EFFECTIVE CLUSTERING OF COMPLEX OBJECTS IN OBJECT-ORIENTED DATABASES

Jia-bing R. Cheng and A. R. Hurson

Department of Electrical and Computer Engineering
The Pennsylvania State University
University Park, PA 16802

ABSTRACT

Clustering is an effective mechanism for retrieving complex objects. Many object-oriented database management systems have suggested variant clustering schemes to improve their performance. Two issues may compromise the effectiveness of a clustered structure, i.e., object updates and multiple relationships. Updates may destroy the initially clustered structure, and in a multiple relationship environment, clustering objects based on one relationship may sacrifice others. This paper investigates the updating effects and suggests a dynamic reclustering scheme to reorganize related objects on the disk. A cost model is introduced to estimate the benefit and overhead of reclustering. Reorganizations are performed only when the overhead can be justified. For environments in which multiple relationships among objects exist, the paper proposes a leveled clustering scheme to order related objects into a clustering sequence. Our simulation results show that the leveled clustering scheme has a better access time compared with a single-level clustering scheme.

1. INTRODUCTION

Database management systems (DBMSs) based on the relational model have dominated the market through the 1980s. This is mainly due to the simplicity and the strong mathematical foundation upon which the relational model is based. However, some of today's data modeling requirements do not fit well into the relational framework. Examples are the databases used for Computer Aided Design, Office Information System, and Artificial Intelligence, in which the representation of complex data entity as well as sophisticated operations are required. Using a relational DBMS in such an environment usually leads to awkward decomposition and hence poor performance. The object-oriented data model allows a natural mapping between conceptual objects and the underlying data model, and hence better performance.

Two important features of the object-oriented DBMSs are the notions of complex objects and object identifiers. A complex object is an aggregation of heterogeneous objects, and object identifiers provide a means for explicit representation of semantic links among objects. This allows navigation through semantic links and retrieval of complex objects. Naturally, for an object-oriented system, it is important to traverse the corresponding graph structure efficiently. The clustering of objects based on their semantic links becomes a powerful tool to improve the performance. This paper investigates problems associated with existing clustering schemes, and proposes new schemes which overcome the deficiencies of the existing ones.

One can identify several common characteristics in existing clustering schemes. Firstly, they are mainly static, i.e., objects are clustered when they are created and are not reclustered afterward. Secondly, many existing clustering schemes use disk pages as the clustering units; the goal is to minimize the number of pages accessed for complex object retrieval. These schemes assumed that each page access takes one disk I/O, and the total access time was measured by multiplying the number of accessed pages by the average access time per page. Hence, they did not consider the actual performance based on the relative dispositions of accessed pages, which may result in non-uniform access time for individual pages. Thirdly, existing schemes were designed for clustering objects based on a single relationship; multiple relationships among objects were not considered. As a result, two potential problems can be recognized: (*i*) Object updates may destroy the initially clustered structure. (*ii*) In environments with multiple relationships among objects, different applications may require different access patterns. Performance will be compromised if a single-relationship clustering scheme is adapted.

This paper introduces two schemes to resolve the aforementioned problems. In our schemes, the cluster-

ing units are chunks (a number of contiguous pages) instead of individual disk pages. For the update problem, we propose a dynamic clustering scheme which reorganizes related objects when the estimated benefit after reorganization is greater than the reorganization overhead. For the problem of multiple relationships, we propose a leveled clustering scheme which clusters objects based on several different relationships in order to meet the requirements of different access patterns.

The remainder of this paper is organized as follows. Section 2 reviews the related issues in object-oriented DBMSs and points out the problems of existing clustering schemes. Section 3 describes the updating effects and introduces a cost model to determine the reorganization points of a dynamic reclustering scheme. Section 4 proposes a leveled clustering scheme and analyzes the effectiveness of this scheme using our simulation results. Section 5 concludes the paper and gives future directions of this research.

2. BACKGROUND

2.1 Relational DBMS

A database management system (DBMS) provides users with an abstract view of data rather than the hardware-level details. Through the 1980s, the relational model has become the trend for commercial DBMSs. This is mainly due to its conceptual simplicity and the strong mathematical foundation it is based on [Date86]. The simplicity and strong mathematical foundation of the relational model, along with sophisticated indexing and query optimization techniques developed in the past have provided an environment which serves the business-oriented database applications—containing a large amount of relatively simple and fixed-formatted data—very well.

Despite these advantages, the relational model does not satisfy the requirements of applications such as Computer Aided Design (CAD), Office Information System (OIS), and Artificial Intelligence (AI). These applications require the modeling of complex data entities which cannot be mapped easily to the relational tables. For example, a VLSI design database requires the support of composite objects as well as different versions of the same object. A multimedia database may contain variable-length text, graphics, images, audio and video data. Finally, a knowledge base system requires a supporting model capable of representing data semantics. In addition, these applications contain data elements of different types and formats. Representation of these databases in a tabular format results in a large semantic gap and hence awkward decomposition and poor performance [GuSt82]. As a matter of fact, DBMSs are excluded from current CAD systems because of their inadequate performance [Maie89]. Many VLSI design tools are built on top of the raw file systems to gain efficiency. This puts extra burden on the tool designer as they have to code for detail I/O operations. In order to meet the requirements of these database applications, a "next-generation" DBMS is necessary.

2.2 Object-oriented DBMS

Object-oriented data models were proposed to increase the modeling power of traditional data models. Object-oriented databases adapted the concepts of object-oriented programming languages [Birt84, GoRo83] and semantic data models [Chen76, KePa76, HaMc81] with added database features such as persistent storage, query capability, concurrency control, transaction management, and consistency enforcement. Several research prototypes as well as commercial products have been reported. This includes Cactis [HuKi89], ORION [BaCG87], Iris [FiBC87], ENCORE [HoZd87], GemStone [MaOP85], VERSANT [VERS90], Ontos [Onto90], and Object-Store [Obje89]. Unlike the relational data model, researchers have not agreed on the definition of a single object-oriented data model. Nevertheless, these systems do share several common features which are essential for an object-oriented DBMS.

Two features are important to the discussion in this paper: *object identity* and *complex objects*. On the one hand, each object has its own identity represented by a unique object identifier (OID). OIDs are system-generated surrogates which remain invariant throughout the database's lifetime. With OIDs, an object has an existence that is independent of its value. On the other hand, the notion of complex objects allows the aggregation of heterogeneous objects as a single unit. A complex object may have its subobjects taken from different classes and linked together as a graph structure. The references to other objects are made explicitly through OID links. A major advantage of object-oriented DBMS is that many relational join operations can be prevented when retrieving complex objects; instead of joining tuples from different relations based on some attribute values, objects are simply accessed through their OIDs.

In many practical applications, a complex object structure can be expressed as a hierarchy or a directed acyclic graph (DAG). Typical operations performed on such structures are the navigation through links and the retrieval of the ancestors/descendants of a given node. A good strategy for clustering objects is to arrange nodes in a hierarchy/DAG into a linear *clustering sequence* such that these operations can be performed efficiently [BKKG88]. To cluster a hierarchy, one can store the nodes of a hierarchy in the *depth-first* order such that any node p in the hierarchy will have all of its descendant nodes stored immediately after p. This is very effective when an object and all of its descendants need to be retrieved together.

The clustering process for a DAG is more complicated. First a DAG has to be transformed into an equivalent hierarchy and then the depth-first clustering sequence can be enforced. The DAG is augmented with a virtual root node. For each node with multiple parents, a particular parent is chosen such that all descendants of a node in the DAG can be fetched in a single forward scan of the DAG. The result is a spanning tree starting from the virtual root node. It has been shown that the DAG clustering

scheme is an effective strategy in improving performance [BKKG88]. Moreover, clustering is more effective when object sizes are relatively small compared with the page size. [KiCB87].

Several characteristics are common to the existing clustering schemes: (*i*) They are static, i.e., objects are initially clustered when they are created; once allocated, objects are not reclustered at run time. (*ii*) They use disk pages as the clustering units, i.e., they assume an average access time for each page and do not take into consideration the physical adjacency of individual pages. (*iii*) Objects are clustered based on a unique type of relationship among the data elements; other relationships are ignored for clustering purposes. Although the existing static clustering schemes have been effective, one can identify two potential problems:

(1) A static clustering scheme initially offers a good placement policy for complex objects but does not take into account the dynamic evolution of objects. In applications such as design databases, objects are constantly updated during early parts of the design cycle. Frequent updates may destroy the initially clustered structure. To keep the object structure optimized, reorganization might be necessary for efficient future accesses [Deux90].

(2) Objects may be connected by several relationships which form independent hierarchies/DAGs. For example, in a design database, a design evolves through several phases, i.e., initial creation, design rule checking, correction, extraction, and simulation. At each phase, the design tool requires its own access pattern which may or may not be the same as the one supported by initial static clustering. Any single clustering scheme is unlikely to suit all such phases. It might be preferable to use different clustering structures in different phases [Maie89]. Furthermore, several users may access the same set of objects concurrently through different access patterns. Clustering objects based on the need of one application may sacrifice others.

3. OBJECT UPDATES

Our study of clustering schemes starts from a simple case where there is only one main relationship among objects: they are connected as a single hierarchy (a DAG can be reduced to its equivalent spanning tree, which is also a hierarchy). Using the aforementioned static clustering scheme, we investigated the following problem: how updates can destroy the originally clustered structure and what the system can do to reorganize it.

3.1 Updating Effects

For environments such as design databases, objects are constantly updated. This is especially true during the early part of the design cycle. Here, we consider the evolution of a hierarchical structure caused by object insertion and deletion. Both insertion and deletion can destroy the organization of a clustered structure. However, deletion is usually not a serious problem. Deletions can be done by simply marking the objects as deleted. The space can be reclaimed later. But, for insertions, new free space must be allocated immediately for new objects. The following discussion deals mainly with the insertion operations.

Regardless of the clustering strategy, the sequence in which nodes in a hierarchy are created is unlikely to be the same as the desired clustering sequence. Using the object hierarchy in Figure 1 as an example, we shall see how a clustered structure can be disturbed due to updates. For the sake of simplicity, we assume all nodes are of the same size and each disk page can hold up to three nodes. The insertions and deletions of nodes would be based on the criteria that maintains each page to be between 50 and 100 percent full. As shown in Figure 1, the nodes from the hierarchy are created in breadth-first order while the desired clustering sequence is depth-first. When a new node is inserted, its position within the clustering sequence is first determined according to the depth-first sequence. If the target page has enough space, the new node is simply added; if the page is full, a new page is acquired and objects are distributed evenly between the two pages. Notice that the final set of pages are chained together in a way that objects are arranged in depth-first order to preserve the desired logical sequence. However, these pages may not be physically close to each other since the adjacent pages may have already been occupied by other unrelated objects.

In previous studies of object clustering schemes [KiCB87, BKKG88, Chan89], performance was measured in terms of the number of disk pages to be accessed. These studies assumed an average access

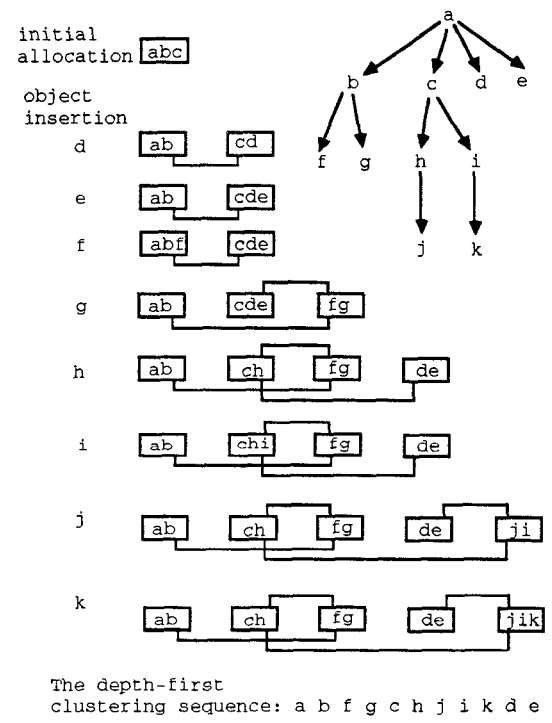


Figure 1. Object Updates.

time for each page and did not distinguish whether the pages were stored sequentially or randomly. It is clear that it takes more time to access a set of scattered pages than it does to access contiguous ones. This is due to the mechanical structure of current disk drives. Reading a data block from a disk involves seek time, latency time, and transfer time. For a set of sequential blocks, only one seek and latency time is required. For randomly distributed blocks, separate seek and latency times are required [Salz87].

3.2 Dynamic Reclustering

To solve the problem of object updates, one has to find a proper scheme to reorganize scattered pages into a contiguous chunk. This is similar to the file reorganization problem. Existing studies on file reorganization can be classified into two categories: *on-line* reorganization [Sode81, Omie85] and *off-line* reorganization [SoGo79, Bato82]. Optimal on-line reorganization is known to be an NP-hard problem [Omie85], thus the major effort is to find some heuristic rules to prevent excess overhead. For off-line reorganization, the major challenge is to determine the optimal reorganization points, that is, how often the reorganization should be performed. Our clustering strategy is based on the same paradigms used for file reorganization algorithms.

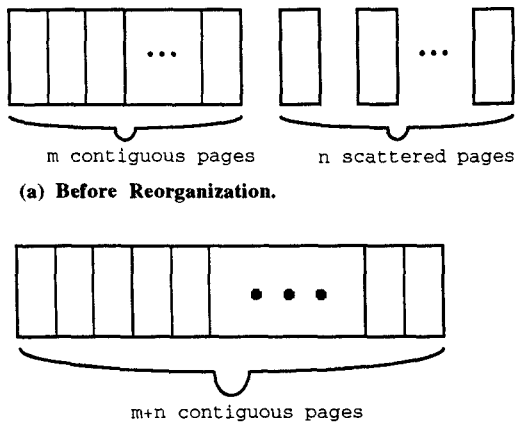
A **static** clustering scheme, as discussed before, does not recluster objects after they are created. A **dynamic** clustering scheme should try to recluster scattered pages when the access cost becomes too high. However, reclustering will generate overhead such as extra disk I/Os, so it is important to determine when a reorganization should occur. If the overhead is not justified, reclustering may actually degrade the performance. We propose a **cost model** to evaluate the benefit and overhead of reclustering.

Our clustering units are called *chunks*. A chunk is a collection of pages stored contiguously on the disk. Initially, a complex object hierarchy is stored in a single chunk. This happens when a complex object is first created, or right after the reorganization of the data. In our cost model, we assume a complex object occupies $m + n$ pages where m pages are clustered in a chunk and n pages are scattered on the disk (Figure 2a). Based on this assumption, m clustered pages need one disk I/O and each of the n scattered pages needs an additional disk I/O. Let ΔS be the average seek time, ΔL be the average latency time, and ΔT be the transfer time for a single page. The cost of accessing all $m + n$ pages in scattered form is estimated by T_{scat}:

$$T_{scat} = (1+n) \times \Delta S + (1+n) \times \Delta L + (m+n) \times \Delta T.$$

If all the $m + n$ pages are clustered in a single chunk (Figure 2b), the accessing cost would be:

$$T_{clus} = 1 \times \Delta S + 1 \times \Delta L + (m+n) \times \Delta T.$$



(a) Before Reorganization.

(b) After Reorganization.

Figure 2. Modeling Page-based Reorganization.

Here, T_{scat} corresponds to the result of a static clustering scheme, while T_{clus} corresponds to that of a fully dynamic clustering scheme in which related pages are always clustered. Using DEC's RA81 disk drive as a guideline [DEC82], we set $\Delta S = 28ms$, $\Delta L = 8.33ms$, and $\Delta T = 1.282ms$ (assume a page size of $2K$ bytes). When $m = 10$ and $n = 2$, the ratio of T_{scat} to T_{clus} is 1.84; when n increases to 10, the ratio becomes 8.04. It is clear that reclustering can reduce the access time significantly.

3.3 Cost Estimation

While a fully dynamic clustering scheme reduces the access time, the overhead of reorganization may offset the benefit. The reorganization cost is the time overhead spent for reorganizing scattered pages. We estimate the reorganization cost based on a *copy-and-reclaim* method [Bato82]. First, related pages residing in the scattered form are retrieved (T_{scat}). Then, the "old" storage space is reclaimed and a "new" contiguous space is located (T_{cpu}). And lastly, all pages are written to the new space (T_{clus}). If there is no contiguous space available on the disk, a total reorganization may be necessary. We assume that it is done off-line, which does not affect our cost estimation. Using this method, the cost of reorganization, T_{reorg}, can be expressed by:

$$T_{reorg} = T_{scat} + T_{cpu} + T_{clus}.$$

As the equation shows, regardless of m and n values, T_{reorg} is always greater than T_{scat}. This means that the reorganization overhead is not justified if the R/U ratio (the average number of retrievals after each update) is smaller than or equal to 1. We would like to determine how many retrieval operations (R) are needed to justify the reorganization cost, i.e.,

$$T_{reorg} + R \times T_{clus} \quad < \quad R \times T_{scat},$$

or

$$R \quad > \quad \frac{T_{scat} + T_{cpu} + T_{clus}}{T_{scat} - T_{clus}},$$

which can be further expanded to

$$R > \frac{(n+2)(\Delta S + \Delta L) + 2(m+n)\Delta T + T_{cpu}}{n(\Delta S + \Delta L)}.$$

Typically T_{cpu} is much smaller compared to the disk access time, i.e., $T_{cpu} \approx 0$. Using timing parameters from DEC's RA81 disk drive [DEC82], a plot of R against different values of m and n is depicted in Figure 3. This figure shows that for a smaller n, a larger R is needed to compensate the reorganization cost. For example, if $m = 50$ and the system performs reorganization whenever 2 scattered pages are found (i.e., use $n = 2$ as a threshold value), it would be beneficial only when the R/U ratio is greater than If the threshold value is chosen to be 8, the minimum R/U ratio needed is only 2. Notice that when R/U falls between the range of 1.2 and 2, the curves are very dense. It seems to suggest that the choice of threshold values is very sensitive to small variations of R/U value. To further exploit this issue, we developed a simulator to compare the actual access and reorganization cost.

3.4 Simulation

The goal of our simulation is to determine the total cost of different clustering strategies based on different sets of parameters such as m, n and read/write ratio. The total cost covers both the access cost and the reorganization cost. In our simulations, PR/W (page reads/writes ratio) represents the average number of pages retrieved after a new scattered page is created. Since a complex object is not always retrieved in its entirety, the introduction of PR/W permits the modeling of partial retrieval. When PR pages are to be retrieved from a set of m clustered and n scattered pages, we assume a uniform distribution for the probability of whether these PR pages are clustered or scattered.

For a fixed value of m, simulations were run to calculate the total access and reorganization cost for different choices of thresholds (n) and PR/W ratio. Figure 4 shows the results of a typical simulation run when $m = 10$. The total cost is accumulated over an interval until a total of 16 scattered pages are created. When PR/W $\leq$ 25, the total cost T_{tot} decreases as n increases. This means that infrequent reclustering yields a lower total cost. When PR/W $\geq$ 35, T_{tot} increases as n increases. The minimum T_{tot} occurs when $n = 1$, which means fully dynamic reclustering has the lowest total cost. When the PR/W value is around 28, T_{tot} does not change much as n changes. We call PR/W = 28 a break-even point for $m = 10$. Simulations were run for different values of m. The results show that for each run the curve pattern is the same as Figure 4 with a different break-even point.

In summary, reclustering should be performed whenever possible to keep the related objects clustered. Our simulation results show that for applications in which the read/write ratio is high, a fully dynamic reclustering strategy is justifiable. However, when the read/write ratio is not high enough, the reorganization overhead degrades the overall performance. In this case, on-line dynamic reclustering is not recommended. However, one should remember that the system can still perform off-line reclustering during off-peak hours to gain the clustering benefit. Since it is done off-line, the overhead will not affect the user's access time.

4. MULTIPLE RELATIONSHIPS

For applications with different relationships among objects, a clustering scheme based on a single relationship satisfies only one of the access patterns and sacrifices others. Based on the observation that some relationships are referenced more frequently than others at a certain phase of the objects' lifetime, we propose a leveled clustering scheme which takes into account multiple access patterns.

4.1 Weighted Digraphs

According to Chang and Katz [ChKa89], at least three types of structural relationships have been found to be useful for clustering objects: configuration, version, and correspondence. The following work is based on the assumption that for a set of related objects, one can find multiple relationship links with different access frequencies among objects. The access frequencies can be either measured from previous runs or predicted by the user's hints. Because of the nature of the algorithms used in application programs such as CAD tools, some relationship links may be traversed more frequently than others at a certain period of time. That is, different relationship links have different *weights*. With the existence of multiple relationships, the object structure is not limited to a single hierarchy or DAG. Related objects may be connected as a general directed graph (digraph), in which nodes represent objects and arcs represent relationships among objects. Arcs are associated with weights. An arc of weight w pointing from node a to node b (denoted by $a \xrightarrow{w} b$) means when a is accessed, the relative possibility of accessing b is w. Therefore, in a graphical representation of complex objects, the notation $a \xrightarrow{w_1} b$, $a \xrightarrow{w_2} c$, $w_1 > w_2$, shows the fact that the relationship between a and b is more significant than the relationship between a and c.

Our leveled clustering approach is a variation of the clustering sequence approach as was described in the last section. Using such a scheme, all nodes in the digraph are ordered into a clustering sequence in

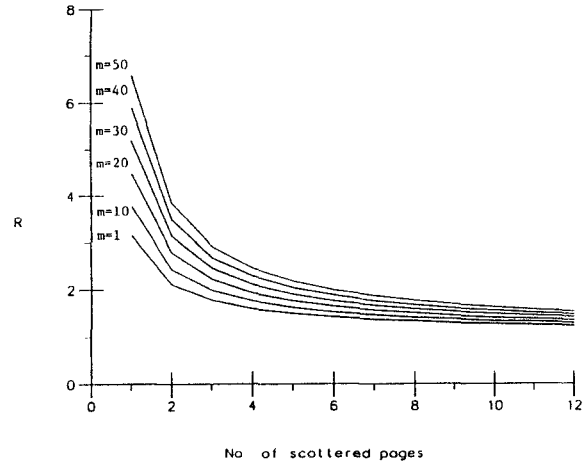


Figure 3. Minimum R to Compensate Reorganization Cost.

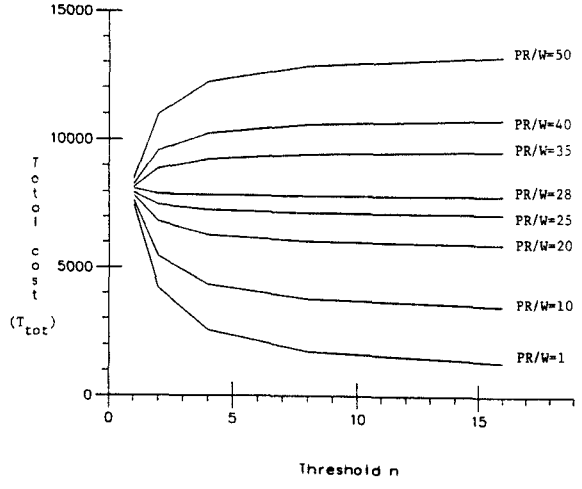


Figure 4. Total Access Cost.

which the more tightly connected objects are closer to each other. If updates occur after the sequence is established, the same page allocation and reorganization schemes described in the last section can be applied. This approach has two advantages: (*i*) When a new object is created, the storage manager can determine where to insert the newly created object easily according to the clustering sequence. No sophisticated calculation is needed. (*ii*) It offers an easy method to reorganize related but scattered pages into a contiguous chunk, which takes into account the saving of seek and latency time.

4.2 Leveled Clustering Algorithm

Given a digraph with weights assigned on arcs, we now propose a **leveled clustering algorithm** for arranging the nodes into a clustering sequence. This algorithm is inspired by Krushal's algorithm which constructs a minimum-cost spanning tree from a general graph [AhHU87]. However, the proposed scheme constructs a maximum weight spanning tree from a digraph. The concept of *supernode* is introduced which has the following properties: (*i*) A single node is a supernode. (*ii*) A group of supernodes connected by arcs of the current maximum weight is a supernode. (*iii*) All nodes in a supernode are ordered as a linear sequence and stored in a *bucket*. A *PriorityQueue* is also introduced which stores all different weights in the digraph in descending order. Our leveled clustering algorithm groups smaller buckets into larger ones recursively until all nodes in a digraph are covered by a single bucket. **Algorithm 1** illustrates the sequence of operations.

Assume a digraph as depicted in Figure 5, initially, every node in the digraph is a supernode and occupies a single bucket. At iteration 1, the current maximum weight is found to be 4. Nodes connected by weight-four arcs form two subgraphs: *ab* and *cd*. The nodes in each subgraph are grouped together as a supernode. The clustering sequence for nodes inside a supernode is determined by the directions of the arcs. Therefore, *a-b* and *c-d* are the clustering sequences inside the two buckets (Figure 5a).

At iteration 2, arcs of weight 3 are considered. Only two supernodes are to be merged, i.e., *ab* and *cd* (Figure 5b). Although there exist multiple arcs between these two supernodes, the order between *ab* and *cd* is determined solely by weight-three arcs; arcs with lower weights are not considered. This will keep our clustering sequence most effective.

Successive iterations are illustrated in Figure 5c and 5d. Notice that the clustering sequence inside a supernode is not interrupted by later iterations. For example, Figure 5c shows that when nodes *e* and *f* are added to the bucket, the existing sequence, *a-b-c-d*, remains contiguous. This is where the name **leveled** comes from. Using this leveled clustering scheme, we expect the retrieval along the most significant relationship links always has the best response time, while for other relationships the response time is expected to be better than that of random storage.

There is another advantage to our leveled clustered scheme. Typical object-oriented applications have only a limited number of relationships that are of interest to the clustering manager. In such cases, we can use a **template** to map relationships to their correspondent weights at current phase. Different phases would have different templates and the weight of a relationship may change from one phase to another. When the access pattern changes, the system can simply follow the new template to change the weights assigned to related links. Then, a reclustering process can be activated to create a new clustering sequence using the leveled clustering scheme. This kind of reclustering can be performed whenever the access pattern changes; clients do not have to wait until the targeted objects are updated, as in the existing clustering schemes.

4.3 Interleaved DAGs

In Algorithm 1, we simply assumed that nodes inside a supernode can be ordered into a linear sequence according to the directions of arcs. The situation is

```
PROCEDURE LeveledClustering (
    V: SET of vertices
    E: SET of arcs {u --w--> v | u, v ∈ V, w : weight} );
VAR
    Q: PriorityQueue (w1 > w2 > ··· > wk);
BEGIN
    Initialize(V); { put each node in V in a bucket }
    WHILE (Q not empty) AND (No. of buckets > 1) DO
    BEGIN
        c_w = DeleteMax(Q); { get the current maximum weight }
        FOR each subgraph s_G connected by arcs with weight c_w DO
        BEGIN
            s_n = Convert(s_G); { convert a subgraph into a supernode }
            Merge(s_n); { merge all buckets in s_n into a single bucket according to the directions
                          of arcs with weight c_w, arcs with lower weights are not considered }
        END
    END
END; {LeveledClustering}
```

Algorithm 1. Leveled Clustering

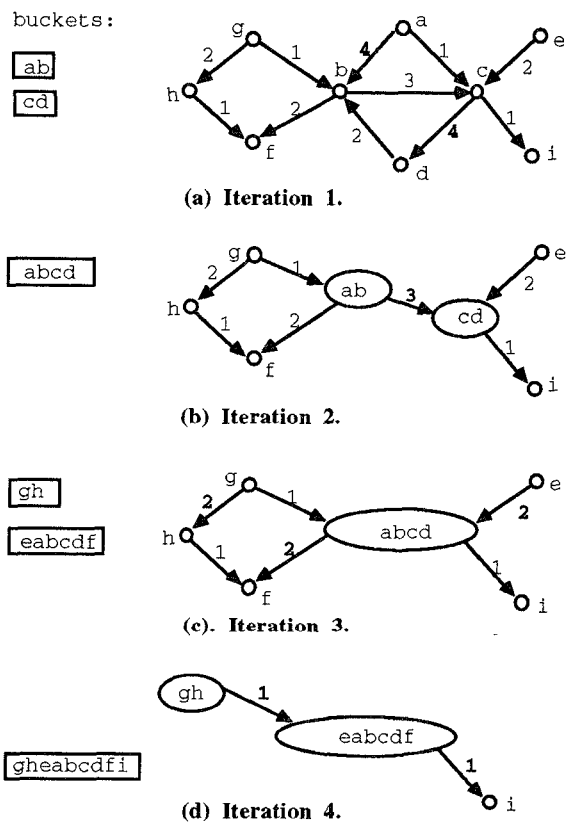


Figure 5. The Leveled Clustering Algorithm.

actually more complicated. If nodes of the same level are connected as a linear list, the ordering is trivial. If they form a tree structure, the depth-first traversal sequence is appropriate. If they form a DAG, the DAG should be transformed into a tree as was described earlier [BKKG88]. However, if the digraph contains cycles, there are no efficient ways to find the best clustering sequence. Fortunately, cycles hardly exist in typical design-oriented database applications. Considering the semantics of object links used to represent composite objects, version objects, and equivalent objects [ChKa89], cycles do not make sense in connecting the objects at all.

A close investigation of semantic links shows that design objects are actually connected by **interleaved DAGs**. Figure 6 shows an example of interleaved DAGs. The version history of an object forms a tree structure. Configurations are formed by combining versions of different objects into composites. Version and configuration relationships are orthogonal; i.e., they form semantic planes which are perpendicular to each other. An equivalence object ties together equivalent objects in alternative representations. As one can see from the figure, multiple DAGs are interleaved but no cycles exist.

If we use a template (version:3, configuration:2, equivalence:1) as the relative weights of the relationship links, the resulting clustering sequence of Figure 6 is quite interesting. First, different versions of the same object are grouped together as a supernode. Then, the version supernodes are ordered according to the DAG structure of the configuration links. Finally, those composite objects which are equivalent are brought close to each other. Applying the leveled clustering scheme on the interleaved DAGs generates a clustering sequence with the following two properties:

[1] **Single scan property**: All nodes reachable from a node o through link traversal are placed after

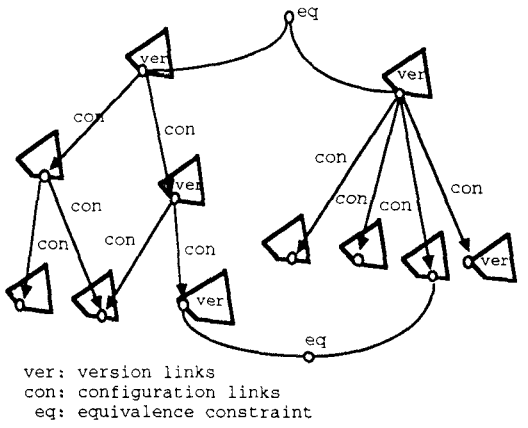


Figure 6. An Example of Interleaved DAGs.

o in the clustering sequence. This ensures that when o is accessed, its related objects can be accessed by a single forward scan of the database.

[2] **Multiple level property**: If $o \xrightarrow{w_1} p$, $o \xrightarrow{w_2} q$, and $w_1 > w_2$, then the distance between o and p is less than the distance between o and q. This ensures that nodes with heavier links are clustered closer than nodes with lighter links.

4.4 Simulations

Given a multiple-relationship environment, we conducted simulations to show the effectiveness of the leveled clustering scheme. The simulator assumes an environment that serves one user at a time. While the system is a multi-user environment, the requests from users can be queued up and served one by one. The specification of the simulated disk drive is shown in Table 1. Different disk drives may show slight variations but the results should be similar. Notice that the maximal seek time is 50 *ms* (moving across 1258 tracks) and the seek time for one track is 7 *ms*. This 7 *ms* includes the overhead for accelerating, decelerating and accurate positioning of the R/W heads. This overhead has to be paid no matter how far the R/W arm moves. To switch from one head to another, it takes up to 6 *ms*. However, the head switching time can be overlapped with the seek time if the R/W heads do move from one cylinder to another.

For the sample database, we assumed a three-level clustering sequence. The level size (number of nodes in a supernode) ranges from 10 to 200. A level size of k means two consecutive level-two objects are actually separated by k level-one objects, while two level-three objects are separated by k^2 objects. The simulated object sizes range from 256 to 2048 bytes ($\frac{1}{8}$ page to one full page). We did not include very large objects in our simulation because: (*i*) Large objects do not benefit from clustering as much as small objects do. (*ii*) A large object by itself can be stored using a B-tree structure [CaDR86]; only the descriptors of large objects need to be clustered.

Figure 7 illustrates the average access time per object for variant object sizes at different clustering levels. The object size ranges from 256 to 2048 bytes,

Table 1. Specification of the Simulated Disk Drive.

Total No. of Cylinders	1258
Heads (tracks) / cylinder	14
Block (page) size	2048 bytes
Blocks (pages) / track	13
Total No. of pages	228,956
Max. latency time	16.6667 ms
Average latency time	8.3333 ms
Transfer time / page	1.282 ms
Max. seek time (1258 tracks)	50 ms
One track seek time	7 ms
Average seek time	28 ms
Head switching time	6 ms (max.)

and each complex object retrieval is assumed to involve 10 individual objects. For level-one clustered objects, the average access time increases from about 4 *ms* to 5 *ms* as the object size increases. The access time for level-two and level-three objects range from 7 to 25 *ms* compared to 43 *ms* for non-clustered objects. This shows the effectiveness of the leveled clustering scheme. Although related objects may not be stored within the same disk page, one can still reduce the access time by putting them within the same track or the same cylinder.

For level-two objects, average access time increases rapidly as object size increases. To further investigate the behavior of the leveled clustering scheme, we designed another simulation which increases the level size to as high as 200. In this simulation, objects have a fixed size of 512 bytes and each complex object retrieval is assumed to involve 10 individual objects. The access time for level-two and level-three objects are plotted in Figure 8. The access time for level-one and non-clustered objects were consistent with the previous simulation. However, we found a sawtoothed pattern for the access time of level-two objects (Figure 8a). This is actually compatible with the mechanical movement of disk R/W heads. The peaks where level sizes are around 50, 100, and 150 show the maximal latency was paid between two consecutive accesses of level-two objects. As it passes the points of maximum latency, the curve drops when the level size is around 60, 110, and 160.

For level-three objects (Figure 8b), an irregular pattern was observed; however, the level-three clustering still offers a better performance than the non-clustered scheme. The saving in access time is partly due to the reduced seek time because objects are stored on adjacent cylinders. Comparing the two curves, we found that there is no significant difference in access time between level-two and level-three clustering. The amounts of saving are comparable in both cases. In summary:

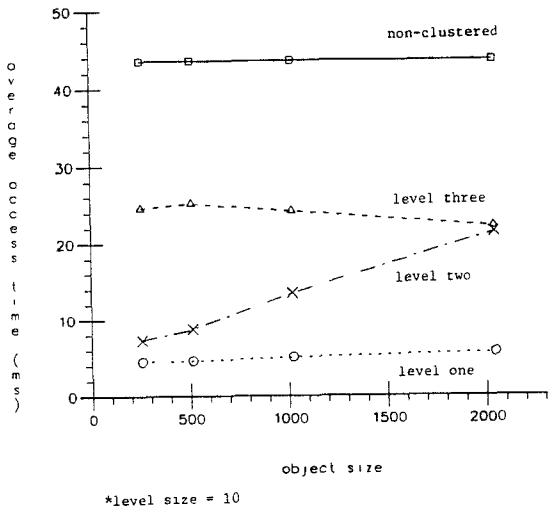
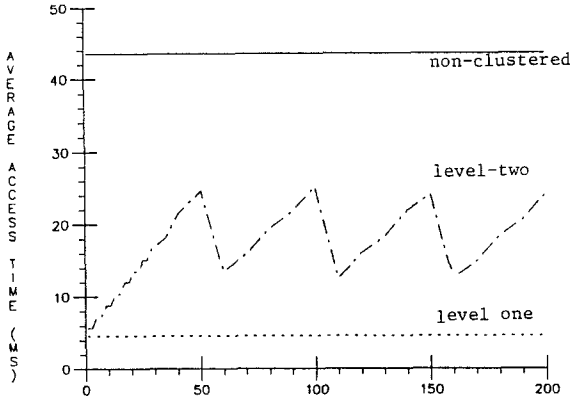
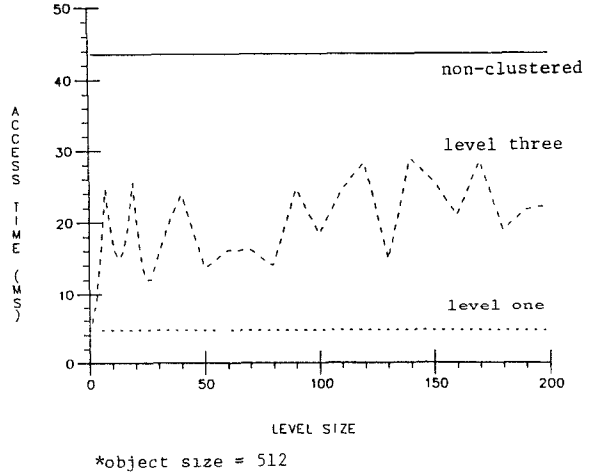


Figure 7. Access Time of Leveled Clustering Scheme.



(a) Level-two.



(b) Level-three.

Figure 8. Access Time of Level-two/-three Clustered Objects.

(1) Level-one clustering is very effective.
(2) Level-two and level-three clustering show improvement compared with a single-level clustering scheme. The average access time for level-two and level-three clustering is around 50% of the random access time.
(3) Clustering objects on the same cylinder provides similar performance improvement as clustering them on the same track. If two related objects cannot be stored on the same page, one should attempt to store them on the same track or the same cylinder.

5. CONCLUSIONS AND FUTURE DIRECTIONS

This paper investigated issues associated with the clustering of objects in object-oriented databases. Existing clustering schemes are static because they do not perform reclustering. In addition, their clustering strategy is based on a single relationship among objects. We explored two problems associated with the existing schemes: object updates and multiple relationships.

Frequent updates on a complex object structure may create scattered disk pages which need to be reclustered to reduce the access time. We proposed a cost model to evaluate the benefit and overhead of reclustering. Simulation results showed that the reorganization points depend primarily on the read/write ratio. When the read/write ratio is high, a fully dynamic reclustering strategy is affordable; when the ratio is low, an off-line reclustering strategy is a better choice.

For environments where objects are linked with multiple relationships, we proposed a leveled clustering scheme to order nodes in a weighted digraph into a clustering sequence. Simulations showed that the access time for leveled clustering is around 50% of the random access time. It was also recognized that clustering objects on the same track and on the same cylinder provide similar performance improvement. If related objects cannot be clustered on the same page, clustering them on the same track or cylinder will improve the performance as well.

Besides magnetic disks, new technologies have made high capacity optical disks economically feasible. Our discussion regarding the leveled clustering scheme will hold for optical disks as well. In addition, adjacent tracks on an optical disk can be accessed without moving the access head. This is equivalent to the increase of track size on magnetic disks, which should have a positive effect to our clustering scheme.

Although the leveled clustering scheme is effective for retrieving objects along level-two and level-three relationship links, the performance improvement is less than that of level-one clustering. A better performance might be made possible by including buffering schemes. In addition to the traditional page-based prefetching and LRU replacement policies, object-based buffering using semantic links is a great

challenge for object-oriented DBMSs. Our future research directions will be in the development of efficient buffering schemes and the integration of clustering and buffering schemes in a real object-oriented database environment.

REFERENCES

[**AhHU87**] Aho, A.V., J.E. Hopcroft, J.D. Ullman, *Data Structures and Algorithms*, 1983 Addison-Wesley, Reading, MA.

[**BaCG87**] Banerjee, J., H.-T. Chou, J.F. Garza, W. Kim, D. Woelk, and N. Ballou, "Data Model Issues for Object-Oriented Applications," *ACM Trans. on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 3–26.

[**Bato82**] Batory, D. S., "Optimal File Designs and Reorganization Points," *ACM Trans. on Database Systems*, Vol. 7, No. 1, March 1982, pp. 60–81.

[**Birt84**] Birtwistle, G.M. et al. *Simula Begin*, 2nd ed., Sweden, Studentlitterature, 1984.

[**BKKG88**] Banerjee J., W. Kim, S.-J. Kim, and J.F. Garza, "Clustering a DAG for CAD Databases," *IEEE Trans. on Software Engineering*, Vol. 14, No. 11, November 1988, pp. 1684–1699.

[**CaDR86**] Carey, M., D.J. DeWitt, J.E. Richardson, and E.J. Shekita, "Object and File Management in the EXODUS Extensible Database System," *Proc. 12th Int'l Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986.

[**Chan89**] Chang, E.E., *Effective Clustering and Buffering in an Object-Oriented DBMS*, Dissertation in Computer Science, University of California, Berkeley, CA, 1989.

[**Chen76**] Chen, P.P., "The Entity-Relationship Model–Toward a Unified View of Data," *ACM Trans. on Database Systems*, Vol. 1, No. 1, March 1976, pp. 9–36.

[**ChKa89**] Chang, E.E., and R.H. Katz, "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS," *Proc. 1989 ACM SIGMOD Int'l Conf. on Management of Data*, Portland, Oregon, June 1989. pp. 348–357.

[**Date86**] Date, C.J., *An Introduction to Database Systems*, 4th ed., 1986 Addison Wesley, Reading, MA.

[**DEC82**] *RA81 Disk Drive User Guide*, 1982 Digital Equipment Corporation, EK-0RA81-UG-001.

[**Deux90**] Deux, O. et al., "The Story of O_2," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 2, No. 1, March 1990, pp. 91–108.

[**FiBC87**] Fishman, D.H., D. Beech, H.P. Cate et al., "Iris: An Object-Oriented Database Management System," *ACM Trans. on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 48–69.

[**GoRo83**] Goldberg, A. and D. Robson, *Smalltalk-80 The Language and its Implementation*, 1983 Addison-Wesley, Reading, MA.

[**Gust82**] Guttman, A., and M. Stonebraker, "Using a Relational Database Management System for Computer Aided Design Data," *IEEE Database Engineering, Bulletin*, Vol. 5, No. 2, June 1982, pp. 21–28.

[**HaMc81**] Hammer, M. and D. McLeod, "Database Description with SDM: A Semantic Database Model," *ACM Trans. on Database Systems*, Vol. 6, No. 3, September 1981, pp. 351–386.

[**HoZd87**] Hornick, M.F., and S.B. Zdonik, "A Shared, Segmented Memory System for an Object-Oriented Database," *ACM Trans. on Office Information Systems*, Vol. 5, No. 1, January 1987, pp. 70–95.

[**HuKi89**] Hudson, S.E., and R. King, "Cactic: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System, " *ACM Trans. on Database Systems*, Vol. 14, No. 3, Sep. 1989, pp. 291–321.

[**KePa76**] Kerschberg, L., and J.E.S. Pacheco, "A Functional Data Base Model," *Tech. Report*, Pontificia University, Catolica do Rio de Janeiro, Rio de Janeiro, Brazil.

[**KiCB87**] Kim, W., H.-T. Chou, and J. Banerjee, "Operations and Implementation of Complex Objects," *Proc. 3rd Int'l Conf. on Data Engineering*, 1987, pp. 626–633.

[**Maie89**] Maier D., "Making Database Systems Fast Enough for CAD Applications," *Object-Oriented Concepts, Databases, and Applications*, W. Kim, and F.H. Lochovsky (eds.), 1989 Addison-Wesley, pp. 573–582.

[**MaOP85**] Maier, D., A. Otis, and A. Purdy, "Object-Oriented Database Development at Servio Logic," *Database Engineering*, Vol. 4, 1985, IEEE Computer Society Press, pp. 294–301.

[**Obje89**] *An Introduction to Object Store, Release 1.0*, 1989 Object Design Inc., Burlington, MA.

[**Omie85**] Omiecinski, E., "Incremental File Reorganization," *Proc. 11th Int'l Conf. on VLDB*, Stockholm, Aug. 21-23, 1985, pp. 346–357.

[**Onto90**] *ONTOS Release 2.0 Data Sheet*, 1990 Ontologic Inc., Burlington, MA.

[**Salz87**] Salzberg, B., *File Structures: An Analytic Approach*, 1987 Prentice Hall, Englewood Cliffs, NJ.

[**Sode81**] Soderlund, L., "Concurrent Database Reorganization — Assessment of a Powerful Technique through Modelling," *Proc. 7th Int'l Conf. on VLDB*, 1981, pp. 499-509.

[**SoGo79**] Sockut, G. H., and Goldberg, R. P., "Database Reorganization—Principles and Practice," *ACM Computing Surveys*, Vol. 11, No. 4, 1979, pp. 371–395.

[**VERS90**] *VERSANT, Product Profile*, 1990 Versant Object Technology Co., Menlo Park, CA.