Dynamic Clustering in Object Databases Exploiting Effective Use of Relationships Between Objects

Frédérique BULLAT, Michel SCHNEIDER
Laboratoire d'Informatique
Université Blaise Pascal Clermont-Ferrand II
Complexe des Cézeaux, 63177 Aubière Cédex, FRANCE
E-mail: schneider@cicsun.univ-bpclermont.fr
Phone: (33) 73.40.74.35
Fax: (33) 73.40.74.44

Abstract

This paper concerns the problem of clustering objects onto units of secondary storage to minimise the number of I/O operations in database applications. We first investigate problems associated with most existing clustering schemes. We then propose STD, a Statistic-based Tunable and Dynamic clustering strategy which is able to overcome deficiencies of existing solutions. Our main contributions concern the dynamicity of the solution without adding high overhead and excessive volume of statistics. Reorganisations are performed only when the corresponding overhead is strictly justified. Clustering specifications are built from observation upon objects life, capturing any type of logical or structural inter-object links. Moreover, our clustering mechanism does not need any user or administrators hints, but remains user-controlled. A partial validation of STD has been made using Texas.

Keywords

Clustering, Buffering, Object-Oriented DataBase System, Performance.

1 Introduction

Clustering is an effective mechanism for improving object-oriented DBMS. Various inter-object links allow navigation through the database and retrieval of complex objects. For an object-oriented system, it is very important to traverse the object graphs structure efficiently. Clustering of related objects on disk minimises the number of pages accessed during a transaction and has thus a great impact on the overall performance of the system. It reduces client-server communication and disk I/O costs but also the number of locks to manage as well as the number of writes to store in journals. It improves paging performance and client buffer memory usage. So, 'intelligent' physical placement of objects is a central issue for performance requirements.

Several characteristics can commonly be identified in existing clustering strategies [3]. First, they are mainly static. Objects are grouped at creation and are not reclustered afterward at run time when use of data evolve. Moreover, systems do not generally offer any measuring and reorganisation tools. Second, clustering is rarely self-driven and user's hints or explicit placement schema are used at object creation time. Third, existing schemes are designed to cluster objects by considering structural relationships among classes inside the database schema. They commonly don't consider logical[1] relationships, neither IS-A relationships nor multiple relationships among objects. Moreover, clustering schemes are all based on a common policy for every objects of a given class. This does not allow any individual object behavior. Finally, most clustering policies use disk pages as clustering unit. Hence, they do not consider performance issued by the physical dispositions of accessed pages. Clustering among several pages may be really efficient especially if coupled with an appropriate buffering policy. Besides, related objects placed in consecutive pages on disk can be accessed avoiding expensive moving of disk heads.

This paper introduces a new clustering scheme that may be convenient for any object-oriented DBMS. Its main objectives are autonomy, flexibility, dynamicity and of course efficiency. It tries to reach the best physical placement regarding the database use. STD clustering specifications are based on operations performed on individual objects in order to meet the requirements of different access patterns. We try to determine how objects are actually used together, regardless the type of relationships. For the reclustering problem, we propose a dynamic scheme that may reorganise the storage space when placement of objects becomes obsolete (i.e. when objects are not used by applications as they usually were). This can occur for many reasons: schema evolution, changing of parent objects, added or removed structural or logical links. A cost model is introduced to assist reorganisation decisions. A great effort is made, first, to reduce the number of statistics to a minimal representative set; then, to correctly estimate the benefit of reclustering before doing any change; and finally, to wait for the appropriate moment, in order to avoid penalising applications when reclustering. Clustering units are chunks (sets of contiguous disk pages) instead of individual pages. Moreover, the clustering proposal is improved by a specific buffering scheme, including appropriate prefetching and replacement policies. The entire solution is self-driven but remains user-controlled through tuning parameters.

The remainder of this paper is organised as follows. Section 2 addresses the problems associated with most of existing clustering schemes. Section 3 stresses out our motivations and presents the principles of the STD clustering scheme. Section 4 presents the underlying machinery existing behind this solution from an implementation point of view. Section 5 proposes a partial validation of STD

---

[1] We mean by 'logical' relationship all links that are not described in the DB schema but that may be constructed at any time by applications in order to manipulate logically related objects. Examples of such relationships can be: extensions of classes containing objects having the same values for some attributes (in order to run database requests), or collections of neighboured objects often accessed together when displaying geographical cards. This kind of relationship is expressed of course with physical pointers but differs from classical structural links.

through an implementation with Texas and some experiments to situate the efficiency of the clustering. Finally, section 6 concludes this paper and discusses costs, feasibility and perspectives for this work.

2 Related Work

This section presents the main characteristics of most existing clustering schemes and points out several associated problems. One can refer to [3] for more details on object clustering strategies. First, in most operational OODBMS, clustering depends on user hand-activation (made through application code, as in ObjectStore [15], ONTOS [16], and GemStone [17] or by means of online commands, as in ORION [14] or with a clustering procedure, as in Cactis [13]). We think that placement of objects should be considered at any time by the system. It may of course need user hints, but should not depends on hypothetical external activations. O_2 seems to be the only operational system to propose a systematic clustering solution integrated to the DBMS [1] (this clustering strategy will be operational for the next version). Other solutions [5, 8, 11] have been suggested but never implemented.

Another common characteristic is the lack of dynamicity. Many suggestions have been made to get reclustering of objects [6, 8]. But these solutions are not really dynamic[2]. In fact, they only control placement of objects when objects are modified. We think that, for many reasons, an object may become misclustered even if it has not been modified (when clustering specifications evolve, for example). We have only found one proposal [11] that can be really considered as 'dynamic'. Based on a garbage collection process, it is very attractive because it adds only small overhead to the existing process. However, disk garbage collection is itself a very expensive process, and, for this reason, is rarely implemented in existing DBMS, except in GemStone [17]. Moreover, we think that dynamicity is at the same time, absolutely necessary but also very dangerous since disk space organisation may never stabilise and prove costly. Before reclustering, costs and benefits must be precisely estimated.

Besides, we consider that, in most cases, the importance of user hints is too high. Often, for example in ONTOS [16], ObjectStore [15] and ORION [14], users state precisely how to cluster some objects by indicating in what target segment or near what object y an object x must be stored (within applications code or by means of online commands). Sometimes, systems accept user hints related to the DB schema. In ENCORE [12] and EOS [11], users affect priorities to structural links between classes. O_2 proposes to define a specific placement schema, by means of placement trees [1] created by the DB Administrator. None of these solutions are really convenient because programmers, administrators and even users may have no correct idea on the way objects will be used together. This is the reason why in Cactis [13], for example, no user hint is required. The system tries to determine itself a good clustering.

---

[2] We qualify by 'dynamic' any solution able to detect and replace every object that becomes misclustered for any reason.

The most critical point of a clustering strategy is, of course, the pertinence of the clustering specifications. There are two major kinds of specifications: (i) those based on structural links between classes (depicted in the DB schema), (ii) those built with observation upon objects life. For the first kind (i), solutions differ essentially on the nature of the considered links. In most cases, systems base their clustering policy on structural links (defined in the DB schema). [7] and [8] propose to cluster upon other kinds of relationships, as equivalence relationships and version relationships. Generally, one type of relationship is considered at a time. [8] suggests a multi-level strategy to order different weighted links in a same clustering sequence. Moreover, IS-A relationships[3] and logical relationships are never considered. Concerning the second kind (ii), many papers propose to base clustering on access probabilities deduced from statistics [13, 19, 20, 21]. Defaults of many published works in this domain are the lack of precise information concerning: (j) the way to collect, store and handle statistics, (jj) the generated costs, (jjj) what to do while there are not enough statistics, (jjjj) the necessity of braking and balancing system reactions when confronted to continuous statistics changes. In the two kinds of strategies (i) (ii), clustering specifications are generalised at the class-level. So that, all objects of a given class are clustered in the same way, denying any individual behavior. Lastly, few systems handle the problem of shared objects in a deterministic way. If an object x is linked through the same type of relationship to several other objects, there are several clustering solutions. Some systems, as ENCORE [12], allow replication of object x, so that each replicated object is stored near its parent. Other systems, as EOS [11], rely on user hints. The others are self-electing systems: i.e. they either place randomly x close to one of the objects, or use statistics to take decisions [13, 19].

We can also note a lack of control and tuning tools. O_2 proposes a cost model [2] based on analysis of objects methods, but it has not been implemented. [6] suggests to control clustering efficiency within the buffering module. But nothing is said about the way to do it. Finally, clustering efficiency is generally reduced by the lack of adapted buffering schemes, and especially by the lack of adapted replacement policies [9]. Traditional LRU replacement policies, for example, do not consider relationships existing between objects of different pages when electing pages for replacement. As a result, some likely-to-be-used objects may be swapped out because the buffer manager does not take into consideration that these objects are strongly related to currently-used objects. The I/O gain introduced by clustering is then decreased by traditional buffering. [6] has suggested a context-sensitive replacement policy which would take care of inter-object relationships to set relevant priorities on pages. However, the proposed solution does not take advantage of a cluster-based policy. It has not yet been implemented but it seems that the corresponding overhead may prove costly.

---

[3] Clustering objects upon the inheritance graph is pertinent only in the case of DBMS having a vertical distribution storage model (inherited attributes and member attributes are not physically stored in the same object).

3 The STD Clustering Strategy

3.1 Motivations

This section addresses the main objectives of our policy. They consist in the following points:

Clustering specifications pertinency
Considering pertinent clustering specifications is the primary point of an effective strategy. First, we opt for clustering objects upon observations, rather than upon user estimations. We think that schema analysis is not completely appropriate to deduce clustering specifications. Then, we consider that different objects of a given class may have different behaviors. So, efforts will be made to avoid clustering on classes links basis. Finally, we want to capture any type of inter-object relationship. We are only interested in the frequency of simultaneous accesses to objects. Our objective is to obtain a distance between two objects on disk proportional to their attractive force, whatever the reason for their simultaneous uses. It includes many kinds of relationships: (i) structural links depicted in DB schema (composite links, references, IS-A relationships), (ii) logical links which lead applications to construct data structures incorporating objects identifiers, (iii) physical fragmentation of objects due to the storage model of object managers (to store variable-length attributes or inherited attributes, for example). Also, we would like to be able to solve shared objects conflicts, thanks to the managed statistics.

Space organisation pertinency
First, clustering must be an integral part of object management fonctionalities. It must be a constant preoccupation of the system, and its activation must not depend on users. Second, we consider that clustering among several pages is an important issue for efficient clustering. So, we will introduce an adapted type of storage unit. Finally, we think that dynamicity of the solution must be closely controlled. This means that: (i) storage costs (statistics storage costs and cluster storage costs) must be reduced, (ii) reorganisation pertinency must be measured. Reclustering must occur only if estimated gains exceed reorganisation costs. It must take into account the transactional rate in order to minimise concurrency problems and user transaction penalisation during reclustering operations. Reactions must be balanced and braked up, when confronted to continuous statistics changes. Besides, reclustering of an object necessitates to consider the whole set of related objects. We must avoid bringing two objects closer, without looking at the other related objects which also have an attractive force to respect.

Buffering policy
Introducing a buffering policy adapted to STD clustering is also one of our preoccupation, in order to benefit of clustering among several pages. This includes prefetching and cluster-based replacement policies. Prefetching will be closely controlled and replacement algorithm will not be more expensive than LRU-like policies.

Control and tuning tool
The last objective is to integrate tuning capabilities, in order to control system reactivity and statistics pertinency, for example.

3.2 About Statistics and Dynamicity

Clustering objects upon observation of objects life means statistics management.

This choice supposes that if, in the recent past, objects were often used together, they are likely-to-be-together-used in the near future. Statistics on the use of databases allow estimations of access probabilities. Besides, a dynamic clustering scheme must recluster scattered related pages when access costs become too high. However, if the corresponding overhead is not justified, reclustering may actually degrade performance. Modifying placement of a set of objects o_i is justified only if reorganisation overhead (T_{reorg}) plus the total time of future accesses (T_{clus}) to the clustered o_i is smaller than the total time of current accesses to the scattered o_i (T_{scat}) [8].

$$T_{reorg} + T_{clus} < T_{scat}$$

If n is the number of future accesses (read or write) to the individual o_i, we have: $T_{scat} = n * t_{scat}$ and $T_{clus} = n * t_{clus}$ where t_{scat} and t_{clus} are elementary access times. So, have we :

$$T_{reorg} < (n * t_{scat}) - (n * t_{clus})$$

$$n > \frac{T_{reorg}}{t_{scat} - t_{clus}}$$

This formula can be paraphrased as follows: the higher the overhead is, the larger the number of future accesses; the higher the clustering benefit is, the smaller the future access rate.

Ideally, we may manage a complete set of statistics in order to keep trace of: (i) any simultaneous access between two objects, (ii) access frequencies of any accessed object. Then, we could evaluate reclustering benefits. Certainly, it is not interesting to recluster when access frequencies are too small, even if objects are strongly related. But according to implementation considerations, it is impossible to manage the whole set of statistics for every objects, and during the whole objects life. The way of managing STD statistics solves this kind of problem.

3.3 The STD policy

Preliminaries

Our solution proposes to place strongly related objects as close as possible to each other. We consider that valuated links can be determined only during objects life. Weights of links are of course unknown at object creation time. We then propose to have a default placement of objects, using classical DB schema clustering specifications (priorities on structural links, as in EOS [11], for example). Modification of this initial placement will be considered as soon as estimated weights of links will become really significant.

The concept of inter-object link

As we said in section 3.1, we want to capture any type of inter-object relationships (structural, logical, inheritance, version, equivalence, etc...). This includes the relationships defined explicitly through the DB schema (aggregation, references, inheritance), as well as those defined by programmers (for example to retrieve some logically related objects), and those introduced by the system (for example, to retrieve all attributes of fragmented objects). All these links are materialised in the database by pointers using OID. So, detecting and valuating links necessitates following each dereferencement of persistent pointers made by the system during database life. This explains why we propose to characterise any inter-object link, leading to simultaneous use of objects, by the following definition.

Definition: There is a *link* (o_i,o_j) between two objects o_i and o_j when o_j is accessed from o_i during a transaction. Each dereferencement of a persistent pointer is considered as a link between the initial object and the referenced object.

Statistics collecting

Statistics are collected within transactions, during an *Observation Period P*. During P, each transaction T_i memorises links detected between objects. At commit time, an asynchronous process p_i is created to analyse each T_i observation results and to compute: (i) the number of times a link has been detected during T_i between each pair of related objects, (ii) the number of accesses to each object accessed during T_i . Observation results computed by each process p_i are stored in two transient data structures: observations (i) are stored in an Observation Matrix called MO, observations (ii) are stored in an Observation Vector called VO[4]. MO and VO are concurrently updated after commit time by processes p_i. MO(i,j) represents the number of times the link (o_i,oj) has been detected during P, from o_i to oj, by all transactions . MO is a sparse matrix where entries are identifiers (OID) of the referenced objects. Its size is incremented by one each time a new entry is added. Since a maximal size n is a priori assigned to MO, the length of P varies according to the DB access rate.

Elementary linking factors

In order to weight a link between two objects o_i and o_j, we propose the following indicator computed from the statistics:

$$\frac{MO(i,j)}{VO(i)} + \frac{MO(j,i)}{VO(j)}$$

It constitutes an estimation of the probability $p(o_i/o_j)$ of accessing o_j when accessing o_i plus the probability $p(o_j/o_i)$ of accessing o_i when accessing o_j. Of course, this estimation is very rough since it comes from a limited observation period. However, we are not at all interested in a fine estimation of this probability since the use of objects dating from a distant period is no longer relevant of the current use. So, in the following definition, we speak about a factor instead of an effective probability.

Definition: The *elementary linking factor* fe_{ij} used to weight attractive force between objects o_i and o_j is deduced from observations within period P and computed, as indicated by Figure 1.

The two thresholds T_{fe} and T_{fa} are used to validate links observed during P. T_{fe} is the value under which a computed linking factor is not considered as significant, since the frequencies of accessing o_i from o_j and o_j from o_i during P were too small. T_{fa} is the value under which the number of accesses to individual objects is too small to justify the reclustering overhead (see section 3.2).

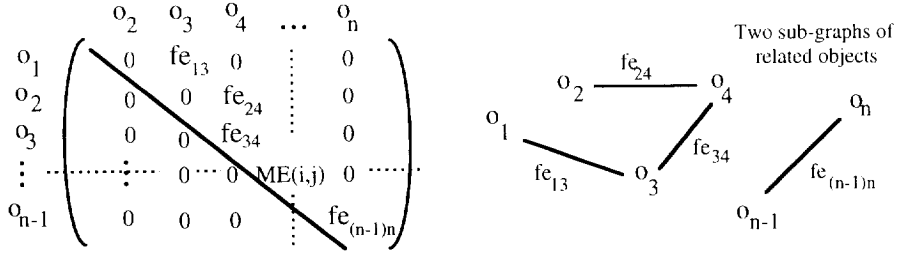
---

[4] From an implementation point of vue, such structures may not be stored as matrices and vectors. Hash tables, for example, may be used to keep and retrieve efficiently STD transient and persistent statistics.

```
for i --> 2 to n, for j --> 1 to n-1,    // Every link belonging to MO is processed
do
 if MO(i,j) ≠ 0
 then
    if (VO(i) >= Tfa) or (VO(j) >= Tfa)
       // Access frequencies to objects must be high enough to
       // consider the corresponding linking factor as significant
    then
       fe_ij = ME(i,j) = MO(i,j)/VO(i) + MO(j,i)/VO(j)
       if fc_ij < Tfe then fe_ij = 0
       // The linking factor is considered as not significant
       // and set to null if its value is under the Tfc threshold
    endif
 endif
enddo
```

Fig. 1. Computing of elementary linking factors

The *elementary linking factor matrix ME* is constructed at the end of the period P. It is sparse and triangular (fe_{ij} values are the same as fe_{ji} values since they represent the frequency of having objects oi and oj accessed simultaneously, regardless the link orientation). It defines a *graph of elementary inter-object links* (Figure 2). This is a non-oriented weighted graph in which nodes represent objects and arcs represent weighted relationships among objects.



ME - The elementary linking factor matrix

The corresponding graph of elementary links

Fig. 2. Elementary inter-object links

Consolidated linking factors

Significance of elementary linking factors is limited to the context of a period P. They are used to compute consolidated linking factors.

Definition: The *consolidated linking factors* fc_{ij} are persistent values updated at the end of period P, as follows:

$$fc_{ij(new)} = w * fc_{ij(old)} + (1 - w) * fe_{ij}$$

where w is a weighting coefficient introduced to respect the different levels of significance (factors computed during a period P are much less significant than factors consolidated by several periods results).

Consolidated factors are stored in the persistent *consolidated linking factors matrix MC*. As ME, matrix MC is a triangular sparse matrix. Each cell MC(i,j) is a composite structure containing: (i) the consolidated linking factor fc_{ij} for the link (o_i,o_j), (ii) the date $udate_{ij}$[5], of the last updating of fc_{ij}, (iii) a flag $cflag_{ij}$ indicating if o_i and o_j have already been clustered together, this flag is managed by the reclustering procedure (see section 4.1.2 for details).

Elimination of obsolete consolidated factors

Updates of consolidated linking factors are dated in order to periodically eliminate links that have not been detected for a long time.

```
for i --> 2 to n, for j --> 1 to n-1,
do
 if fe_ij ≠ 0
 then
     fc_ij(new) = w*fc_ij(old) + (1 - w)*fe_ij
     udate_ij=Pcurrent
     if (fc_ij > T_fc and cflag_ij = 0)
     then
         n_d= n_d+ 1
         clusdemand(n_d)=(o_i,o_j)
     endif
 endif
enddo
```

Fig. 3. Consolidation phase

Definition: Consolidated linking factors fc_{ij} are considered as obsolete after a time T_V following the last link (o_i,o_j) detection. Periodically, consolidated links are eliminated as follows:

$$fc_{ij} = 0 \text{ if } P_{current} - udate_{ij} > n_p \text{ (modulo p)},$$

where $P_{current}$ is the current period P number, n_p is the number of periods P included

---

[5] The date is stored as a period number (modulo p). It is equal to the number of the last period in which a significant link has been detected between o_i and o_j.

in T_V, and p is the modulo value used to increment $P_{current}$.

Elimination of consolidated linking factors does not imply the reclustering of o_i and o_j. It only means that o_i and o_j have not been used together for a long time. So, if they were already clustered, they are now considered as free to be moved later closed to other objects, if it becomes necessary.

Determination and memorisation of reclustering demands
Consolidated linking factors are considered as pertinent indicators for clustering. Figure 3 presents how reorganisation demands are registered, in the *consolidation phase*, after computation of elementary factors at the end of each period P.

Definition: The weight of a consolidated linking factor fc_{ij} is high enough to justify clustering of o_i and o_j, as soon as: $fc_{ij} > T_{fc}$, where T_{fc} is the threshold value under which clustering benefit do not exceed reclustering overhead.

Determination of the T_{fc} value is essential (see section 4.2). It directly influences the DB stability but also the storage space organisation pertinency.

Triggered reclustering works
At the end of a period P, n_d reclustering demands are registered. These demands are grouped in reclustering units in order to distribute reclustering on separate work-times. As soon as the transactional rate is low enough (the I/O rate is periodically scanned), one reclustering unit is processed at a time (details are given in section 4.1).

Buffering
The purpose of buffering is to cache likely-to-be-used objects in main memory in order to avoid I/O operation every time an object is required. Buffering schemes cover two different issues: (i) page replacement policies, (ii) buffer allocation [10]. We are only interested in the first point, which is closely linked to clustering policies. LRU-like policies are usually recognised as good policies because they are simple and efficient. However, when buffers are full, such replacement schemes evict the least-recently-used page, without regarding if contained objects are related or not with currently-used objects. So, some likely-to-be-used objects may be swapped on disk.

We said earlier that having grouping units larger than individual pages is an important issue for clustering. We think that clustering among several pages is improved by: (i) loading partial or entire clusters in memory, (ii) keeping all pages of likely-to-be-used clusters in memory. So, the buffering solution included in the STD clustering proposal consists in: (j) having the possibility of running DB applications in a 'smart'[6] prefetching mode; entire clusters or moving cluster-windows are prefetched in advance, (jj) modifying LRU-like policies, in order to date cluster (or cluster-window) uses instead of page uses, (jjj) electing for replacement the least-recently-used cluster (or the least-recently-used cluster-window) instead of the least-recently-used page. This buffering scheme may be named LRUc (Least-Recently-used cluster).

The 'smart' STD prefetching policy is based on the moving cluster-window concept presented in the following definition and described in Figure 4.

---

[6] 'smart' means that depth of prefetching is closely controlled. Aggressive prefetching policies may not be truly effective since they may prefetch objects not actually needed.

Definition: A *moving cluster-window* is a limited view of the physical stored cluster (composed of s pages), that may be the unit of prefetching. It is a sub-graph of the object graph stored in the cluster, centered to the initial required object. Instead of loading the individual page containing the accessed object o_i, the most strongly related objects, stored around o_i in the cluster, will be prefetched.

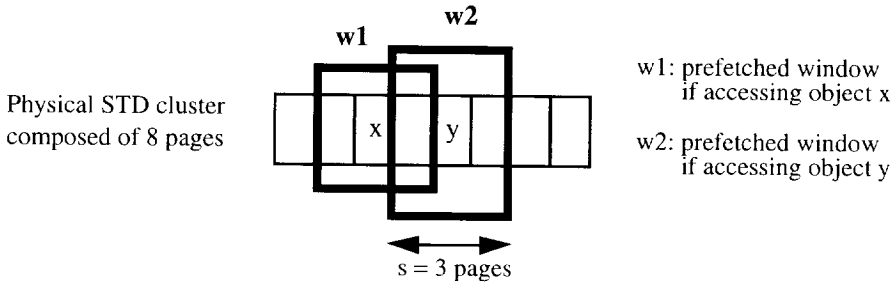


Fig. 4. The concept of moving cluster-window.

4 Algorithmic Details

4.1 Reclustering Technique

We have seen how our statistic-based model allows detection of strong inter-object links, weighted by their access frequencies. Definition and computation of linking factors allows determination of sub-graphs G_k containing strongly related objects.

Definition: *Clusters* are variable-length chunks of storage space, containing strongly related objects o_i. Each cluster is a set of contiguous disk pages, large enough to receive the related o_i and having a unique identifier (clusterID). The system knows at any time the cluster to which any object o_i belongs.

Objective: To have each sub-graph G_k stored within a cluster. Inside each cluster, the distance on disk between two objects will be proportional to the importance of the corresponding linking factor.

Construction of Reclustering Units

Among the n_d reclustering demands registered at the end of period P (each concerning two objects o_i and o_j to cluster), several records may be related to one another, by having one of objects o_i or o_j in common. Related records must be processed at the same time, since all of the concerned objects will be stored in the same cluster.

Definition: A reclustering unit U_k is a sub-graph of related objects, for which reclustering demand(s) have been recorded and which have to be stored in a same cluster. The set of reclustering units is constructed from the reclustering file after the consolidation phase. All objects of a reclustering unit will be processed at a same time. Figure 5 gives an example of the construction of reclustering units.

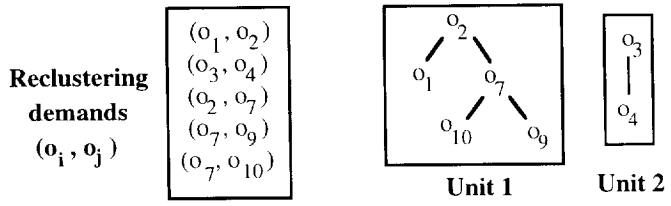


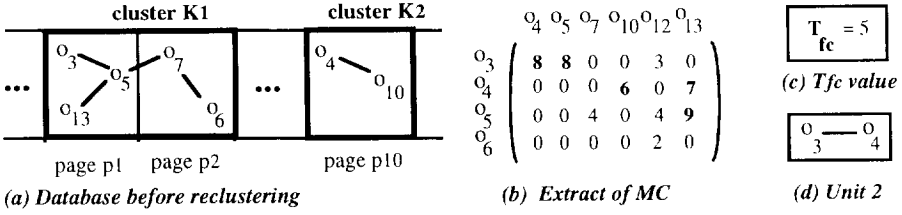
Fig. 5. Construction of reclustering units.

Reclustering Procedure

A reclustering unit U_k is a graph of related objects (minimum two objects), corresponding to new significant attractive forces. Processing a given unit consists in the following phases:

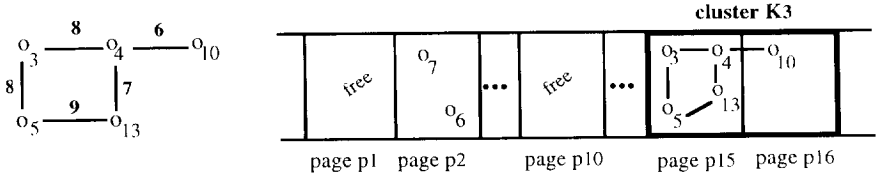
(i) Determining in the MC matrix all objects connected to graph U_k, in order to determine the new graph G_k of consolidated links corresponding to the current reclustering unit.
(ii) Creating the object clustering sequence corresponding to graph G_k, by means of the algorithm presented in section 4.1.3.
(iii) Distributing the obtained object sequence among one or several pages.
(iv) Storing the object sequence into a new cluster (another allocated disk space composed of contiguous pages).
(v) Freeing places occupied previously by moved objects.
(vi) Memorising the new belonging cluster for each reclustered object and updating in the MC matrix flags indicating whether two objects belong to the same cluster or not.

Figure 6 presents an example of such processing, considering: (a) database space organisation before clustering, (b) the current value of consolidated factors in MC, (c) the T_{fc} threshold value, and (d) the unit to process: Unit 2 of Figure 5. (e) and (f) presents how this unit is processed through phases (i) to (v).



(a) Database before reclustering

(b) Extract of MC



(e) U2 extended to graph G2, considering MC - Phase (i)

(f) Database after reclustering - Phase (ii) to (v)

Fig. 6. Processing of a reclustering unit

We can see on the example processed in Figure 6 that if links become obsolete (set to null, as the link (o_6, o_7)) or no longer pertinent (under the T_{fc} value, as the link (o_5,o_7)), they are not considered in the new cluster construction. Then, some objects (o_6 and o_7 in Figure 6) may remain in the same page, but no longer belong to any cluster.

Determination of the Stored Object Sequence

The algorithm used to order the object sequence is inspired by the Kruskal algorithm (construction of a minimum-cost spanning tree from a general graph) and by the solution proposed in [8] to suggest a multi-level strategy able to order at the same time different types of relationships between objects. Of course, we do not adapt this algorithm for the same purpose, since we are not interested in distinguishing the different types of links. Besides, we are not interested in directed links as in [8]. Contrary to the Kruskal algorithm, our proposed scheme must of course consider maximum-costs in order to obtain distance on disk between two related objects proportional to their attractive force. The idea of super-node introduced in [8] is kept but has been slightly modified.

Definition: A *super-node* in an object graph is a set of related objects, constructed progressively, by grouping objects related by the most relevant arc, relatively to the following rules.

Rule R1: The list of objects in a super-node S (for example S= {o_2, o_5, o_4, o_9}), can only be considered in the direct order (o_2, o_5, o_4, o_9) or in the reverse order (o_9, o_4, o_5, o_2), so that the distance between objects remains continuously unchanged.

```
while graph Gk is not reduced to a single super-node
do
  Identify the most relevant weight W among every weighted arcs
  Construct list L containing arcs weighted by W
  for each arc ai belonging to L
      do
          Group and arrange in a same super-node nodes related by arc ai
enddo
  for each super-node S connected to the same node by several arcs
  do
      Determine and keep only the most weighted arc.
  enddo
enddo
The clustering sequence is given by the final super-node.
```

Fig. 7. Ordering the related objects.

Rule R2: When two super-nodes S1, S2 are grouped in a super-node S, we must identify, in the initial graph, which arc (o_i,oj) is responsible for the grouping. Then,

in order to minimise the distance between o_i and o_j (in the super-node S and finally on disk), we may have: (i) S constructed either with {S1, S2} or with {S2, S1}, (ii) objects inside S1 and S2 maintained in the same order or reversed.

Rule 3: When constructing a super-node S, any arc (weighted to w) existing previously between an object of S and another node (simple-node or super-node) N_i, becomes an arc between S and N_i (weighted to w).

Figure 7 presents our proposed ordering scheme. An example is processed in Figure 8.

For each object o_i in the obtained clustering sequence, the distance on disk between o_i and each of its related object is proportional to the weight of the corresponding link. The clustering sequence obtained in the example processed in Figure 8 is: {o_5, o_1, o_4, o_3, o_7, o_8, o_{10}, o_2, o_{18}}.

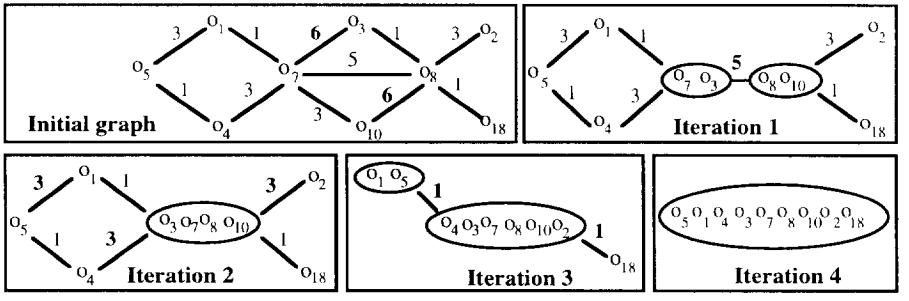


Fig. 8. Example of an obtained clustering sequence.

The problem resulting from shared objects in other clustering strategies is solved here in a deterministic way, since we choose systematically the most weighted link.

4.2 Tuning Capabilities

A set of tuning parameters, described in Figure 9, has been introduced in order to control mainly system reactivity and statistics pertinency. Until now, their determination is under the DB administrator responsibility but we are currently processing extensive experiments in order to fix some of the proposed parameters to optimal values or range of values. In particular, determination of the thresholds value is essential since they directly influence the DB stability as well as the storage space organisation pertinency.

Activation/deactivation options (Figure 10) are proposed to take into consideration particular behaviors of specific databases or transactions. For example, for a special transaction used one time in a year and using intensively the database in a specific way, programmers may choose intentionally not to update STD statistics. Besides, for particular databases inside which objects life are very short, the DB administrator may choose intentionally to keep initial default placement of objects (NoRecluster and NoStatDB options).

n Maximum entries in the Observation Matrix MO during a period P.
n_p Number of observation periods after which a consolidated factor fc_{ij} is obsolete if the link (oi,oj) has not been detected.
p The number of the current observation period is computed modulo p.
T_{fa} Threshold value (in number of object accesses during a period P) under which the number of accesses to individual objects is too small to be considered in elementary linking factors computation
T_{fe} Threshold value under which elementary linking factors are not considered to update consolidated factors.
T_{fc} Threshold value under which a consolidated linking factor is not considered as significant (toactivate reclustering or to be considered in new cluster construction). Reclustering under this value, would be globally negative.
w Weighting coefficient introduced to minimise significance of elementary observations relative to consolidated observations.
s Size of the current moving cluster-window (number of pages prefetched in case of accessing a clustered object).

Fig. 9. Tuning parameters.

For each database:
- *Prefetching* NoPrefetchDB/ PrefetchClusterDB / PrefetchWindowDB
- *Dynamicity* NoRecluster / Recluster
- *Statistics* NoStatDB / StatDB

For each transaction:
- *Prefetching* NoPrefetch / PrefetchCluster / PrefetchWindow
- *Statistics* NoStat / Stat

Fig. 10. STD options.

5 Tests with Texas

5.1 Main Characteristics of Texas

Texas is a persistent storage system for C++, designed and developed at the University of Texas, Austin [18]. Our choice of Texas for testing STD technique comes essentially from the fact that the source is free available and can be easily modified. Texas runs under Unix and is a virtual memory mapped system. The source is coded in C++ and the data formats in memory are those of C++. Persistent objects are stored on disk pages in the same format as they are in memory. Texas uses physical OID coded on four octets. When a disk page is loaded in memory, all the disk adresses toward referenced objects must be converted to memory adresses. The conversion, usually called swizzling, is made in Texas by reserving a memory page for each referenced disk page. But a referenced disk page is effectively loaded in memory only the first time it is needed (when a fault page is detected).

During an execution, a database is represented by an object called Pstore. This object is used to memorize the main information concerning the management of the database and more particularly: (i) a pointer to the zone containing the root objects (each root object can be accessed through a name), (ii) a pointer to the zone of available pages, (iii) a pointer to the swizzling table containing the links beetween each disk page and its corresponding memory page.

5.2 Implementation of the STD Technique Through Texas

The implementation has necessitated a complete comprehension of the internal working of the system and of the code organization. It has been made thanks to many sollicitations of the authors via e-mail. It is not possible to describe here in details the modifications and the adjunctions made on the source. We only give the main principles. Implementation is achieved by tying data structures as attributed of the Pstore object and algorithms as methods.

The new attributes added to Pstore are the following:

- *observation* for the implementation of the observation matrices

- *consolidation* for the implementation of the consolidation matrice

- *clustering* for the implementation of the clustering

- *objectlist* for the implementation of the temporary access sequence to objects during a transaction.

The construction of a cluster unit is made according to the following stages:

i) Each OID of the unit is used to load the referenced page and to capture the size of the corresponding object database.

ii) The cluster is created at the end of the file. The objects of the unit are copied in the order of the cluster sequence. The size of each object is used to control that the current page is not full. Otherwise a new page is allocated to the cluster. A list, called *movedobjectlist*, is used to store the old and the new OID for each object.

iii) To upade pointers to moved objects, each page is loaded in memory. Each time an adress contains an old OID in the *movedobjectlist*, it is replaced by the new one. The replacement is made by the swizzling module of Texas which has been modified to take into account the *movedobjectlist*.

5.3 The Benchmark

We have adapted the well known OO1 Catell benchmark [4] to test STD technique. It is based on (i) a database schema containing two types of objects (ii) recommandations to generate an instance of a database and (iii) several typical manipulations.

The database schema is given on figure 11. The size of the two attributes X,Y of the type *Part* can be modified if necessary to vary the tests. Each *Part* object is connected to three other *Part* objects. Connections are represented through the type *Connection* : there exists one *Connection* object for each pair of connected *Part* objects.

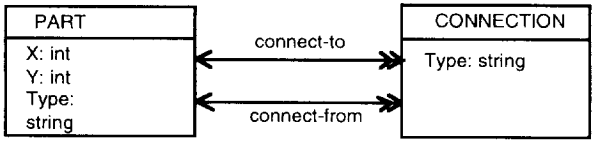


Fig. 11. The database schema for the benchmark

An instance of a database is generated with a number of *Part* objects equal to *partnb*. 90% of the connections are randomly established by respecting a principle of locality. More precisely a *Part* object is randomly connected with three *Part* objects having indices in the interval max(i-refzone*partnb) and min(i+refzone*partnb) where refzone is a ratio fixing the amplitude of the locality (max and min are used to impose the interval to be included in [1, partnb]). The other 10% of the connections are established with any object in the database.

Among the different manipulations suggested by the benchmark, we will use only the traversal. It consists in exploring the different objects which can be reached through the connections from a root object generated with the database. Since cycles are always possible, the traversal is not a finite procedure and must be stopped at a predetermined level from the root. The number of visited objects is a simple function of the level. A traversal with 4 levels reads 160 objects. A same object can be found many times during the traversal. This situation does not induce any problem for STD technique which consists in following the effective use of objects.

5.4 Tests and Results

STD technique has been designed to take into account different applications on the database and it is important to simulate the concept of application. In our tests an application is charaterized by

-a traversal (i.e. a root and a level)

-the number of times *trnb* the traversal is made.

Different roots have been generated with the database to simulate three different applications numbered 1 to 3. The efficiency of STD is measured by the number of page faults, before the clustering and after the clustering.

We now give the most significant results we have obtained.

The first experiment (figure 12) concerns the influence of the refzone parameter with a unique application. The clustering power of STD is very effective even for narrow localities (for his benchmark, Cattel considers that a value of 0.01 for refzone is realistic).

partnb	5000
Tfa	1
Tfc	1

refzone	before	after	ratio
0.001	17	4	4.2
0.01	38	5	7.6
0.05	55	5	11.0
0.1	57	5	11.4
0.2	66	5	14.2

Fig. 12. Number of page faults before and after reclustering depending on refzone

The second experiment (figure 13) has the objective to situate the influence of the T_{fa} parameter with three different applications. The average ratio has been obtained by weighting the individual ratios with the weight of each application. The efficiency of STD is maximal for T_{fa}=4. For T_{fa}=4, application 3 has no longer any influence on the clustering. For T_{fa}=8, only application 1 influences the clustering. When T_{fa}=16, a unique cluster of seven objects is constructed (this cluster results from cycles in the connections between *Part* objects). This run ensures that T_{fa} has really the expected effect. These different threshold values for T_{fa} depend directly on the values of *trnb* (traversal number) for the three applications. *trnb* represents approximately the average number of object uses for an application.

partnb	5000
refzone	0.01
Tfc	1

application			before	after				
#	trnb	level		Tfa=1	Tfa=4	Tfa=8	Tfa=12	Tfa=16
1	15	4	38	7	6	5	5	38
2	5	4	29	7	6	30	30	30
3	2	4	40	7	36	40	40	41
average ratio				5.0	4.1	2.6	2.6	1.0

Fig. 13. Number of page faults before and after reclustering depending on T_{fa}

For this experiment, the three applications share the use of many objects. Other experiments have shown that the number of shared objects have little influence on the results. We can say that STD technique clusters the objects in the best way, independantly the existence of shared objects.

The third experiment (figure 14) situates the influence of the T_{fc} parameter in similar conditions. The efficiency of clustering is maximal for T_{fc}=1 and diminishes slightly when T_{fc} varies from 1 to 50. The diminution becomes very important when T_{fc} reaches number 50. When T_{fc} is greater than 50 we observe a degradation due to the fact that only a few objects are clustered in supplementary pages at the end of the file. The database remains strictly unchanged when T_{fc}=100.

partnb	5000
refzone	0.01
Tfa	1

application			before	after			
#	trnb	level		Tfc=1	Tfc=30	Tfc=35	Tfc=51
1	15	4	38	7	8	9	40
2	5	4	29	7	7	7	31
3	2	4	40	7	7	9	43
average ratio				5.0	4.7	4.2	0.9

Fig. 14. Number of page faults before and after reclustering depending on T_{fc}

The last experiment reported here (figure 15), concerns some measures about the execution time (i) for the traversal before and after the clustering, (ii) for the consolidation phase and (iii) for the clustering. The conditions are those of figure 13. The before traversal includes the observation time which is very small (a fraction of a second) compared with the traversal time. First we can observe that the ratio between the traversal times before and after has not exactly the same value as the ratio between the number of page faults before and after. This comes from the fact that I/O operations resulting from a page fault have variable execution time. When T_{fa} increases, the execution times for the consolidation phase and for the clustering decrease since the collecting activity diminishes and the number of clustered object becomes lower.

partnb	5000
refzone	0.01
Tfc	1

Execution time in seconds	Tfa=1	Tfa=4	Tfa=8	Tfa=16
Before traversal	10	9	9	9
After traversal	3	4	5	8
Consolidation phase	4	3	2	1
Cluster construction	13	10	9	6
Number of objects in clusters	364	274	176	65

Fig. 15. Some execution times for different values of T_{fa}

(runs on SUN SPARC/ELC under SUN OS V4.3.1)

The conclusion that we can draw from these results can be reasonably optimistic. First, it appears that T_{fa} and T_{fc} control effectively the strategy and are not difficult to adjust. Second, the balance between the gain and the overhead of STD technique when T_{fa}=1 and T_{fc}=1 can be established as follows. Including the consolidation time, there remains an excess of 10-(3+4)=3 seconds between the time before and after. Thus, after only five runs of the three applications, the total excess overcomes the clustering time and the reclustering technique becomes globally beneficial. In fact, the consolidation time is relatively smaller than the value taken here since it must be considered on a total period. So, for this benchmark, the balance is really much more favorable. Hovewer, this benchmark does not integrate all the aspects of the STD technique and this preliminary conclusion must be confirmed by more extensive tests.

6. Conclusion

The STD strategy described in this paper is an attempt to incorporate in a unique framework several proposals able to improve performance in object database systems. The main idea is to take advantage of effective use of inter-object relationships to manage clustering of objects on disk and cluster-based buffering. We first have investigated problems associated with most existing clustering schemes in order to justify our precise motivations. Several points are rather innovating in the STD clustering proposal: (i) Clustering specifications are not made at a class level, so that distinct objects of a same class may be clustered in a different way. (ii) Clustering is

made upon observation of effective use of links between objects rather than on a priori user hints or DB schema analysis. (iii) Considering that any relationship, leading to simultaneous use of objects, is expressed in databases by physical pointer traversals, the STD concept of link allows detection of any type of relationships (structural relationships defined in the schema, but also implicit links installed by programmers or by the DBMS itself). Clustering sequences take into consideration the different types of link, but do not have to distinguish them. Distance between related objects on disk only respects attractive force between objects. (iv) Automatic dynamic reclustering, free of user activations, is proposed. It deals implicitly with modifications of programs code and with database schema evolution. (v) Even if the solution is based on statistics management, efforts have been made to reduce storage costs, to filter and keep only pertinent information, and to brake-up reorganisations when confronted to continuous statistics changes. A strict control of reclustering avoids continuous reorganisations and excessive overhead. (vi) STD clustering scheme allows a deterministic solution to the problem of shared objects. (vii) Another idea is to accept large clusters among several contiguous disk pages. This permits to minimise disk head moves when large sets of related objects are accessed. Besides, the buffering policy has been adapted to take advantage of such clusters. In particular, we proposed a 'smart' prefetching policy and suggested to replace LRU policies by cluster-based replacement policies. Our approach remains efficient thanks to the idea of moving cluster-window centered on the page of the accessed object. (viii) The clustering mechanism does not require any user or administrator hints but remains user-controlled, with the presence of tuning parameters and activation/deactivation options.

A partial implementation of the STD technique has been made using Texas. It permits to experiment the clustering efficiency with a benchmark adapted from OO1. It appears that in nominal conditions, the number of fault pages is divided by five after reclustering. Overhead of STD technique is balanced by gain in access time only after five uses of the application.

This strategy may be convenient for any OODBMS, but requires some attention for its implementation. It is necessary to add an observation module inside the object manager in order to follow the dereferencements of persistent pointers. Besides, it implies management of different data structures (matrices, graphs) to store observations and determine the clusters. Many solutions do exist. Adaptations can be made to minimise occupied memory space and processing overhead. Besides, implementing dynamic reclustering in systems having physical object identifiers requires specific solutions. Problems encountered are the same as those encountered with DB schema evolution. Physical OID implies specific techniques to maintain DB integrity when moving objects (forwarding pointers, as in O_2, for example). Moreover, the STD clustering scheme implies specific solutions for storage space management. In particular, technical solutions must be developed inside DBMS running on Unix systems, to be able to allocate contiguous disk pages. The object manager must also be able to manage and reuse free disk space and to store instances of different classes in a same disk page.

We think that the STD proposal relies on pertinent motivations, considering the limited solutions available today in existing systems and the lack of advanced implemented proposals. However, if this approach constitutes an issue to enhance the performance of object systems, we know that its great tuning capabilities have to be reduced to a minimum set of relevant parameters. Experiments on Texas permit us to

evaluate the behavior of T_{fa} and T_{fc}. They confirm our intuitive expectations. We think that these parameters must be fixed at values depending on the use of the database. For example, a good compromise is to fix T_{fa} at the value of the average number of object uses. We lead actually other experiments to validate the dynamical aspect of STD and to determine plausible value for T_{fe}.

Bibliography

1. V. Benzaken, C. Delobel, *"Enhancing performance in a persistent object store: clustering strategies in O_2"*, 4th International Workshop on Persistent Object Systems, September 1990, pp. 403-412.

2. V. Benzaken, *"An evaluation model for clustering strategies in the O_2 Object-Oriented Database System"*, Third International Conference on Database Theory, December 1990, pp. 126-140.

3. F. Bullat, "Regroupement physique d'objets dans les bases de données", to appear in the I.S.I. Journal, 'Ingénierie des Systèmes d'Information', Vol. 2, no. 4, September 1995.

4. R.G.G. Catell, *"An Engineering Database Benchmark"*, in 'The Benchmark Handbook for Database and Transaction Processing Systems", Morgan Kaufman Publishers, 1991, pp. 247-281.

5. E.E. Chang, *"Effective Clustering and Buffering in an Object-Oriented DBMS"*, Ph.D. Dissertation in Computer Science, Report no. UCB/CSD 89/515, University of California, Berkely, June 1989.

6. E.E. Chang, R.H. Katz, *"Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS"*, ACM SIGMOD Conference, New York, 1989, pp. 348-357.

7. E.E. Chang, R.H. Katz, *"Inheritance in Computer-Aided Design Databases: Semantics and Implementation Issues"*, CAD, Vol. 22, no. 8, October 1990, pp. 489-499.

8. J.B. Cheng, A.R. Hurson, *"Effective Clustering of Complex Objects in Object-Oriented Databases"*, ACM SIGMOD Conference, New York, 1991, pp. 22-31.

9. J.B. Cheng, A.R. Hurson, *"On the Performance Issues of Object-Based Buffering"*, ACM SIGMOD Conference, New York, 1991, pp. 22-31.

10. W. Effelsberg, T. Haerder, *"Principles of Database Buffer Management"*, ACM Transactions on Database Systems, Vol. 9, no. 4, December 1984, pp. 560-595.

11. O. Gruber, L.Amsaleg, *"Object grouping in EOS"*, Workshop on Distributed Object Management, University of Alberta, August 1992, pp. 117-131.

12. M. Hornick, S. Zdonick, *"A shared Segmented Memory System for an Object-Oriented Database"*, ACM Transactions on Office Information Systems, Vol. 5, no. 1, January 1987, pp. 70-95.

13. S.E. Hudson, R. King, *"Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System"*, ACM Transactions on Database Systems, Vol. 14, no. 3, September 1989, pp. 291-321.

14. W. Kim, J. Banerjee, H-T. Chou, J. F. Garza and D. Woelk, *"Composite Object Support in an Object-Oriented Database System"*, International Conference on OOPSLA, Orlando (Florida), October 4-8 1987, In proceedings of ACM SIGMOD Conference, 1987, pp. 118-125.

15. C. Lamb, G. Landis, J. Orenstein and D. Weinreb *"The ObjectStore Database System"*, Communications of the ACM, Vol. 34, no. 10, October 1991, pp. 50-63.

16. ONTOLOGIC Cie, *"ONTOS Client Library Reference Manual"*, December 1990.

17. Servio Corporation, *"GemStone V. 3.2 Reference Manual"*, 1992.

18. V. Singhal, S.V. Kakkad, P.R. Wilson, *"Texas: An Efficient, Portable Persistent Store"*, 5th International Workshop on Persistent Object Systems, San Miniato, Italy, September 1992.

19. J.W. Stamos, *"Static Grouping of small objects to Enhance Performance of a Paged Virtual Memory"*, ACM Transactions on Computer Systems,Vol. 2, no. 2, May 1984, pp. 155-180.

20. E.M. Tsangaris, J.F. Naughton,*"A Stochastic Approach for Clustering in Object Bases"*, ACM SIGMOD Conference, Denver, May 1991, pp. 12-21.

21. E.M. Tsangaris,*"Principles of Static Clustering for Object-oriented Databases"*, Technical Report no. 1104, University of Wisconsin-Madison, August 1992.

Acknowledgements: The authors wish to express their deep gratitude to S. V. KAKKAD (OOPS Research Group of Computer Science Deparment, University of Texas) for its collaboration through e-mail about Texas.