programs and available spaces are non-contiguous due to which external fragmentation occurs. Even the sum of free spaces are sufficient than required memory to fit OS could not execute the process. Temporary solution to external fragmentation is compaction which is a process of moving occupied used memory toward one end so as to sum up holes into contiguous memory. But compaction is costlier hence due to many pitfalls variable partition technique is not so used.

### **1.1.2.3 Paging**

Paging is a better solution than swapping because it eradicates the problem of fragmentation which is only due to partition of fixed size memory blocks. Paging is one of the techniques that organize virtual storage. The address referenced by running process is called virtual or logical address whereas the range of address it can reference is called virtual or logical address space. The address available in primary storage is called real or physical address whereas the available range of address is called real or physical address space. Even though a process references only virtual address, the process must run on available real storage. So for every reference Main Memory Unit (MMU) maps logical address into corresponding available physical address for that page table is maintained. A page is fixed sized unit of virtual address space whereas a frame is fixed sized unit of real address space. Generally, size of frame is equals to size of page. If a requested page is unavailable in primary storage page fault occurs [2]. A page table contains record of each page with page frame number. Also each page entry consists of bits like reference bit, caching disabled bit, protection bit, modified bit and additional information like protection bits. Protection bit is a 3 bits information containing rwx where r is for read, w is for write and x is for execute. During page fault MMU notices that the page is unmapped and causes the CPU to trap. Trap is generated by OS to stop CPU until required page is not available. Then OS picks little used page frame as chosen by page replacement policy. If it has dirty bit then the contents are written otherwise if it has clean bit then nothing is written back to secondary storage. Thus

the required page is placed into freed frame. Then after successful mapping, trap was restarted and the process is continued. Still there are several issues while implementing paging technique. Page table can be extremely large and page mapping must be any how fast for faster performance. Execution speed depends upon rate at which CPU can fetch data and instruction out of memory. Additional memory reference is required to access the page table and for mapping. Translation Look-aside Buffer (TLB) is a solution for this problem which can be part of MMU. Similar to page table entry each TLB entry contains valid bit, modified bit, protection bit, virtual page number, page frame number and additional information. TLB fault may occur if the requested page is unavailable this can be detected by checking TLB entry. Then trap is generated if it is unavailable in page table entry as earlier it is brought back to primary storage replacing one of the pages [3].

#### **1.1.2.4 Segmentation**

Segmentation is a technique in which virtual address space is divided into several chunks of segments. Paging doesn't cover the programmer's point of view during execution of a program. Each segment of a program can individually grow and shrink unpredictably. Similarly each segment of a virtual address space segmented using segmentation can grow and shrink individually without affecting other. Hence having two or more virtual address spaces may be much better than having one for managing such segments of code during execution. Segmentation involves the relocation of variable sized segments into the physical address space. Generally these segments are contiguous units and are referred to in programs by their segment number and an offset to the requested data. Although a segmentation approach can be more powerful to a programmer in terms of control over the memory, it can also become a burden, as suggested by [4]. Efficient segmentation relies on programs that are very thoughtfully written for their target system. Even assuming best case scenarios, segmentation can lead to many problems. External fragmentation is the term that is use to denote pieces of memory between segments which may collectively provide a useful amount of memory. But they are useless because of their non-contiguous nature. Since segmentation relies on memory that is located in single large blocks, it is possible that enough free space is available to load a new module, but cannot be utilized. Segmentation may also suffer from internal fragmentation if segments are not variable-sized. Contrarily, paging provides somewhat easier interface for programs rather than segmentation. Because paging operations are more easier and transparent than segmentation.

### **1.1.3 Paging Algorithm**

#### **1.1.3.1 Fetch Algorithm**

Fetch algorithm initially identifies the requested page block. Paging algorithm can be categorized into two major groups. They are demand paging and anticipatory paging. Demand paging algorithm waits for a page requested by a running process. But anticipatory or pre-paging algorithm guesses which pages are needed before they are requested. Generally paging mechanism will not have prior knowledge of the page reference stream or the known order of pages requested in. This causes many systems to employ a demand fetch approach, where a page fault notification is the first indication that a page must be moved into the physical memory. Hence demand paging algorithm is much more effective in real systems than pre-paging algorithm [2]. Demand fetching algorithm always fetches a page that has been requested during a page fault. But pre-fetching is done by using some heuristic before the occurrence of page fault.

#### **1.1.3.2 Placement Algorithm**

Placement algorithm decides where to put the fetched page in available free storage. Initially if placement algorithm allows fully associative then OS can place the requested page anywhere using any algorithm. After a cache is fulfilled then placement policy is static that means a requested page is placed in place of removed victim page. A victim page is always replaced by required page which is chosen by replacement policy used in that particular system. In case of partially associative memory mapping, placement algorithm is restricted only for certain memory location.

#### **1.1.3.3 Replacement Algorithm**

Replacement algorithm identifies the victim page and replaces it by fetched page because of lack of primary storage. After a primary storage is fulfilled one of the block must be replaced for execution of the requested page. The replaced block is also called victim block.

Static page replacement algorithm shares frames equally among all processes such as FIFO, LRU, MRU, random, optimal etc. Adaptive page replacement algorithm replaces page according as the page reference pattern observed for example SEQ, EELRU, LRFU, LIRS, ARC etc. But dynamic page replacement algorithm shares frames according to need rather than equality among all processes such as working set page replacement algorithm. Some processes need more frames than others and sometimes a process needs more frames than

other times so in this case dynamic policy is applied for better performance but it is more complex than static one. For such decision calculated page fault frequency and threshold limit is compared [5].

Also page replacement algorithm can maintain global and local policy. Global policy selects a replacement from the set of all available frames. Local policy selects a replacement from the processes own set of frames.

#### **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.

##### **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 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.

##### **1.1.4.3 Miss Rate & Miss Ratio**

Miss rate (mr) can be calculated by using formula

$$mr = 100 \times ( (\#pf - \#distinct) / (\#refs - \#distinct) )$$

where #pf is the number of page faults, #distinct is the number of distinct pages referenced and #refs is the total number of pages referenced [6]. Miss Ratio is the fraction number of page fault and reference ignoring the distinct references.

### **1.1.5 Memory Design**

Invention of faster, cheaper and smaller device is only possible due to research in memory design from many years. Every year performance speed of memory chip has been improving. Moore's 1965 concluded that "Computer memory increases geometrically not linearly.", for increasing such large performance gap there are three major logical policies of memory design.

Firstly improvement can be achieved by making the common case fast. It means most accessing memories are copied to fastest memory like cache. So that it is always available and directly used without any delay.

Secondly improvement can be achieved by tracking locality of reference in terms of program behavior. The term locality refers to the tendency of referencing particular memory location frequently rather than other memory location. Rule of Thumb- "Most of program spends 90% of execution time in 10% of source code", which also shows locality of reference [7].

Finally improvement can be achieved by taking advantage of parallelism. Number of processes can run simultaneously at a time in a processor which is only possible due to parallelism. Otherwise user should wait for a completion of one process to run other, as historical computer systems. Hence process must be switched to get chance for busy processor.

There are some issues related to cache design like block identification policy, block placement policy, block replacement policy and write strategy. Block identification policy identifies the required block to fetch. Block placement policy identifies where to place fetched block in cache. Block replacement policy identifies a victim block that is to be replaced. Before replacing a block it must be write back if it is modified which is handled by write strategy.

#### **1.1.5.1 Sources of Miss**

##### **1.1.5.1.1 Compulsory Miss**

Miss may occur due to several reasons. First time when the block is referenced, it is always miss which is due to empty volatile cache. This type of miss is known as compulsory miss. Such type of miss can't be reduced but pre-fetching can be done initially.

### **1.1.5.1.2 Capacity Miss**

Some blocks can't be kept in available primary storage due to limited memory space. Hence a block in cache must be replaced to access new unavailable block. Next time if the older block is accessed miss occurs. Here blocks being discarded are later on retrieved. Occurrence of miss due to limited storage capacity is known as capacity miss. There is no any permanent solution for this miss because memory capacity is always limited.

### **1.1.5.1.3 Conflict Miss**

Too many main memory blocks mapped to the same cache set results conflict miss. This is due to direct and set associative mapping. Conflict miss doesn't occur in case of fully associative mapping. This type of miss is also known as collision miss or interference miss.

### **1.1.5.1.4 Policy Miss**

If required block is unavailable in faster memory then the required block is brought back to faster memory replacing some block. This victim block is chosen by page replacement policy. This block may be needed later causes policy miss which is consequence of replacement policy [8].

### **1.1.5.2 Reduction of Miss Ratio**

There are several ways to reduce miss ratio. Miss ratio can be decreased by reducing number of page fault. Right decision of the page replacement policy by choosing worth page reference decreases miss ratio. If simple strategy is used that is by default random page replacement algorithm then miss ratio may increase or decrease randomly which unpredictable. Hence a right policy must be decided for reducing miss ratio which indicates performance gain. A victim page should be like that which is not accessed in future. Reducing policy miss is the best idea. But if the victim page is accessed in future then page fault occurs hence performance decreases due to miss penalty. It seems page fault decreases while number of page frame increases. But some algorithm suffers from Belady's Anomaly [9], which verified strange situation in which page fault increased while increasing number of page frame. Such anomaly will not occur, if any algorithm with allocation of size  $m$  has pages that are guaranteed to be a subset of the allocation  $m + 1$ . Stack based algorithm satisfies the mentioned situation.

Also increasing associativity, adding victim cache, hardware or software pre-fetching, code optimization, increasing the block size etc. are the solutions for reducing miss ratio [7]. Increasing the block size decreases page fault which favors locality. Sometimes highly increasing the block size increases page fault frequency because all available locations of larger block may not be referenced and due to larger size memory locations required are unavailable. Sometimes highly decreasing the block size still increases the page fault frequency because it violates locality. In both cases time consumption is high because of swapping the block back and forth instead of increasing overall performance. Hence appropriate block size with right decision policy decreases miss ratio.

### **1.1.6 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.

#### **1.1.6.1 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 and totaling etc. supports this assumption. Spatial locality is based on space, it 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 [2].

#### **1.1.6.2 Memory Reference Pattern**

##### **1.1.6.2.1 Cyclic Pattern**

Memory locations that are referenced repeatedly in a same order can be viewed as cyclic pattern. Loop generates cyclic pattern. For example if M1, M2, M3, M4 be the memory blocks used then cyclic pattern can be taken as M1, M2, M3, M4, M1, M2, M3, M4, M1, M2, M3, M4, M1, M2, M3, M4, M1, M2, M3, M4, M1, M2, M3, M4, M1, M2, M3, M4 when loop executes six times.

#### **1.1.6.2.2 Correlated Access Pattern**

Access of memory location at particular place then repeated after some duration, such memory reference pattern can be viewed as correlated pattern. Sequential Scan also generates correlated pattern. For example if M1, M2, M3 be the memory blocks frequently used then correlated pattern can be taken as M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M1, M2, M3, M4, M13, M14, M15, M16 when two times correlated access is performed.

#### **1.1.6.2.3 Temporally Clustered Pattern**

A temporally clustered reference pattern has the property that a block referenced more recently will be referenced sooner in the future for some duration. For example temporally clustered pattern can be taken as M1, M2, M1, M3, M2, M4, M3, M1, M2, M5, M6, M7, M8.

#### **1.1.6.2.4 Probabilistic Pattern**

When particular memory block has a stationary reference probability and all other blocks are accessed independently without any associated probabilities, such memory reference pattern can be viewed as probabilistic pattern [6]. Such pattern also generates temporal clustering. For example if M1, M2 be the memory blocks frequently used then probabilistic pattern can be taken as M1, M2, M3, M4, M6, M7, M1, M2, M3, M21, M22, M23, M1, M2, M3, M2, M14, M15, M16, M1.

#### **1.1.6.2.5 Mixed Pattern**

Mixed pattern is generated by the occurrence of cyclic pattern, correlated pattern, temporally clustered pattern and probabilistic pattern. For example of mixed pattern can be taken as M1, M2, M3, M4, M5, M6, M7, M8, M9, M10, M11, M12, M1, M2, M3, M4, M5, M6, M7, M1, M8, M2, M20, M21, M1, M22, M23, M10, M1, M2, M3, M4.

#### **1.1.6.3 Working Set**

Working set is a collection of pages that an active program or a process is actively referencing. If a working set is available in cache then page fault will not occur. Number of page fault increases extensively during change in locality. This is may be due to switching of working set. The term thrashing means regular occurrence of page fault. Here CPU spends more time on page fault handling rather than execution. Working set is the main idea behind dynamic page replacement algorithm [2].



## 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. Recency factor is the virtual time difference between the current time and last time when the oldest block is accessed. 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.

There are different modified versions of LRU algorithm. Among them LIRS page replacement algorithm identifies and eradicates the misbehaviors of LRU on weak locality of references. LRU only uses recency factor whereas LIRS uses additional factor called reuse distance for page replacement. Reuse distance or inter reference recency (IRR) of a block is equal to number of distinct pages accessed between recent consecutive or correlated access of that particular block like IRG. Strong part of LIRS algorithm is the IRR value which maintains recency as well as frequency factor. LIRS algorithm [10] uses two sets of pages based on IRR. Set of pages with low IRR value is taken as hot block and called low inter-reference recency set (LIRS). Set of pages with high IRR value is taken as cold block and called high inter-reference recency set (HIRS). Blocks that can be most probably used in future are taken as hot blocks whereas blocks that may not be used in near future are taken as cold blocks. Hence HIR blocks are always replaced and LIR blocks are never replaced. LIR page is always available in cache whereas HIR page may or may not be available in cache. HIR page that is available in cache is called resident HIR and HIR page that is not available in cache is called non-resident HIR. Hence a page which is accessed first time is taken as non-resident HIR. Fixed number of LIR block and resident HIR block is used which is equal to 99% and 1% of cache size respectively. Partition of cache doesn't obstruct the overall performance. These parameters can be tuned but the consideration is found to be valid for handling weak locality workloads as shown by sensitivity study.

Victim block is always predicted from HIRS block which has high recency value that is the oldest block. It is very easy to search oldest block by maintaining recency stack. Also IRR value is computed which is equals to earlier recency value. Promotion and demotion policy is also used for utilizing history information correctly maintaining HIR and LIR partition size.

### **1.2.1 Problem Statement**

LRU shows anomalous behavior with weak locality of reference. The access patterns of weak locality workloads can be categorized into three different groups.

a) Sequential Scan

Sequential scan pattern consists of many distinct pages where at least one of working set repeatedly occurs. In such case LRU may replace pages that occur repeatedly because of their low recency factor which increases page fault.

b) Loop with working set larger than cache size

In case of loop with working set larger than cache size, LRU always replaces page before they are accessed for the next time. Hence LRU makes 100% page fault.

c) Pages with irregular frequencies (Probabilistic)

LRU can't discriminate between irregularly occurring pages and other pages because it only replaces the page on the basis of recency factor.

### **1.2.2 Objectives**

Primary concern of this thesis work is to successfully handle weak locality workloads, rather than tuning its parameter, because the parameters are found to be efficient for such workloads. It should be noted that page replacement algorithm is just not concerned with memory management in OS. But it is used in different computing device consisting cache and also databases and web proxies where faster memory is essential. But our work only focuses on issues related to general OS. Objectives of this dissertation work are as follows:

- To study the improvement of LIRS algorithm on weak locality workloads over LRU algorithm.
- To modify and simulate LIRS algorithm without violating its logical premise and to compare change in PFF.

### **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. Reading related research papers it is found that LIRS can fulfill these criteria. It is successfully implemented in different fields [11]. It is better if LIRS could store deeper history information. LIRS 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 LIRS. Some more chapters are remaining which clarifies the topics LIRS 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. 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. It includes detail design of the program. Also it includes details about the data structures and programming language used to build the simulation. Chapter 4 consists of data collection and analysis part which includes details about generating traces of memory references that shows trace driven input. The output results with several analyzing graphs which are tested for weak locality workloads. Chapter 5 consists of conclusion of this whole dissertation work and the recommendation which shows guidelines for further research.

## Chapter 2

### LITERATURE REVIEW & METHODOLOGY

#### 2.1 Literature Review

##### 2.1.1 OPT or MIN Page Replacement Algorithm

Various memory management techniques have been used from the beginning for the improvement of performance. Belady [3] in 1966 developed optimal page replacement algorithm called OPT or MIN. His algorithm depends upon principle of optimality which states "To obtain optimal performance the page to replace is the one that will not be used again for the furthest time into the future." His optimal algorithm is not applicable for real implementation because OS doesn't know which pages will be used before execution. Hence it is used as a benchmark for measuring effectiveness of other page replacement algorithms. OPT Replacement algorithm replaces page that will not be used for the longest period of time by computing maximum forward distance.

##### 2.1.2 Random Page Replacement Algorithm

Random page replacement algorithm can replace any page randomly during page fault. Page fault decreases if the replaced page is cold and vice versa. Hence the algorithm shows unpredictable results. Here each page frame involved has an equal chance of being chosen, without taking into consideration of the program behavior. Due to its randomness the behavior of this algorithm is obviously random and unreliable. With most reference streams this method produces an unacceptable number of page faults, as well as victim pages being thrashed unnecessarily. Hence deploying random page replacement algorithm is not an effective technique [1].

##### 2.1.3 FIFO Page Replacement Algorithm

Fist-In-First-Out page replacement algorithm replaces oldest page during page fault. Initially queue is filled by inserting page reference from the tail. When the queue is full new reference is inserted from tail and old reference is evicted from the head. FIFO is simple but suffers from Belady's Anomaly. This strange situation is already discussed in section 1.1.5.2. Like random page replacement algorithm, FIFO still does not take advantage of locality trends. But it can be modified very easily.

#### **2.1.4 FINUFO Page Replacement Algorithm**

A modification to FIFO that makes its operation much more useful by taking advantage of program behavior is First-In Not-Used First-Out (FINUFO). The only modification here is that a single bit is used to track whether or not a page is fresh in the FIFO queue. This referenced bit is then used to determine whether the page is victim or not. A fresh reference has bit 0 whereas referenced one has bit set to 1. A victim is selected by giving priority to reference 0. If every active page has been referenced, victim page is selected by taking locality into consideration. The situation can be tackled by resetting all the bits. In a worst case scenario this could cause minor and temporary thrashing.

#### **2.1.5 LRU Page Replacement Algorithm**

As recent past is a good indicator of the near future. The algorithm considers that a page that is just now used will probably be used again very soon, and a page that has not been used for a long time, will probably remain unused. Recency is evaluated by maintaining LRU stack that is a sorted list on the basis of virtual time, which is the only factor for replacement. When page fault occurs, the page that has been unused for the longest time is evicted. Thus LRU is simple and easy to implement. It can adapt faster according as program behavior. LRU like algorithm doesn't suffer from Belady's Anomaly as FIFO.

LRU shows more page faults in case of weak locality workloads, which can be reduced by applying three major techniques. By taking user-level hints, applications are hinted during caching and pre-fetching which rely on users understanding of data access patterns. Hence such work is only suitable for working manually, which eradicates burden of programmer. Detection and adaptation of access regularities is performed case by case in different algorithms like SEQ, EELRU, DEAR, AFC, UBM etc. Tracing and utilizing deeper history information is performed in different algorithms like LRFU, LRU-K, 2Q, ARC etc. including LIRS. For such deeper history information high implementation cost, and runtime overhead is required [10].

#### **2.1.6 NRU Page Replacement Algorithm**

Pages are categorized into four classes in Not Recently Used (NRU) algorithm. Class 0 contains pages that are neither referenced nor modified. Class 1 contains pages that are modified but not referenced. Class 2 contains pages that are referenced but not modified and

Class 3 contains pages that are modified as well as referenced. During page fault NRU evicts any page from the lowest class [1].

### **2.1.7 MRU Page Replacement Algorithm**

Most Recently Used (MRU) algorithm 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.

### **2.1.8 LFU Page Replacement Algorithm**

Least Frequently Used (LFU) selects a victim page that has not been used often in the past. Instead of using a single recency factor as LRU, LFU defines additional information of frequency which is equal to number of times the page referenced. This frequency is calculated throughout the reference stream by maintaining counting information. Frequency count leads to serious problem after a long duration of reference stream. Because when the locality changes, reaction to such certain change will be very slow. Assuming that a program either changes its set of active pages or terminates and it may be 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, the pages swapped out will most likely be needed again soon which leads to thrashing. One way to remedy this is to reset frequency counter each time a page is loaded, rather than being allowed to increase indefinitely throughout the execution of the program. LFU still tends to respond slowly to change in locality of reference.

### **2.1.9 SEQ Page Replacement Algorithm**

The SEQ algorithm [12] can be considered as an adaptive version of LRU that tries to correct the slow performance caused by the presence of sequential memory accesses. When it identifies one or more memory reference sets to numerically adjacent addresses, the algorithm adopts a pseudo-MRU replacement strategy, otherwise maintaining the original LRU criterion by detecting memory reference pattern.

### **2.1.10 EELRU Page Replacement Algorithm**

Some algorithms use recency as history information like LRU and Most Recently Used (MRU). LRU is suitable for good locality of reference whereas MRU is somewhat suitable for weak locality of workloads. These two algorithms can be tuned to form adaptive algorithm called Early Eviction LRU (EELRU) [13]. EELRU is based only on the positions on the LRU queue that contains information of most of the memory references. This queue is only a representation of the main memory using the LRU stack. EELRU detects sequential access patterns analyzing the reuse of pages. One important feature of this algorithm is the detection of non-numerically adjacent sequential memory access patterns. Two tunable parameters used are early eviction point and late eviction point.

### **2.1.11 LRFU Page Replacement Algorithm**

Least Frequently Used (LFU) algorithm uses frequency factor for page replacement. LRU and LFU are tuned to form adaptive algorithm called Least Recently Frequently Used (LRFU) [14] that considers both recency and frequency factors. Depending upon the access pattern the parameter of LRFU can be adapted.

### **2.1.12 LRU-K Page Replacement Algorithm**

LRU - K [15] 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  $K^{\text{th}}$  most recent reference to page  $p$  where reference string known up to time  $t$  ( $r_1, r_2, \dots, r_t$ ). 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.

### **2.1.13 2Q Page Replacement Algorithm**

2Q [16] algorithm quickly removes sequentially and cyclically referenced block with after a long interval. The algorithm uses special buffer queue  $A_{1in}$  of size  $K_{in}$ , ghost buffer queue  $A_{1out}$  of size  $K_{out}$  and the main buffer  $A_m$ . 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. Some algorithm maintains multiple queues for more deeper history information.

### **2.1.14 LIRS Page Replacement Algorithm**

Another important algorithm is LIRS which is already described in section 1.2. Its objective is to minimizing the deficiencies presented by LRU using history information called IRR that represents the number of different pages accessed between the last two consecutive accesses to the same page. Some issues related to LIRS are:

1. How to effectively utilize multiple sources of access information?
2. How to dynamically and responsively distinguish blocks by comparing their possibilities to be referenced in the near future?
3. How to minimize implementation overhead? [10].

### **2.1.15 CLOCK Based Page Replacement Algorithm**

The clock-based approximations, such as CLOCK [17], CLOCK-Pro [18] and CAR [19] usually cannot achieve the high hit ratio compared to their corresponding original algorithms like LRU, LIRS, ARC [20] respectively. Clock based approach organize pages into circular list and uses a reference bit or a reference counter to record access information for each page. When a page is hit in the cache, the clock-based approximations set the reference bit or increment the counter, instead of modifying the circular list. As a lock is not required for these operations, their caching performance is scalable. However the clock-based approximations can record only limited history access information. The information checks whether a page has been accessed or how many times it has been accessed but not in what order their accesses occur. The lack of richer history information can hurt their hit ratios. Many replacement algorithms do not have clock based approximations since the access information they need cannot be approximated by the clock structure [21].

### **2.1.16 Various Page Replacement Algorithms**

Three other algorithms DEAR [22], AFC [23] and UBM [24] analyze the memory accesses looking for some specific patterns including sequential accesses. They adopt a different replacement strategy for each pattern. For example DEAR applies MRU for sequential accesses and LRU or LFU for other patterns. Recent adaptive algorithms use artificial intelligence techniques in order to adapt according as reference pattern. For example the FPR [25] and FAPR [26] algorithms apply fuzzy inference techniques to manage the replacement priorities of the resident pages. These algorithms bring important conceptual benefits to the traditional page replacement algorithms, but they present more complex implementations. In



many cases additional data structures to hold non-resident pages which increases space requirements. Some algorithms require data update in every memory access, making impracticable its real implementation.

## **2.2 Methodology**

Research is a careful study performed to find out new things in a systematic way. In a scientific method of research at first problem is formulated then according as collected input data, output information is analyzed and finally the information is generalized [27]. This dissertation work is truly scientific and flows in the same way. 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. The main exploration of this dissertation focuses on LIRS algorithm. All data collected are primary in form, 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 of workloads.

## **Chapter 3**

### **PROGRAM DEVELOPMENT**

#### **3.1 Development Methodology and Tools**

The simulator is built by using incremental approach. At first LRU algorithm is simulated by using LRU stack algorithm. The LRU stack automatically maintains recency factor. Information of recently referenced block is available in top of stack and the oldest in bottom of stack. Every time when the block is accessed it is kept in top of stack. LIRS algorithms are also implemented by using same stack algorithm and additional features are added to keep track of IRR. C-language is used for simulating LRU and two approaches of LIRS.

#### **3.2 LRU**

Implementing LRU is an idea to keep track of recency. Bringing the recent one in front means also keeping the older one at bottom of stack in a sorted order on the basis of recency. Initially when the stack is empty then a new reference is inserted from top of stack. After then references are inserted from top in a sorted order on the basis of virtual time. But if any reference is inserted next time again then it is brought to front.

##### **3.2.1 Data Structure**

Implementing LRU by using stack algorithm is quite easy. LRU queue is only a representation of the main memory using the LRU model, ordered by the recency of each page. Here queue is used to keep track of recency instead of stack. Because removing rear from queue is easier operation than removing bottom of stack. The LRU queue keeps information of recently referenced block in front of queue and the oldest in rear of queue. Every time when the block is accessed it is inserted in front of queue and if the reference is not available in queue rear is removed, otherwise queue is maintained in same order.

##### **3.2.2 Algorithm**

STEP 1: Begin

STEP 2: If X is available in Queue then move X to front of Queue. Hit occurs

STEP 3: If X is not available in Queue, miss occurs then insert X to front of Queue.

STEP 4: Before inserting X, if Queue is full in STEP 3, then remove rear.

STEP 5: End

### 3.2.3 Flowchart

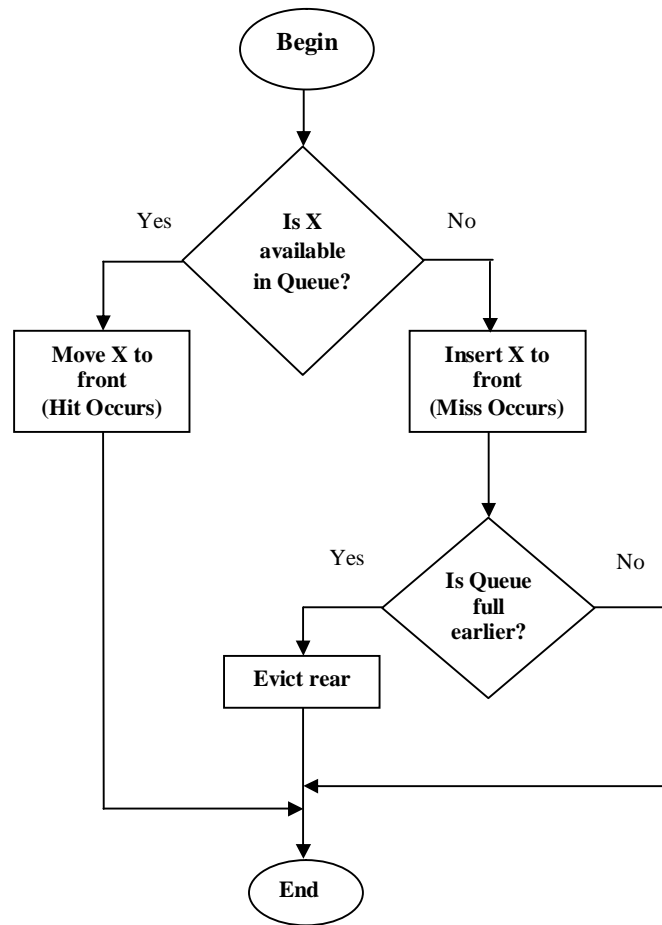


Fig 3.1 Flowchart of LRU Algorithm

### 3.2.4 Tracing

Cache Size: 3

Input References: A D B C B A D A E B

Number of Distinct References: 5

Total Number of References: 10

Accessing A:

A		
Front		

page fault

Fig 3.2.1 LRU Queue at Virtual Time 1

Accessing D:

D	A	
front		rear

page fault

Fig 3.2.2 LRU Queue at Virtual Time 2

Accessing B:

B	D	A
front		rear

page fault

Fig 3.2.3 LRU Queue at Virtual Time 3

Accessing C:

C	B	D
front		rear

page fault

Fig 3.2.4 LRU Queue at Virtual Time 4

Accessing B:

B	C	D
front		rear

Fig 3.2.5 LRU Queue at Virtual Time 5

Accessing A:

A	B	C
front		rear

page fault

Fig 3.2.6 LRU Queue at Virtual Time 6

Accessing D:

D	A	B
front		rear

page fault

Fig 3.2.7 LRU Queue at Virtual Time 7

Accessing A:

A	D	B
front		rear

Fig 3.2.8 LRU Queue at Virtual Time 8

Accessing E:

E	A	D
front		rear

page fault

Fig 3.2.9 LRU Queue at Virtual Time 9

Accessing B:

B	E	A
front		rear

page fault

Fig 3.2.10 LRU Queue at Virtual Time 10

Total Number of page fault: 8

### 3.3 Basic LIRS

Sum of size of HIRS and size of LIRS is equals to size of cache. HIR block that may be resident or non-resident can be promoted to LIR block. At the same time to maintain the LIRS and HIRS size, oldest LIR block must be demoted to HIR-resident block. Then one of the resident HIR block becomes the victim one. The following section contains the clarification of LIRS scheme. The major function stack pruning is illustrated with diagrams in the section 3.3.5.and promotion demotion policy is shown in the fig3.3. Figures 3.4 shows the specific promotion demotion policy among LIR which is always resident, resident HIR and non-resident HIR, so as to maintain partition size.

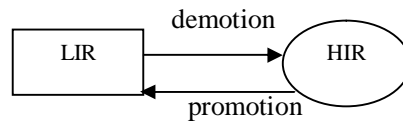


Fig 3.3 General LIR vs. HIR Transition



Fig 3.4.1 LIR vs. Resident HIR Transition Diagram

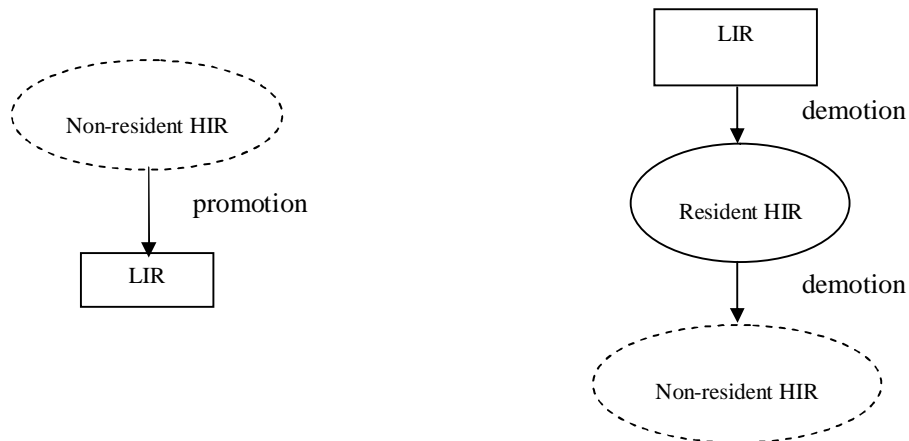


Fig 3.4.2 LIR vs. Non-resident HIR Transition Diagram

### **3.3.1 Data Structure**

Stack S contains page reference accessed. Its main purpose is to maintain recency value. As we move toward bottom recency factor increases. Bottommost one is always LIR block, which is the oldest block having higher recency factor and topmost one is the recent block having recency factor equals to zero. Each stack node contains information about reference block. Here information of every page reference is not available in stack S due to the major event stack pruning. Some information is also available in queue Q and some outdated information is left.

Queue Q contains collection referenced page that are available in cache. But it only tracks resident HIR blocks. Hence size of HIR cache partition determines the size of Queue Q. The block in the Queue can be removed from anywhere if it is promoted to LIR. In that case the bottom most one LIR block of stack is inserted to end of Q then it becomes resident HIR as it is now in Queue. Block in the front of Queue is removed, now the removed block demotes to non-resident HIR. Comparing IRR and recency value is automatically done by the use of Q which increases performance.

### **3.3.2 Major Function**

The major function stack pruning is conducted during status change. Bold assumption of the algorithm is that LIR block always contains in the bottom. Bottom of stack S is always LIR block. While changing status, the page in bottom of stack S is demoted to HIR resident for that it is kept in queue Q. At that time next LIR bottom is chosen which is nearer from bottom of stack S and all other HIR bottom are removed one by one. Information of thus removed HIRs is available in queue Q, if it is resident. Stack pruning is also conducted if the accessed block X is the bottom LIR because recent block is always moved to top of stack S. Stack pruning decreases the size of stack hence the stack doesn't keep track of outdated references. Also outdated HIR can't be promoted if its history information is unavailable even in Q.

### **3.3.3 Algorithm**

STEP 1: Upon accessing LIR block X:

This access is guaranteed to be a hit in the cache. We move it to the top of stack S. If the LIR block is originally located in the bottom of the stack, we conduct a stack pruning.

STEP 2: Upon accessing HIR resident block X:

This is a hit in the cache. We move it to the top of stack *S*. There are two cases for block *X*:  
(a) If *X* is in the stack *S*, we change its status to LIR. This block is also removed from list *Q*. The LIR block in the bottom of *S* is moved to the end of list *Q* with its status changed to HIR. Stack pruning is then conducted.

(b) If *X* is not in stack *S*, we leave its status in HIR and move it to the end of list *Q*.

STEP 3: Upon accessing an HIR non-resident block *X*:

This is a miss. We remove the HIR resident block at the front of list *Q* (it then becomes a non-resident block), and replace it out of the cache. Then we load the requested block *X* into the freed buffer and place it on the top of stack *S*. There are two cases for block *X*:

(a) If *X* is in stack *S*, we change its status to LIR and move the LIR block in the bottom of stack *S* to the end of list *Q* with its status changed to HIR. A stack pruning is then conducted.

(b) If *X* is not in stack *S*, we leave its status in HIR and place it in the end of list *Q*.

### 3.3.4 Flowchart

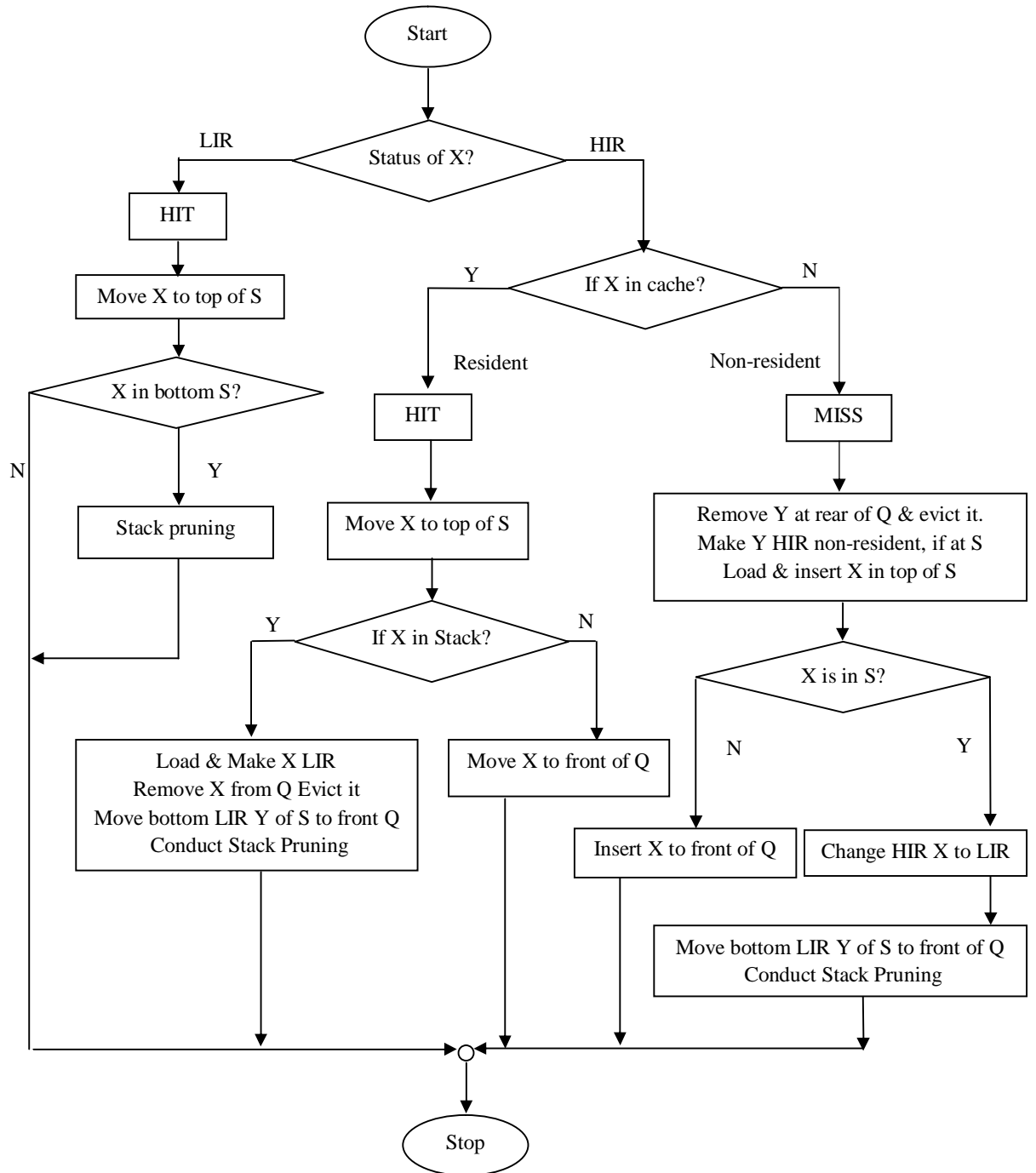


Fig. 3.5 Flowchart of LIRS Simulated Through Data Structure



### 3.3.5 Tracing

Size of LIRS: 2

Size of HIRS: 1

Cache Size: 2+1=3

Input References: A D B C B A D A E B

Number of Distinct References: 5

Total Number of References: 10

Upon accessing A:

(HIR non-resident)

LIRS= {A}

HIRS= {}

Resident HIRS= {}

Non-resident HIRS= {}

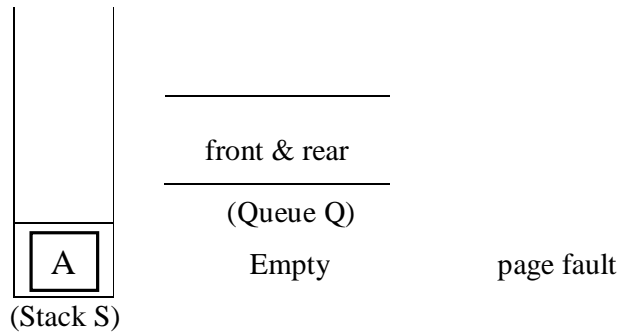


Fig 3.6.1 State at Virtual Time 1

Upon accessing D:

(HIR non-resident)

LIRS= {A, D}

HIRS= {}

Resident HIRS= {}

Non-resident HIRS= {}

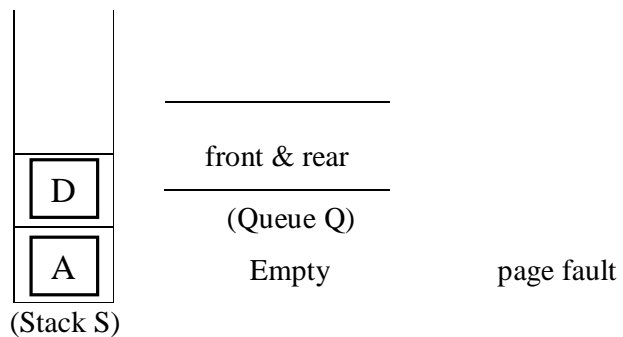


Fig 3.6.2 State at Virtual Time 2

Upon Accessing B:

(HIR non-resident)

LIRS= {A, D}

HIRS= {B}

Resident HIRS= {B}

Non-resident HIRS= {}

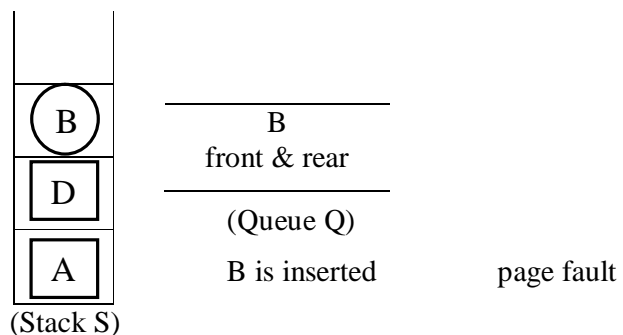


Fig 3.6.3 State at Virtual Time 3

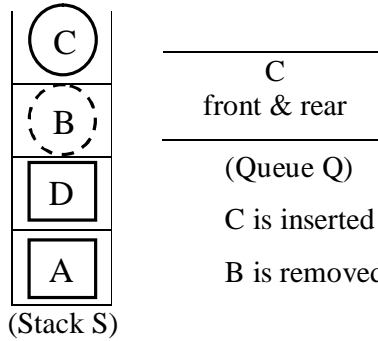
Upon Accessing C:  
(HIR non-resident)

LIRS= {A, D}

HIRS= {B, C}

Resident HIRS= {C}

Non-resident HIRS= {B}



C  
front & rear  
(Queue Q)  
C is inserted  
B is removed

page fault

Fig 3.6.4 State at Virtual Time 4

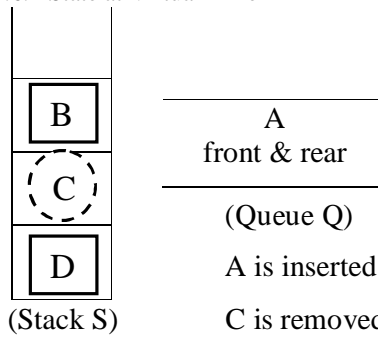
Upon Accessing B:  
(HIR non-resident)

LIRS= {B, D}

HIRS= {A, C}

Resident HIRS= {A}

Non-resident HIRS= {C}



A  
front & rear  
(Queue Q)  
A is inserted  
C is removed

page fault

B is promoted

A is demoted

Fig 3.6.5 State at Virtual Time 5

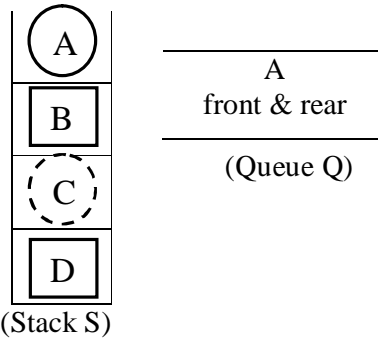
Upon Accessing A:  
(HIR resident)

LIRS= {B, D}

HIRS= {A, C}

Resident HIRS= {A}

Non-resident HIRS= {C}



A  
front & rear  
(Queue Q)

Fig 3.6.6 State at Virtual Time 6

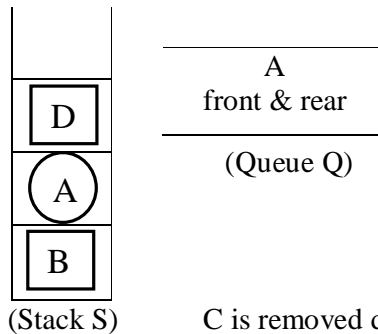
Upon Accessing D:  
(LIR)

LIRS= {B, D}

HIRS= {A}

Resident HIRS= {A}

Non-resident HIRS= { }



A  
front & rear  
(Queue Q)

C is removed during stack pruning.

Fig 3.6.7 State at Virtual Time 7

Upon Accessing A:

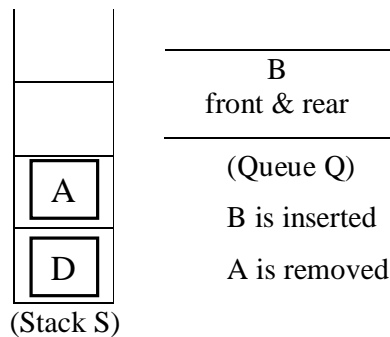
(HIR resident)

LIRS= {A, D}

HIRS= {B}

Resident HIRS= {B}

Non-resident HIRS= { }



A is promoted

B is demoted

Fig 3.6.8 State at Virtual Time 8

Upon Accessing E:

(HIR non-resident)

LIRS= {A, D}

HIRS= {E}

Resident HIRS= {E}

Non-resident HIRS= { }

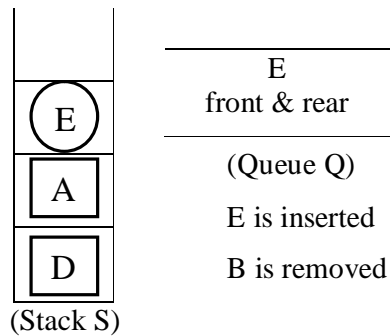


Fig 3.6.9 State at Virtual Time 9

Upon Accessing B:

(HIR non-resident)

LIRS= {A, D}

HIRS= {B, E}

Resident HIRS= {B}

Non-resident HIRS= {E}

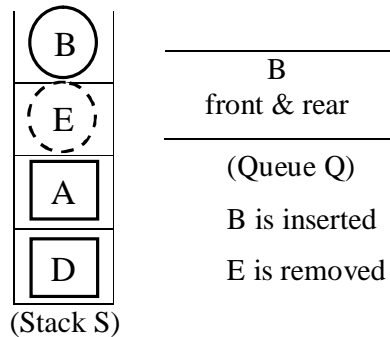


Fig 3.6.10 State at Virtual Time 10

Number of page fault: 7

### 3.4 Basic LIRS

The basic LIRS is similar to LIRS described in section 3.3, but the only difference is the major function stack pruning is not conducted. Here IRR value is also calculated by using outdated recency value of that particular block. Hence there is no need of Queue to maintain

HIR resident block. But during status change the comparison operation increases complexity. The following section clarifies the basic LIRS scheme and its illustration is also shown in section 3.4.4.

### 3.4.1 Data Structure

Stack S contains every page reference which has already been accessed. Its main purpose is to maintain recency value. As we move toward bottom recency factor increases. Bottommost one is the oldest block having higher recency factor and topmost one is the recent block having recency factor equals to zero. Each stack node contains information about reference block indicated by integer, current status (integer 2 indicates LIR, 1 indicates resident HIR and 0 indicates non-resident HIR.) and IRR value which is initially infinite but during correlated access it carries value equal to expired recency.

### 3.4.2 Algorithm

STEP 1: Upon accessing block X, if X is available in the stack S.

We move it to the top of stack S and earlier recency becomes new IRR value which is determined by counting steps to reach top of S. Page fault occurs if status of X is non-resident HIR otherwise this access is guaranteed to be a hit.

STEP 2: Upon accessing block X, if X is not available in the stack S that is non-resident HIR. This access is guaranteed to be a miss, so page fault occurs. We insert it to the top of stack S and its IRR value is assigned infinite.

STEP 3: While changing status, if  $\text{minimum IRR value of HIRS} < \text{maximum recency of LIRS}$ . Now status changes from LIR to HIR and vice versa. Block having minimum IRR value of HIRS is admitted to LIRS and block having maximum recency of LIRS is admitted to HIRS.

a) If status of block having minimum IRR value of HIRS was resident HIR.

Status of resident HIR is switched to LIR. At the same time status of LIR is switched to resident HIR.

b) If status of block having minimum IRR value of HIRS was non-resident HIR

Status of non-resident HIR is switched to LIR. Status of LIR is switched to resident HIR and status of resident HIR with high recency value is switched to non-resident HIR.

STEP 4: While changing status, if status of top of Stack S is non-resident HIR (i.e. block X from STEP 2)

Now status of X is changed to resident HIR. Status of resident HIR having highest recency (i.e. bottom most resident HIR of S) is changed to non-resident HIR.

### 3.4.3 Flowchart

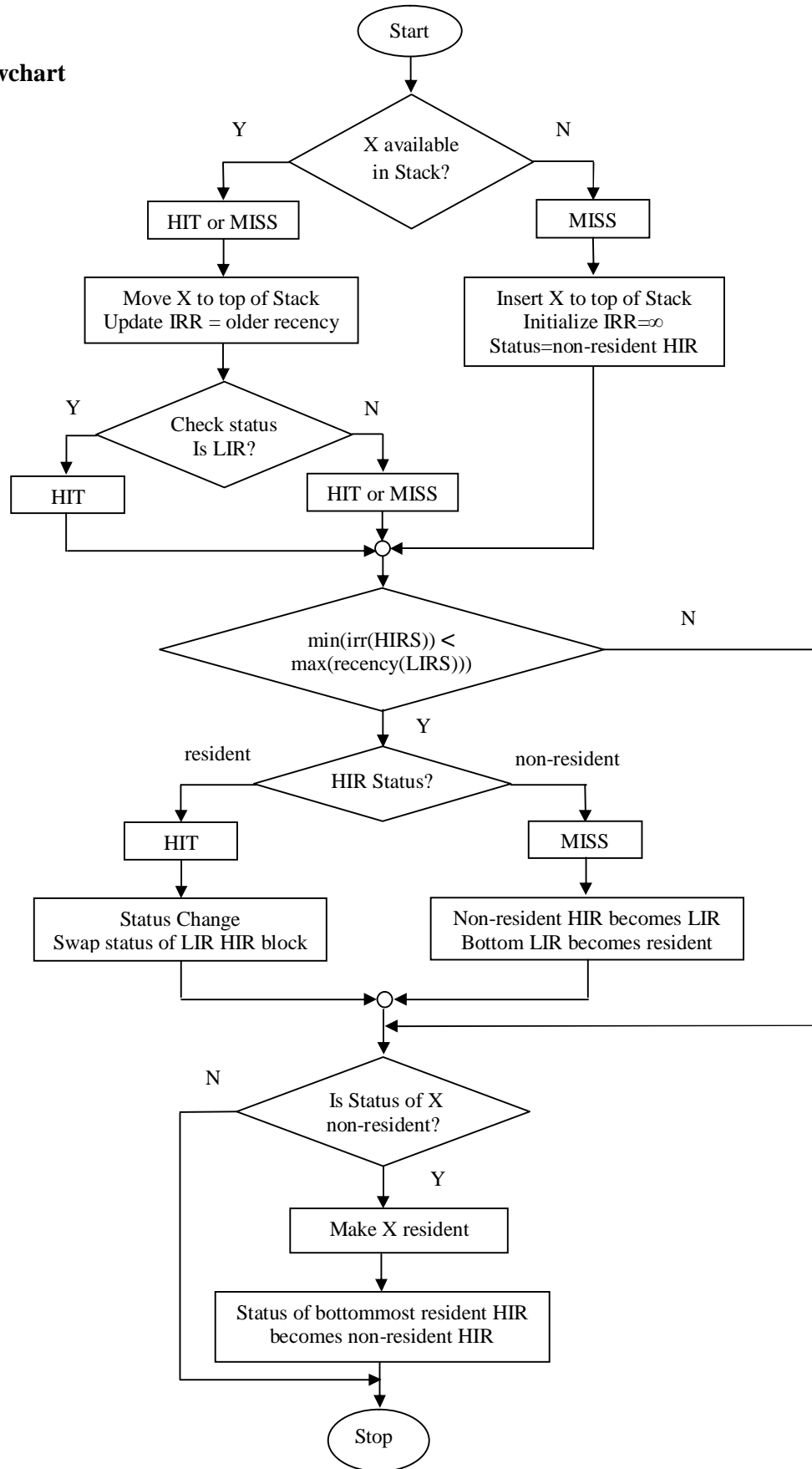


Fig 3.7 Flowchart of Basic LIRS Algorithm

### 3.4.4 Tracing

Size of LIRS: 2

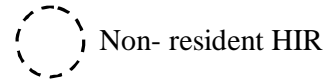
Size of HIRS: 1

Cache Size: 2+1=3

Input References: A D B C B A D A E B

Number of Distinct References: 5

Total Number of References: 10



Upon accessing A:

(HIR non-resident)

LIRS= {A}

HIRS= {}

Resident HIRS= {}

Non-resident HIRS= {}

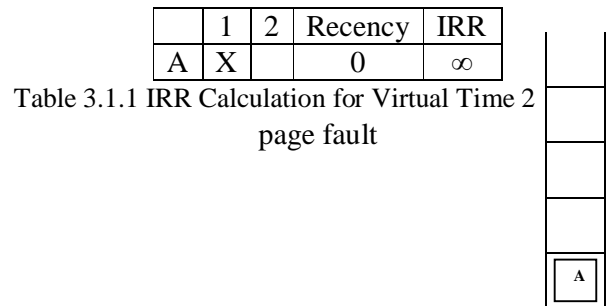


Fig 3.8.1 Stack S at Virtual Time 1

Upon accessing D:

(HIR non-resident)

LIRS= {D, A}

HIRS= {}

Resident HIRS= {}

Non-resident HIRS= {}

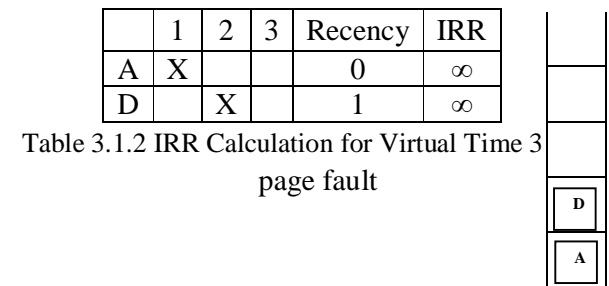


Fig 3.8.2 Stack S at Virtual Time 2

Upon accessing B:

(HIR non-resident)

LIRS= {D, A}

HIRS= {B}

Resident HIRS= {B}

Non-resident HIRS= {}

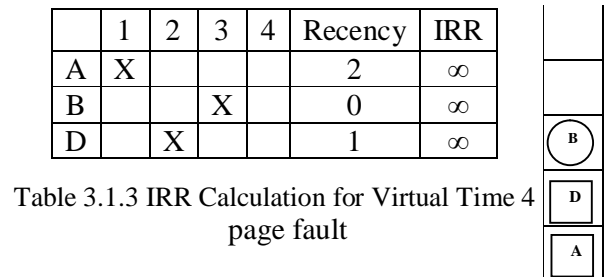


Fig 3.8.3 Stack S at Virtual Time 3

Upon accessing C:

(HIR non-resident)

LIRS= {D, A}

HIRS= {C, B}

Resident HIRS= {C}

Non-resident HIRS= {B}

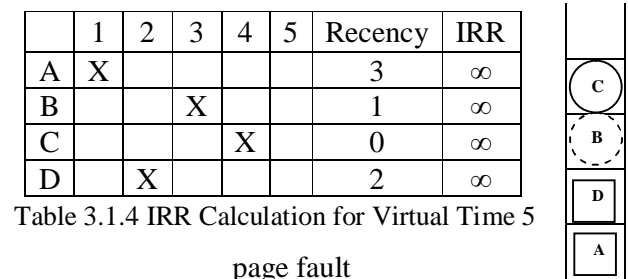


Fig 3.8.4 Stack S at Virtual Time 4

Upon accessing B:

(HIR non-resident)

LIRS= {B, D}

HIRS= {A, C}

Resident HIRS= {A}

Non-resident HIRS= {C}

Upon accessing A:

(HIR resident)

LIRS= {B, D}

HIRS= {A, C}

Resident HIRS= {A}

Non-resident HIRS= {C}

Upon accessing D:

(LIR)

LIRS= {D, B}

HIRS= {A, C}

Resident HIRS= {A}

Non-resident HIRS= {C}

Upon accessing A:

(HIR resident) Status Change

LIRS= {A, D}

HIRS= {B, C}

Resident HIRS= {B}

Non-resident HIRS= {C}

Upon accessing E:

(HIR non-resident)

Status Change

LIRS= {A, B}

HIRS= {E, D, C}

Resident HIRS= {E}

Non-resident HIRS= {D, C}

Status Change

	1	2	3	4	5	6	Recency	IRR
A	X						3	$\infty$
B			X		X		0	1
C				X			1	$\infty$
D		X					2	$\infty$

$$\max(\text{recency}(\text{LIRS})) > \min(\text{irr}(\text{HIRS}))$$

Table 3.1.5 IRR Calculation for Virtual Time 6 page fault

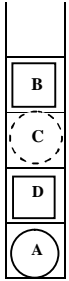


Fig 3.8.5 Stack S at Virtual Time 5

	1	2	3	4	5	6	7	Recency	IRR
A	X					X		0	3
B			X		X			1	1
C				X				2	$\infty$
D		X						3	$\infty$

Table 3.1.6 IRR Calculation for Virtual Time 7

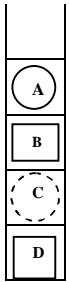


Fig 3.8.6 Stack S at Virtual Time 6

	1	2	3	4	5	6	7	8	Recency	IRR
A	X					X			1	3
B			X		X				2	1
C				X					3	$\infty$
D		X					X		0	3

Table 3.1.7 IRR Calculation for Virtual Time 8

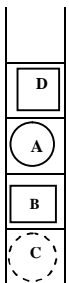


Fig 3.8.7 Stack S at Virtual Time 7

	1	2	3	4	5	6	7	8	9	Recency	IRR
A	X					X		X		0	1
B			X		X					2	1
C				X						3	$\infty$
D		X					X			1	3

Table 3.1.8 IRR Calculation for Virtual Time 9

$$\max(\text{recency}(\text{LIRS})) > \min(\text{irr}(\text{HIRS}))$$

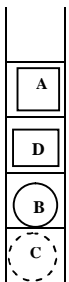


Fig 3.8.8 Stack S at Virtual Time 8

	1	2	3	4	5	6	7	8	9	10	Recency	IRR
A	X					X		X			1	1
B			X		X						3	1
C				X							4	$\infty$
D		X					X				2	3
E								X			0	$\infty$

Table 3.1.9 IRR Calculation for Virtual Time 10

$$\max(\text{recency}(\text{LIRS})) > \min(\text{irr}(\text{HIRS})) \quad \text{page fault}$$

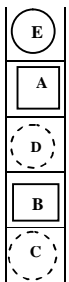


Fig 3.8.9 Stack S at Virtual Time 9

Upon accessing B:

(LIR)

LIRS= {B, A}

HIRS= {E, D, C}

Resident HIRS= {E}

	1	2	3	4	5	6	7	8	9	10	11	Recency	IRR
A	X					X		X				<b>2</b>	<b>1</b>
B			X		X					X		0	3
C				X								4	$\infty$
D		X					X					3	<b>3</b>
E									X			1	$\infty$

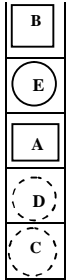


Table 3.1.10 IRR Calculation for Virtual Time 11

Non-resident HIRS= {D, C}

Fig 3.8.10 Stack S at Virtual Time 10

Number of page fault: 6



## Chapter 4

### DATA COLLECTION & ANALYSIS

#### 4.1 Data Collection

Data are the sources of information. Hence data should be collected very carefully. All the data are collected by means of primary sources. In this dissertation work data are generated by using number of C source codes. These source files are used for generating memory references that are so called as workloads. The workload represents weak locality of memory reference pattern that are generated during execution of process in real OS. Here Weak locality workloads can be categorized into reference of sequential scan as Workload 1, reference of loop which is larger than cache size as Workload 2 and reference of probabilistic pattern as Workload 3. Each category contains ten thousand memory references and more. Sample of Workload 1, Workload 2 and Workload 3 is in appendix A, appendix B and appendix C respectively.

#### 4.2 Testing

These three workloads are separately tested in our simulator. Each workload is tested in LRU, LIRS simulated through data structure and basic LIRS simulator by varying the cache size from 4 to 1024. In case of LIRS algorithms HIR, LIR partition is maintained as 1% and 99% of cache size. But in case of Workload 2 size of LIR is maintained cache size-1 or HIR is maintained 1, because page fault decreases with minimum LIR size.

##### 4.2.1 Test Result of Workload 1

No. of References = 10000

No. of Distinct Reference = 8840

Cache Size	LRU			Basic LIRS			LIRS Simulated Through Data Structure		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	10000	100%	0%	9969	97.33%	2.67%	9985	98.71%	1.29%
8	10000	100%	0%	9929	93.88%	6.12%	9965	96.98%	3.02%
16	10000	100%	0%	9849	86.98%	13.02%	9910	92.24%	7.76%
32	10000	100%	0%	9689	73.19%	26.81%	9814	83.97%	16.03%
64	10000	100%	0%	9600	65.52%	34.48%	9760	79.31%	20.69%
128	10000	100%	0%	9600	65.52%	34.48%	9760	79.31%	20.69%
256	10000	100%	0%	9600	65.52%	34.48%	9760	79.31%	20.69%
512	9440	51.73%	48.27%	9600	65.52%	34.48%	9720	75.86%	24.14%
1024	9440	51.73%	48.27%	9600	65.52%	34.48%	9680	72.41%	27.59%

Table 4.1 Test Result of Workload 1

### 4.2.2 Test Result of Workload 2

No. of References = 10000

No. of Distinct Reference = 200

Cache Size	LRU			Basic LIRS			LIRS Simulated Through Data Structure		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	10000	100%	0%	9853	98.50%	1.50%	9973	99.72%	0.28%
8	10000	100%	0%	9657	96.50%	3.50%	9930	99.29%	0.71%
16	10000	100%	0%	9265	92.50%	7.50%	9850	98.47%	1.53%
32	10000	100%	0%	8481	84.50%	15.50%	9659	96.52%	3.48%
64	10000	100%	0%	6913	68.50%	31.50%	9181	91.64%	8.36%
128	10000	100%	0%	3777	36.50%	63.50%	6825	67.60%	32.40%
256	200	0%	100%	200	0%	100%	200	0%	100%
512	200	0%	100%	200	0%	100%	200	0%	100%
1024	200	0%	100%	200	0%	100%	200	0%	100%

Table 4.2 Test Result of Workload 2

### 4.2.3 Test Result of Workload 3

No. of References = 10500

No. of Distinct Reference = 2417

Cache Size	LRU			Basic LIRS			LIRS Simulated Through Data Structure		
	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate	Page Fault	Miss Rate	Hit Rate
4	10456	99.45%	0.55%	10469	99.62%	0.38%	10492	99.90%	0.10%
8	10429	99.12%	0.98%	10422	99.03%	0.97%	10491	99.89%	0.11%
16	10392	98.66%	1.34%	10215	96.47%	3.53%	10471	99.64%	0.36%
32	10317	97.74%	2.26%	9848	91.93%	8.07%	10412	98.91%	1.09%
64	10295	97.46%	2.54%	9245	84.47%	15.53%	10221	96.55%	3.45%
128	10253	96.94%	3.06%	8534	75.67%	24.33%	10023	94.01%	5.99%
256	10149	95.66%	4.34%	7858	67.31%	32.69%	9820	91.59%	8.41%
512	8370	73.65%	26.35%	6933	55.87%	44.13%	9420	86.64%	13.36%
1024	6587	51.59%	48.41%	5303	35.70%	64.30%	8314	72.95%	27.05%

Table 4.3 Test Result of Workload 3

### 4.3 Analysis

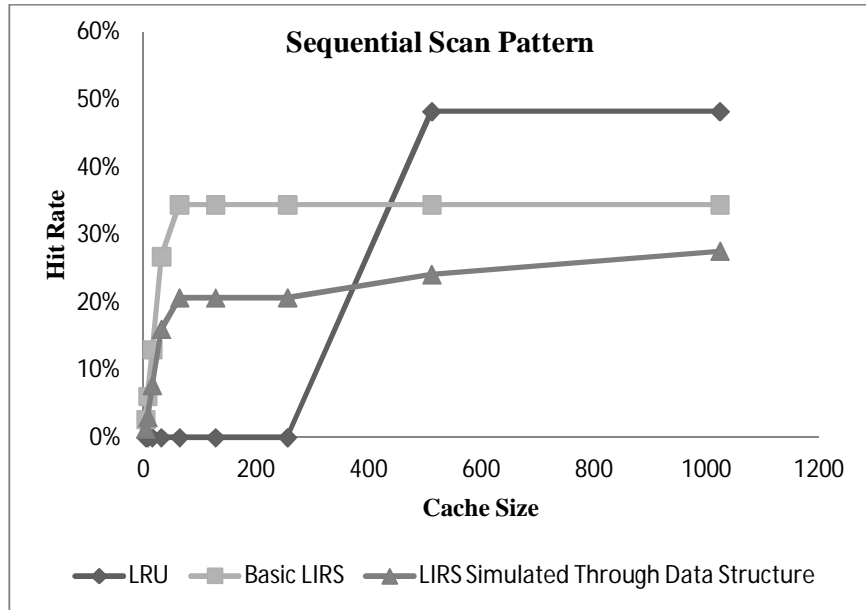


Fig 4.1 Graph of Workload 1

Workload 1 is the sequential scan type of trace as the cache size increases the page fault also decreases. After increasing the cache size to 300 and above, LRU performances drastically changes than other LIRS algorithms. This shows that memory reference pattern is moving toward strong locality of reference which is favorable for LRU.

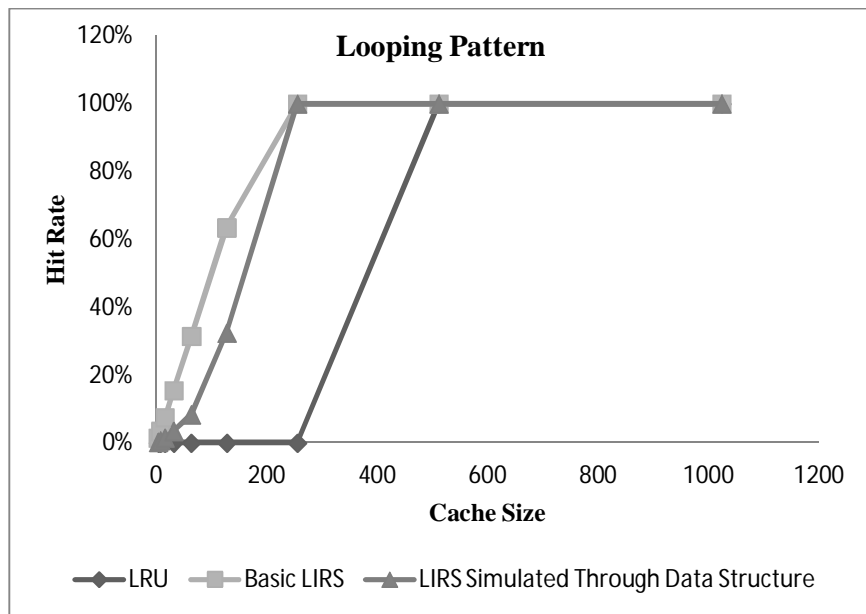


Fig 4.2 Graph of Workload 2

Workload 2 is the trace of loop larger than cache size as the cache size increases the page fault also decreases. After increasing the cache size to 200 and above, all the algorithms shows 100% hit rate because the trace consists of loop from memory reference 1 to 200. Since all 200 distinct blocks are resident, the trace then doesn't favor our assumption of weak locality workload. Also fixing HIR partition to 1 is the best way to get minimum page fault.

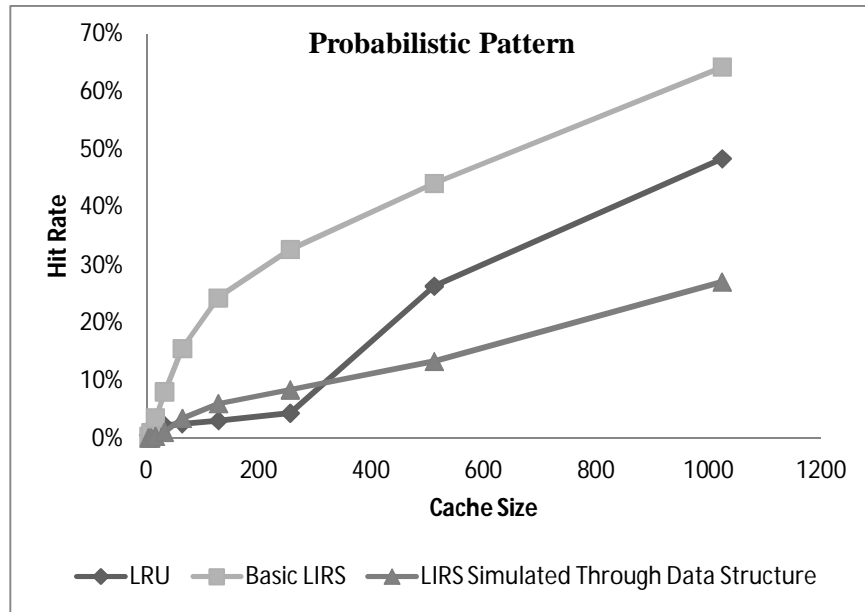


Fig 4.3 Graph of Workload 3

Workload 3 is the trace of probabilistic pattern where each memory reference has their own frequency they are accessed. When the cache size is 350 and above LRU shows change because of change in locality. Because the trace consists of repetition of frequency blocks after every 350 references and besides frequency blocks all other blocks are randomly occurring. Hence unlike other cases 100% hit rate can't be achieved. Here basic LIRS is working better, since it can store deeper history information than LIRS simulated through data structure.

The graphs of figure 4 show that the basic LIRS algorithm is better than LIRS simulated through data structure. Since all the workloads that we have used in this work represent weak locality memory references, LRU misbehaves but as the locality changes due to change in cache size performance gain can be achieved.

## Chapter 5

### CONCLUSION AND RECOMMENDATION

#### 5.1 Conclusion

Replacement algorithms are valuable components of operating system design and can affect system performance significantly. Common LRU failures can be solved by using user level hints, tracing and utilizing history information and detection and adaptation of access regularities. LIRS can solve problems regarding weak locality of reference by tracing and utilizing history information. The failure of LRU is due to the bold assumption on recency. Negative effects caused by taking only recency value are removed by considering IRR as history information. The algorithm successfully handles weak locality of reference.

LIRS is a valuable replacement algorithm. It is simple as LRU. The basic approach is the idea behind its success. The policy decides more accurately than LIRS simulated through data structure for pages not in memory to make replacement. This is only due to storage of deeper history information, which is lost during stack pruning in LIRS. Unlike other traditional page replacement algorithm, it can change very faster according to program behavior. Basic LIRS is easier to implement because of simple data structure. But LIRS simulated through data structure contains additional data structure to hold resident HIR but that doesn't increase space requirement as it is 1% of cache size. Basic LIRS algorithm requires data update in every memory access which makes it impracticable in real OS. Both of these implementations will be important for future research work.

#### 5.2 Recommendation

The LIRS page replacement algorithm consists of two parameters, i.e. size of LIR and size of HIR which can be self tuned according as workloads. As we know minimum size of HIR that is equal to 1 is the best parameter in case of loop with larger cache size. Hence dynamic approach can be used to self tune this parameter. The data structure used can be improved so as to decrease the computation complexity. Here three different sample traces are used which is actually not the real trace recorded during the execution of program. This work can be standardized by using real traces.

In case of basic LIRS we can limit the size of LRU stack so as it is applicable in real time implementation. Also the size of stack can be self-tuned according as the workload. Considering weak locality of reference and program complexity during simulation is taken as the limitations of this work.

## References

- [1] A.S. Tanenbaum, Modern Operating Systems (Prentice Hall, 2nd Edition).
- [2] H.M. Deitel, Operating Systems, Chap.9 Virtual Storage Management (Pearson Education, 2nd Edition).
- [3] A. Silberschatz, P. B. Galvin, & G. Gagne, Operating System Concepts (Wiley, 7th Edition).
- [4] G. Nutt, Operating Systems A Modern Perspective (Addison Wesley Longman, 2nd Edition).
- [5] [www.docstoc.com/docs/21106969/Role-of-OS-in-virtual-memory-management](http://www.docstoc.com/docs/21106969/Role-of-OS-in-virtual-memory-management)
- [6] H. Paaanen, Page Replacement in Operating System Memory Management, Master's Thesis in Information Technology, University of Jyväskylä, Department of Mathematical Information Technology, October 23, 2007.
- [7] W. Stallings, Computer Organization and Architecture: Designing for Performance, (Prentice-Hall, 6th Edition).
- [8] J. Kubiawicz, Operating System and System Programming, Lec.15 Page Allocation and replacement , 2009, <http://inst.eec.berkeley.edu/~CS162>
- [9] K. McMaster, S. Sambasivam, N. Anderson, How Anomalous Is Belady's Anomaly?, Issues in Informing Science and Information Technology, Vol. 6, 2009, pp 827-836.
- [10] S. Jiang and X. Zhang, Making LRU Friendly to Weak Locality Workloads: A Novel Replacement Algorithm to Improve Buffer Cache Performance, IEEE Transactions on Computers, Vol. 54, No. 8, 2005, pp 939-952.
- [11] <http://www.cse.ohio-state.edu/~zhang/influential-papers.html>
- [12] G. Glass, P. Cao, Adaptive Page Replacement Based on Memory Reference Behavior, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems, 1997, pp 115-126.
- [13] Y. Smaragdakis, S. Kaplan, P. Wilson, EELRU: Simple and Effective Adaptive Page Replacement, In Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems, 1999, pp 122-133.
- [14] G. P. Joshi, Calculation Of Control Parameter  $\lambda$  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 in Computer Science and Information Technology, Tribhuvan University, Central Department of Computer Science and Information Technology.

- [15] J. Elizabeth, O'Neil, Patrick E. O'Neil, and G. Weikum, The LRU-K Page Replacement Algorithm for Database Disk Buffering, ACM SIGMOD, 1993, pp 297-306.
- [16] T. Johnson, D. Shasha, 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm, Proceedings of the 20th International Conference on VLDB, 1994, pp 439-450.
- [17] F. J. Corbató, A paging experiment with the Multics system. In Honor of P. M. Morse, MIT Press, 1969, pp 217–228.
- [18] S. Jiang, F. Chen, X. Zhang, CLOCK-Pro: An effective improvement of the CLOCK replacement. In Proceedings of the 10th Annual USENIX Technical, 2005, pp 323-336.
- [19] S. Bansal, D.S. Modha, CAR: Clock with Adaptive Replacement, In Proceedings of the USENIX Conference on File and Storage Technologies, 2004, pp 187-200.
- [20] N. Megiddo, D. S. Modha, ARC: A Self-Tuning, Low Overhead Replacement Cache, In Proceedings of the USENIX Conference on File and Storage Technologies, 2003, pp 115-130.
- [21] X. Ding, S. Jiang, X. Zhang, BP-Wrapper: A System Framework Making Any Replacement Algorithms (Almost) Lock Contention Free, IEEE International Conference on Data Engineering, 2009, pp 369-380.
- [22] J. Choi et al., An Implementation Study of a Detection-Based Adaptive Block Replacement Scheme, USENIX Annual Technical Conference, 1999, pp 239-252.
- [23] J. Choi et al., Towards Application/File-Level Characterization of Block References: A Case for Fine-Grained Buffer Management, In Proceeding to the 25th International Conference on Measurement and Modeling of Computer Systems, 2000, pp 286-295.
- [24] M. Sabeghil, M. H. Yaghmaee, Using Fuzzy Logic to Improve Cache Replacement Decisions, IJCSNS International Journal of Computer Science and Network Security, Vol. 6, No. 3A, 2006, pp 182-188.
- [25] J. M. Kim 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 USENIX, 2000, pp 119-134.
- [26] S. Bagchi, M. Nygaard, A Fuzzy Adaptive Algorithm for Fine Grained Cache Paging. 8th International Workshop, 2004, pp 200-213.
- [27] M. L. Singh, Understanding Research Methodology, Chap.1 Scientific Method and Research, pp 4.

## Appendix A: Sample Trace of Sequential Scan, Workload 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 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 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 29 30 31 32 33 34 35 36 37 38 39 40  
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  
532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552  
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 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 29 30 31 32 33 34 35 36 37 38 39 40 601 602 603 604 605 606 607 608 609 610  
611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631  
632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652



## Appendix B: Sample Trace of Looping Pattern, Workload 2

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 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 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 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 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 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 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 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 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 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

### **Appendix C: Sample Trace of Probabilistic Pattern, Workload 3**

366 377 8 458 31 448 93 17 330 438 291 354 81 344 246 328 124 307 149 57 366 259 68  
247 455 85 451 26 43 339 73 431 235 284 414 209 486 403 225 388 376 206 369 293 435  
441 73 39 163 59 5 20 69 82 56 88 95 35 50 74 858 532 725 542 878 843 996 586 602 784  
734 719 970 924 980 658 973 941 543 616 661 534 626 873 895 657 756 744 801 989 684  
663 689 548 500 912 606 879 734 540 729 519 799 925 997 592 965 532 760 675 105 120  
169 192 156 178 195 145 150 174 1491 1487 1109 1223 1254 1231 1170 1287 1407 1129  
1151 1496 1487 1386 1067 1274 1328 1309 1119 1043 1387 1308 1203 1414 1300 1311  
1444 1145 1049 1355 1311 1001 1368 1270 1031 1429 1483 1082 1054 1484 1116 1477  
1307 1103 1493 1465 1200 1170 1465 1071 203 220 269 282 256 287 295 235 251 274 1751  
1900 1655 1786 1614 1667 1885 1770 1879 1642 1681 1737 1844 1699 1863 1756 1999  
1507 1797 1630 1883 1599 1815 1941 1842 1636 1903 1539 1703 1540 1746 1964 1783  
1907 1630 1893 1567 1895 1672 1519 1579 1855 1978 1797 1820 1946 1576 1517 1716  
1962 325 320 369 382 336 388 395 335 350 384 2192 2104 2093 2307 2003 2086 2100 2234  
2447 2118 2264 2126 2198 2364 2161 2288 2171 2135 2269 2253 2487 2457 2024 2168  
2133 2192 2361 2137 2337 2485 2287 2090 2444 2256 2421 2374 2391 2163 2375 2394  
2165 2367 2253 2494 2300 2312 2311 2244 2343 2419 405 420 469 482 456 418 495 435  
450 474 48 240 474 350 45 29 26 77 203 52 135 279 194 206 432 108 427 312 258 146 8 81  
429 464 0 335 369 425 434 157 109 211 469 51 357 369 316 319 162 177 176 467 142 180  
233 229 452 66 110 106 5 20 69 82 56 88 95 35 50 74 726 950 540 602 763 673 933 676 951  
527 577 773 666 892 639 957 956 910 876 743 640 987 721 605 696 574 762 723 864 939  
780 949 933 678 738 610 984 669 953 941 662 919 681 597 980 820 675 628 510 812 105  
120 169 192 156 178 195 145 150 174 1444 1156 1359 1127 1371 1406 1159 1427 1029  
1112 1038 1275 1022 1252 1425 1244 1194 1069 1232 1327 1229 1438 1352 1241 1478  
1459 1259 1257 1001 1349 1466 1349 1465 1128 1322 1196 1241 1047 1356 1175 1406  
1376 1243 1049 1202 1305 1181 1220 1018 1170 203 220 269 282 256 287 295 235 251 274  
1766 1762 1974 1856 1969 1784 1530 1992 1568 1524 1717 1588 1863 1545 1669 1822  
1722 1623 1935 1963 1938 1658 1644 1898 1695 1834 1946 1845 1671 1667 1599 1597  
1638 1929 1879 1879 1797 1766 1687 1785 1706 1871 1685 1895 1515 1601 1967 1623  
1907 1612 325 320 369 382 336 388 395 335 350 384 2322 2063 2474 2433 2056 2279 2427  
2163 2163 2298 2355 2039 2327 2475 2379 2130 2272 2166 2431 2381 2054 2005 2311  
2017 2486 2332 2383 2031 2151 2063 2089 2056 2371 2185 2133 2130 2087 2048 2182