CLUSTERING TECHNIQUES FOR OBJECT-ORIENTED DATABASE SYSTEMS

Sophie Chabridon Jen-Chyi Liao Yichen Ma Le Gruenwald

School of Computer Science
The University of Oklahoma
200 Felgar Street, Room 114 EL
Norman, Ok 73019-0631

Abstract

In an Object-Oriented Database system (OODB), the interrelationships among objects and inheritance semantics can be used by clustering techniques to improve the performance in terms of system response time. In this paper, we conduct an analysis to compare three clustering strategies, Cactis, ORION, and CK, in terms of space and time overhead. We also examine the level, page or segment, at which clustering should take place. The dynamic clustering technique CK is found to be best in exploiting the structural relationships between objects and inheritance semantics to identify an efficient storage scheme. However it creates high overhead and is only best suited for applications in which the Read/Write ratio is high. To remove this limitation, we present how segment clustering, instead of page clustering, could reduce the number of cases where a page split is necessary.

1. Introduction

Clustering refers to storing related objects close together on secondary storage. This means that whenever one object is accessed from disk and brought into the main memory, all the objects clustered with it are also brought into memory. Subsequent access to any of these objects is then a main memory access and is much faster than a disk access. Clustering is used to minimize the I/O cost of retrieving a set of related objects. Therefore to improve the performance of a database management system (DBMS) in terms of response time, it is essential that an efficient clustering strategy be utilized.

In most conventional DBMS, clustering is usually based on grouping per relation or at a lower granularity. Grouping can be done on the values of an attribute or on a combination of attributes. In object-oriented DBMS, besides the fact that some data belongs to the same class, additional relationships, mainly structural properties of composite objects and inheritance, can exist between data. This means that new clustering algorithms are needed to address the specificity of these systems.

In an OODB, "classes are used as a primary means of grouping objects and searching for objects. In some sense, classes take the place of relations. There is an assumption that there must be system supported mechanisms for grouping objects into collections automatically" [5]. This remark illustrates the problem faced by clustering techniques for OODB in that there are many different ways objects can be clustered.

Four basic clustering options are described in [13].
1. Cluster all objects belonging to the same class in the same segment of disk pages. This method expedites the sequential scanning of the objects of a class. This is a simplistic solution, similar to what is done in conventional databases.
2. Cluster objects belonging to a class hierarchy rooted at a user specified class.
3. Cluster objects and other objects that they recursively reference. In this case, objects may belong to different classes.
4. Combine options 2 and 3, and simultaneously cluster classes on a class hierarchy and a subset of the class attribute graph (or composite hierarchy).

In [6], various clustering strategies are proposed.
1. Composite objects: Clustering based on aggregation relationships can be defined at run-time or within the schema definition.
2. References: Clustering according to relationships

with other objects.
3. Object types: Clustering objects by their type (class). This is useful only if repeated access is expected to objects of just one type.
4. Indexes: Clustering objects by an index on their attributes. While an index is defined on a string attribute, this kind of clustering is good if objects are accessed frequently in order.
5. Custom: "on the fly" clustering. For example, an existing object may be supplied as a parameter to an object creation procedure and the system attempts to create the new object physically close to the existing one.

Objects are usually stored according to one of these rules. However, when rules do not conflict, a multiple clustering can be performed.

Several object-oriented database systems under research or already implemented use clustering to improve their performances. However, most of them are only based on users' hints that must be provided to the clustering algorithm. Gemstone [14] requires the database administrator to specify that certain objects are often used together and so would be clustered on disk. The VBase system [1] allows explicit clustering hints when objects are created. In ONTOS [2], the strategy adopted is to allow the programmer to specify clustering and to provide tools for reclustering when more experience with the application permits better choices to be made.

In this paper, we present and compare three methods for clustering. Our motivation is to consider methods that are significantly different from one another and that do not only rely on users' hints. We so intend to illustrate several solutions for performing clustering in an OODB. We have chosen to analyze the clustering algorithm of the Cactis system [11], the method followed in ORION [12, 3], and the CK clustering algorithm [7, 8]. We consider these three algorithms different enough to be representative of the current research on clustering techniques.

The paper is organized as follows. The three clustering algorithms, Cactis, ORION, and CK, are presented in Sections 2, 3, and 4, respectively. An analytical comparison of the three algorithms is given in Section 5. Section 6 examines the levels, page and segment, at which clustering should take place. Conclusions are provided in Section 7.

2. The Cactis Algorithm

Cactis is an object-oriented, multiuser DBMS developed at the University of Colorado. It was "designed to support applications that require rich data modeling capabilities and the ability to specify functionnaly-defined data" [11]. The clustering algorithm implemented in Cactis is also used by the Zeitgeist system [10]. The algorithm is given in Figure 1.

Repeat
 Choose the most referenced object in the database that has not yet been assigned a block.
 Place this object in a new block.
 Repeat
  Choose the relationship belonging to some object assigned to the block such that:
   1) the relationship is connected to an unassigned object outside the block and,
   2) the total usage count for the relationship is the highest.
  Assign the object attached to this relationship to the block.
 Until the block is full.
Until all objects are assigned blocks.

Figure 1 - Cactis Clustering Agorithm [11].

It is a static algorithm in the sense that it is used periodically when the system is idle to recluster the database. It makes use of usage patterns which are the total number of times each object in the database is accessed and the number of times a relationship between objects is crossed. Basically, clustering is done at the page level, called block in the algorithm, and which is the unit of a disk I/O. The algorithm tries to "place objects that are frequently referenced together in the same block" [11]. Proper clustering can result in improvements of about 60%.

Some remarks can be made on this algorithm. It does not require users hints. Instead it is based on information determined by the usage of the database. This can be an advantage from the point of view that the algorithm is user-independent and no arbitrary choice has to be made by the user. Moreover, this method implies that the first time the system is running, the database is not clustered since no usage information is yet available. Then the database is reorganized when the system is idle and the first

clustering takes place after the database has been utilized during a period of time and statistics information is available. The pertinence of this information has a great impact on the system. The algorithm performs better if the database has reached a steady state. But this is compromised by the DB evolution and all the possible changes that can take place. This algorithm gives good results if "query streams exhibit commonalities over time" [11].

3. Clustering in ORION

ORION is a prototype database system built in the MCC Database program [3, 12]. It was designed for applications in Artificial Intelligence, Computer-Aided Design and Manufacturing (CAD/CAM), and Office of Information Systems (OIS).

In [3], no clustering algorithm is given, only the method followed is described. It considers a composite object as a unit for clustering. By default, all the instances of a class are placed in the same storage segment. ORION automatically allocates a separate segment for each class. However, for clustering composite objects where the component objects belong to different classes, "user assistance is required to determine which classes should share the segment" [3]. The user can issue a Cluster message containing a "ListOfClassNames" argument specifying the classes that are to be placed in the same segment. The authors propose to store a composite object in a sequence of pages linked in the manner of a B-Tree. The authors focus on the storage of composite objects. However, their method allows clustering of a class hierarchy as well. The user is free to specify any kind of grouping by listing the classes to use for clustering.

One advantage of this method is its simplicity. Users' hints are used to determine how to cluster objects together. No cost model is defined and no overhead is implied to determine what is the optimal storage unit for an object, as it is done in the other two algorithms we studied. This simplicity can also turn to be a limitation of this method. The decision is left to the user whose choice is based uniquely on the static information given by the data model. It is not considered whether a better clustering exists with regard to the usage of the database. As mentioned in [13], the "decision should be done on the basis of an analysis of the expected frequency and cost of different types of access".

Although user hints are given at run-time (the user sends a Cluster message), the clustering is not totally dynamic. Reclustering seems to be performed in only some cases. "ORION does not recluster objects in response to an ExchangePart message" [3], even if, when two dependent objects are exchanged between two parts, they really should exchange storage positions as well. This implementation is preferred for its simplicity. Through the description given in [3], it appears that all the instances of a class will be stored in the same segment. All the instances of a class are supposed to have the same behavior. This limitation is removed by the algorithm we present next.

4. The CK Algorithm

This algorithm, that we call CK from the names of its authors CHANG and KATZ, is detailed in [8]. A simulation study was also performed and the results are presented in [7]. The CK algorithm is defined in the context of CAD/CAM applications. Several new concepts are tightened to this algorithm and are introduced here.

4.1. Structural Relationships

The algorithm makes use of structural relationships which are: versions, configurations and correspondence (or equivalence).

The concept of version is the easiest one to apprehend and is also presented in [4]. An object is viewed as a "molecular object" which has an interface description and an implementation description. They are represented separately in two distinct sets of records. Objects that share the same interface but have different implementations are called versions. They represent various design alternatives.

A very important characteristic of OODB is the presence of composite (complex or nested) objects. This concept is represented through composite/component relationships among objects. A configuration then results from the coupling of versions with composite objects. "A configuration is a composite unit whose components are bound to specific versions" [8].

The third kind of relationships is equivalence. Two

objects are equivalent if they are alternative representations of the same entity.

4.2. Instance-to-Instance Inheritance

Besides structural relationships, inheritance provides also additional semantics. As in object-oriented programming languages, a class/subclass hierarchy can be defined for an OODB based on the IS-A relationship. A subclass inherits the structure (attributes' definitions) and the methods of its ancestor. However, in OODB this form of inheritance, which is called type inheritance, is not enough.

[15] explains the necessity of having instance or value inheritance for CAD/CAM and Artificial Intelligence applications. The "instance inheritance relationship does not only transfer the existence of attributes from one object to another, but moreover the values of these attributes". It is this kind of inheritance which is used in the CK algorithm. It is called instance-to-instance inheritance and its rationale is largely detailed in [8]. It is aimed to provide inheritance of information along any kind of relationship known to the system: type-instance (from a class to its objects), ancestor-descendent (through a class hierarchy), composite-component (through a composite hierarchy) or even among equivalents.

4.3. Presentation of the Algorithm

The CK algorithm's pseudo-code is given in Figure 2. Here clustering is done at the page level.

"Instance-to-instance inheritance introduces more complexity because it allows attributes to be selectively inherited at run-time" [8]. This run-time flexibility requires a sophisticated approach for clustering.The algorithm is based on inter-objects access frequencies given by the user at data type creation time. A frequency has to be specified for each kind of structural relationship, e.g. 20% of access along version relationships, 75% along configuration relationships and 5% along equivalences. In this example, since the frequency of accessing configuration relationships is higher than others, a new instance will most probably be placed in the same page as its composite objects.

When a new object is created, the clustering algorithm determines an initial placement based on which relationship is most frequently traversed. Then a set of cost formulas is used to choose between implementation by copy or by reference for each inherited attribute. The combination of the relationship traversal frequencies and the inheritance traversal costs identifies the best candidate page. Then two cases can happen, whether the page has sufficient space or not. In the latter case, the algorithm can either choose the next best candidate page which has space or split this page if the access cost resulting from the split is an improvement over the placement in the next best page.

The idea of using a page splitting is a very important contribution of the CK algorithm. Page splitting is realized by a greedy algorithm which partitions the nodes of the inheritance-dependency graph into two subsets that can fit into one page individually. This algorithm is not optimal but it has the advantage to be linear in the number of edges of the graph, whereas an exact graph-partitioning algorithm would be NP-complete. The CK algorithm has been evaluated by simulation in [7]. Run-time clustering always improves the overall system response time. Moreover, when both the structure density and the Read/Write ratio is high, the improvement can reach 200%.

5. Comparison of the Clustering Approaches

5.1. General Characteristics

Figure 3 summarizes the characteristics of the three approaches we have just presented. In ORION, the clustering unit is a segment, whereas in the other two, clustering takes place at the page level. Although it could bring some improvements, in all the three methods, no other level of clustering is explored. We discuss this point in more detail in the next section.

The CK algorithm and the ORION clustering are dynamic and operate at run-time, whereas the Cactis algorithm is static. In [8], it is claimed that "static clustering is not effective for applications which require high availability". So the method used in Cactis, where the database is reorganized periodically, would not give good performance in such domains. However, dynamic clustering can create too much overhead when the system becomes heavily loaded. For this reason, the users of the CK algorithm are allowed to enable/disable clustering based on the characteristics of the operations and data of an application.

```
PROCEDURE Cluster_Object(target_object)
BEGIN
1. /* Step1: Get initial information */
  cluster_policy:= get_policy();
  copy_set:= get_by_copy_set();
  ref_set:= get_by_ref_set();
5. inh_page_set:= get_all_inh_page();
  struct_page_set:= get_all_struct_page();
  page_set:-= inh_page_set + struct_page_set;

/*Step2: calculate ref_set lookup cost for each page */
  FOR p IN page_set
    FOR r IN ref_set
10.   IF r not_in p
      Begin
        weight(p):= 1/(prob(p,struct_rel));
        ref_lookUp(p):= rel_lookUp(p) + weight(p);
      End;
/*Step3: calculate copy_set lookup and storage cost for each page */
15. FOR c IN copy_set
      FOR p IN page_set
        IF c not_in p
          Begin
            weight(p):= 1/(prob(p,struct_rel));
20.         copy_storage(p):= copy_storage(p) + size_of(c);
            copy_lookUp(p):= copy_lookUp(p) + weight(p);
          End;
/*Step4: Calculate total cost of every page. If by_copy attributes are
 implemented by reference, the total cost of storing target object
 in page p is represented by Total_cost(p,1). Otherwise, the cost
 is given by Total_cost(p,2)     */
  FOR p in page_set
    Total_cost(p,1):= ref_lookUp(p) * Lookup_cost +
      Copy_lookUp(p) * Lookup_cost;
25. Total_cost(p,2):= ref_lookUp(p) * Lookup_cost +
      Copy_storage(p) * Storage_cost;

/*Step5: pick up best candidate page and try to insert the object */
candidate_page:= minimum(Total_cost);
  IF (cluster_policy = no_split)
    WHILE (not_fit(candidate_page))
    candidate_page:= Next_min(total_cost);

30. IF ((cluster_policy = page_split) And
        (not_fit(candidate_page))
        Split_page(candidate_page);

  END;
```

Figure 2. Pseudo Code for the CK Clustering Algorithm [8]

Approach / Criterin	Cactis	ORION	CK algorithm
Level	Page (block)	Segment	Page
Nature	Static	Dynamic	Dynamic
Require users' hints	No	Yes	Optional
Based on	Actual usage of objects and relationships	Users' hints uniquely	User-specified access frequencies along structural relationships

Figure 3. Three Clustering Strategies

As we indicated earlier, the method used in ORION is to let the user define how clustering should be performed, based only on the knowledge given by the data model. Moreover, all the instances of a class are stored within the same segment and are supposed to have the same behavior. This is not as flexible as what is done in the Cactis system or in the CK algorithm, where each object can be placed in a different storage unit, with regard to relationships between objects rather than between classes. Finally, the performance of the ORION clustering is very difficult to predict, since it depends entirely on the hints given by the user.

We now concentrate on a more detailed comparison between the Cactis and the CK algorithms. First we consider the space requirements of the two algorithms and then their time complexity.

5.2. Space Requirements

The Cactis clustering algorithm must maintain two counters when the database is in use. One counter keeps track of the number of times each object is accessed. The size of this counter is not mentioned, but we can easily guess that it has to be more than one byte (for values larger than 255). We assume that at least a 2-bytes counter is necessary for each object. The number of objects in the database is dynamic and varies whenever a new object is created or an existing one is deleted. We call N the current number of object contained in the database.

A second counter stores the number of times a relationship between two objects is crossed. The number of relationships is maximum when each existing object is connected to every other object, that is the largest possible number of relationships is $O(N^2)$. However, it would semantically not be correct to connect an object to every other one. Thus, we can expect the actual number R of relationships to be much less than N^2. In total, the space requirements for the algorithm are: 2*N + 2*R bytes.

Additional space requirements exist concerning secondary storage. Before reorganizing the database, data has to be stored on an intermediate storage unit. So the space on disk is required to be at least twice the size of the database for the reorganization to take place. Moreover, by the way clustering is performed by the Cactis system, a lot of space may be wasted on disk. If it happens that an object is too large to fit into a block, a new block is simply allocated to store the large object. The remaining space in the other block is no longer considered. For this reason, a page split algorithm, like what is done in the CK algorithm, could be useful. By organizing a block such that it contains as many as possible highly related objects, could save space on disk and consequently the number of I/Os required to fetch objects.

The CK algorithm makes use of a one-byte counter for each attribute of every object present in the database. A larger number of counters is needed compared to the Cactis algorithm, where a counter is for the entire object. However, each counter requires only one byte. When it is overflowed, it is set to be 255 permanently. We call nA, the average number of attributes per object and N the total number of objects in the system. Then nA * N bytes are needed to maintain the counters.

In order to evaluate and compare the space requirements for the two algorithms, we make here some assumptions. We consider a small database containing 100 objects, N = 100, and assume that on the average each object has 20 attributes, nA = 20. With these figures, the CK clustering algorithm requires nA * N = 20 * 100 = 2000 bytes. The space requirements of the Cactis algorithm are presented on Figure 4. The maximum value is obtained for the largest number of relationships, that is $R = N^2 = 10000$. However, as we saw previously a fully connected database does not make much sense. Consequently, the Cactis algorithm tends to require between 2K and 20K bytes. This is 1 to 10 times more than for the CK algorithm which requires only 2Kbytes.

5.3. Time overhead

The cost of the Cactis algorithm does not have a very important impact, since it is run only when the system is idle. However, in order to determine the complexity of this algorithm this can be evaluated as follows.

First, the most referenced object in the database has to be determined. The search of a maximum over N objects takes O(N) steps. Another solution would be to sort the objects while storing the database on intermediate storage by decreasing number of references. Using an efficient sorting algorithm, this

would take O(NlogN) operations. Then, for the selected object, the most referenced relationship has to be found. If no specific data structure is used to store the relationships, the list of all the existing relationships has to be scanned. This takes O(R) operations. Finally, in the worst case the main loop of the algorithm is performed on each object of the database. This gives an upper bound of O(N*(N + R)) steps for the entire algorithm. If the maximum number of relationships is defined, which is N^2, the complexity of the algorithm is in $O(N^3)$. This makes this algorithm very time consuming. Figure 5 illustrates the time complexity of the Cactis algorithm.

At run-time, whenever an object is accessed or a relationship is traversed, the corresponding counter has to be incremented. This could create a substantial overhead, but depends largely on how low-level computations are implemented.

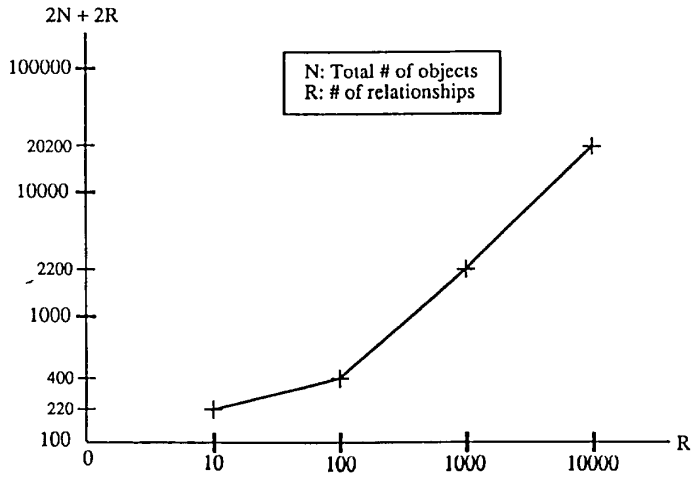

Figure 4. Space Requirements of the Cactis Algorithm

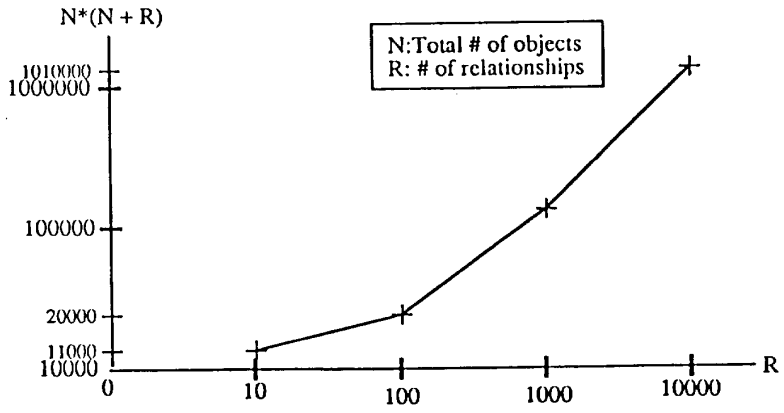

Figure 5. Time Complexity of the Cactis Algorithm

The CK algorithm contains two parts. The first part determines the best candidate page where to place the new object. The second part is the page-splitting algorithm which has a complexity of O(n) where n is the total number of edges between the objects of the candidate page [8].

The time complexity of the first part is detailed here. At the beginning of step 1, a choice between implementation by copy and by reference is made for each inherited attribute of the new object. It is based on the value of the one-byte counter associated with each attribute. If this value is larger than the predefined update threshold corresponding to the type of the attribute, by reference implementation is preferred. Otherwise, the inherited attribute is duplicated. This is explained by the fact that it is more costly to maintain several copies of an attribute it they are updated frequently due to consistency problems. Lines 3 and 4 of the algorithm require the set of the inherited attributes of the object to be scanned. This takes O(nb_attr_inh) steps, where nb_attr_inh is the number of inherited attributes of the new object. We will evaluate this variable by using the probability Pinh which represents the probability for an attribute to be inherited from an ancestor object. Then nb_attr_inh is simply the product nA * Pinh, nA being the average number of attributes per object that we used to compute the space requirements in the previous paragraph.

Then, in the same step 1, on lines 5 and 6, two functions are called. The first one, get_all_inh_page(), returns the set of disk pages containing the source instances for the inherited attributes. We assume that this function requires O(P) steps in order to scan all P pages of the database and check whether the page contains a source object for an inherited attribute of the new object. The second function, get_all_struct_page(), gives the pages which contain objects related to the new object by any kind of structural relationships such as its ancestor and component instances. We assume as previously that this function takes O(P) steps. On line 7, the two sets of pages are then merged to form the set of the candidate pages. We call N_cand the number of candidate pages. We will evaluate this variable by using the probability Pcand which represents the probability for a disk page to be candidate as defined by the clustering algorithm. Then N_cand is given by the product P * Pcand, P being the total number of disk pages necessary to store the database. P can be determined only dynamically, after the clustering algorithm is run and that the database is stored in a particular way on disk. For computations to take place, we consider the worst case where there is only one object per page, that is no way to cluster objects was found. Thus, the value of P that we will use is 100, which is the number of objects in the database.

Steps 2 and 3 each contains two embedded loops. The external loop is done for each candidate page. In step 2, the inner loop is based on the number of inherited attributes implemented by copy (nb_attr_copy) and in step 3 on the number of attributes implemented by reference (nb_attr_ref). We can notice that the sum of nb_attr_copy and nb_attr_ref gives the number of inherited attributes nb_attr_inh. In total, steps 2 and 3 require:
O(N_cand * nb_attr_copy + N_cand * nb_attr_ref) = O(N_cand * nb_attr_inh) operations.

Finally, there is a single loop on the number of candidate pages taking O(N_cand) steps. Consequently, the total number of steps required by the algorithm to determine the best candidate page is:

nb_attr_inh + (2*P) + (N_cand*nb_attr_inh) + N_cand
= (nA*Pinh) + (2*P) + (P*Pcand * nA*Pinh) + (P*Pcand).

With the values we assumed previously (nA=20, P=100) and the fact that Pcand and Pinh are probabilities, an upper bound can be defined for each term in this expression. The first term is < = 20, second term = 200, third term < = 2000, and the fourth term < = 100. The third term is dominant and can be used to represent the complexity of the entire expression. Consequently, the time complexity of the CK algorithm can be stated as: O(P*Pcand * nA*Pinh). This result is illustrated in Figure 6.

When comparing Figures 5 and 6, the Cactis algorithm is found to be much more time consuming than the CK algorithm. It requires no less than 11000 operations and can reach a number of 1,010,000 of operations if the number of relationships between the objects of the database is very high. On the other hand, the CK algorithm can necessitate a maximum of operations of the order of 2000. However, there exists

additional time overhead for the CK algorithm. The counter associated with each attribute of every object of the database has to be incremented whenever the attribute is updated. This can be very time consuming. This explains at least partially why the CK clustering algorithm performs the best when the Read/Write ration is high, that is when few updates are done. This last point is also insisted on in [9].

The comparison results of the two algorithms are summarized in Figure 7.

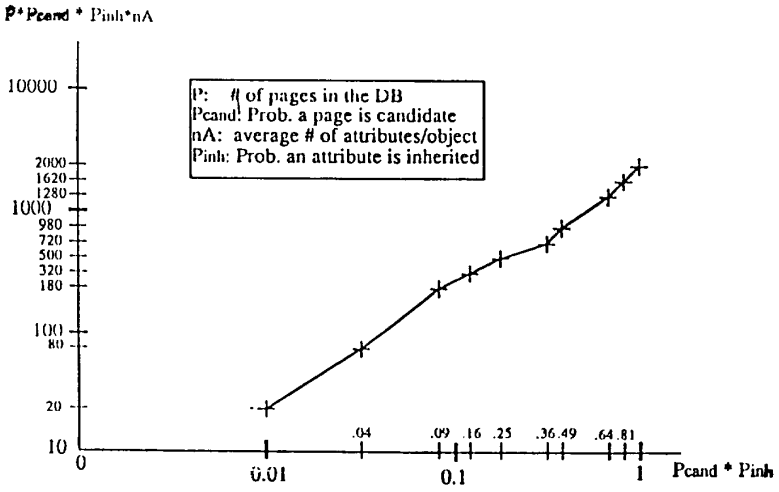

Figure 6. Time Complexity of the CK Clustering Algorithm

Algorithm / Criteria	Cactis algorithm	Ck algorithm
Space requirements	2*N + 2*R 2K to 20K bytes	nA * N 2K bytes
Time overhead	Syst. running: update 2 counters syst.idle: O(N*(N + R)) $O(10^4)$ to $O(10^6)$ operations	update one counter O(N_Cand * nb_attr_inh) =O(P*Pcand * nA*Pinh) <= 2000 operations
Maximum improvements	60 %	200 % with high Read/Write ratio and high structure density.

Figure 7. Comparison of the Cactis and CK algorithms

6. Page or Segment Clustering ?

Physically, there are two levels of clustering: page and segment levels. According to [6], page clustering, where a page is the smallest physical unit read from disk, is more suitable for clustering by index, reference and composite objects. On another hand segment clustering, in which a segment is a logical grouping specified by the user, is more useful for type clustering and composite objects while being used at a sufficient coarse grain. In the three clustering methods we presented, two of them use page clustering and the other one segment clustering.

In the CK algorithm, page clustering is used but this choice is not explained by the authors and no other alternatives were explored. More precisely, in the simulation study realized in [7], the page size is taken as a static parameter and no experiment was conducted to determine the impact of this parameter. It is our belief that this parameter has a great influence over the CK algorithm, especially because this algorithm causes a page split when a page is full. A smaller number of page splits could be required with a larger page size. The algorithm as proposed in [8] was only under design and did not consider an implementation within a particular DBMS. Moreover, no requirement is specified toward the operating system. So various alternatives could have been compared. This leads us to consider the use of segment clustering for the CK algorithm. Using segment clustering, when a new object is created the same principle as in the CK algorithm can be followed to identify the best segment. However, all the cost formulas have to be changed to consider a segment instead of a single page.

Segment management can be done in different ways. First, the size of a segment can be fixed by the system or variable. When segments are of fixed size, the number of pages they contain gives the number of I/Os necessary to load the segment. When a segment runs out of space, two options exist to extend it. A new page can be allocated and linked to the segment (a pointer must be maintained in the segment descriptor). This is what is suggested in ORION where composite objects can be stored in a B-tree of pages [3]. On another hand, a new segment can be allocated and linked to the previous one. The first option implies some overhead to find the address of each additional page, when all the related pages are prefetched if this is provided by the buffer manager. In the second solution, the difficulty is to find enough contiguous free space to allocate one entire segment at a time.

With regard to the page splitting used by the CK algorithm, segment clustering would change the splitting radically. It may not be useful anymore if we consider that most of the candidate pages will happen to be present in the same segment. Consequently, new space has to be found by extending the segment, instead of splitting it. On another hand, a segment-splitting would have to be used instead of the page-splitting. All the objects present in the segment would have to be scanned and distributed in two storage units. Segment-splitting would of course implies more overhead than page-splitting because of the larger number of elements involved, but this would be balanced by the fact that it would be performed less often.

The Cactis DBMS runs in the UNIX/C Sun workstations environment. The influence of this environment over the implementation is not discussed by the author of [11]. The clustering operates at the page level by manipulating blocks of fixed size. This size is not mentioned and the trade-off that it implies is not discussed, but we can consider that this is a parameter imposed by the environment. However, with regard to the way the clustering algorithm performs, some improvement could be obtained by allocating contiguous blocks to store related objects. Thus less overhead would appear during the loading of related objects because the heads of the disks would not need many moves.

7. Conclusions

Among the three clustering strategies we presented, the method in ORION is the most close to what is accomplished in conventional DBMS. A class is considered as equivalent to a relation and is the unit of clustering. Even if ORION allows to cluster composite objects by putting objects of different classes in the same segment, all the objects of a class are stored together and are supposed to have exactly the same behavior. Moreover, the fact that ORION relies entirely on users' hints to cluster the database prevents it to benefit as much as possible from the rich relationships existing among objects.

The solution adopted by the Cactis algorithm

makes clustering a static procedure which is run periodically when the system is quiesced. This limits the domain of applications of this system, since it requires periodic idle times. It also creates a substantial overhead when the system is running to keep track of the number of times each object is accessed and the number of times a relationship is crossed. Moreover, this algorithm wastes space on disk by allocating a new block whenever an object cannot fit into a block. The remaining space in this block is no longer considered.

The CK algorithm is the most sophisticated one and gives the largest improvements. It is dynamic and performs reclustering whenever a new object is created. It is designed for the field of CAD applications and makes use of the structural relationships and instance-to-instance inheritance semantics to improve the DBMS response time. However, it can create a considerable overhead and may need to be disabled when the system becomes heavily loaded. Consequently, the benefits of the algorithm appear only if the Read/Write ratio of the application is high [7]. To remove this limitation, we presented how segment clustering, instead of page clustering, could reduce the number of cases where a page split is necessary.

7 - References

[1] ANDREWS, T. "Programming with VBase", in Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD, R. GUPTA and E. HOROWITZ Editors. Prentice Hall, 1991, pp 130-177.

[2] ANDREWS, T., C. HARRIS, K. SINKEL, "ONTOS: A Persistent Database for C++", in Object-Oriented Databases with Applications to CASE, Networks and VLSI CAD, R. GUPTA and E. HOROWITZ Editors. Prentice Hall, 1991.

[3] BANERJEE, J., H. T. CHOU, J.F. GARZA, W. KIM, D. WOELK, N. BALLOU and H-J KIM, "Data Model Issues for Object-Oriented Applications", ACM Trans. on O.I.S., Vol. 5, No. 1, Jan. 1987, pp 3-26.

[4] BETARY, D. S., W. KIM, "Modeling Concepts for VLSI CAD Objects", ACM Trans. on Database Systems, Vol. 10, No. 3, Sept. 1985, pp 322-346.

[5] BLOOM, T., S.B. ZDONIK, "Issues in the Design of Object-Oriented Database Programming Languages", ACM OOPSLA'87 Proceedings, Oct. 4-8 1987.

[6] CATTELL, R., Object Data Management, Addison-Wesley Publishing Company, 1991.

[7] CHANG, E., R.H. KATZ, "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS", ACM Sigmod Record, Conference, 1989, pp 348-357.

[8] CHANG, E., R.H. KATZ, "Inheritance in Computer-Aided Design Databases: Semantics and Implementation Issues", CAD, Vol. 22, No. 8, Oct. 1990.

[9] CHENG, J., A.R. HURSON, "Effective clustering of complex objects in Object-Oriented Databases", ACM Proc. of the SIGMOD Int. Conf. on Management of Data, pp 22-31, Denver, Colorado, May 1991.

[10] FORD, S., et al., "ZEITGEIST: Database Support for Object-Oriented Programming", in Advances in Object-Oriented Database systems, Lecture Notes in computer Science, 2nd International Workshop on Object-Oriented Database Systems, Springer-Verlag, Sept. 1988.

[11] HUDSON, S.E., R.KING, "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Management System", ACM Trans. on Database Systems, Vol. 14, No. 3, Sept. 1989, pp 291-321.

[12] KIM, W., J. BANERJEE, H-T. CHOU, J.F. GARZA, D. WOELK, "Composite Object Support in an Object-Oriented Database System", ACM OOPSLA'87 Proceeding, Oct. 1987.

[13] KIM, W., "Architectural issues in Object-Oriented Databases", JOOP, March/April 1990, pp 29-38.

[14] MAIER, D., J. STEIN, A.PURDY, "Development of an Object-Oriented DBMS", ACM OOPSLA'86, Sept. 1986.

[15] WILKES, W., "Instance Inheritance for Object Oriented Databases", in Advances in Object-Oriented Database systems, Lecture Notes in computer Science, Springer-Verlag, Sept. 1988.