The University of Alberta

Experiments in Database Buffer Management Strategies in a Virtual Memory Environment

by

© Meei Fen Teo

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring, 1989

ISBN 0-315-52990-3

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Meei Fen Teo

TITLE OF THESIS: Experiments in Database Buffer Management Strategies in a Virtual Memory Environment

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: Spring 1989

(Signed) ....

Permanent address:

10, Chestnut Close

S'pore 2367

Republic of Singapore

Date: 13th April 1989

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Experiments in Database Buffer Management Strategies in a Virtual Memory Environment** submitted by **Meei Fen Teo** in partial fulfillment of the requirements for the degree of **Master of Science**.

....

(Supervisor)

....

....

....

Date: 12th April 1989

Dedicated to my parents

Abstract

In most database management systems, a cache buffer is used to reduce the number of accesses to the disks. Due to the special properties of the DBMS queries, some of the common page replacement policies are not very suitable for the DBMS buffer manager. Special DBMS buffer replacement algorithms have been proposed and their performance in a main memory system seem promising. Most DBMSs are run on top of a virtual memory operating system, but no study exists that investigates these algorithms in this environment.

In this thesis, the performance of two special DBMS buffer replacement algorithms in a virtual memory system is investigated. The replacement algorithms are the hot set model and the QLS model. The performance study is simulation-based where a simulator is used to emulate the execution of the DBMS queries in a virtual memory system. The performance results of these two special DBMS buffer managers indicate that they are quite similar when the DBMS is run on top of a virtual memory system.

Acknowledgements

I wish to thank my supervisor, Dr. Tamer Özsu, for providing a friendly environment for research and for his patience and guidance during the course of this thesis.

I would also like to thank the members of my examining committee, Dr. L. Y. Yuan, Prof. U. Maydell and Dr. J. Mowchenko, for their helpful comments. Thanks are also dued to the members of the distributed database group: Tse-Men Koon, Ken Barker, Dave Straube and You-Ping Niu, for their unselfish opinions and comments.

I am very grateful to the members of my family for their devoted support, both spiritually and financially. I also wish to acknowledge my friends in Edmonton for making my stay here a pleasant one. I thank Brian Wong and Keith Fenske for proof reading this thesis. Finally, I would like to thank Joon Kiat Lee for inspiring the final push.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

In most database management systems (DBMSs), a cache buffer is used to reduce the number of accesses to the disks or secondary storage. Studies have been conducted to compare the suitability of various operating system (OS) file manager page replacement algorithms for use by the DBMS buffer manager. As pointed out in [Gra78,Sto81,TM82,Ozs88], these well-known algorithms such as Least Recently Used (LRU) are not very suitable for DBMS buffer management.

Figure 1.1 gives a high level logical view of the DBMS buffers with respect to the virtual memory and main memory. Each box represents the size of the various types of memory. As can be seen from the figure, the DBMS buffers are mapped into a virtual memory partition of equal size. This portion of the virtual memory may (completely or partially) not be in the main memory. Bringing them into the main memory to make them accessible is what causes the various types of *faults*. In general, the relationship between the DBMS buffers and the virtual memory manager of the operating system can be represented as in Figure 1.2. The database is organised on secondary storage as a collection of equi-sized units called *database pages*. The virtual memory and the main memory is also organised similarly. The units are called

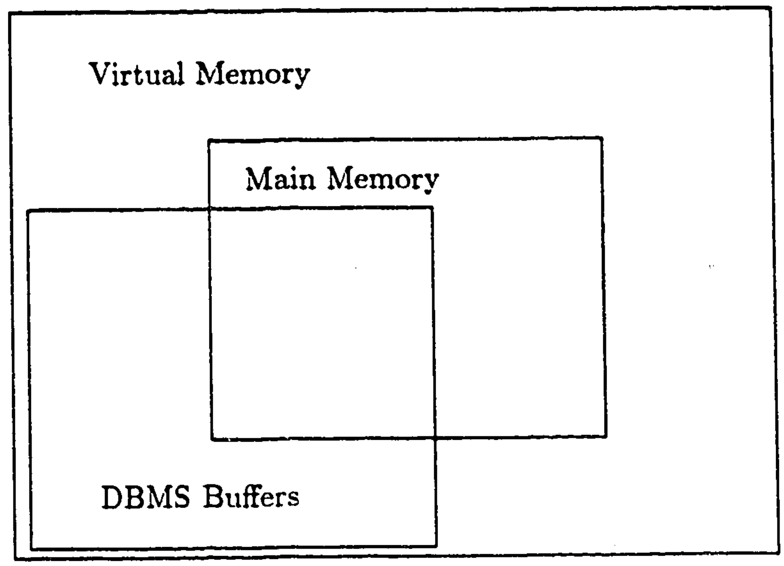

Figure 1.1: Mapping of DBMS Buffers in a Virtual Memory System

virtual memory pages for virtual memory and *frames* for main memory. The size of a frame, the size of a database page and the size of a virtual memory page are equal. Two replacement algorithms are involved: one for the replacement of DBMS buffers and the other for the replacement of frames. Note that these two algorithms are totally independent of each other even though they might employ the same policies.

Figure 1.3 shows two ways in which a DBMS can be implemented on a general purpose virtual memory operating system environment. In the first method (Figure 1.3a), the DBMS is implemented on top of the virtual memory. In the second method (Figure 1.3b), the main memory is partitioned into two parts: main memory which maps the DBMS buffers directly and main memory which are used by other tasks.

In the first method, where the database buffers are mapped into virtual memory, the DBMS buffer manager does not have complete control over its buffers. This is because the entire main memory and the mapping of virtual memory pages to frames is under the control of the OS virtual memory manager. Consider a database page p_i that is referenced by the DBMS which is in a database buffer that is already mapped to a main memory frame, f_i.

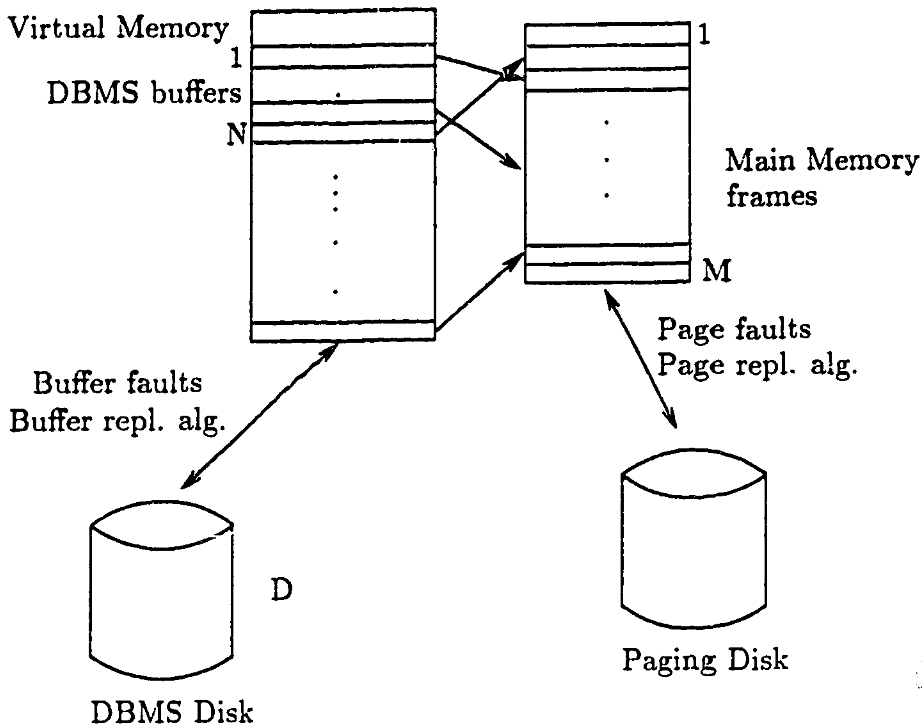

Figure 1.2: Relationship between DBMS Buffers and a Virtual Memory System

During the DBMS access to this page, if the virtual memory manager decides to allocate f_i to another process, the DBMS buffer manager is not in a position to prevent this allocation. This results in extra disk accesses to bring the page p_i back into the main memory when the DBMS accesses it again.

This problem is not encountered in the second method where the DBMS buffer manager and the virtual memory manager share the main memory. This is because the DBMS buffers are mapped into main memory and it is the DBMS buffer manager that decides when to keep a page in a main memory buffer and when to write a page back to the secondary storage.

There are two ways by which the DBMS and the virtual memory manager can share the main memory: static or dynamic partitioning. In the static partitioning method, the main memory is divided into a virtual memory partition and a DBMS partition. Frames do not migrate between the two partitions. That is, once the size of the partitions has been determined, it remains fixed regardless of the possible load differences between the two systems. This prob-

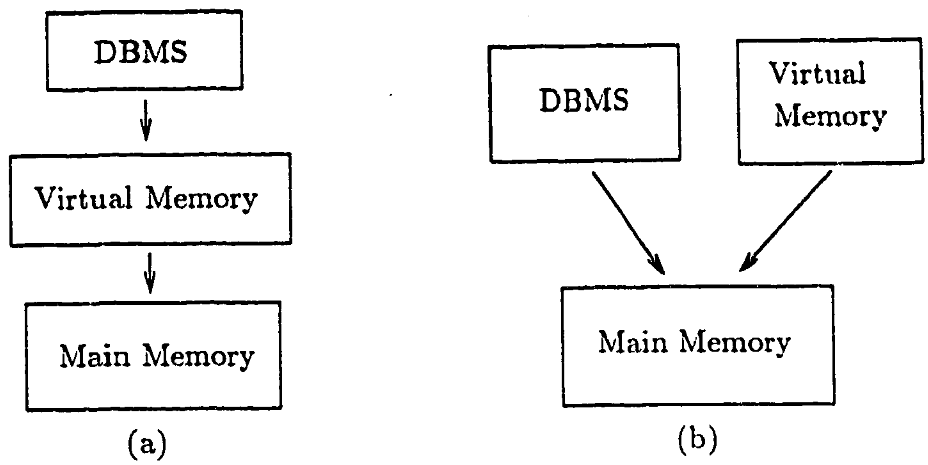

Figure 1.3: DBMS in a General Purpose Virtual Memory OS

lem is overcome in the dynamic partitioning method. In this method, the size of the two partitions changes dynamically, depending on the load of both sides.

A *buffer fault* occurs when the needed data is not in the DBMS buffers requiring an access to the database disk. A *page fault* occurs when an access to the paging disk device is required. For example, a page fault is induced when the virtual memory page which maps a DBMS buffer is not in the main memory. In this case, if the requested data page is in a DBMS buffer but the virtual memory page which maps the DBMS buffer is not in the main memory, the fault is termed a *reference fault*. If the requested data page is not in the DBMS buffer and the virtual memory page which maps the DBMS buffer which has been chosen for replacement is not in the main memory, a *double fault* is said to have taken place. Note that in a reference fault, only one disk access is required whereas in a double fault, two disk accesses are generated. Also, a double fault can occur only if a buffer fault has already taken place.

When a fault is generated, a frame or buffer has to be chosen for replacement. Some common replacement algorithms are:

- LRU, where the page that has not been referenced for the longest time is chosen.

- Most recently Used (MRU), where the most recently used page is replaced. This is used in cyclic page references.

- First In First Out (FIFO), where the page that has been in the main memory the longest (regardless of when it was referenced) is replaced.
- Random (R), where a page is randomly selected for replacement. This policy is used when the paging behavior of a process obeys the random reference model.
- Last In First Out (LIFO), is the converse of FIFO where the page that has been in main memory for the shortest time is replaced. This policy is used when the available space is not enough to hold all the pages that are repeated scanned.
- Clock, where the frames are scanned until a unused frame is found. A reference bit is associated with each frame. As the frames are scanned in a cyclic manner, each bit associated with the frame being scanned is reset if the bit has been set. If it has not been set then, the frame is chosen for replacement. Sometimes, this policy is also called Second Chance.

Usually, the cost of a disk access to a database disk is much higher than the cost of an access to a paging disk device. The *I/O cost* is therefore defined as the sum of the number of buffer faults and the number of page faults, where the latter is divided by the ratio of the cost of an access to the database disk to the cost of an access to the paging device (r).

$$I/O\ cost \;=\; number\ of\ buffer\ faults \;+\; \frac{number\ of\ page\ faults}{r}$$

The sum of the number of buffer faults and reference faults is termed the *buffer I/O*. The number of page faults caused by the DBMS program is called the *system I/O*. The sum of the above two terms is defined as the *total I/O*. A summary of the above definitions can be found in Appendix A.

1.2 Objective of Thesis

In the past several years, special algorithms have been proposed for DBMS buffer management. These algorithms take advantage of the fact that in a DBMS, a query's reference behavior can be known in advance. In this thesis, the performance of two such algorithms in a virtual memory environment are studied. The two algorithms are the Hot Set model [SS86] and the query locality set (QLS) model [CD86]. The Hot Set model predicts the memory requirement of a query that will be executed in a LRU managed buffer pool. The QLS model adapts the buffer replacement policy and buffer requirement associated with each query to the access pattern that the query exhibits when accessing a file.

The results of earlier performance studies of these two special buffer replacement algorithms where DBMS buffers are mapped to main memory are promising. Most DBMSs are run on top of a virtual memory operating system, but no study exists that investigates the algorithms in this environment. In this thesis, we study the performance of the Hot Set and the QLS buffer replacement algorithms in a virtual memory operating system environment. We have implemented the alternative as shown in Figure 1.3, with static partitioning of the main memory. The policy used by the page replacement algorithm of the virtual memory manager is global LRU. In the global LRU method, all the frames in the system are shared among all the processes. When a page fault occurs, the least recently used frame is chosen for replacement.

This performance study is simulation-based where we use a simulator to emulate the execution of queries of the DBMS in a computer system with virtual memory. The simulator consists of a computer system model and a page reference string generator. For each query, its page reference string is generated before it is executed in the computer system model. A virtual time unit is used to control event timing in the simulator. Each reference to a page in main memory constitutes a virtual time unit.

The measurements considered are the mean number of page faults, the mean number of buffer faults, the mean number of double faults, the mean number of I/Os, the mean response time and the throughput which is defined as the number of queries processed per second in the total interval of the simulation time. The mean number of the various types of faults will give us an indication of the percentage each type contributes to the response time and I/Os. Faults have also been used in many studies to determine performances. The throughput is a good measure for the overall performance of the system. The response time and faults will give the performance with respect to each query but the throughput gives the performance of the whole system.

1.3 Organization of Thesis

In Chapter 2 and Chapter 3, a review of the studies which have been performed on the DBMS buffer replacement algorithms is given. Chapter 2 reviews the studies on the effects of implementing various types of conventional page replacement policies such as FIFO (First In First Out), Random, Clock and LRU as the DBMS buffer replacement algorithm when the DBMS is running on top of a virtual memory operating system. In Chapter 3, the Hot Set model and the QLS model are discussed. Chapter 4 gives a description of the simulator used in our study. Chapter 5 discusses the results obtained from the performance studies of the two buffer replacement algorithms. Finally, Chapter 6 gives the conclusions and some suggestions for further research.

Chapter 2

Previous Studies

2.1 Introduction

This chapter reviews the work which involves studies on the effects of running a DBMS on top of a virtual memory operating system. These studies are mainly concerned with minimizing the number of I/O generated by the queries being processed by the DBMS. The research work which has been done in this area mainly concentrates on finding the best combination of conventional replacement policies between the DBMS buffer replacement algorithm and the main memory replacement algorithm. The effects of factors such as the DBMS buffer size, the main memory size, the buffer replacement policy and memory replacement policy are considered in this research.

2.1.1 Effects of a Prefix Table

The earliest work is done by Tuel [Tue76]. He proposes a model to study the total I/O activity generated by the IMS version 2.4 [Mac73] in a virtual memory system. In this version of IMS, a search in the DBMS buffers for a requested data page generates faults if the buffer page being searched is not in the main memory. In other words, no effort is taken to reduce the number of faults caused by searching. Therefore, whenever a page is requested, all the

DBMS buffers have to be brought into the main memory, thus increasing the number of I/Os. Thus, as the size of the DBMS buffer pool (N) increases with respect to the size of the main memory (M), the number of faults also increases, leading to an increase in the total I/O activity. This has led Tuel to conclude that the paging effect becomes significant and the performance deteriorates as N becomes larger than M.

The effect of using a table which gives an indication of the presence of a data page in the DBMS buffers is studied in [LWF77]. In this work, three models are proposed. In the first one (which we call model A), the buffer replacement policy used is Least Recently Used (LRU). The buffers are searched from most recently used to least recently used and the virtual memory manager uses the Random policy (R) for replacement of frames. In the second model (which we call B), the LRU policy is used both in the buffer replacement algorithm and in the memory replacement algorithm. In addition, the memory LRU stack is not updated during a buffer search. This is done so that the ordering of the frames in the LRU stack is more related to the actual sequence of references shown by the queries. The third model (model C) is quite similar to the model proposed by Tuel except for the addition of a prefix table in the main memory to indicate which data pages are in the DBMS buffers. Since the table is always in the main memory, faults will not be generated by a search of the table. In this model, the LRU replacement policy is also used in both the buffer replacement algorithm and the page replacement algorithm.

The results of the [LWF77] study show that

$$I/O\ cost_{model\ C} < I/O\ cost_{model\ B} < I/O\ cost_{model\ A}$$

given that N is greater than $(M + 1)$. In this study, empirical results are also obtained from the above models. It is found that in situations where the main memory is shared dynamically between the buffers and the other active tasks in the virtual memory system, it is advantageous to increase N. In addition, the use of the LRU policy in both replacement algorithms is more advantageous than using a combination of the LRU and R policies. The use of a prefix table is shown to give a significant improvement in I/O cost. In all three models, it is found that increasing N will reduce the I/O cost if $r > 1$. The final conclusion is that Tuel's results hold only if $r = 1$ and M is fixed at a value (that is, memory is statically allocated to the DBMS buffer manager). In all three models, if M is varied (that is, if the DBMS is competing with other tasks in the system for frames), the I/O cost will be reduced.

2.1.2 Effects of Main Memory Partitioning

Brice and Sherman [SB76,BS77] further expand the studies to include the effects of dynamic and static partitioning of the main memory manager and the compatibility of the various buffer replacement algorithms with various memory replacement algorithms. In [SB76], the main memory is shared dynamically between the DBMS buffers and the code running the DBMS, and four factors are studied: the buffer management algorithm, the DBMS buffer pool size, the main memory size and the memory replacement algorithm. The replacement algorithms studied are First In First Out (FIFO), Random (R), Second Chance (SCH) and LRU. The main memory size ranges from 72 frames to 96 frames, and the buffer pool size ranges from 1 to 20 pages. It is found that for all values, the LRU replacement algorithm (in both replacement of buffers or frames) gives a better performance than SCH while the R buffer replacement algorithm is found to give the best performance. It consistently produces a lower double paging rate but a higher number of reference faults. The higher paging rate is lowered when R is also used as the memory replacement algorithm. This is due to the fact that the buffer size used in the study is not large enough to contain the locality of the queries and thus the LRU policy is not effective. As in other studies, it is found that the buffer I/O decreases as N increases thus leading to the conclusion that if N is increased to be greater than M, the resulting performance advantages will overcome the effects of the resulting double paging.

The above study is extended to a static partitioning of the main memory [BS77]. In this study, the main memory is statically partitioned into system frames and buffer frames. When a page fault occurs, a frame is chosen from the partition in which the fault occurs. That is, frames in one partition are not considered for replacement by the replacement algorithm when a fault occurs in the other partition. The buffer replacement algorithms tested are FIFO, R, SCH and LRU. The memory replacement algorithms considered are R and SCH. The buffer partition size and the DBMS buffer pool size considered are 1, 5, 10, 15 and 20 frames/pages. The main memory size ranges from 80 frames to 96 frames. It is found that there is a peak in the buffer I/O when the buffer ratio (N/M) increases to just above 1. This is due to an increase in double faults or reference faults. Therefore, to reduce the double fault component, N has to be significantly larger than M. This will lead to a decrease in the buffer faults thus leading to a decrease in double faults. The total I/O decreases when M increases and/or when the system partition size becomes larger. When the system partition size becomes larger, though the buffer I/O increases, this increase is not as significant as the decrease in system I/O. It is also found that the variation in the buffer I/O is mostly caused by the buffer replacement algorithm rather than the memory replacement algorithm. For the system partition, SCH is found to be a better memory replacement algorithm and for the buffer partition, the combination of R buffer replacement algorithm and R memory replacement algorithm is found to give the best performance. R performs better because the page reference pattern of the database used is characterized by a few highly referenced database pages, separated by strings of references which are similar to [SB76]. Since the buffer pool size is not large enough to contain the locality of the queries, the LRU replacement algorithm does not perform as well.

2.1.3 Effects of Page Replacement Algorithm

In a second experiment performed by Fernãndez, Lang and Wood [FLW78], the effects of the memory replacement algorithms on a DBMS running in a virtual memory system are studied. The buffer replacement algorithm is fixed as LRU; the memory replacement algorithms tested are LRU, R and generalized LRU. In this study, only one DBMS user is assumed. As in [BS77], a buffer I/O peak is found when *N* is just greater than *M* for the case of an LRU memory replacement algorithm. As in [LWF77], the R policy generally produces better performance than the LRU policy when the locality set of the database cannot be contained in the memory. The paging rate of generalized LRU is found to be smaller than that of the LRU policy. As in all other cases, the I/O cost decreases with increasing buffer pool size.

2.1.4 Effects of Buffer Partitioning

In a later study, besides studying the effects of static and dynamic partitioning of the main memory, Effelsberg and Haerder [EH84] also perform some experiments on local and global buffer allocation policies and local and global replacement algorithms. It is shown that the LRU replacement policy is superior to Clock, FIFO and R. An allocation policy is one which determines whether a frame in main memory is to be allocated to a process/transaction/query. Though the local allocation schemes have buffer fault rates similar to the global allocation schemes, they are not appropriate for an interactive DBMS application because of long user response time, extra overhead in handling the buffer frames for shared pages and an increase in complexity with an increase in the number of active transactions. For the study on static and dynamic partitioning, it is shown that the dynamic scheme gives better performance than the static scheme and the buffer fault rate for a three partition system (separated into (DBMS) system, access paths and database) is slightly better than that of a two partition system (separated into (DBMS) system and database inclusive of access paths) or global LRU.

2.2 Summary

In this chapter, we review the previous research work in buffer management in DBMS. All of these studies concentrate on optimizing the execution of a DBMS on top of a virtual memory operating system. The compatibility of using some common page replacement algorithms such as FIFO, LRU and R for DBMS buffer replacement is also studied.

We can conclude that the use of a prefix table to indicate the contents of the buffers is beneficial. Also, in general, when the DBMS is running on top of a virtual memory system, the DBMS buffer pool size should be significantly larger than the size of the real memory allocated to the DBMS buffer pool so that the I/O cost can be reduced. The LRU policy should be preferred to others for both the buffer replacement algorithm and the memory replacement algorithm. This is beacuse although the results from the Brice and Sherman studies have shown that R is a better policy, [LWF77] states that LRU can be shown to be a better replacement algorithm if N is increased to above 20 pages in their studies. Also, the studies of [EH84] and [LWF77] have shown that LRU is the superior replacement algorithm in the DBMS buffer manager. In [EH84], it is also found that the dynamic allocation of main memory gives a better performance than that of a static allocation scheme.

Chapter 3

Special Purpose DBMS Buffer Management Algorithms

3.1 Introduction

In this chapter, we discuss two buffer management schemes which have been designed specifically for DBMS use. The main feature of these two schemes is that they incorporate the knowledge of the page reference pattern of a query to be executed into the buffer manager. Most memory managers in general purpose operating systems assume that the page reference pattern of a process is not known but assume that some form of locality of reference will be exhibited by the process. For the general purpose processes, these assumptions are generally valid but for a database process, they are not. In many of the cases, not only the page reference pattern but also the optimal buffer space can be determined before each query is executed. This can be done by inspecting the type of query and the databases the query has to access when it is in execution. In order to incorporate this knowledge, the query optimizer and the buffer manager are required to cooperate with one another. The optimizer, with the knowledge of the number of free buffers available, will be able to choose the best path to execute a query with the limited number of free buffers. According to the chosen best path, the page reference pattern and the optimal buffer space can be determined. For a given page reference pattern, the most suitable buffer replacement algorithm can be determined. In the following, we shall discuss two methods which have been proposed to estimate the buffer requirement and/or the buffer replacement algorithm for any particular query.

3.2 Hot Set Model

This model is introduced by Sacco and Schkolnick [SS82], and is further refined in [SS86] for a relational DBMS. The Hot Set model attempts to estimate the run-time buffer space requirement of a query (when it is being executed) before it starts execution. This buffer space requirement does not change as the query is being executed.

The purpose of the Hot Set model is to reduce internal and external thrashing. *Thrashing* [Den68a] is a term used to describe a situation in which the resources are not engaged to execute active processes, but are engaged in servicing system operations which are produced as a result of processes competing for resources. In the case of buffer management, it is caused by not having enough buffers to satisfy the buffer needs of the active processes. Internal thrashing is local within a process. It happens when every new reference to a page causes a fault. This happens because the number of available buffers is not large enough to contain the locality set of the process. External thrashing occurs in a multi-user environment. For example, when a process steals buffers from another one which has a looping behavior, the latter process will generate a fault for every page reference.

The key idea behind the Hot Set model is the observation that for each query running in a DBMS which has an LRU buffer replacement policy, the number of faults generated by a query as a function of the available buffer space is a curve consisting of a number of *stable intervals*, separated by a small number of discontinuities, called *unstable intervals* (Figure 3.1). Inside

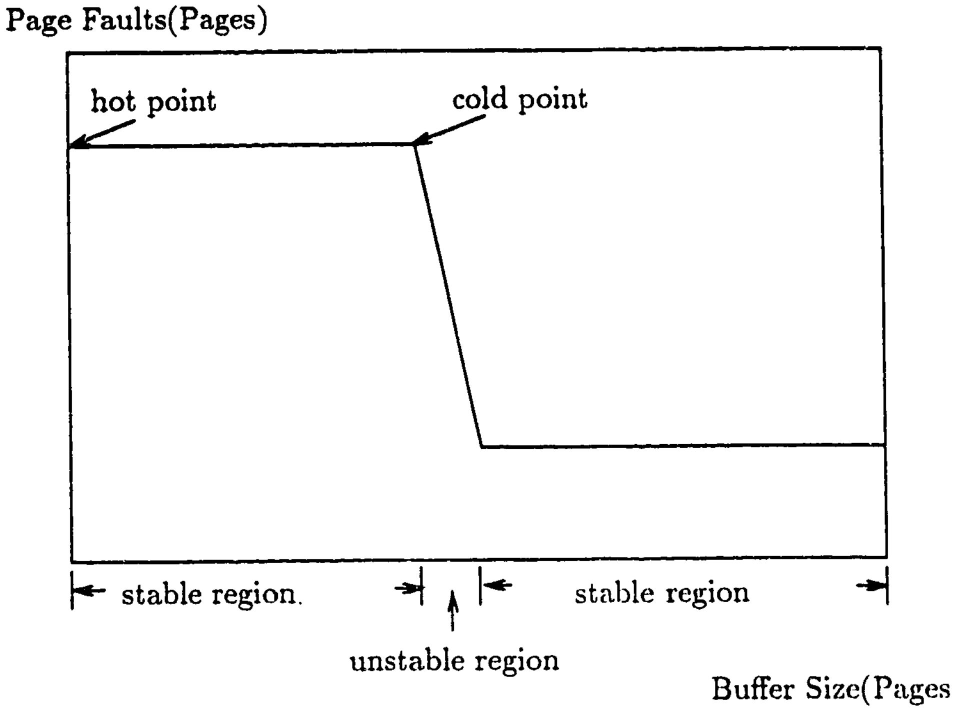

Figure 3.1: Determination of Hot Points

a stable interval, the number of faults is a constant. The lower extreme of a stable interval is called a ***hot point*** and the upper extreme is called a ***cold point***. It is called the cold point to indicate that this buffer size should never be chosen for the execution of this query. If it is chosen, the process will be consuming more buffer resources than is necessary without any reduction in faults. In fact, the buffer size at the cold point will produce as many faults as that at the hot point although the hot point indicates a smaller buffer size. Therefore, the hot point is the optimal buffer size of the query. Notice that the hot point of a stable interval (except for the minimum hot point) is the upper extreme of an unstable interval and the lower extreme of an unstable interval is the cold point of a stable interval. In all cases, only the buffer sizes of the hot points inside stable intervals of a curve should be chosen for the execution of that query. Otherwise, buffer resources will be wasted without any fault reduction.

In order to identify the hot points and the stable and unstable intervals for a given query, Sacco and Schkolnick classify the different types of page reference patterns exhibited by queries when they are executed in a LRU buffer replacement policy environment.

Besides giving a classification of the different types of page reference patterns, they also propose a buffer management algorithm which is based on the LRU replacement policy. We next give a description of the classification, followed by the algorithm.

3.2.1 Hot Set Page Reference Classification

In the Hot Set model, the page reference patterns are classified into *simple reusal, loop reusal, unclustered reusal* and *index reusal.*

Simple Reusal

In simple reusal, once a page is referenced and has been replaced, it will never be referenced by the same query again. Sequential scans exhibit such a reference behavior. Therefore, the buffer size requirement for a query exhibiting such a page reference pattern is exactly one. Thus, the hot point is one.

Loop Reusal

Queries which reference pages in a loop are classified into this category. For example, a join between two relations R_1 and R_2 which is being executed by a nested loop method using sequential scans. In such a case, there are two hot points : 1 and $(1 + P_2)$ where P_2 is the size of (number of pages) R_2. These hot points can be extended to n-way joins. Another example is the merge scan join. In this case, looping occurs on runs of equal values of R_2, matching a run of equal values in R_1. The hot points in this case are 1 and estimated to be 1 + *rlen2*, with

$$rlen2 = \lceil \frac{P_2}{NDV_2} \rceil$$

or

$$rlen2 = \lceil \frac{P_2}{P_1} \rceil$$

where NDV_2 is the number of distinct values in the range of the join attribute of R_2 and P_1 is the size of R_1. The second estimate is used when NDV_2 is not known.

Unclustered Reusal

Queries which use only an unclustered index over a relation are classified into this category. In such a case, for each index-leaf level entry which satisfies the predicate, the corresponding tuple is accessed. Since the index is unclustered, each entry might point to a different page or to a page which has been referenced. Therefore, an estimate of the number of the unique pages to be accessed is required. In the Hot Set model, Yao's function [Yao77] is used for such an estimate.

Yao's function:

$$Yao(m,p,k) = \begin{cases} m(1 - \prod_{i=1}^{k} \frac{(n-p-i+1)}{(n-i+1)}) & \text{if } k \leq (n-p) \\ m & k > (n-p) \end{cases}$$

n = number of tuples = K_2

m = number of blocks or pages containing n tuples = P_2

p = number of tuples in each page

k = number of tuples randomly selected from n tuples = KP

$Yao(m,p,k)$ = expected number of blocks or pages hit.

The hot points are 1 and $(1 + Yao(P_2,\ K_2/p,\ KP))$ where KP is the number of index entries satisfying the index predicate predicted by the query optimizer, and K_2 is the cardinality of R_2.

Index Resual

Queries which use an index repeatedly are classified into this category. For example, in the evaluation of a nested loop join, R_1 is processed by sequential scan and R_2 by clustered index scan. In this category, two cases can be observed. The first occurs when both the outer and inner relations are ordered on the joining attribute(s) in the same manner. In this case, there is no reusal of the pages making up the nonleaf levels of the index tree. At the leaf level, there will be a looping behavior. Therefore, the hot points for this case are 1, $(1 + DI_2)$ and $(DI_2 + rli)$, where

$$rli = \frac{PI(leaf\ level)}{NDV_2}$$

and PI is the number of pages at leaf level i and DI_2 the depth of the index on R_2.

The second case is when the outer relation is not ordered. In such a case, the hot points are 1, $(1 + DI_2)$ and $(1 + T + T(leaf\ level))$ where

$$T(i) = Yao(NDV_1, NDV_2, PI(i))$$

$$T = \sum_{i=1}^{leaf\ level-1} T(i)$$

Summary

Though the classification given above is not exhaustive, it can be used to characterize other access strategies. In special cases where temporary results have to be computed and stored, the query is decomposed into several sub-evaluation plans which can be independently characterized by the Hot Set model.

3.2.2 Hot Set Buffer Management Strategy

Besides proposing the Hot Set model, Sacco and Schkolnick have also proposed a buffer management algorithm which implements the ideas of the Hot Set model. In this section, we shall give a description of the buffer manager used in the study [SS86]. This manager will be one of the two implemented in our study.

The Hot Set buffer replacement algorithm uses a local LRU replacement policy. Buffers are allocated to processes. Each buffer can have only one owner. Two numbers are associated with each process in the system: the hot set size determined by the Hot Set model before the process is executed (*P.HS*) and the number of buffers currently allocated to the process (*P.NALL*). Also, included in each process is a LRU stack (*P.LRU_Stack*). This stack is used by the buffer replacement algorithm to choose a buffer local to the faulting process for replacement. If *P.NALL* is less than *P.HS*, the process is said to be deficient. If a deficient process faults, the buffer manager tries to obtain a free buffer from a LRU ordered list. This list, called a *free list*, contains all the buffers in the system which are not allocated to any process. If the free list is empty, then a buffer is chosen for replacement among the local buffers of this deficient process. When a non-deficient process faults, a buffer is chosen from its own local buffers for replacement.

Initially, all buffers are free and on the free list. When a new process enters the system, *P.HS* is set to its hot set size, *P.NALL* is set to zero and the *P.LRU_Stack* is set to null. When a data page is referenced, a *global prefix table* is searched. If the page is in a buffer belonging to the process, the local LRU stack is updated. If the page is in a buffer which belongs to another process, nothing is done. If the page is in a free buffer and the process is non-deficient, a buffer will be chosen from the local LRU stack and inserted into the free list. Then, the requested page/buffer is added to the local LRU stack. If the process is deficient, no buffer is taken from the local LRU stack. The free buffer is just added to the local LRU stack. Whenever an addition or deletion of a buffer to/from the local LRU stack takes place, *P.NALL* is also updated. When a process terminates, its LRU stack is appended to the free list.

3.3 Query Locality Set Model

The Query Locality Set Model (QLSM), proposed by Chou and DeWitt [Cho85, CD86], not only tries to estimate the buffer space requirement of a database query before it starts execution, it also tries to determine the appropriate type of buffer replacement algorithm for that query. The estimates given by the QLSM are more dynamic (in terms of a query's buffer replacement policy and optimal buffer size at any given time) than that of the Hot Set model for the following reason. In QLSM, the different page reference patterns exhibited at different stages of a query in execution are taken into account. Rather than observing that a query being executed in a DBMS with a particular buffer replacement algorithm generates a particular fault curve, QLSM makes the observation that a query can exhibit different page reference behaviors when accessing different relations/files. With each file being accessed by a query, a locality set is associated. This locality set has its own buffer replacement policy and its own buffer space requirement. When no further access to this file is required by this query, the file is closed and the locality set is disassociated from the query. Therefore, each query will have as many locality sets as the number of files it currently has open. As the query is being executed, it opens and closes files. Therefore, its buffer space requirement and buffer replacement policy vary dynamically.

The page reference behavior to each file can be characterized by the operation of the query accessing it. In a relational database system, these operations are limited and their page reference behaviors are very regular and predictable. QLSM makes use of the regularity and predictability of these operations. In the following, we will give a brief discussion of the various classifications of the page reference behaviors of these operations, followed by the buffer management algorithm.

3.3.1 QLSM Page Reference Classification

QLSM classifies the page reference patterns exhibited by relational DBMS operations into *sequential, random* and *hierarchical.*

Sequential References

Sequential references can be classified into *straight sequential* (SS), *clustered sequential* (CS) and *looping sequential* (LS). In a SS reference, only one buffer is required because once a page is referenced, it will never be referenced again. Operations which can be included in this classification are projections, selections without index and simple aggregations such as MIN, MAX, SUM and AVG.

In a CS reference pattern, a group of records is referenced repeatedly, such as the records belonging to the inner relation in a merge scan join. In such a case, the locality set size or buffer space requirement is

$$\left(\frac{size\ of\ largest\ cluster}{blocking\ factor} + 1\right)$$

and the buffer replacement algorithm is FIFO or LRU. A selection with clustered index also has a CS reference pattern.

LS occurs especially in nested loop joins where the inner relation is scanned for every tuple in the outer relation which satisfies the predicate. Therefore, the inner relation file should be kept in memory. Similar to CS, the LRU policy should be used for buffer replacement. If the file is too large, then the buffer space requirement is a few buffers and the Most Recently Used (MRU) replacement policy is adopted.

Random References

In random references, the policy used in the buffer replacement is irrelevant since the reference pattern consists of a series of independent accesses. Two types of random references exist: *independent random* (IR) and *clustered random* (CR). An IR reference pattern is exhibited when a non-clustered index is used to access data pages, for example in a selection with non-clustered index. CR is exhibited especially in joins where the outer relation has a clustered and non-unique index and the inner relation has a non-clustered and non-unique index. The locality set size for IR is a function of Yao's formula, whereas for CR it is the number of records in the largest cluster.

Hierarchical References

The difference between hierarchical references and sequential references is that in the former, an access to an index is required. There are four types of hierarchical references: *straight hierarchical* (SH), *hierarchical with straight sequential* (H/SS), *hierarchical with clustered sequential* (H/CS) and *looping hierarchical* (LH). In SH, only one buffer is required since the index is traversed only once. In both H/SS and H/CS, after the index has been traversed, a sequential scan on the leaves will follow. Therefore, both of them have requirements similar to SS and CS.

In LH, as in LS, a relation is repeatedly scanned. The difference is that in the LH, the relation is indexed on the join field and thus the index will be repeatedly scanned too. Due to the large fan-out factor in most indices, the root page might be the only one worth keeping, or use a LIFO (Last In First Out) replacement policy with a few buffers.

Summary

QLSM is an exhaustive classification of the page reference patterns of the various operations available in a relational database system. The main difference between QLSM and the Hot Set model lies in the fact that QLSM does not tie a particular replacement algorithm to a query. In fact, the buffer space requirement and the replacement policy of a query change as it is being executed. Besides QLSM, Chou and DeWitt also proposed a buffer management algorithm, called DBMIN, using QLSM. This algorithm is discussed in the next section.

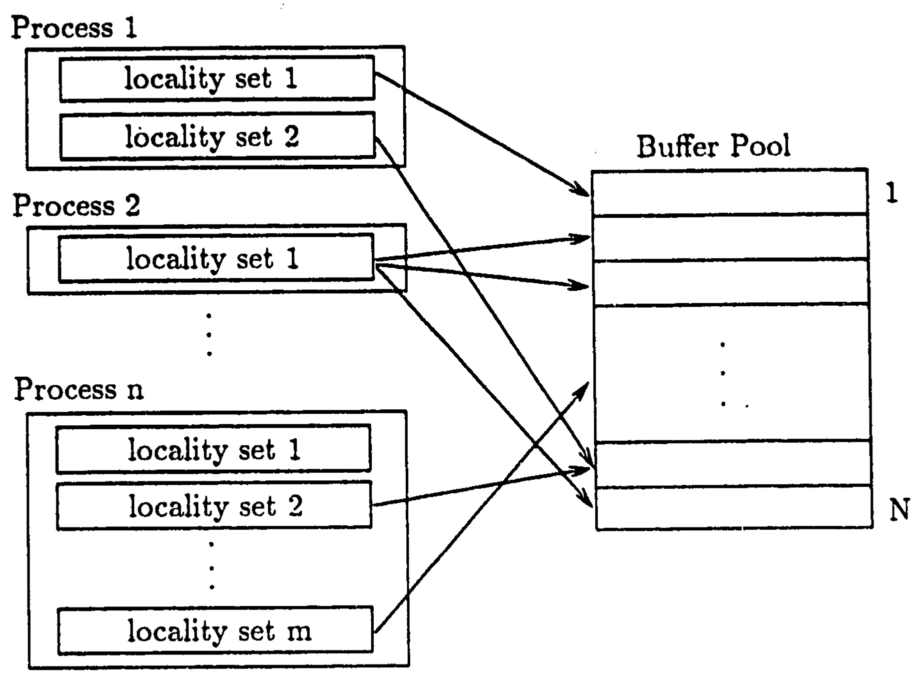

Figure 3.2: Organization of locality sets in QLSM

3.3.2 DBMIN Buffer Management Strategy

Similar to the Hot Set buffer replacement algorithm, DBMIN is also a local replacement algorithm. But, unlike the Hot Set algorithm which considers the buffers within a process as a unit, DBMIN separates the buffers in the process into groups, called locality sets, where each locality set contains the set of buffers which are used to contain pages from a certain file. Each of these groups has its own locality set size and its own replacement policy depending on the pattern of accesses to the file by this process. The replacement policy of a process changes dynamically, depending on which file it is currently accessing. The total buffer requirement of a process also changes dynamically, depending on how many files it has open currently. A process will have as many locality sets as the number of files it accesses. Figure 3.2 gives a representation of the organization of the locality sets of the processes.

Associated with each locality set j for a process i is r_{ij} which is the number of buffers allocated to this locality set, and l_{ij} to indicate the locality set size determined by QLSM. When a file is opened, the locality set size and the replacement policy are given to the buffer manager. A load controller will then check if $\sum_i \sum_j I_{ij} < N$. If so, then the process may proceed, otherwise it is suspended. When a process/query is suspended, its buffers are released and the query is placed at the front of a waiting queue.

When a page is requested by a process and the page is in the buffer pool, the following senario takes place. If the page is found in the locality sets of the process, the statistics are updated.[1] If the page is not found in the locality sets but it belongs to another process, nothing is done. If the page is not in the locality sets and it does not belong to anyone, it is added to the faulting locality set. If $r_{ij} > I_{ij}$, the replacement policy of the faulting locality set will choose a buffer to be added to the free list. If the page cannot be found in the buffer, an I/O request has to be made to bring in this page. A buffer is chosen for replacement if $r_{ij} > I_{ij}$.

When a file is closed, the buffers of the corresponding locality set are returned to the buffer manager. The load controller will then select the first waiting process which satisfies the condition $\sum_i \sum_j I_{ij} < N$.

3.4 Summary

In this chapter, we review two strategies which have been proposed for the DBMS buffer manager. Their theories and buffer replacement algorithms are described.

Chou has done extensive performance studies on various buffer replacement algorithms (including the Hot Set model and the QLSM). He finds that the DBMIN buffer manager constantly out-performs the others (in terms of throughput). No other performance study has been done to confirm his claims. Though the results from both buffer algorithms seem promising, no study has been performed on running these algorithms in a virtual memory system. Therefore, in this thesis, we shall attempt to confirm Chou's claims and,

[1]The type of statistics update depends on the replacement policy of the locality set.

at the same time, study the performance of the two buffer managers running on top of a virtual memory system.

Chapter 4

Study Environment

4.1 Introduction

In this chapter, the simulator which is used to study the performance of the various buffer management algorithms, as well as the experimental set-up for this study are described. Simulators can be probabilistic or trace driven. Since no probabilistic approach has been devised for the DBMS address reference pattern, we use the trace driven approach in our simulator. The simulator is basically an emulator with a preprocessor. The emulator consists of a page address generator and a computer model while the preprocessor consists of a query generator. The queries are generated according to a pre-determined mix. Before a query begins execution, its page reference addresses are generated by the page address generator. This query is then executed in the computer system. Figure 4.1 gives a graphical view of the environment. We shall give a description of the database model in the first section, followed by a description of the computer configuration. The criteria of the performance studies will also be discussed in this chapter.

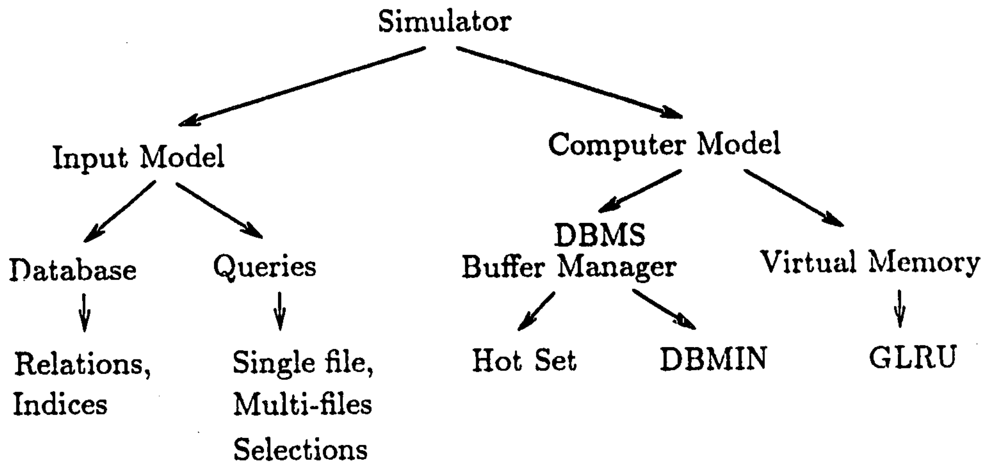

Figure 4.1: Study Environment

4.2 Input Model

The page reference addresses of each query are generated by tracing the execution path of the query as though it was executed in a DBMS. For example, when a sequential scan on a relation is required, the access path is either a sequential scan of the database or a sequential scan on the leaves of the index of the relation. The page number is recorded for every address reference to a file made by the executing query. This trace will produce a copy of the addresses of the database pages accessed during the execution of this query. After this trace is obtained, the query is executed in the simulated computer system.

4.2.1 Database Model

The database which the queries access is made up of two relations with five thousand tuples each. Each tuple has four columns: two columns with unique values and two columns with duplicates. The first and third columns of the relations are sorted. The unique columns contain numbers ranging from 0 to 4999, and that of the non-unique columns from 0 to 490. In the unsorted columns, the numbers are placed in their positions through a uniform random number generator. Integers, rather than character strings, are used so that the selectivity of each query can be computed easily. Some columns are not sorted

Sorted	Unsorted	Sorted	Unsorted
0	3709	0	30
1	45	0	100
2	36	.	340
.	231	0	50
.	.	1	60
.	.	1	40
	.	.	:
		490	100
4999	678	490	20

Figure 4.2: A View of the Database used in the Study

and some contain non-unique numbers so that we can build a combination of clustered, non-clustered, unique and non-unique indices from the database. An index on a relation is clustered if the relative physical location of the tuples of the relation is similar to the relationship between the values of the indexed field. Similarly, an unclustered index is one in which the relative physical location of the tuples of the relation is not similar to the relationship between the values of the indexed field. This database design is a modification of the one in [BD84]. Figure 4.2 gives a view of the database.

4.2.2 Indices

The indices built on the above database are a special type of *B-tree* [Com79, BE77,BM72], called B^+-tree, as shown in Figure 4.3. Each B^+-tree contains an index and a sequence set. The index is organized as a *B-tree*. Each node in the index, unlike *B-trees*, contains only the key values. They only serve as a guide to the leaf nodes. Therefore, the key values in the index do not need to be unique. Similar key values can be found at different levels of the tree, but within each level, the key values are unique. At the leaf level of the B^+-tree, the key values in the nodes are unique. The nodes at the leaf level, called the sequence set, are linked so that common operations such as fetching

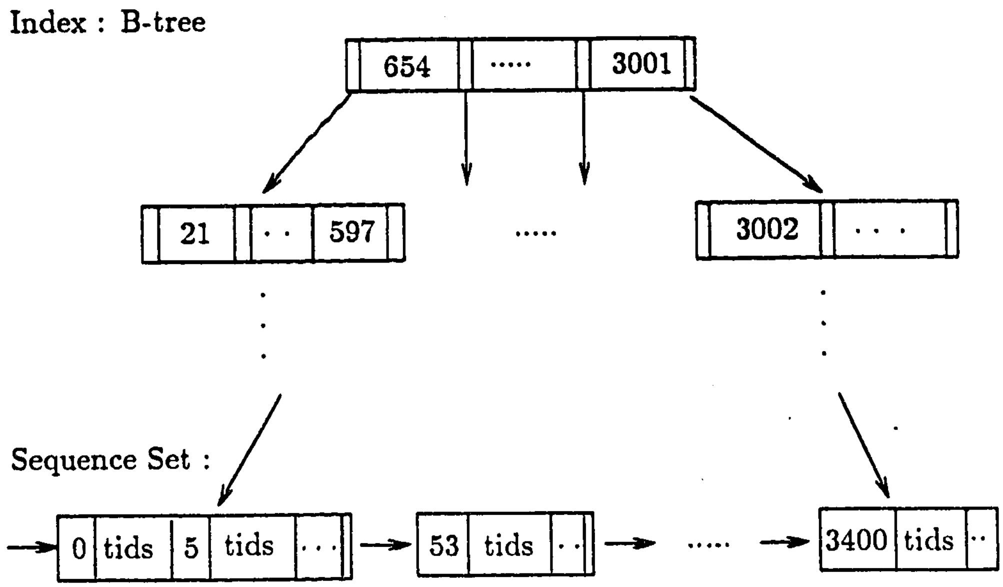

Figure 4.3: A $B^+ - tree$

the next record in sequence order or a sequential scan of the relation can be easily performed. Only the leaf level nodes contain tuple identification which contains the position of the tuple in the database, such as the block number and the position within that block. When a key is deleted from a relation, the corresponding value in the index need not be deleted since it only serves as a guide. In searching for a key, the search does not stop when a similar key value is found in the index, rather the nearest right pointer is followed and the search proceeds all the way to the leaf. For example, in Figure 4.3, if we are searching for tuples with key value 5, the left-most pointer of the node 654 at the root level is followed since 5 is less than 654. Then, at the next level, the left most pointer of node 21 is chosen since 5 is less than 21. When the sequence set is reached, the node containing the key value of 5 will be pointed to by the node last accessed at the upper level in the search for key value 5.

In our system, each node, whether it is an index node or a sequence node, is contained in a page. In each leaf page, there are sets of pairs. A pair contains the key value and the tuple identifiers of the tuples with similar key value. In each index page/node, there are n key values and $(n+1)$ pointers. The left-most pointer of each key points to the page which contains all the keys which have values less than the key value (inclusive) but greater than the previous key value and less than the next key value. Therefore, unlike the above case, when searching for a key, we follow the nearest left pointer when a similar key value is found in the non-leaf level. Each node/page contains only a maximum of fifty key values. Thus, in our database, the indices of the unique fields have a depth of three while that of the non-unique fields have a depth of two. Figure 4.3 also shows this environment.

4.2.3 Query Model

In this section, we describe the queries that are used in the simulation. They consist of simple selections and joins. These two operations are chosen since a selection permits us to simulate accesses to a single file whereas a join enables the simulation of multi-file accesses. Two methods are used to compute the joins: the merge scan method and the nested loop method.

As noted before, each relation contains four different indices built on the four attributes. They are: clustered and unique index for the first field, non-clustered and unique index for the second field, clustered and non-unique index for the third field and non-clustered and non-unique index for the fourth field. Indices might not always be available in a real situation, therefore besides using the indices as an access path to a relation, we also have queries which require a sequential search of the whole database. Note that the database contains data pages and index pages. Each data page contains fifty tuples and each index page contains at most fifty key values and additional pointers or tuple identifiers. Therefore, with 10,000 tuples in the database, the number of pages containing only database tuples is 200. The total number of pages in the database is 976.

Simple Selections

In the simple selection queries, the access paths available are sequential scan, clustered index scan and non-clustered index scan. In the sequential scan, the

Inner \ Outer	No Index	Clustered Index	Non-Clustered Index
No Index	×	×	×
Clustered Ind.	✓	✓	✓
Non-Clust. Ind.	✓	✓	✓

Table 4.1: Access Paths for a Nested Loop Join

whole database, that is 10,000 tuples, are searched in sequence since tuples of the two relations are placed in the database without any order. In addition, in a real situation, the relations will not remain clustered after a number of insertions and deletions of records.

According to the Hot Set model, the buffer requirement of a query using the sequential scan access path is one. If an index is used (regardless of whether it is clustered or not), then two buffers are required.

In the QLS model, the number of locality set in a query with a sequential scan access method is one with a simple replacement policy which replaces the only page in the locality set on a DBMS buffer fault. If an index is used, then the query has two locality sets, one for the index and the other for the DBMS data page. The buffer requirement of both locality sets is one with the simple replacement policy.

Nested Loop Joins

In Table 4.1, the access paths available in executing a nested loop join are shown. In a nested loop join, for every tuple in the outer relation, the inner relation is scanned completely for tuples which match the join predicate. Note that when the inner relation does not have an index on the join field, the nested loop method is never used to execute the join. This access path has been shown to be very inefficient [ML86].

As can be seen from Table 4.1, access paths exist only in cases where the inner relation is indexed. When the outer relation is not indexed and the inner relation is accessed by a clustered index, then the Hot Set model states that the minimum buffer requirement is the height of the index tree, plus a buffer to contain a DBMS data page and another one to contain a page for the outer relation. If the index to the inner relation is not clustered, then Yao's formula is used to estimate the number of pages which will be accessed in using the index. The buffer requirement in this case is given by Yao's formula plus one buffer for the outer relation plus the height of the index tree. If the outer relation has an index and if more than one tuple from the outer relation satisfy the predicate, one is added to the number calculated from the above formula to contain the index page for future reusal.

In the QLS model, the same formula is used to calculate the buffer requirement regardless of whether there is an index for the outer relation. The only difference is that an additional locality set is required for the index of the outer relation. This locality set has a buffer requirement of one with a simple replacement policy. When there is no index on the outer relation, only three locality sets are required since the query is accessing the outer relation without any index and accessing the inner relation with an index. The first one has a buffer requirement of one with a simple replacement policy. This locality set contains the data pages from the outer relation. The second and the third locality sets are used to access the inner relation, one for the index and the other for the data. Their specifications depend on the type of the join attribute of the outer relation. If the outer join attribute is sorted and not unique, then the replacement policy of both the second and third locality sets is LRU, with the second one having a buffer requirement of the tree height and the third one a buffer requirement of the largest cluster divided by the blocking size plus one. If the outer join attribute is sorted and unique and if the number of tuples in the outer relation which satisfies the predicate is more than one, then both locality sets have the same specifications as the above. On the other hand, if only one tuple satisfies the predicate, then both locality sets have a simple buffer replacement policy with one buffer. In the other cases, the buffer

Inner \ Outer	No Index	Clustered Index	Non-Clustered Index
No Index	✓	✓	✓
Clustered Ind.	✓	✓	✓
Non-Clust. Ind.	✓	✓	✓

Table 4.2: Access Paths for a Merge Scan Join

requirement of the second locality set is two with a LIFO replacement policy and the buffer requirement of the third is determined using Yao's formula and they are managed with a LRU replacement policy.

Merge Scan Joins

The merge scan join method is considered to be the most efficient access path in most cases [BE77]. In this method, the relations involved are first scanned via the available access paths during which the tuples which will participate in the join are collected into corresponding temporary files. These files are then sorted via a combination of merge sort and quicksort. Quicksort is used to sort the tuples in a page and merge sort is used to combine the sorted pages. Then, the sorted temporary files are scanned with respect to their join fields. Since the files are sorted, there is no need to rescan those tuples which have been scanned in the inner relation. All that is required is a pointer to the last tuple scanned in the inner relation. This pointer can be moved forward or backward depending on the current join values of the outer and inner relations. Table 4.2 shows the access paths available to execute a merge scan join.

The minimum buffer requirement of a query using the merge scan join using the Hot Set model is two, with an addition of one if the number of tuples which satisfies the prior selection predicate is greater than one and at least one index is available to access either the outer or inner relation. This is because at any moment, a maximum of three files are opened: either the index file, the data file and the temporary file or two temporary files.

Query Type	Access Paths	Outer Relation	Inner Relation
1 2	Simple Selection	NI NCI	NI NI
3	Merge Scan Join	CI	CI
4 5	Nested Loop Join	NCI CI	CI CI

NI : no index

NCI : non-clustered index

CI : clustered index

Table 4.3: Types of Queries Studied

In the QLS model, the maximum number of locality sets is six each of which uses a simple replacement policy, and a buffer to access the indices, the data files and the temporary files. If the number of tuples in the outer relation which satisfies the predicate is more than one, then the last locality set has a buffer requirement of the size of the largest cluster divided by the block size and has the LRU replacement policy so that the locality of the inner temporary file can be captured.

As can be seen from the above, in some cases, similar formulae are used to determine the buffer requirements of different types of queries. The type of query is determined with respect to its access path. For example, in the merge scan join method, at least three types of queries share the same formula. In order to eliminate redundancy and reduce the number of runs we have to make, we conduct performance studies only on those queries which have unique specifications. Table 4.3 gives a list of the queries which are studied.

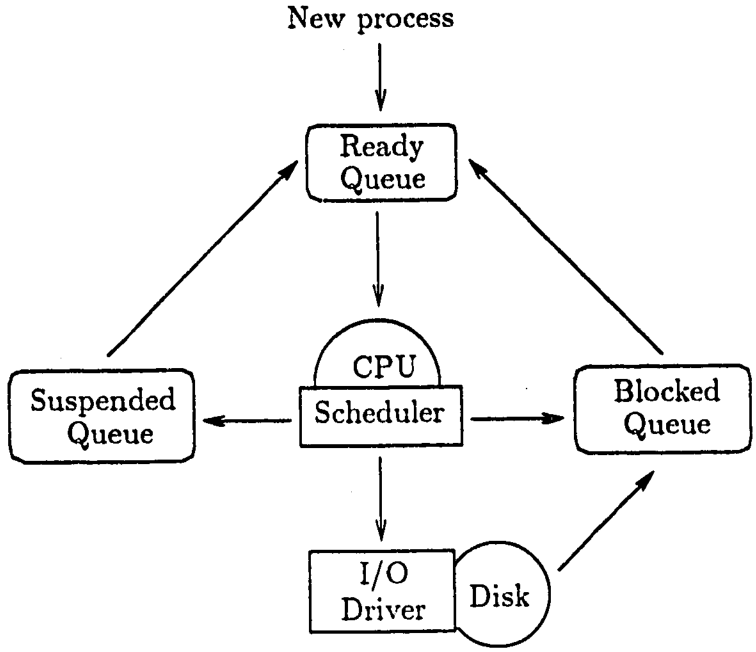

Figure 4.4: Model of the Computer System

4.3 Computer Configuration

In our computer system model, the basic hardware components are the central processing unit (CPU), a disk used as database storage as well as the paging device and main memory whose size is fixed at the start of each simulation run. This is a two level memory system. The software components used to handle the above resources are a scheduler, an I/O driver, a memory manager and a buffer manager. The interactions among the various components are shown in Figure 4.4 [MOO87].

In this model, a virtual time unit[1] is used to control event timing. This virtual time unit is the amount of time taken to make a successful page reference. A page reference is successful if the requested page is in the main memory. Otherwise, a page fault is generated. The amount of virtual time needed to read a page from the disk is 30,000 virtual time units. It takes an average of about 30 milliseconds for the file servers on the Sun 3/50 worksta-

[1]The term virtual is used here because the time unit does not represent a real time measurement.

tions to read/write a 4K page from/to the Fujitsu Eagle or Super Eagle disks which are used at the Department of Computing Science in the University of Alberta. In most computer systems, a successful page reference takes about 50 nanoseconds to 1.0 microseconds [Smi81]. Thus, using a ratio of 30,000 in our system is justified.

Scheduler

The scheduler uses a round-robin scheduling policy that is preemptable. A process in the system is in one of the following states at any given moment: *ready, blocked, suspended* or *running*. A process is said to be running if it currently has control of the CPU. A running process can be preempted (i) if it already has held the CPU for the time quantum or (ii) if an interrupt is generated by the I/O driver when a page has been read (written) from (into) the main memory. If it is preempted because of (i), then the process is suspended and queued into a *suspended list*. If the reason for preemption is (ii), then the process is queued into a *ready list*. The interrupt is served and a new process is selected to run.

When a running process makes a reference to a page that is not in the main memory, a page fault occurs. An I/O operation is then scheduled to read this page into the main memory. Meanwhile, the faulting process loses control of the CPU and waits for this page to be read into the main memory. This process is said to be blocked because it cannot do anything until the page has been read in. Thus, it is queued into a *blocked list*. In all three scheduling queues, the FIFO policy is used.

When the I/O for a blocked process has been serviced, the blocked process is removed from the blocked queue and inserted into the ready queue to contest for the CPU.

A new process entering an "overloaded" system will be queued at the end of the ready queue. When a process terminates, the memory that it occupies is returned to the memory manager. Then, the ready list and the suspended list are checked in that order to get the next process which is ready to run. Therefore, the ready processes have priority over the suspended processes.

I/O Driver

We use a very simple I/O driver. It services the I/O requests on a first come first serve basis. As stated before, each I/O request takes 30,000 virtual time units, excluding the waiting time.

Memory Manager

As discussed in Chapter 1, DBMS can be implemented in a general purpose virtual memory operating system environment in two ways. In our computer system model, we have implemented the alternative as shown in Figure 1.3a.

Virtual Memory Manager

In a virtual memory operating system[2], the amount of memory space available to a user is actually much more than the amount that actually exists. For example, in the IBM RT [SH87], a process can reference up to 2^{40} bytes of virtual memory but the main memory size only needs to be a fraction of this. This allows the execution of a process which need not be in main memory completely. In order to facilitate such a scheme, the frames in the main memory are constantly written over with different pages from the secondary storage. That is, frames have to be chosen to contain the pages which are being referenced. It is the role of the virtual memory manager to minimize the page fault overhead and memory requirements of the processes.

Four types of policy are involved in the implementation of a virtual memory system: fetch, allocation, placement and replacement. The fetch strategy determines when to fetch a page from the secondary store into the main

[2]The author assumes that the reader already has some background on virtual memory. This is only a brief summary. Avid readers should refer to [PS85,Dei84,MOO87] for further details.

memory. The virtual memory manager can either fetch a page when it is requested (*demand paging*) or use a mechanism to predict which page a process will request next (*anticipatory paging*). The anticipatory strategy is optimal if the prefetch mechanism is accurate, otherwise, a lot of overhead is involved due to the fetching of wrong pages. In the demand paging strategy, a page will not be read into the main memory unless it is requested. That is, pages are brought into the main memory as they are requested. This produces the minimal amount of overhead. The virtual memory system implemented in this thesis utilizes the demand paging concept for fetching a page.

The allocation policy determines how many frames should be allocated to a process. This number can be static or dynamic. In the former case, each process is allocated a fixed number of frames, regardless of its memory requirement or faulting frequency. Frames do not migrate from one process to another. Before a system is started up, this number is determined. Therefore, the maximum number of processes in the system is fixed. Furthermore, frames not used by one process cannot be allocated to a different process which needs more frames than those allocated to it. In dynamic allocation, the number of frames allocated to each process varies with time and system load. Processes will have different number of frames allocated to them at different stages, depending on their memory requirements and the system load. This strategy allows the maximum number of concurrent processes in the system, bounded by the system load. Some memory managers use a hybrid of the above two methods.

The placement strategy determines the place in the main memory where the fetched data from the secondary store is put. In a paging system, this problem does not exist because the virtual memory and the main memory are divided into partitions of equal size. The only problem is in choosing a page to be paged out. The placement problem is applicable to a virtual memory system which uses only segmentation. In a segmented virtual memory system, the main memory and virtual memory are partitioned into segments whose size varies with time. The number of segments does not remain the same either. Memory, whether virtual or real, is allocated in segments of varying size. Mechanisms are required to determine which segment to allocate to a process to give optimal performance.

When a page fault occurs and a page needs to be paged out, the replacement policy will determine the candidate page. Selecting an appropriate page to be replaced is crucial because if the replaced page is to be referenced in the near future, a page fault will occur and another page has to be paged out to bring in this page. This increases the page fault frequency, thus increasing the paging overhead. Page replacement policies can be classified into global and local. In a global page replacement policy, all the frames are shared among the processes. Processes are allowed to write over frames which are currently being used by another process. In a local page replacement policy, each process has its own set of frames, and will only overwrite frames belonging to itself. Some common demand paging replacement policies are: GLRU, FIFO, LRU, Clock, MRU, LIFO and PFF (page fault frequency) [CO72]. Some of these policies are briefly described in Chapter 1.

In a multi-programming system, there is a conflict of policies between process and memory management. In order to maximize throughput, we want to have as many active processes as possible to utilize the resources to the fullest. But, to the memory manager, in order to minimize page fault overhead, each process should be allocated as much main memory as possible. If we do not control the number of active processes in the system, thrashing can occur. But, if we have too few active processes in the system, throughput can decrease. *Working sets*, introduced by Denning [Den68b,Den70,Den80], can help to determine the optimum point between policies for process and memory management. The working set, $W(t, \Delta)$, of a process, at time t, consists of the set of pages which the process referenced over the last Δ time units. The *Working Set Principle* states that:

A process may execute only if its working set is resident in main

> *memory. A page may not be removed from main memory if it is in the working set of an executing process.*

This implies that if the working set of a process contains those pages currently needed, frequent page faulting is eliminated. By limiting the number of active processes to a group whose working sets fit in the main memory, memory overcommitment is avoided and thrashing can be eliminated. The problem here is in determining the optimal Δ for every process. If the working set principle is followed closely, when a suspended or blocked process is ready to run, its working set pages need to be loaded into the main memory before the process actually starts running. This prefetching is advantageous when the page reference pattern of the process is stable or the phase transition rate is low [Mas77]. An example of the implementation of the Working Set Principle is as follows. When a page is brought in by a process, a counter belonging to this page is set to Δ. At every memory reference generated by this process, the following takes place: (1) if the process references a page that is in its working set, the counter belonging to this page is reset to Δ, and (2) for other pages that are in the working set, the counters are decremented. If the counter of a page reaches zero, the page is expelled from the working set. When a page is expelled from a working set, it is queued into a free list which is managed by a LRU policy. At a page fault, a frame is selected from this list (if it is not empty) and the new data page is copied into this page.

In our study, we have implemented a global page replacement algorithm, the GLRU. In the GLRU policy, all the frames in the main memory are shared by all the active processes. That is, a process can cause a page which has been referenced by another process to be paged out. When a page fault occurs, the least recently used page in the main memory is replaced. Another way of looking at it is that the DBMS is regarded as a process. All the page reference addresses generated by the DBMS queries in execution are considered to belong to this process. The local LRU page replacement policy is used to determine which page should be replaced when a page fault which belongs to this process occurs. But, in actual fact, the page fault is caused by the current running query in the DBMS.

Buffer Manager

The role of a buffer manager in an operating system is to manage the portion of the main memory for the transfer of data between the secondary memory (for example, paging disks, database disks) and the main memory used by the active tasks. When a page, in main memory, which has been written in (dirtied) is to be replaced, the contents of this page is copied to a chosen buffer from its frame. Then, this frame is available and another page can be copied to it. Transfer of data between the main memory and the secondary memory will take place in the chosen buffer. This main memory space used by the buffer manager may be statically or dynamically allocated. In some systems, for example UNIX[3] [Bac86], this buffer pool is common to all executing programs/tasks. That is, a global buffer replacement policy is used.

In our system, we assume that when a page is chosen for replacement by the virtual memory manager, the same frame is used to transfer data between the main memory and the secondary memory. We do not place a distinction between a frame used for the transfer of data between the main memory and the secondary memory and other frames. The virtual memory manager will have total control of the complete main memory. The page replacement policy can be a local or a global one. As stated before, the buffer manager in the DBMS may be managing real memory or virtual memory. If it is managing virtual memory, two buffer managers are involved: one belonging to the DBMS and the other belonging to the virtual memory system. In the DBMS buffer manager, the global or local schemes can be used also. In both of the buffer replacement algorithms proposed by Sacco and Schkolnick, and Chou and DeWitt, a local replacement policy is used. That is, each individual query has its own set of buffers. A description of the two buffer replacement algorithms has been given

[3]UNIX is a trademark of AT&T Bell Laboratories.

in Chapter 3.

4.4 Performance Studies

In this section, we discuss the criteria used in our performance studies. The model is a closed network event driven simulation. Each event record contains a time which indicates when the event is to take place and the type of event that is to take place. The types of events are I/O, scheduling and memory management. In each event class, there are different methods. For example, in the scheduling class, there are methods to block, to suspend and to run a process.

In this study, we perform runs based on the type and access path of the queries. Each run is equivalent to simulating the execution of a thousand queries, with measurements starting from the point where steady state has been reached. The steady state point will be defined later. In each run, the type and access path of the queries are fixed. The predicate(s) of each query entering the system is(are) determined by the preprocessor. Then, the address reference sequence of the query is generated before it is fed to the simulated computer. A random number generator is used to determine the predicates of the queries. The maximum number of concurrent queries is twenty-four and the selectivity of each query has to be less than ten percent. The page reference addresses of each query are stored in a file. On the UNIX operating system, a process can open a maximum of thirty files. Therefore, in each simulation run, we can open only thirty files. We open some files for input of data and output of the measurements and we are left with twenty-four files to store the page addresses of the queries. We have chosen a selectivity of ten per cent since this is the selectivity factor in common database accesses.

In this closed network queue simulation, at time zero, twenty-four queries are generated and they arrive concurrently at the simulated computer. When a query terminates, a new query is immediately fed to the computer.

In this study, the measurements that we are concerned with are the following:

- Mean number of page faults,
- Mean number of (DBMS) buffer faults,
- Mean number of double faults,
- Mean number of I/Os,
- Mean response time,
- Throughput which is defined as

$$Throughput = \frac{number\ of\ processed\ queries}{total\ simulation\ time}$$

The above measurements are taken only after the system has reached steady state. The steady state is reached when two conditions are met: (i) when the DBMS buffers have already been filled with database pages and (ii) when the difference between two consecutive normalized cumulative product of the buffer faults and the time it occurred is less than a threshold. In our study, the threshold is fixed at 0.1. When the above two conditions are met, measurements are taken starting from the next new query until the thousandth query.

The factors that are varied are:

- Type of query,
- Main memory size,
- Buffer pool size,
- Buffer replacement algorithms.

4.5 Implementation

The simulator is written in the C language. It consists of about 6000 lines of code for each buffer algorithm, including the memory manager. The pseudo code used in the simulator is given in Appendix B.

4.6 Summary

In this chapter, we describe the the DBMS environment and the simulator used in our study. The simulator can be divided into two parts: an input model and a computer model. The input model consists of the relations and indices used in our performance study. The computer model consists of a page address generator and a simulated computer system with virtual memory. Queries are fed into a closed network simulator until a thousand queries have been processed.

In the next chapter, we will give an analysis of the results of our performance studies on the two buffer replacement algorithms using the above simulator.

Chapter 5

Analysis of Results

5.1 Introduction

In this chapter, the results of the performance study are presented and discussed. For each algorithm type, we present six graphs: throughput, mean number of I/Os, mean response time, mean number of page faults, mean number of (DBMS) buffer faults[1] and mean number of double faults. The throughput is defined as the number of queries processed per second. The mean number of I/Os represents the summation of the mean number of major page faults and major buffer faults that took place in the duration of each simulation run. The mean response time (seconds) is the average amount of time needed to execute a query. The components of the page faults consist of double faults, minor page faults and major page faults. A page fault is minor if the addressed DBMS page[2] can be found in the main memory but is not in the process' page table; it might not belong to any process but is in the free list. A page fault is major if an I/O is required to bring in the DBMS page. If a process addresses a page that is in I/O, a page fault is considered to have taken place. Similarly, for the buffer faults, if a process addresses a page that is in I/O, a buffer fault

[1]Unless stated otherwise, the term buffer faults is equivalent to (DBMS) buffer faults in all references.

[2]A DBMS page can be either a data page or an index page from the DBMS.

is said to have taken place. A major buffer fault takes place when it is required to bring the page in from the disk. A double fault occurs when both a buffer fault and a page fault occur on a page reference. When a fault occurs and no buffer or frame can be allocated to the faulting process to bring in the data page, the process is suspended.

The results are obtained from simulation runs of about 1000 queries each. To give an accuracy of the results, 95% confidence intervals are computed on the various measurements, $\bar{Y}$, as:

$$(\bar{Y} - 1.96\sqrt{\frac{s^2}{n}} \ , \ \bar{Y} + 1.96\sqrt{\frac{s^2}{n}} \ )$$

where s^2 is the sample variance obtained from the simulation runs and n is the total number of samples. Though samples obtained from simulation runs are usually correlated because the value of one sample can affect the values of other samples, we can assume that the samples are independent if the number of samples is large which is true in our case. For each query type, we generate 3 runs, each with a different random seed number. In each run, about 1000 queries are sampled.

In the first section, the results on simple selections are presented, following that will be the discussions on merge scan joins in the second section and finally the nested loop joins in the third section.

5.2 Simple Selection

In this section, we discuss the results obtained from the simple selection queries. A simple selection query can be executed by three access paths: sequential scan of the whole database, via clustered index or non-clustered index. In the first section, we present the results on the sequential scan path. In the case of indexed access, we have chosen to execute the queries with non-clustered indices since the algorithms used to determine the optimal number of buffers for each query (whether it is via clustered indices or non-clustered indices) are similar in both the Hot Set model and the DBMIN model.

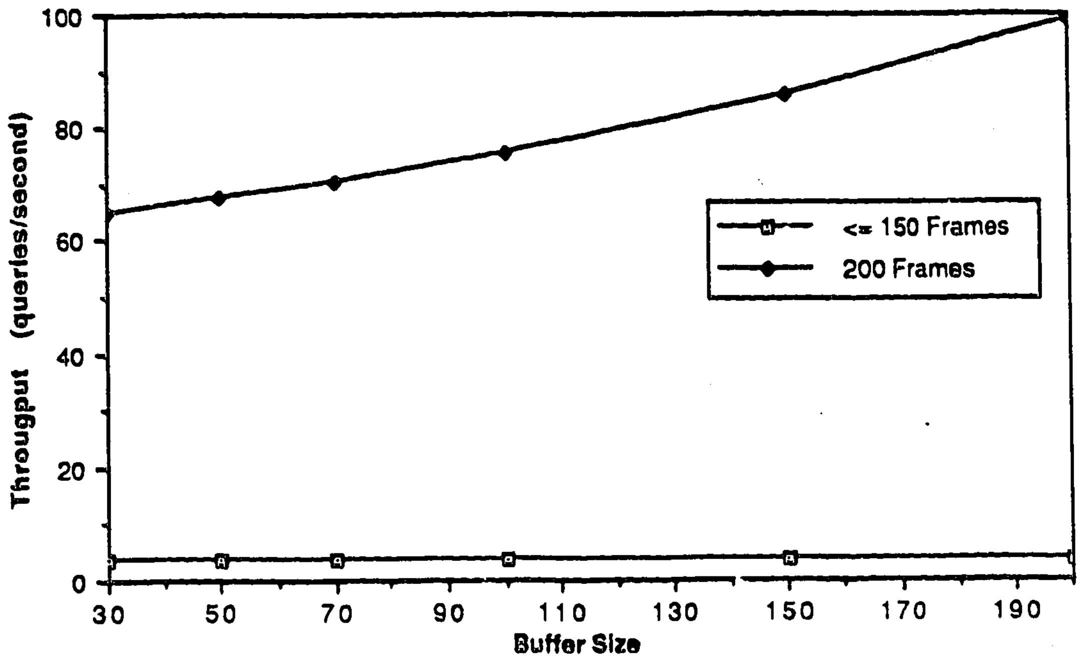

Figure 5.1: Throughput for Simple Selection Queries Without Indices

5.2.1 Simple Selection Without Indices

To execute this type of query, a sequential scan of all the data pages in the database is required for each query. Therefore, each query touches two hundred pages, the size of the database, and references 10,000 tuples. Since the pages are sharable and the sequence of data pages accessed by each query is similar in the order of 0, 1, 2, ..., 199, only the buffer size and the main memory size play the major role. There is not much difference in the results between the two buffer replacement policies because the optimal buffer size is one and the buffer replacement policy is LRU in both cases.

When the main memory size is 20, 30, 50, 100 and 150 frames, the throughputs (Figure 5.1) are similar for the buffer sizes of 20, 30, 50, 70, 100, 150 and 200. This is because at these main memory sizes, the 200 page database cannot be contained in the main memory. At buffer size less than 200, each buffer fault will induce a major page fault, invoking a double fault. The initial I/Os are invoked with the double faults. At buffer size of 200, with main memory size less than 200, most of the I/Os are induced by the page manager

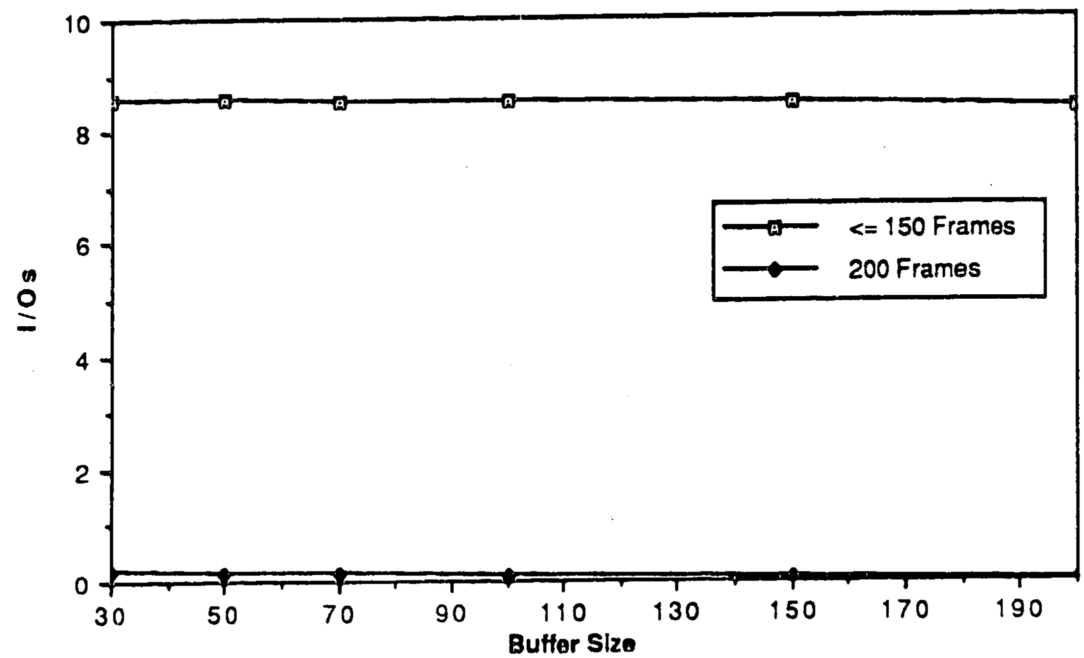

Figure 5.2: I/Os for Simple Selection Queries Without Indices

because all the data pages cannot be contained in the main memory. Because the pattern of the address reference is sequential, there are no fluctuations in the I/O curves (Figure 5.2).

The throughput curve at 200 frames is not straight because of the following. In our experiments, the start of measurement is determined by the buffer size. For example, if the buffer size is fifty, then we start measuring after the fiftieth I/O. This is to ensure that the buffers have reached steady state. Thus, in runs with buffer size less than 200, I/Os still exist because not all the data pages have been read into the buffers. This increases the total simulation time thus causing a corresponding decrease in the throughput. The response times of the queries are not affected because the I/Os are caused by processes which have been started before the start of the simulation time. These early processes would have paged in all the data pages and thus the latter processes would not inflict any I/Os. Besides the amount of time taken to process a successful address reference, the response time, as shown in Figure 5.3, is also affected by the allocated time quantum of the Round-Robin scheduling policy.

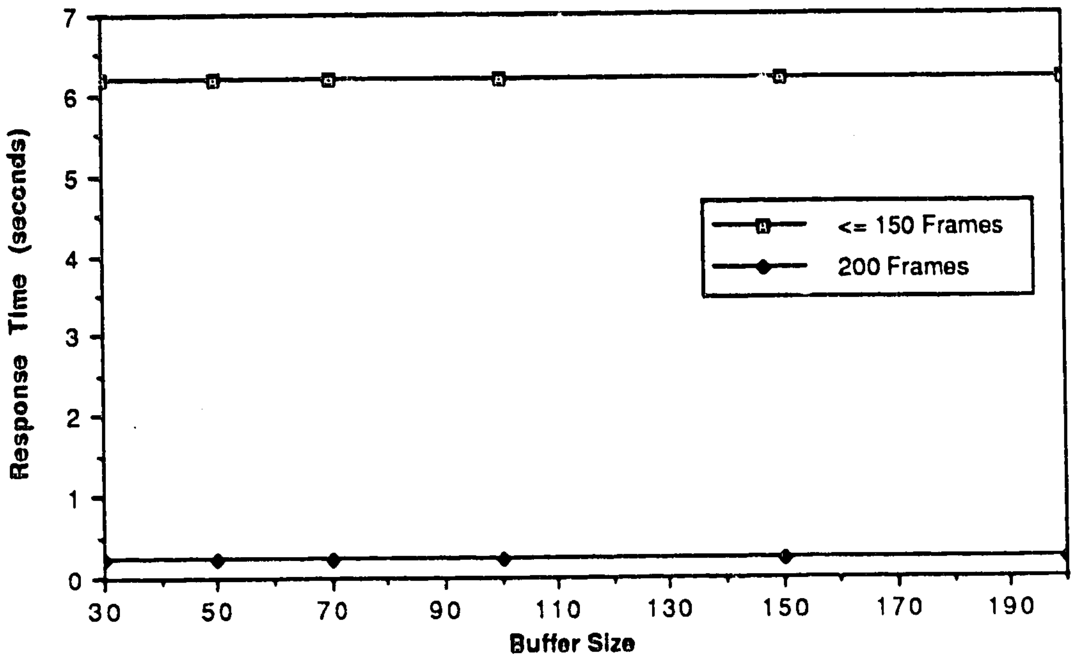

Figure 5.3: Response Time for Simple Selection Queries Without Indices

There is a dramatic difference (6 seconds) between the mean response time of a query being executed in a main memory size of 200 frames and less than 200 frames because of the occurrence of double faults and major page faults in the latter.

The page fault curves (Figure 5.4) of all the two buffer algorithms at different buffer sizes and main memory sizes are the same because page fault is the summation of minor and major page faults. In a minor page fault, only the process' page table needs to be updated whereas in a major page fault, an I/O is also required to bring the addressed data page into main memory. When a process addresses a data page that is in I/O, the process has a page fault and is blocked until the I/O for this data page has been completed. The page fault curves are similar because in both algorithms, each query has the same buffer replacement policy, that is LRU, and an optimal buffer size of one, and the page reference patterns of the queries are similar.

Similarly, for the buffers, when a process addresses a data page that is in a buffer which is locked for I/O, the process has a buffer fault, a page

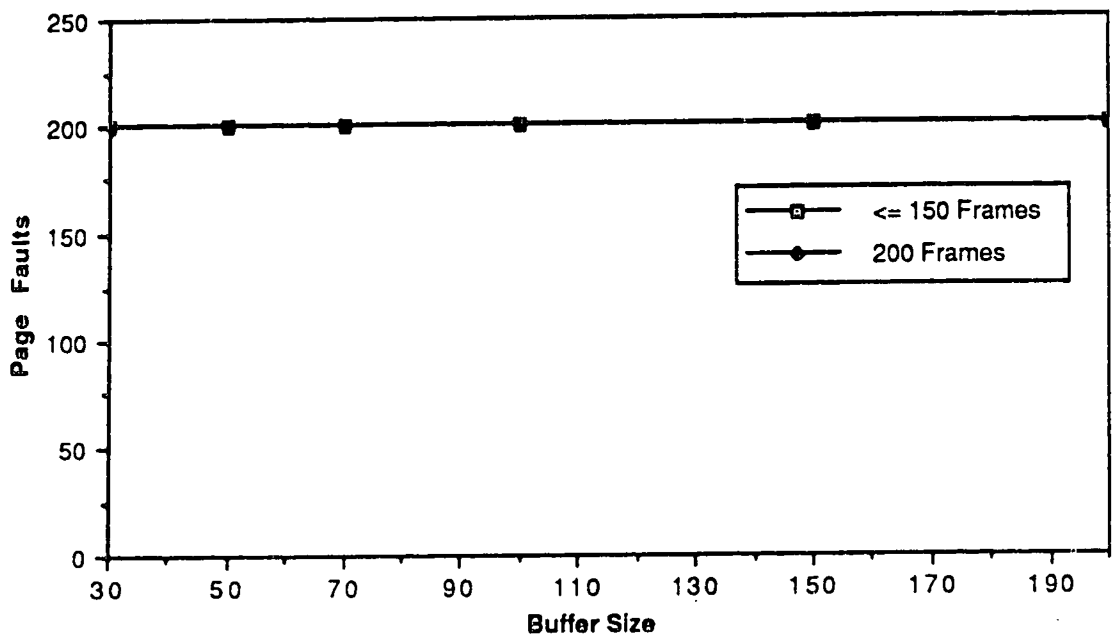

Figure 5.4: Page Faults for Simple Selection Queries Without Indices

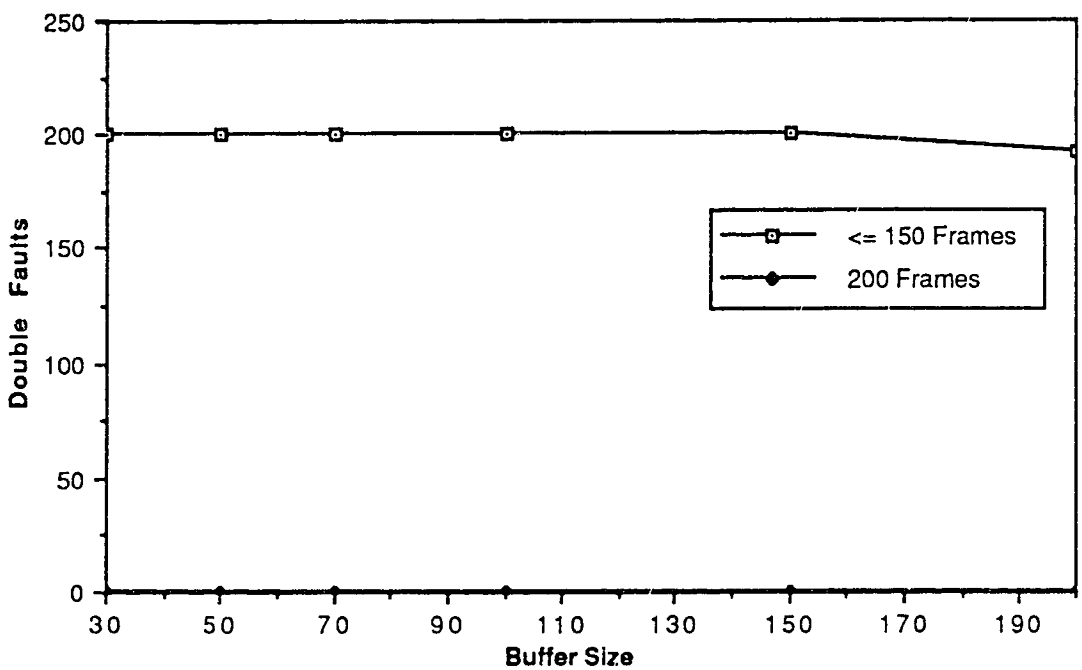

Figure 5.5: Double Faults for Simple Selection Queries Without Indices

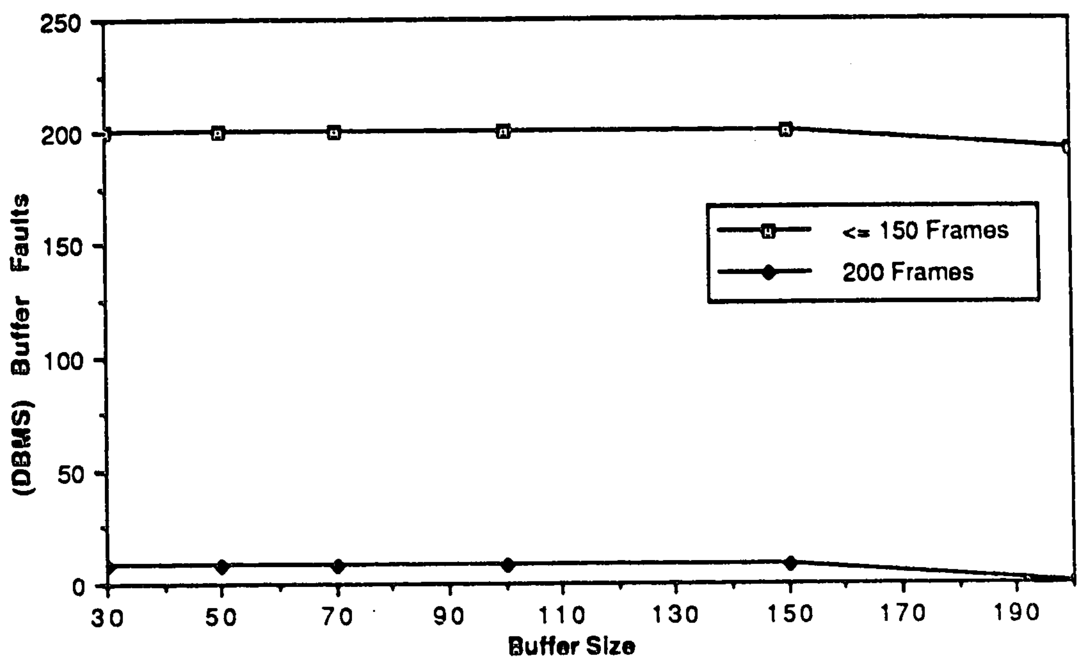

Figure 5.6: (DBMS) Buffer Faults for Simple Selection Queries Without Indices

fault and a double fault, and it is blocked for I/O too. There is a significant difference between the double fault curves (Figure 5.5) of 20, 30, 50, 100 and 150 frames and that of 200 frames because at a main memory size of 200, all the buffers are in the main memory, therefore no major page faults will take place, thus a zero double fault curve results.

At a main memory size of 200 and a buffer size of 20, 30, 50, 70, 100 and 150, though all the buffers are in the main memory, buffer faults will still occur because these buffer sizes are not large enough to contain all 200 data pages. On the other hand, the double fault curve at 200 frames is zero. This is because all the buffers are in the main memory and thus there will not be any major page faults. When the main memory size is less than 200 frames, the buffer fault curves (Figure 5.6) are very similar to the double fault curves. This is because i) both the buffer sizes and main memory sizes are not large enough to contain the whole database, ii) the replacement policies of both the page manager and buffer manager is LRU, and iii) the page reference patterns of all the queries are sequential and similar. Therefore, a buffer fault will cause

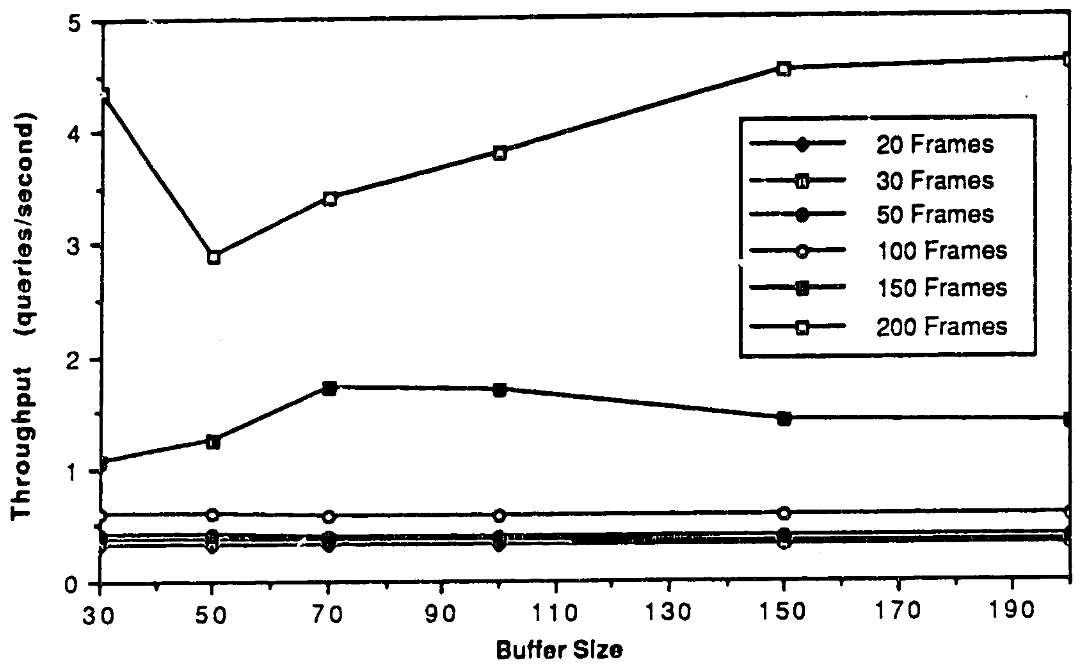

Figure 5.7: Throughput (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

a double fault too.

5.2.2 Simple Selection With Non-Clustered Indices

In these simulation runs, the data pages which contain the tuples and the pages which contain the indices to the database will be accessed. Therefore, the total number of pages that can be accessed is 976 which is the total number of DBMS pages. Thus, the 200 buffers/frames are not large enough to contain all the DBMS pages that can be accessed. This results in a higher number of I/Os, higher number of page faults, higher number of buffer faults and higher number of double faults, resulting in a much lower throughput and a higher response time.

The throughputs (Figures 5.7, 5.8) of these runs do not only increase with respect to the buffer size but also with respect to the main memory size. This is because not all the buffers and frames are shared at any moment. In the previous case, all the queries access all the data pages in the same order

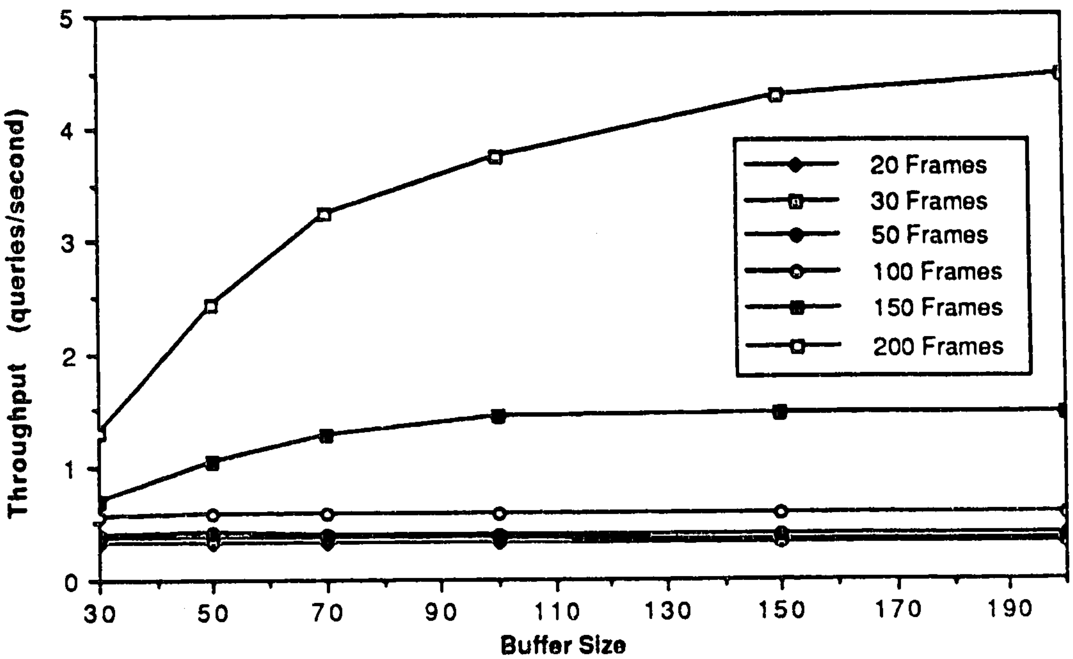

Figure 5.8: Throughput (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

so the probability of finding a data page in a buffer/frame is very high. But, in this case, the queries access the DBMS pages in a more random order and the range of the number of pages that can be accessed is higher, thus, the probability of finding a page in a buffer/frame is lower.

At a main memory size of 20, 30 and 50 frames, the throughputs are close to one another because the main memory sizes are not large enough to contain the local set of the queries. For the Hot Set model (see Figure 5.7), there is a dip at buffer size of 50 and main memory size of 200. This is due to the following. The average optimal buffer requirement of each process is 2. Since there are always 24 concurrent processes in the system, the total optimal buffer requirement is 48 which is very close to the actual buffer size. At buffer size of 30, on each buffer fault, a process will use a buffer in its local set for replacement. The page address references passed to the memory manager will be more local to the current running process. At buffer size of 50, at each buffer fault, a process replaces a buffer which belongs to another process most of the time. This is because, at each buffer fault, a buffer will be taken from the faulting process if the number of allocated buffers is greater than or equal to the optimal buffer size. This chosen buffer is then inserted into a free list which is managed by a LRU policy. The free list is seldom empty since the actual buffer size is greater than the total optimal buffer requirement. The buffer which is least recently used will be deleted from the list and given to this faulting process. Since the free list is seldom empty, the faulting process will be replacing a buffer which has been inserted by another process. This causes more faults, thus increasing the double faults and the number of I/Os (5.9, 5.10). As the buffer size increases, the probability of replacing a buffer which does not belong to any process increases because the length of the free list will increase too. The free buffers which are chosen for replacement mostly belong to processes which have left the system. Thus, the throughput increases. This phenomenon does not exist with the DBMIN buffer replacement algorithm because in the DBMIN case, the replacement algorithm is more localized. On a buffer fault, a buffer is chosen from the faulting locality set for replacement. A process can have many locality sets but a fault causes replacement only within its locality set. Therefore, faultings/replacements are under more control. For example, a process with a sequential scan using a non-clustered index path will have two locality sets; one for the index pages and the other for the database pages. A fault caused by a reference to an index page will replace the buffer containing an index page. A fault caused by a reference to a database page will replace the buffer containing a database page.

For the Hot Set model, at a main memory size of 150 frames, the throughput varies up and down at different buffer sizes. From a trace of the runs, it is found that at the peak, on a page fault, the probability of choosing a frame that similarly contains the DBMS page which has been chosen by the buffer manager for replacement on a buffer fault is very high. That is, there is a high probability that the buffer manager and the page manager choose the buffer/frame that contains the same DBMS page for replacement. This would imply that the page replacement policy is similar to that of a local replacement

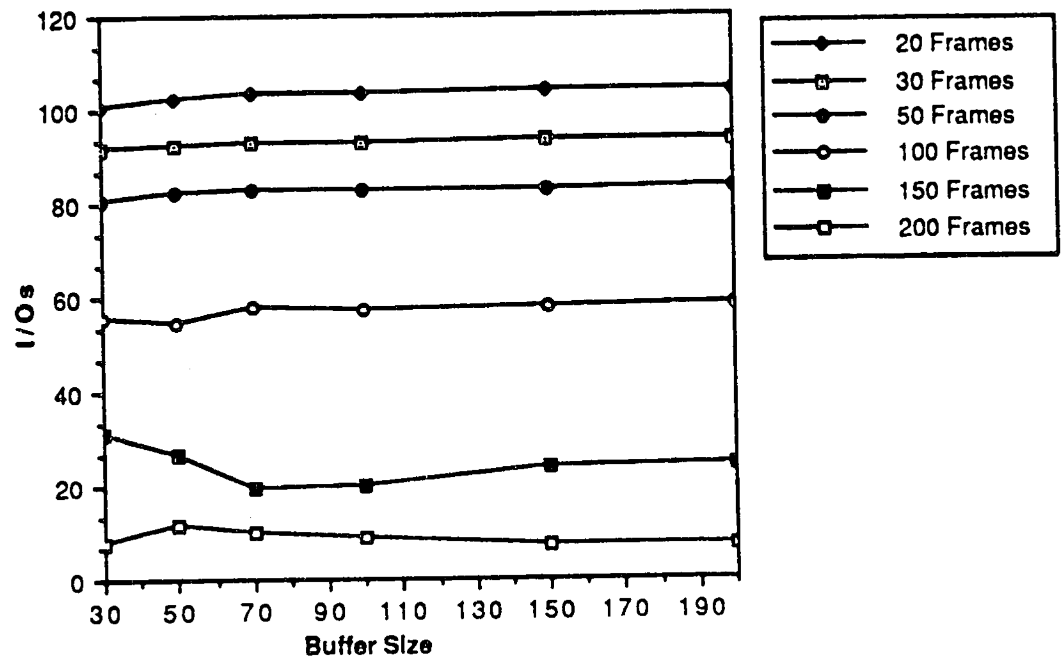

Figure 5.9: I/Os (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

policy. And, thus less frame stealings among the processes occur.

The larger the main memory size, the lower the response time (Figures 5.11, 5.12). This is due to a decrease in the number of major page faults as the probability of finding a DBMS page in main memory increases. The response time is quite stable for a given main memory size and various buffer sizes. This is because though the number of major buffer faults decreases as the buffer size increases, the number of major page faults increases since more virtual pages are mapped to a fixed number of frames.

At a main memory size of 20 and 30 frames and low buffer sizes, the response times are lower than at larger buffer sizes. This is because there are fewer ready/running processes to compete for resources since the lower number of buffers/frames limits the number of ready/running processes. When a process has a buffer fault/page fault and no buffer/frame is available for replacement, the process is suspended. With fewer processes competing for resources, each running process is able to finish in lower amount of time. At

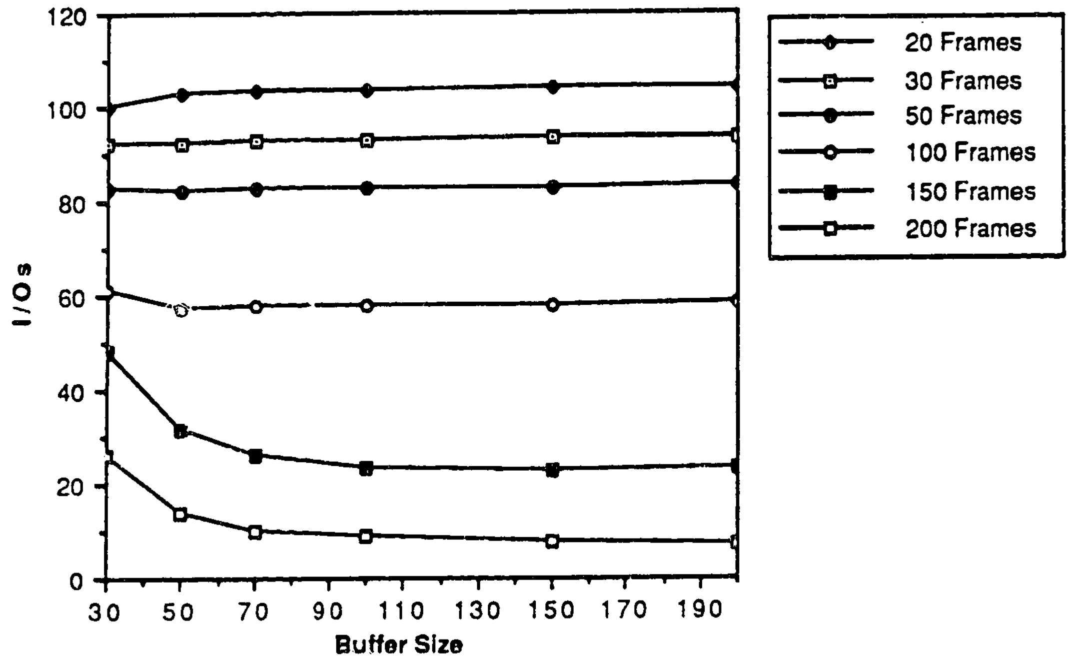

Figure 5.10: I/Os (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

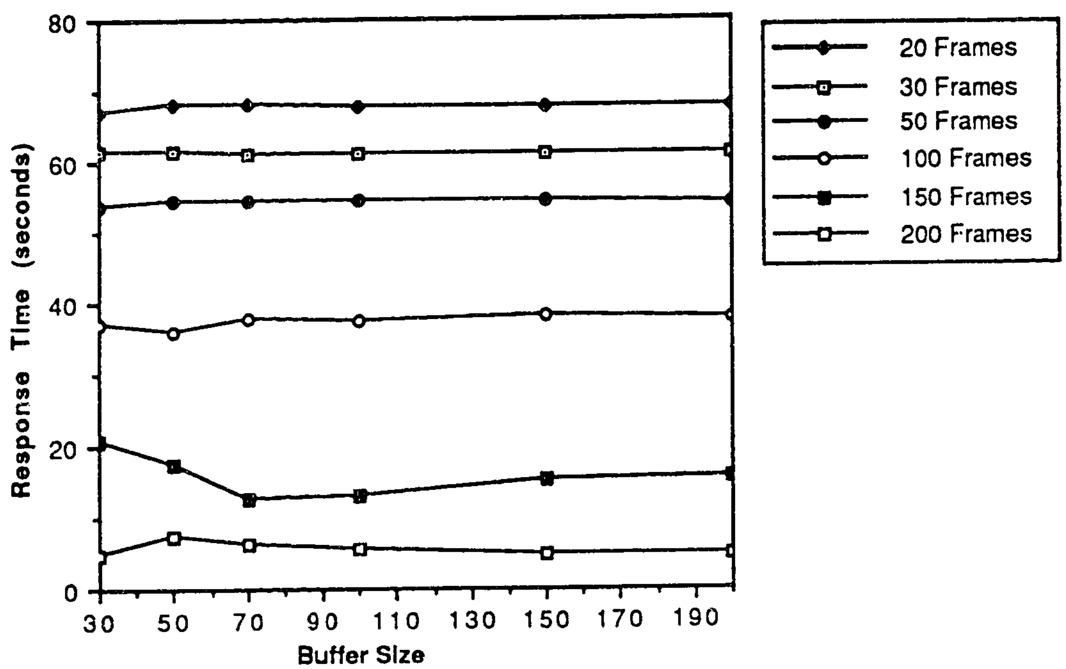

Figure 5.11: Response Time (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

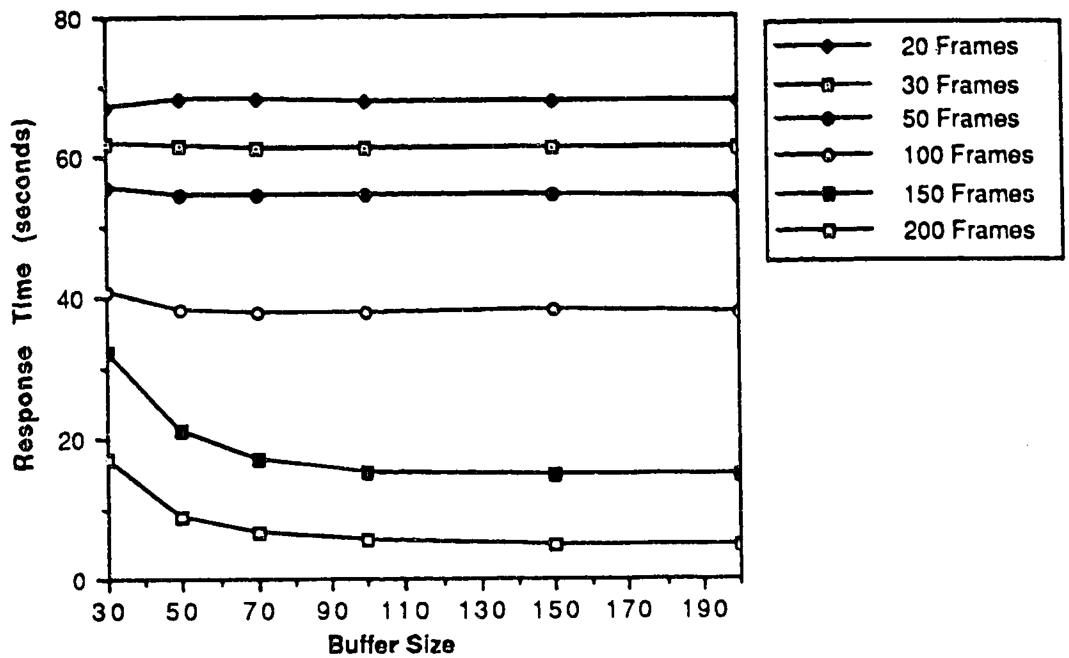

Figure 5.12: Response Time (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

larger main memory sizes (50, 100, 150 and 200), the probability of capturing the locality of the queries increases. Because the start of the measurement is determined by the buffer size, at low buffer sizes and high main memory sizes, fewer commonly used DBMS pages are in the main memory when the measurement is initiated. Thus, the response times at the lower buffer sizes are higher since I/Os are required to bring in the more commonly used DBMS pages. This phenomenon is especially evident in the DBMIN model. Though we gave the same explanation for the increment in the throughput of the DBMIN model at a main memory size of 200 in the previous query type, the response time in that case is constant for all buffer sizes. The response time is decremental in this case because the referenc patterns of the queries are random and the range of DBMS pages touched by the queries are wider in this case.

Notice that for all the algorithms, the double fau curves (Figures 5.13, 5.14) decrease significantly from the point where buffer size is equal to the main memory size. This is because more DBMS pages are in the buffers with a buffer

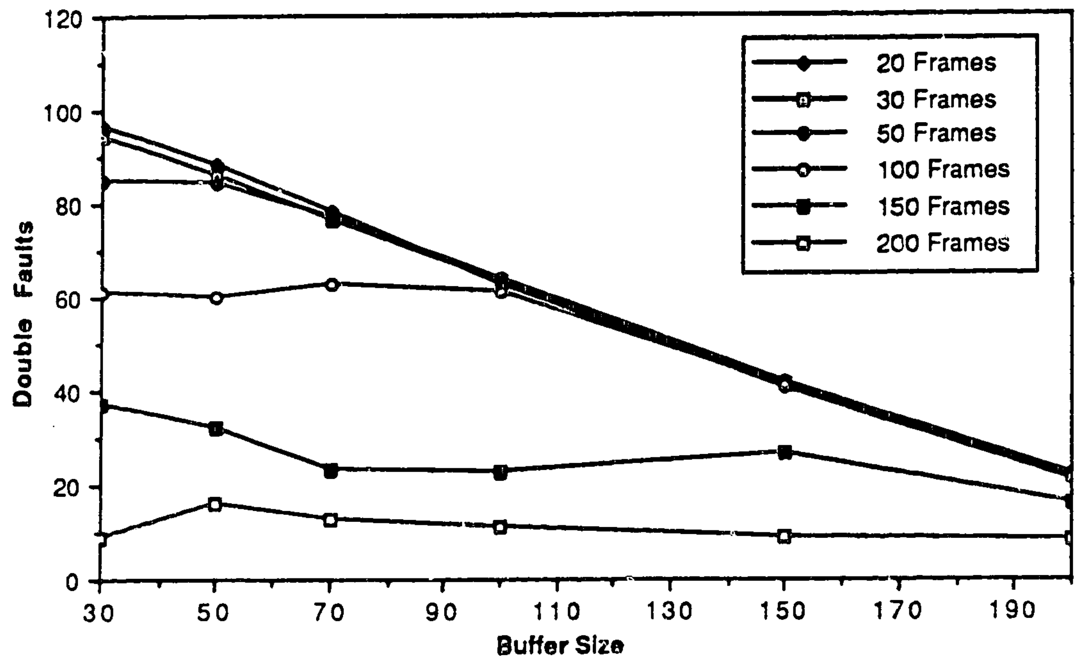

Figure 5.13: Double Faults (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

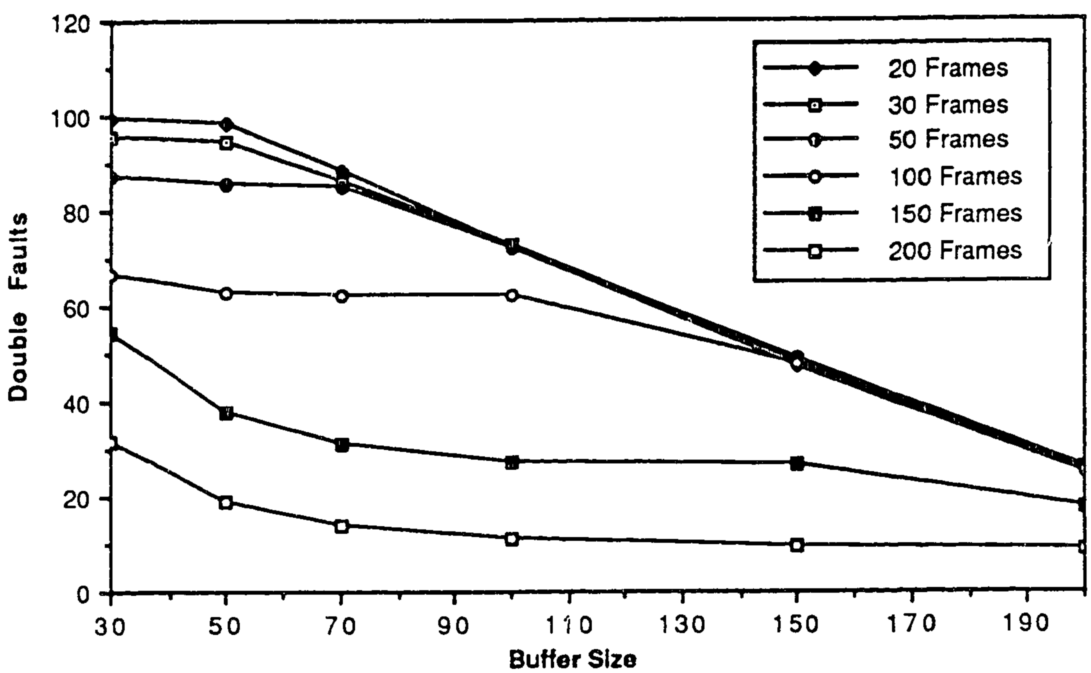

Figure 5.14: Double Faults (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

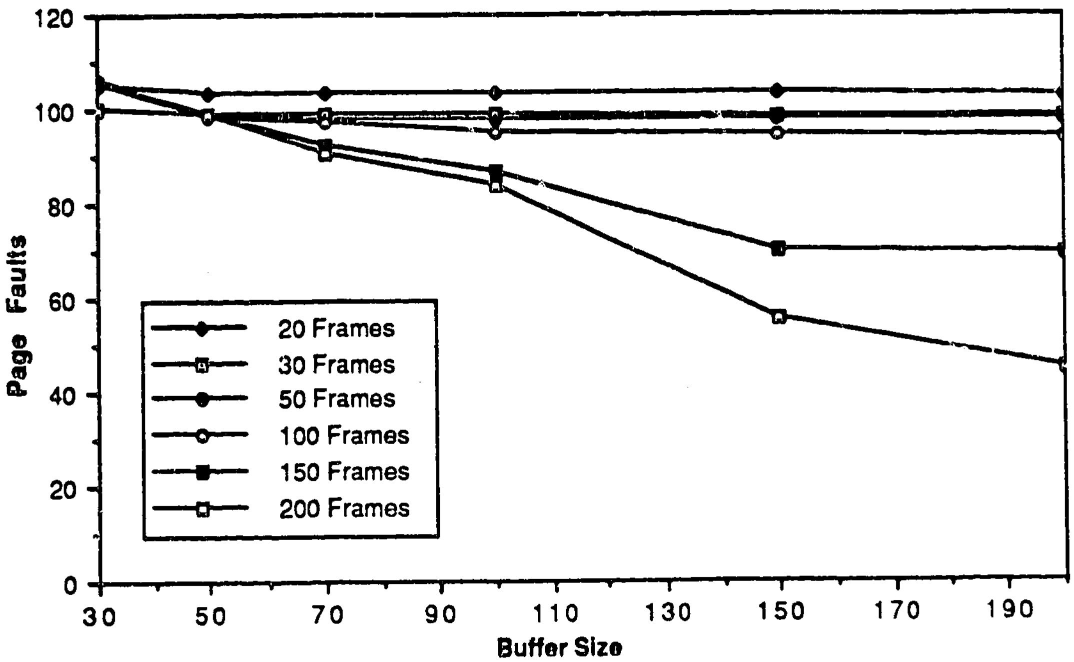

Figure 5.15: Page Faults (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

size greater than the main memory size. Therefore, the probability of having to choose a buffer for replacement is lower. A double fault can only happen if a buffer fault has occurred. Since the number of buffer fault decreases as the buffer size increases (see Figures 5.17, 5.18), the number of double faults also decreases.

The page fault curves (Figures 5.15, 5.16) of 150 and 200 frames are decreasing because the components of the page faults consist of the double faults, the major and minor page faults. Though the major page fault component is increasing as the buffer size increases, the double fault component and the minor page fault components are decreasing. The double fault component decreases due to the reason given above. The minor page fault component decreases because, as the buffer size increases, the page table size also increases. This is because at a main memory size of 150 and 200, more data pages are in the main memory. Besides the page fault required to initialize the page table, less page faults occur because the probability of finding a DBMS page in the main memory is higher at higher main memory sizes and therefore it is less

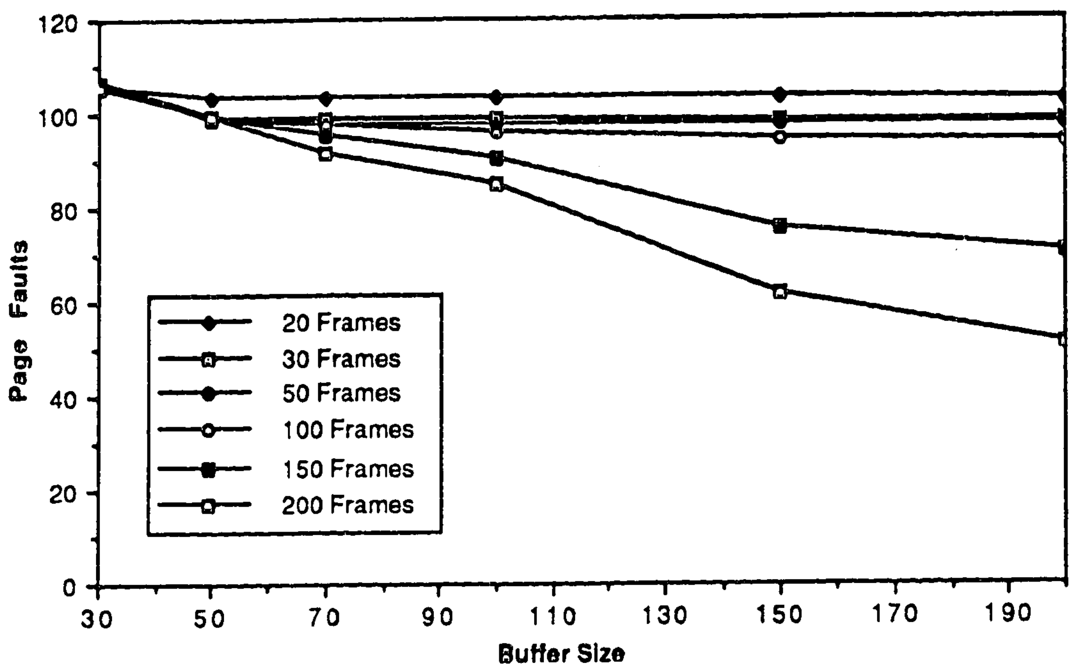

Figure 5.16: Page Faults (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

likely to invoke the page replacement policy.

5.2.3 Summary

In these runs, the performance results produced by the queries in the DBMIN model are more consistent and predictable than for the Hot Set model. This is because in the DBMIN model, there is more control over the selection of buffers for replacement. The DBMIN model is able to separate the replacement of data pages from replacement of index pages and the Hot Set model does not differentiate between the two types of pages. In addition, the locality of the queries is easy to capture in these runs.

5.3 Merge Scan Joins

In this section, we shall discuss the results on the runs obtained from the queries executed with the merge scan join. We have only done the simulation

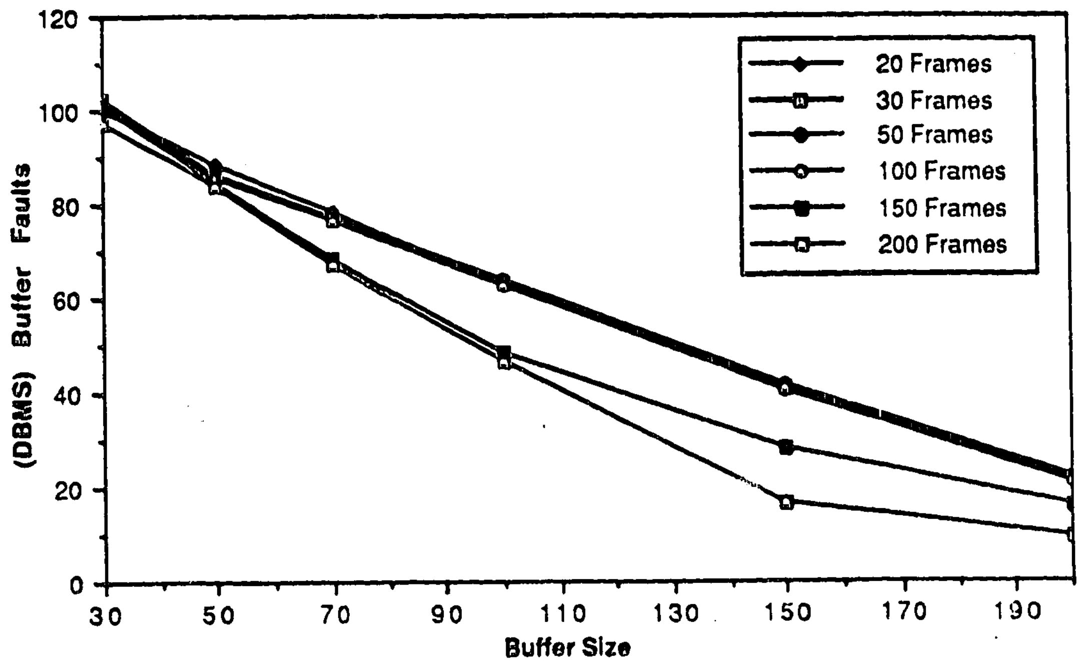

Figure 5.17: (DBMS) Buffer Faults (Hot Set model) for Simple Selection Queries With Non-Clustered Indices

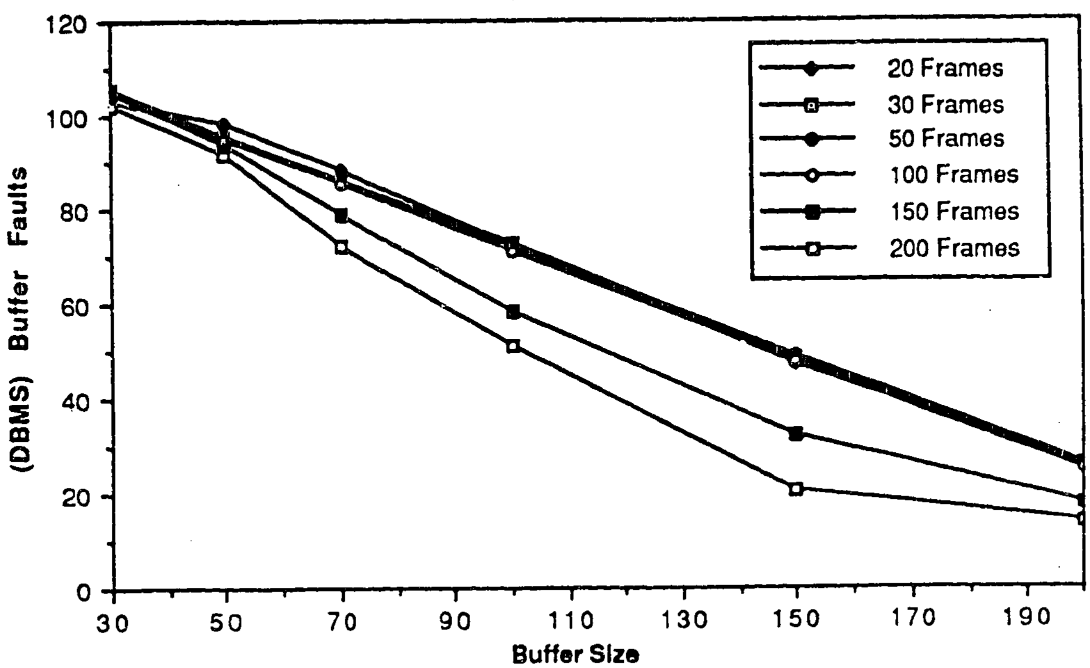

Figure 5.18: (DBMS) Buffer Faults (DBMIN model) for Simple Selection Queries With Non-Clustered Indices

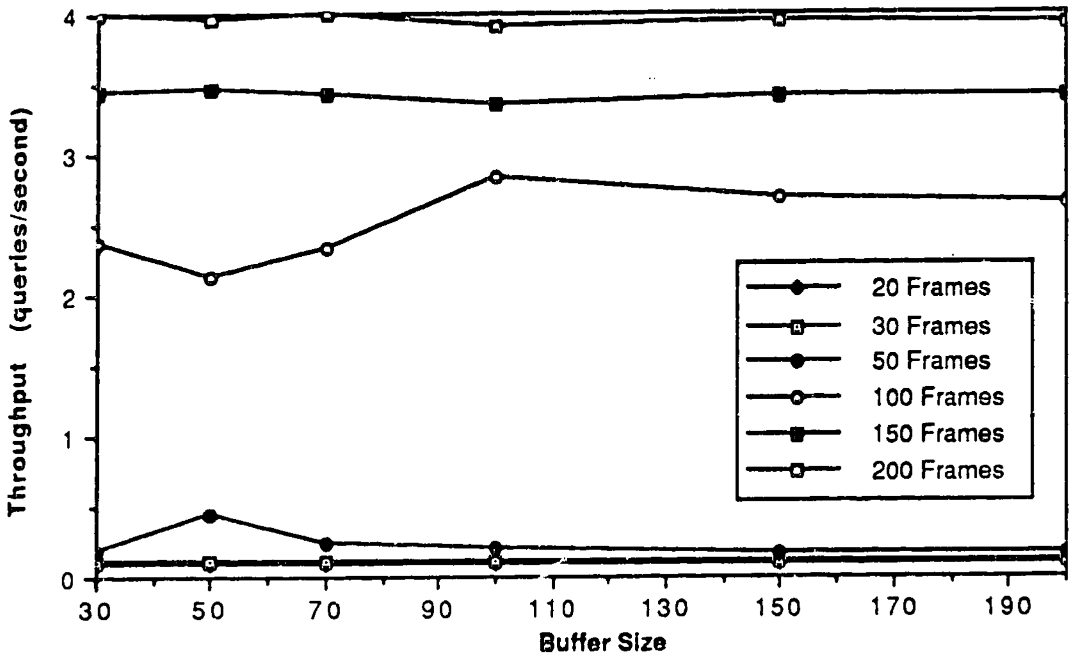

Figure 5.19: Throughput (Hot Set model) for C-C Merge Scan Join Queries

on paths that use clustered indices on both the outer and inner relations. This is because the formula used to calculate the optimal buffer size in both local buffer replacement algorithms are similar for both clustered and non-clustered indices.

In the merge scan method, temporary pages are used. In our implementation, temporary pages are not shared and each process has its own set of temporary pages. The number of temporary pages owned by each process is dependent on the selectivity factor of the process. Once a temporary page is assigned to a process, it can only be used by that process.

As in former simulation runs, the throughput (Figures 5.19, 5.20) increases with increasing main memory size. In cases where the main memory size is 100, 150 and 200, the throughput also increases with increasing buffer size. At lower main memory size, the throughput decreases with increasing buffer size. For example at main memory size of 30 frames in the Hot Set model and main memory size of 50 in the DBMIN model. At lower main memory size, the locality of the queries cannot be captured. When the buffer size

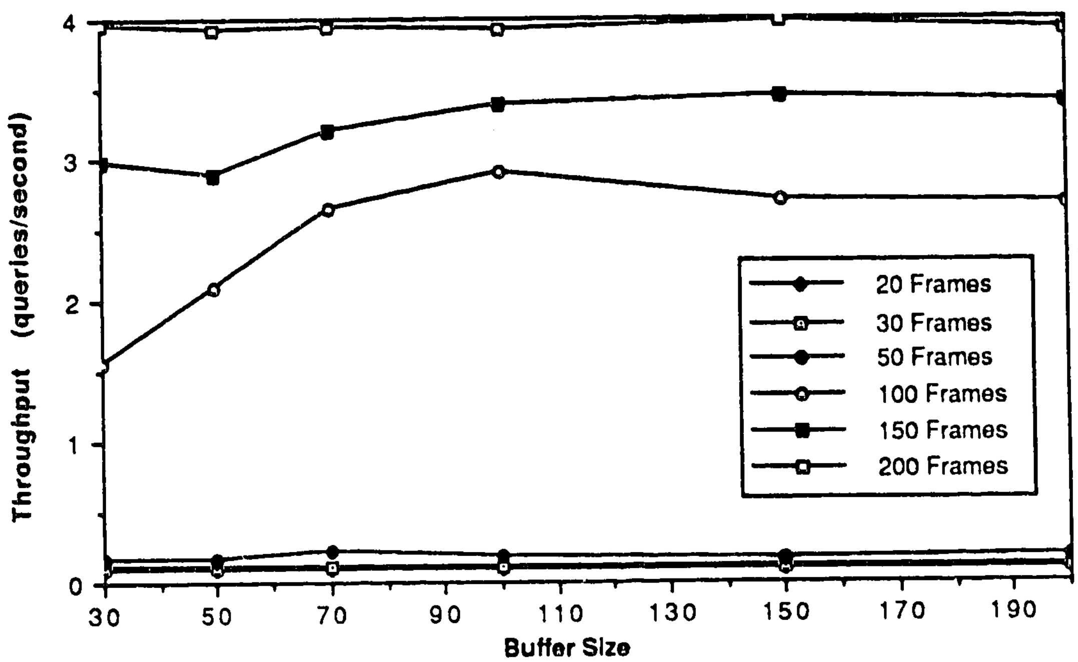

Figure 5.20: Throughput (DBMIN model) for C-C Merge Scan Join Queries

is small, the number of processes that are ready to run is limited and most processes will be in the suspended queue. This is because at lower buffer sizes, although there may be frames available to run a ready process, there might not be buffers to allocate to a ready process. If no buffer can be allocated to a ready process, the ready process is suspended. This also explains the slopes of the buffer fault curves (Figures 5.21, 5.22), the double fault curves (Figures 5.23, 5.24), and the page fault curves (Figures 5.25, 5.26).

With fewer ready processes, the range of pages that is referenced at any moment is also smaller. The probability of capturing the locality therefore is higher. As the buffer size increases, the number of processes that are ready to run becomes higher. Therefore, the range of pages that can be referenced becomes higher and the probability of capturing the locality decreases. This causes more major page faults and thus the throughput decreases. At larger main memory sizes, more commonly used DBMS pages are captured in the main memory. As the buffer size increases, the throughput becomes more stable and does not increase because I/Os are required to fetch the temporary

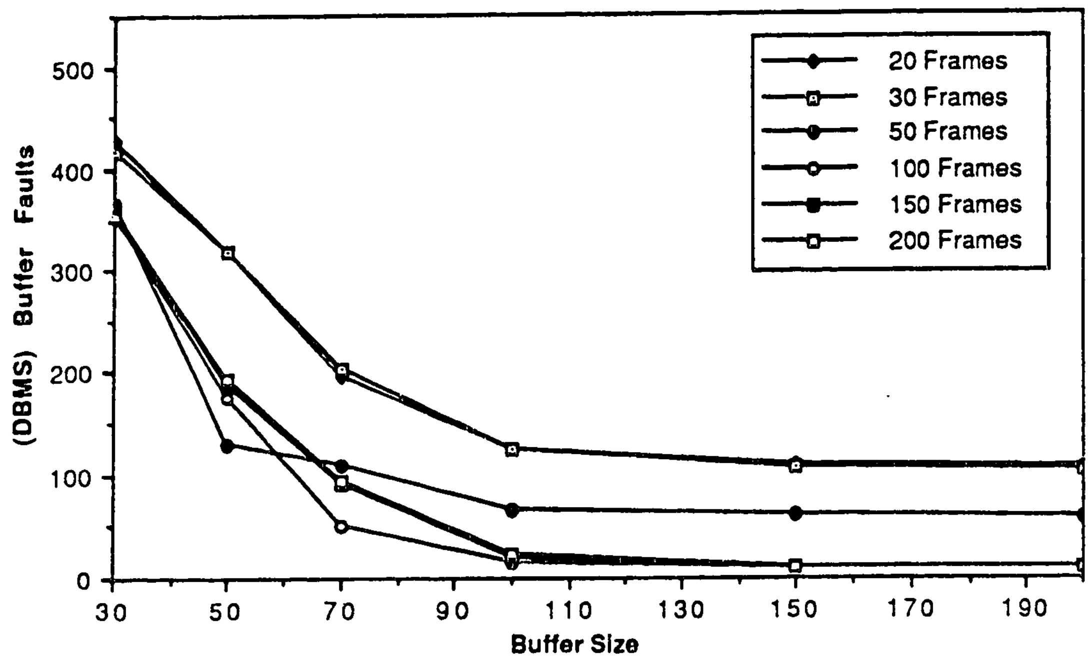

Figure 5.21: (DBMS) Buffer Faults (Hot Set model) for C-C Merge Scan Join Queries

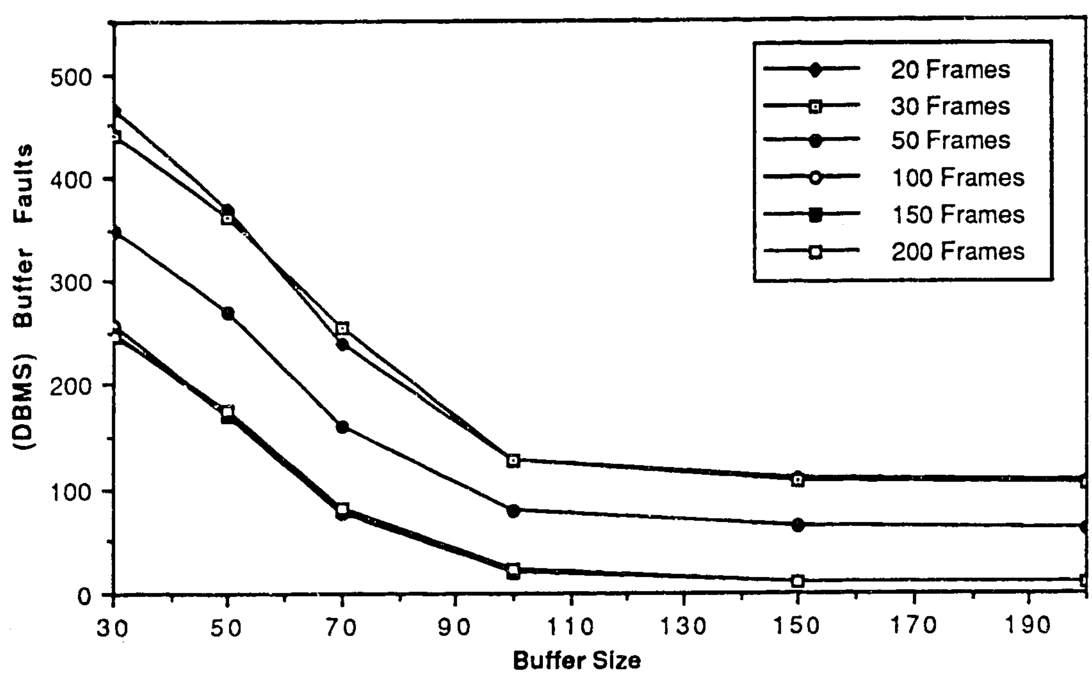

Figure 5.22: (DBMS) Buffer Faults (DBMIN model) for C-C Merge Scan Join Queries

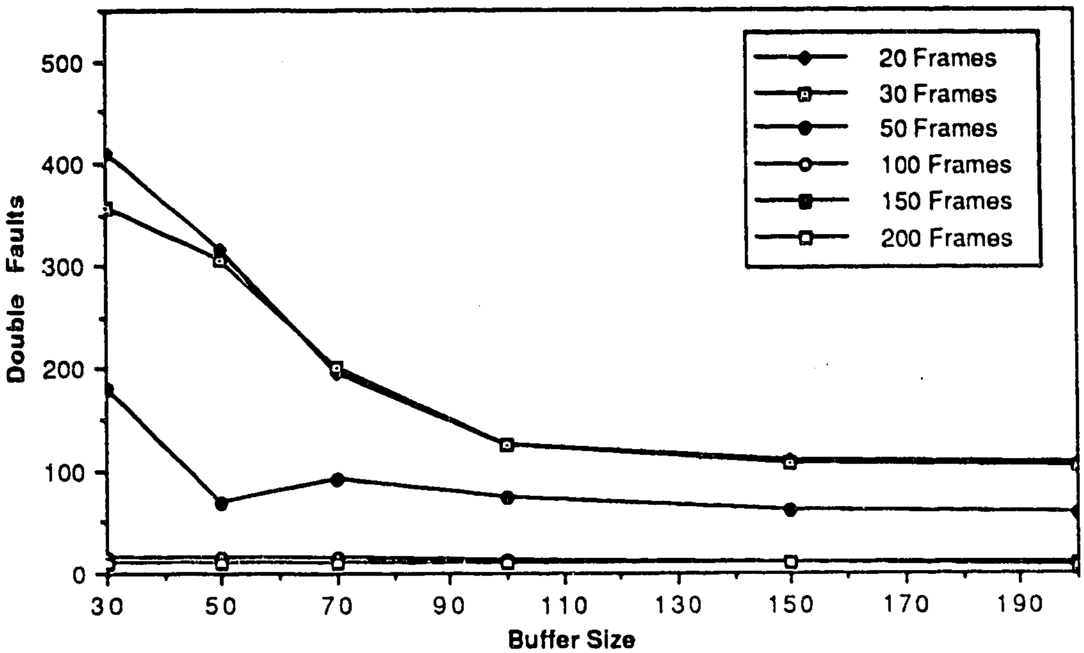

Figure 5.23: Double Faults (Hot Set model) for C-C Merge Scan Join Queries

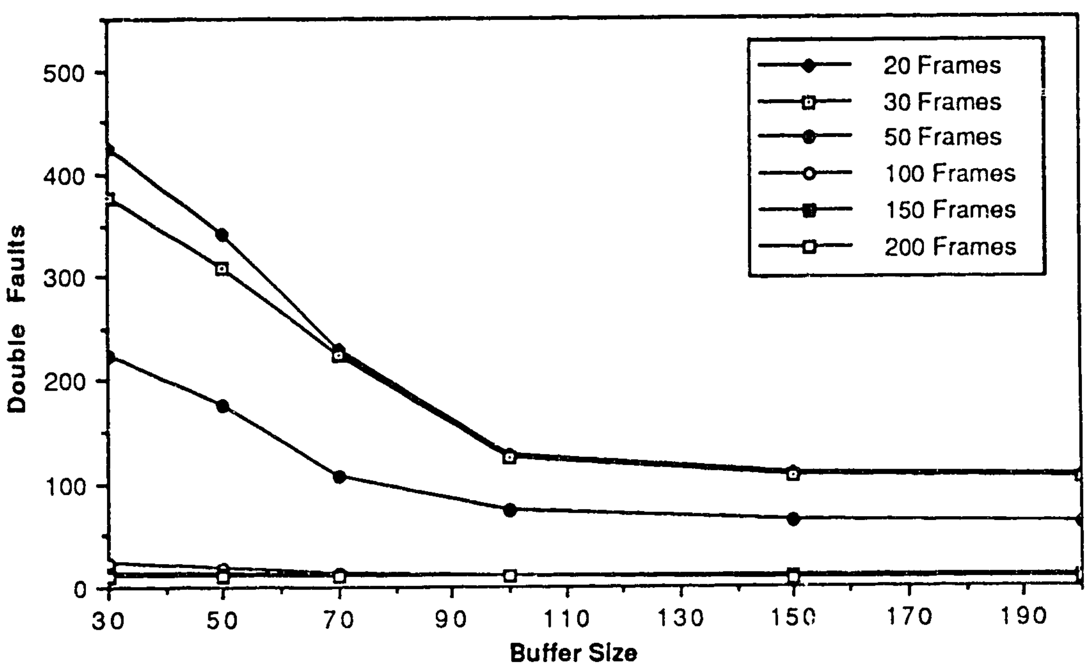

Figure 5.24: Double Faults (DBMIN model) for C-C Merge Scan Join Queries

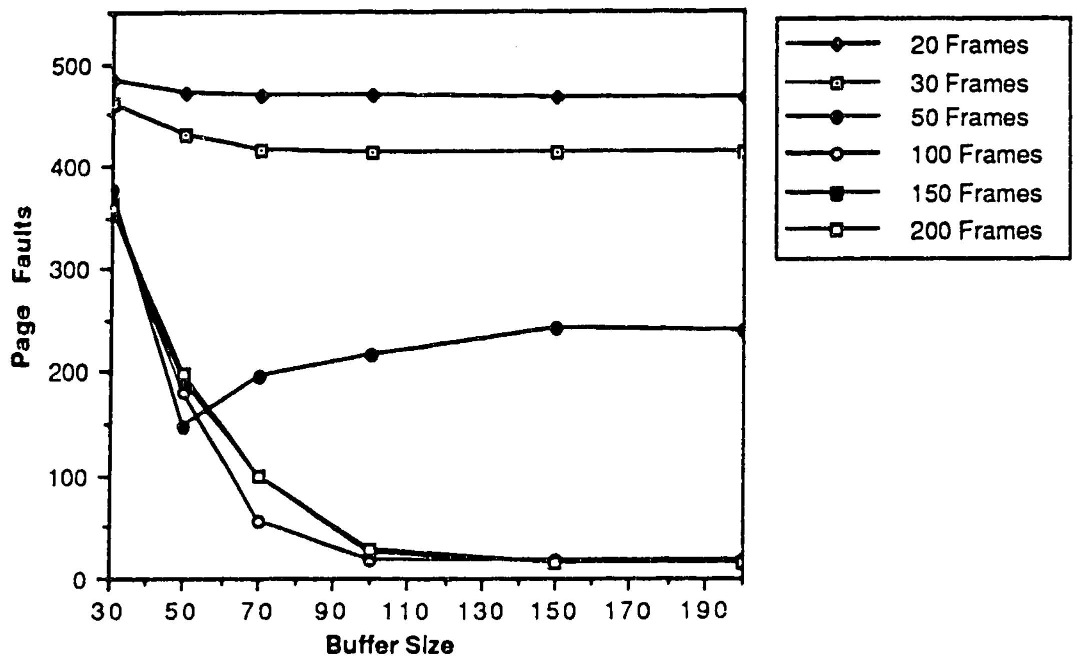

Figure 5.25: Page Faults (Hot Set model) for C-C Merge Scan Join Queries

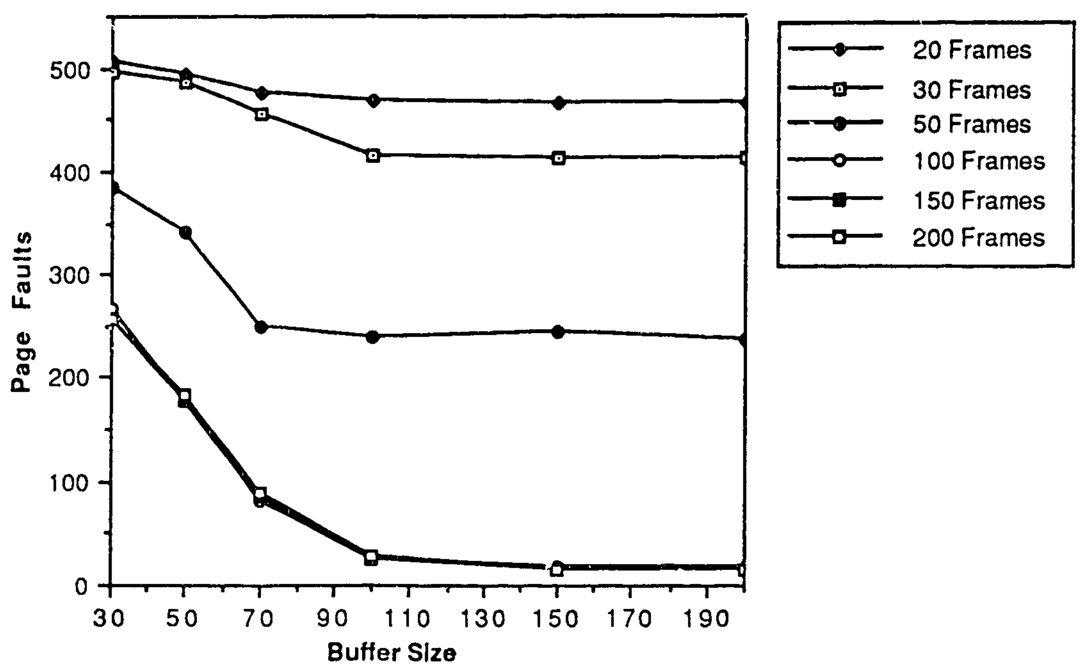

Figure 5.26: Page Faults (DBMIN model) for C-C Merge Scan Join Queries

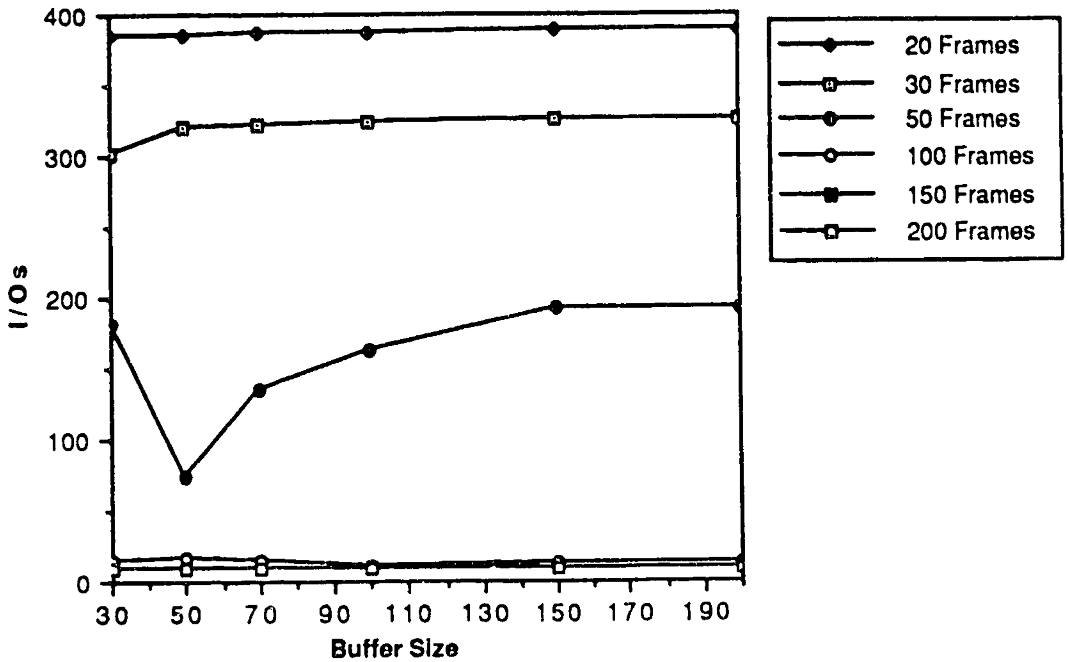

Figure 5.27: I/Os (Hot Set model) for C-C Merge Scan Join Queries

pages which are unique to every process. Thus, the I/Os curves (Figures 5.27, 5.28) and the response time curves (Figures 5.29, 5.30) are similarly stable.

At a main memory size of 50 frames, the throughput curves of both the Hot Set and DBMIN models are decreasing, but the DBMIN curve does not have any fluctuation whereas there is in the Hot Set model. At a buffer size of 50 in the Hot Set curve, there is a significant jump. From the trace of a simulation run with 50 buffers and 50 frames, we discovered that in this situation, in most cases, the frames that are chosen for replacement coincide with the buffers that are chosen for replacement. That is, with high frequency, when there is a buffer fault, the replaced buffer matches the frame that is chosen for displacement (if the same page reference causes a major page fault), with respect to their contents. That is, the replaced buffer and replaced frame have the same DBMS or temporary page number. Thus, the page replacement algorithm behaves almost like one with a local LRU replacement policy. Therefore, the pages chosen for replacement can be said to be more localized. Since the fluctuation is only reflected in the page fault curve but not in the other fault curves, we can

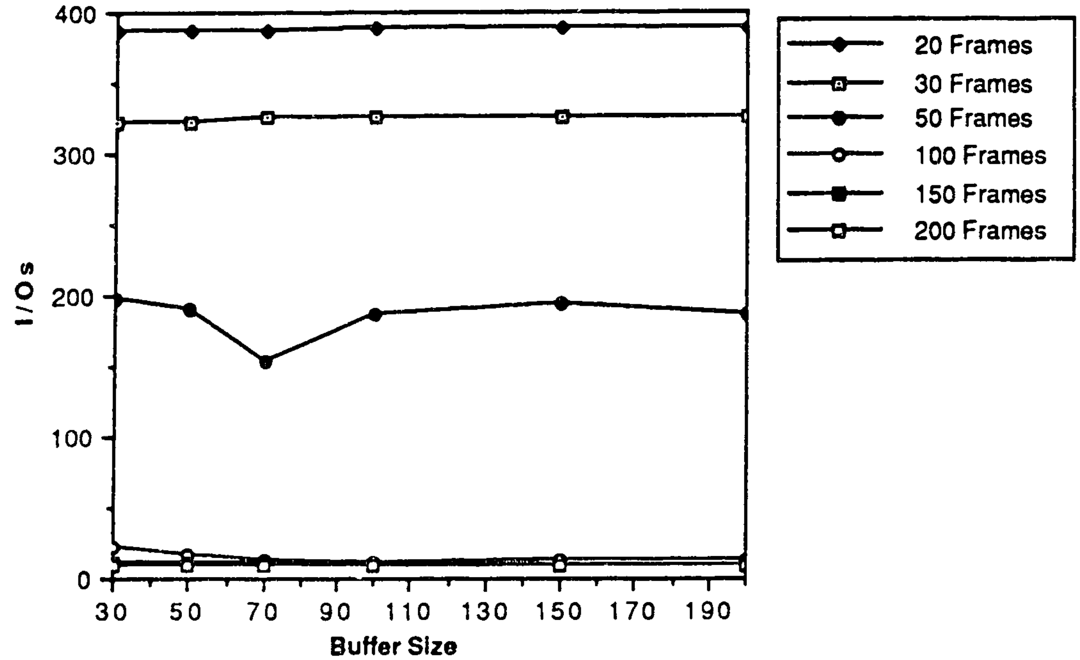

Figure 5.28: I/Os (DBMIN model) for C-C Merge Scan Join Queries

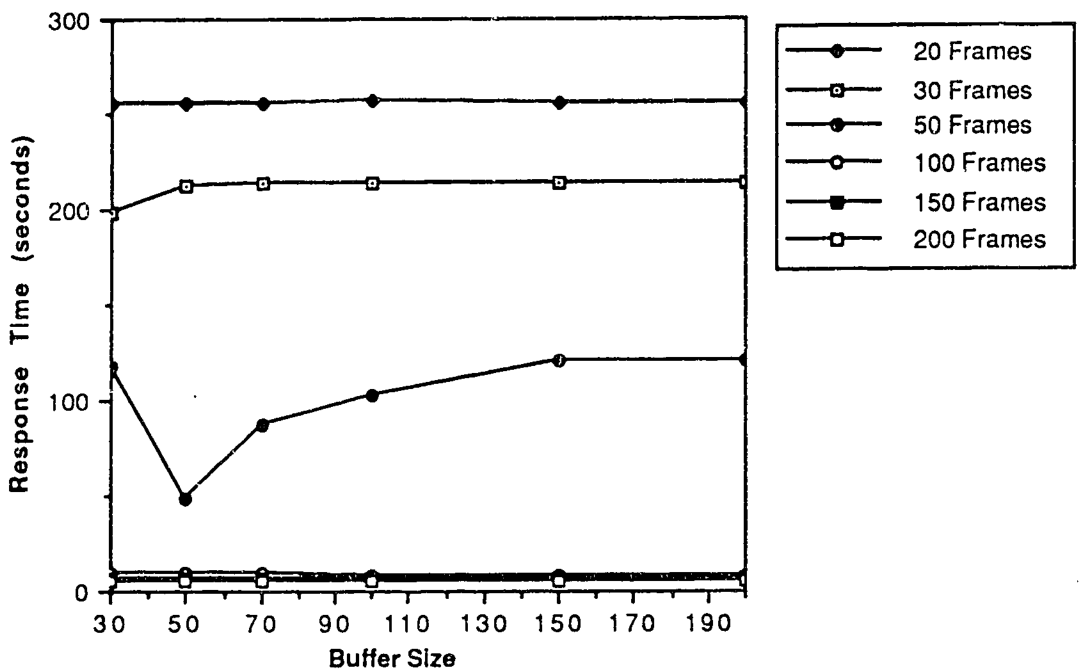

Figure 5.29: Response Time (Hot Set model) for C-C Merge Scan Join Queries

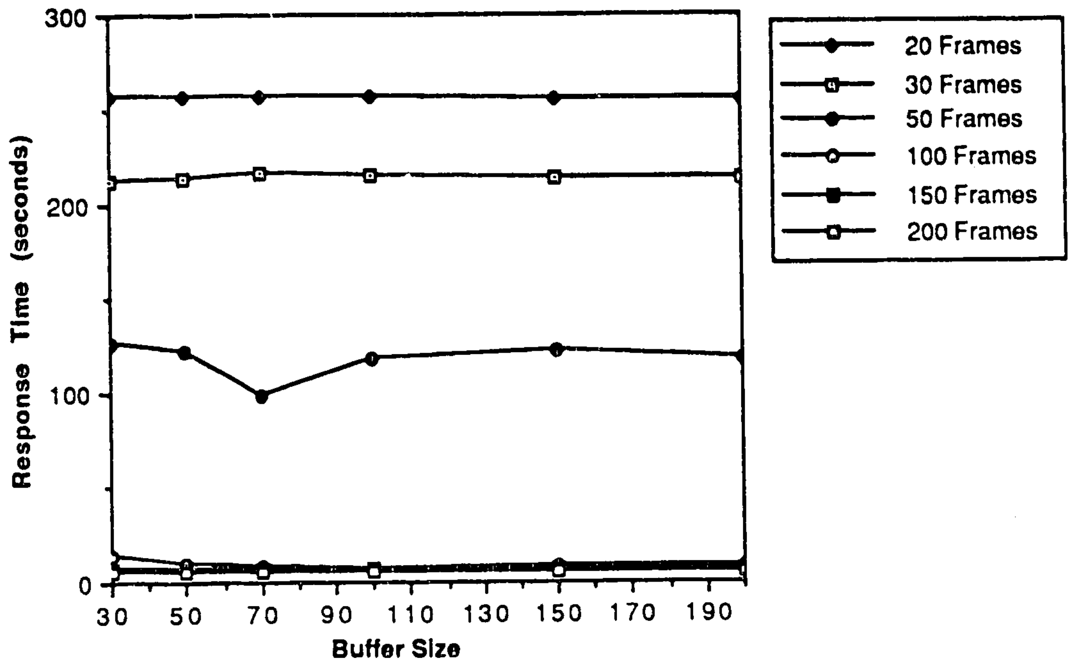

Figure 5.30: Response Time (DBMIN model) for C-C Merge Scan Join Queries

say that the page manager has caused the behavior. Since the page replacement policy is close to that of a local replacement policy, we do not have stealing of frames from other processes, therefore we have a lower number of I/Os.

Note that at a main memory size of 30 and a buffer size of 30, the number of buffer faults and number of double faults in both local buffer replacement algorithms are higher than that at main memory size of 20 and buffer size of 30. This is because with a higher main memory size, more processes are ready to run but since the buffer pool size remains the same, more buffer faults are generated due to more processes competing for a limited resource. The number of page faults at a main memory size of 30 and a buffer size of 30 in the DBMIN model is lower than that at a main memory size of 20 and a buffer size of 30 whereas in the Hot Set model, the number of page faults is higher. This is due to the fact that in the DBMIN model, the buffer allocation method is more local. When a buffer fault occurs, a replaced buffer can only be chosen from the faulting locality set. If such a buffer cannot be found, the buffer manager will try to get a buffer from another process. Since the buffer size is only 30, the probability of getting an available buffer is low. Therefore, the process is likely to be suspended. This is further demonstrated by the fact that the number of buffer faults at this point in the DBMIN model is higher than that of the Hot Set model at the same point. In the Hot Set model, a faulting process replaces a buffer from its single locality set. Therefore, the probability of suspension is lower. However, this increases the probability of causing a page fault.

5.3.1 Summary

In these runs, the throughput increases as the main memory size increases in both models. Also, the curves approach constant values at higher buffer sizes (a buffer size greater than or equal to 100). The number of buffer faults and number of pages faults in the DBMIN model are lower at a buffer size of 30 in most main memory sizes as compared to those of the Hot Set model at similar points. The throughputs at these points are also higher than that of the Hot Set model. However, the response times of both models are quite similar at these points. This phenomenon may be due to the fact that at lower buffer sizes, the DBMIN buffer manager has more control over the execution sequence of the processes.

5.4 Nested Loop Joins

In this section, we discuss the results obtained from the simulation runs which execute joins with the nested loop method. In a nested loop join, indices are required to access both the outer and inner relations to obtain reasonable performance. We have done two types of studies for the nested loop join: one with clustered indices on both the outer and inner relations and the other with non-clustered indices on the outer relation and clustered indices on the inner relation.

Since temporary pages are not used in the nested loop method, the

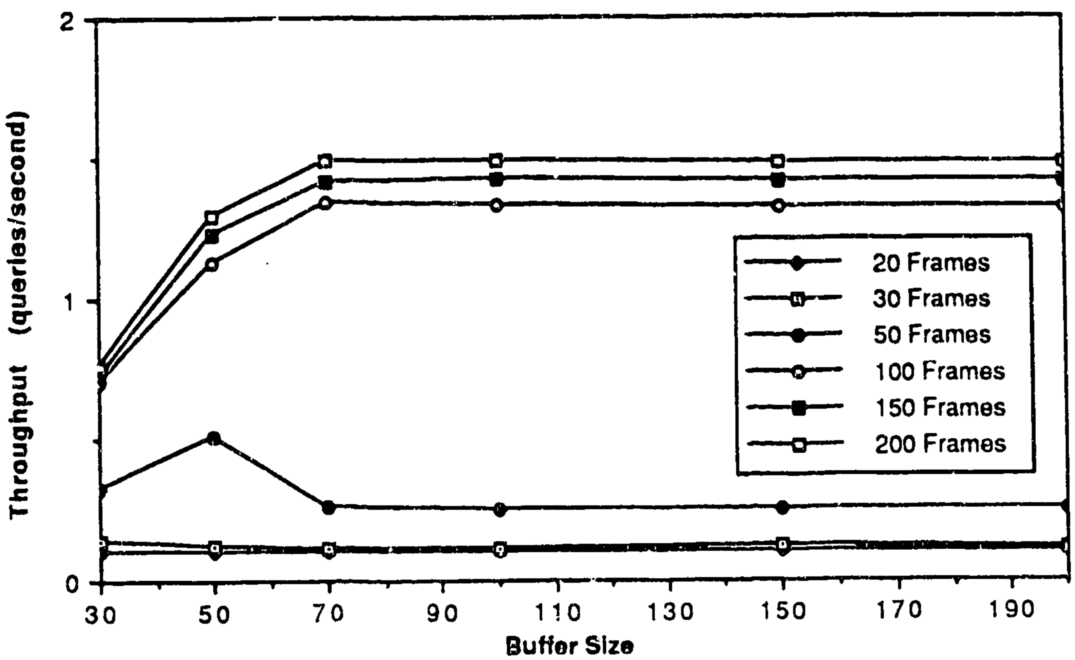

Figure 5.31: Throughput (Hot Set model) for C-C Nested Loop Join Queries

range of pages that is touched is smaller than that in the merge scan method; the throughputs of these simulation runs are therefore higher than those which executed the joins with the merge scan method.

5.4.1 Clustered Outer and Inner Indices

The throughput curves (Figures[3] 5.31, 5.32) for both algorithms are very similar to the throughput curves of the merge scan join, with respect to the algorithms. This might be due to the fact that in both join methods, clustered indices are used to access both relations. Since the merge scan join method accesses temporary pages while the nested loop join method does not, the throughputs of the latter are higher.

For both the Hot Set model and the DBMIN model, the throughput at a main memory size of 100 frames slightly decreases as the buffer size exceeds 100. From traces on the outputs from both models, it is discovered that the

[3]The abbreviation C-C in the figures means that clustered indices are used in both outer and inner relations.

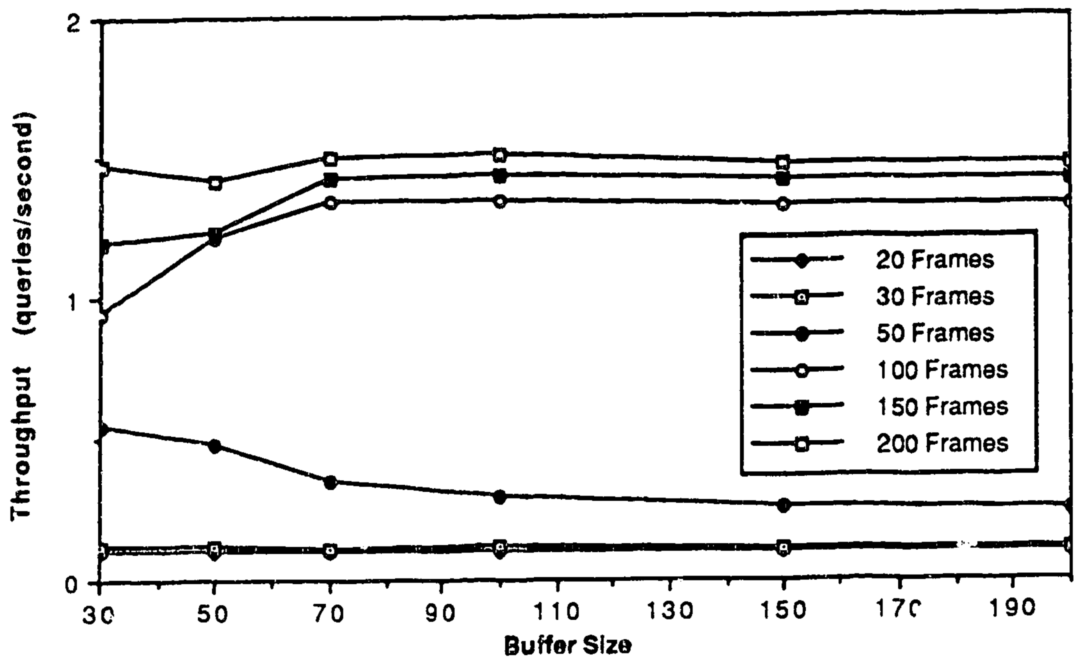

Figure 5.32: Throughput (DBMIN model) for C-C Nested Loop Join Queries

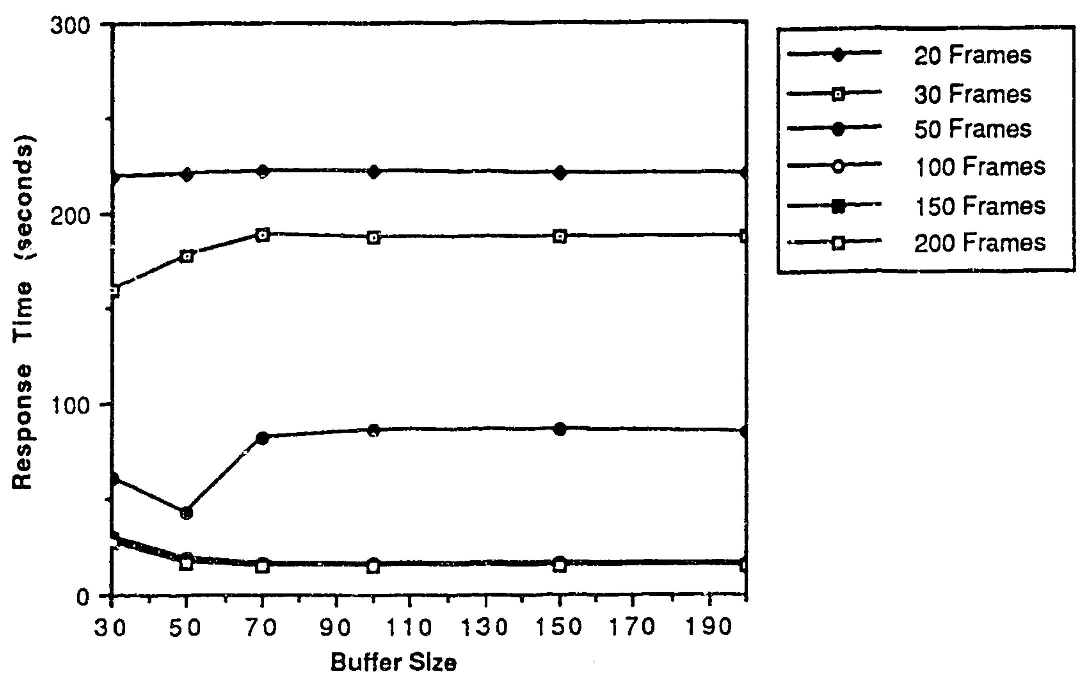

Figure 5.33: Response Time (Hot Set model) for C-C Nested Loop Join Queries

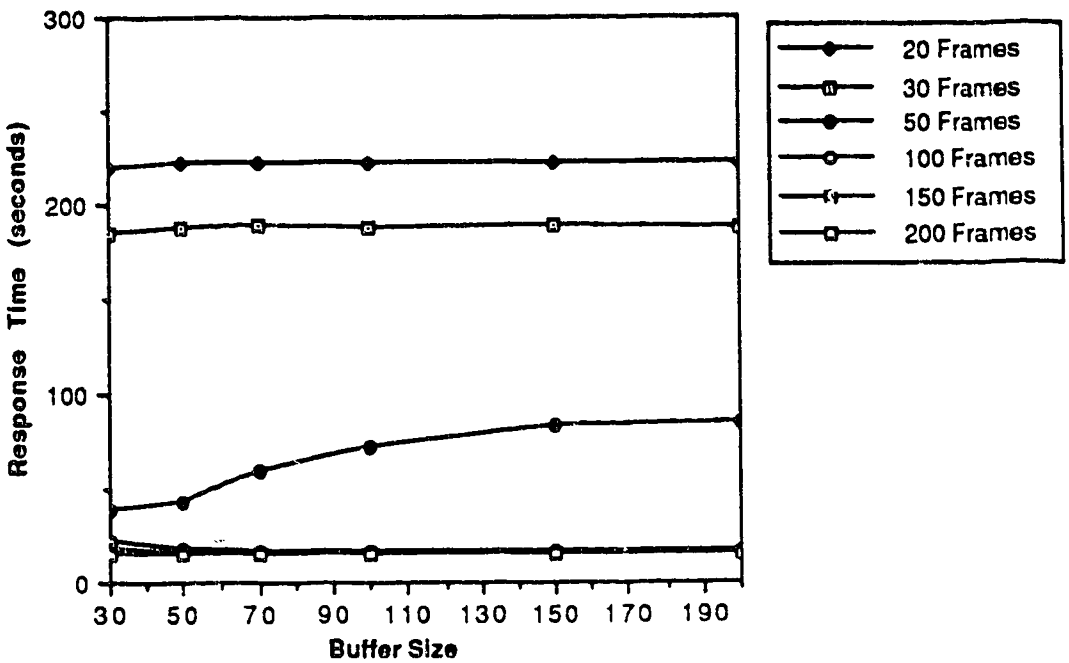

Figure 5.34: Response Time (DBMIN model) for C-C Nested Loop Join Queries

same thing happens at these two points as at a main memory size of 50 frames and a buffer size of 50 in the Hot Set model. That is, at these points, there is a high probability that a buffer that is chosen for replacement by the buffer manager will also be chosen by the page manager for replacement (if the buffer is in the main memory) when there is a major page fault. This phenomenon is also exhibited for the DBMIN model at a buffer size of 70 and a main memory size of 50.

The fluctuation in the throughput at a buffer size of 50 and a main memory size of 50 in the Hot Set model is reflected in the page fault curve (Figures 5.35, 5.36), the double fault curve (Figures 5.37, 5.38), and the buffer fault curve (Figures 5.39, 5.40). At these points, there is a high probability that a running process will find a page it is referencing is being written out to disk or being read into the main memory. That is, there is more sharing of DBMS pages at these points in the Hot Set and DBMIN model in the nested loop join runs than at similar points in the merge scan join runs. This is

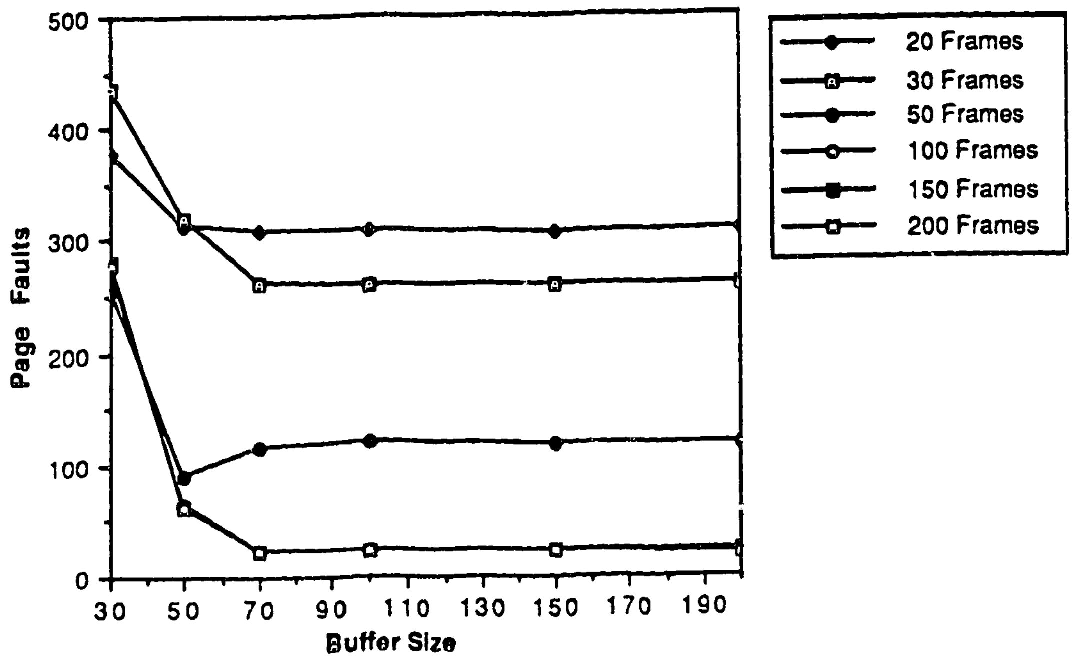

Figure 5.35: Page Faults (Hot Set model) for C-C Nested Loop Join Queries

because in the merge scan runs, unsharable temporary pages are used. Also, note that in some instances, the number of page faults and buffer faults are higher at main memory sizes of 200 and 150 than at a main memory size of 100 in the Hot Set model. Since the double fault curves do not show such a trend, it is the scarcity of frames that causes this behavior. At these points, process suspension occurs frequently due to unavailability of a frame to bring in a DBMS page. The suspension can either be due to a major buffer fault or a major page fault.

At a buffer size of 70, the number of buffer faults and double faults at a main memory size of 30 are higher than those at a main memory size of 20 for the Hot Set model. This is not exhibited in the sequential scan or the merge scan join simulation runs. Even though there is an instance of the merge scan join where the number of buffer faults and double faults are higher at a main memory size of 30 than at 20, the reason behind it is different: the number of double faults is close to the number of buffer faults at these points (in fact, at most points, especially at a buffer size greater than or equal to a

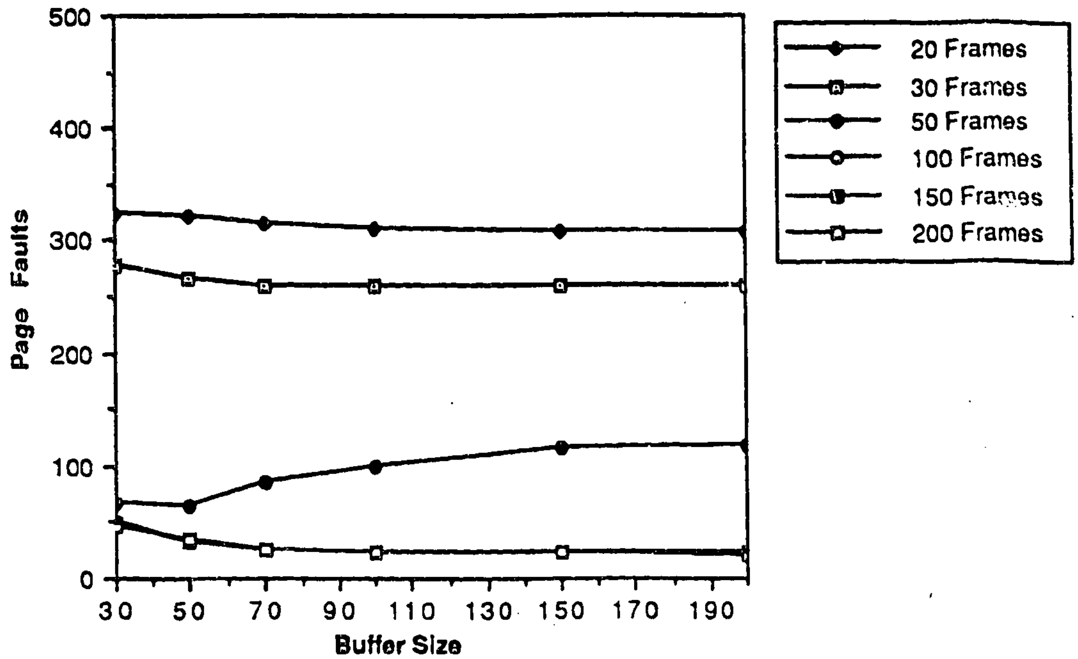

Figure 5.36: Page Faults (DBMIN model) for C-C Nested Loop Join Queries

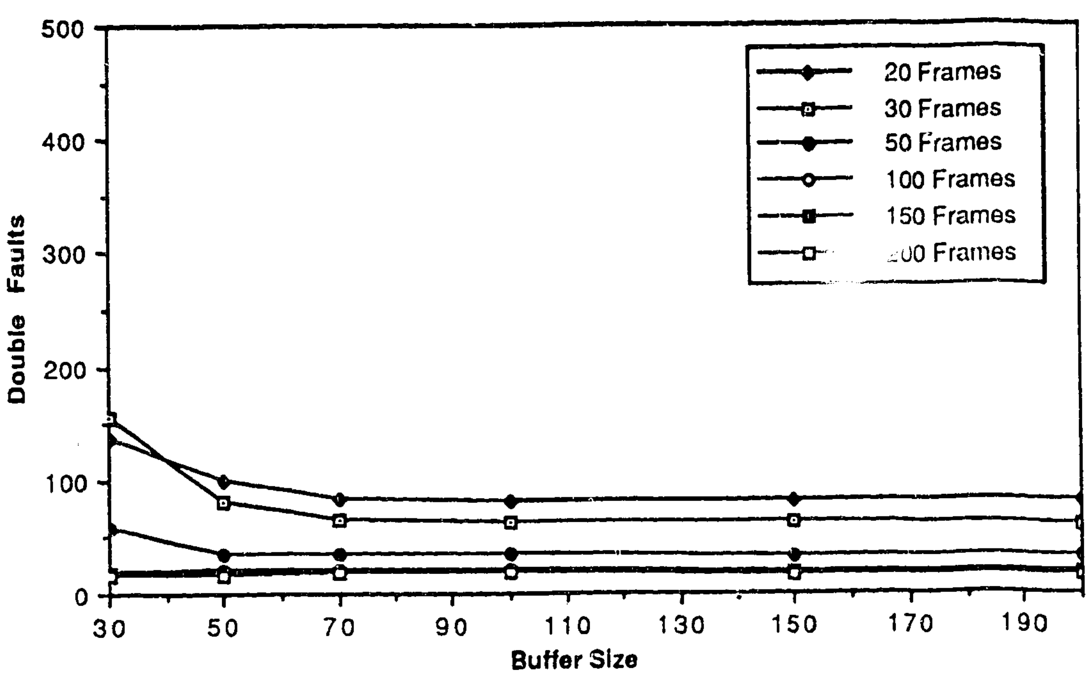

Figure 5.37: Double Faults (Hot Set model) for C-C Nested Loop Join Queries

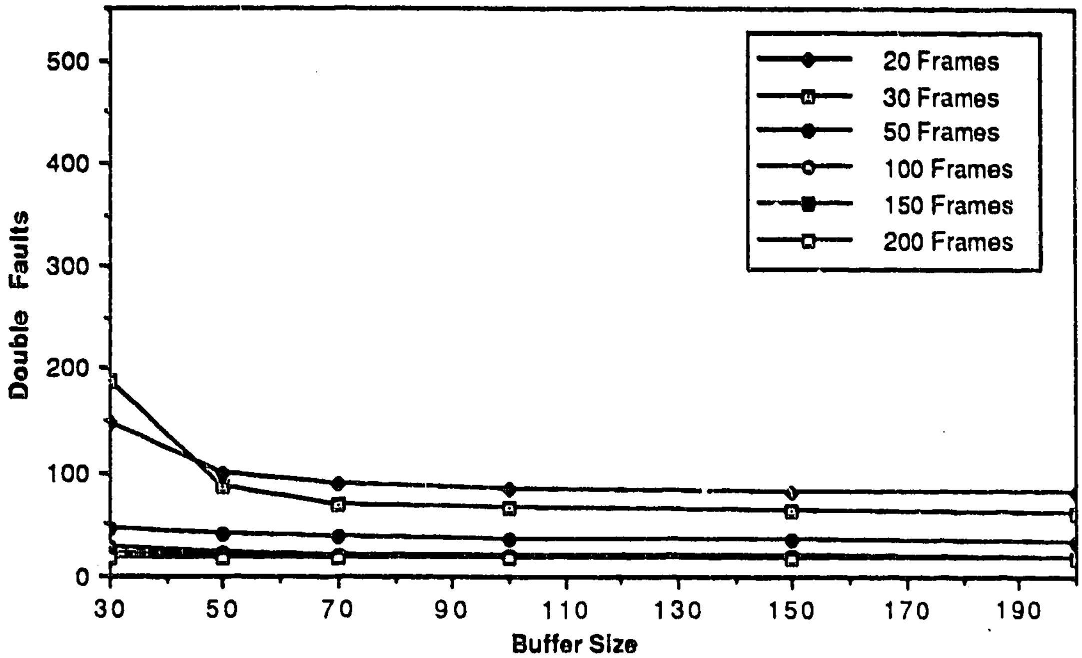

Figure 5.38: Double Faults (DBMIN model) for C-C Nested Loop Join Queries

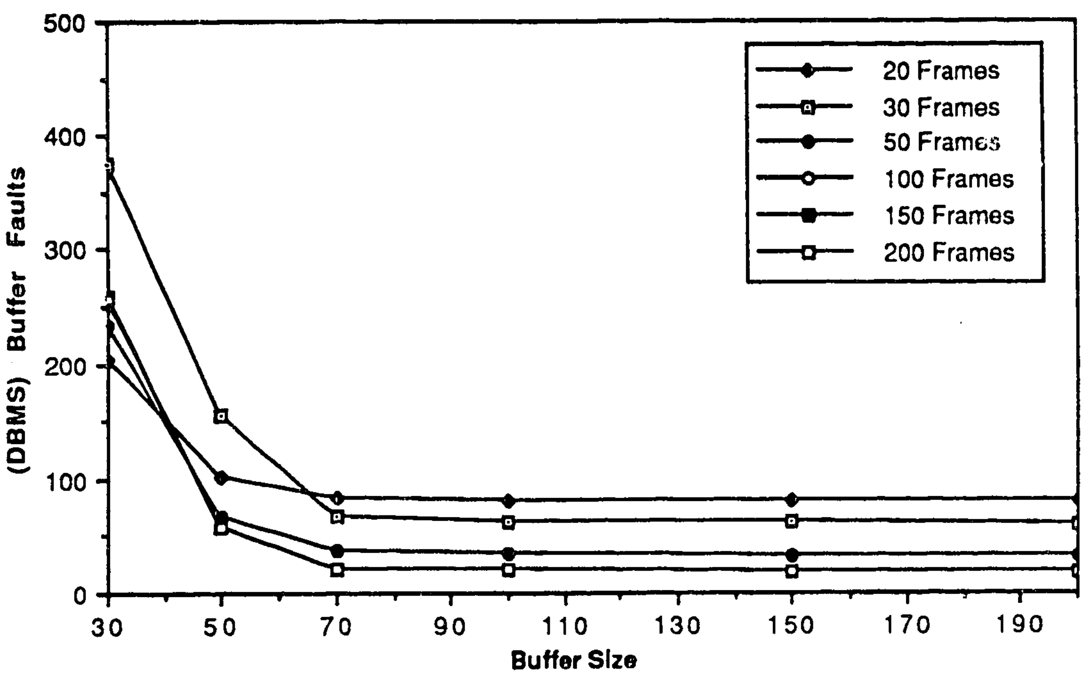

Figure 5.39: (DBMS) Buffer Faults (Hot Set model) for C-C Nested Loop Join Queries

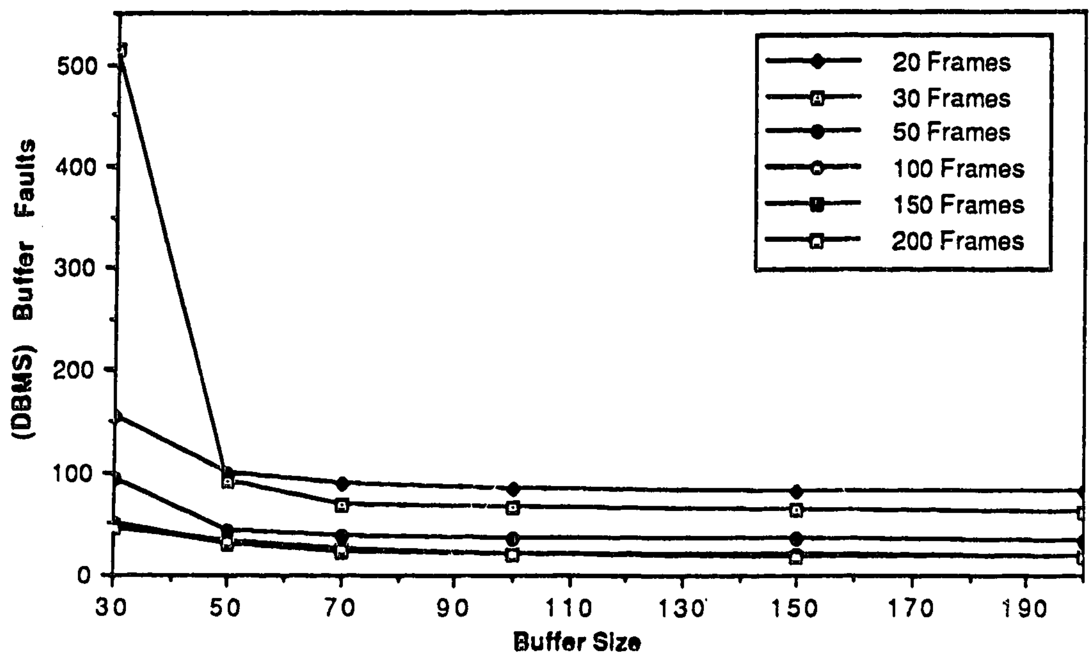

Figure 5.40: (DBMS) Buffer Faults (DBMIN model) for C-C Nested Loop Join Queries

100 in both algorithms) in the nested loop join runs. Similar to the case in the Hot Set model at a buffer size of 50 and a main memory size of 50, at these particular points, the probability of finding that a DBMS page that is being referenced by the running process is in I/O is higher at a main memory size of 30 frames. When a running process references a DBMS page that is in I/O, we increment the double fault, page fault and buffer fault counters for the current running process if the event occurs in the buffer manager. If the event occurs in the page manager, only the page fault counter is incremented. This is further demonstrated by the fact that even though the number of buffer fault and double fault at main memory size of 30 are higher than that at 20, the number of I/Os (see Figures 5.41, 5.42) is not.

In the DBMIN model, at a buffer size of 70, though the number of buffer faults at a main memory size of 30 is higher than that at a main memory size of 20, the number of double faults does not exhibit such a behavior. Also, note that the number of page faults at a buffer size of 70 and a main memory

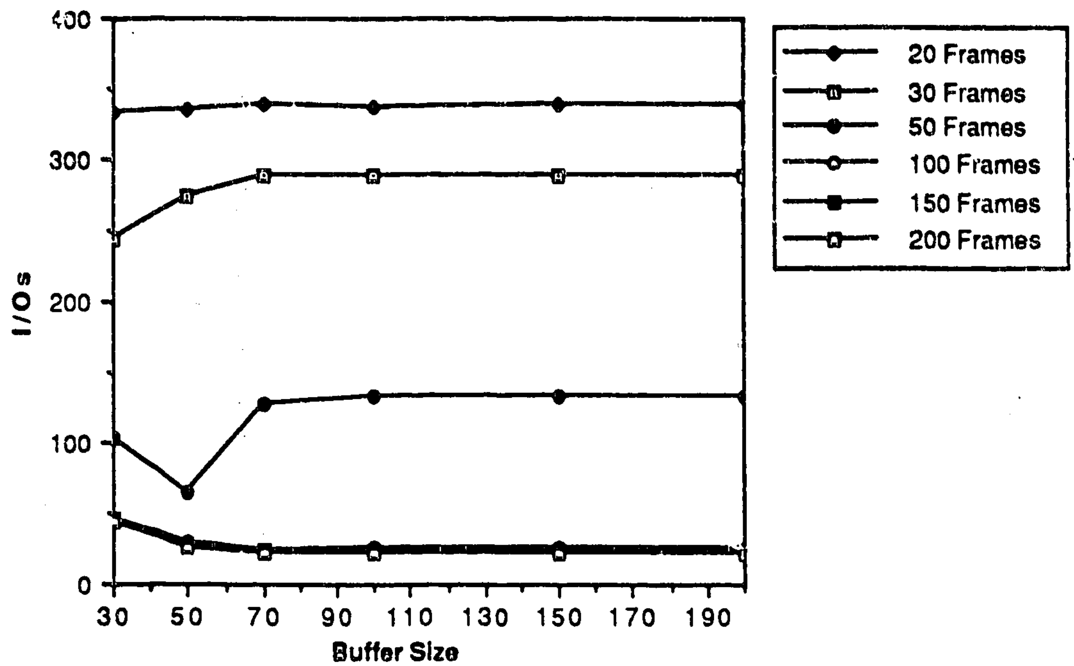

Figure 5.41: I/Os (Hot Set model) for C-C Nested Loop Join Queries

size of 30 is higher than that at the same point in the Hot Set model. Though the number of buffer faults and page faults at this point in the DBMIN model are higher, the number of I/Os in both algorithms at this point is similar. Therefore, we can say that at a main memory size of 30 and a buffer size of 70, there is more process suspension in the DBMIN model than in the Hot Set model due to the unavailability of buffers/frames for replacement.

5.4.2 Summary

In the DBMIN model, the number of page faults and double faults at lower main memory sizes are always higher than the respective faults at higher main memory sizes. This is not true in some cases in the Hot Set model. This shows that suspension of processes rarely occurs in the page manager in the runs of the DBMIN model. Also, the curves of the DBMIN model do not have fluctuations which exist in the curves of the Hot Set model. Thus, the DBMIN model has better control than the Hot Set model.

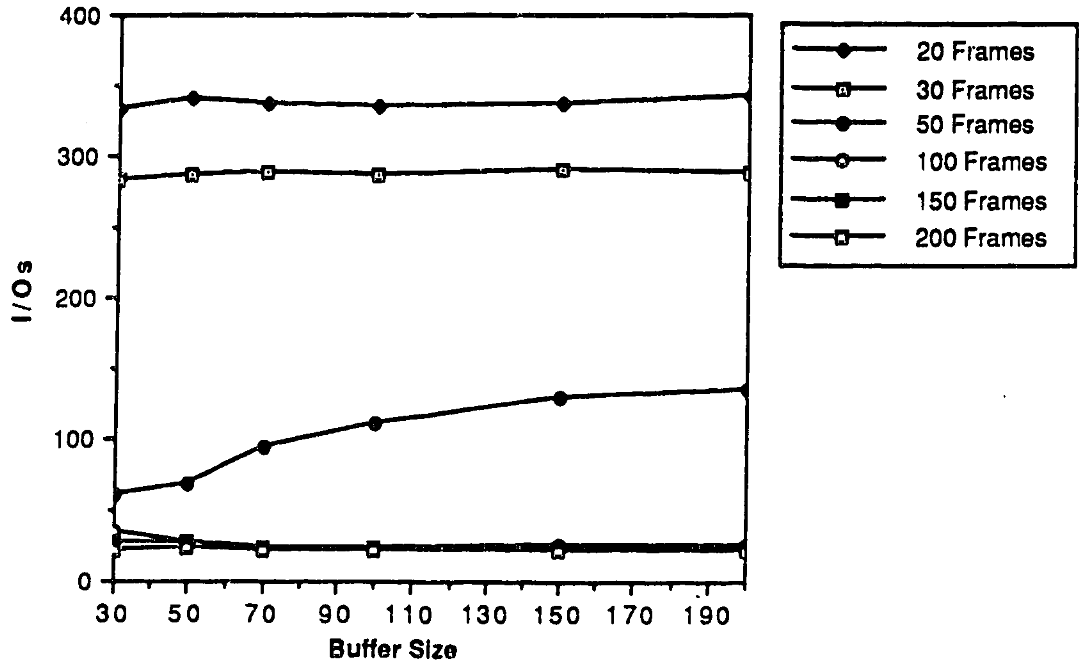

Figure 5.42: I/Os (DBMIN model) for C-C Nested Loop Join Queries

5.4.3 Non-Clustered Outer Indices and Clustered Inner Indices

The shapes of the curves representing these simulation runs are quite similar to those of the previous runs. Since the indices to the outer relation are non-clustered, the locality of the reference pattern is harder to capture. In fact, using Yao's function, 32 different blocks/pages can be accessed in executing the join of each query in our runs. Thus, the throughput of these runs (Figures[4] 5.43, 5.44) are lower than the previous ones.

The highest throughput achieved with the Hot Set model is about 1.23917 queries/second whereas with the DBMIN model, it is about 1.10131 queries/second. In general, the throughput of the Hot Set model seems to be higher than that of the DBMIN model. This might be due to the differences in the buffer size estimates. For the Hot Set model, the estimated optimal buffer size is 33.8. In the DBMIN model, the estimated optimal buffer size is 38.4.

[4] The abbreviation NC-C in the figures means that non-clustered indices are used in the outer relation and clustered indices are used in the inner relation.

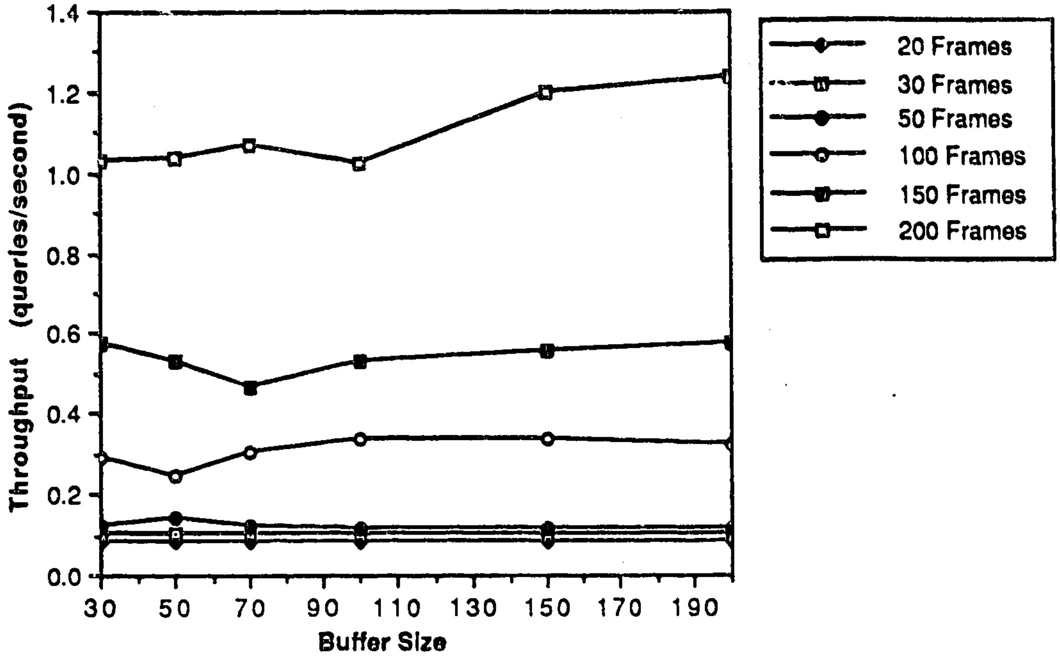

Figure 5.43: Throughput (Hot Set model) for NC-C Nested Loop Join Queries

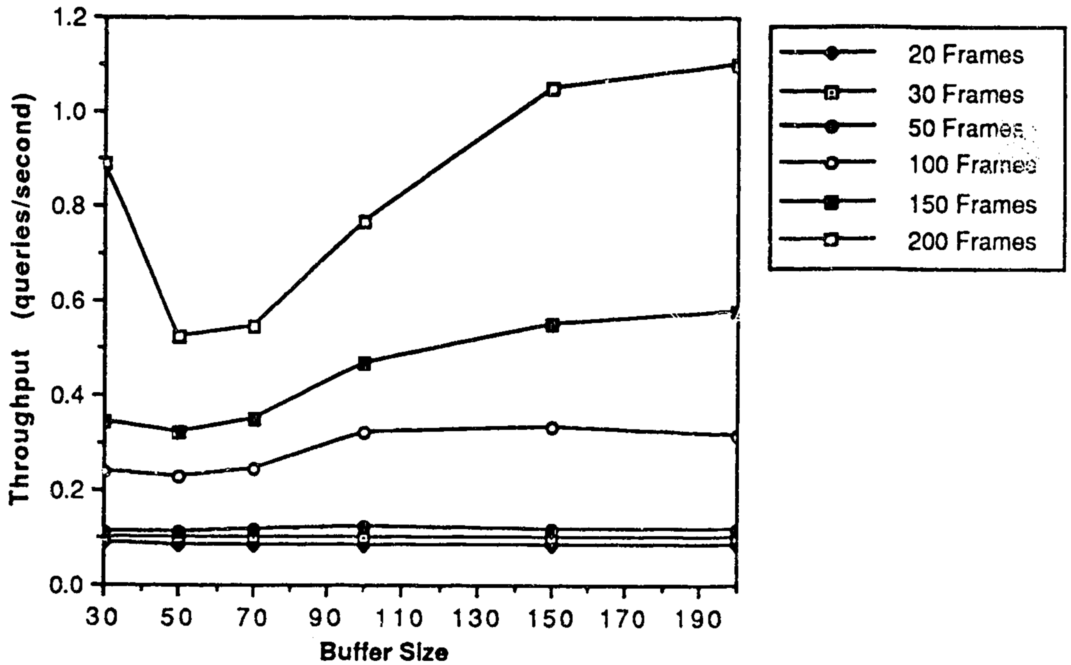

Figure 5.44: Throughput (DBMIN model) for NC-C Nested Loop Join Queries

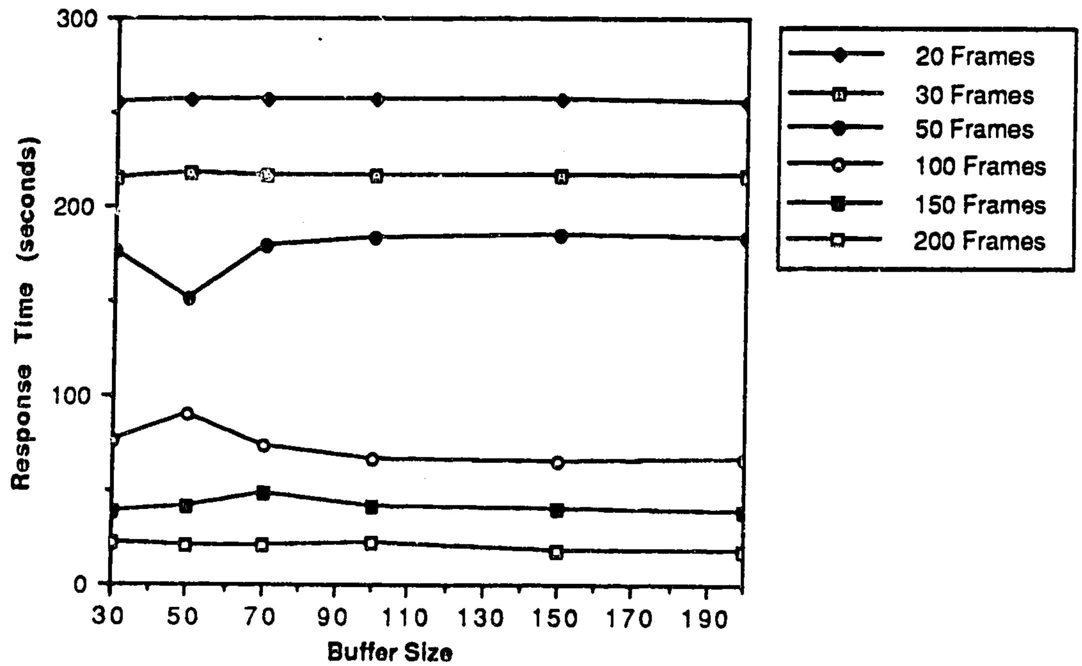

Figure 5.45: Response Time (Hot Set model) for NC-C Nested Loop Join Queries

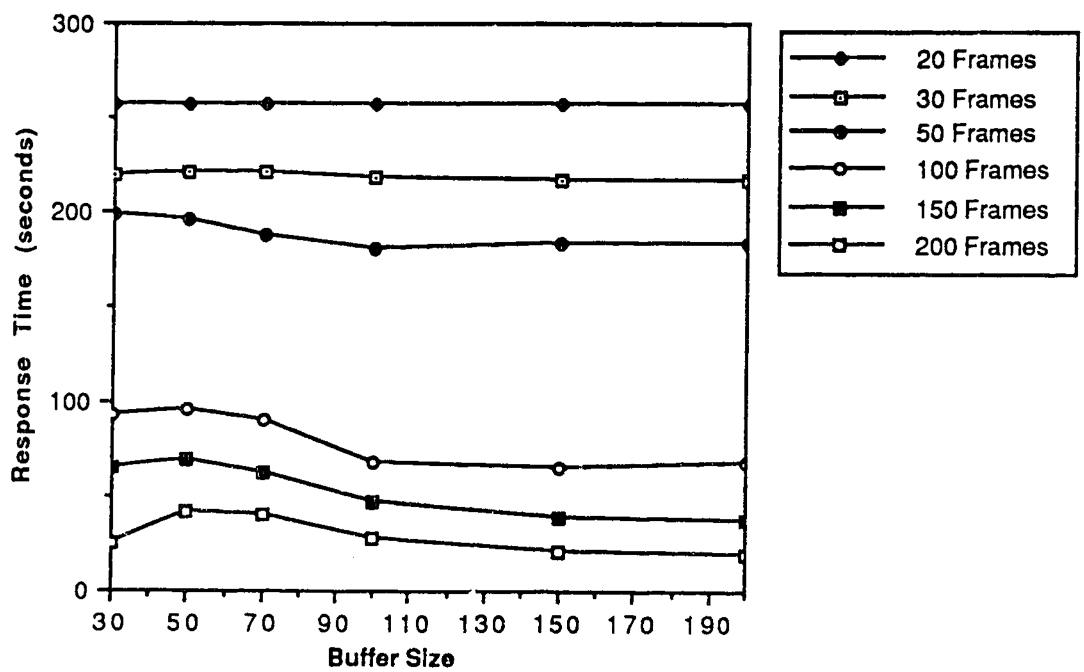

Figure 5.46: Response Time (DBMIN model) for NC-C Nested Loop Join Queries

Since the sizes of the buffer pool and main memory are not enough to contain the locality of all 24 concurrent queries, most of the queries are faulting pages from each other. This is especially significant in the DBMIN model, where if a buffer cannot be found in a locality set for replacement, the manager will try to get it from another process though this faulting process might have buffers for replacement in other locality sets. In the Hot Set model, all the buffers allocated to the faulting process are available for replacement, thus the probability of stealing a buffer from another process is lower. The probability of stealing a buffer from another process is higher in the DBMIN model also because the optimal buffer size of a query is higher than that of the Hot Set model. A process will seldom replace a buffer from its own locality set.

At a lower buffer size, with fewer processes in the running/ready state, the probability of capturing the locality in the main memory is higher, therefore at a smaller buffer size, the throughput is higher in most cases. This is especially exhibited by the DBMIN model at a main memory size of 200 frames. There are dips at some points in both models because more processes are allowed to compete for resources. At these point, the stealing of buffers/frames among the processes becomes significant as the locality of the queries cannot be contained in the buffers/frames. As the buffer size increases, the throughput increases because the stealing of buffers becomes less significant. This is because as the buffer size increases, the probability of capturing the the locality of the queries is higher though there are more processes running. Also, buffers can be shared. The above phenomenon also explains the curves of the response time, the double faults and the number of I/Os.

The increase at a main memory size of 50 and a buffer size of 50 in the Hot Set model of the previous run is also exhibited here but it is not as significant. This is because the page reference pattern is more randomized in a non-clustered index access path. The same explanation given in the previous run at this point can also be given to the corresponding decrease in double faults, page faults, number of I/Os and response time.

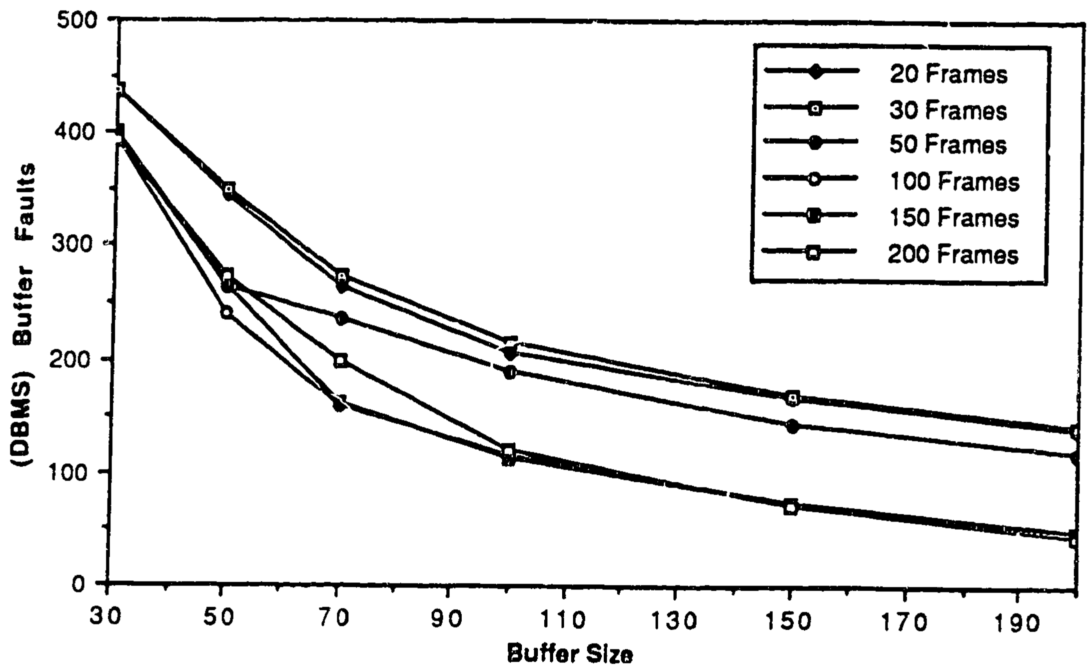

Figure 5.47: (DBMS) Buffer Faults (Hot Set model) for NC-C Nested Loop Join Queries

Notice that as in the previous runs, at some buffer sizes, the number of buffer faults (Figures 5.47, 5.48) and the number of double faults (Figures 5.49, 5.50) are higher at a main memory size of 30 than at a main memory size of 20. The same explanation can be given for this behavior. The trend with the number of page faults (Figures 5.51, 5.52) and buffer faults in the Hot Set model at a main memory size of 200 of the previous run can also be seen here in the Hot Set model.

5.4.4 Summary

Though there are dips/peaks in the DBMIN model at some points, if we compare the curves of the Hot Set model and DBMIN model, the fluctuations in the curves of the DBMIN model are less significant. Also, note that the response time curves of the DBMIN model approach constant values with less fluctuations than the Hot Set model, so do the buffer fault curves, double fault curves, page fault curves and I/Os curves. These again show that the DBMIN

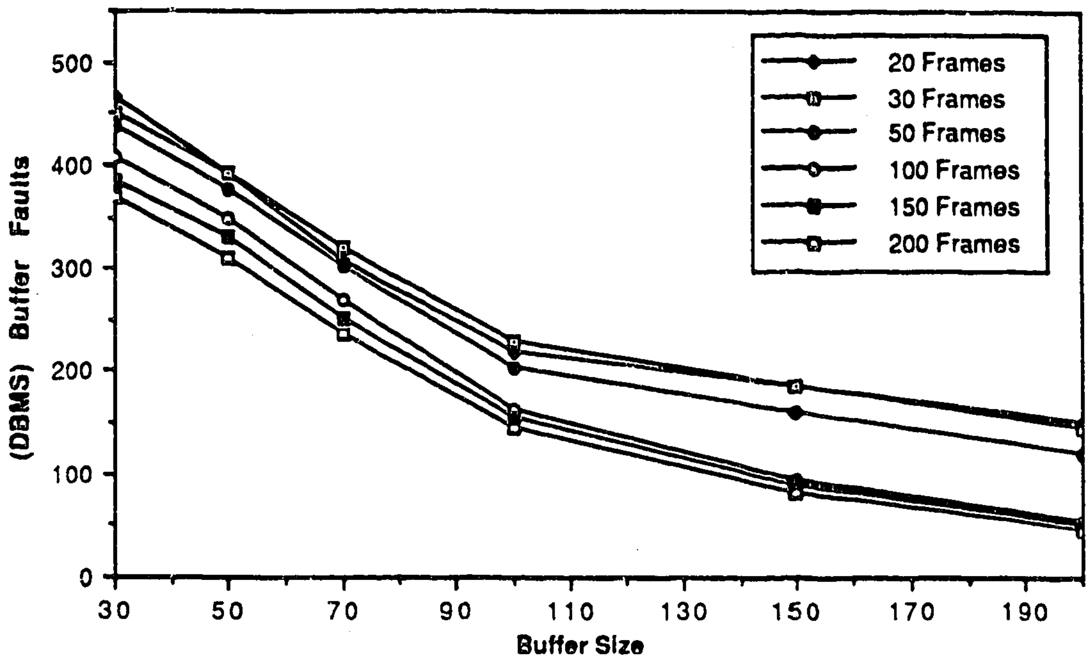

Figure 5.48: (DBMS) Buffer Faults (DBMIN model) for NC-C Nested Loop Join Queries

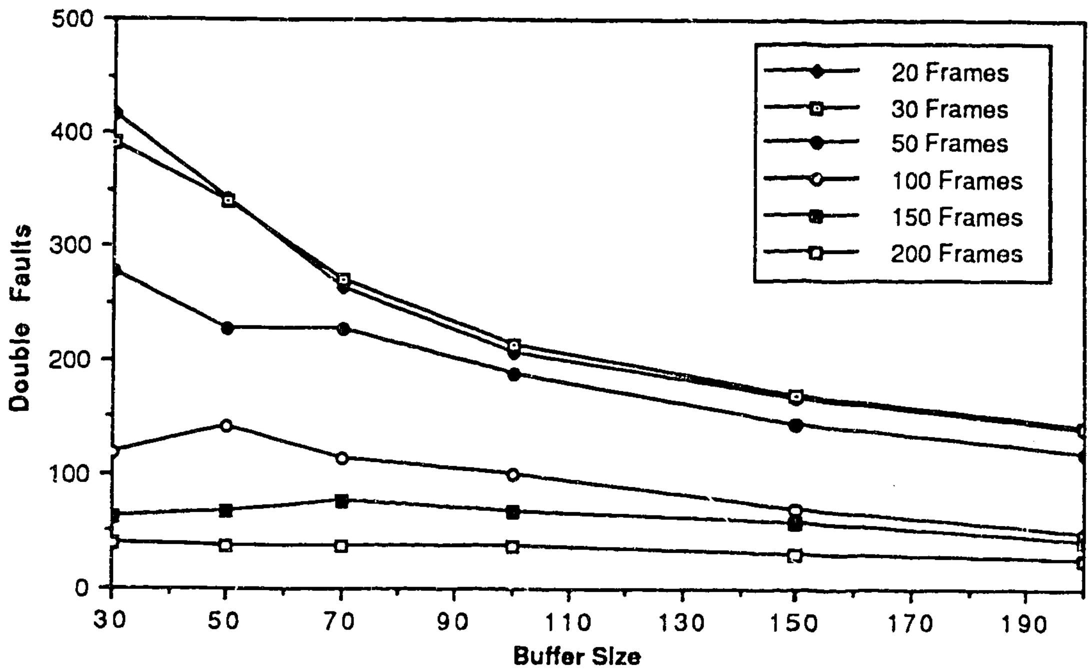

Figure 5.49: Double Faults (Hot Set model) for NC-C Nested Loop Join Queries

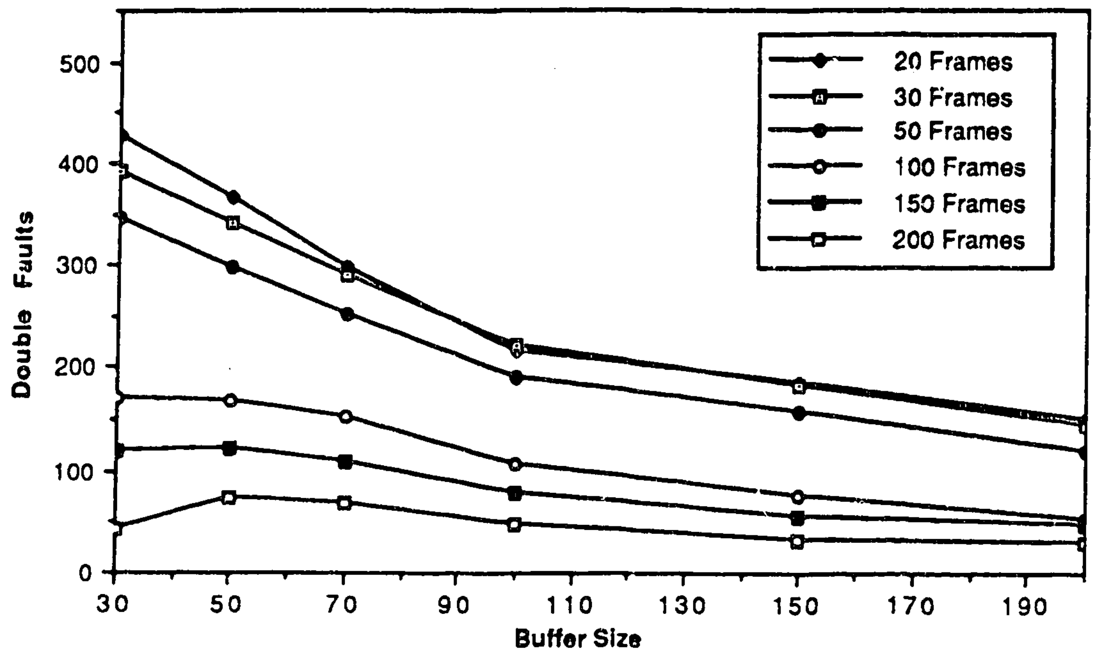

Figure 5.50: Double Faults (DBMIN model) for NC-C Nested Loop Join Queries

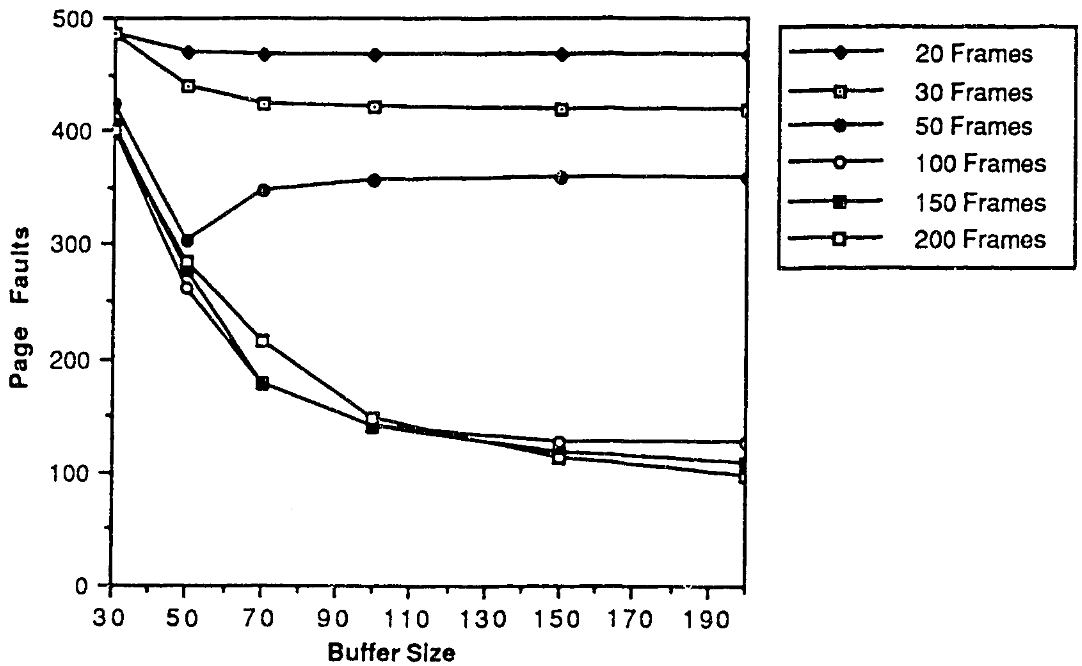

Figure 5.51: Page Faults (Hot Set model) for NC-C Nested Loop Join Queries

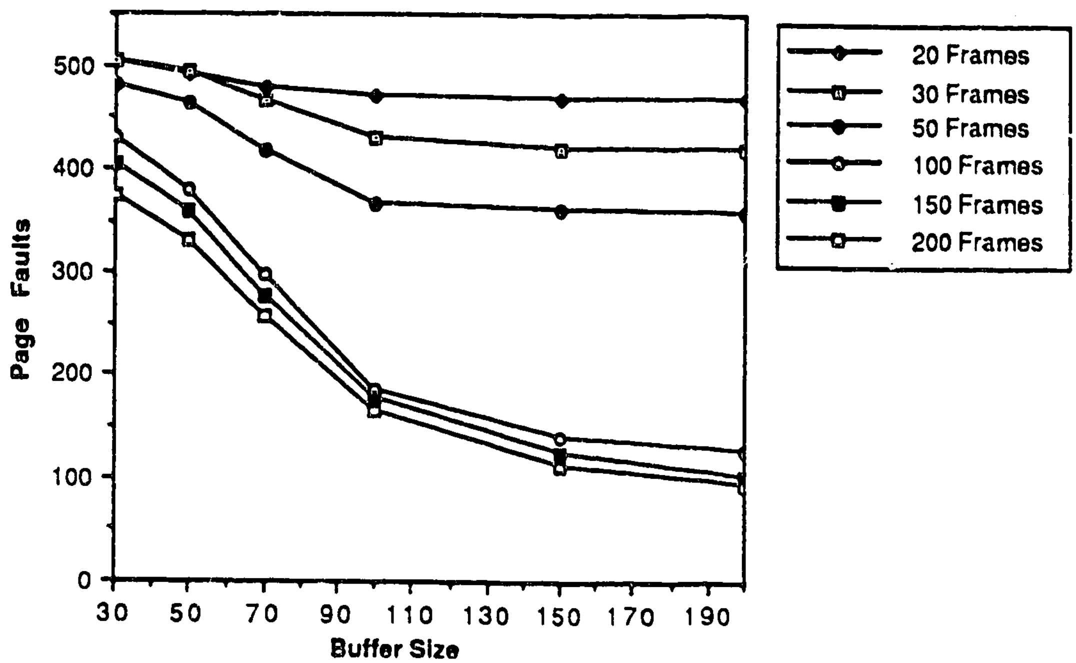

Figure 5.52: Page Faults (DBMIN model) for NC-C Nested Loop Join Queries

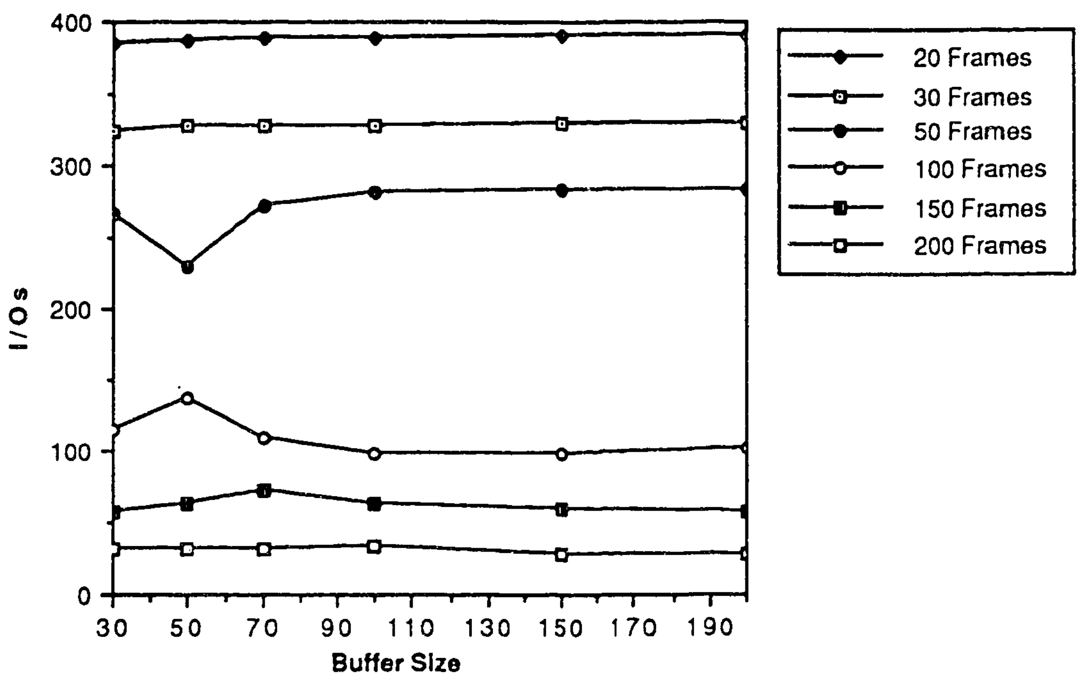

Figure 5.53: I/Os (Hot Set model) for NC-C Nested Loop Join Queries

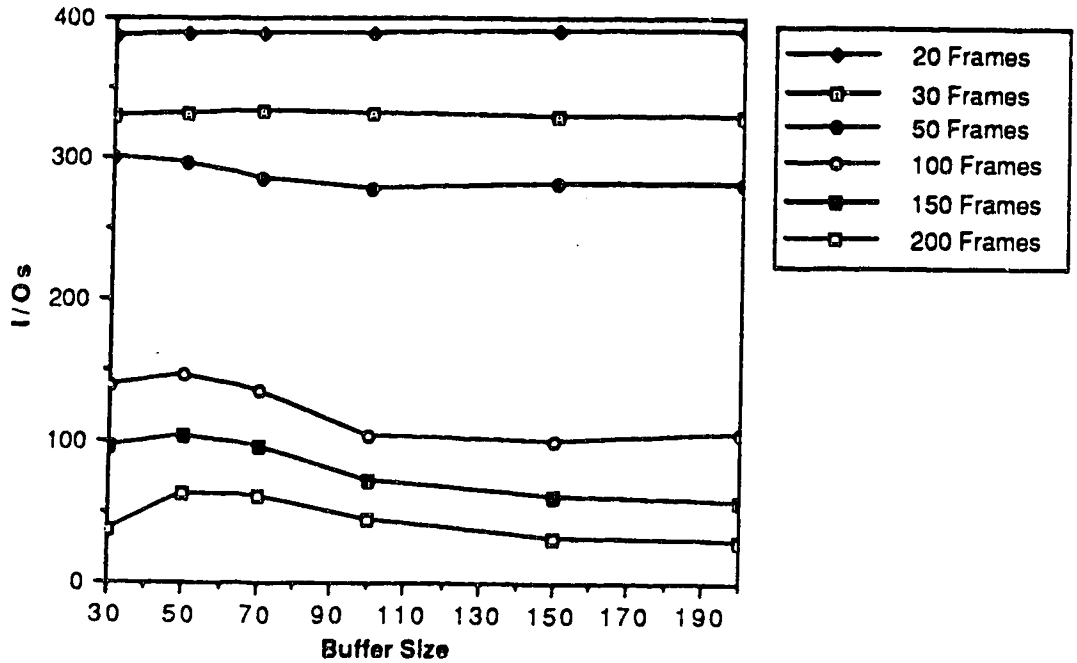

Figure 5.54: I/Os (DBMIN model) for NC-C Nested Loop Join Queries

model has better control over the sequence of execution of the processes.

5.5 Summary

In this chapter we present the results from the performance studies which have been done in this thesis. Results are obtained from simple selection queries which are executed without indices, simple selection queries which are executed with non-clustered indices, join queries which are executed with the merge scan join method with clustered indices on both the outer and inner relations, join queries which are executed with the nested loop join method with clustered indices on both the outer and inner relations, and non-clustered indices on the outer relation and clustered indices on the inner relation.

It is found that the response time is closely related to the number of I/Os. This is because the amount of time taken to process an I/O is 30,000 virtual time units.

If the main memory size is large enough to contain the locality of the queries, then increasing the buffer size is beneficial. Otherwise, higher throughput is obtained with a lower buffer size. With a lower buffer size, the number of running/ready processes is limited by the buffer manager. With fewer ready/running processes, there is less stealing of buffers/frames among the processes. Thus, less faults are generated and the response times of the queries are therefore reduced.

As the buffer size increases, the number of I/Os caused by the buffer replacement policy decreases and the number of I/Os caused by the page replacement policy increases. This is because as the buffer size increases, more DBMS pages are in the DBMS buffers but more virtual pages are mapped into a fixed number of frames. Therefore, if the cost ratio of an I/O between the DBMS disk and the paging disk is greater than 1, then the response times will not be similar to the results reported here.

From the results studied, we can see that the curves of the DBMIN algorithm have less fluctuations than those of the Hot Set algorithm. This is due to the fact that faulting/replacement of buffers/frames are controlled more strictly in the DBMIN model. This is because, in most cases, there is more than one locality set in each process and a faulting process can only replace a buffer within the faulting locality set, whereas in the Hot Set model, each process has only one locality set.

Chapter 6

Conclusions

6.1 Summary of Thesis

In this thesis, the performance of two special DBMS buffer managers, the Hot Set model and the DBMIN model, in a virtual memory system is studied. A closed network trace driven simulator which emulates the execution of the DBMS queries in a virtual memory system is used. The simulator consists of a computer model and a page reference string generator. The page reference string of each query is generated before the query is executed in the computer model. A random number generator is used to determine the predicates of the queries. Each query has a selectivity factor of less than ten per cent. The types of queries studied are

- simple selections with and without indices,
- joins executed via the merge scan method with clustered indices on both outer and inner relations,
- joins executed via the nested loop method with clustered and non-clustered indices on the outer relation and clustered indices on the inner relation.

The virtual memory manager uses a GLRU page replacement policy and the maximum number of concurrent queries is twenty-four. A virtual time unit is used to control event timing. This virtual time unit is equivalent to the amount of time taken to make a successful page reference. It takes about 30,000 virtual time units to read a page from the disk. The scheduler uses a Round-Robin policy to schedule processes and the I/O driver implements a First-Come-First-Serve policy.

6.2 General Conclusions

From most of the mean buffer fault curves, mean page fault curves and mean double fault curves, we can conclude that as the buffer size increases, the number of faults decreases. However, from the throughput curves, we cannot conclude that the throughput increases proportional to the buffer size, nor can we conclude that there is an inverse correlation between the buffer size and the mean response time or the mean number of I/Os.

In our system, if a process references a page that is being brought into the main memory by another process, the process is said to have caused a fault. The type of fault that will be generated depends on whether the page is being processed by the DBMS buffer manager or the virtual memory manager. If the page is processed by the DBMS buffer manager then, a page fault, a buffer fault and a double fault take place. If the fault occurs while the virtual memory manager is processing the page reference then, a page fault occurs. Therefore, if a page is not in the main memory and it is being referenced, updates to the various fault counters will take place regardless of whether an actual I/O will take place. Thus, these fault counters can give us an estimate of how the system will perform if each process causes an I/O on referencing a page taht is not in main memory (even if the page is being brought in by another process). Accordingly, an analysis of the various fault curves supports a prediction that the throughput will increase and the mean response time will decrease with respect to an increase in buffer size at a fixed main memory size. This predication is based on the observation that the number of buffer faults decreases as the buffer size increases. This phenomenon exists in both the Hot Set model and the DBMIN model.

We cannot be certain about whether the number of I/Os will decrease since the number of major page faults increases as the buffer size increases at a fixed main memory size. The number of I/Os decreases if the main memory can contain the locality of the queries. If the cost ratio between an access to the database disk and an access to the paging disk is greater than 1, then the merge scan method should give better results than the results we have obtained. This is because, in the merge scan method, the number of I/Os to the database disk is minimized by the pre-selection of tuples which satisfy the predicates before the join is actually performed. Once the tuples that will participate in the join are selected, the DBMS pages are never reaccessed again to execute the query, only the temporary pages that hold the pre-selected tuples and which reside in virtual memory will be accessed. Therefore, if the cost ratio is greater than 1, then we expect to see a better performance for this join method.

In this performance study, we did not implement a load controller in the buffer manager. If there is one, the results might be different since stealing of buffers among queries will be a rare event. Also, the number of ready/running processes will be limited which has been shown to be desirable in some cases in our study, such as, at low main memory and low (DBMS) buffer sizes.

At the peaks/dips of some curves, it has been shown that the compatibility between the page replacement policy and the buffer replacement policy is important. It is desirable to have the buffer manager and the page manager choose the same buffer/frame (with respect to DBMS page) for replacement. This either implies that the behavior of the page replacement policy approaches that of a local policy or the behavior of the page manager approaches that of the buffer manager.

In most cases, the performance of the Hot Set model and the DBMIN model are quite similar; that is, the mean values of the various measurements in the Hot Set model are within the confidence interval of the respective mean values in the DBMIN model and vice versa. The DBMIN model does not exhibit these fluctuations and the curves that describe its behavior are more monotonic (increasing or decreasing) than the Hot Set model. This is due to the DBMIN model's exercise of control over its locality sets. Since the performance of both models are quite similar, one would choose the less complex one to implement. In this case, the Hot Set model is the less complex since each query has only one locality set and only one type of replacement policy. In the DBMIN model, the maximum number of locality sets in our simulation runs is six and each process can have a maximum of three different replacement policies. This implementation will not only increase the complexity but also the size of the system. Unfortunately, we cannot confirm Chou's [Cho85] claims that the DBMIN model is superior to the Hot Set model in the experimental environment of this thesis.

6.3 Suggestions for Implementation of a DBMS in a Virtual Memory System

The types of faults that are generated when a DBMS is run on top of a virtual memory system are the buffer faults generated by the DBMS buffer manager, the page faults or the reference faults generated by the page manager and the double faults. A double fault occurs when a buffer fault and a page fault are generated on a page reference. The page fault component in a double fault is generated when the virtual memory page which maps the replaced DBMS buffer is not in the main memory. Since the virtual memory page will be written over with a different DBMS page, there is no need to bring in the virtual memory page from the paging device because the data in the virtual memory page will not be accessed but will be replaced. Therefore, only a frame to bring in the DBMS page from the DBMS disk is required. This will eliminate the page fault component in a double fault and thus there will not be any double faults. To implement such an approach, the page manager has to know whether a page reference has already generated a buffer fault. Therefore, the DBMS buffer manager and the page manager require some form of cooperation.

Another way to eliminate double faults is to map all the DBMS buffers into main memory. Dynamic sharing of the main memory has been shown to give a better performance than static sharing in previous studies. One way of implementing a dynamic sharing system is to have a "moderator" to check on the load of the DBMS and the load of the system handling the other tasks running in the system. If the load of the DBMS is low and the load of the system handling the other tasks is high then, some frames from the DBMS can be transferred to the system handling the other tasks and vice versa.

6.4 Suggestions for Future Research

There are several extensions that could be made to the work presented here.

- The presence of a load controller in the buffer manager has been shown to be desirable in some cases in the study performed. Thus, implementation of a load controller in the buffer manager is desirable. In some DBMS, for example in a airline ticket reservation system, most transactions are read only and thus most of the database pages can be shared. Therefore, a desirable load controller is one that considers the sharing of pages among the transactions.

- Implementation of a virtual memory manager with different page replacement policies, like local LRU, since the decisions of the page replacement algorithm have been shown to affect the performance of the two models studied in our environment.

- In most DBMS disk accesses, the cost of an access (with respect to time) to the DBMS disk is higher than the cost of an access to the paging device, therefore, experiments using different cost ratios should be performed.

- Experimenting with partitioned main memory where the database buffers all map to main memory frames rather than virtual memory pages. This approach will eliminate the overhead in the paging of the virtual pages which map the DBMS buffers. That is, page faults will not be generated in all page references belonging to the queries since all the DBMS buffers are mapped into main memory frames. Thus, there will not be any double faults.

  In the earlier studies by Effelsberg and Haerder [EH84], and Brice and Sherman [SB76,BS77], dynamically partitioning of main memory, that is when the main memory is shared dynamically between the DBMS buffers and other tasks, has been shown to improve the performance of the system. Therefore, experiments with dynamic partitioning of main memory should be performed with the two models studied.

Bibliography

[Bac86] M. J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1986.

[BD84] H. Boral and D. J. DeWitt. A methodology for database system performance evaluation. In *ACM Proceedings of the International Conference on Management of Data*, pages 176–185, ACM, June 1984.

[BE77] M. W. Blasgen and K. P. Eswaran. Storage and access in relational data bases. *IBM Systems Journal*, 16(4):363–377, 1977.

[BM72] R. Bayer and E. McCreight. Organisation and maintainance of large order indexes. *Acta Informatica*, 1(3):173–189, 1972.

[BS77] R. S. Brice and S. W. Sherman. An extension of the performance of a database manager in a virtual memory system using partially locked virtual buffers. *ACM Trans. on Databases*, 2(2):196–207, June 1977.

[CD86] H. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. *Algorithmica*, 311–336, 1986.

[Cho85] H. Chou. *Buffer Management of Database Systems.* Technical Report 597, Computer Science Department, University of Wisconsin, Madison, U.S.A., May 1985.

[CO72] W. W. Chu and H. Opderbeck. The page fault frequency replacement algorithm. *Proceedings of the AFIPS Fall Joint Computer Conference*, 597–609, 1972.

[Com79] D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11(2):121–137, June 1979.

[Dei84] H. M. Deitel. *An Introduction to Operating Systems (2nd Edition).* Addison-Wesley Publishing Company, 1984.

[Den68a] P. J. Denning. Thrashing: its causes and prevention. *Proceedings of the AFIPS Fall Joint Computer Conference*, 33:915–922, 1968.

[Den68b] P. J. Denning. The working set model for program behavior. *ACM Communications*, 11(5):323–333, May 1968.

[Den70] P. J. Denning. Virtual memory. *ACM Computing Surveys*, 2(3):153–189, September 1970.

[Den80] P. J. Denning. Working sets past and present. *IEEE Trans. on Software Engineering*, SE-6(1):64–84, January 1980.

[EH84] W. Effelsberg and T. Haerder. Principles of database buffer management. *ACM Trans. on Databases*, 9(4):560–595, December 1984.

[FLW78] I. B. Fernandez, T. Lang, and C. Wood. Effects of replacement algorithms on a paged buffer database system. *IBM Journal of Research and Development*, 22(2):185–196, March 1978.

[Gra78] J. N. Gray. Notes on data base operating systems. In Goos and Hartmanis, editors, *Operating Systems - An Advanced Course*, pages 393–481, Springer-Verlag, 1978.

[LWF77] T. Lang, C. Wood, and I. B. Fernandez. Database buffer paging in virtual storage systems. *ACM Trans. on Databases*, 2(4):339–351, December 1977.

[Mac73] International Business Machines. *Information Management System/360, Version 2-General Information Manual.* Technical Report GH 20-0765, IBM Corporation, White Plains, N. Y., U.S.A., 1973.

[Mas77] T. Masuda. Effect of program localities on memory management strategies. *Proceeding of Sixth ACM Symposium on Operating Systems Principles*, 117–124, November 1977.

[ML86] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for local queries. In *ACM Proceedings of the International Conference on Management of Data*, pages 84–95, May 1986.

[MOO87] M. Maekawa, A. E. Oldehoeft, and R. R. Oldehoeft. *Operating Systems Advanced Concepts.* The Benjamin/Cummings Publishing Company, 2727 Sand Hill Road, Menlo Park, CA 94025, U.S.A., 1987.

[Ozs88] M. T. Ozsu. *Distributed Database Operating Systems.* Technical Report TR88, University of Alberta, Edmonton, Canada, February 1988.

[PS85] J. L. Peterson and A. Silberschatz. *Operating System Concepts.* Addison/Wesley Publishing Company, 1985.

[SB76] S. W. Sherman and R. S. Brice. Performance of a database manager in a virtual memory system. *ACM Trans. on Databases*, 1(4):317–343, December 1976.

[SH87] R. O. Simpson and P. D. Hester. The ibm rt pc romp processor and memory management unit architecture. *IBM Systems Journal*, 26(4):346–360, 1987.

[Smi81] A. J. Smith. Input/output optimization and disk architectures: a survey. *Performance and Evaluation*, 1:104–117, 1981.

[SS82] G. M. Sacco and M. Schkolnick. A mechanism for managing the buffer pool in a relational database system using the hot set model. In *ACM Proceedings of the Eight International Conferences on Very Large Databases*, pages 257–262, September 1982.

[SS86] G. M. Sacco and M. Schkolnick. Buffer management in relational database systems. *ACM Trans. on Databases*, 11(4):473–498, December 1986.

[Sto81] M. Stonebraker. Operating system support for database management. *Communications of the ACM*, 24(7):412–418, July 1981.

[TM82] A. S. Tanenbaum and S. J. Mullender. Operating system requirements for distributed data base systems. In Schneider, editor, *Distributed Data Bases*, pages 105–114, North-Holland, Amsterdam, 1982.

[Tue76] W. G. Tuel Jr. An analysis of buffer paging in virtual storage systems. *IBM Journal of Research and Development*, 20:518–520, September 1976.

[Yao77] S. B. Yao. Approximating block accesses in database organization. *Comm. of ACM*, 20(4):260–261, April 1977.

Appendix A

Nomenclature

frame : a unit of the main memory. The main memory is divided into equal sized partitions, called frames.

page : a unit of the virtual memory or secondary storage. Virtual memory is divided into equal sized partitions, called (virtual memory) pages. Similarly, the secondary storage is divided into (data) pages. The smallest unit of transfer between the secondary storage and main memory is a page.

N : size of the DBMS buffer pool (in pages).

M : size of the main memory (in frames/pages).

D : size of the DBMS database (in pages).

r : ratio of the cost of an access to the database disk to the cost of an access to the paging device.

DBMS buffer : a buffer which is used solely by the DBMS. It is either mapped into a page in virtual memory or a frame in main memory.

fault : a fault occurs when an access to secondary storage is required.

buffer fault : occurs when an access to the database disk is required. That is, the requested data page is not in the DBMS buffer.

page fault : this fault is caused by an access to the paging disk device.

reference fault : this fault occurs only when the DBMS is running on top of a virtual memory operating system. It is a subset of the page fault. A reference fault occurs when the requested data page is in the DBMS buffer but the virtual memory page to which the DBMS buffer is mapped is not in the main memory.

double fault : this faults also occurs only when the DBMS is running on top of a virtual memory operating system. It is caused by both a buffer fault and a page fault. It can only happen if a buffer fault has already occurred. It happens when the requested data page is not in the DBMS buffer and the virtual memory page, containing the DBMS buffer which has been chosen for replacement, is not in the main memory either.

buffer I/O : sum of buffer faults and reference faults.

system I/O : number of page faults caused by the DBMS program.

total I/O : buffer I/O + system I/O.

I/O cost : number of buffer faults + number of page faults/r.

clustered index : a index on a relation which is sorted on the indexed field. The relative physical location of the tuples of the relation is quite similar to relationship between the values of the indexed field.

unclustered index : a index on a relation which is not sorted on the indexed field. Therefore, the relative physical location of the tuples of the relation is not similar to the relationship between the values of the indexed field.

Appendix B

Pseudo Code

B.1 Main

```
Main :

     Initialize;

     While ( event_list <> empty ) Do
     Begin
         pop ( event );
         Case event_class Of :
              Scheduling         : SCHEDULER;
              I/O                : I/O DRIVER;
              Buffering          : BUFFER MANAGER;
         End Case;
     End While ;
```

B.2 Scheduler

```
Scheduler :

     Case event_method Of :
          Ready, Block, Suspend : Insert into appropriate queue;
                                  Select_Process_To_Run;

          Stop                  : Update simulation statistics;
                                  Return buffers and frames;
                                  Create a new process;
                                  Select_Process_To_Run;
     End Case;

     Select_Process_To_Run :
     Begin
         If no process is running, select a process to run;
         Selected process must have at least one frame
         allocated or the length of the free list is at
         least one;
         If ( process selected ) Then
             Push_Event ( Buffering : Address Reference );
     End; { Select_Process_To_Run }
```

B.3 Buffer Manager

```
Buffer Manager :

     Case event_method Of :
          Address
          Reference : If ( data page in buffers ) Then
                          If ( buffer in I/O ) Then
                              Buffer fault;
                              Major page fault;
                              Double fault;
                              Push_Event ( Scheduling : Block );
                          Else
                              Update buffer statistics;
                              Memory_Manager;
                          End If;
                      Else
                          Buffer fault;
                          Get a buffer from process;
                          If ( no buffer available ) Then
                              Get_From_Others;
                          If ( no buffer available ) Then
                              Push_Event ( Scheduling :
                                                   Suspend );
                          Else
                              Insert buffer into process' buffer
                              table;
                              Update buffer hash table;
                              Memory_Manager;
                          End If;
                      End If;
          Buffer
          Unlock    : For all processes blocked under the given
                      data page,
                          Push_Event ( Scheduling : Ready );
                      Unlock buffer with given data page;
    End Case;

    Get_From_Others :
    Begin
        Get a buffer from suspended processes;
        If ( no buffer available ) Then
           Get a buffer from blocked processes;
    End; { Get_From_Others }
```

```
Memory_Manager :
Begin
    If ( virtual page not in process ) Then
        Page fault;
        Insert into process' page table;
        Try_Allocate_Frame;
    Else
        If ( virtual page in process ) Then
            If ( virtual page_sec_addr == data page ) Then
                Update frame statistics;
                Timeout;
            Else
                Page fault;
                Try_Allocate_Frame;
            End If;
        End If;
     End If;
End; { Memory_Manager }

Timeout :
Begin
    If ( allocated time to process < quantum time ) Then
        Push_Event ( Buffering : Address Reference );
    Else
        Push_Event ( Scheduling : Suspend );
End; { Timeout }
```

```
Try_Allocate_Frame :
Begin
    If ( data page in main memory ) Then
        Minor page fault;
        Allocate frame to process;
        Update frame statistics;
        Timeout;
    Else
        Major page fault;
        Get a free frame;
        If ( no frame available ) Then
            Release buffer;
            Push_Event ( Scheduling : Suspend );
        Else
            If ( process already has a buffer fault ) Then
                Double fault;
            Lock frame, buffer;
            Push_Event ( Scheduling : Block );
        End If;
    End If;
End; { Try_Allocate_Frame }
```

B.4 I/O Driver

```
I/O DRIVER :

    Case event_method Of :
        Read, Write : If ( I_O_list <> empty ) Then
                          Insert ( I_O_list );
                      Else
                          Insert ( I_O_list );
                          Insert_Event ( I/O : Wakeup );
                      End If;

        Wake up     : Delete ( I_O_list );
                      Push_Event ( Buffer Manager :
                                   Buffer Unlock );
                      Update frame statistics;
                      If ( I_O_list <> empty ) Then
                          Insert_Event ( I/O : Wakeup );
    End Case;
```

Appendix C

Performance Data for Simple Selection

In the following, the confidence intervals are given as a single real number. They should be interpreted as the associated mean value ± the confidence interval value.

C.1 Simple Selection without Indices

Hot Set & DBMIN			
Buffer Size	Throughput	I/Os	Response Time
30	3.76476	8.5758	6.19465
50	3.77378	8.55533	6.19465
70	3.78284	8.53484	6.19465
100	3.79651	8.50410	6.19465
150	3.81952	8.45287	6.19465
200	3.84281	8.40164	6.19465

Table C.1: Throughput, I/Os and Response Time of Query Type 1 at 20, 30, 50, 100, 150, 200 Frames

Hot Set & DBMIN			
Buffer Size	Page Faults	Buffer Faults	Double Faults
30	200.000	200.000	200.000
50	200.000	200.000	200.000
70	200.000	200.000	200.000
100	200.000	200.000	200.000
150	200.000	200.000	200.000
200	200.000	191.598	191.598

Table C.2: Faults of Query Type 1 at 20, 30, 50, 100 and 150 Frames

Hot Set & DBMIN			
Buffer Size	Throughput	I/Os	Response Time
30	64.60692	0.174180	0.240021
50	67.3693	0.153688	0.240021
70	70.3785	0.133197	0.240021
100	75.4326	0.102459	0.240021
150	85.6884	0.0512295	0.240021
200	99.1746	0.000000	0.240021

Table C.3: Throughput, I/Os and Response Time of Query Type 1 at 200 Frames

Hot Set & DBMIN			
Buffer Size	Page Faults	Buffer Faults	Double Faults
30	200.000	8.40164	0.000000
50	200.000	8.40164	0.000000
70	200.000	8.40164	0.000000
100	200.000	8.40164	0.000000
150	200.000	8.40164	0.000000
200	200.000	0.00000	0.000000

Table C.4: Faults of Query Type 1 at 200 Frames

C.2 Simple Selection with Non-Clustered Indices

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.331314	0.333538
50	0.326302	0.325305
70	0.322861	0.322861
100	0.322290	0.322290
150	0.322006	0.322006
200	0.321157	0.321157

Table C.5: Throughput of Query Type 2 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	100.746	5.52464	100.052	4.68488
50	102.277	4.90415	102.564	4.33976
70	103.365	4.93012	103.366	4.92954
100	103.548	4.94055	103.548	4.94055
150	103.639	4.92097	103.639	4.92097
200	103.907	4.81423	103.907	4.81423

Table C.6: I/Os of Query Type 2 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	67.1883	3.31074	66.9815	3.26017
50	67.9727	2.81251	68.0313	2.89136
70	68.0038	2.87759	68.0038	2.87759
100	67.8575	2.88586	67.8575	2.88586
150	67.7622	2.96883	67.7622	2.96883
200	67.6788	2.99880	67.6788	2.99880

Table C.7: Response Time of Query Type 2 at 20 Frames

Buffer Size	Hot Set Page Faults	Hot Set Confidence Interval	DBMIN Page Faults	DBMIN Confidence Interval
30	105.112	5.00169	105.326	4.68488
50	103.505	4.22140	103.396	4.15538
70	103.512	4.10947	103.512	4.10843
100	103.272	4.12313	103.272	4.12313
150	103.119	4.26277	103.119	4.26277
200	102.983	4.31112	102.983	4.31112

Table C.8: Page Faults of Query Type 2 at 20 Frames

Buffer Size	Hot Set Buffer Faults	Hot Set Confidence Interval	DBMIN Buffer Faults	DBMIN Confidence Interval
30	99.8976	4.92161	102.857	4.91797
50	88.1187	3.92608	98.4745	4.05135
70	78.1047	3.50701	88.4111	3.81798
100	62.9418	2.76735	72.1302	3.28062
150	40.3615	1.91175	47.2445	2.34984
200	20.8515	0.662974	24.7335	1.05845

Table C.9: Buffer Faults of Query Type 2 at 20 Frames

Buffer Size	Hot Set Double Faults	Hot Set Confidence Interval	DBMIN Double Faults	DBMIN Confidence Interval
30	96.8346	4.79522	99.2525	4.80151
50	88.1101	3.92750	98.4700	4.05184
70	78.1043	3.50631	88.4105	3.81775
100	62.9418	2.76735	72.1299	3.27997
150	40.3615	1.91175	47.2445	2.34984
200	20.8515	0.662974	24.7335	1.05845

Table C.10: Double Faults of Query Type 2 at 20 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.363206	0.361138
50	0.360929	0.361349
70	0.358960	0.358960
100	0.359080	0.359080
150	0.358415	0.358414
200	0.357358	0.357358

Table C.11: Throughput of Query Type 2 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	91.8828	4.37569	92.3863	3.90576
50	92.4640	4.41090	92.3649	4.57859
70	92.9700	4.43161	92.9700	4.43161
100	92.9390	4.43151	92.9390	4.43151
150	93.1132	4.46297	93.1135	4.46354
200	93.3786	4.26973	93.3786	4.26973

Table C.12: I/Os of Query Type 2 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	61.3794	3.17560	61.7062	3.06135
50	61.3981	2.54658	61.3298	2.73011
70	61.2099	2.68539	61.2099	2.68539
100	61.2099	2.68539	61.2099	2.68539
150	61.1506	2.74113	61.1506	2.74113
200	60.9692	2.90880	60.9466	2.90880

Table C.13: Response Time of Query Type 2 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	105.663	4.81323	106.552	5.08641
50	99.0557	4.14269	99.0131	4.15121
70	98.6383	4.03741	98.6383	4.03741
100	98.6383	4.03741	98.6383	4.03741
150	98.5355	4.12388	98.5355	4.12388
200	98.2449	4.43527	98.2449	4.43527

Table C.14: Page Faults of Query Type 2 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	102.208	4.74191	105.467	5.06564
50	86.2351	3.78371	95.6137	4.04364
70	76.7764	3.28033	85.9147	3.56732
100	63.5054	2.80648	72.5288	3.13291
150	41.1504	1.82957	48.5782	2.23976
200	21.7793	0.863711	25.8823	1.19948

Table C.15: Buffer Faults of Query Type 2 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	94.5714	4.60488	95.6658	4.64482
50	86.1580	3.78371	94.2110	4.02194
70	76.7747	3.27916	85.9147	3.56732
100	63.5054	2.80648	72.5288	3.13291
150	41.1504	1.82957	48.5782	2.23976
200	21.7793	0.863711	25.8824	1.19947

Table C.16: Double Faults of Query Type 2 at 30 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.415521	0.402138
50	0.406635	0.406629
70	0.403141	0.404322
100	0.403496	0.403494
150	0.402634	0.402632
200	0.401048	0.401048

Table C.17: Query Type 2 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	80.3275	4.07995	82.9983	4.16228
50	82.1081	4.59616	82.0752	3.96357
70	82.7809	3.91549	82.5943	4.89741
100	82.7063	3.89444	82.7066	3.89390
150	82.8831	3.88623	82.8834	3.88680
200	83.1994	3.66126	83.1994	3.66126

Table C.18: I/Os of Query Type 2 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	53.7386	2.93396	55.4169	2.85313
50	54.5398	2.61752	54.5750	2.27428
70	54.3773	2.22606	54.3377	2.67266
100	54.3781	2.25599	54.3781	2.25599
150	54.2624	2.36469	54.2624	2.36469
200	54.1514	2.46005	54.1514	2.46005

Table C.19: Response Time of Query Type 2 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	105.828	4.84106	106.902	5.09771
50	98.2751	4.22137	98.8423	4.20984
70	97.8877	4.00199	97.9955	4.04134
100	97.8939	3.99835	97.8939	3.99835
150	97.6850	4.19585	97.6850	4.19585
200	97.5007	4.36636	97.5007	4.36636

Table C.20: Page Faults of Query Type 2 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	102.218	4.65038	105.586	5.07517
50	86.0361	3.86712	95.3527	4.04020
70	76.9796	3.50513	86.0463	3.74358
100	63.9521	2.77300	72.4586	3.01928
150	41.4856	1.87599	48.6397	2.20662
200	22.1069	0.921637	26.1542	1.22532

Table C.21: Buffer Faults of Query Type 2 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	84.8261	4.30897	87.4853	4.34130
50	84.6879	3.85175	85.4710	3.63202
70	76.9603	3.50068	85.0582	3.70198
100	63.9515	2.77373	72.4579	3.02004
150	41.4856	1.87599	48.6390	2.20745
200	22.1069	0.921637	26.1542	1.22532

Table C.22: Double Faults of Query Type 2 at 50 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.602888	0.547863
50	0.610393	0.580793
70	0.577198	0.578967
100	0.583833	0.576913
150	0.576805	0.575184
200	0.572392	0.572082

Table C.23: Query Type 2 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	55.3631	2.79493	60.9249	3.12289
50	54.6562	2.20637	57.4645	2.80485
70	57.8206	2.78539	57.6374	2.64792
100	57.1916	2.69517	57.8557	2.92734
150	57.8499	2.58552	58.0242	2.82741
200	58.3008	2.718[illegible]7	58.3323	2.71448

Table C.24: I/Os of Query Type 2 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	36.9356	1.90112	40.6226	2.16124
50	36.0549	1.64231	38.1425	1.61238
70	37.9202	1.79849	37.8866	1.63826
100	37.5660	1.34134	37.8420	1.83584
150	38.0118	1.53722	38.0104	1.61590
200	37.8677	1.68939	37.8382	1.70754

Table C.25: Response Time of Query Type 2 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	106.108	5.18324	106.867	5.02937
50	98.2322	4.16184	99.1116	4.16099
70	96.9784	4.02082	97.9457	4.02765
100	94.7552	2.82756	96.1073	3.98174
150	94.2298	4.03590	94.2246	4.03746
200	93.8115	4.24952	93.8084	4.25147

Table C.26: Page Faults of Query Type 2 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	102.159	5.07249	105.256	5.10525
50	85.5369	3.68918	94.9975	3.96770
70	76.4665	3.26352	85.7001	3.64410
100	62.5476	2.15178	71.2730	3.02214
150	40.9038	1.74279	47.7773	2.00821
200	21.7993	0.896202	25.4875	1.28676

Table C.27: Buffer Faults of Query Type 2 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	61.1550	3.00358	66.3904	3.21702
50	60.0560	2.58521	62.6678	2.53942
70	62.5015	2.80396	62.3247	2.50422
100	61.3858	2.14280	61.9911	2.74985
150	40.8962	1.74279	47.7752	2.27141
200	21.7983	0.895625	25.4871	1.28724

Table C.28: Double Faults of Query Type 2 at 100 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	1.07271	0.691867
50	1.25247	1.04738
70	1.70336	1.26801
100	1.67983	1.43358
150	1.41144	1.46249
200	1.37976	1.44780

Table C.29: Query Type 2 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	31.0861	0.846712	48.2883	3.15476
50	26.6187	0.490695	31.8367	0.838366
70	19.5920	0.937170	26.3316	1.48688
100	19.8560	0.696577	23.2596	0.594199
150	23.6269	0.690120	22.8705	1.52383
200	24.1715	0.773831	23.1084	1.59272

Table C.30: I/Os of Query Type 2 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	20.5760	0.668300	32.1876	1.92995
50	17.2802	0.103070	20.9958	0.649324
70	12.6645	0.463866	17.1460	0.940520
100	12.6777	0.413855	15.0431	0.277619
150	15.0528	0.231818	14.9355	1.06201
200	15.3775	0.491006	14.9591	0.924832

Table C.31: Response Time of Query Type 2 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	105.510	5.09011	106.783	5.35881
50	98.7251	4.21059	99.6526	4.28442
70	92.1419	3.79778	95.3514	4.11710
100	86.5402	4.38498	90.9567	3.51168
150	70.1270	8.69897	75.7868	1.96128
200	69.5548	1.41117	70.4092	3.20746

Table C.32: Page Faults of Query Type 2 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	100.845	4.75365	104.913	5.26576
50	84.3816	3.30658	94.1418	4.00419
70	68.4742	2.38400	78.9419	3.45468
100	48.4459	1.78060	58.4724	1.76844
150	28.0600	0.553767	32.3628	2.03811
200	16.3696	0.112729	17.8550	1.02999

Table C.33: Buffer Faults of Query Type 2 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	37.1001	1.39540	54.2681	3.18926
50	31.9878	0.515100	37.6399	1.38730
70	23.2139	1.05756	31.1227	1.70497
100	22.9893	0.867714	27.3212	0.504788
150	26.8619	0.562557	26.7478	1.68714
200	16.3665	0.113879	17.8550	1.03343

Table C.34: Double Faults of Query Type 2 at 150 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	4.34843	1.29430
50	2.89061	2.43936
70	3.40683	3.25134
100	3.79271	3.75863
150	4.55675	4.28481
200	4.58636	4.47130

Table C.35: Query Type 2 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	7.66735	1.52736	25.9149	2.90804
50	11.5324	0.134027	13.6687	0.321031
70	9.78714	0.234531	10.2554	0.252077
100	8.79545	0.334976	8.87386	0.302152
150	7.32580	0.421196	7.78056	0.129637
200	7.28243	0.456722	7.47213	0.492865

Table C.36: I/Os of Query Type 2 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	4.87867	0.977304	17.1212	1.66307
50	7.58347	0.992964	0.901678	0.266289
70	6.28634	0.112246	6.60473	0.267117
100	5.63781	0.0930908	5.66680	0.167378
150	4.74333	0.113154	4.94281	0.086806
200	4.71804	0.203270	4.84066	0.225999

Table C.37: Response Time of Query Type 2 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	99.8491	8.68791	105.344	4.61003
50	98.7859	4.24630	99.4196	4.32363
70	90.6340	3.30994	91.6510	3.88429
100	83.6344	2.83507	84.8438	3.42685
150	55.2036	1.26997	61.4397	1.59345
200	44.8339	1.34532	50.9462	11.8579

Table C.38: Page Faults of Query Type 2 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	96.9676	20.3134	102.473	4.39687
50	83.9171	3.13165	91.6925	3.95066
70	67.0286	1.77005	71.9596	3.24243
100	46.6041	1.34582	51.0113	2.03211
150	16.8010	0.349671	20.7132	0.647080
200	9.31960	0.467372	13.7311	7.36564

Table C.39: Buffer Faults of Query Type 2 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	10.4867	2.49276	31.6038	2.50191
50	16.2026	0.424583	18.8434	0.665399
70	12.8626	0.226275	13.6834	0.570641
100	11.2160	0.229560	11.2611	0.418333
150	8.81106	0.209464	9.17998	0.210927
200	8.45398	0.421742	8.77142	0.649407

Table C.40: Double Faults of Query Type 2 at 200 Frames

Appendix D

Performance Data for Merge Scan Joins

The following tables contain the data from the runs in which queries are executed with the merge scan join method, using clustered indices on both the outer and inner relations.

The confidence intervals of the following data are given as a single real number. They should be interpreted as the associated mean value ± the confidence interval value.

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.100084	0.0998852
50	0.0992809	0.0976626
70	0.098177	0.0990093
100	0.0989979	0.0994185
150	0.0984262	0.0988742
200	0.098272	0.0975386

Table D.1: Throughput of Query Type 3 at 20 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.100084	0.0998852
50	0.0992809	0.0976626
70	0.098177	0.0990093
100	0.0989979	0.0994185
150	0.0984262	0.0988742
200	0.098272	0.0975386

Table D.1: Throughput of Query Type 3 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	333.235	10.7864	333.841	8.97405
50	335.814	6.48495	341.427	8.67142
70	339.539	3.23592	336.776	8.3124
100	336.909	11.4334	335.509	11.9815
150	338.69	4.18988	337.383	12.8258
200	339.334	9.51374	341.954	11.6285

Table D.2: I/Os of Query Type 3 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	219.171	5.81348	219.992	6.11206
50	221.124	5.76412	222.373	5.79542
70	221.724	5.04078	221.77	6.20359
100	221.6	6.21414	221.55	7.31628
150	221.272	5.81705	221.931	6.77488
200	221.382	4.5389	222.231	4.59192

Table D.3: Response Time of Query Type 3 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	376.726	10.7244	324.236	9.29385
50	313.43	7.87	321.848	8.36142
70	307.301	6.66856	315.25	8.80517
100	307.134	8.19284	309.103	9.86438
150	306.62	7.83065	307.403	9.24824
200	306.809	5.61068	307.901	5.92158

Table D.4: Page Faults of Query Type 3 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	204.365	5.83808	155.147	3.88911
50	100.815	2.83925	99.765	3.22353
70	83.4269	2.0497	89.7872	2.9645
100	81.9743	2.30307	84.1612	3.19858
150	80.8714	2.38498	81.965	3.0898
200	80.3368	1.9279	81.253	1.85699

Table D.5: Buffer Faults of Query Type 3 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	137.541	3.13773	148.573	3.6492
50	98.9716	2.76739	99.542	3.20358
70	83.4266	2.05029	89.7452	2.96555
100	81.9743	2.30307	84.155	3.19769
150	80.8714	2.38498	81.965	3.0898
200	80.3368	1.9279	81.253	1.85699

Table D.6: Double Faults of Query Type 3 at 20 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.135958	0.117954
50	0.121949	0.115881
70	0.115071	0.11535
100	0.115739	0.115839
150	0.115743	0.114828
200	0.115297	0.115649

Table D.7: Throughput of Query Type 3 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	245.176	1.01003	282.624	3.94673
50	273.346	2.11061	287.668	3.11282
70	289.689	2.70306	289.003	3.89442
100	288.041	4.46175	287.79	4.33616
150	288.066	6.39308	290.338	5.13618
200	289.141	4.18525	288.286	5.71928

Table D.8: I/Os of Query Type 3 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	160.033	2.56692	184.778	3.94727
50	178.291	3.23942	187.794	3.95221
70	189.132	3.30828	188.438	4.48485
100	187.955	4.88297	188.16	4.75481
150	188.064	5.16263	188.525	3.88475
200	187.832	3.09648	187.755	4.40818

Table D.9: Response Time of Query Type 3 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	434.278	9.61584	278.733	6.26446
50	318.97	10.1023	266.065	5.57194
70	261.991	4.59273	260.224	5.60885
100	258.265	6.38301	258.98	6.23637
150	258.105	6.86742	259.108	5.35907
200	258.19	4.31678	258.414	5.9827

Table D.10: Page Faults of Query Type 3 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	374.461	8.68247	513.934	44.437
50	155.931	7.53592	90.9211	2.40654
70	68.2155	1.94659	69.8282	2.02637
100	63.1598	1.97913	66.5888	2.25455
150	61.5825	2.72774	64.1884	2.11916
200	61.0244	1.63965	62.1593	2.39419

Table D.11: Buffer Faults of Query Type 3 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	155.302	2.30277	188.493	3.24882
50	82.1045	1.78733	87.6612	2.13662
70	65.7275	1.83734	69.4904	1.97211
100	63.1595	1.97928	66.4839	2.23436
150	61.5825	2.72774	64.1836	2.12183
200	61.0244	1.63965	62.1593	2.39419

Table D.12: Double Faults of Query Type 3 at 30 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.323475	0.544359
50	0.508688	0.484935
70	0.26193	0.355929
100	0.24883	0.2986
150	0.249082	0.258504
200	0.250095	0.246405

Table D.13: Throughput of Query Type 3 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	103.182	5.18305	61.2377	0.652854
50	65.5622	2.06335	68.7387	0.371618
70	127.361	4.91806	93.6831	2.36389
100	134.018	3.83216	111.634	0.592633
150	133.884	3.86881	128.95	0.802547
200	133.377	4.87214	135.37	4.8237

Table D.14: I/Os of Query Type 3 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	61.547	3.65482	38.6447	1.09353
50	42.6267	1.78771	43.4977	1.16865
70	81.931	3.22875	60.393	2.29506
100	86.6659	3.35258	71.6322	2.3836
150	85.4634	2.77749	83.3695	2.4493
200	85.3203	2.01386	85.3714	2.25782

Table D.15: Response Time of Query Type 3 at 50 Frames

Buffer Size	Hot Set Page Faults	Hot Set Confidence Interval	DBMIN Page Faults	DBMIN Confidence Interval
30	257.792	15.6346	67.0656	2.0229
50	90.5253	4.41633	63.8169	1.74454
70	115.406	4.85794	85.0837	3.17563
100	120.695	4.81971	100.023	3.52668
150	119.142	3.9738	116.102	3.4795
200	119.03	2.95167	119.043	3.33972

Table D.16: Page Faults of Query Type 3 at 50 Frames

Buffer Size	Hot Set Buffer Faults	Hot Set Confidence Interval	DBMIN Buffer Faults	DBMIN Confidence Interval
30	234.79	15.8293	93.0874	36.5943
50	67.1031	3.1987	44.2859	1.02131
70	36.9334	1.48833	38.0458	0.965037
100	34.265	0.857095	36.1192	1.03662
150	32.7096	0.762236	34.5411	1.08899
200	31.5996	0.795246	32.8333	0.915875

Table D.17: Buffer Faults of Query Type 3 at 50 Frames

Buffer Size	Hot Set Double Double Faults	Hot Set Confidence Interval	DBMIN Double Double Faults	DBMIN Confidence Interval
30	57.3262	5.27961	46.0382	1.18723
50	34.6421	1.18239	40.0912	0.897018
70	35.7538	1.23687	37.0557	1.00478
100	34.2623	0.855082	35.807	1.02284
150	32.7096	0.762236	34.5202	1.08976
200	31.5996	0.795246	32.8333	0.915875

Table D.18: Double Faults of Query Type 3 at 50 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.705992	0.947042
50	1.13201	1.21062
70	1.34027	1.34447
100	1.33113	1.34309
150	1.32586	1.32195
200	1.32453	1.32005

Table D.19: Throughput of Query Type 3 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	47.2158	0.291555	35.2066	0.789261
50	29.474	1.24441	27.5379	0.449722
70	24.8772	0.564817	24.801	0.623253
100	25.0501	0.64684	24.826	0.605829
150	25.1531	0.763991	25.2247	0.67393
200	25.1774	0.738965	25.2596	0.627114

Table D.20: I/Os of Query Type 3 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	30.7048	0.100866	22.4235	0.525184
50	19.2654	0.342199	18.1558	0.489813
70	16.4718	0.389796	16.4265	0.410696
100	16.5803	0.343888	16.3882	0.344244
150	16.5315	0.280668	16.5415	0.286968
200	16.5099	0.329203	16.5287	0.366902

Table D.21: Response Time of Query Type 3 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	276.794	1.54386	50.4293	1.02498
50	63.7237	1.41926	31.5765	0.652783
70	23.0451	0.3585	25.0339	0.515784
100	22.8174	0.352549	23.0026	0.320924
150	22.5974	0.260341	22.5887	0.256225
200	22.5612	0.319528	22.5553	0.328839

Table D.22: Page Faults of Query Type 3 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	256.013	1.70219	49.7401	1.10886
50	58.86	1.35844	30.8125	0.706817
70	21.2365	0.365859	24.0086	0.535764
100	20.2613	0.354335	21.3203	0.36572
150	18.9476	0.260682	19.3054	0.294015
200	17.9621	0.305162	18.246	0.341323

Table D.23: Buffer Faults of Query Type 3 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Double Double Faults	Confidence Interval	Double Double Faults	Confidence Interval
30	19.2958	0.37753	27.9621	0.624191
50	19.8265	0.333425	22.015	0.532611
70	19.6917	0.374949	19.6453	0.433025
100	19.8942	0.354399	19.6832	0.344599
150	18.9452	0.261964	19.2877	0.292017
200	17.9618	0.304496	18.2436	0.3403

Table D.24: Double Faults of Query Type 3 at 100 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.725464	1.19454
50	1.22868	1.23468
70	1.41659	1.42409
100	1.42167	1.43111
150	1.41715	1.41785
200	1.41343	1.41343

Table D.25: Throughput of Query Type 3 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	45.9822	1.73882	27.9078	0.410216
50	27.166	1.39662	27.0441	1.57185
70	23.5448	0.797135	23.4149	0.604392
100	23.4635	0.874508	23.3095	0.88637
150	23.5409	0.93631	23.5271	0.884667
200	23.5991	0.844639	23.6002	0.874555

Table D.26: I/Os of Query Type 3 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	29.5274	0.914098	18.218	0.296431
50	17.8128	0.42427	17.3007	0.49735
70	15.5985	0.372884	15.5373	0.29911
100	15.4951	0.356924	15.3842	0.363514
150	15.3914	0.316059	15.4129	0.288101
200	15.3905	0.324987	15.3956	0.34402

Table D.27: Response Time of Query Type 3 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	279.078	10.1995	48.4013	0.867329
50	62.3743	0.870065	32.238	0.662998
70	23.0301	0.361489	25.0116	0.29363
100	22.5447	0.305597	22.8645	0.300652
150	22.1135	0.269719	22.1169	0.243043
200	22.0035	0.307009	22.0073	0.303811

Table D.28: Page Faults of Query Type 3 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	258.727	9.42676	47.5987	0.925808
50	57.6889	0.828686	31.3975	0.680435
70	21.0848	0.394994	23.9996	0.328861
100	19.9153	0.355452	21.3045	0.376777
150	18.4187	0.343097	18.8694	0.312608
200	17.6286	0.356089	17.8932	0.412481

Table D.29: Buffer Faults of Query Type 3 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	17.987	0.380339	22.1107	0.302525
50	18.1113	0.442389	20.6783	0.568813
70	18.4262	0.374089	18.2751	0.340795
100	18.3381	0.374839	18.1761	0.398055
150	18.1156	0.330189	18.1807	0.310365
200	17.6234	0.354102	17.8536	0.415434

Table D.30: Double Faults of Query Type 3 at 150 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.758261	1.47031
50	1.29998	1.4135
70	1.48768	1.49663
100	1.4874	1.50822
150	1.48226	1.47366
200	1.47851	1.47395

Table D.31: Throughput of Query Type 3 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	43.9614	0.310739	22.6841	0.756806
50	25.6496	0.639822	23.5857	0.403833
70	22.4116	0.474109	22.2827	0.666889
100	22.4275	0.85677	22.1207	0.911165
150	22.5019	0.771564	22.6368	0.865945
200	22.5604	0.808453	22.6313	0.843263

Table D.32: I/Os of Query Type 3 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	28.1736	0.414405	14.9858	0.355967
50	17.0364	0.398769	15.5615	0.60792
70	14.8905	0.32081	14.7156	0.363028
100	14.7678	0.345652	14.5902	0.37065
150	14.6679	0.197698	14.7631	0.234997
200	14.6874	0.267322	14.7212	0.289922

Table D.33: Response Time of Query Type 3 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	276.828	6.87779	46.8074	1.17794
50	62.2025	1.26043	33.5994	1.12995
70	23.0352	0.347387	25.5291	0.381113
100	22.5433	0.291862	22.9404	0.346776
150	22.0968	0.240412	22.1079	0.225842
200	21.9562	0.289472	21.9641	0.287718

Table D.34: Page Faults of Query Type 3 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	257.296	6.73089	45.9286	1.18405
50	57.7264	1.05753	32.7148	1.12121
70	21.0902	0.408096	24.4939	0.401953
100	19.8265	0.35937	21.2901	0.417453
150	18.2403	0.286048	18.7596	0.346775
200	17.4027	0.302745	17.6716	0.390479

Table D.35: Buffer Faults of Query Type 3 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	16.8917	0.386613	17.5926	0.410916
50	17.2835	0.450439	18.3299	0.71516
70	17.4614	0.391962	17.149	0.383559
100	17.4274	0.388796	17.1007	0.395081
150	17.1633	0.259191	17.3476	0.267754
200	16.9975	0.296143	17.0859	0.354104

Table D.36: Double Faults of Query Type 3 at 200 Frames

Appendix E

Performace Data on Nested Loop Joins

In the following, the confidence intervals are given as a single real number. They should be interpreted as the associated mean value ± the confidence interval value.

E.1 Nested Loop Join with Clustered Inner and Outer Indices

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.0868645	0.0864047
50	0.0864938	0.0863008
70	0.0863913	0.0861178
100	0.0861421	0.0860193
150	0.0858412	0.0857916
200	0.0856706	0.0856706

Table E.1: Throughput of Query Type 4 at 20 Frames

Buffer Size	Hot Set I/Os	Hot Set Confidence Interval	DBMIN I/Os	DBMIN Confidence Interval
30	384.273	20.0856	386.34	20.5931
50	385.873	19.2444	386.761	19.7776
70	386.325	19.141	387.645	21.0125
100	387.456	19.4759	388.055	20.4096
150	388.934	21.7931	389.144	21.5313
200	389.658	20.9218	389.658	20.9218

Table E.2: I/Os of Query Type 4 at 20 Frames

Buffer Size	Hot Set Response Time	Hot Set Confidence Interval	DBMIN Response Time	DBMIN Confidence Interval
30	255.092	12.308	256.824	12.3874
50	256.216	12.2774	256.911	12.2626
70	255.966	11.9383	256.491	12.325
100	256.292	11.5421	256.623	11.8068
150	255.787	10.3103	255.797	10.054
200	255.793	10.3244	255.793	10.3244

Table E.3: Response Time of Query Type 4 at 20 Frames

Buffer Size	Hot Set Page Faults	Hot Set Confidence Interval	DBMIN Page Faults	DBMIN Confidence Interval
30	484.494	22.725	506.371	23.9779
50	470.049	21.8997	494.486	22.898
70	468.013	21.5159	476.421	22.1318
100	468.059	20.3897	469.216	20.3426
150	466.814	17.6954	466.917	17.8164
200	466.802	17.7348	466.802	17.7348

Table E.4: Page Faults of Query Type 4 at 20 Frames

Buffer Size	Hot Set Buffer Faults	Hot Set Confidence Interval	DBMIN Buffer Faults	DBMIN Confidence Interval
30	427.825	20.9243	465.487	22.2292
50	318.823	18.5515	369.123	18.2749
70	195.428	10.7355	238.454	13.3577
100	124.948	4.98685	127.767	4.13611
150	108.808	3.512	109.327	3.98258
200	107.367	3.56087	107.266	3.46458

Table E.5: Buffer Faults of Query Type 4 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	409.164	20.2556	425.616	20.5396
50	316.025	18.2458	341.876	17.0764
70	195.038	10.6751	228.837	12.8097
100	124.937	4.9863	126.757	4.19982
150	108.808	3.512	109.305	3.97594
200	107.367	3.56087	107.266	3.46458

Table E.6: Double Faults of Query Type 4 at 20 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.110543	0.103829
50	0.104215	0.103361
70	0.103669	0.102344
100	0.103179	0.102704
150	0.102531	0.102242
200	0.102511	0.102511

Table E.7: Throughput of Query Type 4 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	301.954	15.5319	321.506	17.1505
50	320.422	19.0288	322.993	17.8373
70	322.146	19.6876	326.246	18.7517
100	323.608	18.6473	325.068	18.0929
150	325.643	18.6071	326.613	19.4956
200	325.699	18.4858	325.699	18.4858

Table E.8: I/Os of Query Type 4 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	198.88	10.376	213.019	11.375
50	212.627	12.0961	214.435	11.123
70	213.434	11.8207	216.583	10.9525
100	213.432	10.566	214.809	10.326
150	213.664	9.64073	214.046	9.97356
200	213.735	9.55385	213.735	9.55385

Table E.9: Response Time of Query Type 4 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	460.505	23.2606	496.241	24.1255
50	430.919	22.9532	485.189	23.6644
70	414.903	20.6275	454.968	22.9264
100	412.234	18.436	416.117	17.99
150	411.474	16.7651	411.998	16.9749
200	411.611	16.6182	411.611	16.6182

Table E.10: Page Faults of Query Type 4 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	416.813	22.2378	440.637	20.0939
50	319.273	20.255	362.438	16.4084
70	204.031	15.3437	254.004	13.5765
100	124.138	5.73227	127.958	6.12477
150	106.516	4.95992	106.305	4.68906
200	105.142	4.88033	105.037	4.86267

Table E.11: Buffer Faults of Query Type 4 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	356.304	17.6406	377.737	19.3591
50	304.932	19.1479	308.829	15.2273
70	200.955	15.2337	225.007	12.5179
100	124.055	5.6859	124.838	5.85427
150	106.516	4.96043	106.244	4.70692
200	105.141	4.88082	105.037	4.86267

Table E.12: Double Faults of Query Type 4 at 30 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.1851	0.16887
50	0.448435	0.174368
70	0.245885	0.216458
100	0.203837	0.177685
150	0.174702	0.171286
200	0.172974	0.177943

Table E.13: Throughput of Query Type 4 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	180.654	13.8986	197.791	12.2899
50	74.351	1.61325	191.37	8.59569
70	135.588	2.4854	154.335	9.95296
100	163.683	6.92676	187.769	7.77389
150	191.895	13.4352	194.904	10.5253
200	193.286	11.7867	187.612	10.0796

Table E.14: I/Os of Query Type 4 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	118.177	8.70667	127.06	9.38672
50	48.3306	1.60139	122.536	5.3741
70	86.9036	2.14122	98.969	5.12557
100	102.086	8.73013	118.728	3.77304
150	121.425	10.6373	121.861	9.68143
200	120.899	7.67652	118.536	7.11638

Table E.15: Response Time of Query Type 4 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	375.96	17.3041	385.656	19.5514
50	148.572	4.17268	340.67	15.8513
70	197.286	5.52787	248.921	10.7407
100	215.934	17.8717	239.293	6.33414
150	242.035	18.4846	243.233	20.2057
200	239.73	12.945	236.112	14.3602

Table E.16: Page Faults of Query Type 4 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	366.386	16.7482	347.92	17.102
50	128.795	4.8348	269.442	11.8025
70	110.439	2.92033	159.537	5.73385
100	69.6041	1.1351	77.8173	1.0748
150	61.7469	2.3455	64.7501	6.6618
200	59.3089	0.724438	60.513	4.50661

Table E.17: Buffer Faults of Query Type 4 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	182.044	13.0165	224.795	17.0825
50	68.7737	1.66634	176.207	8.53187
70	90.8786	3.8243	107.885	5.39147
100	67.472	1.50667	73.1675	1.02503
150	61.7336	2.34359	64.3006	6.1973
200	59.3026	0.722403	60.5099	4.50589

Table E.18: Double Faults of Query Type 4 at 50 Frames

Buffer Size	Throughput Hot Set	Throughput DBMIN
30	2.40546	1.55602
50	2.23413	2.09843
70	2.33642	2.65138
100	2.83762	2.90745
150	2.71449	2.70034
200	2.64287	2.68187

Table E.19: Throughput of Query Type 4 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	13.8837	0.69933	21.4576	1.21351
50	14.9333	0.612449	15.9039	0.764833
70	14.2748	0.46996	12.5874	0.604699
100	11.7569	0.477201	11.4808	0.597141
150	12.3716	0.833745	12.3729	0.825712
200	12.6193	0.0852994	12.4467	0.647465

Table E.20: I/Os of Query Type 4 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	9.14347	0.442637	13.9372	0.758034
50	9.7351	0.489234	10.3331	0.537573
70	9.64193	0.257967	8.39661	0.352166
100	7.76816	0.155756	7.56522	0.242048
150	7.96717	0.376375	8.11179	0.328396
200	8.23454	0.669828	8.03382	0.238047

Table E.21: Response Time of Query Type 4 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	358.021	10.5232	267.039	11.2753
50	181.228	8.59477	178.544	9.69612
70	56.0374	0.488833	81.9829	8.36847
100	18.7825	0.455341	25.6104	0.440246
150	17.0574	0.488459	17.3353	0.461684
200	17.2656	0.0660994	17.1997	0.377511

Table E.22: Page Faults of Query Type 4 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	353.181	10.5606	257.289	9.94424
50	175.148	8.22669	171.101	9.36864
70	51.1543	0.434968	76.5201	8.26952
100	14.2195	0.413403	20.7943	0.471808
150	10.5386	0.365944	10.6471	0.499624
200	9.65399	0.301931	9.5839	0.354056

Table E.23: Buffer Faults of Query Type 4 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	14.4552	0.633343	23.7162	1.33863
50	15.2829	0.650337	16.7829	0.857361
70	15.1	0.402717	13.2283	0.516223
100	11.9888	0.336367	11.44	0.363089
150	10.4561	0.362964	10.3629	0.448762
200	9.63402	0.307071	9.49537	0.328974

Table E.24: Double Faults of Query Type 4 at 100 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	3.45117	2.98432
50	3.46585	2.89315
70	3.43447	3.2001
100	3.35318	3.38718
150	3.41221	3.44589
200	3.4109	3.40891

Table E.25: Throughput of Query Type 4 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	9.67008	0.465292	11.195	0.752626
50	9.6346	0.563059	11.5445	0.725789
70	9.72053	0.534754	10.4277	0.477349
100	9.95243	0.475368	9.85033	0.423846
150	9.77943	0.450279	9.68583	0.486659
200	9.78376	0.463871	9.78932	0.459967

Table E.26: I/Os of Query Type 4 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	6.37697	0.233649	7.28144	0.532922
50	6.34209	0.303094	7.62059	0.491476
70	6.39231	0.277231	6.85689	0.305938
100	6.49469	0.184362	6.45204	0.158482
150	6.33818	0.233059	6.28152	0.263028
200	6.34492	0.266641	6.34926	0.253598

Table E.27: Response Time of Query Type 4 at 150 Frames

Buffer Size	Hot Set Page Faults	Hot Set Confidence Interval	DBMIN Page Faults	DBMIN Confidence Interval
30	363.098	12.7718	257.797	13.0584
50	194.559	10.7615	177.59	6.6798
70	98.6496	10.7079	85.9196	0.610454
100	24.2002	1.73448	26.3924	0.814281
150	15.6539	0.330352	15.7784	0.381361
200	15.5525	0.310787	15.6466	0.338548

Table E.28: Page Faults of Query Type 4 at 150 Frames

Buffer Size	Hot Set Buffer Faults	Hot Set Confidence Interval	DBMIN Buffer Faults	DBMIN Confidence Interval
30	358.1	13.1665	249.381	13.3408
50	188.929	11.0565	170.24	6.84422
70	93.8517	10.6426	80.0701	0.820592
100	19.5753	1.76041	21.4466	0.713969
150	10.3359	0.359118	10.2842	0.411628
200	9.52753	0.340911	9.39672	0.368506

Table E.29: Buffer Faults of Query Type 4 at 150 Frames

Buffer Size	Hot Set Double Double Faults	Hot Set Confidence Interval	DBMIN Double Double Faults	DBMIN Confidence Interval
30	10.3009	0.37113	11.8689	0.71956
50	10.245	0.437953	12.4476	0.768481
70	10.3029	0.400656	11.0319	0.462491
100	10.3966	0.286742	10.2924	0.257411
150	9.97775	0.37903	9.66089	0.337443
200	9.4959	0.34434	9.25417	0.34032

Table E.30: Double Faults of Query Type 4 at 150 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	3.99141	3.97005
50	3.96522	3.92402
70	3.99882	3.94262
100	3.90995	3.9277
150	3.94142	3.98749
200	3.93382	3.92042

Table E.31: Throughput of Query Type 4 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	8.35827	0.333095	8.40369	0.351465
50	8.41765	0.424889	8.50092	0.321706
70	8.34513	0.384691	8.46831	0.478754
100	8.53711	0.441688	8.4906	0.252574
150	8.46279	0.302032	8.36532	0.308962
200	8.47991	0.31984	8.50973	0.341428

Table E.32: I/Os of Query Type 4 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	5.46722	0.152166	5.52782	0.175877
50	5.51276	0.229916	5.62609	0.197278
70	5.44451	0.162398	5.56528	0.249288
100	5.5027	0.178446	5.50077	0.101853
150	5.4444	0.188209	5.41556	0.155541
200	5.46679	0.15246	5.4719	0.169087

Table E.33: Response Time of Query Type 4 at 200 Frames

Buffer Size	Hot Set Page Faults	Hot Set Confidence Interval	DBMIN Page Faults	DBMIN Confidence Interval
30	358.722	12.1999	256.166	12.8307
50	198.244	10.5455	182.387	11.3287
70	98.2058	6.29702	88.2022	0.717615
100	27.8868	2.14139	27.204	0.824962
150	15.6839	0.321595	15.7903	0.360641
200	15.5677	0.301621	15.6415	0.328178

Table E.34: Page Faults of Query Type 4 at 200 Frames

Buffer Size	Hot Set Buffer Faults	Hot Set Confidence Interval	DBMIN Buffer Faults	DBMIN Confidence Interval
30	353.439	11[illegible]42	247.602	12.3216
50	192.4	10.4[illegible]3	175.654	10.973
70	93.0716	6.36511	82.3558	0.734269
100	23.0393	2.10573	22.0838	0.712789
150	9.98233	0.362251	10.0469	0.349014
200	9.0965	0.317799	9.05374	0.37545

Table E.35: Buffer Faults of Query Type 4 at 200 Frames

Buffer Size	Hot Set Double Faults	Hot Set Confidence Interval	DBMIN Double Faults	DBMIN Confidence Interval
30	9.13559	0.303143	9.19126	0.34497
50	9.18619	0.404827	9.26278	0.315665
70	9.07235	0.322734	9.17922	0.400194
100	9.13284	0.278175	9.02619	0.185302
150	8.94713	0.324084	8.80335	0.321034
200	8.63276	0.316255	8.52179	0.357153

Table E.36: Double Faults of Query Type 4 at 200 Frames

E.2 Nested Loop Join with Non-Clustered Outer Index and Clustered Inner Index

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.086633	0.0861974
50	0.0861527	0.086006
70	0.086013	0.0857882
100	0.0857829	0.0858999
150	0.0853953	0.0854335
200	0.0853496	0.0854138

Table E.37: Throughput of Query Type 5 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	385.217	18.4687	387.256	20.4026
50	387.457	20.4301	388.037	18.8601
70	388.036	19.4582	389.082	20.0913
100	389.09	19.7896	388.549	19.5512
150	390.922	21.1527	390.745	21.0879
200	391.111	20.776	390.773	19.9163

Table E.38: I/Os of Query Type 5 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	256.09	11.7559	257.272	12.1672
50	256.837	12.0721	257.551	11.9446
70	256.784	11.3479	257.475	11.6929
100	256.722	10.86	256.938	10.6238
150	256.351	9.67009	256.505	9.58124
200	256.225	9.98061	256.292	9.71462

Table E.39: Response Time of Query Type 5 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	486.114	23.0297	503.445	23.9834
50	470.567	21.6696	492.386	22.9144
70	468.705	21.0468	477.994	21.5964
100	468.617	19.8572	469.924	20.1053
150	467.458	17.9027	467.915	17.8426
200	467.366	18.1716	467.293	18.0563

Table E.40: Page Faults of Query Type 5 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	436.666	20.463	465.88	22.1157
50	345.792	16.0277	392.718	18.0686
70	264.814	12.3638	308.524	14.1164
100	207.175	8.66631	218.185	22.502
150	166.01	6.2511	186.037	8.77306
200	139.674	5.39045	151.424	15.6992

Table E.41: Buffer Faults of Query Type 5 at 20 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	417.216	19.7353	428.198	20.4103
50	342.944	16.0122	366.971	17.1651
70	264.24	12.289	297.609	13.8861
100	207.03	8.66114	216.424	21.8278
150	165.968	6.25298	185.719	8.57199
200	139.656	5.39129	151.424	15.6985

Table E.42: Double Faults of Query Type 5 at 20 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.103187	0.101234
50	0.101859	0.100422
70	0.101942	0.100032
100	0.101808	0.100875
150	0.101185	0.101064
200	0.101204	0.101175

Table E.43: Throughput of Query Type 5 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	323.395	15.0617	329.666	16.049
50	327.619	15.3827	332.336	16.2692
70	327.402	16.4029	333.668	17.0084
100	327.85	16.8068	330.877	16.836
150	329.861	16.746	330.267	16.9912
200	329.803	16.7742	329.915	17.1353

Table E.44: I/Os of Query Type 5 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	214.692	9.85804	219.209	10.4186
50	217.541	10.2801	220.79	10.5767
70	216.852	10.0152	221.184	10.3656
100	216.434	8.53034	218.313	8.80024
150	216.484	8.47356	216.685	8.63855
200	216.439	8.71471	216.522	8.57227

Table E.45: Response Time of Query Type 5 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	485.321	22.1513	504.556	23.3918
50	440.194	21.2834	494.479	23.32
70	424.128	19.4837	465.004	21.4134
100	420.675	16.9442	430.794	17.5585
150	419.719	16.5496	421.013	16.406
200	419.553	16.4209	419.596	16.3986

Table E.46: Page Faults of Query Type 5 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	438.056	19.2513	451.169	20.1663
50	350.141	17.6498	392.627	16.7531
70	273.836	11.9401	319.853	13.8616
100	214.446	9.41215	229.395	10.0169
150	169.325	6.46643	184.865	6.94881
200	142.288	5.06553	144.621	5.72702

Table E.47: Buffer Faults of Query Type 5 at 30 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	391.642	18.3307	391.167	18.1034
50	339.411	16.904	342.352	15.5292
70	270.815	11.7693	290.124	12.7818
100	213.621	9.21809	222.311	9.53223
150	169.166	6.47403	183.785	6.73531
200	142.22	5.07994	144.62	5.72673

Table E.48: Double Faults of Query Type 5 at 30 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.125085	0.111192
50	0.145145	0.112637
70	0.12275	0.117104
100	0.118765	0.120344
150	0.117801	0.118368
200	0.117856	0.118269

Table E.49: Throughput of Query Type 5 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	266.742	11.5804	300.073	13.1178
50	229.905	10.4813	296.209	12.5881
70	271.802	11.3653	284.908	12.0305
100	280.911	11.5396	277.246	11.8773
150	283.326	14.1807	281.915	13.0227
200	283.117	12.5846	282.169	13.3876

Table E.50: I/Os of Query Type 5 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	176.462	7.8167	199.168	8.39915
50	151.272	8.57995	195.983	7.85067
70	179.293	8.71426	187.443	8.47745
100	183.799	7.18238	181.074	6.80752
150	184.125	7.0577	183.226	6.46385
200	183.556	6.81448	183.06	7.31702

Table E.51: Response Time of Query Type 5 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	423.382	16.6787	481.988	21.1686
50	302.667	16.1161	462.578	21.3516
70	346.332	16.5112	416.729	17.7582
100	357.151	12.8965	366.16	13.9125
150	358.34	12.672	361.085	12.7284
200	359.138	12.6985	358.619	13.8037

Table E.52: Page Faults of Query Type 5 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	398.793	15.9407	439.12	19.2364
50	263.643	13.8992	377.983	16.0678
70	236.48	12.2179	302.91	12.2906
100	189.96	3.78718	203.995	8.28565
150	144.562	4.50676	161.115	2.73264
200	118.225	4.28782	119.719	4.19271

Table E.53: Buffer Faults of Query Type 5 at 50 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	278.51	11.7598	346.108	14.0083
50	227.639	13.5696	297.839	12.7462
70	226.707	12.3576	251.341	10.1608
100	186.817	3.97685	191.151	7.11054
150	143.789	4.52671	159.028	2.78649
200	117.91	4.24087	119.708	4.18915

Table E.54: Double Faults of Query Type 5 at 50 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.291593	0.239643
50	0.244238	0.229159
70	0.307508	0.246994
100	0.337222	0.321062
150	0.339842	0.332276
200	0.327091	0.318933

Table E.55: Throughput of Query Type 5 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	114.472	5.95103	139.232	6.0915
50	136.604	5.79985	145.603	6.38629
70	108.484	4.26238	135.215	8.23532
100	98.9113	3.52845	104.043	6.66837
150	98.2148	5.01143	100.541	6.66338
200	102.13	6.69707	104.762	7.16599

Table E.56: I/Os of Query Type 5 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	76.276	3.33507	93.0463	4.70327
50	90.3869	3.30509	96.3983	4.85184
70	72.932	2.97975	89.9148	5.15789
100	66.4372	2.20267	68.6013	3.25721
150	64.6645	2.74792	65.2924	3.19547
200	66.6692	3.07051	67.9001	3.37292

Table E.57: Response Time of Query Type 5 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	403.291	18.7858	430.653	20.2991
50	260.507	8.90074	380.314	17.3654
70	177.731	8.96793	298.379	12.3455
100	141.995	4.98519	186.973	6.96664
150	127.711	5.29924	140.976	9.01045
200	127.914	5.925	128.042	5.61999

Table E.58: Page Faults of Query Type 5 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	396.327	18.5922	406.757	19.8515
50	240.983	8.01795	347.994	15.3013
70	160.162	7.80397	269.667	10.8358
100	115.829	3.79808	161.932	6.27345
150	75.0753	2.99056	95.1964	6.4039
200	49.6059	2.24948	52.943	2.20549

Table E.59: Buffer Faults of Query Type 5 at 100 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	117.482	5.28199	170.472	8.12177
50	140.182	5.6488	169.095	7.60408
70	113.17	5.30839	152.916	7.58004
100	98.5063	3.48633	107.294	4.72831
150	69.9711	2.62809	76.7289	3.26595
200	47.4594	2.08873	52.7907	2.19462

Table E.60: Double Faults of Query Type 5 at 100 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	0.579723	0.344085
50	0.531965	0.319664
70	0.467189	0.350113
100	0.530711	0.465177
150	0.554896	0.550772
200	0.574034	0.578299

Table E.61: Throughput of Query Type 5 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	57.5584	2.57657	96.9781	4.41865
50	62.7589	3.48767	104.354	3.95918
70	71.5088	4.74973	95.372	5.49737
100	62.8995	3.35424	71.7988	4.46722
150	60.133	2.67704	60.5911	2.88743
200	58.1187	2.37363	57.7138	2.84905

Table E.62: I/Os of Query Type 5 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	38.4708	1.77127	64.8442	3.10912
50	41.4273	1.80312	69.0258	3.38521
70	47.9626	2.86633	62.7465	3.90323
100	42.1089	1.76119	47.4844	3.03691
150	40.0406	1.51955	39.3112	1.67902
200	38.2099	1.49859	37.8402	1.79023

Table E.63: Response Time of Query Type 5 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	405.734	17.9628	404.534	18.4862
50	278.02	10.9787	358.82	16.4185
70	179.12	8.11317	277.071	12.4629
100	140.907	7.07036	177.572	11.2611
150	117.448	4.83482	124.798	6.18532
200	107.746	4.61849	102.269	4.40715

Table E.64: Page Faults of Query Type 5 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	400.941	17.8122	384.595	16.7661
50	266.557	10.8417	330.638	14.702
70	160.994	7.8929	251.529	11.0368
100	114.527	5.75664	155.637	10.3045
150	74.367	3.16857	90.228	4.49302
200	47.7174	1.76756	50.7622	1.90789

Table E.65: Buffer Faults of Query Type 5 at 150 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	63.2524	2.86221	119.438	5.8374
50	66.9833	3.03433	122.285	5.52341
70	76.4631	4.52409	108.746	6.40926
100	67.5039	2.86637	78.7833	4.46287
150	57.6444	2.48426	55.4272	2.14208
200	40.8453	1.40391	48.3241	1.91029

Table E.66: Double Faults of Query Type 5 at 150 Frames

Buffer Size	Throughput	
	Hot Set	DBMIN
30	1.02982	0.891554
50	1.0389	0.522297
70	1.06689	0.543637
100	1.02346	0.76573
150	1.20151	1.04918
200	1.23917	1.10131

Table E.67: Throughput of Query Type 5 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	I/Os	Confidence Interval	I/Os	Confidence Interval
30	32.3941	1.28664	37.4211	1.55207
50	32.1113	1.28007	63.86	2.19438
70	31.2923	1.73066	61.3794	2.76765
100	32.5988	1.34708	43.5748	1.92544
150	27.7544	0.789317	31.7895	1.06194
200	26.9103	0.747721	30.2795	0.860262

Table E.68: I/Os of Query Type 5 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Response Time	Confidence Interval	Response Time	Confidence Interval
30	21.6639	0.997389	24.9378	1.09862
50	21.3335	1.11316	42.14	1.23371
70	20.8039	0.965157	39.8519	2.07448
100	21.7882	0.876355	28.3345	0.869482
150	18.3969	0.443059	20.6972	0.315122
200	17.7738	0.339456	19.7032	0.507448

Table E.69: Response Time of Query Type 5 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Page Faults	Confidence Interval	Page Faults	Confidence Interval
30	399.645	19.0795	375.181	17.3069
50	285.415	13.9614	331.77	12.9825
70	216.283	9.96841	258.244	11.2237
100	147.061	7.68702	165.116	9.41752
150	113.249	4.10782	112.432	4.07364
200	97.1104	3.70708	94.6364	3.49661

Table E.70: Page Faults of Query Type 5 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Buffer Faults	Confidence Interval	Buffer Faults	Confidence Interval
30	394.104	18.7661	369.869	17.1662
50	274.118	13.7815	309.462	11.4407
70	199.202	9.40287	235.726	9.85028
100	119.609	6.79869	144.386	8.24883
150	71.1568	2.15138	81.0423	2.38322
200	43.8498	1.40974	45.8085	1.409

Table E.71: Buffer Faults of Query Type 5 at 200 Frames

Buffer Size	Hot Set		DBMIN	
	Double Faults	Confidence Interval	Double Faults	Confidence Interval
30	38.2148	1.72183	43.404	1.89007
50	37.5492	1.8948	74.2961	2.44589
70	36.4142	1.58668	69.714	3.62815
100	36.9006	1.50697	49.2392	1.65016
150	30.7116	0.808264	32.7361	0.326341
200	25.6362	0.477676	29.5354	0.461046

Table E.72: Double Faults of Query Type 5 at 200 Frames