Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS

Ellis E. Chang, Randy H. Katz

Report No. UCB/CSD 88/473

November 1988

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS

Ellis E. Chang, Randy H. Katz

Computer Science Division
Electrical Engineering and Computer Science Department
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT Object-oriented databases provide new kinds of data semantics in terms of inheritance and structural relationships. In this paper, we examine how to make use of the additional semantics to obtain more effective object buffering and clustering. We have instrumented real-world object-oriented applications, i.e., the Berkeley CAD Group's OCT design tools, and have used the collected information as the basis for a simulation model in which to investigate alternative buffering and clustering strategies. We have observed from our measurements that real CAD applications exhibit high data read to write ratios. We propose a run-time reclustering algorithm whose initial evaluation indicates that system response time can be improved by a factor of 200% when the read/write ratio is high. We have also found it useful to limit the amount of I/O allowed to the clustering algorithm as it examines candidate pages for reclustering at run-time. Basically, there is little performance distinction between limiting reclustering to a few I/Os or many, so 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.

KEY WORDS AND PHRASES: Object-oriented DBMS, Design Databases, Clustering, Context-sensitive buffering

1. Introduction

Object-oriented concepts, developed originally in Smalltalk, have been accepted by database researchers as the basis for dealing with semantics-rich applications like CAD/CAM. The key aspect of an object-oriented system is the mapping from an application's logic into a set of abstract data types, with associated operations and attributes. In addition, information (operation/attribute definitions) can be propagated along the lattice formed by instances, types and supertypes through inheritance.

These object-oriented concepts have been further enhanced to support complex object, version and correspondence management, and new ways to propagate information between instances [KATZ87, CHAN87a]. For example, in our Version Data Model, we explicitly support three structural relationships, i.e. configuration, version history and correspondence (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.

We have proposed a new *instance-to-instance* inheritance in which information can be propagated among instances along structural relationships. For instance, 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 its type. Consider the following example. If ALU[2].layout corresponds to ALU[3].netlist, then a new descendant of ALU[2].layout should inherit this correspondence relationship by default. Just as the relational query optimizer uses the table cardinality and indexing information to produce efficient access plans, object-oriented systems could also use knowledge of structural relationships and inheritance to improve system performance [CHAN87a].

In this paper, we are particularly interested in how structural relationships can be used by clustering and buffering algorithms to improve DBMS response time. For instance, a design browser may walk through multiple representations of the same design objects, and clustering across correspondence is advantageous. Alternatively, a simulation tool traverses the net list representation hierarchy, and clustering along the configuration hierarchy is best. 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.

Several object-oriented database management systems have implemented object-based clustering mechanisms [MAIE86, ATWO85, ZDON84, KIM87]. The common characteristics of these are: (1) the segment is the clustering unit, and (2) user's hints are used, but only at object creation time. For instance, users may provide a hint such as "place near object XX". The system would then try to store the target object with object XX in the same page or adjacent page. Since these systems do not model structural relationships as first class objects, the storage component has no structural information to exploit during clustering. That is, users' hints are the only useful semantics used by the storage component. Furthermore, physical placement hints based on instance-to-instance inheritance are not exploited by any system. Neither do these systems consider reclustering when object structures are changed.

We will describe a run-time reclustering algorithm which can improve overall system response time by 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

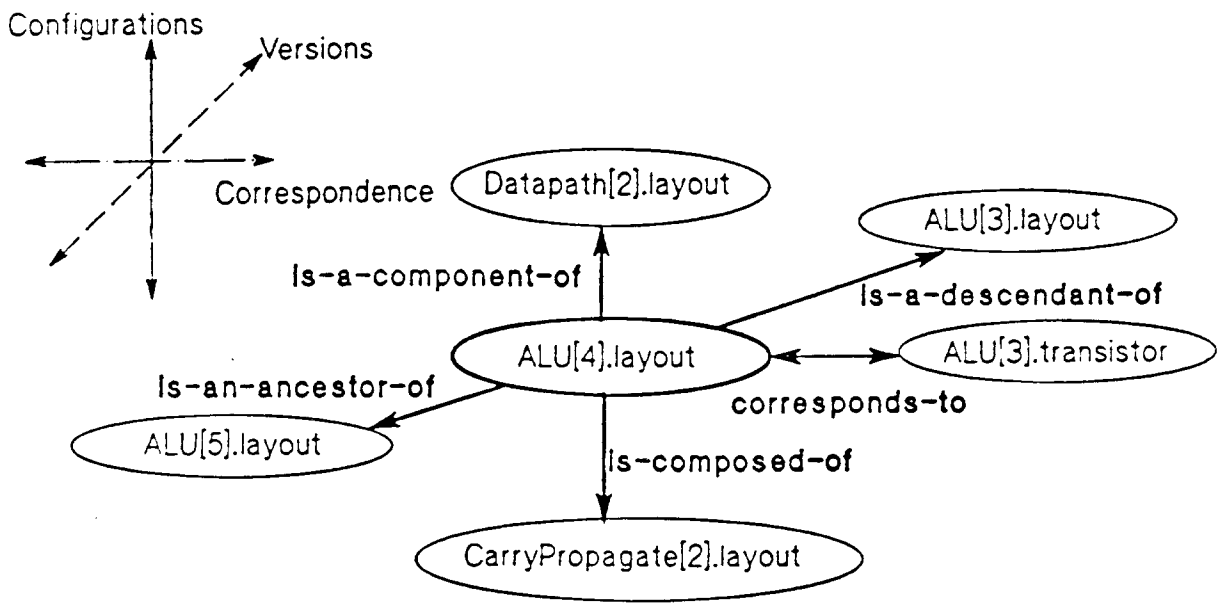

Figure 1.1 -- Version Data Model

Design data is organized as a collection of typed and versioned design objects, interrelated by configuration, version, 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.

reclustering at run-time. We will also discuss how prefetching with 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 are recorded representing approximately 400 hours of design work. In the measurement results, we have observed very high data read to write ratios from the OCT tools environment, implying that dynamic clustering and context-sensitive buffering can be a big pay-off in object-oriented applications.

The rest of the paper is organized as follows. Section 2 discusses how structural relationships and inheritance can be used in various clustering and buffering algorithms. The OCT tools' access pattern, read/write ratio, I/O rate and structure densities are reported in Section 3. They are used to define the workload in the simulation model described in Section 4. The simulation model is constructed in a modeling language called Performance Analyst's Workbench System (PAWS) [PAWS83] to evaluate the performance impact of

different algorithms. The simulation results are discussed in Section 5 through "Two-level factorial" analysis technique. Our conclusions and future directions are given in Section 6.

2. Effective Buffering and Clustering

Engineering design, manufacturing and CASE applications demand high performance object management systems. They utilize the structural relationships and inheritance mechanisms to effectively model their complex environments. Since conventional relational database management systems do not support such modeling primitives explicitly, the storage components cannot take advantage of them at run-time.

Moreover, these complex applications frequently perform materializations of an object hierarchy. Loading a large object hierarchy into memory has become the bottleneck. In this section, we describe how to obtain better bandwidth by exploiting inheritance and structural semantics in buffering and clustering for object-oriented applications.

2.1. 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. When high availability is required by applications such as manufacturing, *static* clustering is not effective. *Dynamic* clustering, on the other hand, is done at run-time when concurrent accesses are permitted. Although clustering on object creation or update may degrade the response time of writers, the simulation results in Section 5 show that such degradation to writers can be offset by a large improvement of readers' response time. *Dynamic* clustering, therefore, becomes very attractive in object-oriented applications where reads dominate writes (Refer to Section 3).

The inputs to the clustering algorithm can be user's hints such as "access by configuration", the interobject access frequencies, and the 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 of its instances.

The clustering algorithm can be sketched as follows (the full algorithm is presented in [CHAN87a]). For each newly created instance, it chooses an initial placement 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 could be affected by the choice of how to implement 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.

A set of parameters are used to control the clustering algorithm:

(a) Candidate page pool. When looking for a candidate page for placement, the clustering algorithm may only use 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. If the I/O limit is infinite, then the algorithm would use the entire database as the candidate page pool. Therefore, the candidate page pool can be

Within_Buffer, With_IO_limit, or **Within_DB.**

(b) 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 it. 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 it.

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. We have identified a greedy algorithm [CHAN87a] that partitions the nodes of the inheritance-dependency graph into two subsets that can fit into a page individually. At the same time, the greedy algorithm tries to minimize the total cost of broken arcs. Because it 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.**

(c) User Hints policy. Many computer-aided design applications make frequent use of configuration relationships, However, most inheritance references are along version history relationships, since a descendant version typically obtains information from its ancestor. If users can make this known to the system, it can cluster objects based on the characteristics of the application. We would like to find out the effectiveness of these user hints. Therefore, the user hints policy can be **User_Hints** or **No_Hints.**

2.2. Smart Buffer Replacement

The most common operations of object-oriented tools are navigation along the structural relationships and simple retrieval of design objects. In general, these tools have pretty static access patterns. Unfortunately, these are ignored by most database systems.

To obtain better response time, the buffer manager must exploit knowledge about the structural and inheritance relationships to determine prefetching and buffer replacement strategies. Response time is improved if appropriate objects can be prefetched before actually being needed, or if related objects can be kept in the buffer pool even if the relationships span disk pages.

The implementation overview of an object-oriented buffer manager is presented here. (Refer to [CHAN87a] for more detailed algorithms) The least sophisticated buffer manager uses a simple LRU buffer replacement policy and attempts no prefetching. A more sophisticated approach uses a priority scheme such that the lowest priority pages are the ones to be replaced 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 (e.g. pages containing its components and its inherited attributes) might be in the buffer pool already. The traditional LRU buffer replacement algorithm could easily choose these pages to be replaced, and thus cause extra I/Os to bring them in later on.

Another way to obtain good response time is to be smart about prefetching. 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. This 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.

A set of parameters are used to control the buffering algorithm:

(a) Buffer Pool Size.

(b) Buffer Replacement Policy. Three policies are examined: **Context-sensitive**, **LRU** and **Random**.

(c) Prefetch Policy. The candidate pages for prefetching can be constrained to either the buffer pool or the entire database. Notice that prefetching within buffer pool does not create any extra logical I/Os. However, it will cause the buffer priority to be adjusted to the requesting applications.

2.3. Experiments to be evaluated

These control parameters are used in the simulation model described in Section 4. In this paper, we only report the results of the following experiments:

(1) Alternative clustering policies with varying page splitting policies.

(2) Alternative buffer replacement policies with varying prefetching policies.

For more detailed results, refer to [CHAN89].

3. Object-Oriented Application Access Pattern

We have instrumented OCT, an object-oriented data manager, interviewed several OCT's users who are VLSI CAD tool developers, and collected the access patterns from real user environments. OCT has more than 20 client applications, and it is being used heavily in the Berkeley CAD community. Although it only provides a subset of object-oriented concepts, we felt that OCT will give us a very good starting point in understanding the access pattern of object-oriented applications. In the following section, we report on our findings.

3.1. OCT Data Manager Overview

OCT is a data manager for VLSI/CAD applications [HARR86]. It supports a set of primitive object types which are used frequently by VLSI CAD tools and arbitrary attachments among these objects. For instance, in Figure 3.1, a net is attached to a facet which is a basic design unit, four terminals are attached to the net, and three paths are attached to various terminals. Every attachment creates a link between two objects. These links are bidirectional and provide basic composition hierarchy information. However, it is users' responsibility to maintain the legal attachment among objects and the system does not provide any structure validation. Further, OCT support no explicit inheritance mechanisms.

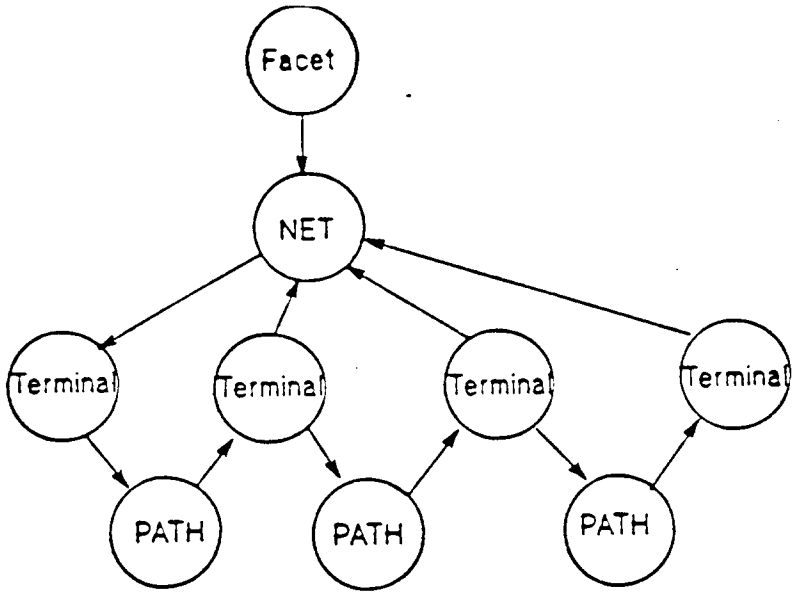

Figure 3.1 -- MOSAICO Access Pattern

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.

3.2. Information Collected

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

(1) Tool Identifier such as "SPARCS" or "VEM". This helps us understand the run-time behavior of an individual application. It also allows us to relate tools' functionalities with their overall run-time behavior.

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

(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 and understand the correlation between session time and applications' functionalities.

(4) Fan-out of upward and downward structural access. Reading a composite object may only trigger a subset of its component objects being retrieved. Similarly, reading a component object may cause several of its composite objects being returned. For instance, a netlist simulator would only navigate along the <cell>, <net>, and path, and the remaining component objects attached to <cell> are totally untouched. That is, not all of the component objects are read when an application is reading a composite object at run-time.

3.3. R/W ratio

For every tool invocation, we collect the number of structure read, structure write, simple read, and simple write. All these read and write operations 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 3.2 shows the read/write ratio of nine OCT tools. VEM, a graphical editor not included in Figure 3.2, has the highest read/write ratio of 6000. The rest of the OCT tools' read/write ratios vary from 0.52 to 170. 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 3.2 shows that the read/write ratio within one run varies from 0.52 to 170. This is quite unusual compared to traditional debit/credit applications where the read/write ratio is quite stable throughout an application. Due to this dynamics of applications' read/write ratio, the clustering algorithm must be adaptive to achieve adequate response time at different phases of an application. The impact of varying read/write ratio on effective clustering is reported in [CHAN89].

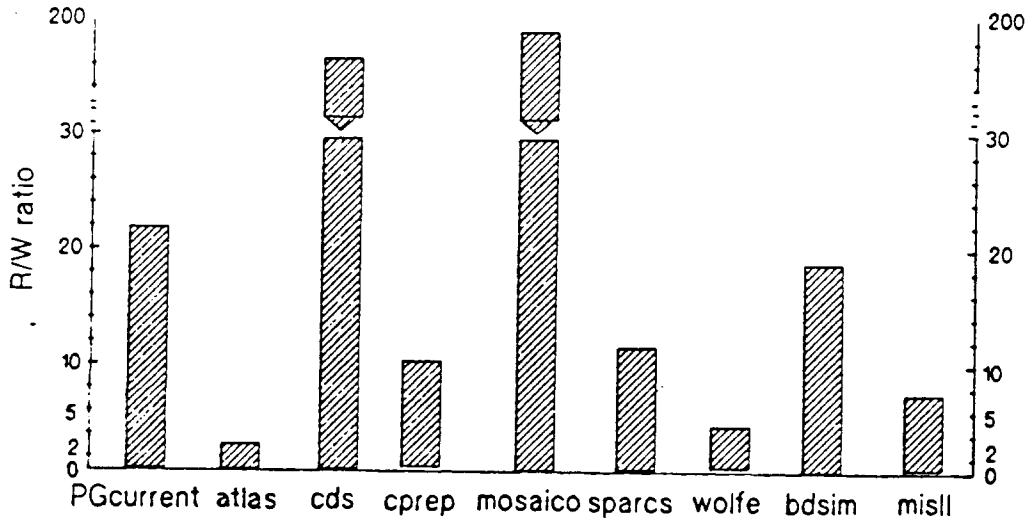

Figure 3.2 -- OCT Tools' Read-Write Ratio

Ten OCT tools are measured. The highest read/write ratio is 6000 belonging to VEM, a graphical editor. The rest of OCT tools' read/write ratio 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 those seen by the buffer manager.

Figure 3.3 shows the I/O rate of various OCT tools. They are calculated by counting all logical read and write operations and are then divided by the session time. Except VEM, the rest of the tools are run in batch mode and the session time does not include think time. The I/O rates give us which tool is the most I/O intensive.

3.4. Structure Density

Figure 3.4 show the downward access structure density of ten OCT tools. Although both upward and downward accesses are measured at run-time, we observed that most of the upward accesses have only one object returned. Therefore, we only report the fan-out of downward access here.

Except Wolfe, most of the OCT tools' downward access are dominated by 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.

3.5. Access Pattern

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

(1) Object-oriented applications perform more navigation than ad-hoc query during run-time. For instance, in Figure 3.1, the macro cell router MOSAICO navigates along the configuration to find all the paths used by a certain net. No ad-hoc query is needed in MOSAICO.

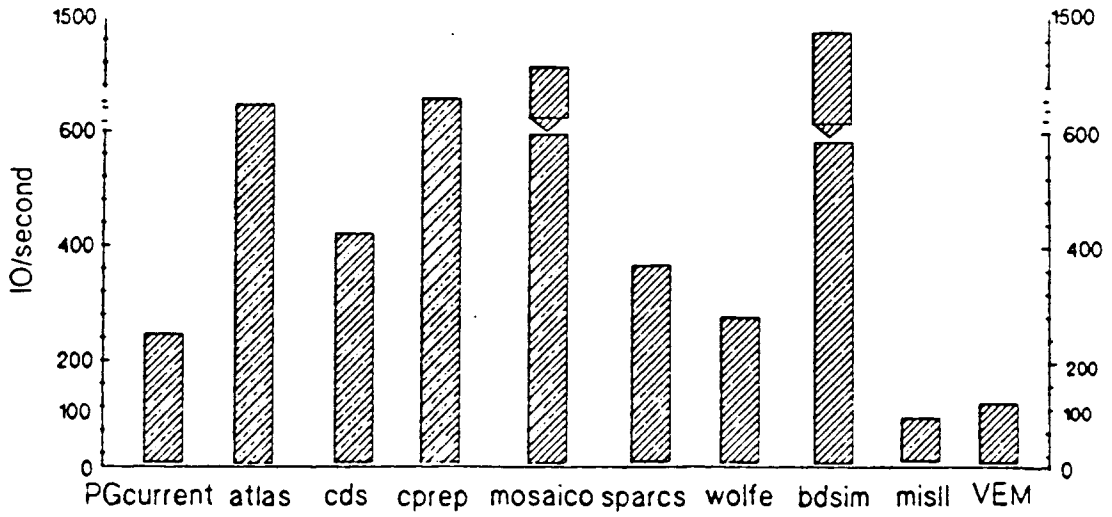

Figure 3.3 -- OCT Tools' Object I/O Rate

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

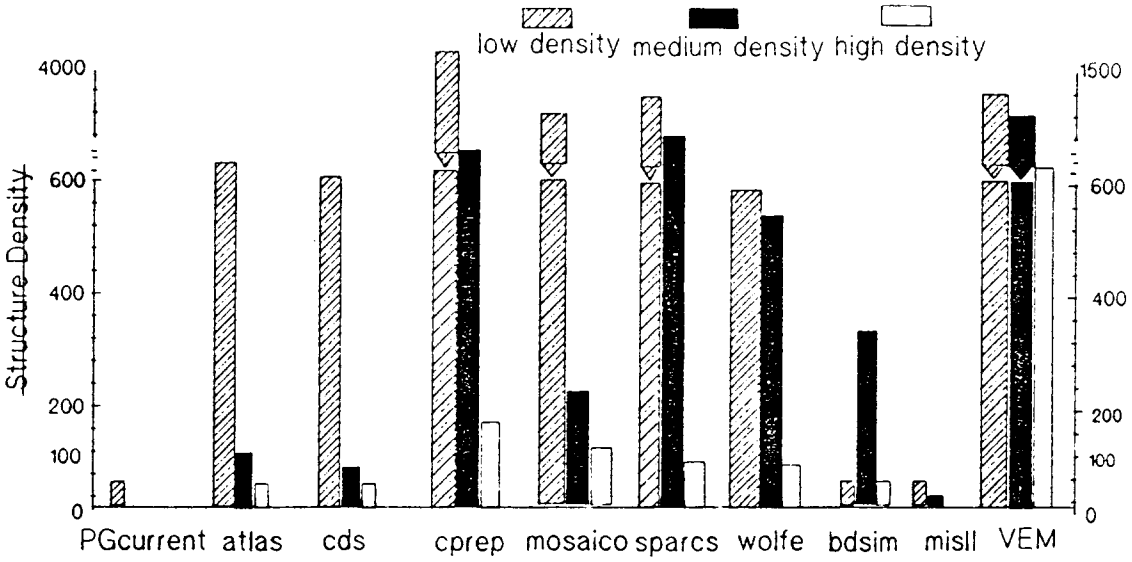

Figure 3.4 -- OCT Tool Structure Density Distribution

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

(2) Certain access patterns can be eliminated if the underlying system supports referential integrity. 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 is to assure that SPARCS will not run into a loop at run-time. However, it introduces a tremendous number of unnecessary I/Os.

(3) Most of the access patterns are predictable. Certain objects, such as the net instance in Figure 3.1, are accessed several times during navigation. If this access pattern is known, the buffer manager could increase its containing page priority to increase the buffer hit ratio.

4. Simulation Model

A simulation model allows us to evaluate the performance impact of different clustering and buffering algorithms on different applications' characteristics. It also helps us rapidly answer certain what-if questions. For instance, what is the relationship between choice of clustering algorithm and frequency parameters such as inheritance density and read/write ratio.

We used the Performance Analysis's Workbench System (PAWS) to construct the simulation because it supports a number of high level primitives, e.g., various queueing disciplines, and a set of detailed statistics output. It also allows us to refine and enhance the model easily. In the following section, we discuss the workload and our model in term of PAWS primitives. We summarize all the modeling parameters in section 4.2.

4.1. The Engineering DB Model

We represent our model of engineering database interaction in the PAWS language. The PAWS language allows the user to simply declare the characteristics of the system being modeled rather than coding detailed simulation algorithms.

The simulation model consists of several interacting model blocks. Transactions representing user requests flow among these model blocks carrying information about the work units. As shown in Figure 4.1, the major blocks are:

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

** **Workload Definition:** representing workload characteristics

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

** **Buffer Manager:** the buffer management module

** **Cluster Manager:** defining and imposing various clustering algorithms

** **CPU:** representing the processor

Figure 4.1 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 needs to search through the buffer pool and issues a physical I/O if needed. During this buffer searching phase, it is possible that some dirty pages are flushed out as well as some transaction log records. Therefore, a logical I/O, generated in the workload definition node, can be translated into zero to 3 physical I/Os in the worst

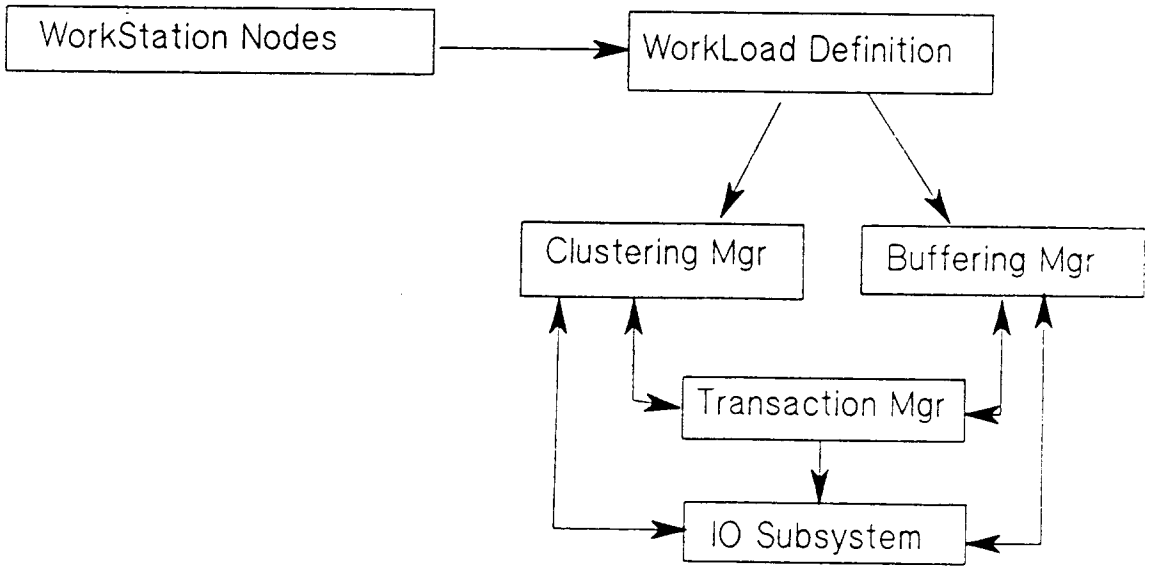

Figure 4.1 -- Simulation Model Overview

case. That is, one I/O to flush the dirty page, one I/O for transaction logging, and one I/O to bring in the data.

A few unique things about 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, are maintained explicitly. This allows us to construct a 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.

A more detailed model description is shown in Figure 4.2.

The *workload definition* node is intended to capture the workload characteristics. The definition of this node is specific to a generic workload class. By generic workload class, we mean a wide range of application based on a single software package. For instance, the

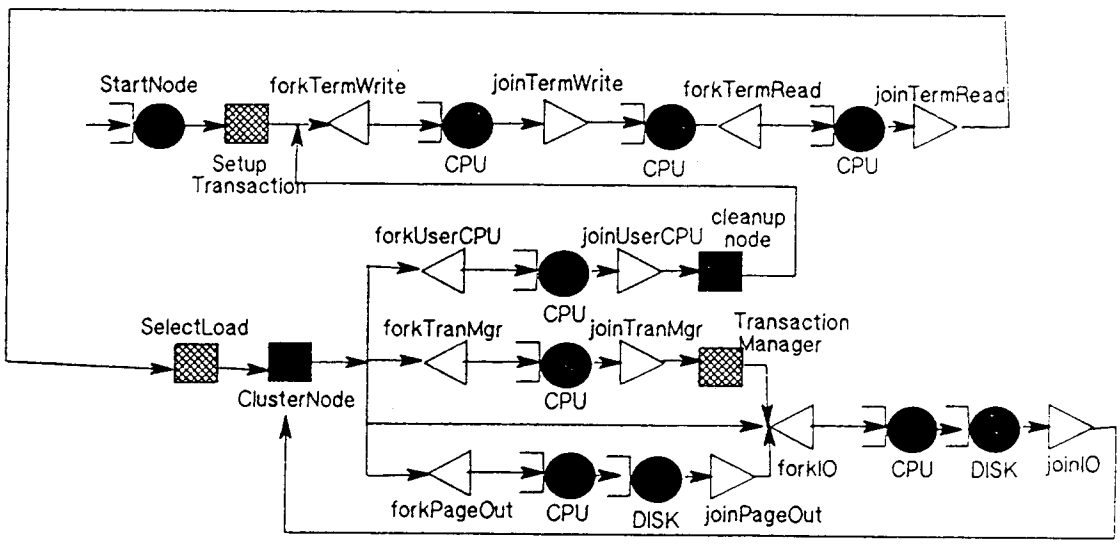

Figure 4.2 -- Simulation Model Detail

family of CAD tools using OCT belong to the same generic OCT workload.

Each generic workload class is defined by a set of parameterized primitives that are most natural to describe the workload. Various applications in the class are specified by assigning values to parameters. For instance, a generic OCT workload is defined by the distribution of OCT procedure calls. Each OCT procedure call can have an average path length, I/O content, and lock request behavior.

The fundamental unit of recovery and concurrency control is the object and composite object. An object can be read by specifying an unique name or navigating along some structural relationships from other objects. We decided that every object read and write operation is a transaction. Furthermore, a user session is composed of 5 to 20 transactions with various read/write ratios. If the target object for a read operation is composite, several logical I/Os may be triggered to retrieve the complete object. On the other hand, the write operation would only create or update a single object.

Generally speaking, engineering design applications' procedure calls can be categorized into seven query types: (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. These seven different query types are assigned to transactions in the workload definition phase.

In our simulation model, the checkin and checkout operations are modeled by these seven query types. For instance, a checkout operation may consist of several component object retrievals and one corresponding object retrieval. Similarly, a checkin operation invokes some object insertions and updating.

4.2. Parameters

Table 4.1 shows all the parameters modeled by the simulation model and their operating levels. They are divided into static parameters, fixed for all the simulation runs, and control parameters which have various operating levels.

Label	Static Parameters	Default Value
A	**Database Size**	500 Mbytes
B	**Page Size**	4 Kbytes
C	**Number of Users**	10
D	**Number of Disks**	10
E	**Think Time**	4 seconds
	Control Parameters	Operating Levels
F	**Structure Density**	low-3,med-5,high-10
G	**Read-write Ratio**	5,10,100
H	**Clustering Policy**	No_Cluster,Cluster_within_Buffer 2_IO_limit,10_IO_limit, No_limit
I	**Page Splitting Policy**	No, Greedy, Optimal
J	**User Hint Policy**	No_hint, User_hint
K	**Buffer Replacement Policy**	LRU, Context-sensitive,Random
L	**Buffer Pool Size**	100,1000,10000
M	**Prefetch Policy**	No_prefetch, Prefetch_within_buffer_pool Prefetch_within_Database

Table 4.1: Simulation Parameters

The operating levels and default values of these parameters are partially influenced by section 3 where the structure density and read/write ratio are observed. The number of users 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 use 10 disks in the server and assume that page size is 4 Kbytes. The database size is 500 Mbytes and the average think time for each user is 4 seconds, which we believe are representative of interactive computer-aided design environments.

Eight control parameters are studied to understand their impacts on the overall system response time. Out of these seven control parameters, (F) and (G) determine the workload characteristics, (H), (I) and (J) control the clustering policy whereas (K), (L), and (M) define the buffering strategy. The operating levels of Structure Density is chosen based on the actual observation from OCT tools access patterns. Low-3 means every structural retrieval returns less than or equal to three component or composite objects. Similarly, med-5 means more than 3 but less than 10 objects are returned through structural retrieval. High-10 means more or equal to 10 objects are returned.

5. Simulation Results

The control parameters, listed in Table 4.1, may interfere with each other. For instance, higher structure density means more I/Os for composite object retrieval and more candidate pages to consider during object placement phase. With higher read/write ratio, the extra I/Os caused by writers during the clustering phase may be amortized by the readers and improves overall system response time.

In Section 5.1, we discuss the run-time clustering effects using different clustering policies and various buffering strategies are studied in Section 5.2.

5.1. Run-Time Clustering Effect

The operating level of buffering control parameters used in this clustering effects study are: No prefetch, 1000 buffers, and LRU buffer replacement policy. We first discuss the clustering effects when selected candidate page for object placement never overflow. That is, when the preferred candidate page is full, the storage manager chooses the next best candidate page which has enough space. Then, we study how various page overflow handling mechanisms affect clustering.

5.1.1. No Page Overflow

Figure 5.1 shows how various clustering policies affect system response time under different workloads. Some key observations are:

(a) 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

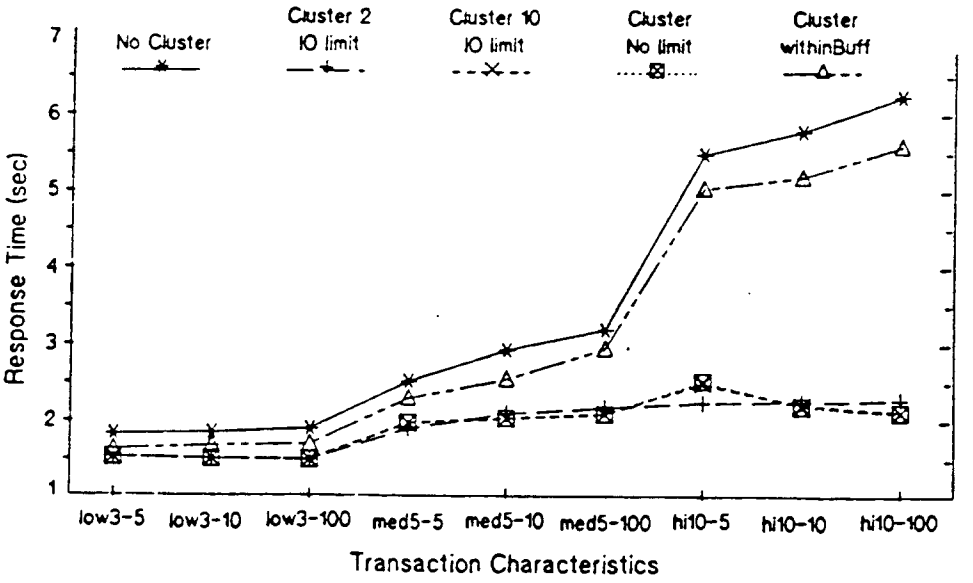

Figure 5.1 -- Clustering Effects Analysis

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 low structure density which is less or equal to 3, and its read/write ratio is 5. Similarly, "hi10-100" means the transaction has high structure density which is greater than or equal to 10 and its read/write ratio is 100.

200%.

(b) 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 or comparable to *clustering without I/O limitation.*

(c) *Clustering within buffer pool* performs reasonably well when the structure density is low. However, its performance 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, its performance is close to *No_Clustering.* However, its performance improves when a better buffering policy is used (Refer to Section 5.2)

However, where is the break-even point of the read/write ratio when *No_Clustering* has similar response time as any clustering mechanisms? We compare *No_Clustering* with *clustering without any I/O limitation* and the results are summaried in Table 5.1. Different structure densities have different break-even points, due to the amount of logical I/Os caused by writers during the clustering phase.

Structure Density	Read-write ratio
low-3	3.0
med-5	3.6
high-10	4.3

Table 5.1: Read-write ratio break-even points

Figure 5.2, 5.3 and 5.4 show the structure density effect on response time when the read/write ratio is fixed. 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 and the possibility of buffer hit for these logical I/Os decreases, the increasing physical I/Os result in higher response time. On the other hand, the response time rises slowly from low structure density to high structure density when any clustering mechanism except *clustering within buffer pool* is used. The low variation of response time over different structure densities is very critical when application access patterns are not fully predictable.

Clustering within buffer pool cannot perform well if the buffer pool hit ratio is low during 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. If the buffer manager understands the relationships among objects, it may increase the priority of these pages and increases the hit ratio. We will discuss the various buffering policies in Section 5.2.

When the read/write ratio is low such as 5, as shown in Figure 5.2, *clustering with a 2 I/Os limitation* provides the best response time in all structure densities. 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 low structure density implies fewer candidate pages to choose and fewer logical I/Os are needed during 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

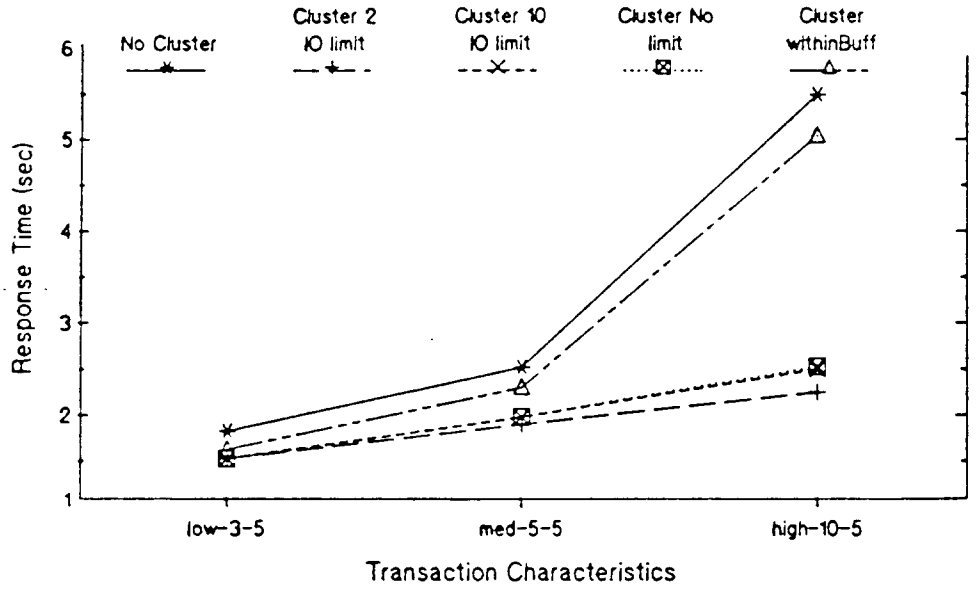

Figure 5.2 -- Clustering Effect Under R/W ratio 5

The buffering control parameters are fixed to 1) No prefetch, 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 low structure density which is less or equal to 3, and its read-write ratio is 5. Similarly, "high-10-5" means the transaction has high structure density which is greater than or equal to 10 and its read-write ratio is 5.

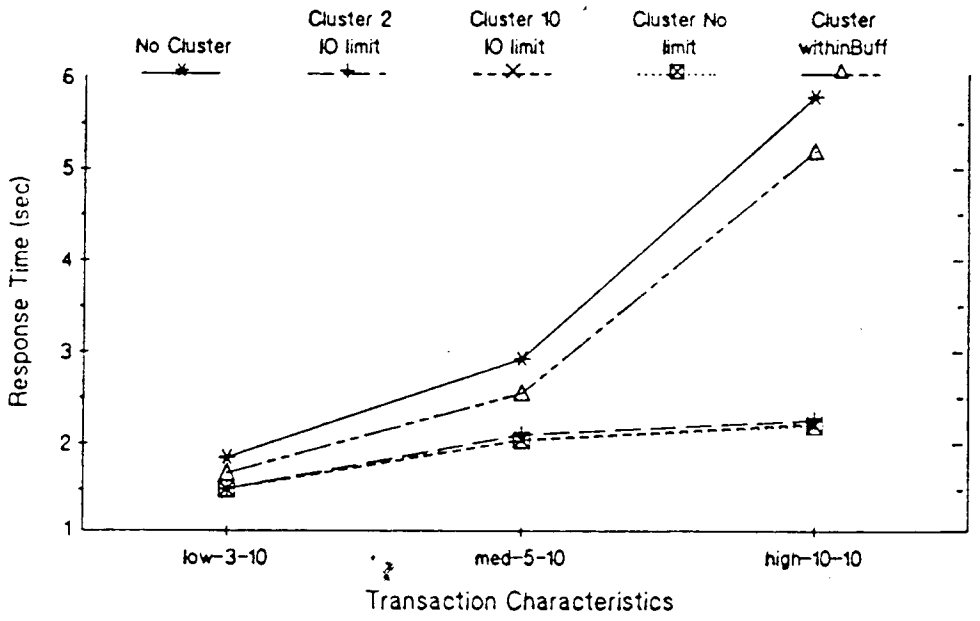

Figure 5.3 -- Clustering Effect Under R/W ratio 10

The X axis represents various transaction characteristics. "low-3-10" means the transaction has low structure density which is less or equal to 3, and its read-write ratio is 10. Similarly, "high-10-10" means the transaction has high structure density which is greater than or equal to 10 and its read-write ratio is 10.

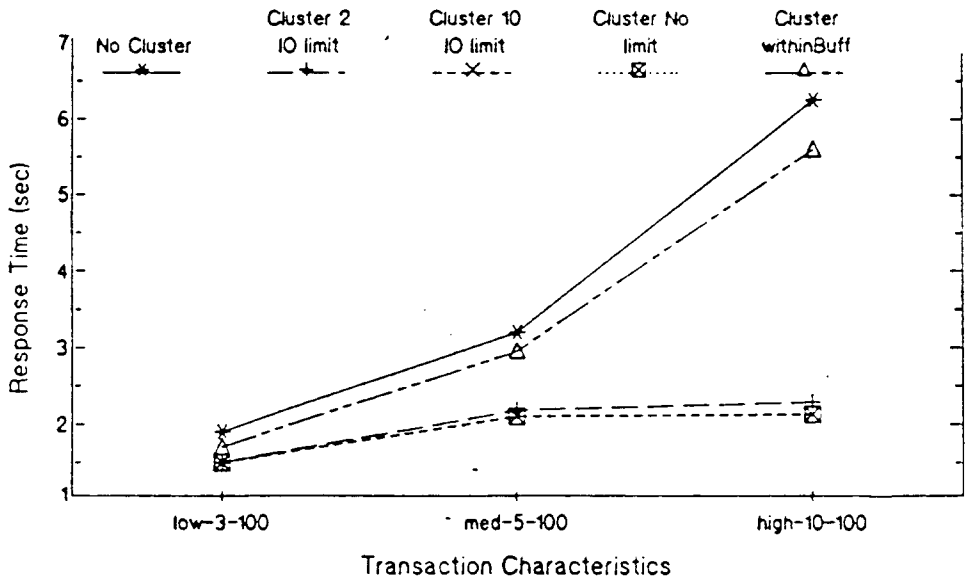

Figure 5.4 -- Clustering Effect Under R/W ratio 100

The X axis represents various transaction characteristics. "low-3-100" means the transaction has low structure density which is less or equal to 3, and its read/write ratio is 100. Similarly, "high-10-100" means the transaction has high structure density which is greater than or equal to 10 and its read/write ratio is 100.

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 5.3 shows that clustering with a limitation of 10 I/Os has the same response time as clustering with no I/O limitation when structure density is medium. 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 5.4. If the clustering mechanism can be selected based on the read/write ratio at run-time, we can get the best response time of both, either setting small I/O limitation or no I/O limitation at all.

Clustering not only helps the retrieval but also reduces the number of I/Os in the transaction logging phase. This is shown in Figure 5.5 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 only needs to flush the original page once. Since related objects are clustered on the same page, the probability of having multiple updates on the same page within a transaction increases, thus reducing the number of physical logging I/Os.

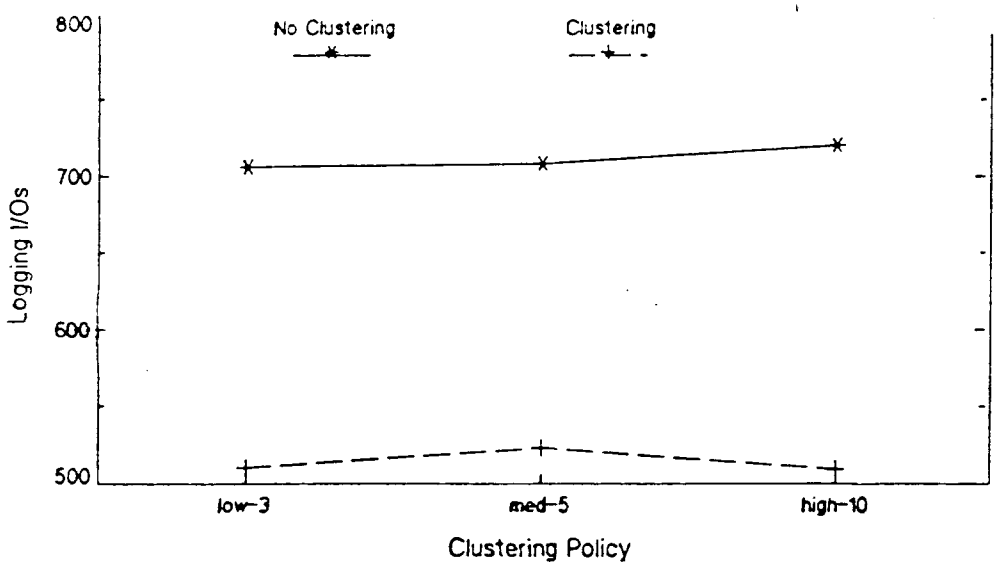

Figure 5.5 -- Clustering Effect on Transaction I/Os

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.

One interesting thing we learned from collecting OCT tools' access patterns is that 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 5.6 shows that any clustering mechanism performs 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 5.7, *clustering without I/O limitation* performs the best when applications' read/write ratio is greater than 10. Notice that the response time of *clustering without I/O limitation* case is almost no change for all read/write ratios. Such a stable response time may be required by some real-time applications.

Figure 5.8 shows the clustering effect on response time when the structure density is high. Notice that the difference between the *clustering within buffer* case and 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.

5.1.2. With Page Overflow

When the chosen candidate page overflows with the newly inserted object, the system needs to either split the target page or picks the next best candidate page. To split the

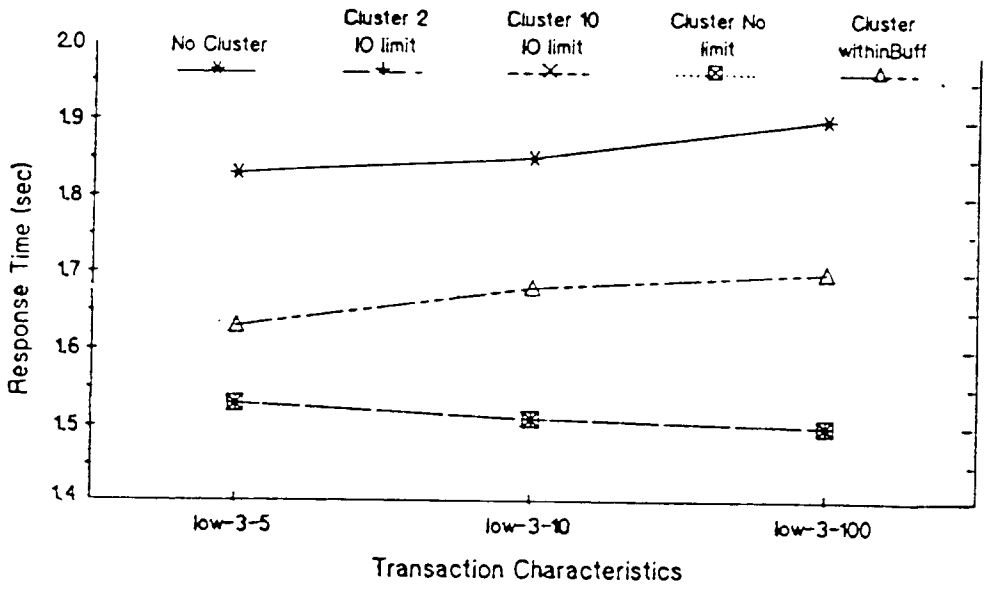

Figure 5.6 -- Clustering Effect Under Low Structure Density

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

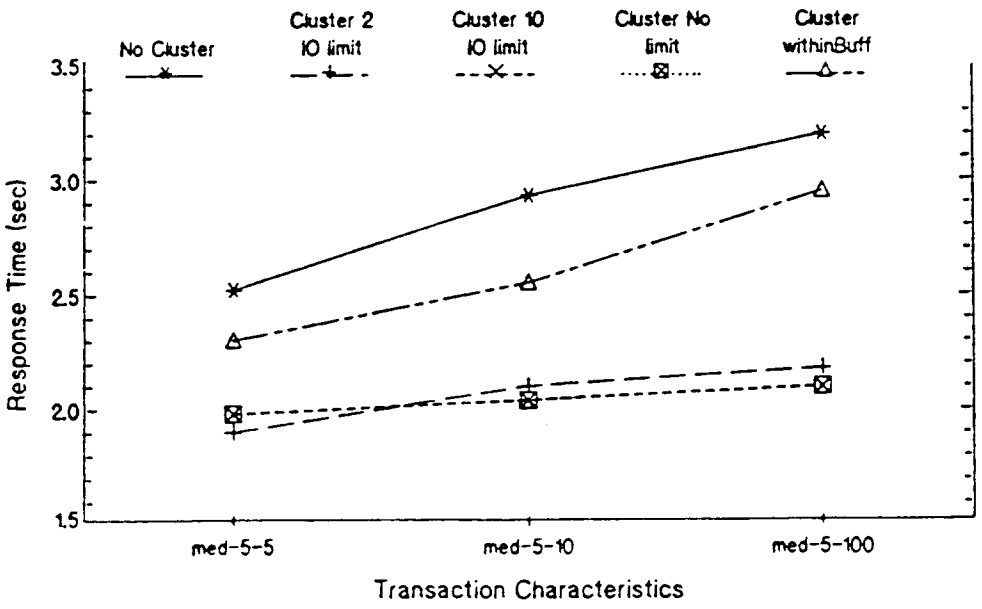

Figure 5.7 -- Clustering Effect Under Medium Structure Density

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

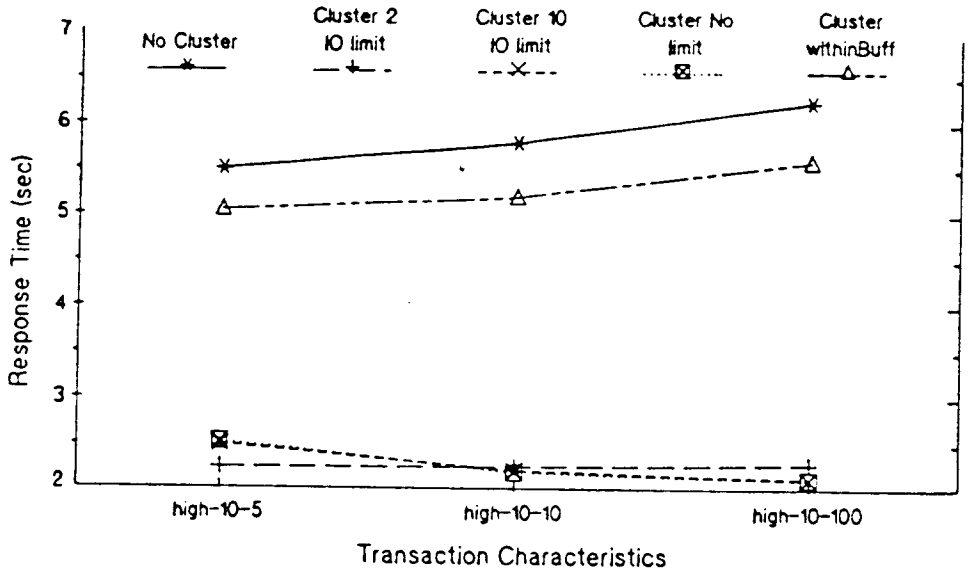

Figure 5.8 -- Clustering Effect Under High Structure Density

The X axis represents various transaction characteristics. "high-10-5" means the transaction has high structure density which is greater than equal to 10, and its read/write ratio is 5.

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. Compared to no splitting, page splitting requires one extra I/O to flush the newly allocated page and one extra log record that may trigger an I/O. Moreover, the page splitting algorithm needs extra CPU time and the new page allocation may cause buffer pool contention.

As mentioned in Section 3, we have evaluated two different page splitting algorithms with the simulation model: *Linear Split* and *NP Split*. Figure 5.9 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 splitting 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. This is due to 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. Since NP Split algorithm always finds the minimum access cost partition of objects, Figure 5.10 shows the total cost difference from the Linear Split under different transaction characteristics.

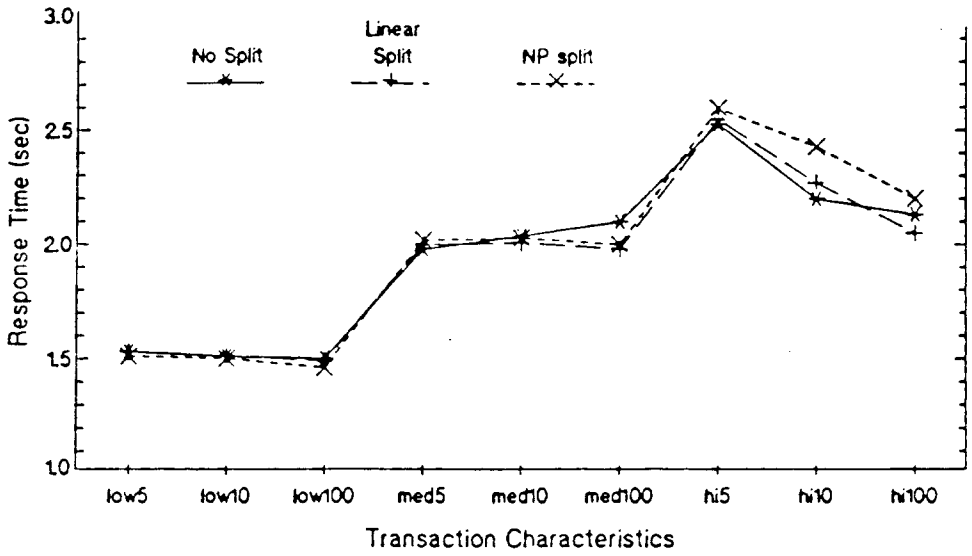

Figure 5.9 -- Page Splitting Effects Analysis

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. Solid line represents the response time for no page splitting case. Dotted line with plus sign is the response time when linear page splitting mechanism is applied. Dotted line with cross sign is the NP split case.

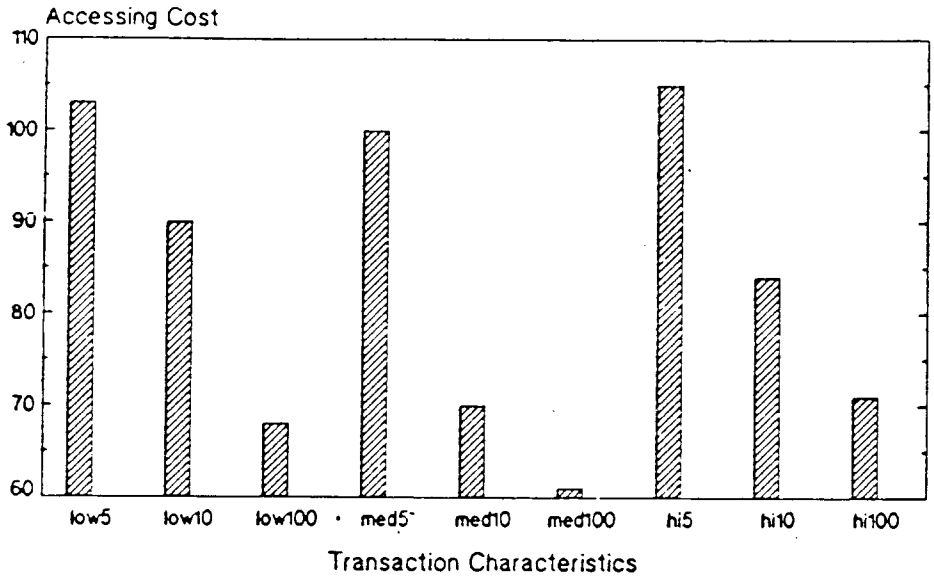

Figure 5.10 -- Cost Difference between NP Split and Linear Split

The X axis represents various transaction characteristics. "low10" means the transaction has low structure density which is less than or equal to 3 and its read/write ratio is 10. Similarly, "hi100" means high structure density and the read/write ratio is 100. Accessing cost difference, represented by the Y axis, is measured in term of *logical I/O*.

5.2. Impact of Smart Buffering

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. No user hints are provided and the number of buffers is 1000. The effect of buffer pool size on various buffering strategies are discussed in [CHAN89].

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.

Figure 5.11 shows how various buffering policies affect system response time under different workloads. Some key observations are:

(a) Context-sensitive buffer replacement policy always improves 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.

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

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

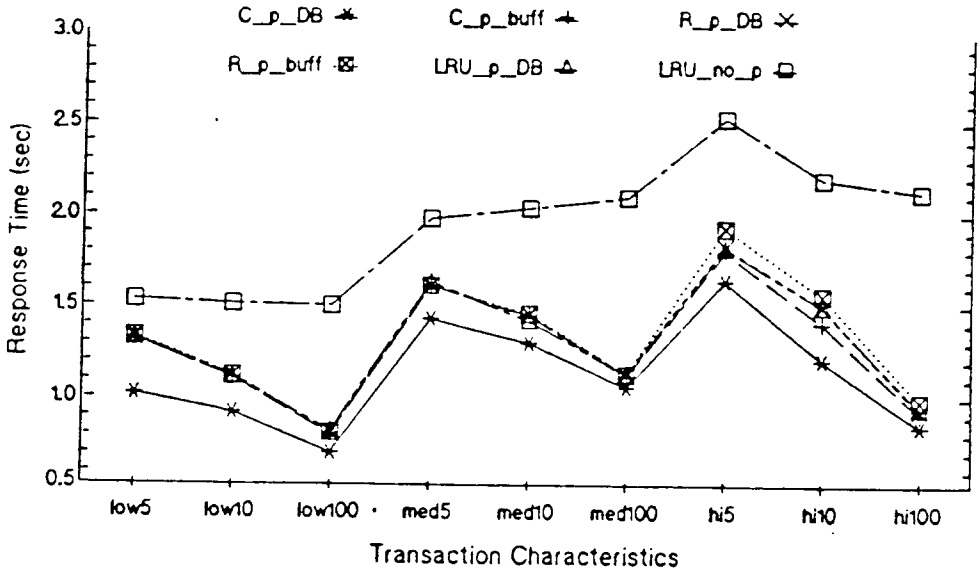

Figure 5.11 -- Buffering Effects Analysis

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 high structure density and the read/write ratio is 100.

Figure 5.12, 5.13, and 5.14 show the prefetching effect on response time when the buffer replacement algorithm is fixed. In all cases, *prefetch within database* performs the 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 to applications in memory, 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 5.12, *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 future needs of data contained in these buffers. Since 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 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 better response time.

When the buffer replacement algorithm is not context-sensitive, as shown in Figure 5.13 and 5.14, 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*, the overall response time is similar.

For more detailed analysis, refer to [CHAN89].

6. Effect Analysis and Conclusions

Figure 6.1 shows the overall effect analysis for the eight control parameters specified in Table 4.1. There are 8! (i.e. 40,320) possible combinations, and for each one, we calculated how much, on the average, the response time has changed when the combined parameters go from low operating level to high operating level. Although the response time change can be positive or negative, only the absolute value is useful in this effect analysis. Two interesting observations are: the structure density and buffering policy most influence system response time, while different page splitting algorithms have little influence on response time.

Two control parameters, say A and B, are said to "interact" if the effect of A is *different* at different operating levels of B. They can be represented in terms of X-Y diagram with X axis represents the different operating levels and Y axis represents the effect (i.e. response time). If two lines are in parallel, there is no interaction between the corresponding control parameters. For intersected lines, we know there is a strong

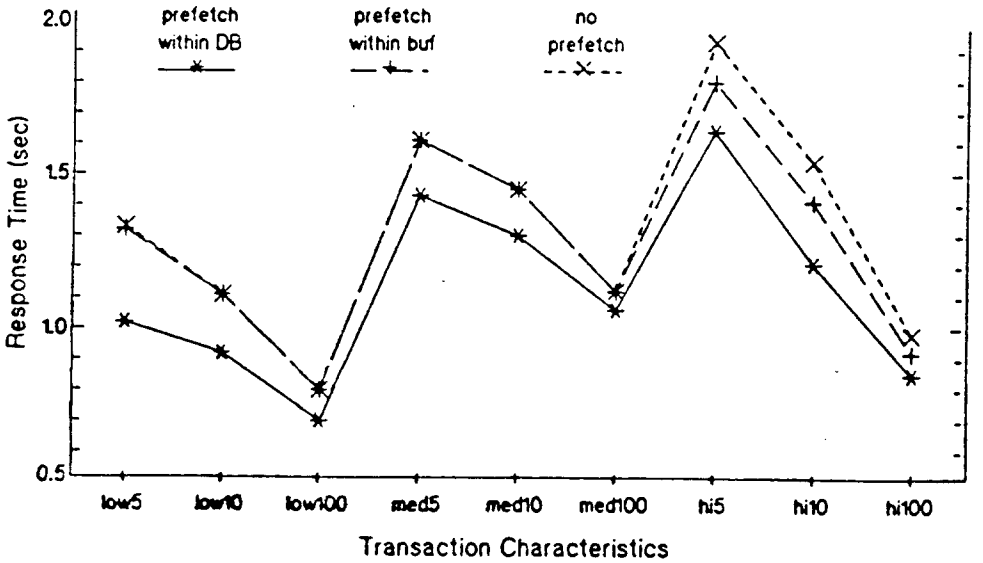

Figure 5.12 -- Prefetching Effect under Context-sensitive Buffer Replacement Policy

Three prefetching policies are evaluated here under a Context-sensitive buffer replacement algorithm with various transaction characteristics. "hi100" means the transaction has high structure density and the read/write ratio is 100.

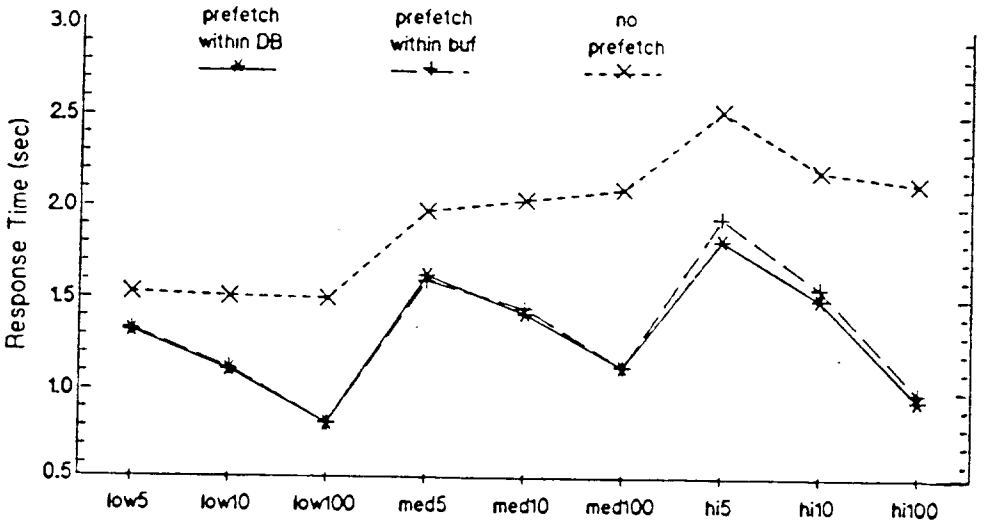

Figure 5.13 -- Prefetching Effect under LRU Buffer Replacement Policy

Three prefetching policies are evaluated here under a LRU buffer replacement algorithm with various transaction characteristics. "hi100" means the transaction has high structure density and the read-write ratio is 100.

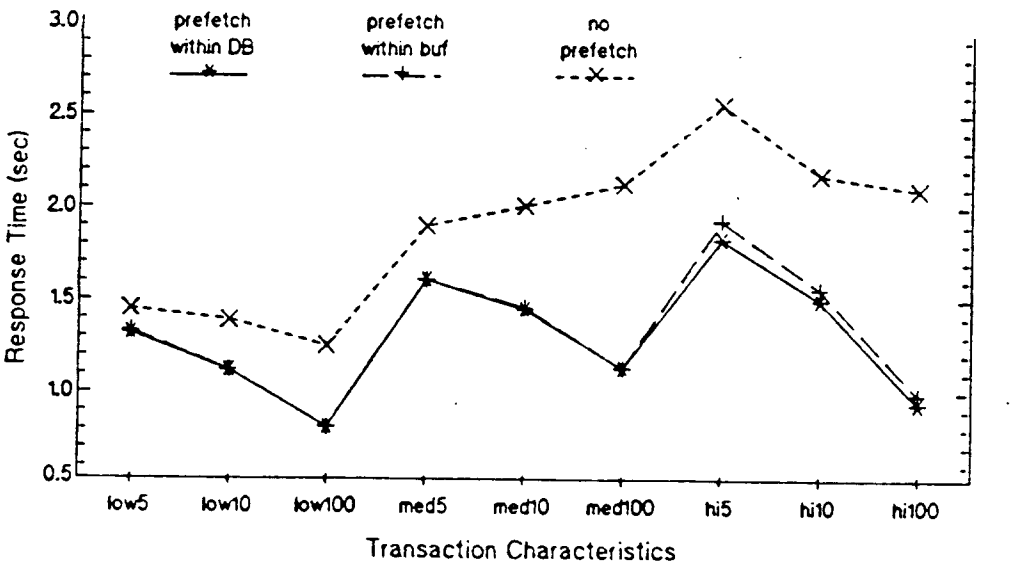

Figure 5.14 -- Prefetching Effect under Random Buffer Replacement Policy

Three prefetching policies are evaluated here under a Random buffer replacement algorithm with various transaction characteristics. "hi100" means the transaction has high structure density and the read-write ratio is 100.

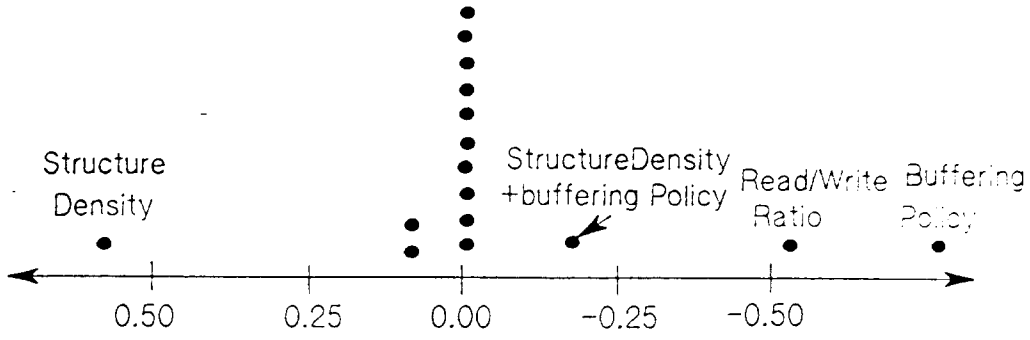

Figure 6.1 -- Overall Effect Analysis

Larger absolute effect value means major effect on response time. When the effect value is zero, it implies that the corresponding control parameter(s) has minimum effect on the overall response time. Each blob represents the effects of a parameter or combined parameters on response time. For instance, "Structure density+buffering policy" represents a combined effect whereas "Read/write Ratio" is a single parameter effect. Most of the parameter or combined parameters have small response time change and are represented by a line of blobs centered in the figure.

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.

Figure 6.2 shows a subset of interaction analysis graphs. The key observations are:

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

(b) 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, and page splitting policy and clustering policy.

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

In this study, we have exploited inheritance and structural relationships for improved buffering and clustering. We have proposed a run-time reclustering algorithm which under certain conditions can improve system response time by a factor of 200%. We have also shown that it is effective to limit 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

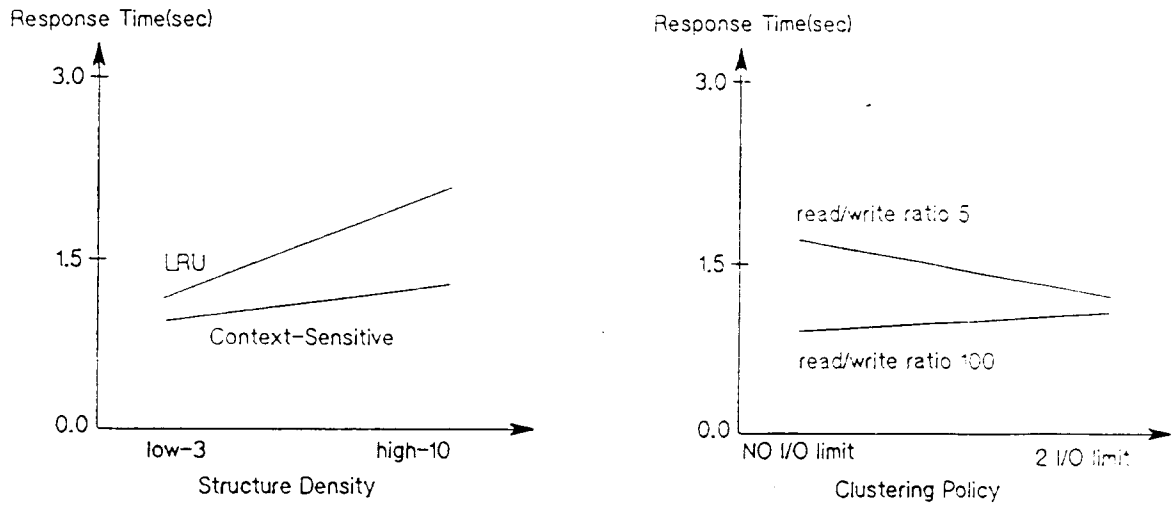

Figure 6.2 -- Interaction Analysis Graph

Both of these interaction analysis graphs represent minor interaction case. When the two lines are in parallel, it implies no interaction. If two lines intersect, it means major interaction.

on response time under various buffer replacement policies and transaction characteristics. Context-sensitive buffer replacement policy with *prefetch within database* performs the best whereas *LRU with no prefetching* is the worst.

We have also studied the effectiveness of *user hints* and how varying read-write ratios within a transaction affect the response time under various clustering and buffering algorithms [CHAN89]. However, the dynamics of access pattern of object-oriented applications are still not well understood. Furthermore, the physical representation of structural relationships and inheritance links among objects needs to be addressed.

Based on the simulation results shown in this study, we strongly recommend that future object-oriented DBMSs to (a) model structural relationships and inheritance links as first class objects, and (b) use the proposed dynamic clustering algorithm when the read-write ratio is high.

7. Reference

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

[BATO85b] Batory, D., W. Kim, "Supporting Versions of VLSI CAD Objects," M.C.C. Technical Report, Austin, TX, (1985).

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

[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.

[CHAN89] Chang, E.E., "Effective Buffering and Clustering in Object-oriented Database Management Systems", Ph.D. Dissertation, May 1989.

[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.

[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).

[KIM87] Kim, Won, Jay Banerjee, Hong-Tai Chou, Jorge F. Garza, Darrel Woelk, "Composite Object Support in an Object-Oriented Database System," Proc. OOPSLA'87 Conf., Orlendo, FL, (Oct. 1987).

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

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

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