Effective Clustering and Buffering in an Object-Oriented DBMS

By

Ellis E-Li Chang

B.S. (National Taiwan University) 1979
M.S. (University of Wisconsin at Madison) 1983

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA at BERKELEY

Approved: Randy H. Katz 4/13/89
Chair Date

Richard D. Newton 5/17/89

Anne Segal 5/17/89

****

Effective Clustering and Buffering in an Object-Oriented DBMS

Ellis E. Chang

ABSTRACT

Object-oriented databases provide new possibilities for inheritance and structural relationships in data semantics. This dissertation examines how to use these additional semantics to obtain more effective object buffering and clustering. We use the information collected from real-world object-oriented applications, such as the Berkeley CAD Group's OCT design tools, as the basis for a simulation model with which to investigate alternative buffering and clustering strategies. Observing from our measurements that real CAD applications exhibit high data read to write ratios, we propose a run-time reclustering algorithm whose evaluation indicates that system response time can be improved by a factor of 200% when the read/write ratio is high. In our study, we have found it useful to limit the amount of I/Os allowed to the clustering algorithm as it examines candidate pages for reclustering at run-time, and, because performance varies little according to the number of I/Os involved, a low limit on I/O appears to be acceptable. We also examine, under a variety of workload assumptions, context-sensitive buffer replacement policies with alternative prefetching policies. Using these simulation results, we provide implementation hints for existing or future Object-Oriented DBMSs.

ACKNOWLEDGEMENT

Being a graduate student at UC - Berkeley Computer Science Department was a simulating experience. From this environment, I have learned and improved my skills in developing, presenting, and quantifying new ideas in a very competitive field. My advisor, Professor Randy Katz, who brought me into the database research back in 1982 at University of Wisconsin - Madison, has provided me with generous support and continued encouragement. I am deeply indebted to him. I also like to thank my committee members, Professors Richard Newton, Larry Rowe, and Arie Segev, who have given me helpful suggestions on this research. The valuable OCT access pattern trace information won't be collected without OCT group's help. Special thanks to Peter Moore, Jeff Burns, and the entire MOSAICO group who helped me analyze the trace results.

My colleagues and friends, Dr. Ming-Chien Shan from Hewlett-Packard Laboratory, and Dr. Tu-Ting Cheng, my former project manager at Hewlett-Packard Database Language Division, have given me continued support while I was working at Hewlett-Packard as well as studying at Berkeley. My officemates, Gaetano Borriello, David Gedye, Fred Obermeier, and David Wood were always giving me prompt response and support during my preliminary and qualifying examinations.

Finally, I like to devote this dissertation to my wife Caroline, who has constantly given me support during the past five years.

Table of Contents

Lists of Figures

Lists of Tables

CHAPTER 1

INTRODUCTION

1. OODBMS Trend and Barriers

While they are currently addressing different classes of business applications, such as Engineering Information System (EIS) and on-line transaction processing (OLTP), vendors of relational DBMSs (ORACLE, RTI, Informix) and emergent suppliers of object-oriented systems (e.g., Ontologic, Servio Logic) are both proclaiming that object-oriented approaches to information systems development will dominate in the 1990s. The idea of "objects" extends the conventional Entity-Attribute-Relationship modeling approach to allow finer shades of meaning, the inheritance of characteristics, and the reflection of multiple states (which is required in design applications). Whereas relational systems can cause poor performance by forcing the definition of natural objects to be fragmented into multiple atoms of meaning, object systems try to maintain the key facts about objects at the level of composition at which they are most used, thus avoiding excessive data aggregation [KIM87, LAND86].

However, there are some major barriers object-oriented database management system vendors and researchers need to overcome. First of all, there are too many object-oriented data models, such as ORION [KIM87], IRIS [FISH87], POSTGRES [ROWE87], and VBase [LAND86]. Each model adopts or invents different terminologies to describe its model which has confused the potential object-oriented database users. Secondly, the performance of these prototypes or products is only comparable to or worse than the relational database system [CATT88]. Without observing large performance improvement from object-oriented database system, users can not be convinced to use the object-oriented

database technology. Lastly, because no stable and good object-oriented design methodology is available now, users cannot develop real-world applications using any object-oriented database systems.

In this dissertation, we focus on the second object-oriented database system barrier, performance, by exploiting data semantics provided by object-oriented database systems.

1.1. Inheritance in OODBMS

The key function of an object-oriented system is to map an application's data manipulation logic into a set of *abstract data types*, with associated operations and attributes. In addition, object-oriented systems often support the concept of *type hierarchies*, in which a taxonomic classification is applied to instances, which belong to types, which in turn belong to supertypes, etc. Operation and attribute definitions can be propagated along the lattice formed by instances, types, and supertypes through *inheritance* mechanisms. Thus, operations and attributes defined within a type are callable and usable by all its subtypes and instances, providing both a short-hand specification of definitions as well as a means of information hiding.

The model of inheritance implemented in Smalltalk-80 [GOLD83] has several desirable features. First, new object types can be introduced into the lattice without the type definer needing to know all aspects of its supertypes, since inherited definitions can always be overridden within the new type. Information hiding and independence are thus supported. Further, new types (or instances) can be created which inherit the properties and behavior of their supertype (or type). Inheritance thus provides defaults for new type definitions and creation of instances.

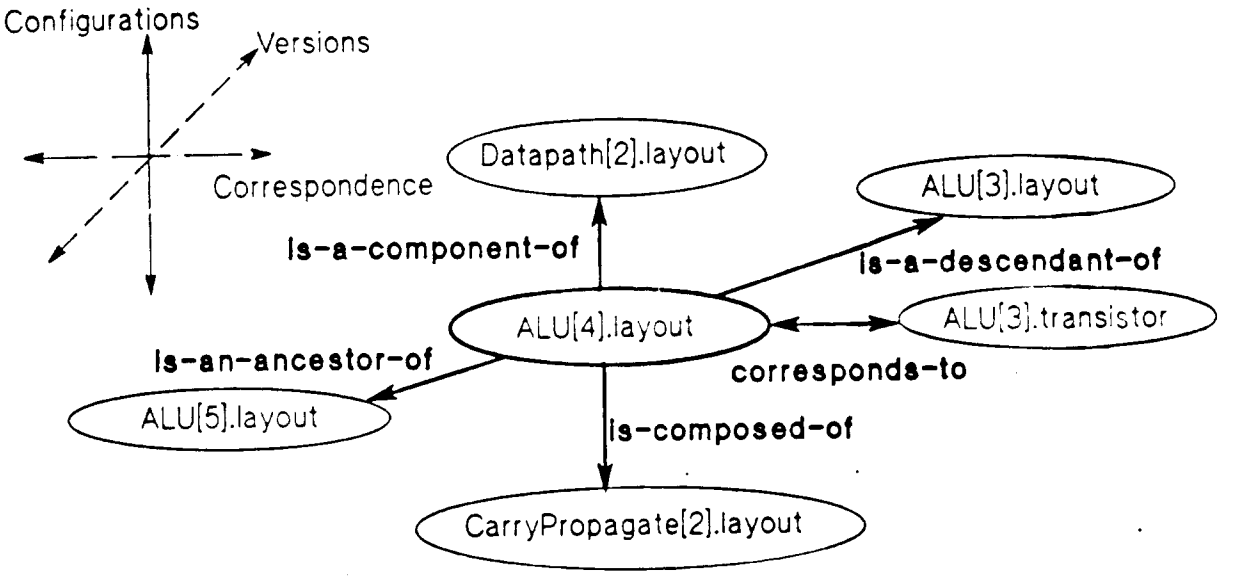

Figure 1-1 -- Version Data Model

Design data is organized as a collection of typed and versioned design objects, interrelated by configuration, version history, and correspondence relationships. Objects are internally denoted by the triple *name[i].type,* where *name* is the object name, *i* is the version number, and *type* is its representation type. For example, ALU[4].layout is descended from ALU[3].layout and is the ancestor of ALU[5].layout. It is also a component of DATAPATH[2].layout and is composed of CARRY-PROPAGATE[2].layout. Additionally, ALU[4].layout corresponds to other objects, such as ALU[3].transistor.

Some aspects of CAD data, particularly the introduction of versions and composite objects, complicate the Smalltalk-80 style of inheritance. For instance, in our Version Data Model [KATZ86a], three structural relationships; configuration, version history, and correspondence, are explicitly supported (see Figure 1-1). Note that objects are named by the triple *name[i].type*, where *name* is the object name, *i* is the version number, and *type* is its representation type. Version histories maintain **is-a-descendent-of** and **is-an-ancestor-of** relationships among version instances of the same real world object (e.g., ALU[4].layout **is-a-descendent-of** ALU[3].layout, both of which are versions of ALU.layout). A structural

version object is associated with each collection of version instances. Structural *configuration objects* relate composite representational objects to their components via **is-a-component-of** and **is-composed-of** relationships. Finally, correspondence identify objects across types that are constrained to be different representations of the same real world object, e.g., ALU[4].layout **corresponds-to** ALU[3].transistor if these are different representations of the same ALU design. More generally, correspondence can denote arbitrary dependencies among representational objects They are explicitly represented by structural *correspondence objects*. These three relationships provide alternative hierarchical organizations for design objects. Embedding these structures into type-instance models while extending the inheritance mechanisms to cover them becomes problematic. For example, all versions of an object could be modeled as instances of the same generic type [BATO85]. Thus, certain properties and behaviors, common to all versions, can be defined in the type definition and inherited by each version instance.

However, there are some properties and behaviors (as well as structural relationships and constraints) that an offspring version might wish to inherit directly from its parent version rather than from its type. Consider the following example. If ALU[1].layout corresponds to ALU[3].netlist, then a new descendant of ALU[1].layout should inherit this relationship as a default (see Figure 1-2). While it is possible to constrain all ALU layout versions to correspond to some ALU netlist instance, it is not possible to specify particular correspondences on an instance-by-instance basis (see Figure 1-3).

We can envision cases where it is desirable to inherit along relationships other than version histories. For example, [BUCH86] describes how constraints on a composite object can be inherited by its components. In fact, it should be possible to inherit information along any kind of relationship known to the system, be it type-instance, ancestor-

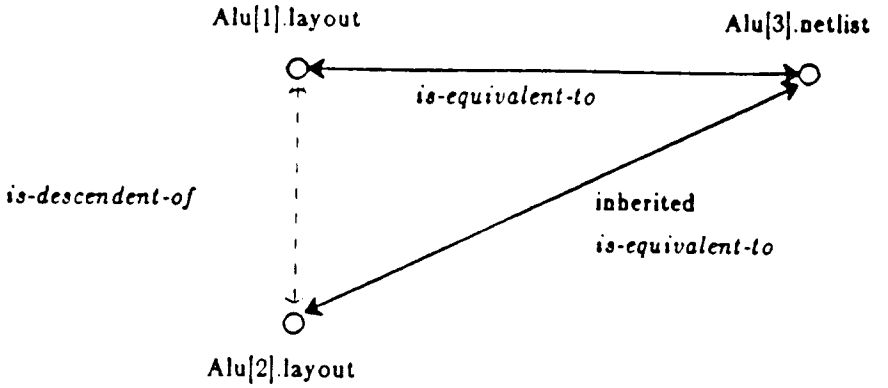

Figure 1-2 -- Inheriting Instance Specific Correspondences

Correspondence relationships constrain two objects to be equivalent. A new version typically inherits these relationships from its immediate ancestor. This is a good example of inheritance along version relationships which are not easily modeled with conventional type-instance inheritance.

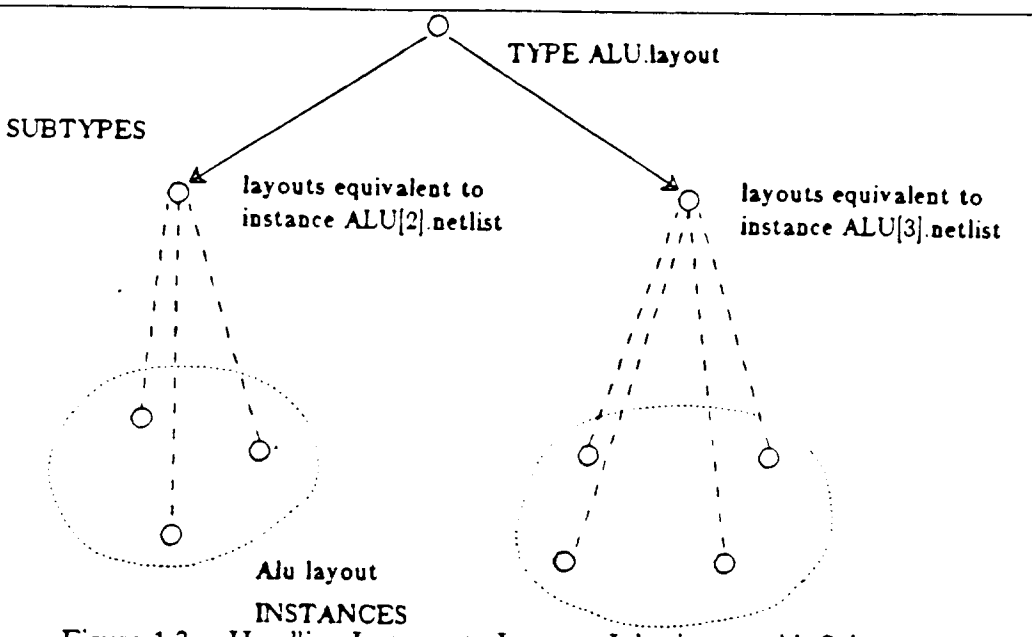

Figure 1-3 -- Handling Instance-to-Instance Inheritance with Subtypes

A subtype is created for each group of instances that shares a common attribute or constraint. Obviously this leads to an undesirable proliferation of subtypes.

descendent, or composite-component, or even among correspondent objects. Standard type-instance inheritance cannot model these kinds of information propagations by itself. While it is possible to implement the above inheritance semantics using user-defined operations, it is not desirable to do so, since the inheritance semantics are not obvious from the data model and cannot be exploited by the database system. Therefore, we have proposed an alternative *instance-to-instance* inheritance which allows direct inheritance among instances [CHAN87a].

1.2. Thesis Overview

In this dissertation, we are particularly interested in how clustering and buffering algorithms can use structural relationships and inheritance semantics to improve DBMS response time. For example, if a design browser walks through multiple representations of

the same design objects, clustering across correspondence will reduced the number of I/Os to retrieve all these representations and improve the response time. Alternatively, if a simulation tool traverses the netlist representation hierarchy, clustering along the configuration hierarchy is preferred. If information is frequently inherited along the version history, the system may place the object near its ancestor object to save disk space. Moreover, the buffer manager may accept *hints* from the clustering algorithm to keep candidate pages for clustering in the buffer pool, to avoid I/Os during clustering and to improve response time.

We will describe a run-time reclustering algorithm which we have observed to improve the overall system response time by up to 200% when the read/write ratio is high. We have found it useful to limit the amount of I/O allowed to the clustering algorithm as it examines candidate pages for clustering at run-time. We will also discuss how prefetching with an alternative scope of candidate pages affects response time when different buffer replacement policies are used.

We have instrumented an object-oriented Data Manager OCT, developed by the UC-Berkeley CAD group [HARR86], and have collected the access pattern information of more than ten CAD tools running on top of OCT. About 5000 tool invocations, representing approximately 400 hours of design work, are recorded. We have observed very high data read to write ratios from the OCT tools environment in the measurement results, which implies that dynamic clustering and context-sensitive buffering can be very useful in object-oriented applications.

The rest of the dissertation is organized as follows: Chapter 2 describes relevant work on inheritance mechanisms and how they are exploited by database management systems. Chapter 3 discusses how structural relationships and inheritance can be used in various clustering and buffering algorithms. Chapter 4 describes the OCT tools' access pattern,

read/write ratio, I/O rate and structure densities which are subsequently used to define the workload in the simulation model. Chapter 5 describes the simulation model, constructed in a modeling language called Performance Analyst's Workbench System (PAWS) [PAWS83] to evaluate the performance impact of various object clustering and buffer management algorithms. The simulation results are discussed in Chapter 6 through "Two-level factorial" analysis technique. Finally, our conclusions and future directions are given in Chapter 7.

CHAPTER 2

RELATED WORK

2. Introduction

Two bodies of work relate directly to this research. The first concerns the alternative inheritance semantics in data models. The second addresses techniques and algorithms to implement such mechanisms. The following table (Table 2-1) summarizes representative work done on the data modeling and implementation aspects of inheritance. The system built by Ontologic Inc. provides the inheritance semantics closest to those proposed here.

System	Authors	Inheritance Semantics	Techniques
ORION	*[KIM87]*	Type-Instance/Type	**flat schema**
POSTGRES	*[ROWE86]*	Type-Instance/Type	**caching**
EXODUS	*[CARE86]*	Type-Instance/Type	**clustering hint**
PROBE	*[DAYA86]*	Type-Instance/Type	**not known**
IRIS	*[FISH87]*	Type-Instance/Type	**none**
CACTIS	*[KING86]*	Derived/Attribute Graph	**clustering, trigger**
KEE	*[KEE86]*	Type-Instance/Type	**smart copy**
ART	*[ART86]*	User-definable	**not known**
Ontologic	*[ATWO85]*	Instance-Instance	**caching**
GemStone	*[MAIE86]*	Type-Instance/Type	**clustering**
Smalltalk	*[GOLD83]*	Type-Instance/Type	**Hash Table**
CommonLOOPS	*[MITT86]*	Prototypical Objects	**Virtual Copies**
Actor	*[LIEB86]*	Prototypical Objects	**Delegation**

Table 2-1: Relevant Works Summary

2.1. ORION

ORION [KIM87, KIM88], an object-oriented database management system developed by the Microelectronics Computer Technology Corp. database group, supports inheritance along the abstract data type lattice. An instance has only one associated abstract data type. As a user defines a abstract data type which inherits attributes from other abstract data types, the system flattens the inheritance hierarchy and stores only the flat form of the schema in the database catalog. This approach improves run-time performance by eliminating the lookup overhead along the inheritance link. Consistency among abstract data types is guaranteed because any schema changes cause the system to regenerate the flat form. In other words, the notion of *inheritance* is known only to the schema processor and is used by the front end application as an abstraction mechanism. The storage system and buffer manager have no knowledge about *inheritance*. The ORION DBMS, however, also ignores the inheritance semantics information, which is needed to support an instance-to-instance inheritance mechanism efficiently.[1] The inheritance-based run-time clustering techniques and the context-sensitive buffer replacement algorithm described in Chapter 3 will demonstrate how inheritance semantics information can be exploited at run time.

2.2. POSTGRES

The current POSTGRES data model [ROWE87] supports objects that are abstract data types and procedures. There is no direct support for inheritance in POSTGRES. Although procedures can be used to emulate inheritance, the task is not trivial from user's point of view. In [ROWE86], the shared type hierarchy is cached in users' virtual memory and constructed by an object manager upon applications' request. We believe that this

[1] No instance inheritance is supported by ORION.

complicated cache consistency protocol can be avoided if the underlying system (POSTGRES) supports inheritance directly. Furthermore, smart prefetching and run-time clustering become feasible if both inheritance and structural relationships are known by POSTGRES. Chapter 6 addresses how the storage system and buffer manager can use inheritance and structural relationships to improve the overall system response time.

2.3. EXODUS

EXODUS [CARE86] is an extensible database system being developed at the University of Wisconsin, Madison. It is a "DBMS generator" that supports the rapid construction of application-specific DBMS. The Type Manager maintains class hierarchies and allows inheritance along the class lattice. No instance-to-instance inheritance is supported in EXODUS. The Database Implementor can pass *hints* for object placement to the Storage Object Manager through the E language interface. However, no detailed clustering techniques are proposed in EXODUS, and structural relationships and inheritance semantics are not used by the storage manager at run-time.

2.4. PROBE

PROBE [DAYA86] is a knowledge-oriented DBMS being developed at CCA. Its data model is an extension of the DAPLEX functional model. PROBE supports two basic types of data objects: *entities* and *functions*. Entities are further grouped into collections called *entity types*. Although PROBE can emulate the *instance-to-instance* inheritance semantics through *functions*, the performance of such emulation is unclear at the present time. Most of the research work done by PROBE is in multiargument and computed functions. No detailed description about the implementation of PROBE's inheritance mechanism is available.

2.5. IRIS

IRIS [FISH87], an object-oriented DBMS developed at Hewlett-Packard database lab, adopts the DAPLEX functional model and supports type-instance inheritance through its OSQL interface. No instance-to-instance inheritance is provided and structural relationships such as versioning and configuration are not known to the storage manager. Since the underlying storage manager of IRIS is very similar to the storage component of System R, no navigation or complex objects are supported.

2.6. CACTIS

The goal of CACTIS, a DBMS that developed at the University of Colorado, is to support very complex derived information in as efficient a fashion as possible. (See [KING86]) One interesting property of the CACTIS data model is its ability to attach constraints to attributes. Although CACTIS does not support any inheritance mechanisms explicitly, it can emulate the instance-to-instance inheritance through its attribute evaluation mechanism. CACTIS also addresses some implementation issues such as indexing and clustering. However, its clustering algorithm is greedy in nature and does not consider the inheritance semantics during its clustering process.

2.7. KEE

KEE [KEE86] is a sophisticated knowledge engineering tool that incorporates several powerful and versatile AI methodologies for solving complex problems requiring non-algorithmic solutions. It uses frame-based knowledge representation with inheritance for representing domain knowledge. In KEE, a knowledge base is built up of five basic building blocks: units, slots, slot values, facets, and facet values. Units are *objects*. Slots are *attributes* of units. Slots may have various values which describe different units. Facets

similarly describe slots.

The KEE's inheritance algorithm is "smart" about making copies of slots that are exactly like a parent slot. If a slot is created via inheritance and has no local information in the child's version, the parent and the child share the same slot data structure. If the child slot is ever given any local information after creation, a separate data structure is created. This data structure continues to exist, even if the slot loses all of its local information.

The KEE's implementation assumes the availability of an infinitely large virtual memory. To emulate this assumption, most of the expert systems use the *dump* and *load* utility programs to transfer objects between memory and disc, but because these utility programs do not normally support direct access capability, the system cannot update an object in place without extensive storage reorganization. The clustering techniques we will propose in Chapter 3 will not work since KEE is not capable of storing objects at a prescribed position. Without smart clustering techniques on the physical level, the resulting referential locality will cause extra I/Os at run-time.

2.8. ATR

ART is an expert system development shell from Inference Inc [ART86]. It provides two kinds of relationships: *inheritance relationships* and *non-inheritance relationships*. The **is-a** and **instance-of** relationships are *inheritance relationships* and users can also define any new relationships within schemas to be inheritance or non-inheritance relationships. Conceivably, this feature allows applications to do instance-to-instance inheritance easily. However, the instance-to-instance inheritance is not directly supported in ART's model.

No detailed implementation information is available at the present time.

2.9. Ontologic

Vbase, an object-oriented database system being built by Ontologic [LADI86], supports type/subtype, type/instance inheritance and the following abstractions: **is-a**, **a-part-of**, and **an-instance-of**, along with built-in support for both version and alternatives. Inheritance within **is-a** and **an-instance-of** is exactly the type-instance inheritance found in most of AI systems. The Object Manager also supports property inheritance along the **a-part-of** abstraction which is similar to the **is-composed-of** and **is-component-of** relationships described in [KATZ87]. However, no notion of **corresponds-to** relationship is supported, and inheritance does not occur along the version histories relationship.

One interesting feature of the Vbase system is its clear distinction between abstract types and storage classes. In Vbase, an abstract type defines a set of abstract behaviors, such as operations, properties, attributes, and relationships. The storage class, on the other hand, manages these abstract behaviors' storage, including dereferencing, clustering and object recovery. An abstract type provides a default storage class which implements these behaviors and can have a wide range of storage classes. This simplifies adding a new storage class for use by existing abstract types.

Raising the storage problem so that it is a semantically visible portion of the system has another advantage for clustering. The Vbase storage managers all support a series of clustering levels. A normal arrangement supports chunks (contiguous storage), segments (a clustering of chunks) and areas (a clustering of segments); however any given storage manager is free to support any number of levels of clustering. In Vbase, objects may be clustered at any clustering level, along semantic relationships. Typically, all of the separate storage pieces required to implement an object are clustered within the same chunk, however, no run-time clustering is supported at this point. The detailed physical

implementation information is not available at the present time.

2.10. GemStone and Smalltalk

GemStone is an object-oriented database server which supports a model of objects similar to that of Smalltalk-80 [MAIE86]. Inheritance is restricted to the type-instance and type-type lattice, and no instance-to-instance inheritance is allowed. One interesting feature of GemStone is that it allows users to control object placement explicitly. If the chosen segment has no room for the target object, no segment splitting is invoked and another free segment is used. Applications can cluster related objects together by the unit of *segment*, which is mapped physically to a set of pages. However, the system does not automatically cluster objects for the users.

2.11. Prototypical Object

[MITT86] and [LIEB86] have proposed the concept of *prototypes* to replace types. Instances are associated with a prototype object, and are defined to behave like the prototype object unless the instance overrides some aspect of the prototype object definition. The difference between prototypes and types is primarily conceptual. By providing mechanisms such as delegation [LIEB86], which is essentially message passing, it is possible to implement inheritance-like propagation directly among instances. For example, the request to obtain an instance's attribute is forwarded to its prototype, if the attribute value is not locally defined.

2.12. OBject Server

OBject Server, known as ObServer, is the storage manager of the ENCORE database system [HORN87]. One interesting feature of ObServer is its object clustering mechanism, which utilizes two types of heuristics: *transaction-oriented* and *single-object* evaluation of

object usage. *Transaction-oriented* heuristics monitor object usage within the context of a transaction; that is, transaction-oriented heuristics monitor how objects are used together. *Single-object* heuristics use three measurements amassed over a period of time for monitoring: the access count, the open count, and the access ratio. The access count refers to the number of times an object is accessed in a given segment. The open count refers to the number of times the segment is opened. The access ratio is the quotient between the access count and the open count. However, these heuristics and detailed statistics are only used for database tuning.[2] No dynamic clustering is invoked at object creation time. Since the ObServer has no knowledge of object-oriented semantics such as inheritance and structural relationships, it cannot exploit such information at the storage level. We claim that such an ability is critical to OODBMS performance.

2.13. Object-based Clustering

A number of object-oriented database management systems, either prototypes or products, have implemented object-based clustering mechanisms. (See [MAIE86, ATWO85, ZDON84, KIM87].) Two characteristics are common to each of these implementations: (1) a segment (a collection of pages) is used as a clustering unit, and (2) user's hints are utilized at object creation time. For example, if a user hint such as "nearby object XX" is provided, the system tries to store the target object with object XX in the same page or adjacent page. Otherwise, the first buffer page with available space is chosen even though other buffer pages may contain the target object's structural related objects, e.g., ancestor or composite objects. Since these systems do not model structural relationships as first class objects in their data models, the storage component has no information to exploit dur-

[2] How these heuristics and statistics are used to make what kinds of decision is not addressed in [HORN87].

ing the clustering process. Users' hints are the only useful semantics which can be used by the storage component. Moreover, none of these systems do reclustering when object structures are changed, e.g., new components are included in a composite object. Neither do these systems exploit inheritance semantics.

2.14. File/Record-based Dynamic Restructuring

The problem of incremental file restructuring has been studied in [OMIE85] and is shown to be NP-hard. Omiecinski focuses on finding efficient heuristics to minimize the number of pages swapped in and out of the buffer during the restructuring process. However, his algorithms consider neither the overall system response time and the applications' characteristics, nor the rich semantics, such as structural relationships, captured by object-oriented systems.

[CHAN82] has proposed ways to cluster multiple record types for one-to-many or one-to-one relationships in the storage manager of ADAPLEX, a semantic data model. The paper introduced the notion of combining a physical pointer with a logical pointer to form a hybrid. For example, when representing a function from course to student, it may be useful to store both the student entity identifier, and the physical pointer to the student record. Although ADAPLEX supports the notion of inheritance and relationships, the storage manager does not use this information during the clustering or buffer replacement phase.

[SELL87] has proposed and evaluated various caching policies to maintain pre-computed SQL tables. Using a cache, the cost of processing a query can be reduced by preventing multiple evaluations of the same SQL procedure. Problems associated with cache organizations, such as replacement policies and validation schemes are examined.

However, because Sellis' proposal is based on relational model, structural relationships and inheritance semantics are not exploited to improve overall system performance.

2.15. Sun Benchmark

Sun Microsystem has proposed benchmarks for engineering DBMS, with emphasis on response time [RUBE87, CATT88]. The group has constructed a benchmark to run against: a relational DBMS, (i.e. RTI INGRES), an object-oriented DBMS, (i.e. Ontologic VBase), and a simple in-memory data structure manager from Sun Microsystem. The group strongly believes that the operations engineering applications perform on data cannot be expressed in the abstract form provided by SQL or other high-level DMLs. Simply adding transitive closure to the query language cannot meet the 1000 operations per second requirement from CAD or CASE applications. In this benchmark, no clustering or buffering mechanisms are discussed.

The following benchmark characteristics proposed by Sun Microsystem are related to this dissertation: (1) the read/write ratio is set at 10, (2) three basic operations are measured: traversal, lookup, and insert, (3) each object has at most three connections with other objects, and (4) the database size ranges from 3 MBytes to 300 MBytes. However, none of the numbers used in this benchmark has any supporting data as no formal process has been used to collect these statistics from real user environments. Further, as admitted by Duhl and Damon in [DUHL88], the benchmark model does not attempt to approach the complexity that an object-oriented application normally includes. For instance, the structural relationships and type-hierarchy discussed in Chpater 1 are not modeled in this benchmark at all. The benchmark results show that the simple in-memory data structure manager performs best in all cases. However, without concurrency control and recovery support, such a simple data manager is not suitable for object-oriented applications.

CHAPTER 3

EFFECTIVE CLUSTERING AND BUFFERING

3. Introduction

Engineering design, manufacturing and CASE applications demand high performance object management systems. These applications utilize structural relationships and inheritance mechanisms to effectively model their complex environments. Because conventional relational database management systems do not support such modeling primitives explicitly, the storage components cannot take advantage of them at run-time. In addition, these complex applications frequently perform materializations of an object hierarchy. For instance, a VLSI simulator or a software structure analysis tool needs to flatten an object hierarchy before a simulation run. Loading a large object hierarchy into memory becomes a problem. This section describes how to obtain better bandwidth and response time by exploiting inheritance and structural semantics in buffering and clustering for object-oriented applications.

3.1. Smart Clustering

The objective of clustering is to place frequently co-referenced objects near each other in physical memory. For *static* clustering, the system is quiesced, and the database administrator decides on a partitioning of objects. *Static* clustering is not effective for applications such as manufacturing which requires high availability. *Dynamic* clustering, on the other hand, is done at object creation and updating time when concurrent accesses are permitted. Although dynamic clustering may degrade the response time of writers, such degradation to writers can be offset by a large improvement of readers' response time as the simulation

results in Chapter 6 show. *Dynamic* clustering, therefore, becomes very attractive in object-oriented applications where reads dominate writes (Refer to Chapter 4).

The inputs to our clustering algorithm can be user's hints such as "access by configuration", inter-object access frequencies, or characteristics of inherited attributes. The user's hints are registered into the system through a procedural interface. The interobject access frequencies are inherited from the type at object creation time. For instance, in CASE applications, the run-time debugger frequently navigates from the object code to the source code and not in the other direction. Such information can be predefined at type creation time and used by all instances.

The clustering algorithm chooses an initial placement for each newly created instance based on which of the instance's relationships is most frequently traversed. This frequency information is available in the corresponding data type and is inherited by the newly created instance. These frequencies may be affected by the implementation of the instance's inherited attributes. For inherited attributes, the clustering algorithm uses an additional set of cost formulas to choose between implementation by copy versus by reference. The augmented access frequencies may change the initial placement of the instance.

The following set of parameters is used to control our clustering algorithm:

(1) *Candidate page pool.* Since the information used by the clustering algorithm is on an instance basis, the clustering algorithm needs to retrieve the physical page in order to get the corresponding information for clustering decision. Notice that the free space map in the page table is not sufficient to determine the potential candidate page. When looking for a candidate page for placement, the clustering algorithm may use only the pages available in the buffer pool, avoiding any I/Os during the clustering process. Or, the algorithm may search a limited number of pages on disk.

Alternatively, the algorithm may use the entire database as the candidate page pool. The candidate page pool can be: **Within_Buffer, With_IO_limit,** or **Within_DB.**

(2) *Page splitting policy.* When the preferred candidate is full, the storage manager must either split the page to make room for the new object, or choose the next best candidate page which has space for the new object. The page is split if the expected access cost resulting from the page-split is better than the cost of putting the new object in the next best candidate page. Otherwise, the next candidate page is examined, and the decision process recurses if there is insufficient room on the chosen page.

The problem of estimating the cost resulting from the page-split is similar to the Graph Partitioning Problem. However, this is known to be NP-complete and may not be suitable for a run-time clustering algorithm. [CHAN87a] proposes a greedy algorithm that partitions the nodes of the inheritance-dependency graph into two subsets each of which can fit into a page. At the same time, the greedy algorithm tries to minimize the total accessing cost. Because the algorithm does not try to find the optimal partition and only scans through the set of arcs once, the total running time is guaranteed to be linear. If the degree of connection and the number of nodes are small, the complexity of the page splitting algorithms should have no major impact on the overall system response time. Therefore, the page splitting policy can be: **No_Splitting, Linear_Split,** or **NP_Split.**

(3) *User Hints Policy.* Many computer-aided design applications make frequent use of configuration relationships, for example, a design rule checker walks the configuration from leaves to root as it performs its checks. However, most inheritance references are along version history relationships, since a descendant version typically obtains information from its ancestor. If users can make this reference information known to

the system, it can cluster objects based on the characteristics of the application. In our simulation model, we assume that users are responsible for making new reference information available to the system when access patterns are changed at run-time.

User hints can be simple, such as "place near object XX," or complicated, like "place near configuration X used in project P." To determine the set of candidate pages for run-time clustering, these user hints are translated into one or a set of object identifiers including page identifiers by looking up all objects specified by the corresponding user hint in the object table. The translation of simple user hints can be very fast. However, the handling of complicated user hints may require several I/Os. For example, the system needs to traverse the complete configuration hierarchy to translate the complicated user hint "place near configuration X" into a set of object identifiers. To avoid poor writer's response time during clustering process, the complicated user hint can be precomputed, permanently stored, and used at run-time. (This is very similar to the *view materialization* mechanisms studied in [HANS87].) Due to the complexity of materialized view validation protocols, we assume the simulation model translates every user hint, either simple or complicated, on-line without any precomputation. The user hint policy, therefore, can be designated as **User_Hints** or **No_Hints.**

3.1.1. Algorithm Details

The clustering algorithm is presented in pseudo code in Figure 3-1. The system calls the *cluster_object()* procedure to do clustering for a target object, either a newly created object or an updated object. In the latter case, if the change is a structural update to the target object, the system also calls *cluster_object()* to do clustering based on the latest statistics.

A detailed description of the five steps in the *cluster_object()* follows:

(1) Before choosing a candidate page on which to place the target object, the algorithm needs to know whether the clustering policy is to split the candidate page if it is out of space or to choose the next candidate page instead.

By examining all the inherited attributes of the target object, the algorithm determines the implementation strategy for every inherited attribute. Inheritance can be implemented *by copy* or *by reference*, i.e., the inherited value can be cached with the instance or the inheriting instance can point to the type (or source instance) that defines the value. The former approach has the advantage of fast access, but slow propagation of changes. The latter has better update performance, but exhibits slow access. A one-byte counter is used to monitor the update frequency of each attribute. When the counter is overflowed by the number of updates, it is permanently set to be 255 (i.e., 2^8-1). Using this one-byte counter, the system can make the inheritance implementation decision on an instance basis. If an inherited attribute has been updated frequently, *by reference* implementation is used. Otherwise, the algorithm will implement this inherited attribute *by copy*. These implementation decisions are made by calling *get_by_copy_set()* and *get_by_ref_set()*, and the results are returned in *copy_set* and *ref_set*.

The system also needs to determine the placement strategy within the physical space. All pages which contain source instances for these inherited attributes are returned in *inh_page_set* by calling *get_all_inh_page()*. Similarly, pages which contain the target object's interrelated objects, for example, its ancestor instance and component

```
PROCEDURE cluster_object(target_object)
BEGIN
   /* step 1: get initial information */
   cluster_policy := get_policy();/* Is page splitting enabled? */
   copy_set := get_by_copy_set();                          /* Inherited attributes implemented by copy.
   ref_set  := get_by_ref_set();  /* Inherited attributes implemented by reference.*/
   inh_page_set := get_all_inh_page();                     /* Source pages for inherited attributes.*/
   struct_page_set:= get_all_struct_page();                /* Source pages for structural objects.*/
   user_hint_set:= get_all_hint_page();       /* Source pages from user hints.*/
   page_set := inh_page_set + struct_page_set + user_hint_set;
   /* step 2: calculate ref_set lookup cost for each page */
   FOR p IN page_set                                       /* If by-reference attribute r is */
      FOR r IN ref_set          /* not in page p, storing target object */
                IF r NOT_IN p                              /* in page p requires one run-time */
                BEGIN                                      /* lookup for attribute r. */
               weight(p) := 1/(prob(p,struct_rel));
               Ref_LookUp(p):= Ref_LookUp(p)+weight(p);
                END;
   /* step 3: calculate copy_set lookup and storage cost for each page */
   FOR c IN copy_set                                       /* If by-copy attribute c is not in page*/
      FOR p IN page_set        /* p, we could either cache it in page p*/
                IF c NOT_IN p                              /* or change its implementation to be*/
                BEGIN                                      /* by-reference.      */
               weight(p) := 1/(prob(p,struct_rel));
               Copy_storage(p) :=Copy_storage(p)+sizeof(c);
               Copy_LookUp(p):= Copy_LookUp(p)+weight(p);
                END;
   /* step 4: calculate total cost of every page. If by-copy attributes are    */
   /*        implemented by reference, the total cost of storing target object    */
   /*      in page p is represented by Total_cost(p,1). Otherwise, the cost        */
   /*      is represented by Total_cost(p,2).                                      */
   FOR p IN page_set
      Total_cost(p,1) := Ref_LookUp(p)*Lookup_cost + Copy_LookUp(p)*Lookup_cost;
      Total_cost(p,2) := Ref_LookUp(p)*Lookup_cost + Copy_storage(p)*Storage_cost;
   /* step 5: pick up best candidate page and try to insert the object */
   candidate_page := Minimum (Total_cost);
   IF (cluster_policy EQ no_split)
      WHILE (NOT_FIT(candidate_page)
                candidate_page := Next_Min (Total_cost);
   IF ( (cluster_policy EQ page_split) AND ( NOT_FIT(candidate_page))
      Split_page(candidate_page);
END;
```

Figure 3-1 -- Pseudo Code for cluster_object

instances, are returned in *struct_page_set* by calling *get_all_struct_page()*.[3] User hints provide another source of candidate pages and are returned in *user_hint_set* by calling *get_all_hint_page()*. The union of *inh_page_set*, *struct_page_set*, and *user_hint_set* creates a set of candidate pages for placement in later steps.

(2) If a *by-reference* attribute is not in a chosen candidate page **p**, the system needs to look it up at run-time. As mentioned in section 3.1, the probability of accessing an individual structural relationship plays a vital role in modeling the run-time look up cost. We define *weight* (**p**) to be the inverse of this structural relationship access probability and the cost of storing the target object in page **p**, modeled by *Ref_lookUp* (**p**), to be the summation of all *weight* (**p**) of *by-reference* attributes of the target object.

(3) Inherited attributes which can be implemented as *by copy*, are either copied to a candidate page **p** or looked up at run-time. There are two cost variables involved: *Copy_storage(* **p** *)* is used to model the cost of storage and is incremented by the size of the by-copy inherited attribute, while *Copy_lookUp(* **p** *)* is used to model the cost of by-reference implementation and is incremented by weight(p) as *Ref_lookup(p)*.

(4) To determine the candidate page, the system needs to transform all the lookup and storage costs into the same scale for comparison. The lookup cost, *Lookup_cost*, is represented by $P_{hit}*C_{buf} + (1-P_{hit})[C_{io}+P_{io}*C_{io}]$, where P_{hit} is the probability of buffer hit, P_{io} is the probability of doing I/O during buffer replacement,[4] C_{io} is the cost of I/O, C_{buf} is the cost of searching through buffers, C_{lookup} is the cost of catalog lookup and C_{os} is the cost of getting a free page from O.S.[5] The storage cost, *Storage_cost*, is

---

[3] We assume that both the ancestor instance and components instances information are available from the target object, and that no I/Os are introduced by this procedure call.

[4] Both P_{hit} and P_{io} are based on the statistics collected during a simulation run.

[5] All these parameters use the number of instructions as the basic cost unit.

represented by *Lookup_cost* scale_factor*, where *scale_factor* is determined by users.

(5) If the clustering policy does not permit page-split, and the candidate page is out of space, the clustering algorithm chooses the next minimum cost candidate page on which to insert the target object. Otherwise, the system tries to split the candidate page to minimize the new run-time lookup cost. When the candidate *page_set* becomes empty, the system will allocate a new page for the target object.

The *Split_page()* procedure described in Step (5) tries to minimize the look up cost from the page-split. If a page contains N inherited attributes, its look up cost is represented as *N*Lookup_cost*, as described in Step (4). The detailed algorithm for procedure *Split_page()* follows:

Page_split Algorithm: Assume that the arc costs C_{e_i} (potential run-time lookup costs) are always maintained and sorted in the page header. The node capacity Cap_{v_j} (the object size) is available from the **object header** which is maintained by the system. Subset A and B represent the new page layout after page-split. Both subset A and B are empty at the beginning, and the available capacity of A and B is set to be (maximum_page_size * 0.75).

(1) Select the maximum value arc from E as e_{target} and set E to be (E - $\{e_{target}\}$). Let v_{head} and v_{tail} be the head and tail nodes of arc e_{target}.

(2) Suppose both v_{head} and v_{tail} are new to subsets A and B. Insert v_{head} and v_{tail} into subset A if $Cap_{v_{head}}$ plus $Cap_{v_{tail}}$ is less than the remaining capacity of subset A. Otherwise, insert v_{head} and v_{tail} into subset B if subset B has space for these nodes. If neither subset A nor B can accommodate both v_{head} and v_{tail}, a broken arc is found, and $C_{e_{target}}$ is added into C_{total}.

(3) Suppose v_{head} is in subset A, and v_{tail} is not in subset A or B. Insert v_{tail} into subset A if there is room. Otherwise, a broken arc is found, and $C_{e_{target}}$ is added into C_{total}.

(4) Suppose both v_{head} and v_{tail} are visited before, then a broken arc is found and $C_{e_{target}}$ is added into C_{total}.

(5) Loop back to (1) until arc set E is empty.

Algorithm Analysis: This algorithm is greedy. Since we only scan through the edges once, the total running time is guaranteed to be O(n) where n is the number of edges in G.

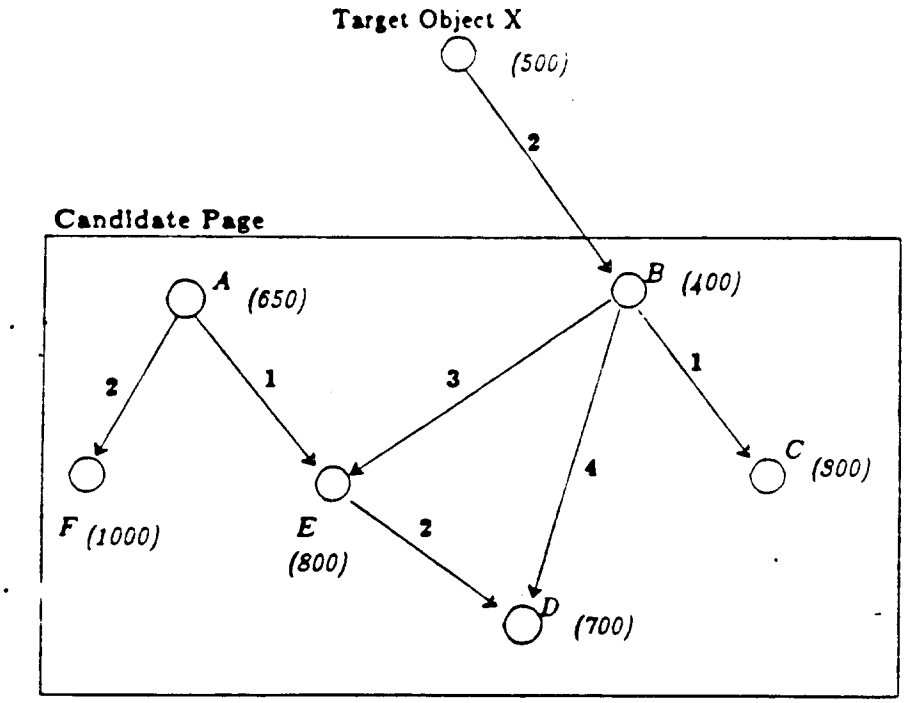

Figure 3-2 -- Page Split Example

Each node represents an object, and the number associated with it is the size of the corresponding object. The arc represents either the inheritance dependency or the structural relationship, and the number associated with it is the potential run-time look up cost. All the nodes within the box are on the same page, and the target object X is too large to fit into the candidate page in this example.

A page split example is shown in Figure 3-2. The clustering algorithm described in Figure 3-1 has chosen a candidate page shown in Figure 3-2 to store the target object X. Suppose that the maximum page size is 4000 bytes and the clustering policy is to split the page when it is out of space. Because the target object X is too large to fit into the candidate page, the clustering algorithm invokes the page splitting algorithm to determine a new page layout for these objects. The page splitting algorithm first places object B and D on the same page since the potential run-time look up cost from B to D is the largest. Object E is then chosen to be on the same page since the potential run-time look up cost from B to E is the next largest. However, due to the limitation of the available capability of a page, object A and F are placed on another page. After scanning through all the arcs in the inheritance dependency graph, the page splitting algorithm produces the final page layout shown in Figure 3-3. Notice that the actual run-time look up cost of the new page layout is 1 (from object A to E), represented by the dotted line in Figure 3-3,[6] whereas the actual run-time look up cost of the no-splitting layout is 2 (from object X to B). Therefore, splitting the candidate page has reduced the actual run-time look up cost by 1.

Discussion

To avoid the extra cpu time caused by the clustering mechanism, users may disable automatic dynamic clustering when the system is heavily loaded. Or they may enable/disable clustering based on the characteristics of their operations and data. For instance, operations like check-in/out require a large amount of data to be inserted into workspaces. With clustering enabled, better response time will be provided for future accesses at the expense of extra cpu time during check-in/out. Users can therefore tune

[6] All the potential run-time look up cost values are still the same as before. That is, the potential look up cost from B to E is still 3 and from A to F is still 2.

the context-dependent clustering policy to provide reasonable response time for their applications. The interaction between access frequency and clustering policy is evaluated in Chapter 6.

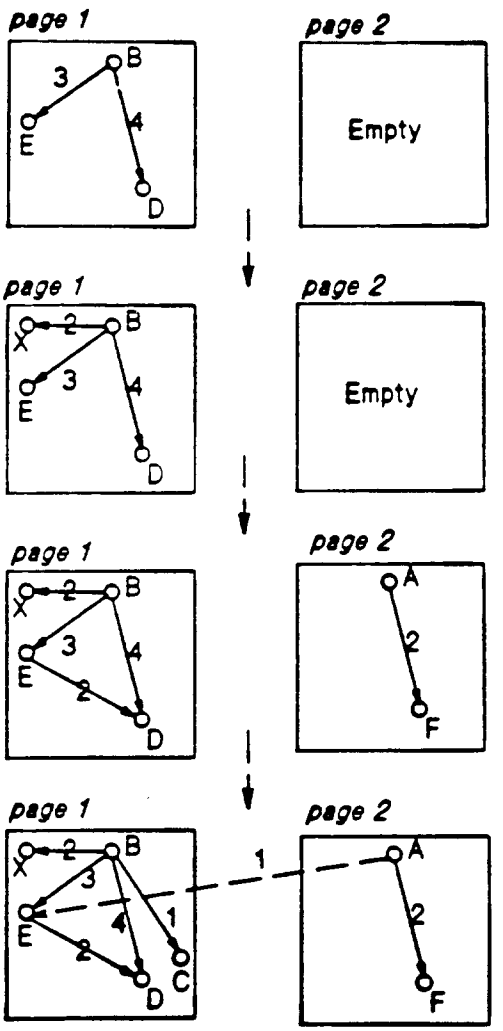

Figure 3-3 -- New page layout after page splitting

After page splitting, objects A and F are on page 2 and the rest of objects are stored on page 1. The dotted line between object A and E represents the actual run-time look up cost after page splitting. All the potential run-time look up cost values are still the same as before. That is, the potential look up cost from B to E is still 3 and from A to F is still 2.

3.2. Smart Buffer Replacement

The most common operations of object-oriented tools are navigation along the structural relationships and simple retrieval of design objects. For example, a macro cell router may navigate all the terminals and paths associated with an individual net. A computer-aided software engineering configuration manager may need to find all the composite modules of a particular component module during the integration phase. In general, object-oriented tools have fairly static access patterns (Refer to Chapter 4). Unfortunately, these are ignored by most database systems.

To obtain better response time, the buffer manager must determine prefetching and buffer replacement strategies by exploiting knowledge about the structural and inheritance relationships. Response time is improved if appropriate objects can be prefetched before they are needed, or if related objects can be kept in the buffer pool even if the relationships span disk pages. For example, if buffer X contains an object that references some attributes in buffer Y through an inheritance link, the buffer manager should try to keep buffers Y and X in the buffer pool at the same time.

The following set of parameters is used to control the buffering algorithm:

(1) *Buffer Pool Size.*

(2) *Buffer Replacement Policy.* An unsophisticated buffer manager uses a simple LRU buffer replacement policy whereas a more sophisticated approach uses a priority scheme which replaces the lowest priority pages first. The key challenge is to use the semantics of the interrelationships among objects on the buffered pages and hints about the access patterns to set the priorities intelligently. Frequently accessed pages have their priority increased. Infrequently accessed pages have their priority reduced, but this may be modified by their interrelationships with other pages, especially if

those are frequently accessed. Whenever an object is accessed, its related pages (these pages containing its component objects and its inherited attributes) might be in the buffer pool already. A good buffer replacement policy should keep these pages in the buffer pool, thus accruing no additional I/Os to bring them in later on. This is accomplished by using the usage pattern as well as the corresponding context of an individual page during the buffer replacement process. We call this new buffer replacement policy **Context-sensitive** and describe it in detail in Section 3.2.1. The other two algorithms examined in this study are: **LRU** algorithm [BELA66, DENN72], which removes the page that has not been referenced for the longest period of time, and the **Random** algorithm [BELA66], which randomly selects a page for replacement when a buffer page is needed.

(3) *Prefetch Policy*. At the beginning of an interaction with the database, the users provide the buffer manager with access hints such as "my primary access is via configuration relationships." This information influences the buffer manager's prefetch strategy. Touching an object causes the page containing it and the pages containing its immediate subcomponents to be brought into the buffer pool and given the same high priority. For example, such a prefetching strategy achieves extremely good performance for applications that walk the configuration hierarchy. Similar prefetch hints can be used to obtain a version object, its immediate ancestor, and its immediate descendents. Also, correspondence relationships can be used to obtain all objects corresponding to the one being accessed. Inheritance is treated in a similar fashion for determining prefetch groups. The candidate pages for prefetching can be constrained to either the buffer pool or the entire database. Notice that prefetching within the buffer pool does not create any extra logical I/Os. However, prefetching

will cause the buffer priority to be adjusted to the requesting applications.

```
PROCEDURE Fetch_Object(ObjID, structural_rel, prefetch)
BEGIN
  /* step 1: fetch the base page of target object. */
  IF (ObjID NOT_IN BufferPool)
  BEGIN
     buffer_id := Get_bufferframe();
     Get_page(ObjID, buffer_id, HighPri);
  END;
  /* step 2: get structural and inheritance links used by the target object.  */
  /*        If their containing pages are in buffer pool, set their priority  */
  /*        to be HighPri.                                                     */
  structural_links := Get_struct_links(ObjID, structural_rel);
  FOR link IN structural_links DO
     IF (link IN BufferPool)
                Set its bufferframe to be HighPri;
  inh_links := Get_inh_links(ObjID);
  FOR link IN inh_links DO
     IF (link IN BufferPool)
                Set its bufferframe to be HighPri;
  /* step 3: If prefetch option is on, prefetch all related pages into memory.  */
  IF (prefetch)
  BEGIN
     FOR link IN structural_links DO
                IF (link NOT_IN BufferPool)
                BEGIN
                           buffer_id := Get_bufferframe();
                           Get_page(link, buffer_id, HighPri);
                END;
     FOR link IN inh_links DO
                IF (link NOT_IN BufferPool)
                BEGIN
                           buffer_id := Get_bufferframe();
                           Get_page(link, buffer_id, HighPri);
                END;
  END;
```

Figure 3-4 -- Pseudo Code for Fetch_Object

3.2.1. Algorithm Details

For every fetch object operation, users can provide access path information and prefetch hints along with the target object identifier. The pseudo code of procedure *Fetch_Object()* is shown in Figure 3-4 and a detailed description follows:

(1) If the target object is in the buffer pool, the buffer manager sets the priority of the containing buffer as **HighPri**. Otherwise, the buffer manager allocates a buffer frame, fetches the object into it, and sets the priority as **HighPri**.

(2) To avoid useful pages being replaced, the buffer manager locates all buffer frames which contain either structural or inheritance information from the target object and sets these buffer frames as **HighPri**.

(3) If the prefetch option is enabled, the buffer manager does the prefetching for all the structural and inheritance related objects on behalf of the target object.

The example shown in Figure 3-5 further illustrates these steps. Suppose the target object X is located on page A, and its component objects are on pages B, C, and D. The prefetch option of this Fetch_Object() call is disabled and the structural_rel parameter is specified as *configuration hierarchy*. Before calling the Fetch_Object() to fetch the target object X, the buffer pool has already prepared pages B and D for replacement because of their low priority status. To fetch object X into the buffer pool, the buffer manager first needs to allocate a free buffer. Since both pages B and D have low priority status, either one of them could be chosen to be replaced. However, such a decision is not desirable because object X may reference these pages very soon. To avoid this, the *Fetch_Object()* procedure considers information provided by callers. The *structural_rel* parameter is set to be *configuration hierarchy*, and the contents of all buffers, e.g., page B and D contain component objects of object X are considered. The buffer manager tries to keep page B and D

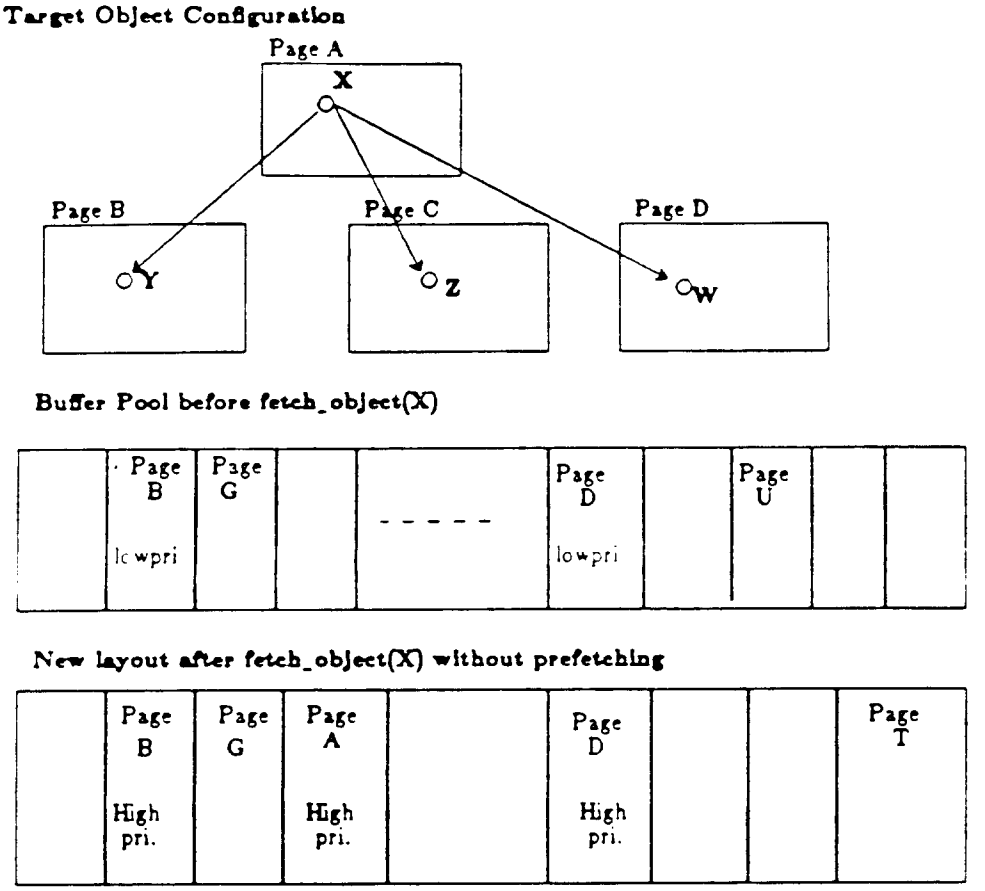

Figure 3-5 -- Smart Buffer Management Example

Each box represents a page frame, and a circle within a box represents an individual object. The arc between circles is the configuration relationship. In this example, object X is composed of objects Y, Z and W. The replacement priority is represented as "low pri" and "high pri." Two buffer layouts are shown to illustrate the difference before and after calling Fetch_Object().

in the buffer pool by increasing their priority status. The final buffer pool layout after this *Fetch_Object()* call is also shown in Figure 3-5.

Discussion

Note the close interplay between the clustering algorithm and buffer management. If the

clustering algorithm is efficient, then interrelated objects will be placed on the same page or in a small collection of pages. If not, and if access along these relationships is frequent, then the clustering algorithm will adapt to the access patterns by reorganizing the placement of objects. The buffer manager can alert the clustering algorithm about the need to reorganize.

3.3. Summary

In this Chapter, we have shown how inheritance and structural semantics can be exploited by clustering and buffering algorithms at run-time. Three parameters, *candidate page pool*, *page splitting mechanism*, and *user hints* are used to control the clustering policy. Another three parameters, *buffer pool size*, *buffer replacement algorithm*, and *prefetch mechanism* are used to define the overall buffering policy. Part of these parameter values are influenced by the actual observation from the OCT design environment. For example, to determine an appropriate candidate page pool, the clustering algorithm needs to understand the target object's structural relationships with other objects. If the target object has only two structural relationship links, then the maximum size of the potential candidate page pool is two and the *clustering with 2 I/O limit* policy may be sufficient (Refer to Section 6.1 for more detailed discussion). In the following Chapter, we will report the OCT design tools' access pattern in detail.

CHAPTER 4

OBJECT-ORIENTED APPLICATION ACCESS PATTERN

4. Introduction

This Chapter presents the access pattern information of an object-oriented data manager, OCT, which is being used extensively (about 20 client applications) in the Berkeley CAD community. Even though OCT provides only a subset of object-oriented concepts, we felt that OCT would give us a very good starting point in understanding the access pattern of object-oriented applications as well as helping us evaluate various clustering and buffering mechanisms for object-oriented applications. The access pattern information we collected covers about 5000 tool invocations representing approximately 400 hours of design work. We have observed very high data read to write ratios from the OCT tools environment in the measurement results, which implies that dynamic clustering and context-sensitive buffering discussed in the previous Chapter can be very useful in object-oriented applications. The following section reports on our findings.

4.1. OCT Data Manager Overview

OCT, data manager for VLSI/CAD applications [HARR86], supports a set of primitive object types which are used frequently by VLSI CAD tools and arbitrary attachments among these objects. Figure 4-1, for example, shows a net attached to a facet which is a basic design unit, four terminals attached to the net, and three paths attached to various terminals. Every attachment creates a link between two objects. These links are bidirectional and provide basic composition hierarchy information. However, users are responsible for maintaining the legal attachment among objects, and the system does not provide any

structure validation. Further, OCT does not allow user-definable operations to be associated to the object type and no explicit inheritance mechanisms is supported.[7]

4.2. Information Collected

When an OCT tool is invoked, either in a batch mode or interactive mode, its OCT data access information is accumulated in several local variables and logged to a file at the octEnd() time. To obtain the access pattern information for each individual tool, several OCT routines are modified. However, no OCT routines' external interface is affected by the instrumentation. All the ten OCT tools we measured only need to relink with the new

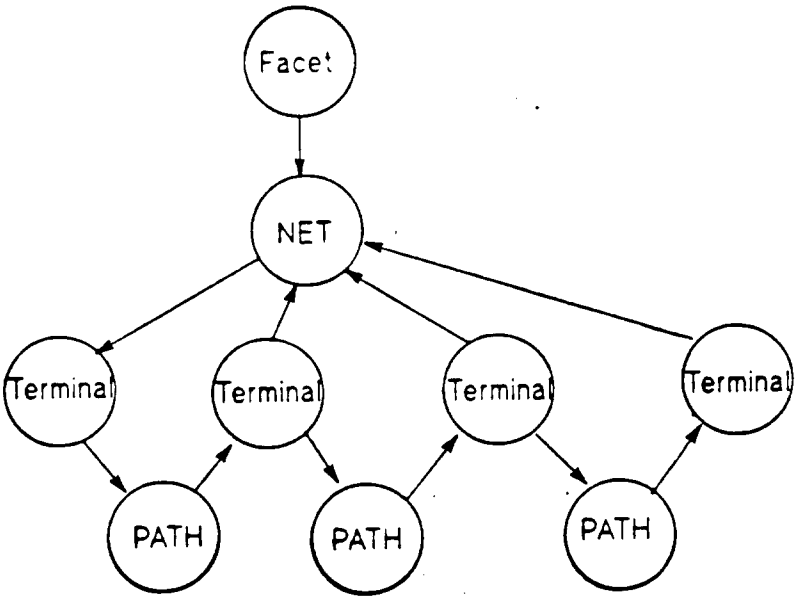

Figure 4-1 – MOSAICO Access Pattern

The run-time navigation path is represented by the arc direction from one object to another. A net is attached to the facet which is basic unit of design cell. Terminals are attached to the net whereas paths are attached to the terminals.

[7] However, upon creation of the interface, the formal terminal definitions are included in the interface facet automatically by OCT. This situation, the only one where OCT uses inheritance, is not available to OCT users.

OCT library and a tool specific file. The tool specific file is used to set up the tool identifier information in a global variable. At octEnd() time, the tool identifier is logged together with the corresponding access pattern information. To synchronize concurrent update to the log file, we used the UNIX flock() primitive to guarantee one writer at a time. The measurements are done on the VAX 8600 machine with the Ultrix operating system.

For each OCT tool invocation, we recorded the following information:

(1) The tool Identifier, such as "SPARCS" or "VEM", helps us understand the run-time behavior of an individual application and also allows us to relate the tools' functionalities to their overall run-time behavior. For example, this make it possible for us to know whether all the placement and routing tools have similar read/write ratios.

(2) Read and write activities. When objects are retrieved through "attachment", they are recorded as *structure read*. Any relationships created between objects via "attachment" are viewed as *structure write*. Remaining read or write operations are viewed as *simple read* or *simple write*. We also recorded the object type information for every read and write operation.

(3) Session time. A session is defined as the time interval between octBegin() and octEnd(). With this, we can measure the I/O rate per session[8] and thus better understand the correlation between the session time and the applications' functionalities. For example, the layout tool like "VEM" tends to have longer session than placement and routing tools.

(4) Fan-out of upward and downward structural access. Reading a composite object may only trigger the retrieval of a subset of its component objects. Similarly, reading a

[8] Since octEnd() is called at the end of each tool invocation, we cannot measure the actual I/O rate and the I/O distribution within a sesstion is not measured either.

component object may cause the return of several of its composite objects. For example, a netlist simulator would only navigate along the <cell>, <net>, and path, leaving the remaining component objects attached to <cell> totally untouched. In other words, not all of the component objects are typically read when an application is traversing a composite object at run-time.

(5) OCT object usage profile. OCT supports various object types which are primarily useful for VLSI CAD applications. This object usage profile provides tool developers with information such as how OCT objects are being used at run-time.

4.3. R/W ratio

For every tool invocation, we collected the number of structure read, structure write, simple read, and simple write operations, all of which are at the logical level. We define the read/write ratio of every tool invocation to be the total number of structure reads and simple reads divided by the total number of structure writes and simple writes. Figure 4-2 shows the read/write ratio of nine OCT tools. VEM, a graphical editor not included in Figure 4-2, has the highest read/write ratio of 6000. The rest of the OCT tools' read/write ratios vary from 0.52 to 170. Each of these read/write ratios is the average read/write ratio of a specific tool. One interesting note is that different phases of the same application may have wide variations in the read/write ratio. For instance, the macro cell router MOSAICO is composed of **atlas, cds, cprep, PGcurrent** and **mosaico**. Figure 4-2 shows the read/write ratio within one run to vary from 0.52 to 170.[9] This is quite unusual compared to traditional debit/credit applications, in which the read/write ratio is quite stable throughout an application. Due to this dynamic of the applications' read/write ratio, the clustering algorithm

[9] Within each phase, the read/write ratio is fairly stable.

must be adaptive to achieve adequate response time during the different phases of an application.

Figure 4-3 shows the I/O rate of various OCT tools, calculated by counting all logical read and write operations and then dividing by the session time. All the tools except VEM are run in batch mode, and the session time does not include think time. Notice that most of the OCT tools have the following pattern: a read phase, a CPU intensive processing phase, and a write phase. When the CPU intensive part becomes smaller due to the MIPS

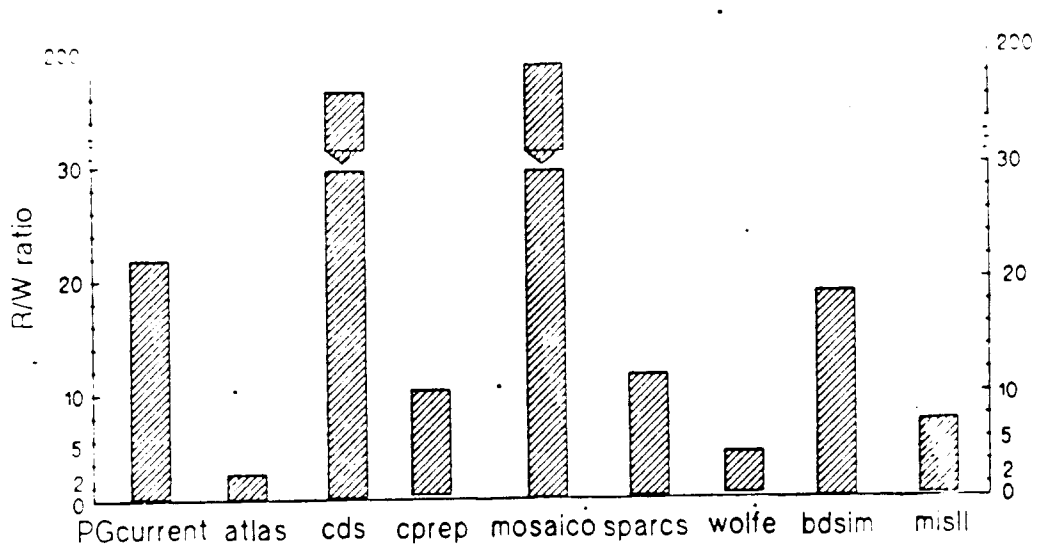

Figure 4-2 -- OCT Tools' Read-Write Ratio

Ten OCT tools are measured. The highest read/write ratio 6000 belongs to VEM, a graphical editor. The rest of the OCT tools' read/write ratios are shown here. Wolfe is a standard cell placement and global router. SPARCS is a symbolic layout spacer. MisII is a multiple-level logic optimizer and bdsim is a multiple-level simulator. MOSAICO is a macro cell router which is composed of atlas, cds, cprep, and mosaico. All these read and write operations are recorded from the buffer manager's point of view.

per CPU may double every year, the rest of the operations, read and write, will then be the determining factor of the application's performance. Although the I/O rates shown in Figure 4-3 only demonstrate which tool is the most I/O intensive, they still provide an useful information for future OODBMS design.

4.4. Structure Density

Figure 4-4 shows the downward access structure density of ten OCT tools. Although both upward and downward accesses are measured at run-time, we have observed that most of the upward accesses have only one object returned. Therefore, we only report the fan-out of downward accesses here. The structure density is broken into three categories: 0 to 3 for low density, 4 to 10 for medium density and above 10 for high density. Observing

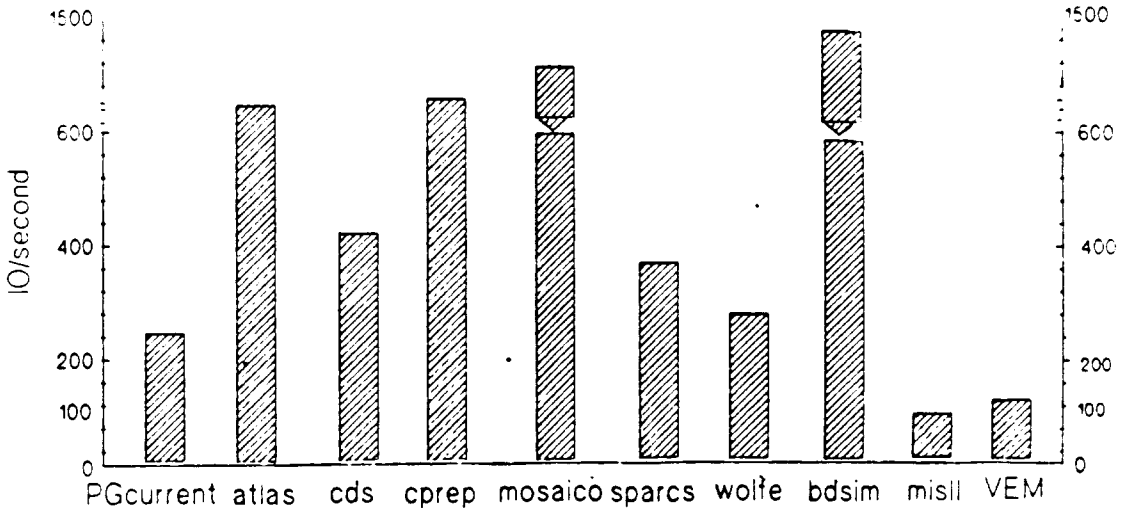

Figure 4-3 -- OCT Tools' Object I/O Rate

The Y axis is derived from the total number of logical I/Os including read and write operations and the session time. The X axis shows the 10 representative OCT tools.

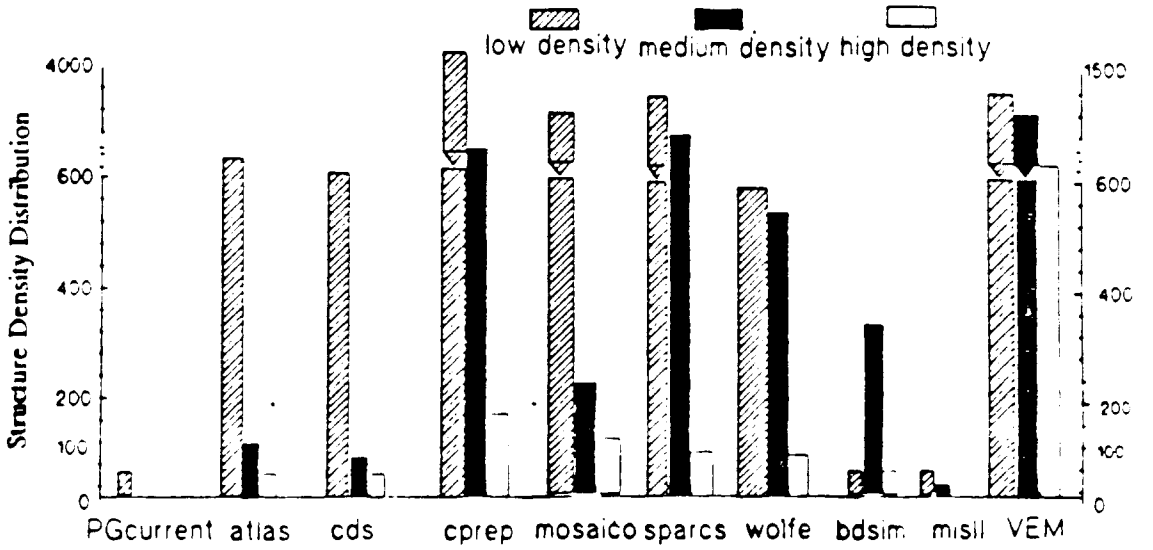

Figure 4-4 -- OCT Tool Structure Density Distribution

The structure density is broken into three categories: 0 to 3 for low density, 4 to 10 for medium density and above 10 for high density. The X axis represents the usage of every category for each tool.

---

from our measurements, the distribution of structure density above 10 has a very large variation and the next structure density higher than 10 may be 30 or 400. Therefore, we chose structure density 10 as the the breakdown point between medium and high density.

Most of the OCT tools' downward access, except VEM, are dominated by a low structure density (0 to 3 objects). VEM has the highest structure density, since it typically displays all the objects attached to the composite object. The VEM's downward access structure density can be viewed as the static structure density of the OCT data being measured. Averaging the VEM' downward access structure density, we estimate that the average static structure density of the OCT data being measured is between 3 to 9, which is quite different from the actual downward access structure density, 0 to 3 objects.

4.5. Access Pattern

After interviewing several OCT tool developers, we observed the following:

(1) Structural object-oriented applications[10] emphasize navigation rather than ad-hoc queries. For example, in Figure 4-1, the macro cell router MOSAICO navigates along the configuration to find all the paths used by a certain net. An ad-hoc query like "select all nets having 6 or more terminals" is not needed in MOSAICO.

(2) Certain access patterns can be eliminated if the underlying system supports integrity constraint. For instance, the cell compactor SPARCS scans through the entire design to make sure that no two terminals have more than one path between them. Such checking assures that SPARCS will not run into a loop at run-time, but also introduces a number of unnecessary I/Os. The SPARCS's average read/write ratio will only be affected by 0.4 if such checking is eliminated.

(3) Most of the access patterns are predictable. Certain objects, such as the net instance in Figure 4-1, are accessed several times during navigation.[11] Once this access pattern is known, the buffer manager can increase the priority of the object's containing page to improve the buffer hit ratio.

4.6. Object Usage

Several OCT object types are predefined for the OCT users. For example, octFacet, the fundamental unit in OCT, consists of a collection of objects that are related by attaching one to another. OctTerminal describes a terminal of an instance whereas octNet represents a logical connection between the terminals that it contains. OctPath describes a

[10] Since OCT does not allow user-definable operations to be associated to the object type, it only represents a part of the object-oriented applications.

[11] The re-read ratio information is not captured by our measurements.

'wire' of certain width and octBag describes an object whose only purpose is to hold other objects. Figures 4-5 and 4-6 show the profile of OCT object usage for both downward and upward access. Some key observations are:

(1) Bounding box (i.e. BOX as labeled in the Figure) is the most frequently accessed OCT object in both downward and upward retrievals. This is quite consistent with the OCT tools' characteristics; the router, layout compactor and graphic editor need bounding box information all the time.

(2) The low usage of <Facet> (labeled as FAC in Figures) is due to the fact that most OCT tools only work within the contents of a <Facet>, not across different <Facets>. Once a <Facet> object is retrieved, it is unlikely accessed again by the same application. However lots of its component objects are accessed..

(3) Both <terminal> and <path> have a similar amount of usage in downward access. Since <terminal> is connected by <path>, any navigation pattern will produce such a usage pattern.

(4) Almost no OCT tools use <string>, <real array>, <integer_array>, and <objectID>. This object usage profile can be used to decide which OCT object should not be supported in future extension of OCT.

(5) Most OCT tools use more downward access than upward access. This information can be used by the clustering and buffering algorithm, described in Chapter 3, to improve overall system response time.

4.7. Summary

This Chapter presents the OCT object access pattern information, broken down into read/write activities, session time, downward and upward structural access pattern, and

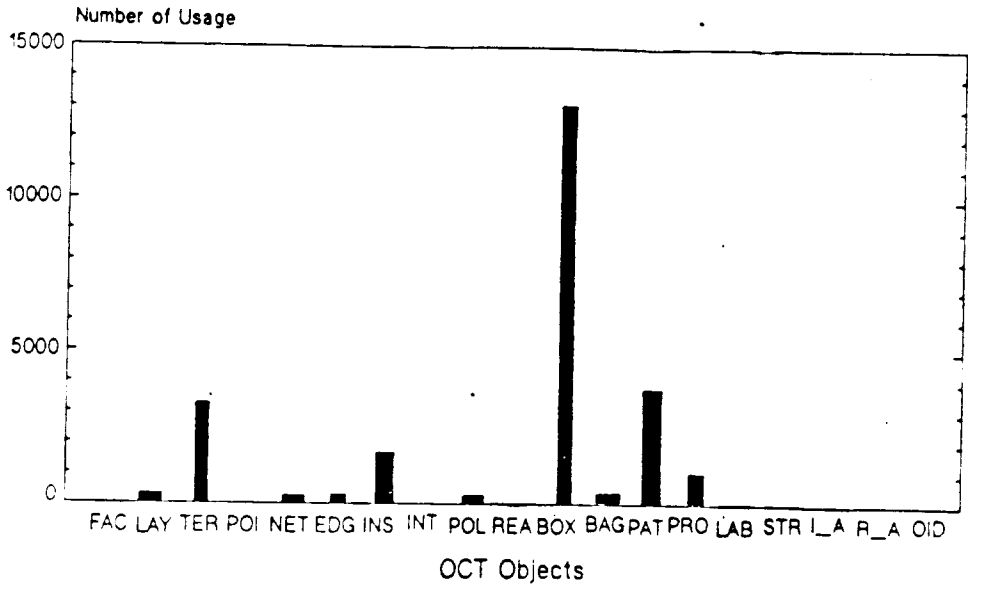

Figure 4-5 -- OCT Downward Access Object Usage Pattern

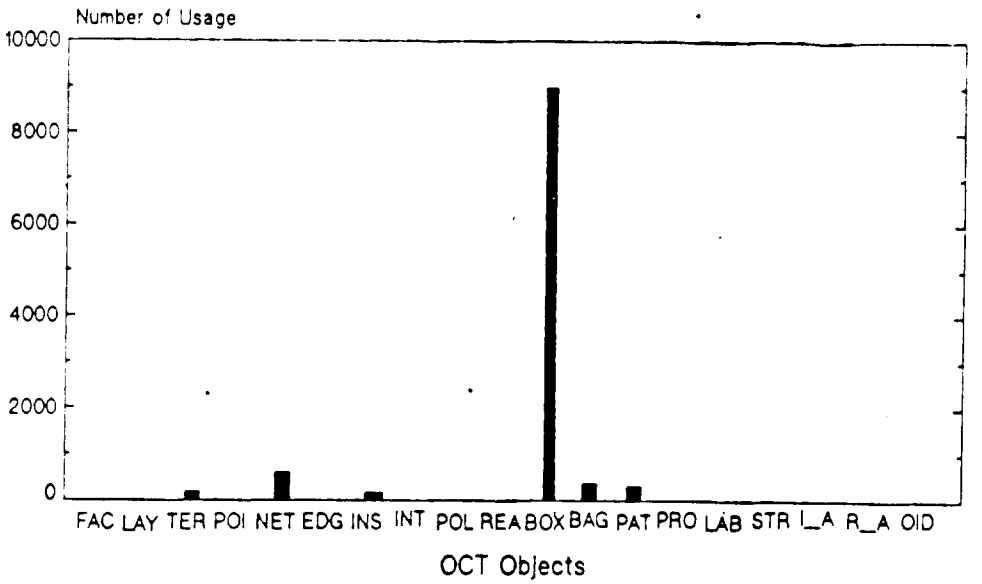

Figure 4-6 -- OCT Upward Access Object Usage Pattern

The X-axis represents the OCT object type: facet (FAC), terminal (TER), net (NET), instance (INS), polygon (POL), bounding box (BOX), path (PAT), label (LAB), property (PRO), bag (BAG), layer (LAY), point (POI), edge (EDG), integer (INT), real (REA), string (STR), real array (R_A), integer array (I_A), and object identifier (OID). The Y-axis represents the number of times the object type is used at run-time.

OCT object usage profile. Based on the measurements, we have the following observations:

(1) Different phases of the same application may have wide variation in the read/write ratio. However, within each phase, the read/write ratio is fairly stable.

(2) Most of the upward accesses have only one object returned and most of the OCT tools' downward access, except VEM, are dominated by the low structure density (0 to 3 objects).

(3) Supporting integrity constraint can reduce some physical I/Os while the overall read/write ratio is not dramatically affected.

These real-world object-oriented applications' access patterns are used to construct the simulation model described in the following Chapter.

CHAPTER 5

SIMULATION MODEL

5. Introduction

This Chapter presents a simulation model for engineering database application environments composed of several workstations and file servers. A user activity file, including data access function (open/close/read/write), data access mode (database and files), and data volumes, is constructed by using the previous Chapter's measurement results. Several questions we like to answer by running this simulation:

(1) How do different clustering and buffering algorithms affect the performance while varying the characteristics of applications?

(2) What is the relationship between the choice of the clustering algorithm and frequency parameters such as inheritance density and read/write ratio?

We used the Performance Analysis's Workbench System (PAWS) to construct our simulation model because PAWS supports a number of high level primitives; for example, various queuing disciplines and a set of detailed statistics outputs. PAWS also allows us to refine and enhance the model easily. In the following section, we discuss the workload and our model in terms of PAWS primitives. We summarize all the modeling parameters in Chapter 5.2.

5.1. The Engineering DB Model

We represent our model of engineering database interaction in the PAWS language which allows the user to simply declare the characteristics of the system being modeled instead of coding detailed simulation algorithms. An example of CPU node definition in the PAWS language is shown in Figure 5-1.

The simulation model consists of several interacting model blocks. Transactions representing user requests flow between these model blocks carrying information about the work units. The major blocks, shown in Figure 5-2, are:

(1) **Workstation Cluster:** a set of workstations representing interactive users and think times

(2) **Workload Definition:** workload characteristics

(3) **I/O subsystem:** a set of related components describing the I/O configuration

(4) **Buffer Manager:** the buffer management module

```
CPU
TYPE    SERVICE
QUANTITY 1
QD      RRFQ 10.0
REQUEST <BATCH,ALL>  HYPER(10.0, 14.1);
```

Figure 5-1 -- Example Node Definition in PAWS

Here, the name of the node is CPU, the node type is SERVICE, there is one server, the queueing discipline is round-robin with a fixed quantum of 10 time units, and all BATCH transaction regardless of phase request service times draw from the hyper-exponential distribution with mean 10.0 and standard deviation 14.1.

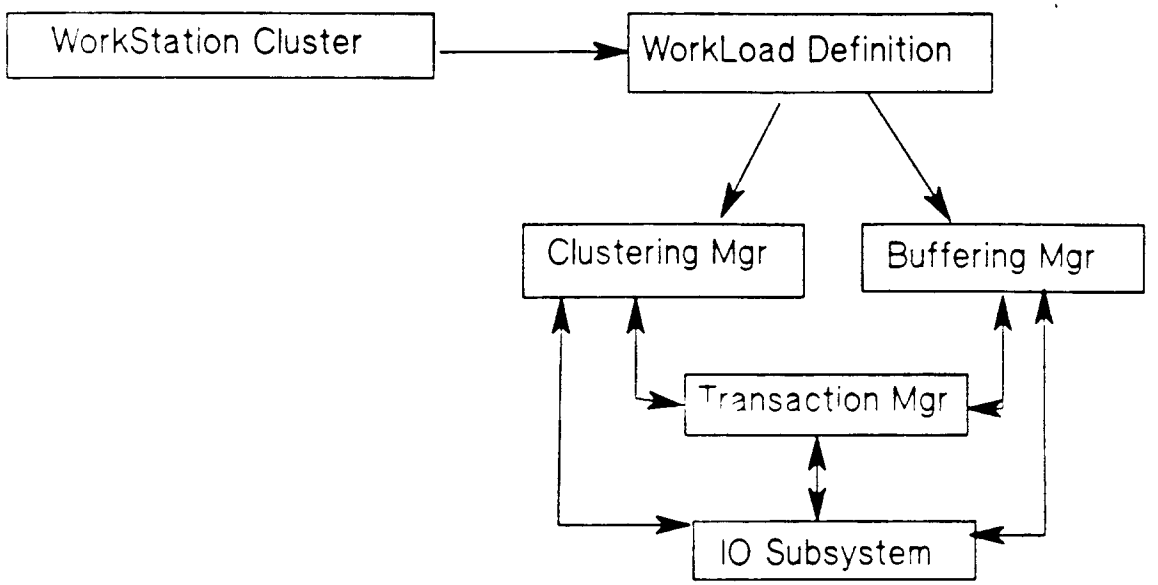

Figure 5-2 -- Simulation Model Overview

(5) **Cluster Manager:** various clustering algorithms management modules

(6) **CPU:** the processor

Figure 5-2 shows the relationship between these blocks. A transaction starts at a *workstation cluster* node and submits a request to the file server after a predefined think time. The request is defined in the *workload definition* node. If the request is a read operation, the buffer manager searches through the buffer pool and issues a physical I/O if needed. The buffer searching phase flushes out some dirty pages as well as some transaction log records. Therefore, in the worst case, a logical I/O, generated in the workload definition node, can be translated into zero to 3 physical I/Os; one I/O flushes the dirty page, one I/O logs the transaction, and one I/O brings in the data.

A few notes of our simulation model:

(1) We model not only the buffer pool activities but also the transaction logging details to obtain realistic log I/O rates. A log record is constructed based on the size of the newly created or modified object. A circular in-memory log buffer is used and log records are flushed when the circular log buffer is full.

(2) Object information such as object size, structural links with other objects, and containing page identifier, is maintained explicitly, allowing us to construct a realistic sample database used by all the buffering and clustering algorithms.

(3) Actual physical I/O activities are modeled through the *I/O subsystem* node. The I/O path length and the disk service time are also modeled by the system.

The detailed model description, shown in Figure 5-3, uses the following PAWS primitive and concepts:

(1) Each transaction is associated with a *category* and a *phase*. The category of a transaction is a name (denoted by any string of alphanumeric symbols) and it is permanent; a transaction has one unique category throughout its lifetime. The phase of transaction (denoted by an integer) may be changed as the transaction progresses. Two transactions of the same category and phase are processed identically at every node. Thus, the notion of category and phase is used to distinguish between various transactions.

(2) The *service* node, represented by a black circle, is used by a transaction to acquire active resources like terminal and CPU. A transaction arriving at a *service* node requests the use of a server for a specified length of time. If all servers at the node are busy, the transaction may have to wait in the node's queue until that transaction is selected for service. Transaction leaves the service node when its request is satisfied. A service node definition consists of a node name followed by a TYPE SERVICE

field; a QUANTITY field, which specifies the number of servers at the node; a QD field, which specifies the queueing discipline used to select transactions from the node's queue for service; and a REQUEST field, which specifies by category and phase with a queueing priority and a service time distribution.

(2) The *user* node, represented by a black box, is used by a transaction to invoke a user-written FORTRAN subroutine.[12] A transaction arriving at a user node requests that a specific user function be performed. The simulation is suspended and a user subroutine is invoked to perform the requested function. At the same time, all the system's global variables, the transaction's local variables, the transaction identifier and the present simulation time are passed into the corresponding user function. The requesting transaction will leave the user node when the user subroutine returns control to the PAWS system. In our simulation model, the user node is used to: 1) interact with the user activity file to define each individual transaction, 2) control clustering and buffering policies, 3) implement various clustering and buffering mechanisms, and 4) emulate the transaction manager function.

(3) The *fork* and *join* nodes, represented by triangles, are used by a transaction to create children transaction. The children transactions are created immediately and leave the fork node. The parent transaction remains behind at the fork node until its children transactions arrives at the same common join node, at which time the parent transaction replaces its children and proceeds from the join node. In our simulation model, the fork and join nodes are used to analyze the following services: 1) transaction management, 2) window management like terminal display, 3) virtual memory paging,

[12] FORTRAN is the only language supported by PAWS. Since PAWS is written in FORTRAN, this limitation simplifies the PAWS' implementation of *user* node.

and 4) disk I/O.

(4) The *compute* node, not shown in Figure 5-3, is used by a transaction to interrogate and alter variables and to report the statistics. Operations, such as assignment, sequence-control, and simulation-control, requested in compute node are performed immediately and the transaction leaves the compute node without any delay.

The *workload definition* node is intended to capture the workload characteristics. The OCT workload is defined by the distribution of OCT procedure calls and each OCT procedure call can have an average path length, I/O content, and lock request behavior.

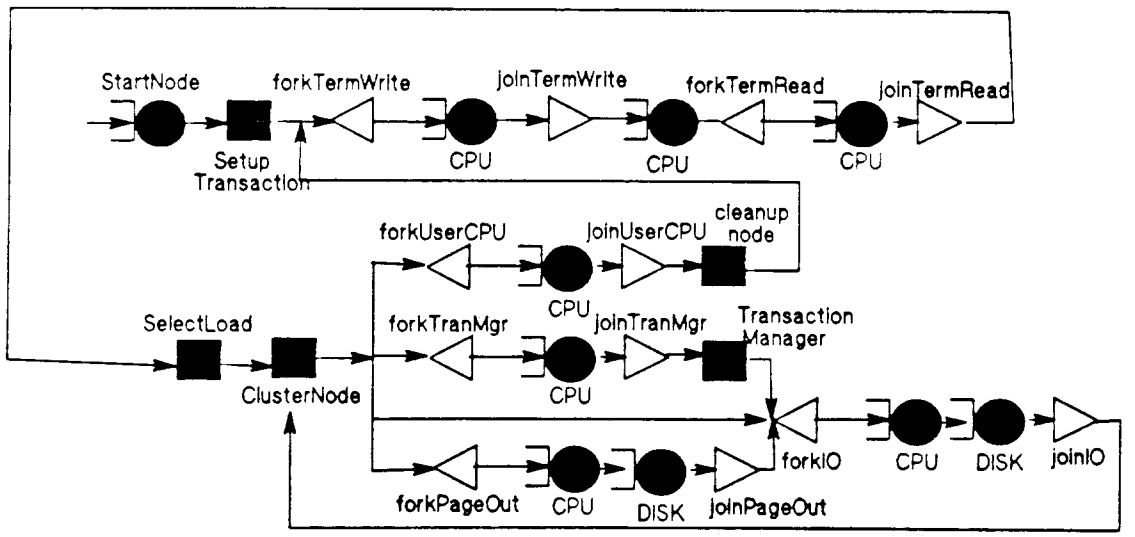

Figure 5-3 -- Simulation Model Detail

The fundamental units of recovery and concurrency control are the object and composite object. An object can be read by specifying a unique name or navigating along structural relationships from other objects. If the target object for a read operation is composite, several logical I/Os may occur to retrieve the complete object. Alternatively, the write operation would only create or update a single object. Based on our measurements, we choose that every tool invocation (user session) is composed of 100 to 10000 queries with various read/write ratios.[13]

Observing from our measurements, we decided that engineering design applications' procedure calls can be categorized into seven query types which are assigned to transactions in the workload definition phase: (1) Simple object lookup, (2) Component object retrieval, (3) Composite object retrieval, (4) Descendant version retrieval, (5) Ancestor version retrieval, (6) Corresponding objects retrieval, and (7) object insertion/deletion/updating. The probabilistic usage distribution of these query types is determined by the read/write ratio parameter described in Section 5.2.

In our simulation model, the checkin and checkout operations are modeled by these seven query types. For example, a transaction consisting of several component object retrievals can be viewed as a checkout operation. Similarly, a checkin operation is emulated by a transaction consisting of several object insertions and updates.

The complete PAWS implementation of the Engineering DB model is given in Appendix A, and a subset of the PAWS' output is listed in Appendix B.

---

[13] One tool invocation is viewed as one user session in our simulation model.

5.2. Parameters

Table 5-1 shows all the parameters modeled by the simulation model and their operating levels. They are divided into static parameters, which are fixed for all the simulation runs, and control parameters, which have various operating levels.

Static Parameters	Default Value
Database Size	300 Mbytes
Page Size	4 Kbytes
Number of active sessions	10
Think Time	4 seconds
Control Parameters	**Operating Levels**
Structure Density	low-3,med-5,high-10
Read-write Ratio	5,10,100
Clustering Policy	No_Cluster,Cluster_within_Buffer 2_IO_limit,10_IO_limit, No_limit
Page Splitting Policy	No_Split, Linear_Split, NP_Split
User Hint Policy	No_hint, User_hint
Buffer Replacement Policy	LRU, Context-sensitive,Random
Buffer Pool Size	100,1000,10000
Prefetch Policy	No_prefetch, Prefetch_within_buffer_pool Prefetch_within_Database

Table 5-1: Simulation Parameters

The operating levels of the control parameters are partially influenced by the structure density and read/write ratio observed in Chapter 3. The default values of the static parameters are influenced by the fact that the ultimate goal of this dissertation is to study the control parameters' impact on the overall system response time. The number of active sessions sharing the same database is assumed to be 10 since most of the CAD/CAM designs can be partitioned into small units shared by less than 10 engineers. We assumed the page size is 4 Kbytes and the number of disks of the server, not interested by this simulation study, is set to be 2 with a 30 ms mean disk access time. The database size is 300 Mbytes which is the maximum database size in Sun engineering application benchmark described in

Chapter 2 [CATT88]. The 4 seconds think time for each user we believe represents what is the average think time in interactive computer-aided design environments.

We studied eight control parameters to understand their impacts on the overall system response time. Out of these eight control parameters, *structure density* and *read/write ratio* determine the workload characteristics, *clustering policy*, *page splitting policy*, and *user hint policy* control the clustering policy, and *buffer replacement policy*, *buffer pool size*, *prefetch policy* define the buffering strategy. We based the operating levels of Structure Density on actual observation from OCT tools access patterns. "Low-3" means that every structural retrieval returns fewer than or equal to three component or composite objects. "Med-5" means that more than 3 but fewer than 10 objects are returned through structural retrieval, and "high-10" means that 10 or more objects are returned.

CHAPTER 6

SIMULATION RESULTS

6. Introduction

The control parameters listed in Table 5-1 are not independent from one another. For example, a higher structure density means more candidate pages to consider during the object placement phase. A higher read/write ratio means that the extra I/Os caused by the writers during the clustering phase may be amortized by the readers and, consequently, that the overall system response time improves.

In Section 6.1 we discuss the run-time clustering effects using alternative clustering policies under a fixed set of buffering control parameters. We then analyze the effects of the various page splitting policies on response time in Section 6.2. In Section 6.3 we examine the effectiveness of *User hints*. After discussing various buffer control parameters effects on response time in Section 6.4, we complete our study with an overall effect analysis in Section 6.5. For every set of control parameter settings, at least ten simulation runs are used to produce the average response time.

6.1. Run-Time Clustering Effect

The operating levels of buffering control parameters used in this clustering policy effects study are: No prefetch, 1000 buffers,[14] and LRU buffer replacement policy. The *No Split* page splitting and *No User hint* policies are also used, insuring that when the preferred candidate page is full, the storage manager chooses the next best candidate page with

[14] This is the standard configuration in Sun Benchmark.

enough space.

Figure 6-1 shows how various clustering policies affect system response time under different workloads. Some key observations are:

(1) Run-time clustering always improves overall system response time. When both the structure density and read/write ratio are high, the response time is improved by 200%.

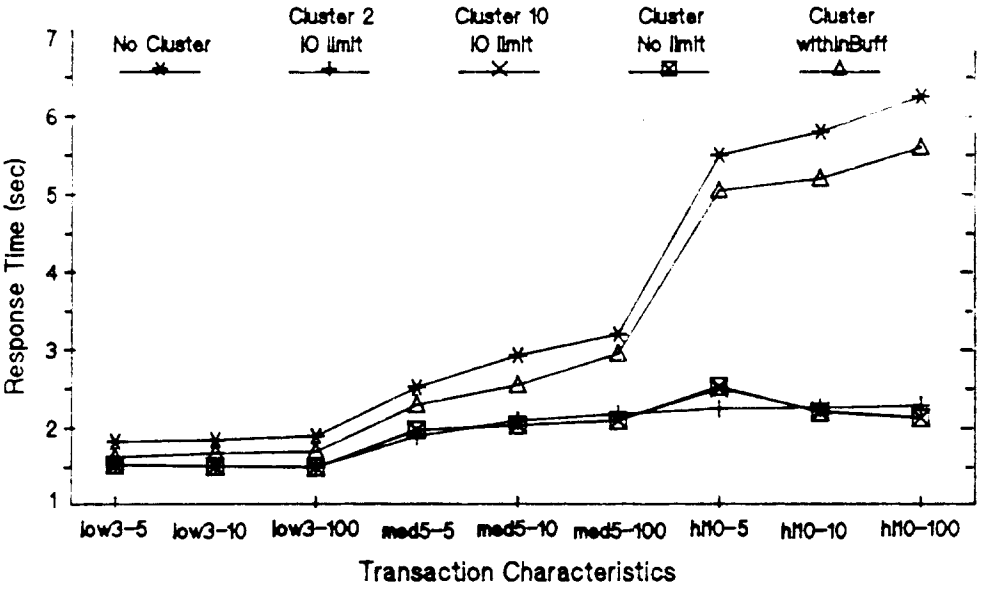

Figure 6-1 -- Clustering Effects Analysis

The buffering control parameters are fixed to 1) No prefetch, 2) 1000 buffers and 3) LRU buffer replacement policy. Five different clustering policies are studied: 1) No clustering, 2) Clustering within buffer pool, 3) Clustering with a 2 I/O limitation, 4) Clustering with a 10 I/O limitation, and 5) Clustering without I/O limitation. The X axis represents various transaction characteristics. "Low3-5" means the transaction has a low structure density fewer than or equal to 3 and read/write ratio 5. "Hi10-100" means the transaction has a high structure density which is greater than or equal to 10 and read/write ratio 100.

(2) Setting I/O limits on potential candidate pages search is valid. When the structure density is low, *clustering with 2 I/O limitation* performs better than or comparably to *clustering without I/O limitation*.

(3) *Clustering within buffer pool* performs reasonably well when the structure density is low, but degrades to the *No_Clustering* case when the structure density is high. The buffer hit ratio also affects the performance of *clustering within buffer pool*. When the hit ratio is low, *clustering within buffer pool*'s performance is close to *No_Clustering*. However, its performance improves when a better buffering policy is used (Refer to Chapter 6.4)

(4) In general, the ordering of the clustering strategies is not changed under various read/write ratios. Clustering with 2, 10, or unlimited I/Os always performs better than *clustering within buffer pool* which is better than the *No_Clustering* case.

However, what is the point in read/write ratio where the clustering mechanisms begin to become effective, i.e., the point at which the clustering mechanisms lead to similar response as the no clustering case? We compare *No_Clustering* with *clustering without any I/O limitation*. The results are summarized in Table 6-1 under alternative prefetching policy, page splitting policy, and various structure densities. Different structure densities have different break-even points, due to the amount of logical I/Os caused by writers during the clustering phase. More detailed analysis on prefetching and page splitting policies is in Section 6.2 and 6.4.

	No Prefetch		Prefetch	
	No Split	Split	No Split	Split
low-3	3.0	3.2	2.8	2.9
med-5	3.6	3.7	3.4	3.4
high-10	4.3	4.5	4.0	4.1

Table 6-1: Read-write ratio break-even points

Figures 6-2, 6-3 and 6-4 show both the structure density effect and the prefetching effect on response time when the read/write ratio is fixed. Notice that the ordering of the clustering strategies is not changed under alternative prefetching policies.[15] In all cases, the response time rises sharply between medium structure density and high structure density when no clustering is done. Since a composite object retrieval may trigger 10 or more logical I/Os under high structure density and that these are less likely to hit in the buffer pool, the increasing physical I/Os result in a higher response time. However, the response time rises slowly from a low structure density to a high structure density when any of the clustering mechanisms except *clustering within buffer pool* is used. Both the *clustering within buffer pool* and the *No_Clustering* cases have sharp rise of the response time from a low structure density to a high structure density. The low variation of response time over different structure densities is critical, especially when applications' dynamic structure densities are not fully predictable.

Clustering within buffer pool cannot perform well if the buffer pool hit ratio is low during the candidate page searching phase. By tracing the buffer pool usage, we found that the native LRU replacement policy frequently overlays the potential candidate page for new object placement and decreases the hit ratio. The buffer manager, if it understands the

[15] More detailed analysis on prefetching effect is in Section 6.4.

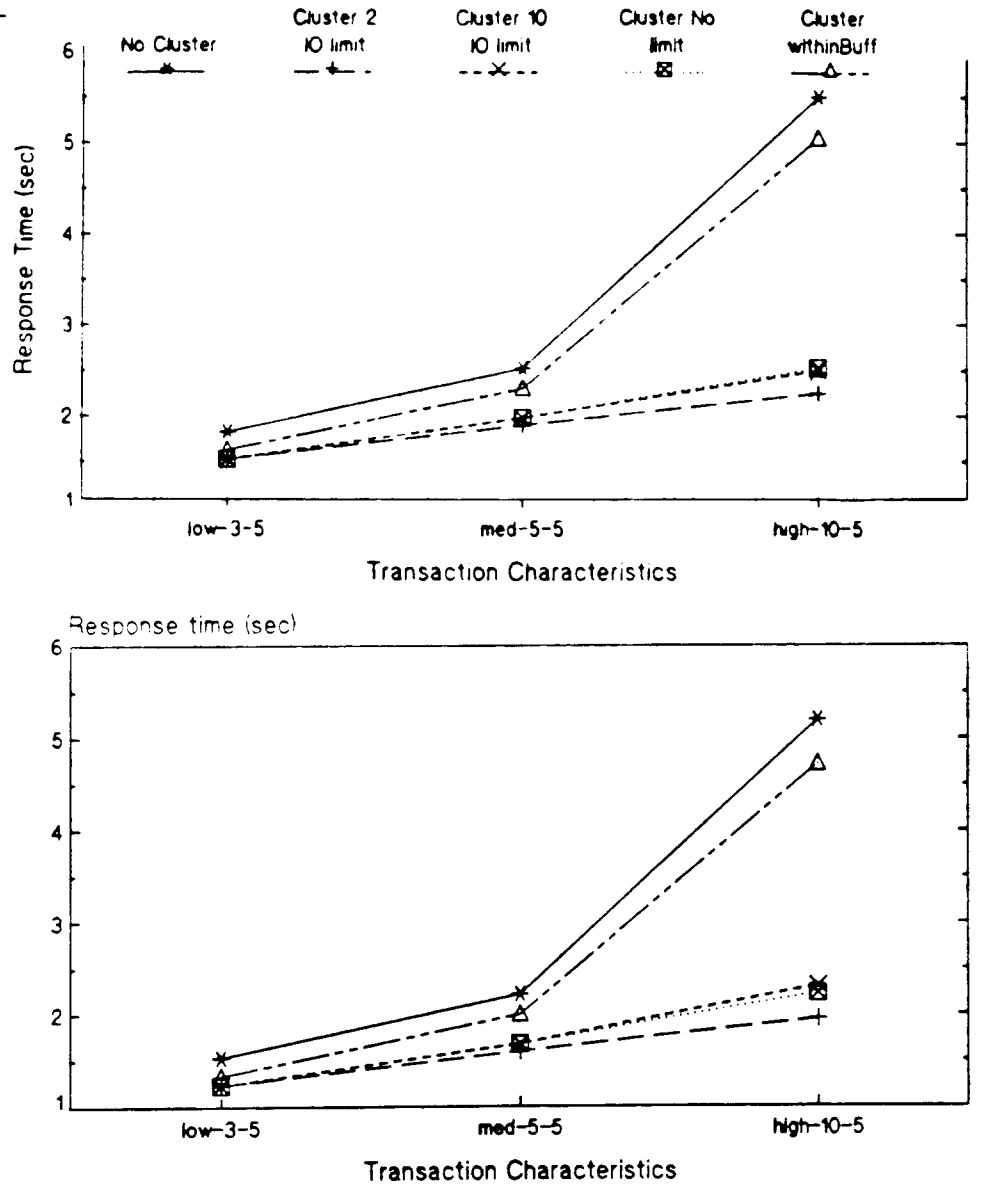

Figure 6-2 -- Clustering Effect Under R/W ratio 5

The buffering control parameters are fixed to 1) No prefetch for the top graph and prefetching for the bottom one, 2) 1000 buffers and 3) LRU buffer replacement policy. The read/write ratio is set to 5. Five different clustering policies are studied: 1) No clustering, 2) Clustering within buffer pool, 3) Clustering with a 2 I/O limitation, 4) Clustering with a 10 I/O limitation, and 5) Clustering without I/O limitation. The X axis represents various transaction characteristics. "Low-3-5" means the transaction has a low structure density which is less than or equal to 3 and read/write ratio 5. "High-10-5" means the transaction has a high structure density which is greater than or equal to 10 and read/write ratio 5.

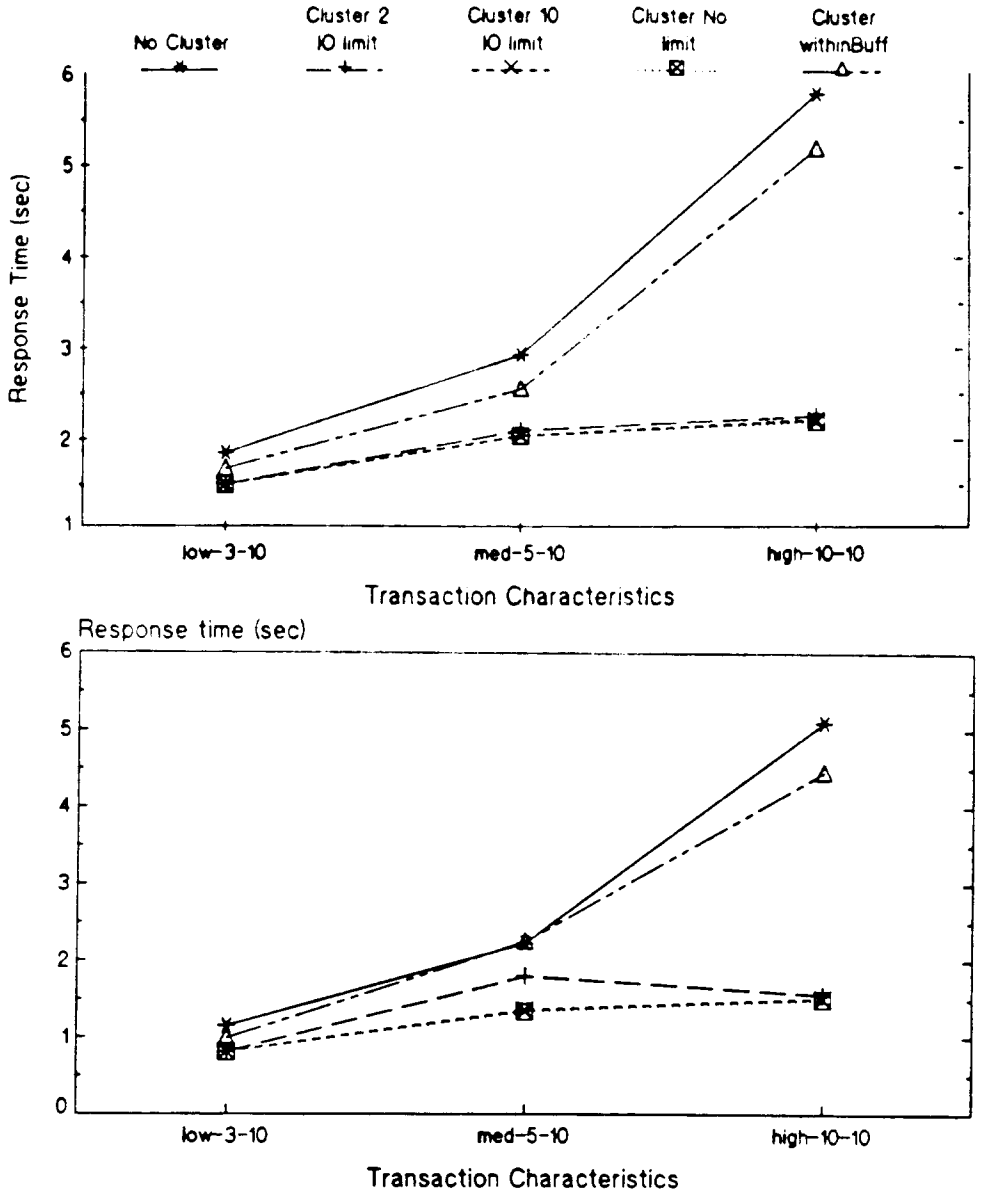

Figure 6-3 -- Clustering Effect Under R/W ratio 10

The buffering control parameters are fixed to 1) No prefetch for the top graph and prefetching for the bottom one, 2) 1000 buffers and 3) LRU buffer replacement policy. The read/write ratio is set to 10. Five different clustering policies are studied: 1) No clustering, 2) Clustering within buffer pool, 3) Clustering with a 2 I/O limitation, 4) Clustering with a 10 I/O limitation, and 5) Clustering without I/O limitation. The X axis represents various transaction characteristics. "Low-3-10" means the transaction has a low structure density which is less than or equal to 3 and read/write ratio 10. "High-10-10" means the transaction has a high structuredensity which is greater than or equal to 10 and read/write ratio 10.

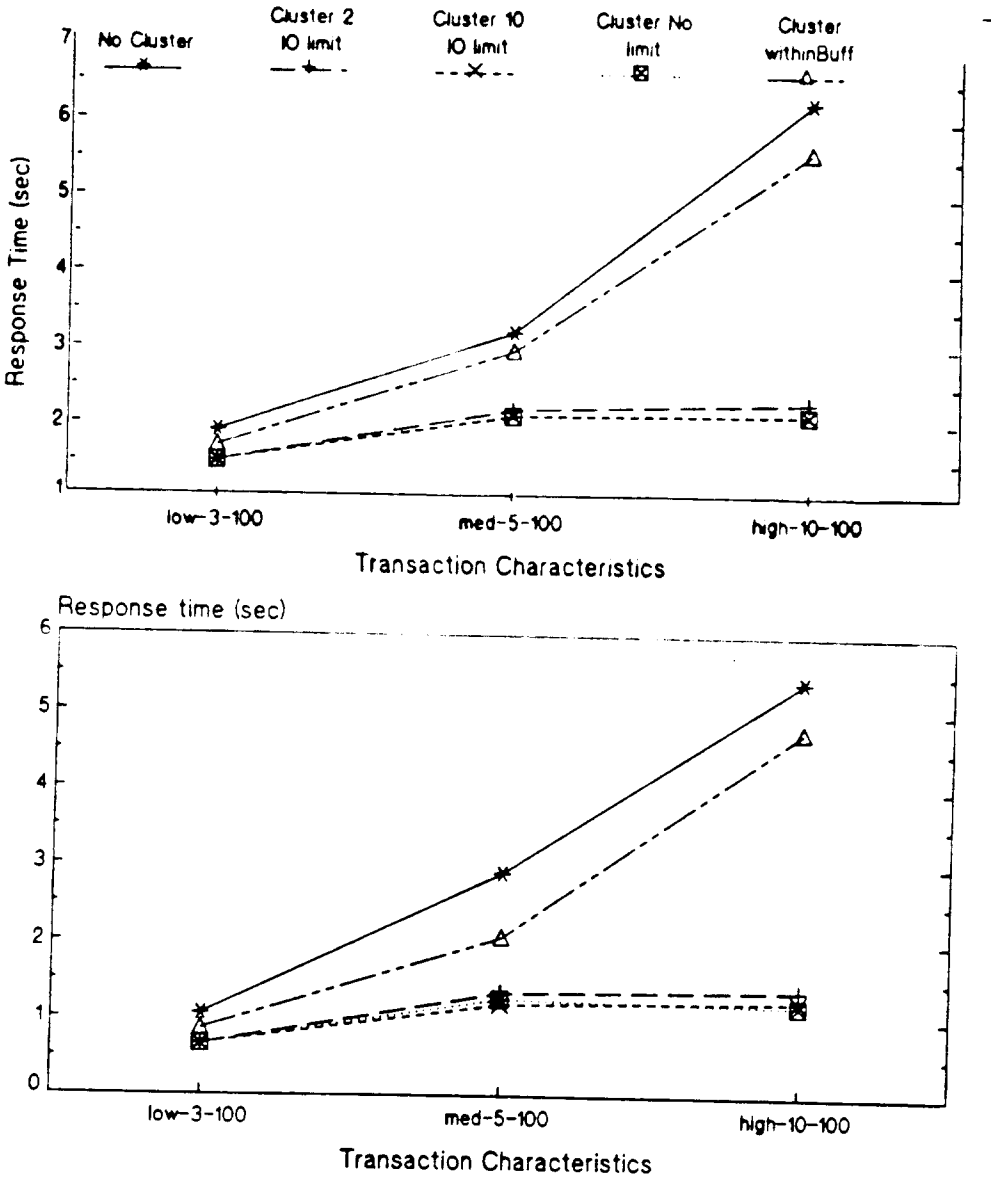

Figure 6-4 -- Clustering Effect Under R/W ratio 100

The buffering control parameters are fixed to 1) No prefetch for the top graph and prefetching for the bottom one, 2) 1000 buffers and 3) LRU buffer replacement policy. The read/write ratio is set to 100. Five different clustering policies are studied: 1) No clustering, 2) Clustering within buffer pool, 3) Clustering with a 2 I/O limitation, 4) Clustering with a 10 I/O limitation, and 5) Clustering without I/O limitation. The X axis represents various transaction characteristics. "Low-3-100" means the transaction has a low structure density which is less than or equal to 3 and read/write ratio 100. "High-10-100" means the transaction has a high structuredensity which is greater than or equal to 10 and read/write ratio 100.

relationships among objects, may increase both the priority of these pages and the hit ratio.

We will discuss the various buffering policies in Chapter 6.4.

When the read/write ratio is low, *clustering with a 2 I/Os limitation* provides the best response time in all structure densities, as shown in Figure 6-2. For low structure density, *clustering with a 2 I/Os limitation* has the same response time as *clustering with no I/O limitation*. Notice that a low structure density implies fewer candidate pages to choose and fewer logical I/Os needed during the clustering phase. As the structure density increases, the number of I/Os caused by searching candidate pages also increases. Such extra I/Os introduced in the clustering phase cannot be amortized by readers when the read/write ratio is low. Therefore, *clustering without I/O limitation* performs worse than *clustering with a 2 I/O limitation* in high structure densities.

Figure 6-3 shows that clustering with a limitation of 10 I/Os has the same response time as clustering with no I/O limitation for a medium structure density. Since the I/O limitation is larger than the maximum possible structure density, 10 I/Os behaves like no I/O limitation.

Clustering without I/O limitation performs consistently better than other clustering mechanisms when the read/write ratio is 100, as shown in Figure 6-4. Selecting a clustering mechanism based on the read/write ratio at run-time affords the best response time of both, setting either small I/O limitation or no I/O limitation at all.

If the transaction logging is done at the physical level, using clustering may reduce the number of I/Os in the transaction logging phase. This is shown in Figure 6-5 where we compare the number of transaction logging I/Os between no clustering and clustering without I/O limitation under different structure densities. When multiple updates occur on the same page within a transaction, the log manager needs to flush the original page only once. Since related objects are clustered on the same page, the probability of having

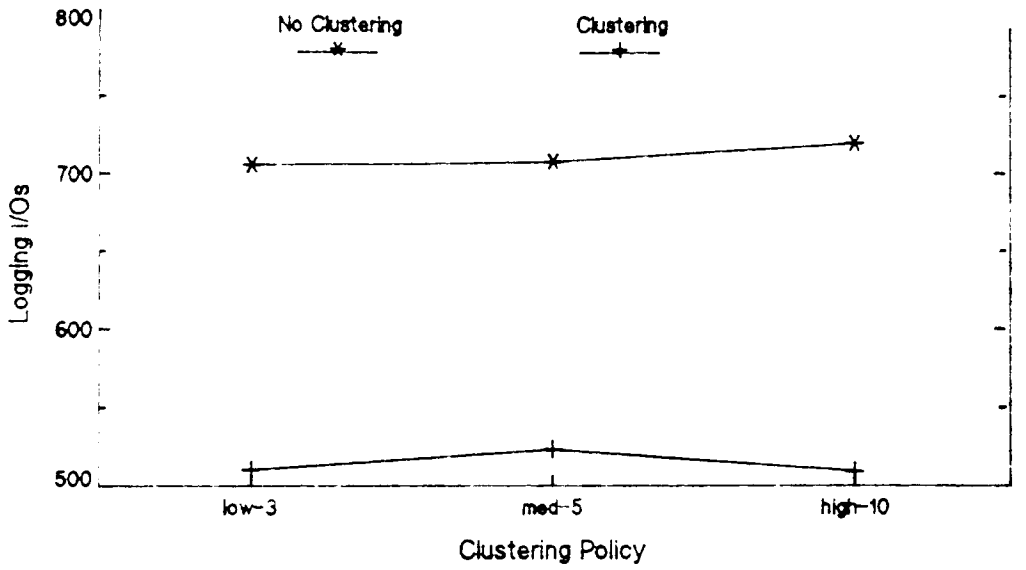

Figure 6-5 -- Clustering Effect on Transaction I/Os

The read/write ratio is fixed at 5 and the buffering control parameters are 1) no prefetch, 2) 1000 buffers and 3) LRU buffer replacement policy. The Y axis represents the number of I/Os caused by transaction logging, and the X axis is the structure density.

multiple updates on the same page within a transaction increases, thus reducing the number of physical logging I/Os. Notice that the number of logging I/Os under high structure density is smaller than the one under medium structure density when clustering is used. After examining the log records from the simulation runs, we conclude that such slight decreasing of logging I/Os is caused by the effective clustering policy.

One interesting thing we learned from the access patterns of the OCT tools is that the read/write ratio may vary across different phases of the same application. Therefore, we wanted to find out how the read/write ratio affects response time when the structure density is fixed.

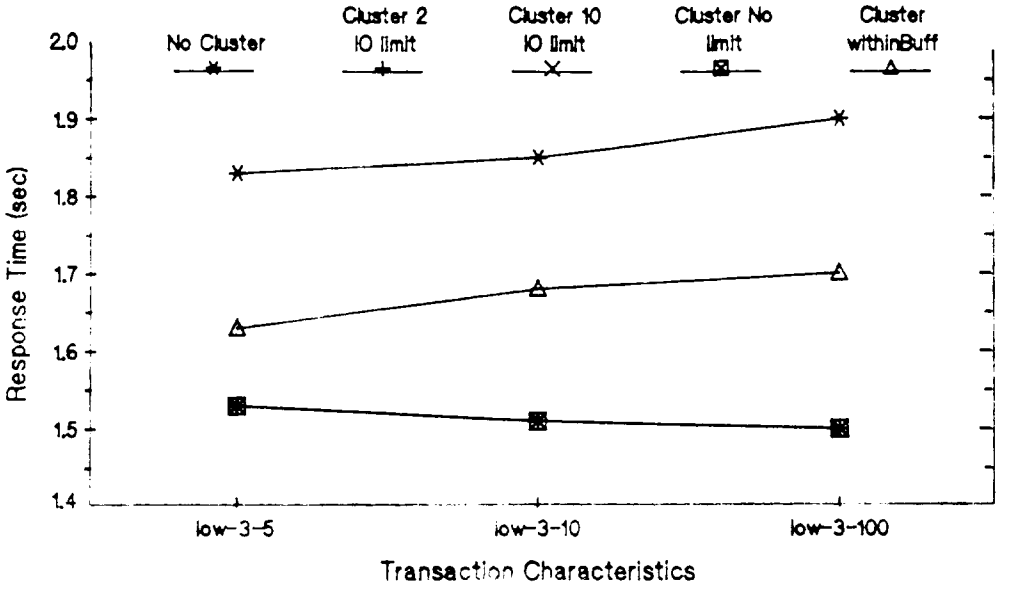

Figure 6-6 -- Clustering Effect Under Low Structure Density

The X axis represents various transaction characteristics. "Low-3-5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5.

Figure 6-6 shows that any of the clustering mechanisms perform better than no clustering in the low structure density case. Clustering with and without I/O limitation perform similarly, and the variation of response time is very small. Therefore, for high read/write ratio applications having low structure density, *clustering with a 2 I/O limitation* may be the best choice. For applications having medium structure density, as shown in Figure 6-7, *clustering without I/O limitation* performs the best when the applications' read/write ratio is greater than 10. Notice that the response time of *clustering without I/O limitation* case changes little for all read/write ratios. Such a stable response time may be required by some real-time applications.

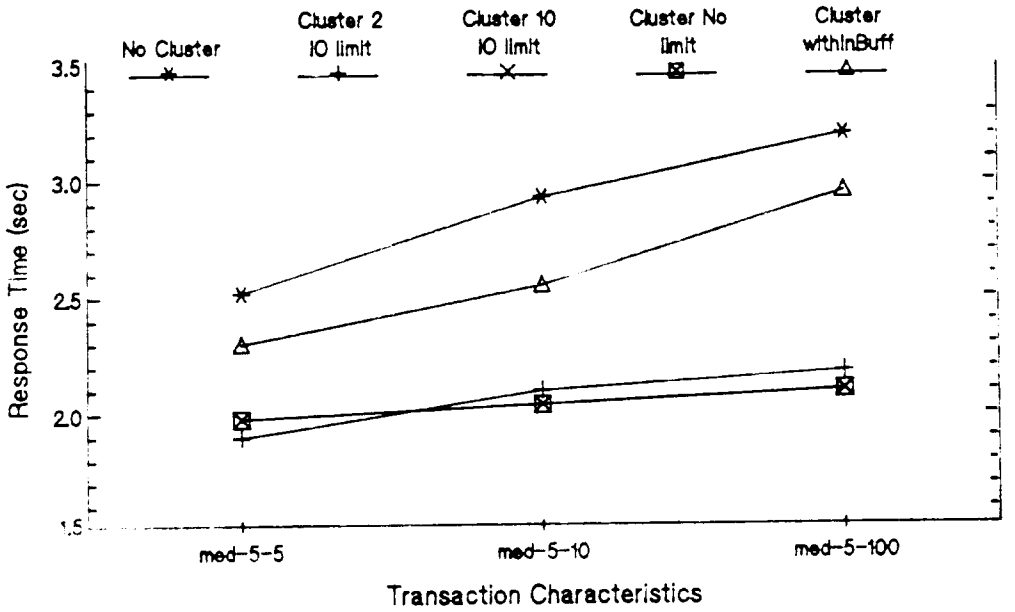

Figure 6-7 -- Clustering Effect Under Medium Structure Density

The X axis represents various transaction characteristics. "Med-5-5" means the transaction has a medium structure density which is greater than 3 but less than 10 and a read/write ratio of 5.

Figure 6-8 shows the clustering effect on response time when the structure density is high. Notice that the difference between the *clustering within buffer* case and the other clustering mechanisms becomes larger. This is due to the lack of potential candidate pages in the buffer pool, which reduces the effectiveness of run-time clustering.

Factor Interaction Analysis

Two control parameters, say A and B, are said to "interact" if the effect of A varies as a function of B. They can be r.5epresented on an X-Y diagram where the X axis represents the different operating levels and the Y axis represents the effect (the response time). If two lines are parallel, there is no interaction between the corresponding control

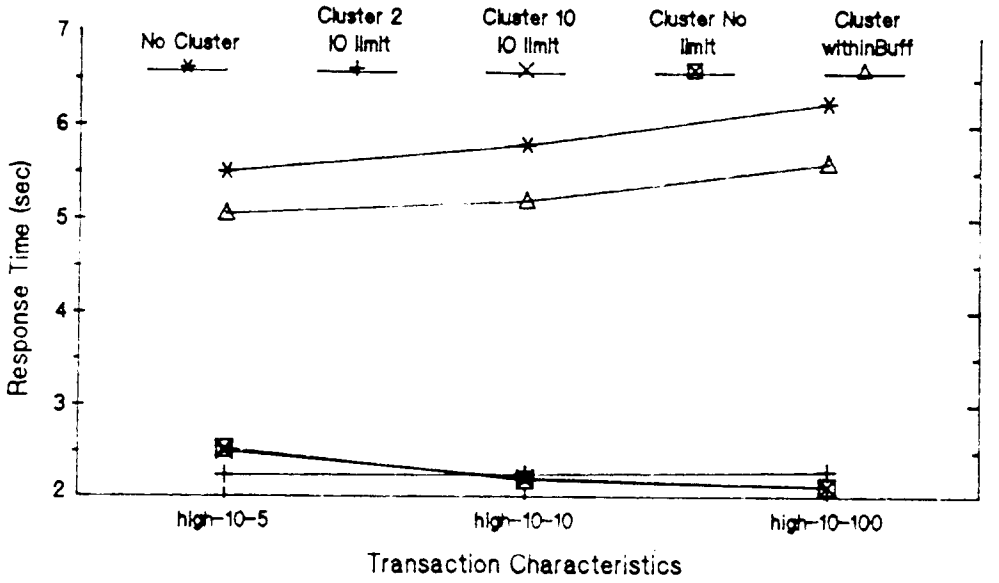

Figure 6-8 -- Clustering Effect Under High Structure Density

The X axis represents various transaction characteristics. "High-10-5" means the transaction has a high structuredensity which is greater than equal to 10 and read/write ratio 5.

parameters. For intersected lines, we know there is a strong interaction between control parameters. If two lines do not intersect in the parameter range but are also not parallel, we say there is a minor interaction between the control parameters. The corresponding factor interaction analysis graphs are shown in Figure 6-9.

Figure 6-10 shows the interaction between the *clustering policy* and the *read/write ratio*. The Y-axis value (the response time), is calculated from averaging all the experiments which have the same control parameter under analysis. For example, in Figure 6-4, we have three data points for *clustering without I/O limitation* and *read/write ratio 100*: low-100, med-100, and hi-100. Averaging these three data points, we get one Y-axis value in the interaction graph. Similarly, we can derive the other three response time values and draw

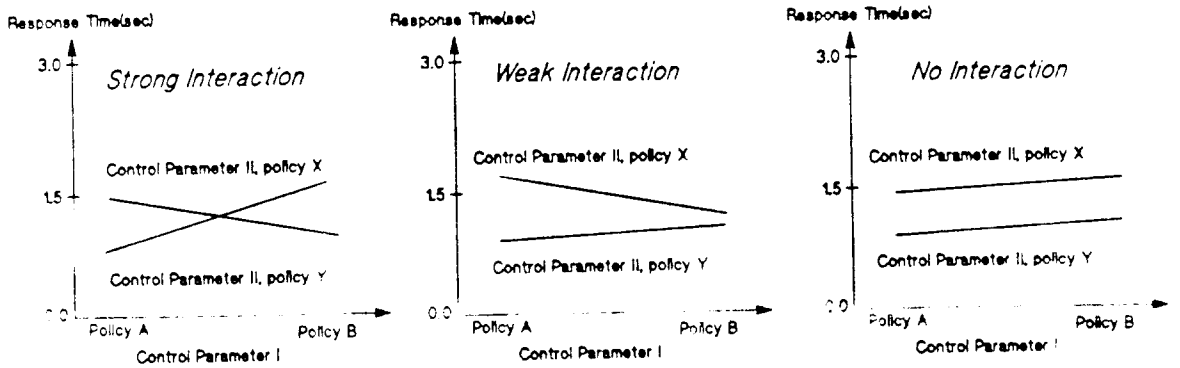

Figure 6-9 -- Factor Interaction Analysis Graph Samples

two lines in the graph. Since these two lines in Figure 6-10 do not intersect, but are not parallel either, we conclude that the *clustering policy* and the *read/write ratio* have minor interaction between them. That is, the clustering policy cannot be set without understanding the read/write ratio in application environments.

Figure 6-11 shows the interaction graphs between the *clustering policy* and the *structure density*. The left graph shows minor interaction when both clustering policies are using *run-time clustering*. This is consistent with the earlier observation that when the structure density is low, *clustering with 2 I/O limitation* performs better than or comparably to *clustering without I/O limitation*. Further, when the clustering policy is changed to either *run-time clustering* or *no clustering*, there is also a minor interaction between the *clustering policy* and the *structure density*, as shown in the right interaction graph in Figure 6-11. Notice that the right graph has a larger slope than the left one. This is due to the run-time clustering

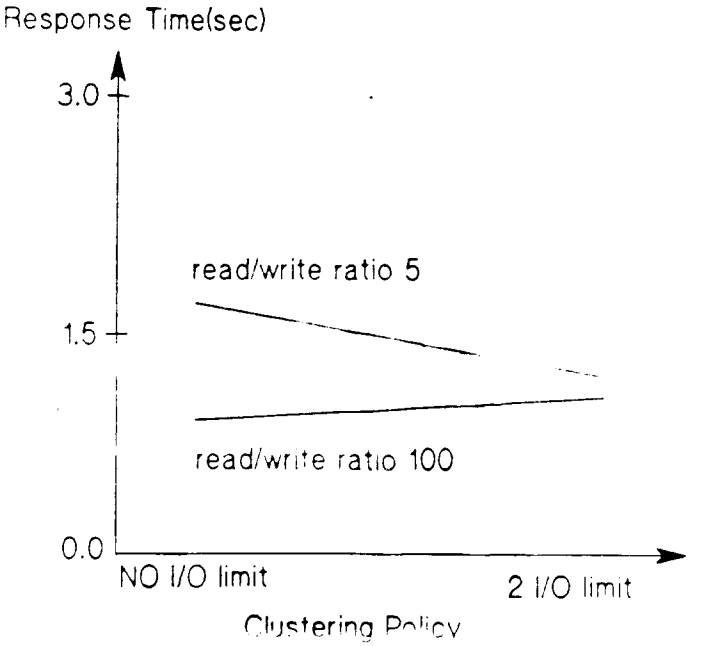

Figure 6-10 -- Clustering vs. Read/Write ratio Interaction Analysis Graph

The X-axis represents the different clustering policies: clustering without I/O limitation and with a 2 I/O limitation. The Y-axis represent the response time for high read/write ratio and low read/write ratio under these two clustering policies. Two lines in parallel imply no interaction. Two lines intersecting mean major interaction. In this graph, we have a minor interaction case.

effect becoming more significant when the structure density is changed. Finally, Figure 6-12 shows that there is no interaction between the *structure density* and the *read/write ratio*.

6.2. Page Splitting Effect

When the chosen candidate page overflows with the newly inserted object, the system needs either to split the target page or to pick the next best candidate page. To split the target page, a page splitting algorithm is invoked, a new page is allocated, both the target page and the new page are modified, and the changes are logged by the system. Page splitting requires one extra I/O than no splitting to flush the newly allocated page and one extra

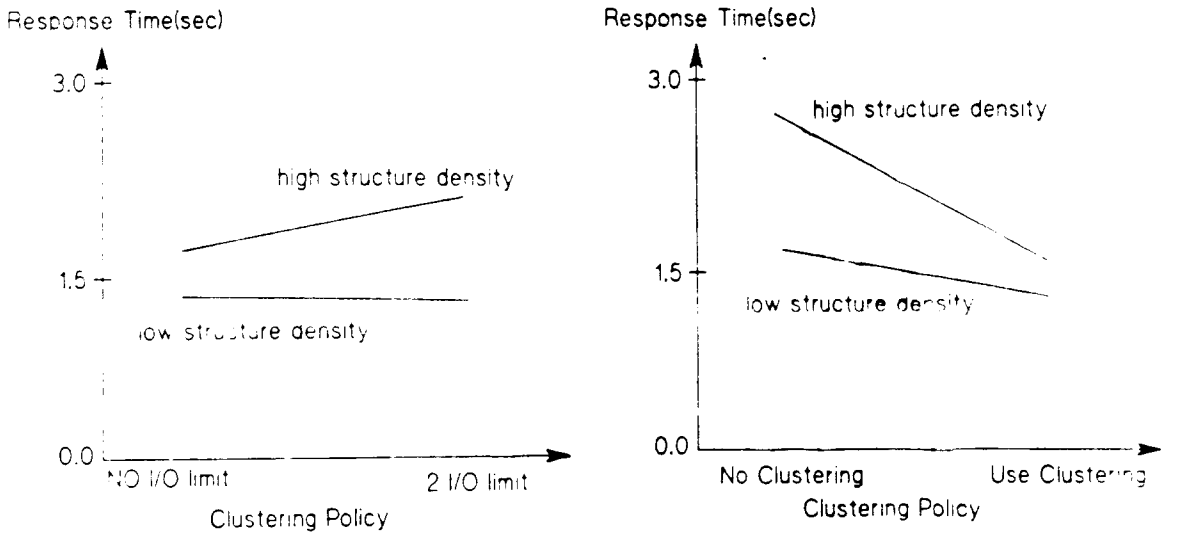

Figure 6-11 -- Clustering vs. Structure Density Interaction Analysis Graph

The X-axis represents the different clustering policies: clustering without I/O limitation and with a 2 I/O limitation for the left graph, no clustering and use clustering for the right graph. The Y-axis represents the response time for high and low structure densities under these different clustering policies. Both of these interaction analysis graphs represent minor interaction cases.

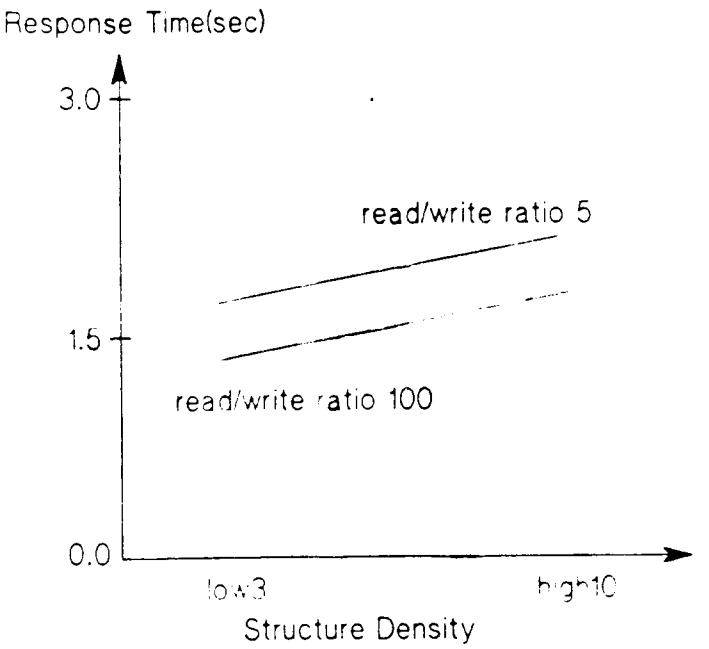

Figure 6-12 -- Structure Density vs. Read/Write Ratio Interaction Analysis Graph

The X-axis represents the different structure densities: low and high. The Y-axis represents the response time for high and low read/write ratios under different structure densities. Two lines in parallel imply no interaction. Two lines intersecting mean major interaction. In this graph, we have a **no** interaction case.

log record that may trigger an I/O.[16] Moreover, the page splitting algorithm needs extra CPU time, and the new page allocation may cause buffer pool contention.

As mentioned in Chapter 3, we have evaluated two different page splitting algorithms in the simulation model: *Linear Split* and *NP Split*. The operating levels of buffering control parameters used in this page splitting effects study are: No prefetch, 1000 buffers, and LRU buffer replacement algorithm. No *User hints* are used.

Figure 6-13 shows the effect of various page splitting algorithms when the clustering policy is *clustering without I/O limitation*. When the read/write ratio is low, *No Split*

[16] Physical logging is assumed here.

performs better than either splitting case. However, *Linear Split* provides the best response time when both the read/write ratio and the structure density are high. Both *NP Split* and *Linear Split* perform similarly when structure density is low, because the number of arcs in the dependency graph is small, and the minor difference between the NP-complete solution and the linear solution is offset by other factors.

The objective of the page splitting algorithm is to minimize the expected access cost resulting from the page split. Ideally, the object partition having the minimum access cost should have the best response time. However, due to the extra I/Os and CPU time caused

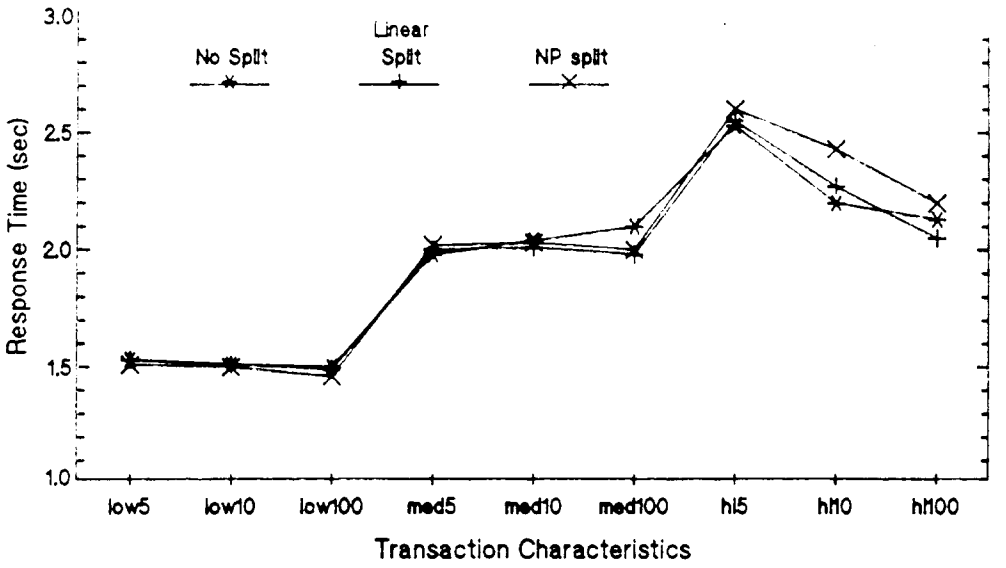

Figure 6-13 -- Page Splitting Effects Analysis

The clustering policy is clustering without I/O limitation and no user hints are provided. The operating level of buffering control parameters are no prefetch, 1000 buffers, and LRU buffer replacement policy. The solid line represents the response time for the no page splitting case. The dotted line with plus sign is the response time when the linear page splitting mechanism is applied. The dotted line with cross sign is the NP split case.

by the minimization process, the page splitting algorithm finding the minimum cost partition of objects may not perform best overall. Figure 6-14 shows the total cost difference between *NP Split* and *Linear Split* under different transaction characteristics. Notice that although the *NP Split* algorithm finds the minimum access cost partition of objects, its overall response time is worse than the *Linear Split* under high structure density.

Factor Interaction Analysis

Figures 6-15, 6-16, and 6-17 show the interaction graphs for the *page splitting policy*. Both the *structure density* and the *read/write ratio* have minor interactions with the *page*

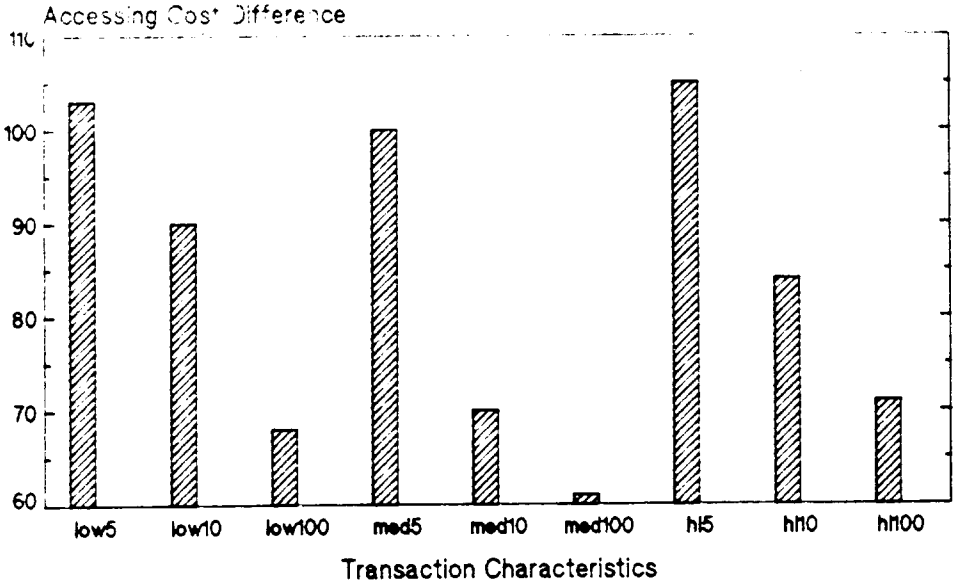

Figure 6-14 -- Cost Difference between NP Split and Linear Split

The X axis represents various transaction characteristics. "Low10" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 10. "Hi100" means a high structure density and a read/write ratio of 100. Accessing cost difference, represented by the Y axis, is measured in terms of *logical I/O*.

splitting policy. This agrees with the observation we made in the previous section that *Linear Split* provides the best response time when both the read/write ratio and the structure density are high.

But, why do we see no interaction between the *page splitting policy* and the *clustering policy* in Figure 6-17? Both policies try to improve the referential locality of objects by either splitting page or retrieving more candidate pages from disk. However, each policy is applied in different phases of object write operation: the clustering policy determines where and when to retrieve the candidate page whereas the page splitting policy is invoked once the candidate page is chosen. Therefore, no interaction between them is possible at run-time.

6.3. Impact of User Hints

In this section, we study the user hint effect under different buffering control parameters and various page splitting policies. The operating level of clustering policy is fixed at *clustering without I/O limitation* and the buffer pool size is 1000.

6.3.1. User Hint Effect Under Prefetch within Database

Figures 6-18, 6-19, and 6-20 show the effects of user hints under different buffer replacement algorithms and various page splitting policies when the *Prefetching within database* is used. In most cases, *User hint* improves overall system response time. The best improvement, of 25%, occurs when the structure density is high and the read/write ratio is 10, with the *Random* buffer replacement algorithm and the *No page splitting* policy. However, when the buffer replacement algorithm is *Context-sensitive*, user hint effect has only 1% to 4% improvement, as shown in Figures 6-21, 6-22, and 6-23. Several factors contribute to the small size of this improvement:

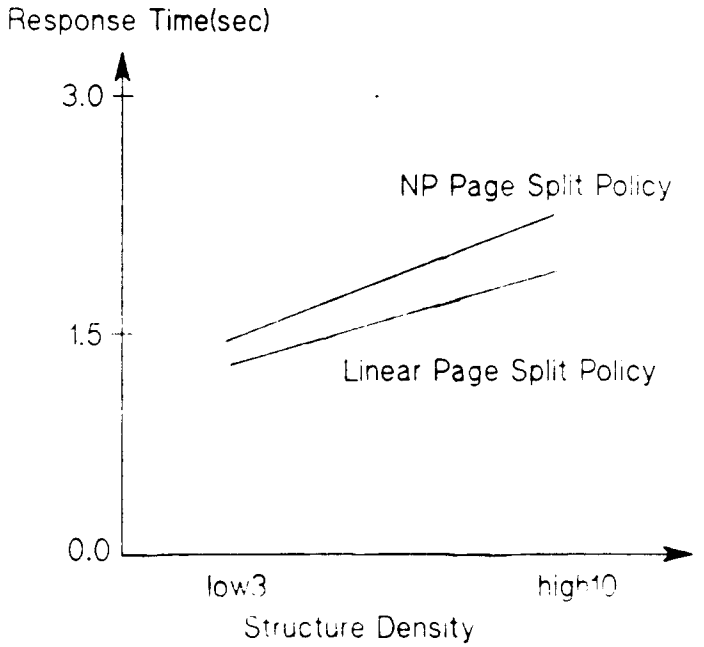

Figure 6-15 -- Structure Density vs. Page Splitting Interaction Analysis Graph

The X-axis represents the different structure density: low and high. The Y-axis represents the response time for Linear Split and NP-Split under these structure densities. In this graph, we have a **minor** interaction case.

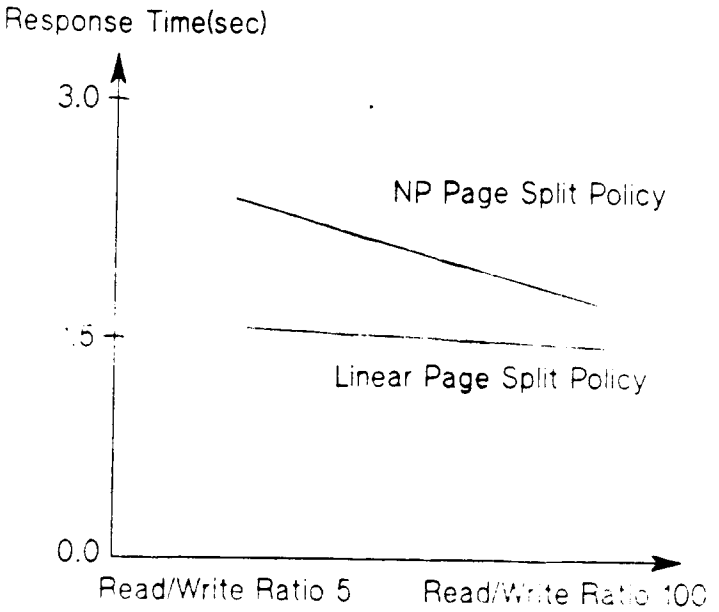

Figure 6-16 -- Read/Write Ratio vs. Page Splitting Interaction Analysis Graph

The X-axis represents the different read/write ratios: 5 and 100. The Y-axis represents the response time for Linear-Split and NP-Split under these different read/write ratios. This graph shows a **minor** interaction case.

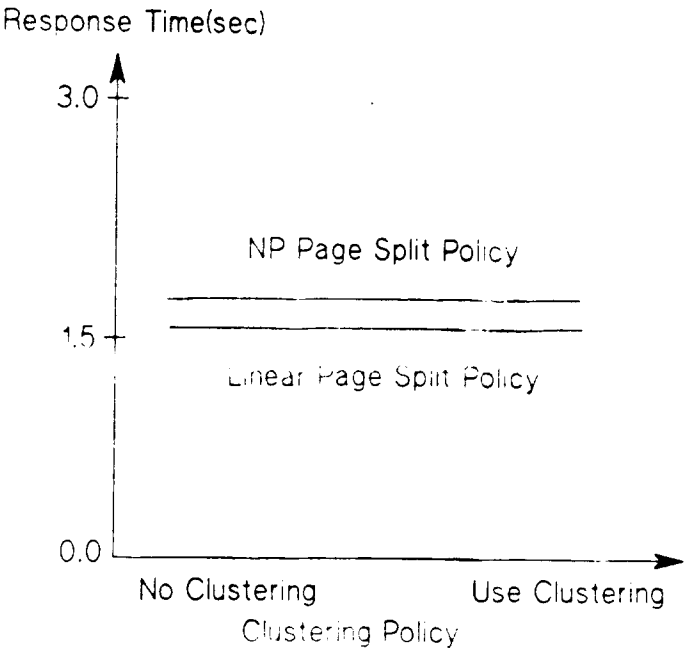

Figure 6-17 -- Clustering Policy vs. Page Splitting Policy Interaction Analysis Graph

The X-axis represents the different clustering polices: clustering without I/O limitation and no clustering. The Y-axis represents the response time for Linear-Split and NP-Split under these different clustering policies. In this graph, we have a no interaction case.

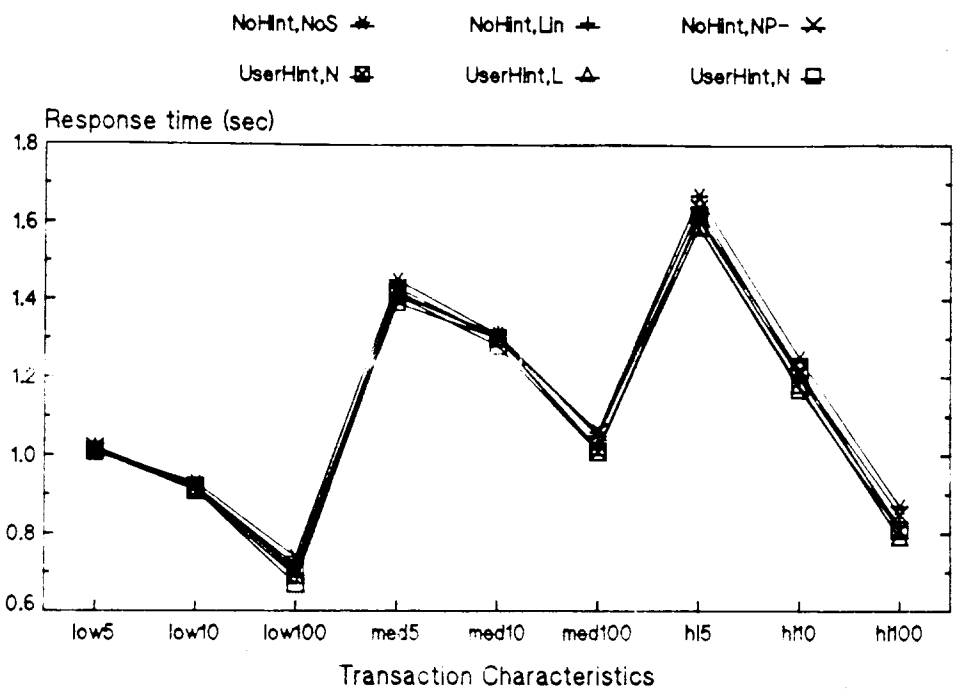

Figure 6-18 -- User Hint Effect Under Context-sensitive Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within database, 2) 1000 buffers and 3) Context-sensitive buffer replacement policy. The clustering control parameter is set to *clustering without I/O limitation*. All the user hint policies and page splitting policies are studied under this environment: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio 5. "Hi100" means the transaction has a high structuredensity which is greater than or equal to 10 and a read/write ratio 100.

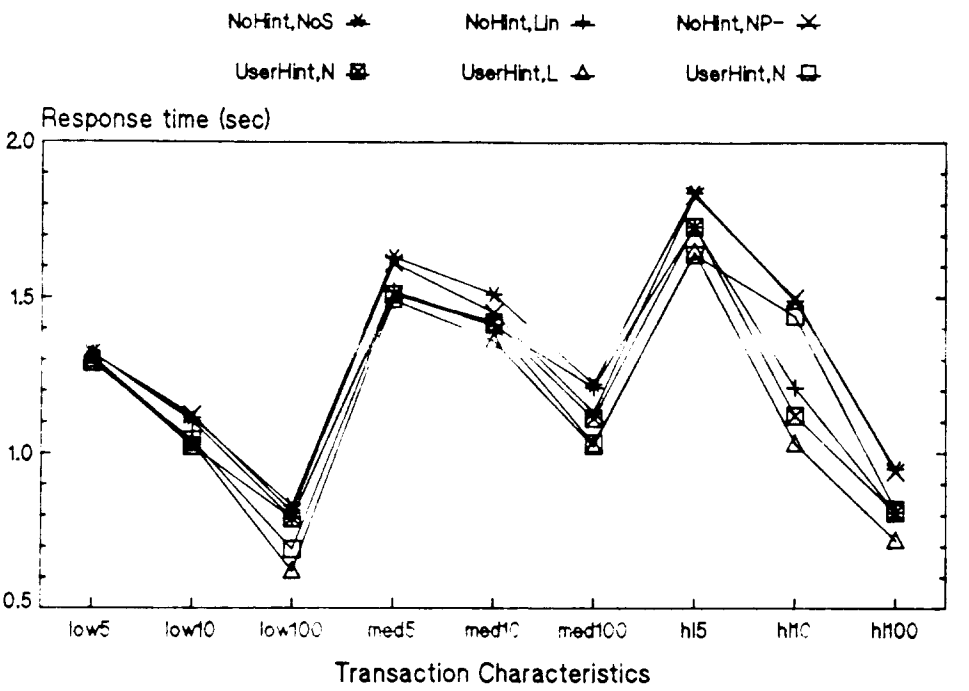

Figure 6-19 -- User Hint Effect Under Random Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within database, 2) 1000 buffers and 3) Random buffer replacement policy. The clustering control parameter is set to *clustering without I/O limitation*. All the user hint policies and page splitting policies are studied under this environment: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio 5. "Hi100" means the transaction has a high structuredensity which is greater than or equal to 10 and read/write ratio 100.

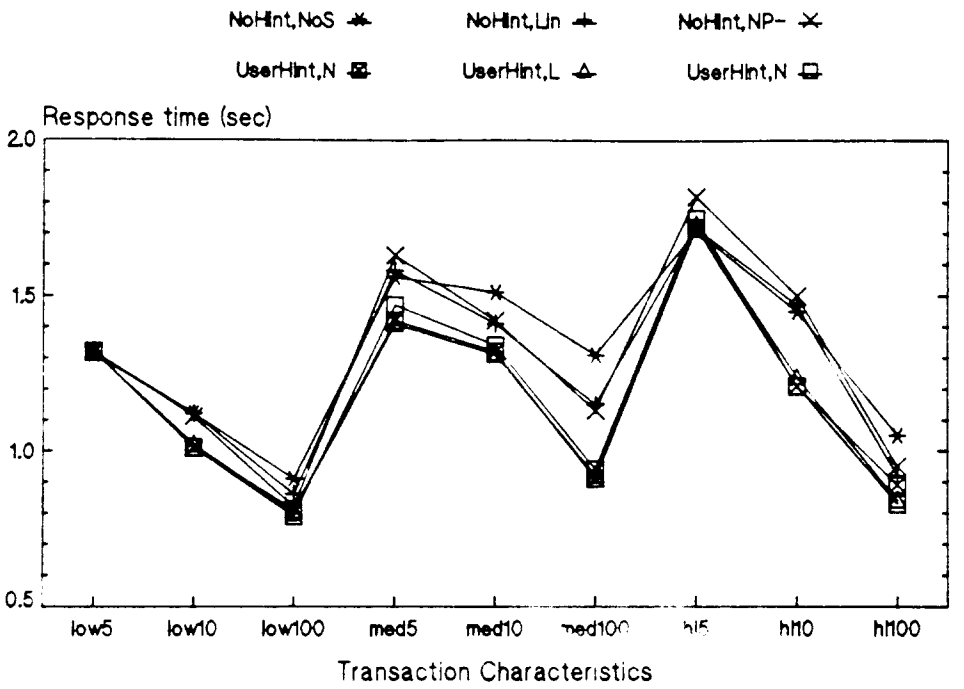

Figure 6-20 -- User Hint Effect Under LRU Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within database, 2) 1000 buffers and 3) LRU buffer replacement policy. The clustering control parameter is set to *clustering without I/O limitation*. All the user hint policies and page splitting policies are studied under this environment: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio 100.

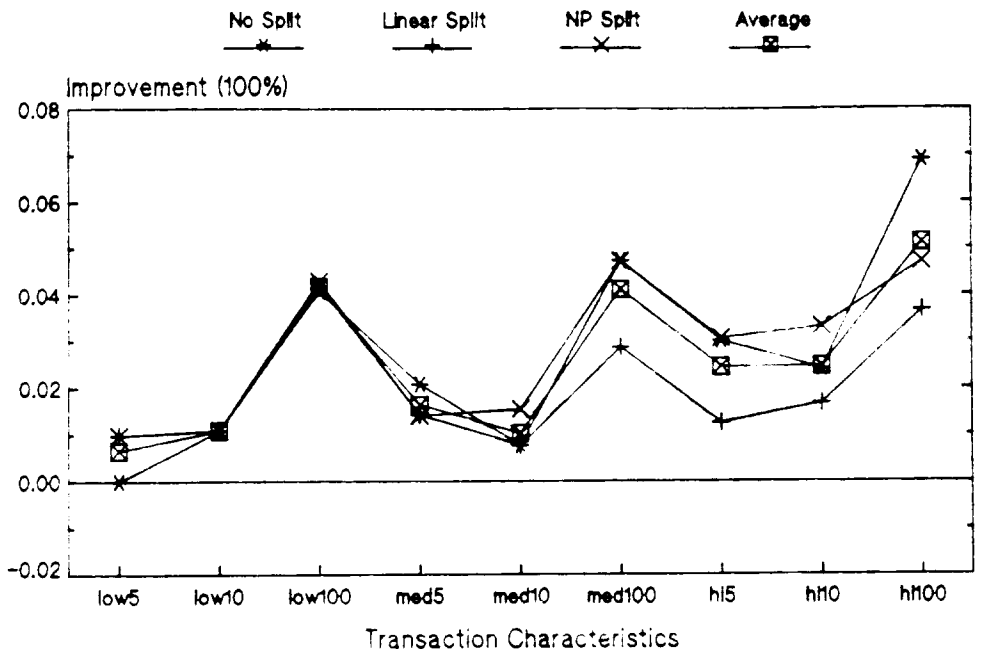

Figure 6-21 -- User Hint Improvement Under Context Buffer Replacement Policy

From Figure 6-18, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over not using user hint. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio 100.

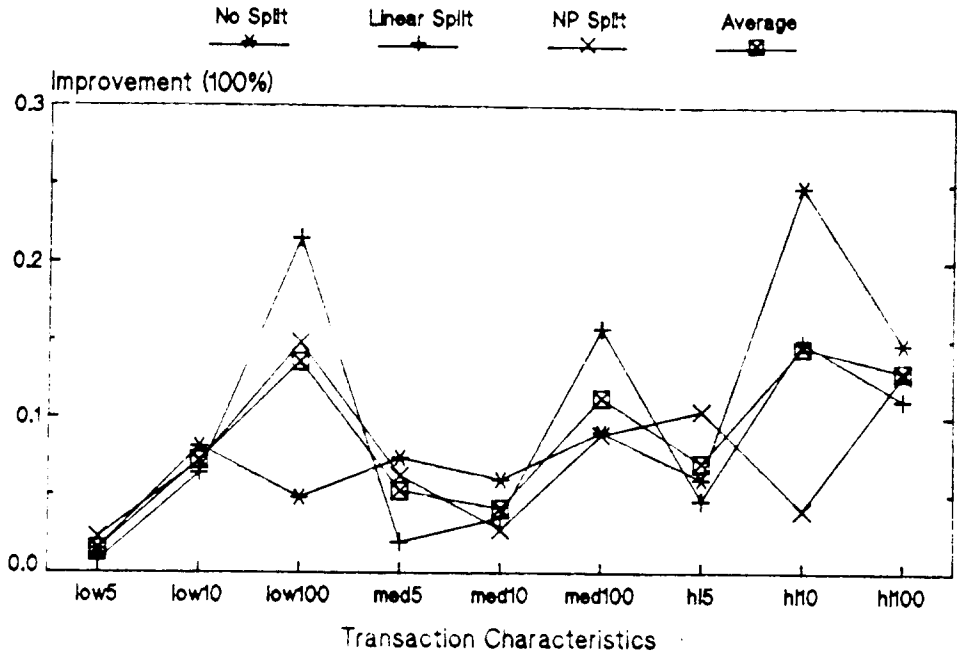

Figure 6-22 -- User Hint Improvement Under Random Buffer Replacement Policy

From Figure 6-19, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over not using user hint. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio 100.

(1) The context-sensitive buffer replacement algorithm has already increased the buffer hit ratio tremendously. Running the clustering algorithm improves the overall referential locality, as shown in Chapter 6.1. Although *User hints* may save some logical I/Os for readers, the real saving of physical I/Os does not greatly improve overall response time.

(2) Translating user hints to object identifiers consumes both CPU time and I/Os. If the real saving of physical I/Os is comparable to the physical I/Os caused by the translation process, *User hints* cannot greatly improve the overall system response time.

(3) When the buffer pool size is large, such as 1000 buffers, *User hint* does not save many physical I/Os, and its effect is not significant.

However, when the buffer replacement algorithm is not *Context-sensitive*, the response time improvement contributed by *User hint* becomes significant. In Figure 6-22 where the buffer replacement is *Random*, the improvement ranges from 2% to 25%. Taking the average of three different page splitting policies, we see the average improvement is 5% for a read/write ratio of 5 and 15% for a read/write ratio of 100. This 10% difference in average improvement is because the extra I/Os introduced by user hint processing cannot be amortized by readers when the read/write ratio is low.

Notice that this average number is independent of the structure density. From Figure 6-22, we can conclude that *User hint* will improve the system response time by at least 5% under the following environments: Random buffer replacement algorithm, prefetching within database, and clustering without I/O limitation.

Figure 6-23 shows the user hint effect under the LRU buffer replacement algorithm. Some key observations are:

(1) The average response time improvement ranges from 2%, when the transaction characteristics are high structure density and low read/write ratio, to 22% when the transaction characteristics are the medium structure density and high read/write ratio.

(2) A negative user hint effect, less than 1%, occurs under the high structure density, low read/write ratio, and *No page split* policy.

(3) The best user hint effect, of more than 30%, also occurs under *No page split* policy but with the medium structure density and high read/write ratio (i.e. 100).

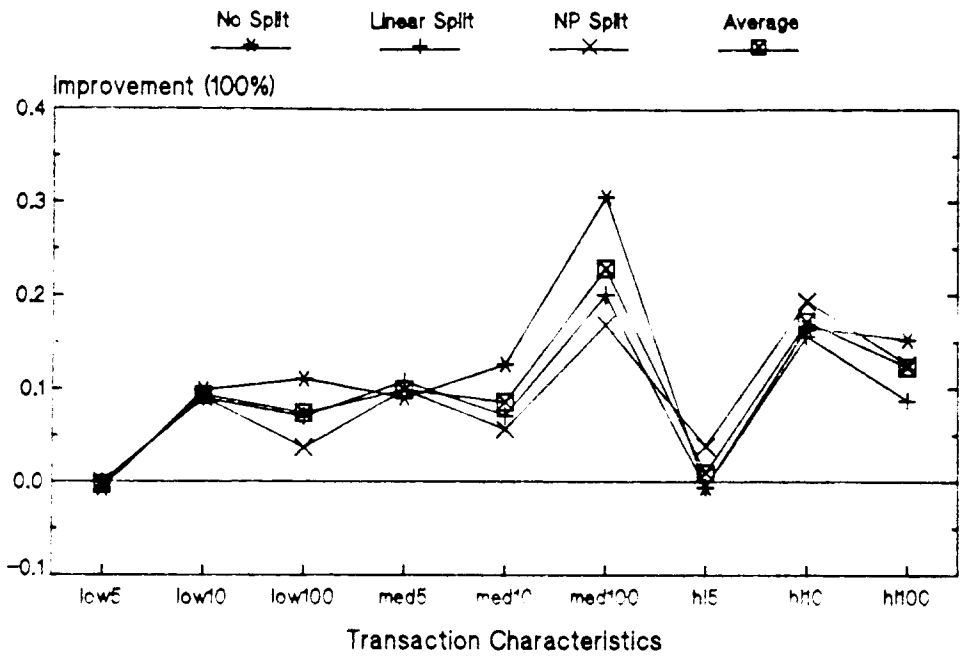

Figure 6-23 -- User Hint Improvement Under LRU Buffer Replacement Policy

From Figure 6-20, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over no user hint is used. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

Factor Interaction Analysis

Complicated user hints, such as "place near configuration X", require more CPU time and I/Os to do object identifier translation. When the read/write ratio is low, these extra I/Os caused by user hint processing cannot be amortized by readers and may contribute negatively to the overall system response time. Under the LRU buffer replacement algorithm, we observe a negative user hint effect for high structure density and low read/write ratio transactions. We also observe that in Figure 6-22, under the Random buffer replacement algorithm, a 6% response time improvement occurs for the same set of control

parameters.

Using the factor interaction graph shown in Figure 6-24 we can conclude that there is a major interaction between *buffer replacement algorithms* and *User hints* under high structure density, low read/write ratio and *No Split* policy. Notice that in Figures 6-19 and 6-20, the LRU buffer replacement algorithm provides a better response time than the Random buffer replacement algorithm in most cases. Even when *User Hint* is used and the page splitting policy is *No Split*, the LRU still has a better response time in all cases.

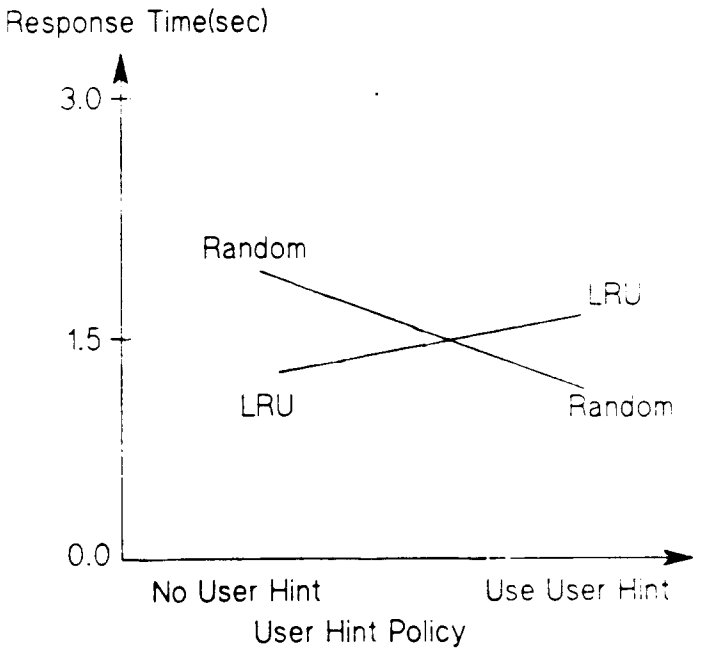

Figure 6-24 -- User Hint Policy vs. Buffering Policy Interaction Analysis Graph

The X-axis represents the user hint policy: No user hint and Use user hint. The Y-axis represents the response time for the LRU and Random buffer replacement policies under these different user hint policies when the structure density is high, the read/write ratio is low, and no page splitting is used. The smaller the response time, the better the buffering policy. This graph shows a **major** interaction case.

From Figures 6-20 and 6-23, we observe some interactions between page splitting policy and structure density, and page splitting policy and read/write ratio. The corresponding interaction analysis graphs have been shown in Figures 6-15 and 6-16.

6.3.2. User Hint Effect Under Prefetch within Buffer

In the previous section, we discussed the user hint effect with *Prefetching within database* under various buffer replacement algorithms and page splitting policies. If the *User hint* can dramatically improve the buffer hit ratio, *Prefetching within database* may perform comparably to *Prefetching within buffer*. Since a high buffer hit ratio reduces the number of logical I/Os for *Prefetching within database*, the difference in response time improvement between *Prefetching within database* and *Prefetching with buffer* becomes small, especially when large buffer pool is used. To verify this hypothesis, in this section, we examine the user hint effect under the *Prefetching within buffer* policy.

Figures 6-25, 6-26, and 6-27 show the *User hint* effect under different buffer replacement algorithms and various page splitting policies. The first interesting observation is that there is no negative user hint effect under the *Prefetching within buffer* policy. One simple explanation for this is that *user hint* has significantly improved the buffer hit ratio and extra I/Os caused by the *user hint* processing are amortized by readers' response time improvement. However, under high structure density, low read/write ratio, with a *No Page Split* policy and the LRU buffer replacement algorithm, *Prefetching within buffer* has a positive user hint effect of 12% whereas *Prefetching within database* has a negative effect of 1% (refer to Figures 6-23 and 6-30). So, why do we have such a wide variation?

Notice that the response time difference in *Prefetching within database* case (refer to figure 6-20) is very small. This is because *Prefetching within database* has already improved

the overall response time, and the user hint cannot further improve the buffer hit ratio. When the user hint processing overhead cannot be offset by the improvement in buffer hit ratio, we observe small response time degradation.

In Figures 6-21 and 6-28, we observe the average response time improvement is 2% to 4% for *Prefetching within database*, and 3% to 13% for *Prefetching within buffer*. This wide variation in response time improvement is caused by a situation similar to that explained before; since *Prefetching within database* has improved the response time effectively, there is not much room for further improvement. However, *Prefetching within buffer* can still utilize user hint to improve its buffer hit ratio and overall response time.

From the above discussion, we can easily explain why Figure 6-31 has shown some minor interaction between *prefetching policy* and *User hint*.

When the buffer replacement algorithm is *Random*, we observe a "random" (or wide variation) user hint effect in Figure 6-22 and 6-29. However, there is some similar behavior between these two different prefetching policies:

(1) *Linear Split* performs best in both prefetching policies under low structure density and high read/write ratio.

(2) *NP Split* excels under high structure density and low read/write ratio.

(3) *No Split* performs consistently well under medium structure density.

Figure 6-32 shows that there is a minor interaction between the user hint policy and the page splitting policy when the buffer replacement algorithm is *Random*. This is because using *User hints* can greatly improve the referential locality when a *No Split* page splitting policy is used, whereas only a small improvement occurs when a *Linear Split* policy is used.

In Figures 6-23 and 6-30, where a *LRU* buffer replacement algorithm is used, we observe no improvement for *Prefetching within database* and a 13% improvement for *Prefetching within buffer* under a high structure density and low read/write ratio. At the same time, under medium structure density and high read/write ratio, we observe a 20% improvement for *Prefetching within database* and only 9% improvement for *Prefetching within buffer.* Why does the *user hint* effect change so rapidly under different transaction characteristics?

Let's use an example to understand the relationship between the buffer hit ratio and the response time improvement. Assuming that the buffer hit ratio is 0.8 when no user hint is used, if the user hint can improve the hit ratio by 12.5%, we will have a 0.9 buffer hit ratio. If we suppose that the total number of logical I/Os invoked by applications is 100, the real physical I/Os are then reduced to 10 from 20, a 50% improvement Since the response time is tightly related to the number of physical I/Os at run-time, the improvement in physical I/Os can be translated into a response time improvement. Therefore, the response time improvement can be defined as formula (1):

(1) $$(X*A)/(1-A)$$

where X is the user hint effect on buffer hit ratio, and A is the base buffer hit ratio. For example, when the base buffer hit ratio is 0.8, and the user hint effect on buffer hit ratio is 10%, the response time improvement will be 40%. If the base buffer hit ratio is 0.5, we will only have a 10% response time improvement under the same user hint effect.

Figure 6-33 shows the buffer hit ratio of *Prefetching within database* and *Prefetching within buffer* under the *LRU buffer replacement policy* and different transaction characteristics. Notice that under medium structure density and high read/write ratio, we get a 0.9 buffer hit ratio for *Prefetching within database* and a 0.8 for *Prefetching within buffer.* The

corresponding user hint effects on buffer hit ratio, using formula (1), are 0.022 and 0.021. This result supports the hypothesis that the *user hint* effect on buffer hit ratio should stay constant under the same transaction characteristics.

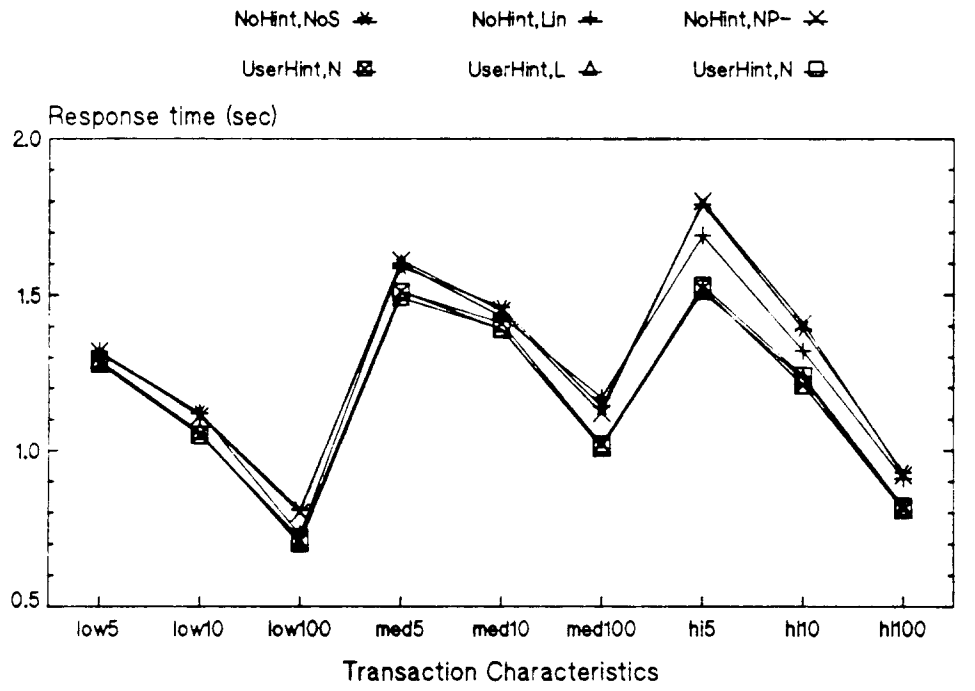

Figure 6-25 -- User Hint Effect Under Context-sensitive Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within buffer, 2) 1000 buffers and 3) Context-sensitive buffer replacement policy. The clustering control parameter, set to *clustering without I/O limitation*, is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

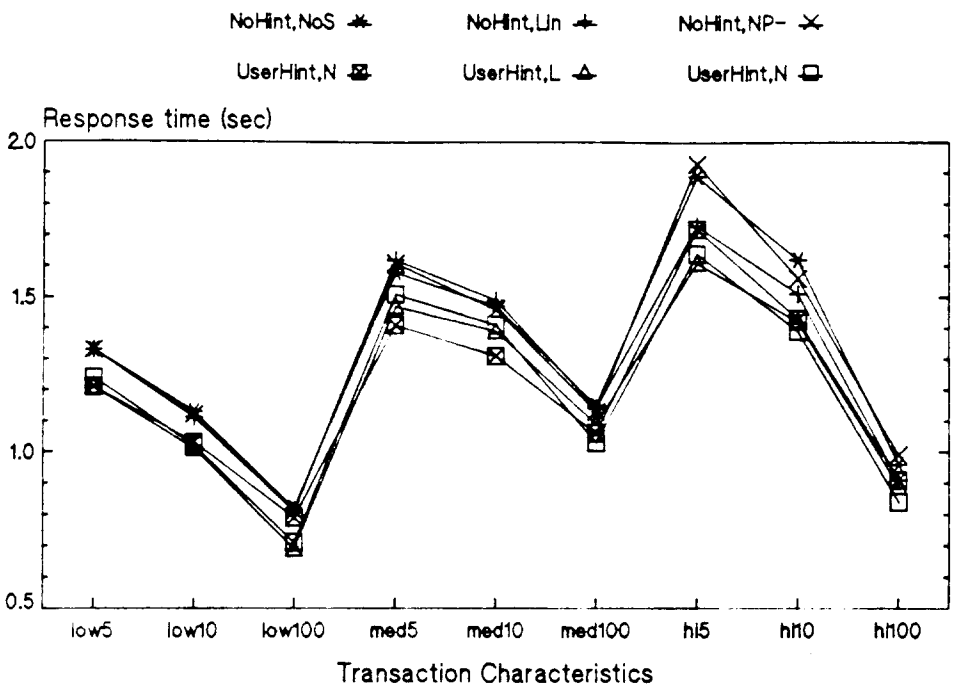

Figure 6-26 -- User Hint Effect Under Random Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within buffer, 2) 1000 buffers and 3) Random buffer replacement policy. The clustering control parameter, set to *clustering without I/O limitation* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

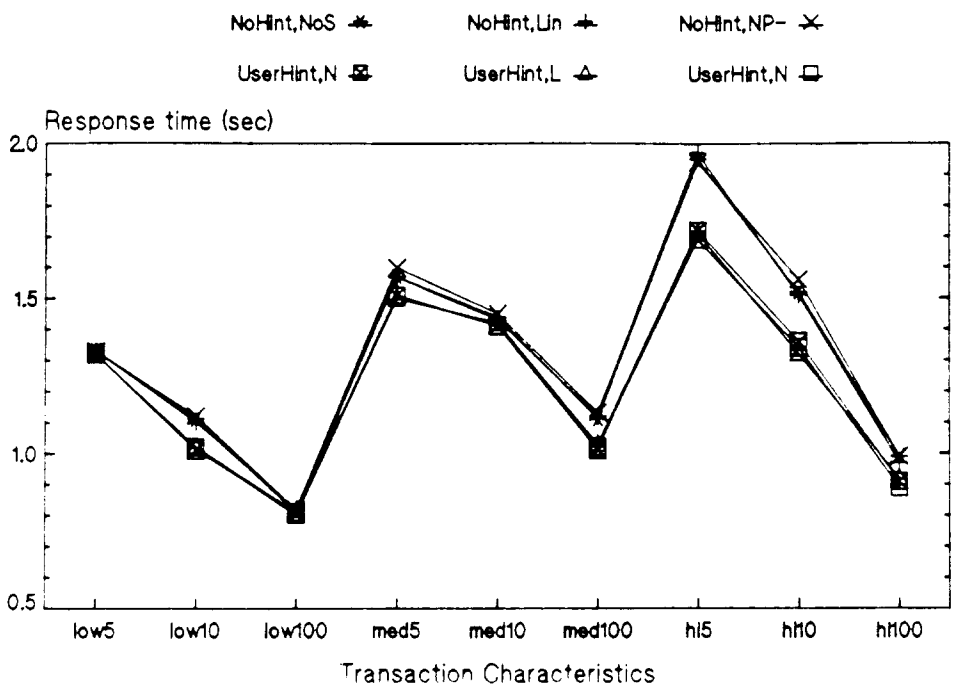

Figure 6-27 -- User Hint Effect Under LRU Buffer Replacement Policy

The buffering control parameters are fixed to 1) prefetching within buffer, 2) 1000 buffers and 3) LRU buffer replacement policy. The clustering control parameter, set to *clustering without I/O limitation* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

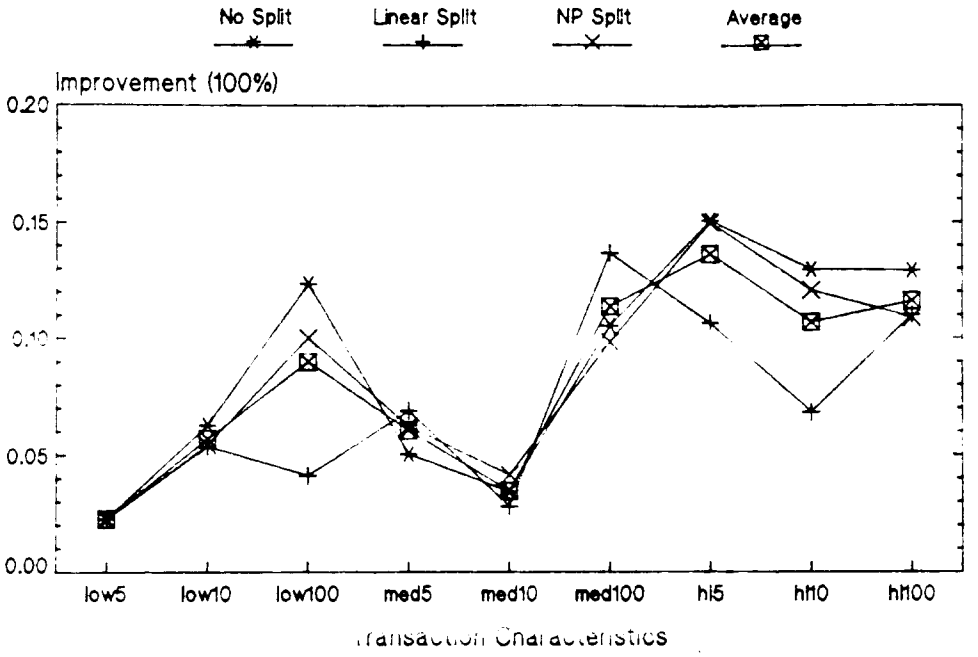

Figure 6-28 -- User Hint Improvement Under Context Buffer Replacement Policy

From Figure 6-25, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over not using user hint. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

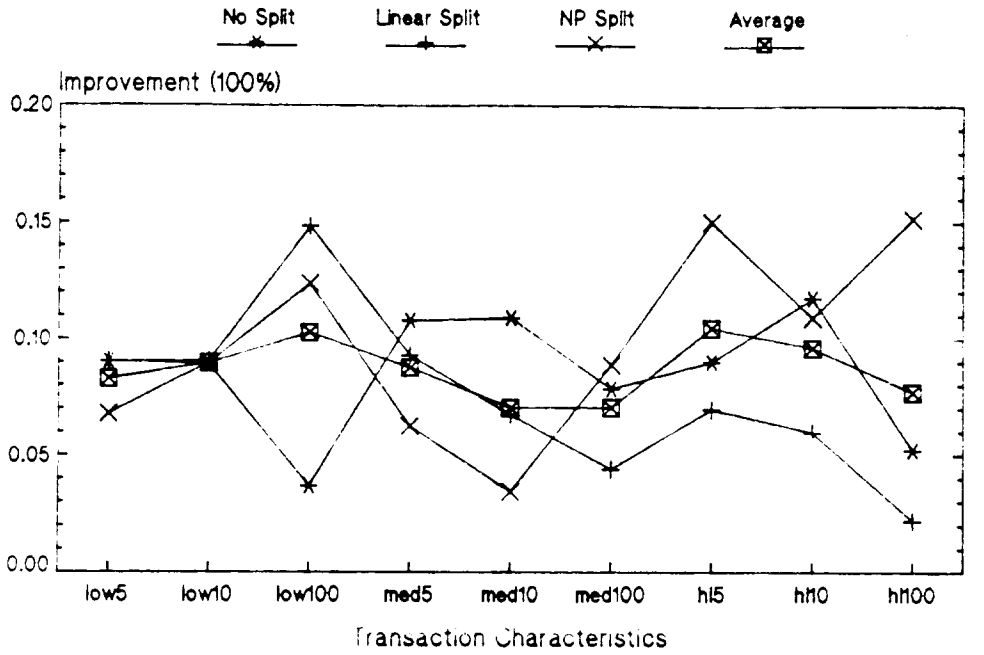

Figure 6-29 -- User Hint Improvement Under Random Buffer Replacement Policy

From Figure 6-26, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over not using user hint. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

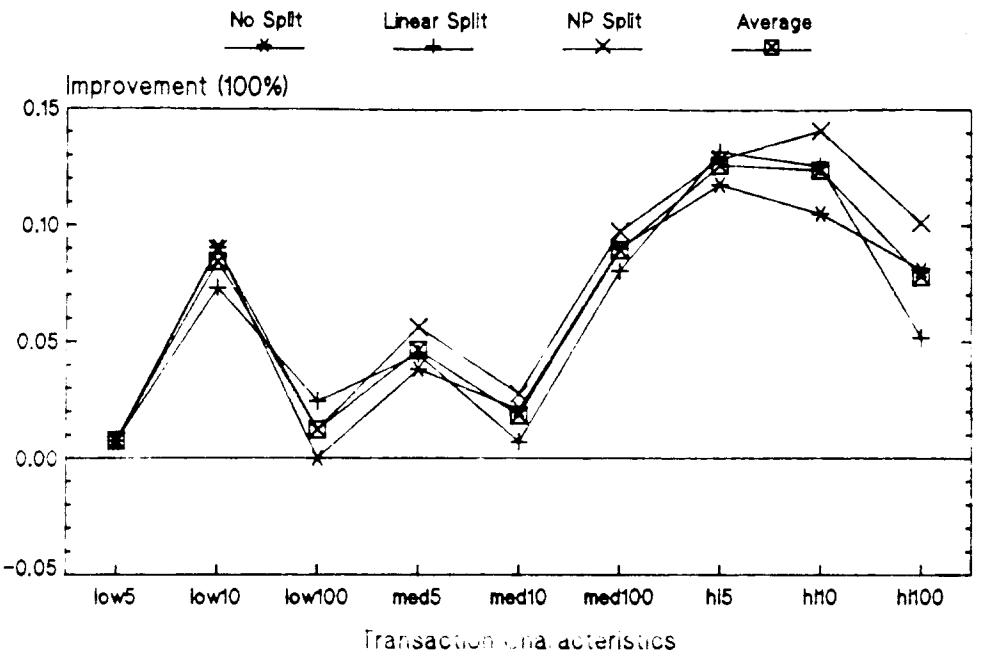

Figure 6-30 -- User Hint Improvement Under LRU Buffer Replacement Policy

From Figure 6-27, we calculate the response time improvement when user hint is used. The Y axis represents the improvement over not using user hint. "0.2" means 20% improvement. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

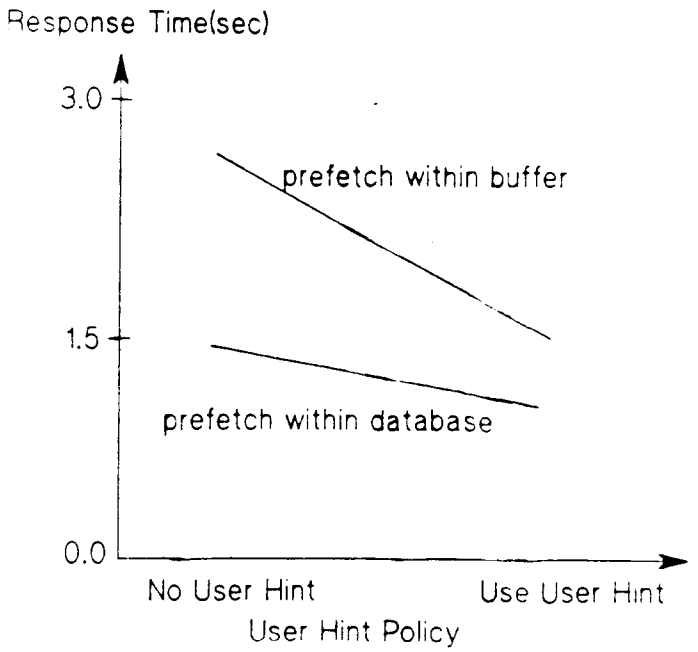

Figure 6-31 -- User Hint Policy vs. Prefetching Interaction Analysis Graph

The X-axis represents the user hint policy: No user hint and Use user hint. The Y-axis represents the response time for prefetching within database and prefetching within buffer. Two lines in parallel imply no interaction. Two lines intersecting mean major interaction. This graph shows a **minor** interaction case.

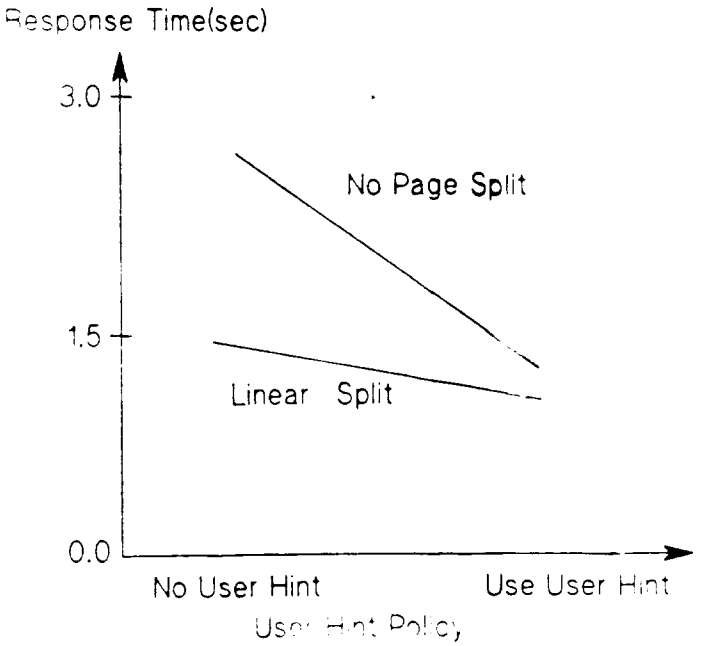

Figure 6-32 -- User Hint Policy vs. Page Splitting Policy Interaction Analysis Graph

The X-axis represents the user hint policy: No user hint and Use user hint. The Y-axis represents the response time for different page splitting policies when the *Random* buffer replacement algorithm is used. This graph shows a **minor** interaction case.

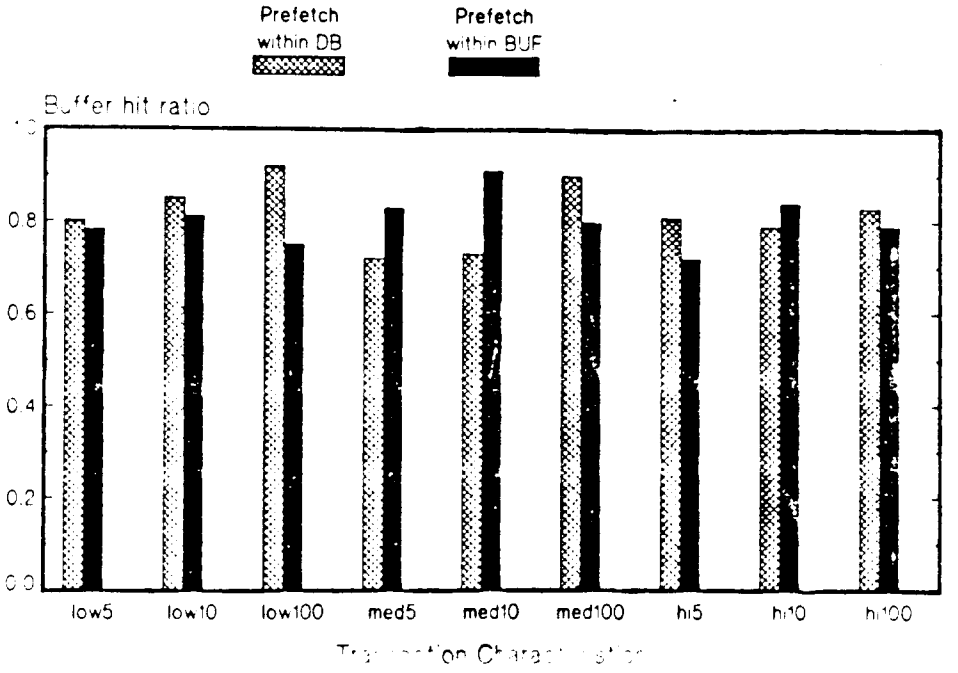

Figure 6-33 -- Buffer Hit Ratio for P_DB_LRU and P_BUF_LRU

The Y axis represents the buffer hit ratio for prefetching within database (P_DB_LRU) and prefetching within buffer (P_BUF_LRU) when the LRU buffer replacement algorithm is used. The X axis represents various transaction characteristics. "Low5" means the transaction has a low structure density which is less than or equal to 3 and a read/write ratio of 5. "Hi100" means the transaction has a high structure density which is greater than or equal to 10 and a read/write ratio of 100.

6.3.3. User Hint Effect Under Varying Clustering Policies

User hints, in general, provide a set of potential candidate pages for clustering based on applications' characteristics at run-time. If there are only one or two candidate pages, *Clustering without I/O limitation* may perform comparably to *Clustering with 2 I/O limitation*. Further, *Clustering within buffer pool* may perform well if we have a large number of candidate pages, some of which are cached in the buffer pool. In this section, we study the *User hint* effect under various clustering policies, focusing especially on the interaction between user hint and clustering policy.

Figures 6-34, 6-35, 6-36, 6-37 show the *User hint* effect on response time under different clustering policies with varying page splitting policies. Some key observations are:

(1) *User hint* improves the response time by 40% when the *Clustering within buffer pool* policy is used. Since *User hint* increases the number of candidate pages for clustering, it also increases the probability of buffer pool hit at run-time. Because the clustering algorithm can then place objects more effectively, the overall response time improves.

(2) The response time variation becomes smaller for all clustering policies when *User hints* are used. Such a stable response time may be required by some object-oriented applications.

(3) On the average, *Clustering without I/O limitation* performs much better than *Clustering with 2 I/O limitation* when the *User hints* is used. The left graph in Figure 6-38 shows the interaction analysis graph for these two clustering policies under different user hints policies.

(4) From observation (1) and Figure 6-1, we can also conclude that there is a minor interaction between clustering policies and user hints, as shown in the right graph in Figure 6-38.

6.4. Impact of Smart Buffering

Three buffering control parameters are studied in this section: **Buffer replacement policy**, **Prefetching policy**, and **Buffer pool size**. The operating levels of clustering control parameters used in this study of buffering effects are: *clustering without I/O limitation* and splitting page when the candidate page overflows. We first discuss the prefetching effect on response time under various buffer replacement policies and user hint in Chapter 6.4.1. The effect of the buffer pool size on prefetching policy and buffer replacement policy is

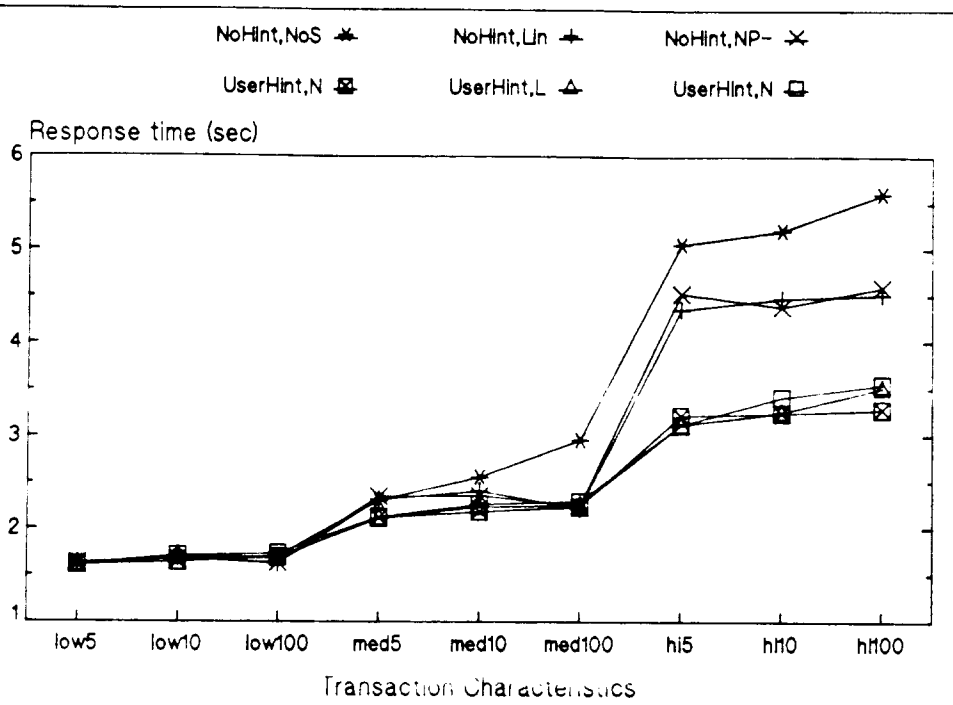

Figure 6-34 -- User Hint Effect Under Clustering within Buffer

The buffering control parameters are fixed to 1) no prefetching, 2) 1000 buffers, and 3) LRU buffer replacement policy. The same clustering control parameter, set to *clustering within buffer* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

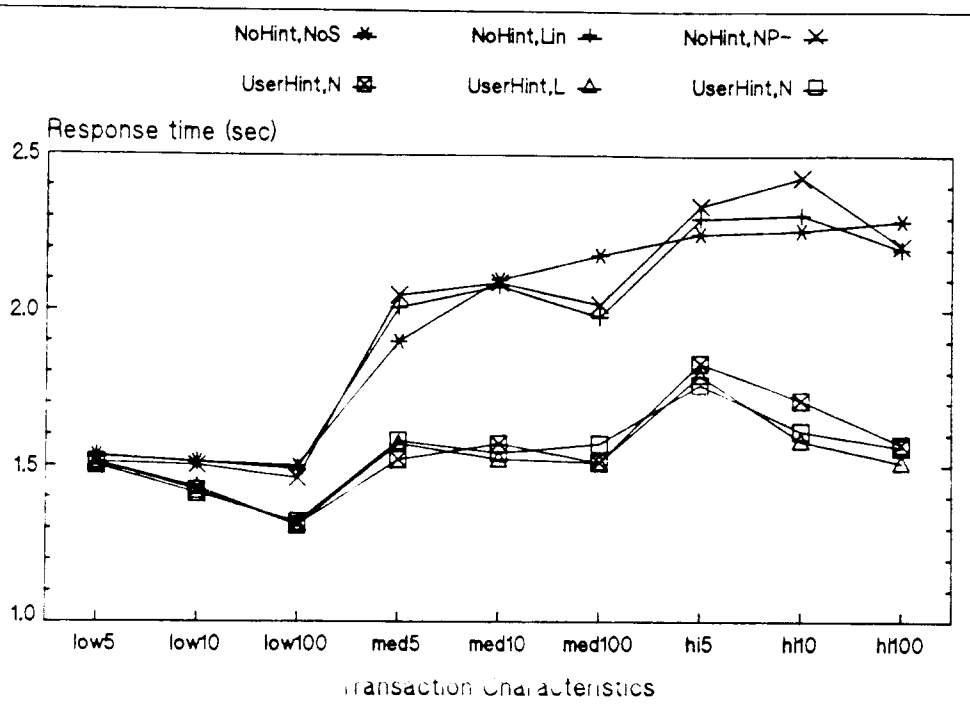

Figure 6-35 -- User Hint Effect Under
Clustering 2 I/O limitation

The buffering control parameters are fixed to 1) no prefetching, 2) 1000 buffers, and 3) LRU buffer replacement policy. The same clustering control parameter, set to *clustering with 2 I/O limitation* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

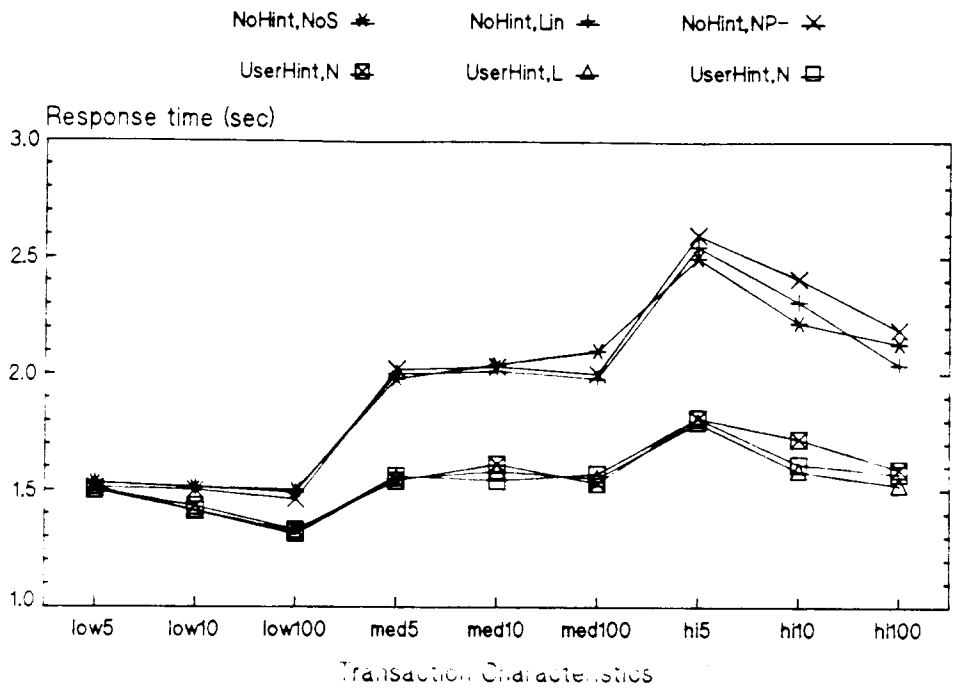

Figure 6-36 -- User Hint Effect Under
Clustering with 10 I/O limitation

The buffering control parameters are fixed to 1) no prefetching, 2) 1000 buffers, and 3) LRU buffer replacement policy. The same clustering control parameter, set to *clustering with 10 I/O limitation* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

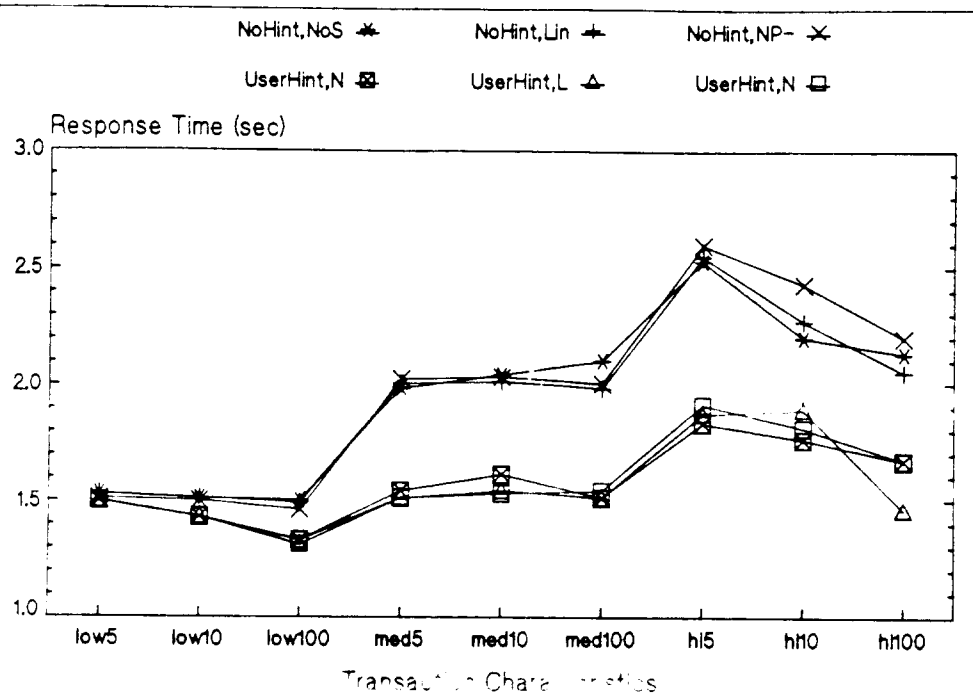

Figure 6-37 -- User Hint Effect Under
Clustering with no I/O limitation

The buffering control parameters are fixed to 1) no prefetching, 2) 1000 buffers, and 3) LRU buffer replacement policy. The same clustering control parameter, set to *clustering with no I/O limitation* is used for all the user hint policies and page splitting policies: no user hint with no page split (NoH,NoSplt), no user hint with linear page split (NoH,L-Split), no user hint with NP page split (NoH,NP-Splt), user hint with no page split (UH,NoSplt), user hint with linear page split (UH,L-Splt), and user hint with NP page split (UH,NP-Splt). The X axis represents various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

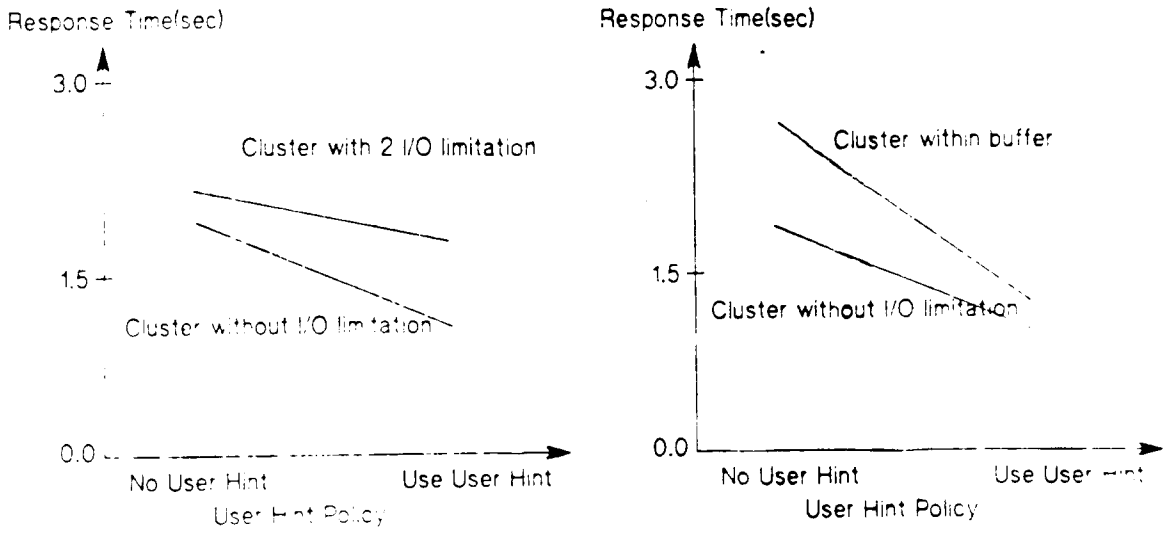

Figure 6-38 -- User Hint Policy vs. Clustering Policy
Interaction Analysis Graph

The X-axis represents the different user hint policies: No user hint and Use user hint. The Y-axis represents the response time under *clustering with 2 I/O limitation* and *clustering with no I/O limitation* for left graph and *clustering within buffer* and *clustering with no I/O limitation* for the right graph. Both of these interaction analysis graphs represent minor interaction cases.

studied in Chapter 6.4.2.

6.4.1. Prefetching Effect

Three buffer replacement strategies, *Context-sensitive*, *Random* and *LRU*, are studied with three different prefetching strategies: prefetching within database, prefetching within buffer pool, and no prefetching. The buffer pool size is fixed at 1000 and no user hints are provided.

Figure 6-39 shows how various buffering policies affect system response time under different workloads. Some key observations are:

(1) The context-sensitive buffer replacement policy always improves the overall system response time. When both the structure density and read/write ratio are high, *Context-sensitive* with *prefetching within database* outperforms *LRU with no prefetching* by 150% in response time.

(2) Setting the scope on prefetching is a valid idea. When using prefetching within the buffer pool, the performance of the LRU and Random buffer replacement strategies is comparable to that of *Context-sensitive without any prefetching* in all workloads.

(3) *Context-sensitive with prefetching within database* performs best, whereas *LRU with no prefetching* performs worst. A similar conclusion on the LRU buffer replacement strategy has been drawn in relational database systems.

Figures 6-40, 6-41, and 6-42 show the prefetching effect on response time when the buffer replacement algorithm is fixed. In all cases, *prefetch within database* performs best. This may not be intuitive since prefetching may bring in some pages not used by applications due to changing access patterns. However, because prefetching has made data available in memory to applications ahead of the request, the overall response time is improved. For object-oriented applications, paying extra I/Os to achieve better response time may be acceptable.

When the context-sensitive buffer replacement algorithm is used, as shown in Figure 6-40, *prefetch within buffer* has the same response time as *no prefetch* for transactions having low and medium structure densities. Notice that *prefetch within buffer* does not trigger any I/Os but only changes the buffer priority to reflect the future needs of data contained in these buffers. Since the context-sensitive buffer replacement algorithm would set up the appropriate priority based on knowledge of structure relationships and inheritance, *no prefetch* only causes very few buffer misses relative to *prefetch within buffer*. However, when

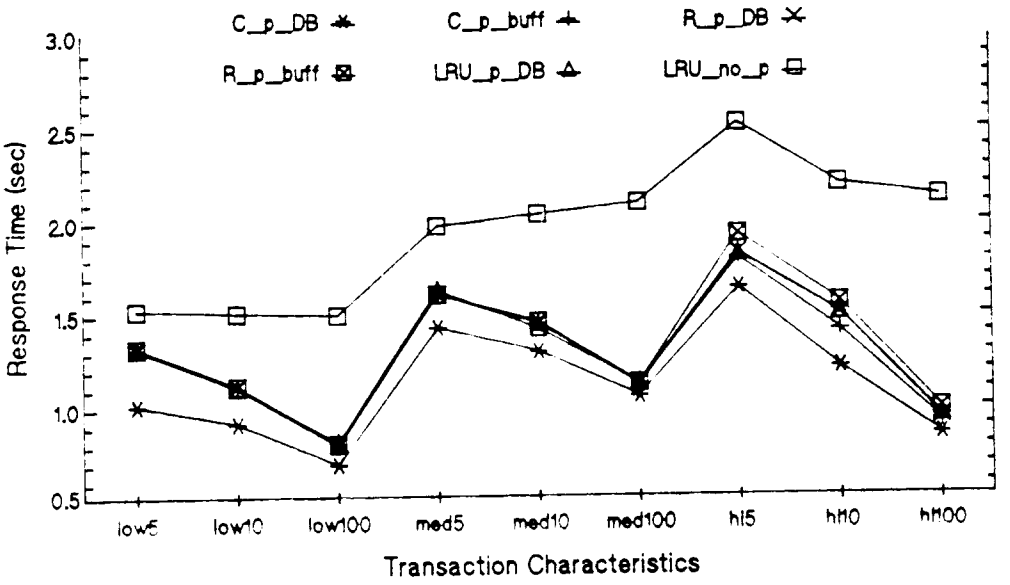

Figure 6-39 -- Buffering Effects Analysis
under 1000 buffers

Six experiments are reported here: Context-sensitive buffer replacement policy with prefetching within database (C_p_DB), Context-sensitive with prefetching within buffer (C_p_buff), Random buffer replacement policy with prefetching within database (R_p_DB), Random buffer replacement policy with prefetching within buffer (R_p_buff), LRU buffer replacement policy with prefetching within database (LRU_p_DB), and LRU with no prefetching (LRU_no_p). Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

the structure density becomes higher, *prefetch within buffer* better reflects the dynamics of the access pattern in how buffer priorities are set than *no prefetch* and has a better response time.

When the buffer replacement algorithm is not context-sensitive, as shown in Figures 6-41 and 6-42, prefetching becomes the only way to reflect the knowledge of structure relationships and inheritance in the buffer priority setting. Both *prefetch within database* and *prefetch within buffer* improve the buffer hit ratio and improve applications' response time.

Although *prefetch within database* has more logical I/Os than *prefetch within buffer*, their overall response time is similar.

Figures 6-43, 6-44, and 6-45 show the combined effect of *User hints* and *Prefetching* under varying transaction characteristics and page splitting policies. Some key observations are:

(1) *Prefetching within database* with *User hints* has the best response time in all cases. This is consistent with the results shown in Chapter 6.3.

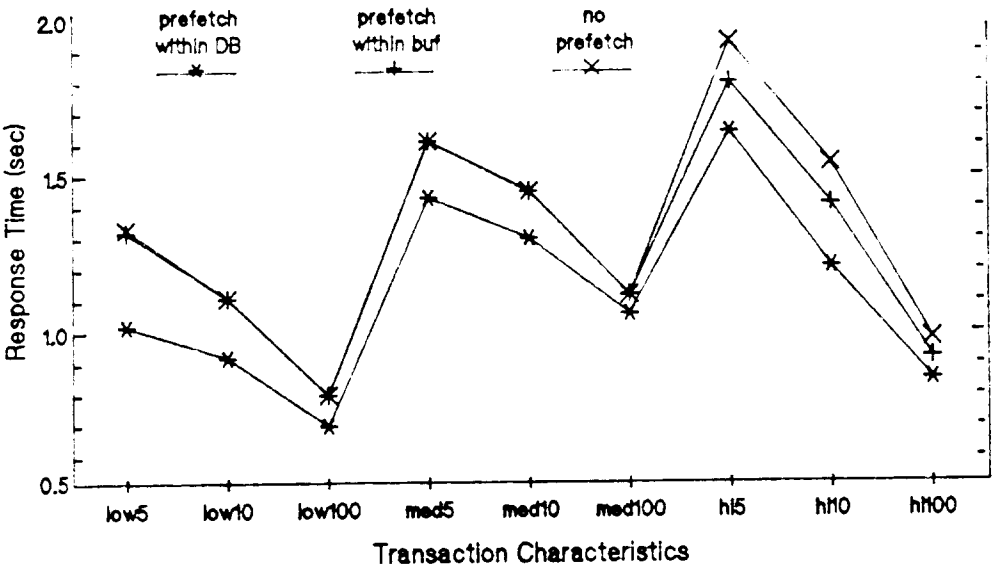

Figure 6-40 -- Prefetching Effect under Context-sensitive Buffer Replacement Policy

Three prefetching policies are evaluated here under a Context-sensitive buffer replacement algorithm with various transaction characteristics. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

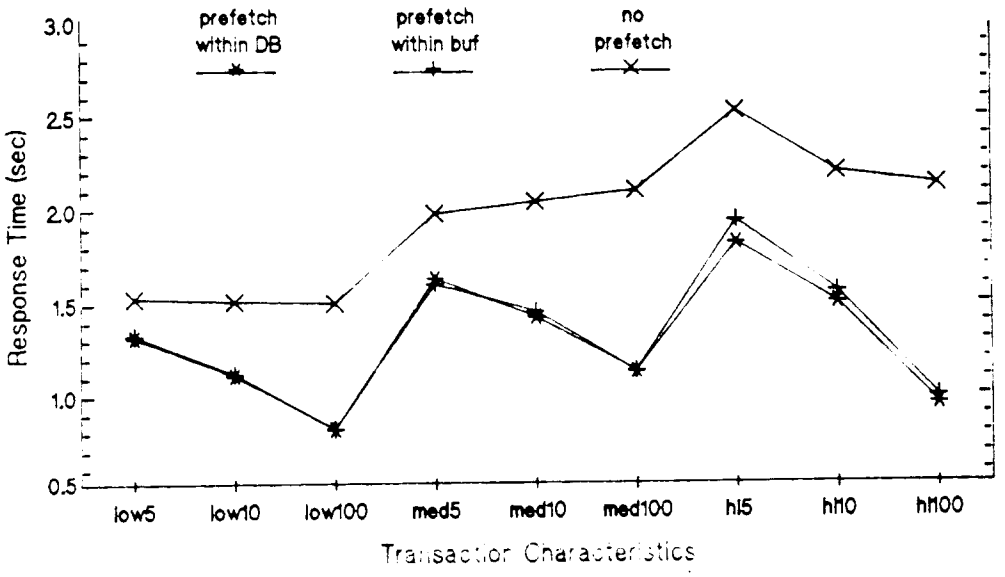

Figure 6-41 -- Prefetching Effect under LRU Buffer Replacement Policy

Three prefetching policies are evaluated here under a LRU buffer replacement algorithm with various transaction characteristics. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

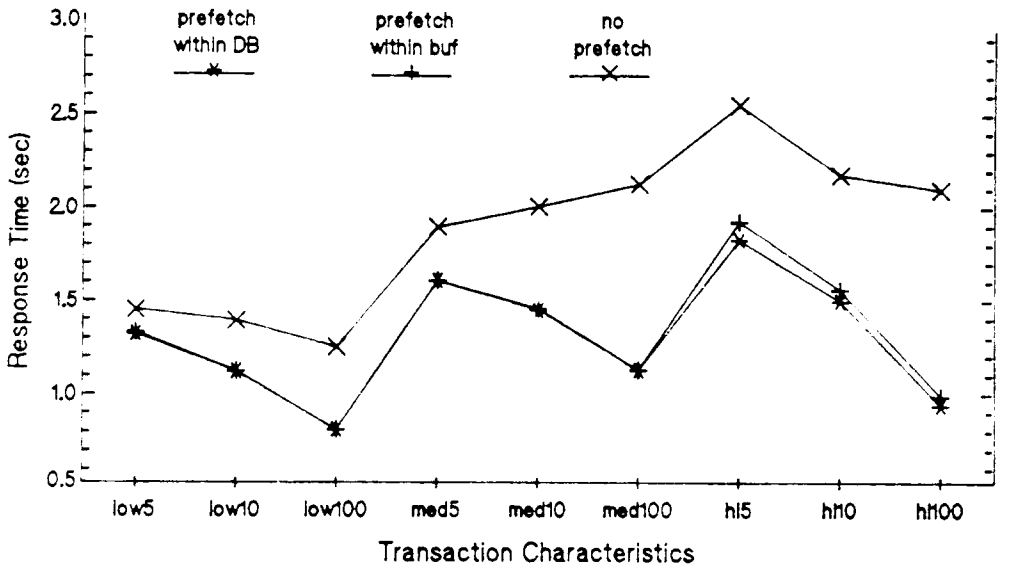

Figure 6-42 -- Prefetching Effect under Random Buffer Replacement Policy

Three prefetching policies are evaluated here under a Random buffer replacement algorithm with various transaction characteristics. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

(2) The relative performance among these six combined effects is very similar in all these figures. This implies that there is no interaction between *Prefetching policy* and *User hints*.

6.4.2. Buffer Pool Size Effect

The buffer pool size is fixed at 1000 buffers, 4k byte per buffer, in all the previous analyses. In this section, we expand Figure 6-39 to consider the effect of the buffer pool size on overall response time. Figures 6-46 and 6-47 show the same set of experiments under 100 buffers and 10,000 buffers. Three interesting observations are:

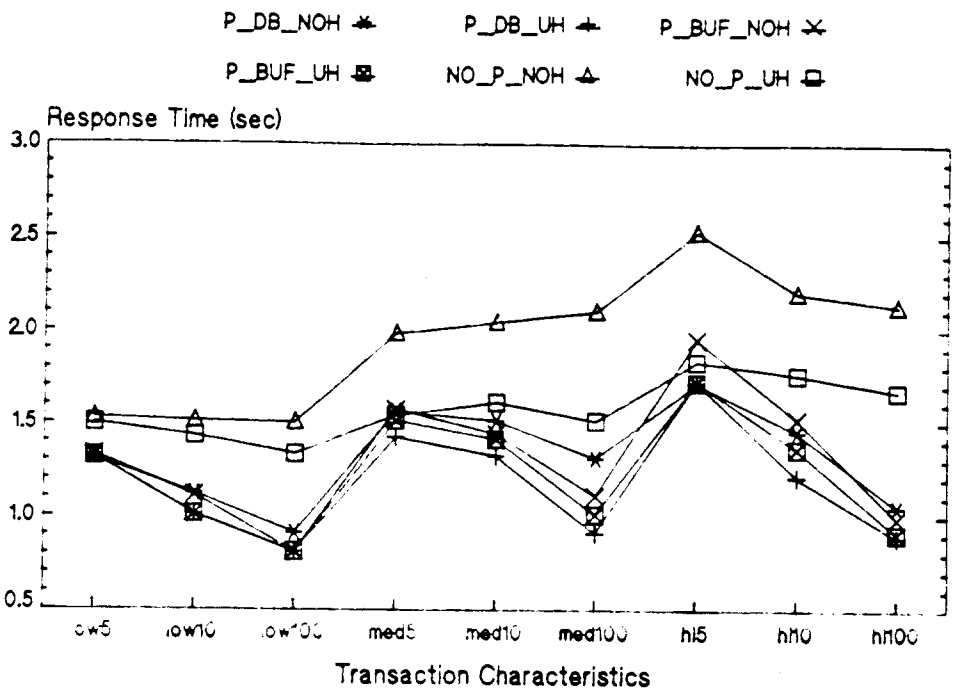

Figure 6-43 -- Prefetching and User Hint Effect under No Split

Six experiments are reported here: prefetching within database with no user hints (P_DB_NOH), prefetching within database with user hints (P_DB_UH), prefetching within buffer with no user hints (P_BUF_NOH), prefetching within buffer with user hints (P_BUF_UH), no prefetching without user hints (NO_P_NOH), and no prefetching with user hints (NO_P_UH). The clustering policy is clustering without I/O limitation, and the buffer replacement policy is LRU. Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

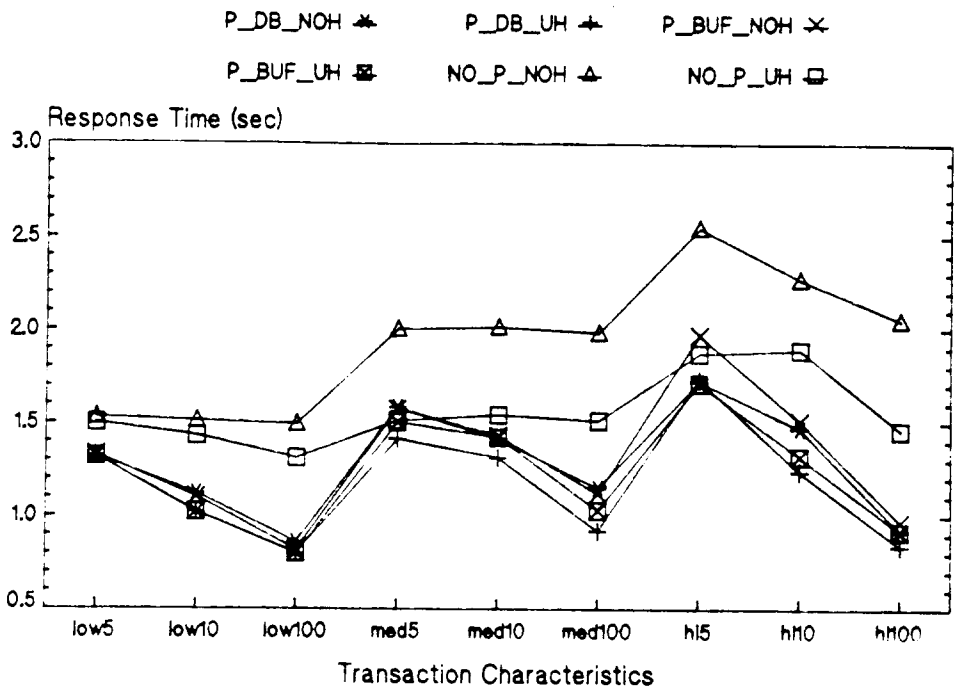

Figure 6-44 -- Prefetching and User Hint Effect under Linear Split

Six experiments are reported here: prefetching within database with no user hints (P_DB_NOH), prefetching within database with user hints (P_DB_UH), prefetching within buffer with no user hints (P_BUF_NOH), prefetching within buffer with user hints (P_BUF_UH), no prefetching without user hints (NO_P_NOH), and no prefetching with user hints (NO_P_UH). The clustering policy is clustering without I/O limitation, and the buffer replacement policy is LRU. Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

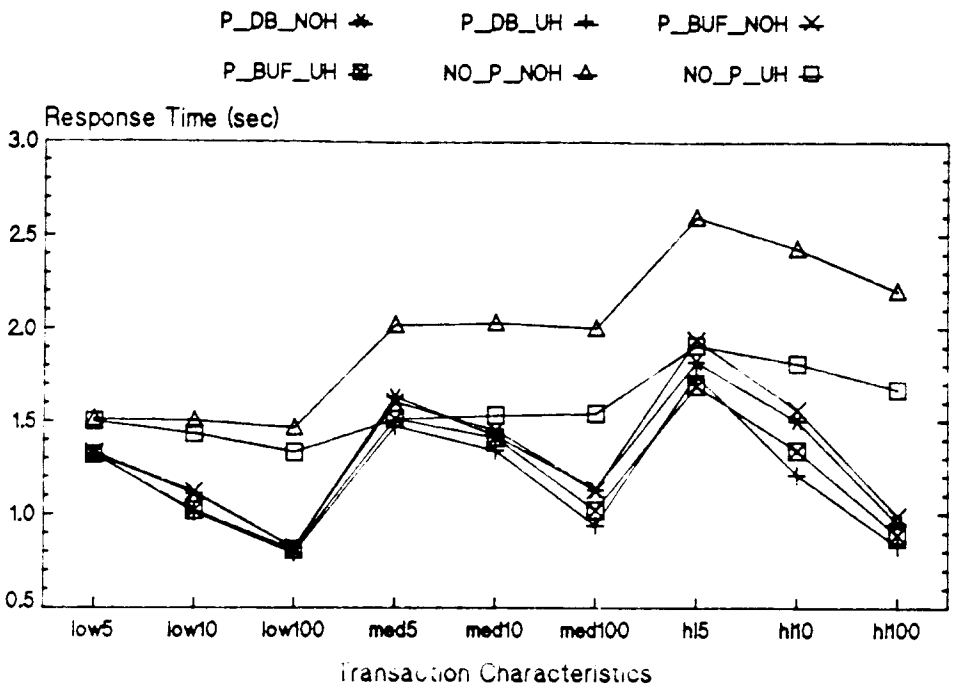

Figure 6-45 -- Prefetching and User Hint Effect under NP Split

Six experiments are reported here: prefetching within database with no user hints (P_DB_NOH), prefetching within database with user hints (P_DB_UH), prefetching within buffer with no user hints (P_BUF_NOH), prefetching within buffer with user hints (P_BUF_UH), no prefetching without user hints (NO_P_NOH), and no prefetching with user hints (NO_P_UH). The clustering policy is clustering without I/O limitation, and the buffer replacement policy is LRU. Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and the a read/write ratio of 100.

(1) Using a smaller buffer pool increases the difference among all six experiments. *Prefetching within database* performs much better than *Prefetching within buffer pool*, and *No Prefetching* has the worst response time overall.

(2) Under 40 Mbyte of buffer pool, the prefetching effect becomes less important, as shown in Figure 6-47. Both Random and LRU buffer replacement algorithms perform similarly under *Prefetching within database*.

(3) Based on these two observations, we can conclude that the buffer pool size has a minor interaction with both the buffer replacement algorithm and the prefetching policy. The corresponding interaction analysis graph is shown in Figure 6-48.

6.5. Overall Effect Analysis

The eight control parameters specified in Table 5-1 have different effects on the system response time. They interact with each other and combine to cause still different

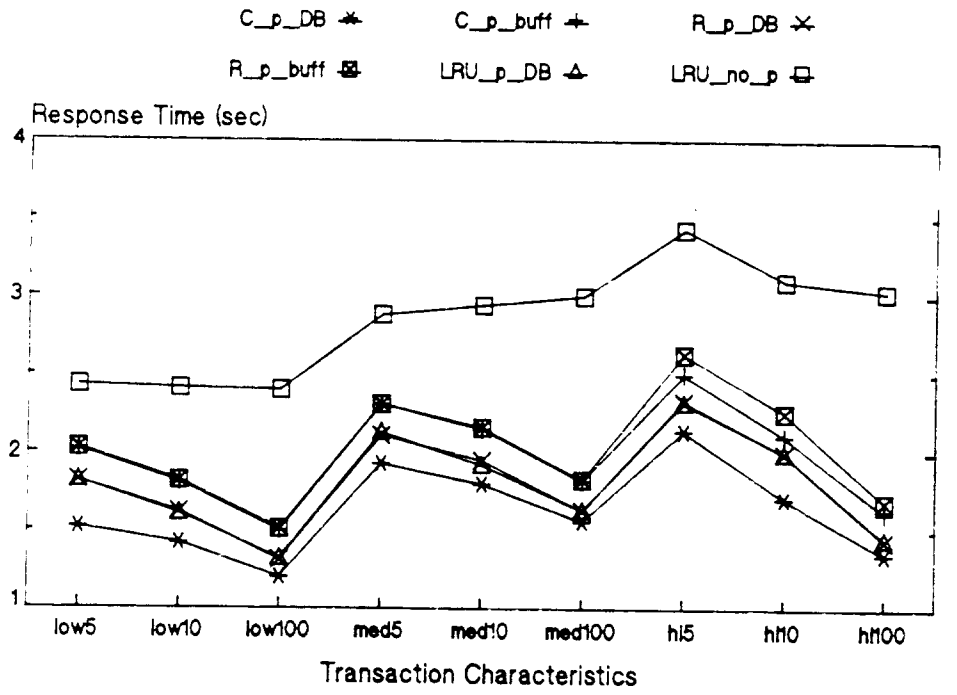

Figure 6-46 -- Buffering Effects Analysis under 100 buffers

Six experiments are reported here: Context-sensitive buffer replacement policy with prefetching within database (C_p_DB), Context-sensitive with prefetching within buffer (C_p_buff), Random buffer replacement policy with prefetching within database (R_p_DB), Random buffer replacement policy with prefetching within buffer (R_p_buff), LRU buffer replacement policy with prefetching within database (LRU_p_DB), and LRU with no prefetching (LRU_no_p). Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

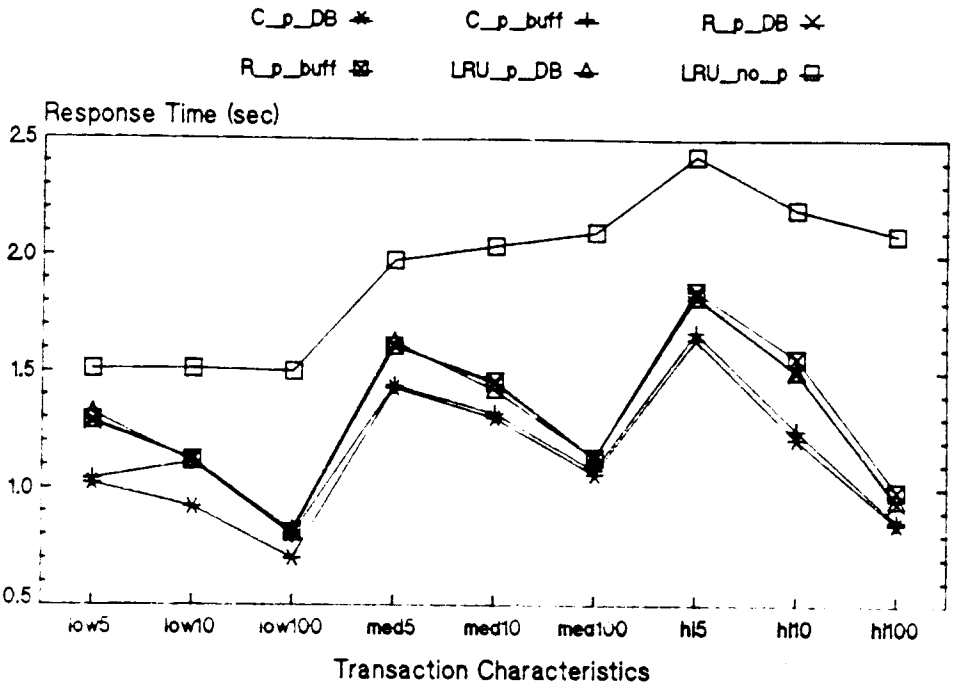

Figure 6-47 -- Buffering Effects Analysis
under 10,000 buffers

Six experiments are reported here: Context-sensitive buffer replacement policy with prefetching within database (C_p_DB), Context-sensitive with prefetching within buffer (C_p_buff), Random buffer replacement policy with prefetching within database (R_p_DB), Random buffer replacement policy with prefetching within buffer (R_p_buff), LRU buffer replacement policy with prefetching within database (LRU_p_DB), and LRU with no prefetching (LRU_no_p). Various transaction characteristics are evaluated. "Hi100" means the transaction has a high structure density and a read/write ratio of 100.

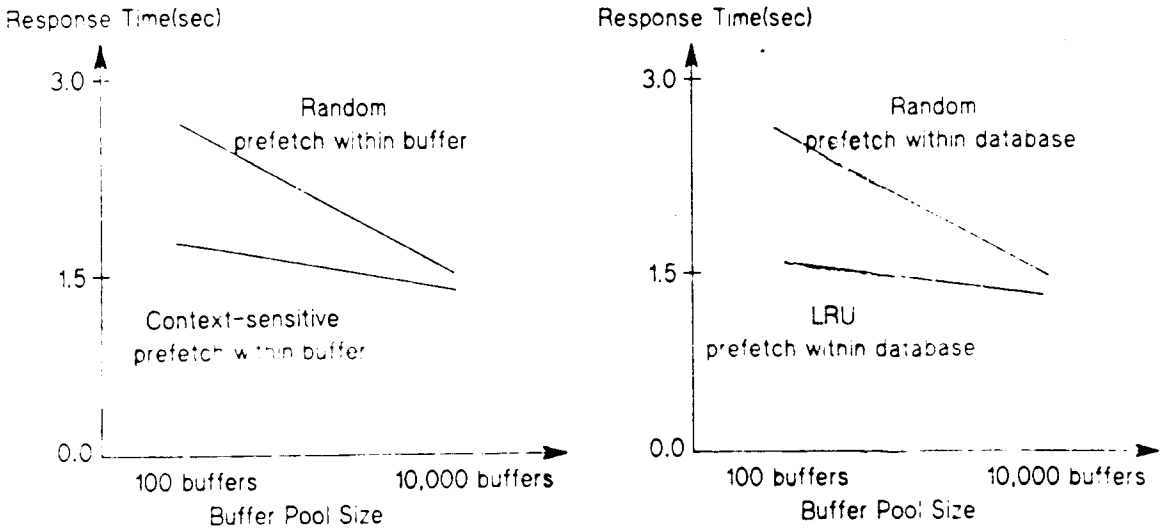

Figure 6-48 -- Buffer Pool Size vs. Prefetching Policy Interaction Analysis Graph

The X-axis represents the buffer pool size: 100 and 10,000. The Y-axis of the left graph represents the response time under Random and LRU buffer replacement algorithms when the *prefetching within buffer pool* is used. The right graph represents the minor interaction case when the *prefetching within database* is used and the buffer replacement algorithms are Random and LRU. Both interaction graphs represent some minor interactions among the buffer pool size, the prefetching policy, and the buffer replacement algorithm.

effects. To understand interactions among these parameters, we used the interaction analysis graph in all the previous sections. However, the interaction graph does not show the average change in response time when a parameter, or combined parameters, is moved from its low level value to its high level value. In this section, we study these combined effects to evaluate which parameter(s) has the most effect on the overall system response time.

Table 6-2 summarizes the operating levels used in this overall effect study. The abbreviations are used later in the effect analysis graph. We combine the two buffering

control parameters: buffer replacement policy and prefetch policy, into one parameter, **Buffering Policy**, and include the user hint effect in the **Clustering policy**. The buffer pool size is fixed to 1000 in this study. For every parameter, two operating levels are used. Table 6-3 shows all the experiments used in this study.

Name	Abbrev	Levels	
Structure_Density	s	low_3	high_10
RW_ratio	W	5	100
Clustering policy	c	2IO_limit	NoIO_limit with User Hints
Page_split policy	P	Linear	NP-alg
Buffering policy	b	LRU	Context with Prefetch

Table 6-2: Factorial Operating Levels

	Factors					Response
I.D.	**StrucDen (arc)**	**RW_ratio**	**Clus_policy**	**Page_split**	**Buf_policy**	**Resp_time (sec)**
[1]	low_3	5	NoIO_limit	Linear	context	1.33
[2]	low_3	5	NoIO_limit	NP-alg	LRU	1.63
[3]	high_10	5	2IO_limit	Linear	context	1.65
[4]	low_3	100	2IO_limit	NP-alg	context	0.85
[5]	low_3	5	2IO_limit	Linear	context	1.32
[6]	high_10	5	2IO_limit	NP-alg	LRU	2.34
[7]	high_10	5	NoIO_limit	NP-alg	context	1.93
[8]	high_10	100	2IO_limit	Linear	context	1.19
[9]	low_3	5	2IO_limit	Linear	LRU	1.63
[10]	low_3	100	NoIO_limit	NP-alg	LRU	1.34
[11]	low_3	100	2IO_limit	Linear	LRU	1.30
[12]	high_10	5	NoIO_limit	Linear	LRU	2.63
[13]	low_3	100	NoIO_limit	Linear	LRU	1.30
[14]	high_10	100	2IO_limit	NP-alg	LRU	2.13
[15]	low_3	5	NoIO_limit	Linear	LRU	1.63
[16]	high_10	5	2IO_limit	NP-alg	context	1.65
[17]	high_10	100	NoIO_limit	NP-alg	context	0.98
[18]	high_10	5	NoIO_limit	NP-alg	LRU	2.63
[19]	high_10	100	NoIO_limit	NP-alg	LRU	1.88
[20]	high_10	5	2IO_limit	Linear	LRU	2.35
[21]	low_3	100	NoIO_limit	Linear	context	0.80
[22]	high_10	5	NoIO_limit	Linear	context	1.93
[23]	high_10	100	NoIO_limit	Linear	LRU	1.82
[24]	low_3	5	NoIO_limit	NP-alg	context	1.32
[25]	high_10	100	2IO_limit	NP-alg	context	1.24
[26]	high_10	100	2IO_limit	Linear	LRU	2.09
[27]	low_3	100	2IO_limit	NP-alg	LRU	1.35
[28]	low_3	100	2IO_limit	Linear	context	0.80
[29]	high_10	100	NoIO_limit	Linear	context	0.93
[30]	low_3	5	2IO_limit	NP-alg	context	1.33
[31]	low_3	5	2IO_limit	NP-alg	LRU	1.63
[32]	low_3	100	NoIO_limit	NP-alg	context	0.85

Table 6-3: 32 Runs for Overall Effect Analysis

Combined Effect	
Average	1.5575
StructDensity (s)	.5600
RW_ratio (W)	-.5050
Clustering_policy (c)	-.0050
Page_split (P)	.0250
Buffering_policy (b)	-.6000
sW	-.1000
sc	-.0050
sP	.0000
sb	-.2000
Wc	.1350
WP	.0250
Wb	-.1000
cP	.0000
cb	.0000
Pb	.0000
sWc	.1350
sWP	.0000
sWb	.0000
scP	.0000
sPb	.0000
WcP	.0000
Wcb	.0000
WPb	.0000
cPb	.0000
sWcP	.0000
sWcb	.0000
sWPb	.0000
scPb	.0000
WcPb	.0000
sWcPb	.0000

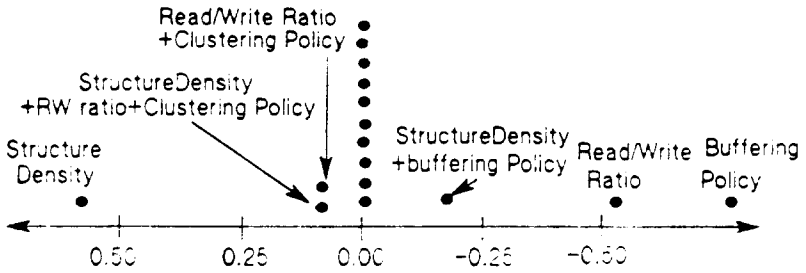

Table 6-4: Overall Effect Analysis

Larger absolute effect value means major effect on response time. A zero effect value implies that the corresponding control parameter(s) has a minimum effect on the overall response time. The left part lists all the effect values and the right part shows the effect analysis graph. We use "sW" to represent the combined effect of structure density (denoted by s) and read/write ratio (denoted by W). Each blob in the effect analysis graph represents the effect of a parameter or combined parameters on response time. For example, "structure density+buffering policy" represents a combined effect of the Structure density and the buffering policy whereas "Read/Write Ratio" is a single parameter effect. Most of the parameters or combined parameters have a small response time change and are represented by a line of blobs centered in the middle of the effect analysis graph.

Table 6-4 shows the overall effect analysis for the five control parameters specified in Table 6-2. There are 32 possible combinations, and for each one, we calculate the average response time change when the combined parameters go from a low operating level to a high operating level. Although the response time change can be positive or negative, only the absolute value is useful in this effect analysis.[17] Two interesting observations are: (1) the structure density and buffering policy most influence the system response time, and (2) different page splitting algorithms have little influence on the response time.

The key observations for the interaction analysis graphs are:

(1) There is no major interaction between any two factors. This means the control parameters we have chosen are quite independent.

(2) There are minor interactions between: structure density and buffering policy, read/write ratio and clustering policy, read/write ratio and page splitting policy, structure density and clustering policy, structure density and page splitting policy, page splitting policy and clustering policy, buffer pool size and prefetching policy, buffer pool size and buffering policy, user hints and clustering policy, page splitting policy, prefetching, and buffering policy.

(3) There is no interaction between: buffering policy and clustering policy, buffering policy and page splitting policy, structure density and read/write ratio, read/write ratio and buffering policy, and page splitting policy and clustering policy.

6.6. Summary and Conclusions

This chapter presents the simulation results of various clustering and buffering strategies' effect on overall response time. All these strategies are evaluated under different

[17] These values are used to determine the characteristics of control parameter interactions.

sets of control parameters to ensure an adequate interpretation of the simulation results. For example, the control parameter *clustering policy* (clustering with I/O limitation, no clustering and etc.) is evaluated under various structure densities, read/write ratios, and alternative prefetching policies. To understand the interactions among these eight control parameters, we use the interaction analysis graph to show whether two parameters interact strongly, interact weakly, or exhibit no interaction. We also study the overall effect on response time of these control parameter under various combinations. In the following, we summarize these findings in a set of tables, and show how these recommendation tables can be used to determine the best policy under a given workload.

Each table represents our recommendation for one clustering/buffering control parameter. For example, Table 6-5 represents the *clustering policy* (clustering with I/O limitation, no clustering, etc.), whereas Table 6-6 represents the *page splitting policy*. The first column of each table indicates the interaction between the various control parameters with the table's corresponding control parameter. For example, the three control parameters: *buffer replacement policy*, *prefetching policy* and *clustering policy* have minor or major interactions with *user hint policy* in Table 6-7. The number in the box represents our recommendation based on the simulation results: the smaller the number, the stronger the recommendation. The same recommendation number is used if two policies have comparable response time. For example, Table 6-5 shows that all three clustering policies: *Clustering with 2 I/O limit*, *Clustering with 10 I/O limit*, and *Clustering without I/O limit*, have similar response time under various read/write ratio, structure density, and prefetching policy, and they all outperform the other two clustering policies: *Clustering within buffer pool* and *No Clustering*.

For a given workload, such as low structure density and read/write ratio 5, how do we use these tables to choose the appropriate policy for every clustering/buffering control parameter? Using Table 6-5 and 6-8, we choose *Clustering without I/O limit* and *Context-sensitive* as the object clustering and buffer replacement policies. We then use Table 6-9 to determine that *Prefetching within database* is the preferred prefetching policy, since we are using *Context-sensitive* buffer replacement. Based on these decisions and Table 6-7, we decide not to use *User Hint*, which is consistent with the observations discussed in Section 6.3.1. The three *page splitting policies* have the same desirability if we examine the two rows labeled by R/W 5 and lowDensity in Table 6-6. Using the more detailed effect analysis graph of Figure 6-12 in Section 6.2, we choose *Linear Split* as the page splitting policy.

	Clustering Policy				
Read/Write Ratio	C_no_limit	C_2_IO	C_10_IO	C_withinBuf	No_Cluster
R/W 5	1	1	1	2	3
R/W 10	1	1	1	2	3
R/W 100	1	1	1	2	3
Structure Density					
lowDensity	1	1	1	2	3
medDensity	1	1	1	2	3
highDensity	1	1	1	2	3
Prefetching Policy					
No Prefetching	1	1	1	2	3
Prefetching	1	1	1	2	3

Table 6-5: Clustering Policy Summary

	Page Splitting Policy		
Read/Write Ratio	Linear Split	NP Split	No Split
R/W 5	2	2	1
R/W 10	2	2	1
R/W 100	1	2	3
Structure Density			
lowDensity	1	1	2
medDensity	1	2	3
highDensity	1	3	2

Table 6-6: Page Splitting Policy Summary

	User Hints Policy	
Buffer Replacement Policy	Use User Hints	No User Hints
Context-sensitive	2	1
LRU	1	2
Random	1	2
Prefetching Policy		
Prefetch within Buffer	1	2
Prefetch within DB	2	1
Clustering Policy		
Clustering without I/O limit	1	2
Clustering within Buffer	1	2

Table 6-7: User Hints Policy Summary

	Buffer Replacement Policy		
Read/Write Ratio	Context-sensitive	LRU	Random
R/W 5	1	2	2
R/W 10	1	2	2
R/W 100	1	2	2

Table 6-8: Buffer Replacement Policy Summary

	Prefetching Policy		
Buffer Replacement Policy	Within DB	Within Buffer	No Prefetch
Context-sensitive	1	3	3
LRU	1	1	3
Random	1	1	3

Table 6-9: Prefetching Policy Summary

CHAPTER 7

CONCLUSIONS AND FUTURE DIRECTIONS

Engineering design applications use structural relationships and inheritance mechanisms to effectively model their complex environments. By explicitly modeling these structural and inheritance semantics as first class objects in an object-oriented DBMS, this dissertation shows various ways to exploit them in the buffering and clustering phase to improve the overall system response time. To understand the characteristics of object-oriented applications, we have collected the access pattern information of more than ten CAD tools running on top of OCT, a structural object-oriented data manager developed by the UC-Berkeley CAD group. Based on the measurement results, we constructed a realistic sample database and an engineering DB model to evaluate the effectiveness of our clustering and buffering strategies. The proposed run-time clustering algorithm, under certain conditions, can improve the system response time by up to 200%. We also showed the effectiveness of limiting the amount of I/Os allowed to the clustering algorithm as it examines candidate pages for reclustering at run-time. We studied the prefetching effect on response time under various buffer replacement policies and transaction characteristics, and concluded that the context-sensitive buffer replacement policy with *prefetch within database* performs best whereas the *LRU with no prefetching* performs worst. We also studied the effectiveness of *User hints* and showed that *User hints* processing overhead must be taken into account in the overall system response time measurement.

Based on the simulation results shown in this dissertation, we have the following recommendations and research directions for existing or future object-oriented DBMSs:

(1) It is important to model the structural relationship and inheritance links as first class objects so their semantics can be exploited by the storage manager during the clustering and buffering phase. However, the physical implementation of structural relationships and inheritance links among objects needs further research. Moreover, an object-oriented query optimizer, such as OSQL [FISH87], needs to understand how structural relationships and inheritance links can be used to generate efficient access plans.

(2) Use *Clustering with 2 I/O limitation* as the default clustering policy, and change to *Clustering without I/O limitation* when stable response time is required.

(3) Adopt *Linear Split* as the basic page splitting policy when the target page is overflowed at run-time, and move to *No Split* when the read/write ratio is low.

(4) Do not use *User hints* if the *Context-sensitive* buffer replacement algorithm is used. Otherwise, *User hints* should be the default. The processing of complex user hints, such as "place object near this configuration", needs more study.

(5) Integrate the *Context-sensitive* buffer replacement algorithm with tunable prefetching policies. Use *Prefetching within database* as the default, and change to *Prefetching within buffer* if a larger buffer pool is used.

Based on the access patterns collected from OCT tools, an object-oriented application, we strongly recommend that OODBMS vendors and researchers to:

(1) Build the data collection process into the product/prototype so that the object usage, access pattern, and read/write ratio can be monitored and studied.

(2) Provide mechanism for users to register their access pattern information into the Object-Oriented DBMS so that underlying systems can be fine tuned.

(3) Refine the basic data types based on the object usage information.

(4) Propose benchmarks for object-oriented applications in a network environment.

The work reported in this dissertation represents an initial effort to validate the performance advantage claimed by object-oriented database researchers and vendors. However, more detailed validations should be done to understand the interactions among various clustering and buffering strategies. This is best accomplished by implementing these strategies in real world object-oriented DBMS products. I believe that the key to attaining better bandwidth and response time in both present and future database management systems is to effectively utilize the structural relationships and inheritance semantics in both clustering and buffering phases.

CHAPTER 8

BIBLIOGRAPHY

[ART86] Inference Inc. ART Manual. (1986)

[ATWO85] Atwood, T., "An Object Oriented DBMS for Design Support Applications," Proc. IEEE COMPINT 85, Montreal, Canada, (Sept. 1985).

[BANE88] Banerjee, J., H.T. Chou, J.F. Garza, W. Kim, D. Woelk, "Clustering a DAG for CAD Databases," IEEE Transactions on Software Engineering, Vol.14, No.11, 1988.

[BATO85] Batory, D., W. Kim, "Supporting Versions of VLSI CAD Objects," M.C.C. Technical Report, Austin, TX, (1985).

[BELA66] Belady, L.A., "A Study of Replacement Algorithms for a Virtual-Storage Computer," IBM Systems Journal, Vol.5., No.2, 1966.

[BERN87] Bernstein, P., "Database System Support for Software Engineering," ACM Software Engineering Conference, CA, (1987).

[BORN86] Borning, A., "Classes vs. Prototypes in Object-Oriented Languages," Proc. FJCC, Dallas, TX, (Nov. 1986).

[BRAC83] Brachman, R. J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," IEEE Computer Magazine, (October 1983).

[BUCH86] Buchmann, A., R. Carrera, M. Vazquez-Galindo, "A Generalized Constraint and Exception Handler for an Object-Oriented CAD DBMS," Proc. 1986 Intl. Workshop on Object-Oriented Database Systems, Asilomar, CA, (September 1986).

[BUTL87] Bulter, M.H., "Storage Reclamation in Object Oriented Database Systems," ACM SIGMOD Conf., San Francisco, CA, (May 1987).

[CARE86] Carey, M.J., D.J. DeWitt, D. Frank, G. Graefe, J.E. Richardson, E.J. Shekita, and M. Muralikrishna, "The Architecture of the EXODUS Extensible DBMS: A Preliminary Report," Proceedings of the International Workshop on Object-Oriented Database System, Sept. 1986.

[CATT88] Cattell, R.G.G. J.Skeen, "Engineering DBMS Benchmark," Sun Microsystems, Database Engineering Group (1988).

[CHAN82] Chan, A., S. Danberg, S. Fox, W.K. Lin, A. Nori, D. Ries, "Storage and Access Structures to Support a Semantic Data Model," 8th Very Large Database Conference, Mexico, (Sept. 1982).

[CHAN87a] Chang, E. E., R. H. Katz, "Inheritance in Computer-Aided Design Databases: Semantics and Implementation Issues," UCB Technical Report 87/377,(1987), *CAD Journal*, in press. (October 1989)

[CHAN87b] Chang, E. E., D. Gedye, R. H. Katz, "The Design and Implementation of a Version Server for Computer-Aided Design Databases," *Software Practice and Experience*, in press.

[DADA86] Dadam, P. et. al., "A DBMS Prototype to Support Extended NF2 Relations: An Integrated View on Flat Tables and Hierarchies," Proc. ACM SIGMOD Conf. Washington, D.C., May 1986.

[DAYA86] Dayal, Umeshwar, "PROBE -- A Research Project in Knowledge-Oriented Database Systems: A Preliminary Analysis," Technical Report CCA-85-03, Computer Coporation of America, July 1986.

[DENN72] Denning, P.J., Spirn, J.R., Savage, J.E., "Some Thoughts about Locality in Program Behavior," Proceedings of the Symposium on Computer Communications Networks and Teletraffic, Polytechnic Institute of Brooklyn, April 1972.

[DUHL88] Duhl, J., C. Damon, "A Performance Comparison of Object and Relational Databases Using the Sun Benchmark," Proc. OOPSLA'88 Conf., San Diego, CA, (Sept. 1988).

[FISH87] Fishman, D.H., D. Badal, D.Beech, E.Chow, T.Connors, C.G. Hoch, W. Kent, P. Lyngbaek, B. Mahbod, M.A. Neimat, T.A. Ryan, M.C. Shan, W.K. Wilkinson, "IRIS: An Object-Oriented DBMS," ACM Trans. on Office Information Systems, Vol. 5, No.1, January, 1987.

[GOLD83] Goldberg, A, D. Robson, "Smalltalk-80: The Language and Its Implementation," Addison-Wesley, 1983.

[HANS87] Hanson, E.N., "A Performance Analysis of View Materialization Strategies," ACM SIGMOD Conference, San Francisco, CA, (May 1987).

[HARR86] Harrison, David, P. Moore, R. Spickelmier, A.R. Newton, "Database Management and Graphics Editing in the Berkeley Design Environment", Proc. IEEE ICCAD, November 1986.

[HORN87] Hornick, M.F., S. Zdonik, "A Shared, Segmented Memory System for an Object-Oriented Database" ACM Transaction on Office Information Systems, Vol. 5, No. 1, January 1987, pages 70-95.

[KATZ87] Katz, R. H., E. Chang, "Managing Change in a Computer-Aided Design

Database," 13th Very Large Database Conference, Brighton, England, (Sept. 1987).

[KATZ86a] Katz, R. H., E. Chang, R. Bhateja, "Version Modeling Concepts for Computer-Aided Design Database," ACM SIGMOD Conf., Washington, DC, (May 1986).

[KATZ86b] Katz, R. H., E. Chang, M. Anwarrudin, "A Version Server for Computer-Aided Design Databases," ACM/IEEE 24th Design Automation Conf., Las Vegas, NV, (June 1986).

[KEE86] Intelli Corp. KEE Manual. (1986)

[KIM87] Kim, W., J. Banerjee, H. Chou, J.F. Garza, D. Woelk, "Composite Object Support in an Object-Oriented Database System," Proc. OOPSLA'87 Conf., Orlendo, FL, (Oct. 1987).

[KIM88] Kim, W., N. Ballou, H.T. Chou, J.F. Garza, D. Woelk, "Integrating an Object-Oriented Programming System with a Database System," Proc. OOPSLA'88 Conf., San Diego, CA, (Sept. 1988).

[KING86] King, R, Hudson, S. E., "CACTIS: A Database System for Specifying Functionally-Defined Data," Proc. International Workshop on Object-Oriented Database Systems, Pacific Grove, Ca, (Sept. 1986).

[LAND86] Landis, G.S., "Design Evolution and History in an Object-Oriented CAD/CAM Database," Proceedings 31st COMPCON Conference, San Francisco, CA (March 1986).

[LIEB86] Lieberman, H., "Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems," Proc. OOPSLA'86 Conf., Portland, OR, (Sept. 1986).

[LORI83] Lorie, R., W. Plouffe, "Complex Objects and their use in design transactions," Engineering Design Application Proceeding from SIGMOD Database Week, (May 1983).

[MAIE86] David Maier, Jacob Stein, Allen Otis, Alan Purdy, "Development of an Object-oriented DBMS," Technical Report CS/E-86-005, Oregon Graduate School (April, 1986).

[MITT86] Mittal, S. J., D. G. Bobrow, K. M. Kahn, "Virtual Copies: At the Boundary of Between Classes and Instances," Proc. OOPSLA'86 Conf., Portland, OR, (Sept. 1986).

[OMIE85] Omiecinski, Edward, "Incremental File Reorganization Schemes," 11th Very Large Database Conference, Stockholm, (1985).

[PAWS83] Information Research Associates, Austin Texas, 1983, "Performance Analyst's Workbench System (PAWS)".

[ROWE86] Rowe, Larry, "Shared Object Hierarchy," Proc. of 1st International Workshop on Object-oriented DBMS, Pacific Grove, CA, (Sept. 1986).

[ROWE87] Rowe, L., Stonebraker, M., "The POSTGRES Data Model," 13th Very Large Database Conference, Brighton, England, (Sept. 1987).

[RUBE87] Rubenstein, W. R., M. Kubicar, R.G.G. Cattell, "Benchmarking Simple Database Operations," Proc. ACM SIGMOD Conference, San Francisco, CA, (May 1987).

[SELL87] Sellis, Timos K., "Efficiently Supporting Procedures in Relational Database Systems," Proc. ACM SIGMOD Conference, San Francisco, CA, (May 1987).

[ZDON84] Zdonik, S. B., "Object Management System Concepts," Proc. 2nd SIGOA Conf. on Office Information Systems, Toronto, Canada, (June 1984).

APPENDIX A

PAWS IMPLEMENTATION

The PAWS implementation of the Engineering DB model is listed in the following:

global variables

gi(1) .. gi(1000) contains the page id in that buffer.
gi(1001).. gi(2000) contains the lock mode. (1 is read lock, 2 is write lock)
gr(1) .. gr(1000) contains the buffer priority.
gb(1) .. gb(1000) contains the dirty bit and other info.

local variables

li(1) contains the query type
li(2) contains the target object id
li(3) contains object size
li(4) contains page id for target object
li(5) contains source page id of its ancestor object
lb(1) is true if there is an ancestor object
lb(2) is true if there is a component object
lb(3) is true if there is a composite object
lb(4) is true if the transaction is done
lb(7) is true if the transaction is done with all I/Os
lb(8) is true if the transaction is not done with all buffer paging
lb(9) is true if the transaction is not done with all its I/O
lr(1) = terminal read/write priority(high)
lr(2) = disk i/o reads (next high)
lr(3) = transaction processing and write
lr(4) = not used (low priority)

Query Types:

1: simple object lookup (no structural relationship involved)
2: version history retrieval (descendant)
3: version history retrieval (ancestor)
4: configuration retrieval (component)
5: configuration retrieval (composite)
6: object insertion

```
declare
 integers
        nterm, nobuffer, oowtph, oordph, ioph, buffscanph,
        termrdph, termwtph, pageinph, pageoutph, xmph, logph,
        ntrans, ndisk, objidcount, cpuph, oldobjid, ooph,
        tempindex, querytype, hasstruct,tempinteger
        ;
```

```
reals
        think, xmlogpath,
        termrdpath, termwtpath, transpath, pageinpath, pageoutpath,
        iopath, ips, tempreal,markbufpath
        ;

nodes
        startopen,
        startnode,
        opensetup,      ! user node, open pagefile and objfile
        xactsetup,      ! set up priority for individual transaction
        fktermwt,
        cpu,
        jntermwt,
        term,
        fktermrd,
        jntermrd,
        selectload,     ! select the work load for individual xaction.
                        ! consists of few reads and writes.
        examload,       ! determine the clustering strategy and read/write.
        getweb,         ! user node, convert objid into pageid.
        bufferscan,
        storeweb,       ! user node, update objfile and pagefile
        clusternode,
        fwrite,
        fread,
        fkxm,
        jnxm,
        xm,
        fkpageout,
        disk,
        jnpageout,
        sendio,
        diskout,
        fkio,
        jnio,
        doneio,
        lock,
        unlock,
        fkusercpu,
        jnusercpu,
        wkin,
        wkout
        ;
categories
        x,              ! parent transaction,
        xxm,            ! transaction manager catagory,
        xin,            ! in page in state,
        xout,           ! in page out state,
        xterm,          ! in terminal related state,
        xio,            ! transaction i/o state,
        xuser,          ! user and other cpu burn.
        ;
```

```
tokens
        pages,bufferpool;
topology
        startopen       opensetup       <all,all>           1.0;
        opensetup       startnode       <all,all>           1.0;
        startnode       xactsetup       <x,termwtph>        1.0;
        xactsetup       fktermwt        <x,termwtph>        1.0;
        fktermwt        cpu             <xterm,termwtph>    1.0;
        cpu             jntermwt        <xterm,termwtph>    1.0;
        jntermwt        term            <x,termwtph>        1.0;
        term            fktermrd        <x,termwtph>        1.0;
        fktermrd        cpu             <xterm,termrdph>    1.0;
        cpu             jntermrd        <xterm,termrdph>    1.0;
        jntermrd        wkin            <x,termwtph>        1.0;
        wkin            selectload      <x,termwtph>        1.0;
        selectload      examload        <x,termwtph>        1.0;
        examload        getweb          <x,ooph>            1.0;
        getweb          bufferscan      <x,ooph>            0.0;
        getweb          clusternode     <x,ooph>            1.0;
        bufferscan      cpu             <x,buffscanph>      1.0;
        cpu             clusternode     <x,buffscanph>      1.0;
        clusternode     storeweb        <x,ioph>            1.0;
        clusternode     storeweb        <x,pageoutph>       1.0;
        clusternode     storeweb        <x,cpuph>           1.0;
        clusternode     storeweb        <x,xmph>            1.0;
        storeweb        fkio            <x,ioph>            if lb(9);
        storeweb        fkpageout       <x,pageoutph>       if lb(8);
        storeweb        fkusercpu       <x,cpuph>           if lb(7);
        storeweb        fkxm            <x,xmph>            1.0;
        fkxm            cpu             <xxm,xmph>          1.0;
        cpu             jnxm            <xxm,xmph>          1.0;
        jnxm            xm              <x,xmph>            1.0;
        xm              fkio            <x,logph>           1.0;
        xm              fkusercpu       <x,xmph>            1.0;
        fkio            cpu             <xio,ioph>          1.0;
        fkpageout       cpu             <xout,pageoutph>    1.0;
        cpu             sendio          <xio,ioph>          1.0;
        cpu             sendio          <xout,pageoutph>    1.0;
        sendio          disk            <all,all>           1.0;
        disk            diskout         <all,all>           1.0;
        diskout         jnpageout       <xout,pageoutph>    1.0;
        diskout         jnio            <xio,ioph>          1.0;
        jnpageout       fkio            <x,pageoutph>       1.0;
        jnio            clusternode     <x,all>             1.0;
        fkusercpu       cpu             <xuser,cpuph>       1.0;
        cpu             jnusercpu       <xuser,cpuph>       1.0;
        jnusercpu       wkout           <x,all>             1.0;
        wkout           fktermwt        <x,termwtph>        1.0;

define
        startopen
        type            service
        quantity 1
```

```
qd              delay
        request         <x,termwtph>  expo(eps);
                                !short launching time.

        startnode
        type            service
        quantity 1
        qd              delay
        request         <x,termwtph>  expo(eps);

        xactsetup
        type            compute
        request         <x,termwtph>
                add     ntrans  ntrans  1       !store tids
                add     tempindex nobuffer ntrans
                add     tempindex tempindex 1
                leteq   gi(tempindex)   tid

                leteq   lr(1)   1.0
                leteq   lr(2)   2.0
                leteq   lr(3)   3.0
                leteq   lr(4)   4.0
                ;
        opensetup
        ! open objfile and pagefile
        !
        type            user
        request         <all,all>       1;

        fktermwt
        type            fork
        request         <x,termwtph>  constant(1.0)<xterm,termwtph>;

        jntermwt
        type            join;

        fktermrd
        type            fork
        request         <x,termwtph>  constant(1.0)<xterm,termrdph>;

        jntermrd
        type            join;

        cpu
        type            service
        quantity 1
        qd              preemptive priority
        request <xterm,termrdph>        lr(1)   expo(termrdpath)
                <xterm,termwtph>        lr(1)   expo(termwtpath)
                <xio,ioph>              lr(3)   expo(iopath)
                <xout,pageoutph>        lr(2)   expo(pageoutpath)
                <xuser, cpuph>          lr(3)   expo(markbufpath)
                <xxm,xmph>              lr(3)   expo(xmlogpath);
```

```
disk
type            service
quantity ndisk
qd              fcfs
request <xio, ioph>             uniform(0.025,0.04)
        <xout,pageoutph>        uniform(0.025,0.04);

sendio
type            branch;

diskout
type            branch;

term
type            service
quantity nterm
qd              delay
request         <x,termwtph>  expo(think);

wkin
type            change
request <x,termwtph>  termwtph        1.0;

wkout
type            change
request <x,xmph>      termwtph        1.0
        <x,pageoutph> termwtph        1.0
        <x,ioph>      termwtph        1.0
        <x,cpuph>     termwtph        1.0;

examload
type            compute
request         <x,all>
        leteq   tphase  ooph
        ;

clusternode
type            user
request         <x,all>         4;

selectload
! randomly select read or write.  If read, randomly select
! an object id from existing objects.
! If write, increment the objectid counter.
! For write, we also generate object size and structural relationships.
! Then, the system would record these info in a user node.
type            compute
request         <x,all>
        eq      lb(4)   2       1
        eq      lb(5)   1       1
        leteq   querytype       fix(empirical(0.25,1.0,1.1,
                                              0.05,2.0,2.1,
                                              0.05,3.0,3.1,
```

```
                                        0.25,4.0,4.1,
                                              0.15,5.0,5.1,
                                              0.25,6.0,6.1))
        leteq   li(1)   querytype       ! remember it.
        neq     lb(10)  querytype 6     ! object insertion
        leteq   li(2)   objidcount
        leteq   li(3)   fix(expo(100.0))! obj size
        exit;

getweb
! for read query, determine the page-set containing its results.
! for write query, convert objid into pageid and attach to the transaction.
type            user
request         <x,ooph>        2;

bufferscan
! scan through the buffer pool.
! If there is a hit, change the tphase to lock the buffer.
! If there is no hit, we will determine a buffer to be reused
! and tphase may be changed to i/o if buffer is dirty.
! Otherwise, we will issue a lock for this buffer.
type            compute
request <x, cpuph>
        leteq   tempindex       1
        ;

storeweb
! update objfile and pagefile
!
type            user
request         <x,all>         3;

fkio
type            fork
request         <x,ioph>        constant(1.0)   <xio,ioph>
                <x,pageoutph>   constant(1.0)   <xio,ioph>
                <x,logph>       constant(1.0)   <xio,ioph>;

jnio
type            join;

fkxm
type            fork
request         <x,xmph>        constant(1.0)   <xxm,xmph>;

jnxm
type            join;

xm
type            compute
request         <x,xmph>
                leteq   tempinteger     iuniform(0,1)
                eq      lb(10)  tempinteger     1
```

```
            leteqif tphase  logph      lb(10)
                    leteqif tphase     xmph    lb(10)
                    exit;

        fkpageout
        type            fork
        request         <x,pageoutph> constant(1.0) <xout,pageoutph>;

        jnpageout
        type            join;

        fkusercpu
        type            fork
        request         <x,xmph> constant(1.0) <xuser,cpuph>
                        <x,ioph> constant(1.0) <xus  cpuph>
                        <x,cpuph> constant(1.0) <xuser,cpuph>
                        <x,pageoutph> constant(1.0) <xuser,cpuph>;

        jnusercpu
        type            join;

        doneio
        type            change
        request         <x,all>         cpuph   1.0;

        fread
        ! if there is no free, set tphase pageoutph
        type            compute
        request         <x,all>
                leteq   tempindex       1
                exit;

        fwrite
        ! if xm exist, set tphase to xm
        ! else set tphase to oowtph
        type            compute
        request         <x,all>
                leteq   tempindex       1
                exit;

        lock
        type            allocate
        quantity 1      bufferpool
        qd              fcfs
        request         <all,all>       constant(1)     bufferpool;

initial
  population
        <x,termwtph>  1               startopen
        <x,termwtph>  nterm           startnode;

statistics report
```

```
 ql      cpu
         response startnode startnode;

run
         leteq    objidcount       559
         leteq    nterm            10
         leteq    nobuffer         1000
         leteq    ips      5180000.0       ! ipsval
         leteq    termrdph         1
         leteq    termwtph         2
         leteq    cpuph            3
         leteq    pageinph         4
         leteq    pageoutph        5
         leteq    xmph             6
         leteq    logph            7
         leteq    oowtph           8
         leteq    oordph           9
         leteq    ioph             10
         leteq    buffscanph       11
         leteq    ooph             12
         leteq    ndisk            1
         leteq    ntrans           0
         leteq    think            4.0
         div      termrdpath   95245.0     ips   ! termrdval
         div      termwtpath   42785.0     ips   ! termwtval
         div      pageinpath   20000.0     ips   ! ioval
         div      pageoutpath  19000.0     ips   ! ioval
         div        markbufpath  1000.0       ips
         div      iopath       36800.0     ips   ! fwrite + post
         div        xmlogpath    5000.0       ips

  go      200.0  20.0
  dump
  clear
  go      800.0  50.0
  dump

  exit;
end;
```

APPENDIX B

PAWS REPORTS

page 1 ira paws - v3.0 summary statistics

produced by:
paws 3.0.06 - performance analysts workbench system,
information research associates, austin, texas.

page 2 ira paws - v3.0.06 (02-06-87) * summary statistics part 1

*** batch number: 1 batch duration = 200.000 (from 0.000 to 200.000) ***

		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
node:	/startopen (1)											
cat	/x	.500E-02	1.	0	.376E-08	1	0	.753E-06	.753E-06	.753E-06	.753E-06	.000
cat	/all	.500E-02	1.	0	.376E-08	1	0	.753E-06	.753E-06	.753E-06	.753E-06	.000
node:	/startnode (1)											
cat	/x	.205E+00	41.	0	.175E-06	40	0	.256E-07	.856E-06	.425E-05	.856E-06	.000
cat	/all	.205E+00	41.	0	.175E-06	40	0	.256E-07	.856E-06	.425E-05	.856E-06	.000
node:	/opensetup (1)											
cat	/x	.005	1.									
cat	/all	.005	1.									
node:	/xactsetup (1)											
cat	/x	.205	41.									
cat	/all	.205	41.									
node:	/fktermwt (1)											
cat	/xterm	6.805	1361.									
cat	/all	6.805	1361.									
node:	/cpu (1)											
cat	/xxm	.265E+01	532.	0	.469E+00	4	2	.229E-04	.176E+00	.131E+01	.886E-03	.236
cat	/xout	.410E+00	82.	0	.392E-02	3	0	.153E-04	.950E-02	.491E-01	.391E-02	.160
cat	/xterm	.135E+02	2694.	0	.227E+00	41	0	.572E-05	.169E-01	.306E+00	.130E-01	17.567
cat	/xio	.144E+02	2879.	0	.250E+01	7	4	.687E-04	.174E+00	.162E+01	.705E-02	10.151
cat	/xuser	3.945	789.	0	.976	5	1	.004	.247	1.641	.073	28.933
cat	/xcluster	.265E+01	531.	0	.720E+00	4	1	.273E-03	.271E+00	.133E+01	.144E+00	38.120
cat	/all	.375E+02	7507.	0	.490E+01	41	8	.572E-05	.130E+00	.164E+01	.254E-01	95.167
node:	/jntermwt (1)											
cat	/x	6.805	1361.									
cat	/all	6.805	1361.									
node:	/term (1)											
cat	/x	6.665	1333.	0	26.611	40	28	.005	3.928	31.867	3.928	261.394
cat	/all	6.665	1333.	0	26.611	40	28	.005	3.928	31.867	3.928	261.394
node:	/fktermrd (1)											
cat	/xterm	6.665	1333.									
cat	/all	6.665	1333.									
node:	/jntermrd (1)											
cat	/x	6.665	1333.									
cat	/all	6.665	1333.									
node:	/selectload(1)											
cat	/x	6.665	1333.									
cat	/all	6.665	1333.									
node:	/examload (1)											

page 3 ira paws - v3.0.06 (02-06-87) * summary statistics part 1

*** batch number: 1 batch duration = 200.000 (from 0.000 to 200.000) ***

		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
cat	/x	6.665	1333.									
cat	/all	6.665	1333.									
node:	/getweb (1)											
cat	/x	6.665	1333.									
cat	/all	6.665	1333.									
node:	/clusternod(1)											
cat	/x	21.035	4207.									
cat	/all	21.035	4207.									
node:	/fkxm (1)											
cat	/xxm	2.670	534.									
cat	/all	2.670	534.									
node:	/jnxm (1)											
cat	/x	2.660	532.									
cat	/all	2.660	532.									
node:	/xm (1)											
cat	/x	2.660	532.									
cat	/all	2.660	532.									
node:	/fkpageout (1)											
cat	/xout	.410	82.									
cat	/all	.410	82.									
node:	/disk (1)											
cat	/xout	.410	82.	0	.038	3	0	.026	.092	.228	.034	1.380
cat	/xio	14.395	2879.	0	1.304	8	0	.025	.091	.265	.032	46.563
cat	/all	14.805	2961.	0	1.342	8	0	.025	.091	.265	.032	47.943
node:	/jnpageout (1)											
cat	/x	.410	82.									
cat	/all	.410	82.									
node:	/sendio (1)											
cat	/xout	.410	82.									

```
cat          /xio               14.395    2879. *                                    *                                      *
cat          /all               14.805    2961. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /diskout    (  1)                *                                    *                                      *
cat          /xout                .410      82. *                                    *                                      *
cat          /xio               14.395    2879. *                                    *                                      *
cat          /all               14.805    2961. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /fkio       (  1)                *                                    *                                      *
cat          /xio               14.415    2883. *                                    *                                      *
cat          /all               14.415    2883. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
1
page    4       i r a   p a w s   -   v 3 . 0 . 06  ( 02 - 06 - 87 )   *   s u m m a r y   s t a t i s t i c s   p a r t   1


             *** batch number:       1 batch duration =     200.000 (from      0.000 to     200.000) ***

                              throughput            queue length                     queueing time                      service
                              rate      count  min     mean      max    end     min        mean          max          mean        util
node:          /jnio       (  1)                *                                    *                                      *
cat          /x                 14.395    2879. *                                    *                                      *
cat          /all               14.395    2879. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /lock       (  1)                *                                    *                                      *
cat          /x                  6.640    1328. *  0      8.151      23      5 *    0.000      1.225         4.496 *
cat          /all                6.640    1328. *  0      8.151      23      5 *    0.000      1.225         4.496 *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /unlock     (  1)                *                                    *                                      *
cat          /x                  6.600    1320. *                                    *                                      *
cat          /all                6.600    1320. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /unlockuser(  1)                 *                                    *                                      *
cat          /x                  6.600    1320. *                                    *                                      *
cat          /all                6.600    1320. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /fkusercpu  (  1)                *                                    *                                      *
cat          /xuser              3.950     790. *                                    *                                      *
cat          /xcluster           2.660     532. *                                    *                                      *
cat          /all                6.610    1322. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /jnusercpu  (  1)                *                                    *                                      *
cat          /x                  6.600    1320. *                                    *                                      *
cat          /all                6.600    1320. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /wkin       (  1)                *                                    *                                      *
cat          /x                  6.665    1333. *                                    *                                      *
cat          /all                6.665    1333. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /wkout      (  1)                *                                    *                                      *
cat          /x                  6.600    1320. *                                    *                                      *
cat          /all                6.600    1320. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /termrd     (  1)                *                                    *                                      *
cat          /x                  6.665    1333. *                                    *                                      *
cat          /all                6.665    1333. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
1

page    5       i r a   p a w s   -   v 3 . 0 . 06  ( 02 - 06 - 87 )   *   s u m m a r y   s t a t i s t i c s   p a r t   2


             *** batch number:       1 batch duration =     200.000 (from      0.000 to     200.000) ***

                                                              response time
                                                         min       mean          max
from           /termrd     (  1)  to            *      /wkout      (  1)
  cat:         /x                               *       .107      2.153        6.210 *                                   *
  cat:         /all                             *       .107      2.153        6.210 *                                   *
-------------------------------------------------------------------------------------------------------------------------------
1
page    6       i r a   p a w s   -   v 3 . 0 . 06  ( 02 - 06 - 87 )   *   s u m m a r y   s t a t i s t i c s   p a r t   3


             *** batch number:       1 batch duration =     200.000 (from      0.000 to     200.000) ***

                                                          token utilization
token:           /pages
  category                  utilization      mean # held
             /x                  0.00            0.000
             /xxm                0.00            0.000
             /xin                0.00            0.000
             /xout               0.00            0.000
             /xterm              0.00            0.000
             /xio                0.00            0.000
             /xuser              0.00            0.000
             /xcluster           0.00            0.000
  all                            0.00            0.000
token:           /bufferpool
  category                  utilization      mean # held
             /x                  0.00            0.000
             /xxm                6.20            1.861
             /xin                0.00            0.000
             /xout                .61             .182
             /xterm              0.00            0.000
             /xio               61.27           18.382
             /xuser             13.92            4.175
             /xcluster           9.90            2.970
  all                           91.90           27.570
1
page    1       i r a   p a w s   -   v 3 . 0 . 06  ( 02 - 06 - 87 )   *   s u m m a r y   s t a t i s t i c s   p a r t   1


             *** batch number:       2 batch duration =     200.000 (from    200.000 to     400.000) ***

                              throughput            queue length                     queueing time                      service
                              rate      count  min     mean      max    end     min        mean          max          mean        util
node:          /fktermwt   (  1)                *                                    *                                      *
cat          /xterm              6.510    1302. *                                    *                                      *
cat          /all                6.510    1302. *                                    *                                      *
-------------------------------------------------------------------------------------------------------------------------------------
node:          /cpu        (  1)                *                                    *                                      *
cat          /xxm             .270E+01     540. *  0   .485E+00      5      0 *  .000E+00   .180E+00     .113E+01 *   .102E-02        .276
```

cat	/xout	.585E+00	117.	*	0	.520E-02	2	0	*	.916E-04	.888E-02	.541E-01	*	.364E-02	.213	
cat	/xterm	13.055	2611.	*	0	.195	4	0	*	0.000	.015	.120	*	.014	17.638	
cat	/xio	14.230	2846.	*	0	2.488	8	2	*	0.000	.175	1.717	*	.007	10.506	
cat	/xuser	.382E+01	763.	*	0	.947E+00	5	0	*	.122E-03	.248E+00	.112E+01	*	.787E-01	30.021	
cat	/xcluster	.269E+01	539.	*	0	.737E+00	5	2	*	.113E-02	.273E+00	.170E+01	*	.134E+00	36.130	
cat	/all	37.080	7416.	*	0	4.858	11	4	*	0.000	.131	1.717	*	.026	94.784	
node:	/jntermwt (1)			*					*				*			
cat	/x	6.510	1302.	*					*				*			
cat	/all	6.510	1302.	*					*				*			
node:	/term (1)			*					*				*			
cat	/x	6.545	1309.	*	14	26.729	40	21	*	.021	4.047	29.033	*	4.047	2648.571	
cat	/all	6.545	1309.	*	14	26.729	40	21	*	.021	4.047	29.033	*	4.047	2648.571	
node:	/fktermrd (1)			*					*				*			
cat	/xterm	6.545	1309.	*					*				*			
cat	/all	6.545	1309.	*					*				*			
node:	/jntermrd (1)			*					*				*			
cat	/x	6.545	1309.	*					*				*			
cat	/all	6.545	1309.	*					*				*			
node:	/selectload(1)			*					*				*			
cat	/x	6.545	1309.	*					*				*			
cat	/all	6.545	1309.	*					*				*			
node:	/examload (1)			*					*				*			
cat	/x	6.545	1309.	*					*				*			
cat	/all	6.545	1309.	*					*				*			
node:	/getweb (1)			*					*				*			
cat	/x	6.545	1309.	*					*				*			
cat	/all	6.545	1309.	*					*				*			
node:	/clusternod(1)			*					*				*			
cat	/x	20.720	4144.	*					*				*			
cat	/all	20.720	4144.	*					*				*			
node:	/fkxm (1)			*					*				*			
cat	/xxm	2.690	538.	*					*				*			
cat	/all	2.690	538.	*					*				*			
node:	/jfxm (1)			*					*				*			

*** batch number: 2 batch duration = 200.000 (from 200.000 to 400.000) ***

		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
cat	/x	2.700	540.	*				*			*	
cat	/all	2.700	540.	*				*			*	
node:	/xm (1)			*				*			*	
cat	/x	2.700	540.	*				*			*	
cat	/all	2.700	540.	*				*			*	
node:	/fkpageout (1)			*				*			*	
cat	/xout	.585	117.	*				*			*	
cat	/all	.585	117.	*				*			*	
node:	/disk (1)			*				*			*	
cat	/xout	.585	117.	*	0	.051	3	0	*	.025	.086	.249	*	.033	1.931	
cat	/xio	14.220	2844.	*	0	1.352	9	2	*	.025	.095	.260	*	.033	46.287	
cat	/all	14.805	2961.	*	0	1.402	9	2	*	.025	.095	.260	*	.033	48.218	
node:	/jnpageout (1)			*					*				*			
cat	/x	.585	117.	*					*				*			
cat	/all	.585	117.	*					*				*			
node:	/sendio (1)			*					*				*			
cat	/xout	.585	117.	*					*				*			
cat	/xio	14.230	2846.	*					*				*			
cat	/all	14.815	2963.	*					*				*			
node:	/diskout (1)			*					*				*			
cat	/xout	.585	117.	*					*				*			
cat	/xio	14.220	2844.	*					*				*			
cat	/all	14.805	2961.	*					*				*			
node:	/fkio (1)			*					*				*			
cat	/xio	14.220	2844.	*					*				*			
cat	/all	14.220	2844.	*					*				*			
node:	/jnio (1)			*					*				*			
cat	/x	14.220	2844.	*					*				*			
cat	/all	14.220	2844.	*					*				*			
node:	/lock (1)			*					*				*			
cat	/x	6.500	1300.	*	0	8.011	21	14	*	0.000	1.226	3.038	*			
cat	/all	6.500	1300.	*	0	8.011	21	14	*	0.000	1.226	3.038	*			
node:	/unlock (1)			*					*				*			
cat	/x	6.510	1302.	*					*				*			
cat	/all	6.510	1302.	*					*				*			
node:	/unlockuser(1)			*					*				*			
cat	/x	6.510	1302.	*					*				*			
cat	/all	6.510	1302.	*					*				*			

*** batch number: 2 batch duration = 200.000 (from 200.000 to 400.000) ***

		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
node:	/fkusercpu (1)			*					*			*		
cat	/xuser	3.810	762.	*					*			*		
cat	/xcluster	2.700	540.	*					*			*		
cat	/all	6.510	1302.	*					*			*		
node:	/jnusercpu (1)			*					*			*		

cat	/x	6.510	1302.
cat	/all	6.510	1302.
node:	/wkin (1)		
cat	/x	6.545	1309.
cat	/all	6.545	1309.
node:	/wkout (1)		
cat	/x	6.510	1302.
cat	/all	6.510	1302.
node:	/termrd (1)		
cat	/x	6.545	1309.
cat	/all	6.545	1309.

1

*** batch number: 2 batch duration = 200.000 (from 200.000 to 400.000) ***

from /termrd (1) to /wkout (1)		response time min	mean	max
cat:	/x	.137	2.178	5.298
cat:	/all	.137	2.178	5.298

1

*** batch number: 2 batch duration = 200.000 (from 200.000 to 400.000) ***

token utilization

token: /pages category		utilization	mean # held
	/x	0.00	0.000
	/xxm	0.00	0.000
	/xin	0.00	0.000
	/xout	0.00	0.000
	/xterm	0.00	0.000
	/xio	0.00	0.000
	/xuser	0.00	0.000
	/xcluster	0.00	0.000
all		0.00	0.000

token: /bufferpool category		utilization	mean # held
	/x	0.00	0.000
	/xxm	6.59	1.977
	/xin	0.00	0.000
	/xout	.96	.287
	/xterm	0.00	0.000
	/xio	61.90	18.570
	/xuser	13.04	3.913
	/xcluster	10.03	3.010
all		92.52	27.757

1

*** batch number: 2 batch duration = 400.000 (from 200.000 to 600.000) ***

		throughput rate	count	queue length min	mean	max	end	queueing time min	mean	max	service mean	util
node:	/fktermwt (1)											
cat	/xterm	6.375	2550.									
cat	/all	6.375	2550.									
node:	/cpu (1)											
cat	/xxm	.259E+01	1034.	0	.431E+00	5	0	.000E+00	.167E+00	.153E+01	.986E-03	.255
cat	/xout	.680	272.	0	.007	2	0	0.000	.011	.119	.004	.277
cat	/xterm	12.740	5096.	0	.191	4	1	0.000	.015	.129	.013	17.127
cat	/xio	14.618	5847.	0	2.382	8	4	0.000	.163	1.717	.007	10.709
cat	/xuser	.379E+01	1515.	0	.911E+00	5	0	.305E-04	.241E+00	.112E+01	.781E-01	29.588
cat	/xcluster	.259E+01	1035.	0	.695E+00	5	0	.122E-03	.269E+00	.170E+01	.139E+00	35.977
cat	/all	36.998	14799.	0	4.618	11	5	0.000	.125	1.717	.025	93.932
node:	/jntermwt (1)											
cat	/x	6.372	2549.									
cat	/all	6.372	2549.									
node:	/term (1)											
cat	/x	6.367	2547.	13	25.703	41	30	.004	4.031	39.012	4.031	2566.936
cat	/all	6.367	2547.	13	25.703	41	30	.004	4.031	39.012	4.031	2566.936
node:	/fktermrd (1)											
cat	/xterm	6.367	2547.									
cat	/all	6.367	2547.									
node:	/jntermrd (1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									
node:	/selectload(1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									
node:	/examload (1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									
node:	/getweb (1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									
node:	/clusternod(1)											
cat	/x	20.983	8393.									
cat	/all	20.983	8393.									
node:	/fkxm (1)											
cat	/xxm	2.580	1032.									
cat	/all	2.580	1032.									
node:	/jnxm (1)											

*** batch number: 2 batch duration = 400.000 (from 200.000 to 600.000) ***

		throughput rate	count	queue length min	mean	max	end	queueing time min	mean	max	service mean	util
cat	/x	2.585	1034.									
cat	/all	2.585	1034.									
node:	/xm (1)											
cat	/x	2.585	1034.									
cat	/all	2.585	1034.									
node:	/fkpageout (1)											
cat	/xout	.680	272.									
cat	/all	.680	272.									
node:	/disk (1)											
cat	/xout	.680	272.									
cat	/xio	14.610	5844.	0	.061	5	0	.025	.089	.249	.033	2.262
cat	/all	15.290	6116.	0	1.417	9	3	.025	.097	.268	.033	47.577
					1.477	9	3	.025	.097	.268	.033	49.840
node:	/jnpageout (1)											
cat	/x	.680	272.									
cat	/all	.680	272.									
node:	/sendio (1)											
cat	/xout	.680	272.									
cat	/xio	14.618	5847.									
cat	/all	15.297	6119.									
node:	/diskout (1)											
cat	/xout	.680	272.									
cat	/xio	14.610	5844.									
cat	/all	15.290	6116.									
node:	/fkio (1)											
cat	/xio	14.618	5847.									
cat	/all	14.618	5847.									
node:	/jnio (1)											
cat	/x	14.610	5844.									
cat	/all	14.610	5844.									
node:	/lock (1)											
cat	/x	6.372	2549.									
cat	/all	6.372	2549.	0	9.203	22	3	0.000	1.445	4.329		
				0	9.203	22	3	0.000	1.445	4.329		
node:	/unlock (1)											
cat	/x	6.375	2550.									
cat	/all	6.375	2550.									
node:	/unlockuser(1)											
cat	/x	6.375	2550.									
cat	/all	6.375	2550.									

*** batch number: 2 batch duration = 400.000 (from 200.000 to 600.000) ***

		throughput rate	count	queue length min	mean	max	end	queueing time min	mean	max	service mean	util
node:	/fkusercpu (1)											
cat	/xuser	3.785	1514.									
cat	/xcluster	2.585	1034.									
cat	/all	6.370	2548.									
node:	/jnusercpu (1)											
cat	/x	6.375	2550.									
cat	/all	6.375	2550.									
node:	/wkin (1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									
node:	/wkout (1)											
cat	/x	6.375	2550.									
cat	/all	6.375	2550.									
node:	/termrd (1)											
cat	/x	6.367	2547.									
cat	/all	6.367	2547.									

*** batch number: 2 batch duration = 400.000 (from 200.000 to 600.000) ***

from	/termrd (1) to	response time min /wkout (1)	mean	max
cat:	/x			
cat:	/all	.125	2.395	7.471

*** batch number: 2 batch duration = 400.000 (from 200.000 to 600.000) ***

token: /pages

category		token utilization	mean # held
	/x	0.00	0.000
	/xxm	0.00	0.000
	/xin	0.00	0.000
	/xout	0.00	0.000

	/xterm	0.00	0.000
	/xio	0.00	0.000
	/xuser	0.00	0.000
	/xcluster	0.00	0.000
all		0.00	0.000

token: /bufferpool

category	utilization	mean # held
/x	0.00	0.000
/xxm	6.17	1.850
/xin	0.00	0.000
/xout	1.14	.341
/xterm	0.00	0.000
/xio	62.09	18.627
/xuser	13.03	3.909
/xcluster	9.95	2.984
all	92.37	27.711

*** batch number: 2 batch duration = 600.000 (from 200.000 to 800.000) ***

node:		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
cat	/fktermwt (1)											
cat	/xterm	6.412	3847.									
cat	/all	6.412	3847.									
node:	/cpu (1)											
cat	/xxm	.260E+01	1559.	0	.427E+00	5	0	.000E+00	.164E+00	.153E+01	.966E-03	.251
cat	/xout	.623	374.	0	.007	2	0	0.000	.011	.177	.004	.247
cat	/xterm	12.812	7687.	0	.193	4	0	0.000	.015	.190	.014	17.380
cat	/xio	14.710	8826.	0	2.385	9	4	0.000	.162	1.717	.007	10.712
cat	/xuser	.381E+01	2288.	0	.918E+00	5	1	.305E-04	.241E+00	.167E+01	.774E-01	29.527
cat	/xcluster	.260E+01	1559.	0	.691E+00	5	1	.122E-03	.266E+00	.170E+01	.138E+00	35.938
cat	/all	37.155	22293.	0	4.621	11	6	0.000	.124	1.717	.025	94.055
node:	/jntermwt (1)											
cat	/x	6.412	3847.									
cat	/all	6.412	3847.									
node:	/term (1)											
cat	/x	.640E+01	3840.	13	.258E+02	41	35	.549E-03	.401E+01	.390E+02	.401E+01	2568.307
cat	/all	.640E+01	3840.	13	.258E+02	41	35	.549E-03	.401E+01	.390E+02	.401E+01	2568.307
node:	/fktermrd (1)											
cat	/xterm	6.400	3840.									
cat	/all	6.400	3840.									
node:	/jntermrd (1)											
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									
node:	/selectload(1)											
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									
node:	/examload (1)											
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									
node:	/getweb (1)											
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									
node:	/clusternod(1)											
cat	/x	21.118	12671.									
cat	/all	21.118	12671.									
node:	/fkxm (1)											
cat	/xxm	2.595	1557.									
cat	/all	2.595	1557.									
node:	/jnxm (1)											

*** batch number: 2 batch duration = 600.000 (from 200.000 to 800.000) ***

cat		throughput rate	throughput count	queue length min	queue length mean	queue length max	queue length end	queueing time min	queueing time mean	queueing time max	service mean	service util
cat	/x	2.598	1559.									
cat	/all	2.598	1559.									
node:	/xm (1)											
cat	/x	2.598	1559.									
cat	/all	2.598	1559.									
node:	/fkpageout (1)											
cat	/xout	.623	374.									
cat	/all	.623	374.									
node:	/disk (1)											
cat	/xout	.623	374.	0	.056	5	0	.025	.090	.249	.033	2.063
cat	/xio	14.710	8826.	0	1.426	9	0	.025	.097	.268	.033	47.953
cat	/all	15.333	9200.	0	1.482	9	0	.025	.097	.268	.033	50.016
node:	/jnpageout (1)											
cat	/x	.623	374.									
cat	/all	.623	374.									
node:	/sendio (1)											
cat	/xout	.623	374.									
cat	/xio	14.710	8826.									
cat	/all	15.333	9200.									
node:	/diskout (1)											
cat	/xout	.623	374.									
cat	/xio	14.710	8826.									
cat	/all	15.333	9200.									
node:	/fkio (1)											
cat	/xio	14.710	8826.									
cat	/all	14.710	8826.									

node:	/jnio	(1)											
cat	/x	14.710	8826.										
cat	/all	14.710	8826.										
node:	/lock	(1)											
cat	/x	6.408	3845.										
cat	/all	6.408	3845.	0	9.091	22	0	0.000	1.420	4.329			
node:	/unlock	(1)											
cat	/x	6.412	3847.										
cat	/all	6.412	3847.										
node:	/unlockuser	(1)											
cat	/x	6.412	3847.										
cat	/all	6.412	3847.										

1
page 3 i r a p a w s - v 3 . 0 . 06 (02 - 06 - 87) * s u m m a r y s t a t i s t i c s p a r t 1

*** batch number: 2 batch duration = 600.000 (from 200.000 to 800.000) ***

		throughput		queue length				queueing time			service	
node:	/fkusercpu	(1) rate	count	min	mean	max	end	min	mean	max	mean	util
cat	/xuser	3.813	2288.									
cat	/xcluster	2.598	1559.									
cat	/all	6.412	3847.									
node:	/jnusercpu	(1)										
cat	/x	6.412	3847.									
cat	/all	6.412	3847.									
node:	/wkin	(1)										
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									
node:	/wkout	(1)										
cat	/x	6.412	3847.									
cat	/all	6.412	3847.									
node:	/termrd	(1)										
cat	/x	6.400	3840.									
cat	/all	6.400	3840.									

1
page 4 i r a p a w s - v 3 . 0 . 06 (02 - 06 - 87) * s u m m a r y s t a t i s t i c s p a r t 2

*** batch number: 2 batch duration = 600.000 (from 200.000 to 800.000) ***

			response time		
from	/termrd	(1) to	min	mean	max
cat:	/x		/wkout	(1)	
cat:	/all		.116	2.365	7.471

1
page 5 i r a p a w s - v 3 . 0 . 06 (02 - 06 - 87) * s u m m a r y s t a t i s t i c s p a r t 3

*** batch number: 2 batch duration = 600.000 (from 200.000 to 800.000) ***

token: /pages — token utilization

category	utilization	mean # held
/x	0.00	0.000
/xxm	0.00	0.000
/xin	0.00	0.000
/xout	0.00	0.000
/xterm	0.00	0.000
/xio	0.00	0.000
/xuser	0.00	0.000
/xcluster	0.00	0.000
all	0.00	0.000

token: /bufferpool — token utilization

category	utilization	mean # held
/x	0.00	0.000
/xxm	6.12	1.836
/xin	0.00	0.000
/xout	1.04	.312
/xterm	0.00	0.000
/xio	62.34	18.702
/xuser	13.13	3.939
/xcluster	9.84	2.951
all	92.47	27.740

1
page 1 i r a p a w s - v 3 . 0 . 06 (02 - 06 - 87) * s u m m a r y s t a t i s t i c s p a r t 1

*** batch number: 2 batch duration = 800.000 (from 200.000 to 1000.000) ***

		throughput		queue length				queueing time			service	
node:	/fktermwt	(1) rate	count	min	mean	max	end	min	mean	max	mean	util
cat	/xterm	6.358	5086.									
cat	/all	6.358	5086.									
node:	/cpu	(1)										
cat	/xxm	.257E+01	2053.	0	.427E+00	5	0	.000E+00	.167E+00	.153E+01	.966E-03	.248
cat	/xout	.668	534.	0	.007	2	0	0.000	.010	.177	.004	.254
cat	/xterm	12.717	10174.	0	.190	4	0	0.000	.015	.190	.007	10.716
cat	/xio	14.861	11889.	0	2.340	8	5	0.000	.157	1.717	.013	17.155
cat	/xuser	.379E+01	3032.	0	.908E+00	5	3	.305E-04	.239E+00	.167E+01	.774E-01	29.351
cat	/xcluster	.257E+01	2054.	0	.675E+00	5	0	.122E-03	.263E+00	.170E+01	.139E+00	35.641
cat	/all	37.170	29736.	0	4.548	11	8	0.000	.122	1.717	.025	93.365
node:	/jntermwt	(1)										
cat	/x	6.358	5086.									
cat	/all	6.358	5086.									
node:	/term	(1)										

cat	/x	.636E+01	5088.	*	12	.255E+02	41	26	*	.549E-03	.401E+01	.517E+02	*	.401E+01	2549.005	
cat	/all	.636E+01	5088.	*	12	.255E+02	41	26	*	.549E-03	.401E+01	.517E+02	*	.401E+01	2549.005	
node:	/fktermrd (1)			*					*				*			
cat	/xterm	6.360	5088.	*					*				*			
cat	/all	6.360	5088.	*					*				*			
node:	/jntermrd (1)			*					*				*			
cat	/x	6.360	5088.	*					*				*			
cat	/all	6.360	5088.	*					*				*			
node:	/selectload(1)			*					*				*			
cat	/x	6.360	5088.	*					*				*			
cat	/all	6.360	5088.	*					*				*			
node:	/examload (1)			*					*				*			
cat	/x	6.360	5088.	*					*				*			
cat	/all	6.360	5088.	*					*				*			
node:	/getweb (1)			*					*				*			
cat	/x	6.360	5088.	*					*				*			
cat	/all	6.360	5088.	*					*				*			
node:	/clusternod(1)			*					*				*			
cat	/x	21.219	16975.	*					*				*			
cat	/all	21.219	16975.	*					*				*			
node:	/fkxm (1)			*					*				*			
cat	/xxm	2.564	2051.	*					*				*			
cat	/all	2.564	2051.	*					*				*			
node:	/jnxm (1)			*					*				*			

*** batch number: 2 batch duration = 800.000 (from 200.000 to 1000.000) ***

		throughput			queue length				queueing time				service	
		rate	count	min	mean	max	end	min	mean	max	mean	util
cat	/x	2.566	2053.	*				*			*			
cat	/all	2.566	2053.	*				*			*			
node:	/xm (1)			*				*			*			
cat	/x	2.566	2053.	*				*			*			
cat	/all	2.566	2053.	*				*			*			
node:	/fkpageout (1)			*				*			*			
cat	/xout	.668	534.	*				*			*			
cat	/all	.668	534.	*				*			*			
node:	/disk (1)			*				*			*			
cat	/xout	.668	534.	*	0	.061	5	0 *	.025	.091	.249	*	.033	2.197
cat	/xio	14.861	11889.	*	0	1.458	9	0 *	.025	.098	.287	*	.033	48.442
cat	/all	15.529	12423.	*	0	1.519	9	0 *	.025	.098	.287	*	.033	50.639
node:	/jnpageout (1)			*				*			*			
cat	/x	.668	534.	*				*			*			

cat	/all	.668	534.	*					*				*			
node:	/sendio (1)			*					*				*			
cat	/xout	.668	534.	*					*				*			
cat	/xio	14.861	11889.	*					*				*			
cat	/all	15.529	12423.	*					*				*			
node:	/diskout (1)			*					*				*			
cat	/xout	.668	534.	*					*				*			
cat	/xio	14.861	11889.	*					*				*			
cat	/all	15.529	12423.	*					*				*			
node:	/fkio (1)			*					*				*			
cat	/xio	14.863	11890.	*					*				*			
cat	/all	14.863	11890.	*					*				*			
node:	/jnio (1)			*					*				*			
cat	/x	14.861	11889.	*					*				*			
cat	/all	14.861	11889.	*					*				*			
node:	/lock (1)			*					*				*			
cat	/x	6.358	5086.	*	0	9.400	22	7	*	0.000	1.479	4.329	*			
cat	/all	6.358	5086.	*	0	9.400	22	7	*	0.000	1.479	4.329	*			
node:	/unlock (1)			*					*				*			
cat	/x	6.358	5086.	*					*				*			
cat	/all	6.358	5086.	*					*				*			
node:	/unlockuser(1)			*					*				*			
cat	/x	6.358	5086.	*					*				*			
cat	/all	6.358	5086.	*					*				*			

*** batch number: 2 batch duration = 800.000 (from 200.000 to 1000.000) ***

		throughput			queue length				queueing time				service	
		rate	count	min	mean	max	end	min	mean	max	mean	util
node:	/fkusercpu (1)			*				*			*			
cat	/xuser	3.793	3034.	*				*			*			
cat	/xcluster	2.566	2053.	*				*			*			
cat	/all	6.359	5087.	*				*			*			
node:	/jnusercpu (1)			*				*			*			
cat	/x	6.358	5086.	*				*			*			
cat	/all	6.358	5086.	*				*			*			
node:	/wkin (1)			*				*			*			
cat	/x	6.360	5088.	*				*			*			
cat	/all	6.360	5088.	*				*			*			
node:	/wkout (1)			*				*			*			
cat	/x	6.358	5086.	*				*			*			
cat	/all	6.358	5086.	*				*			*			
node:	/termrd (1)			*				*			*			

cat	/x	6.360	5088. *	
cat	/all	6.360	5088. *	

1

*** batch number: 2 batch duration = 800.000 (from 200.000 to 1000.000) ***

from	/termrd (1) to		response time min	mean	max
			/wkout	(1)	
cat:	/x	*	.116	2.424	7.471 *
cat:	/all	*	.116	2.424	7.471 *

1

*** batch number: 2 batch duration = 800.000 (from 200.000 to 1000.000) ***

token utilization

token: /pages

category		utilization	mean # held
	/x	0.00	0.000
	/xxm	0.00	0.000
	/xin	0.00	0.000
	/xout	0.00	0.000
	/xterm	0.00	0.000
	/xio	0.00	0.000
	/xuser	0.00	0.000
	/xcluster	0.00	0.000
all		0.00	0.000

token: /bufferpool

category		utilization	mean # held
	/x	0.00	0.000
	/xxm	6.24	1.872
	/xin	0.00	0.000
	/xout	1.12	.336
	/xterm	0.00	0.000
	/xio	62.50	18.749
	/xuser	13.03	3.910
	/xcluster	9.69	2.906
all		92.57	27.772

1