CLUSTERING TECHNIQUES FOR ADAPTIVE HORIZONTAL FRAGMENTATION IN OBJECT ORIENTED DATABASES

Alina Campan, Adrian Sergiu Darabant, Gabriela Serban

Abstract. Optimal application performance in a Distributed Object Oriented System requires class fragmentation and the development of allocation schemes to place fragments at distributed sites so data transfer is minimal. A horizontal fragmentation approach that uses data mining clustering methods for partitioning object instances into fragments has already been presented in [1, 2, 3, 4]. Essentially, our approach takes full advantage of existing data, where statistics are already present, and develops fragmentation around user applications (queries) that are to be optimized by the obtained fragmentation. But real databases applications evolve in time, and consequently require re-fragmentation in order to deal with new applications entering the system and other leaving. Obviously, for obtaining the fragmentation that fits the new user applications set, the original fragmentation scheme can be applied from scratch. However, this process can be inefficient. In this paper we extend our initial fragmentation approach and propose an incremental method to cope with the evolving user application set. Namely, we handle here the case when new user applications arrive in the system and the current fragments must be accordingly adapted.

2000 *Mathematics Subject Classification*: 62H30, 68P15.

1. Introduction

Horizontal fragmentation, in object oriented database systems, distributes class instances (objects) into fragments. There are several theoretical approaches for relational distributed databases fragmentation, some of them with practical results. Few of them have been extended and adapted to distribute object oriented databases. However, this extension of a limited technique, from

a simple model to a complex one, is not natural, as the additional complexity cannot be flexibly coped with. We have proposed in [1, 2, 3, 4] a novel approach for primary and derived fragmentation of object oriented databases. As the relational model can be viewed as a particular case of the object oriented model, the proposed techniques can be successfully applied as well for relational databases. These new techniques are built around data mining clustering methods. We derived a mathematical model of the database that captures the significance of queries and the affinity between them and the class instances. This model allows an almost off-the-shelf automatic database fragmentation using clustering algorithms, as opposed to other existing approaches, which all require a substantial human intervention in the decisional process. But real databases evolve in time, and consequently require re-fragmentation in order to deal with new applications entering the system and others leaving, and with object set evolution. Obviously, for obtaining the fragmentation that fits the new user applications set, the original fragmentation scheme can be applied from scratch, an undesirable alternative from the point of view of processing effort, extended database maintenance and unavailability time etc. To our knowledge, there are no practical approaches for incrementally maintaining an efficient database fragmentation. We propose in this paper an incremental technique to cope with the evolving user application set. Namely, we handle here the case when new user applications arrive in the system and the current primary fragments (e.g. classes have only simple attributes and simple methods) must be accordingly adapted.

2. Mathematical Vector Model of the Object Oriented Database

We will shortly review in this section the *vector model* proposed for modelling an object oriented database. This model is described in extension in [1, 2, 3, 4].

Fragmentation of an object database means fragmenting one by one each of its classes. Thus, we will refer to the fragmentation of one class let it be C. A class C is an ordered tuple $C = (K, A, M, I)$, where A is the set of object attributes, M is the set of methods, K is the class identifier and I is the set of instances of class C. We deal in this paper only with primary fragmentation ([4]). Classes are organized in an *inheritance hierarchy*, in which a subclass is a specialization of its superclass. Although we deal here for simplicity only with *simple inheritance*, moving to multiple inheritance would not affect the

fragmentation algorithms in any way, as long as the inheritance conflicts are dealt with into the data model. Association between an object and a class is materialized by the instantiation operation. An object O is an *instance* of a class C if C is the most specialized class associated with O in the inheritance hierarchy. An object O is *member* of a class C if O is instance of C or of one of subclasses of C. An object oriented database is a set of classes from an inheritance hierarchy, with all their instances. There is a special class Root that is the ancestor of all classes in the database. Thus, in our model, the inheritance graph is a tree.

Other relations acting between classes in a database are the aggregation and association relations ([4]).

An entry point into a database is a meta-class instance bound to a known variable in the system. An entry point allows navigation from it to all classes and class instances of its sub-tree (including itself). There are usually more entry points in an *OODB*.

Given a complex hierarchy H, a path expression P is $C_1.A_1 \ldots A_n$, $n \geq 1$ where: C_1 is an entry point in H, A_1 is an attribute of class C_1, A_i is an attribute of class C_i in H such that C_i is the domain of attribute A_{i-1} of class C_{i-1} $(1 \leq i \leq n)$.

The fragmentation and allocation of an object oriented database aim to optimize the execution of a set of user queries. In general a *query* is a tuple with the following structure: *q=(Target class, Qualification clause)*, where: *Target class* specifies the class over which the query returns its object instances; *Qualification clause* is a logical expression over the class attributes in conjunctive normal form. The logical expression is constructed using simple predicates: $attribute\,\Theta\,value$ where $q \in \{<, >, \leq, \geq, =, \neq\}$.

Let $Q = \{q_1, \ldots, q_t\}$ be a set of queries in respect to which the fragmentation has to be performed. Let $Pred_Q = \{p_1, \ldots, p_q\}$ be the set of all simple predicates Q is defined on. Let $Pred_Q(C) = \{p \in Pred_Q |$ p imposes a condition to an attribute of class $C\}$. Given two classes C and C', where C' is subclass of C, $Pred_Q(C') \supseteq Pred_Q(C)$. Thus the set of predicates for class C' comprises all the predicates directly imposed on attributes of C' and the predicates defined on attributes of its parent class C and inherited from it. The reasons for this condition inheritance are explained in [1, 2, 3].

To each object O_i in the set $Inst(C)$ of all instances of class C, $i = 1..m$, $m = |Inst(C)|$, we associate an *object-condition vector* $a_i = (a_{i1}, \ldots, a_{is})$, where $Pred_Q(C) = \{p_1, \ldots, p_s\}$:

$$a_{ij} = \begin{cases} 0, \ if \ p_j(O_i) = false \\ 1, \ if \ p_j(O_i) = true \end{cases}$$

Objects will be grouped together in fragments so that objects within a fragment to have high similarity with each other and low similarity with objects in other groups. Similarity and dissimilarity between objects are calculated using metric or semi-metric functions, applied to the *object-condition vectors* that characterize objects. We use in this paper the *Euclidian distance* for measuring objects similarity:

$d_E(a_i, a_j) = \sqrt{\sum_{l=1}^{s}(a_{il} - a_{jl})^2}$,

where $a_i, \ a_j$ are the *object-condition vectors* of $O_i, \ O_j \in Inst(C)$.

3. Incremental Fragmentation Using the CBIC Algorithm

3.1 Initial Fragmentation Phase

When passing from a centralized database to a distributed one, an initial fragmentation is required. In our approach ([1,2,3,4]), given a set $Q_{init} = \{q_1, \ldots, q_p\}$ of queries, the initial fragmentation phase of the object set $Inst(C)$ of class C requires first that objects in $Inst(C)$ to be modelled as described above. Then a clustering method (*k-means*) is applied over the vector space describing $Inst(C)$, and the resulting clusters represent the fragments for class C.

3.2 User Applications Change

The existing fragmentation of the distributed object oriented database was developed such that to optimize the execution of the initial query set, Q_{init}. When new queries arrive into system $Q_{new} = Q_{init} \cup \{q_{p+1}, \ldots, q_t\}$, the current fragmentation must be adapted. We apply in this intention an incremental, *k-means* based clustering method, *Core Based Incremental Clustering* (*CBIC*) ([5,6]).

The extension of the query set Q_{init} to Q_{new} means that for a number of classes in the database, their associated set of predicates increases. These classes have to be re-fragmented to fit the new query set. Let C be such a class, for which $Pred_{Q\_init}(C) = \{p_1, \ldots, p_n\}$ evolves to $Pred_{Q\_new}(C) = Pred_{Q\_init}(C) \cup \{p_{n+1}, \ldots, p_s\}$. Consequently, the *object-condition vector* for each object $O_i \in Inst(C)$ is extended as follows:

$a'_i = (\underbrace{a_{i1}, \ldots, a_{in}}_{\text{initial object-condition } a_i \text{ of } O_i}, a_{i,n+1}, \ldots, a_{is})$

The *CBIC* method starts from the partitioning of $Inst(C)$ into clusters established by applying *k-means* in the initial fragmentation phase. Let $\{K_1, K_2, \ldots, K_p\}$ be the initial fragments of $Inst(C)$, $K_i \cap K_j = \emptyset, i \neq j, \bigcup_{l=1}^{p} K_l = Inst(C)$. *CBIC* determines then $\{K'_1, K'_2, \ldots, K'_p\}$ the new partitioning of objects in $Inst(C)$ after query set extension. It starts from the idea that, when adding few components (features, attributes) to the object-condition vectors and these components don't bring to much information in the system, then the old arrangement into clusters is close to the new one. The algorithm determines than those objects within each fragment K_i that have a considerable chance to remain together in the same cluster. They are those objects that, after feature extension, still remain closer to the centroid (cluster mean) of cluster K_i. These objects form what is called the core of cluster K_i, denoted by $Core_i$. Note: the centroid of K_i to which we report the object after extension is, of course, calculated as the mean of the extended object-condition vectors of objects in K_i.

The cores of all fragments K_i, $i = 1..p$, will be the new initial clusters from which the iterative partitioning process begins. Next, *CBIC* proceeds in the same manner as the classical *k-means* does. The *CBIC* algorithm can be found in [5,6]. As experiments show, the result is generally reached by *CBIC* more efficiently than running *k-means* again from the scratch on the feature-extended object set.

4. Results and Evaluation

4.1 Sample Database and Queries

We use a sample object database that represents a reduced university database. The inheritance hierarchy is given in Figure 1.

The queries running on the classes of the database are given bellow.

q_1: This application retrieves all graduate students enrolled in Component Oriented Programming and Intelligent Systems departments.

q_1 = (Grad, Grad.Dept in ("Component Oriented Programming", "Intelligent Systems"))

q_2: This application retrieves all undergraduate students enrolled in Computer Science departments and having grades between 7 and 10.

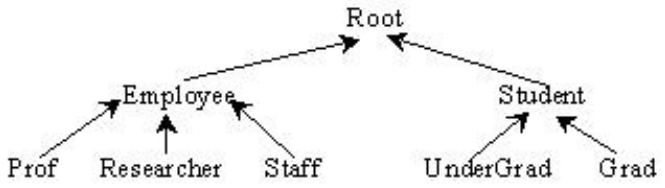

Figure 1: The database class hierarchy

q_2 = (UnderGrad, UnderGrad.Dept like "CS%" and UnderGrad.Grade between 7 and 10)
q_3: This application retrieves all undergraduate students enrolled in Computer Science or Mathematics departments and which are older than 24.
q_3 = (UnderGrad, (UnderGrad.Dept like "Math%" or UnderGrad.Dept like "CS%") and UnderGrad.Age≥24)
q_4: This application retrieves all researchers having published at least two papers.
q_4 = (Researcher, Researcher.count(Reasercher.Doc)≥2)
q_5: This application retrieves all teachers employed in Component Oriented Programming or Intelligent Systems departments and having salaries over 40000.
q_5 = (Prof, Prof.Dept in ("Component Oriented Programming","Intelligent Systems") and Prof.Salary≥40000)
q_6: This application retrieves all teachers having published in IEEE or ACM publications.
q_6 = (Prof, Prof.Doc.Publisher in ("IEEE", "ACM"))
q_7: This application retrieves all students failing to get their remove.
q_7 = (Student, Student.Grade≤5)
q_8: This application retrieves all employees having salaries over 35000.
q_8 = (Employee, Employee.Salary>35000)
q_9: This application retrieves all graduate students having published at least one paper.
q_9 = (Grad, Grad.count(Grad.Doc)≥1)
q_{10}: This application retrieves all staff employees having salaries over 12000.
q_{10} = (Staff, Staff.Salary>12000)
q_{11}: This application retrieves all researchers having published a smaller num-

ber of papers than the average number of papers published by all researchers.
q_{11} = (Researcher, Researcher.Count(Paper)<Avg(Researcher.Count(Paper))
q_{12}: This application retrieves all married graduate students.
q_{12} = (Grad, Grad.MaritalStatus="married")
q_{13}: This application retrieves all undergraduate students enrolled in Mathematics and Computer Science departments.
q_{13} = (Undergraduate, Undergraduate.Dept like "Math-CS%")
q_{14}: This application retrieves all persons older than 30.
q_{14} =(Person, Person, Person.Age>30)
q_{15}: This application retrieves all assistant professors older than 28.
q_{15} = (Prof, Prof.Position="assistant professor" and Prof.age>28)
q_{16}: This application retrieves all students of hungarian or german nationality.
q_{16} = (Student, Student.Nationality in ("hungarian", "german")

Queries from q_1 to q_{12} are the initial ones - the initial horizontal fragmentation and allocation were performed in respect to these queries, so that to optimize their execution: $Q_{init} = \{q_1, \ldots, q_{12}\}$. Queries from q_{13} to q_{16} are newly entered in the system, and the existing fragmentation must be adapted to fit to and to optimize the new application set: $Q_{new} = Q_{init} \cup \{q_{13}, \ldots, q_{16}\}$.

4.2 Quality Measures

We use two different kinds of quality measures: measures for evaluating the performances of the two fragmentation schemes (*CBIC* and *k-means*) and measures for evaluating the quality and performance of the fragmentation results.

Quality measures for the performances of the two fragmentation schemes (*CBIC* and *k-means*). As a quality measure for *CBIC* we take the movement degree of the core objects and of the extra-core objects. In other words, we measure how the objects in either $Core_i$ or $OCore_i = K_i \setminus Core_i$ remain together in clusters after the *CBIC* algorithm ends. As expected, more stable the core objects are and more they remain together in respect to the initial sets $Core_i$, better was the decision to choose them as seed for the incremental clustering process. Also, as the experiments show, the movement degree was in most cases smaller for the core objects than for the extra-core objects. The core stability factor is expressed as follows ([6]):

$$CSF(CORE) = \frac{\sum_{j=1}^{p} \frac{|Core_j|}{no\ of\ clusters\ where\ the\ objects\ in\ Core_j\ ended}}{\sum_{j=1}^{p} |Core_j|}$$

The out-of-core stability factor ($OCSF(OCORE)$) is described accordingly ([6]). Also, for comparing the performances of *CBIC* and *k-means*, we give the number of iterations needed by each of them for reaching the result.

Table 1: Comparative results for the two fragmentation schemes (*CBIC* and *k-means*)

Experiment	UnderGrad	Grad	Prof	Staff	Researcher
No of objects	128	26	30	8	6
No of attributes (m+s)	8	6	7	3	4
No of new attributes (s)	3	2	3	1	1
No of k-means iterations for m attributes	3	3	3	2	2
No of k-means iterations for (m+s) attributes	4	3	3	2	2
No of CBIC iterations for (m+s) attributes	2	3	3	2	2
CSF(CORE)	0.72	1.0	1.0	1.0	1.0
OCSF(OCORE)	0.86	-	0.5	-	-

From Table 1 we observe that using the *CBIC* algorithm the number of iterations for finding the solution is, in the average, smaller, and also the cores' stability factor, $CSF(CORE)$, is high.

In Table 2 we present, for each experiment, the components (attributes) in decreasing order of their information gain (IG). We emphasized the new entered attributed, for observing their significance in differentiating objects.

From Table 2 it results that the importance of the added attributes influence the number of iterations performed by the *CBIC* algorithm for finding the solution. When the information brought by the added attributes was close to that of the initial ones, the number of iterations performed by *CBIC* is also close to the number of iterations performed by *k-means* for all the attributes.

Measures for the quality and performance of the fragmentation results. For evaluation we use a variant of the Partition Evaluator as proposed by Chakravarthy for vertical relational fragmentation ([1, 2]). The Partition Evaluator as proposed by Chakravarthy is composed of two terms: the local irrelevant access cost (EM) and the remote relevant access cost (ER). For

Table 2: The decreasing order of attributes in respect to the information gain measure

Experiment	Order of attributes	IG of new attributes / IG of old attributes (%)
Undergrad	1 3 **6** 4 **7** **8** 5 2	27.5%
Grad	4 1 2 **5** **6** 3	13.7%
Prof	2 4 **6** 3 **5** **7** 1	37.7%
Staff	2 **3** 1	4.7%
Researcher	2 **4** 3 1	13.7%

a given class C, the EM term computes the number of non-accessed local fragment objects in all fragments, while the ER term computes the number of remote objects accessed by all queries running at each site.

$PE(C) = EM^2 + ER^2$, where:

$EM^2(C) = \sum_{i=1}^{M} \sum_{t=1}^{T} freq_{ts}^2 \times |Acc_{it}| \times (1 - \frac{|Acc_{it}|}{|F_i|})$

$ER^2(C) = \sum_{t=1}^{T} \{ \sum_{s=1}^{S} \sum_{i=1}^{M} freq_{ts}^2 \times |Acc_{it}| \times \frac{|Acc_{it}|}{|F_i|} \}$

In EM expression, s is the site where a fragment F_i is located, while in ER s is any site not containing F_i. M is the number of clusters for class C, T is the number of queries and S is the number of sites. Acc_{it} represents the set of objects accessed by the query qt from the fragment F_i. The smaller PE is, better fragmentation quality we have.

In Figure 2 we present the results (quality measure) of applying the *CBIC* method for incrementally clustering a set of classes. We start from the presumption that the classes are already horizontally fragmented. As in any usual database use cases during the database lifecycle, it is normal that new applications/queries arrive. In the same time the requirements of the existing one might change having as result the need to re-write existing queries. This would lead to the fact that the set of query conditions that drove fragmentation when the database was designed is no longer exactly the same. With the new applications arriving there is a new set of fragmentation conditions that arrive into the environment. The impact of the new condition set might impose a new re-fragmentation of the database. Instead of applying fragmentation from zero the *CBIC* method starts with a nucleus of already fragmented classes and in-

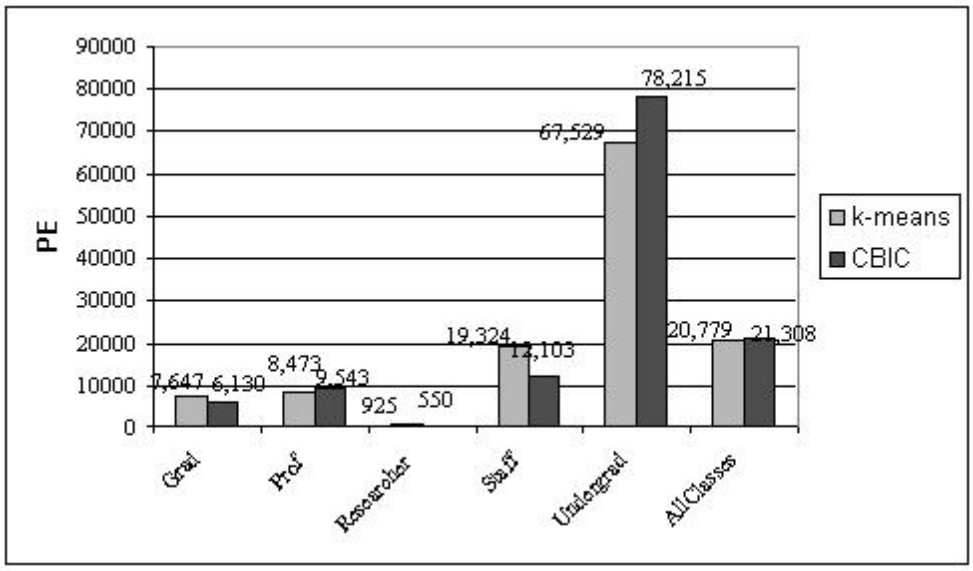

Figure 2: Experimental results

crementally introduces the effect of the new conditions into the fragmentation result. Figure 2 presents the original fragmentation quality factors for each class after applying the *k-means* fragmentation method and the new quality measures for each class after incrementally re-fragmenting the database with the set of the new added conditions. As it can be seen, the incremental fragmentation keeps the quality measures around the original ones without the burden of the re-applying the fragmentation process on the entire database. As a result we stay in about the same quality scale and we improve the processing time as the incremental method always performs in less time than the full fragmentation process. The incremental fragmentation results in better quality for *Grad*, *Researcher* and *Staff* classes and for slightly worse quality factors for the *Prof* and *Undergrad* classes. As a result we keep the fragmentation quality at about the same level without re-fragmenting the entire database once we have new applications accessing data in a distributed architecture.

5. Conclusions and Future Work

We proposed in this paper a method for incrementally maintaining the primary horizontal fragments of an object oriented database. As our experiments

show, this method can be effective and efficient, by reducing the processing effort and time in maintaining and tuning the database. We aim to extend the approach to derived horizontal fragmentation as well.

Also, as we said, in most of the experiments we have made on different data sets, the *CBIC* method proved to be effective. But there are some situations when it is better to resort to a full clusterization of the feature-extended object set, and not to adapt the existing fragmentation using the *CBIC* algorithm. For example such a situation is the addition of a large number of queries. This enlargement of the query set will add a significant number of features to some of the class extensions. If these new features bring a large information gain and contradictory information with respect to the old feature set, than it might be the case that incremental clustering is less effective than a normal clustering process. As future work, we plan to first isolate conditions to decide when is more effective to adapt incrementally (using *CBIC*) the fragmentation of an object oriented database than to recalculate its fragmentation (using *k-means*) from scratch. Next we want to find methods and algorithms for adapting a fragmentation to general modifications of the query set, including not only the appearance of new queries, but also the elimination of some of them. Also, the evolution of class extensions (by creating or deleting class objects) must be dealt with in an incremental manner, especially because the quantity of such modifications is usually small in comparison with the entire object database. We want to explore all the dimensions of this problem: identify algorithms that fit, conditions when they are effective etc.

References

[1] Darabant, A.S., Campan, A., Semi-supervised learning techniques: k-means clustering in OODB Fragmentation, IEEE International Conference on Computational Cybernetics ICCC 2004, Vienna University of Technology, Austria, August 30 - September 1, 2004, pp. 333–338.

[2] Darabant, A.S., Campan, A., Hierarchical AI Clustering for Horizontal Object Fragmentation, In Proc of Int. Conf. of Computers and Communications, Oradea, May, 2004, pp. 117–122.

[3] Darabant, A.S., Campan, A., AI Clustering Techniques: a New Approach to Object Oriented Database Fragmentation, in Proceedings of the 8th IEEE International Conference on Intelligent Engineering Systems, Cluj Napoca, 2004, pp. 73–78.

[4] Darabant, A.S., Campan, A., Cret, O., Hierarchical Clustering in Object Oriented Data Models with Complex Class Relationships, in Proceedings of the 8th IEEE International Conference on Intelligent Engineering Systems, Cluj Napoca, 2004, pp. 307–312.

[5] Şerban, G., Campan, A., Core Based Incremental Clustering, Studia Universitatis "Babeş-Bolyai", Informatica, XLXI(2), 2005, pp. 89–96.

[6] Şerban, G., Campan, A., Incremental Clustering Using a Core-Based Approach, in Proc. of the 20th International Symposium on Computer and Information Sciences (ISCIS'05), Istanbul, Turkey, 2005 (to appear).

Alina Campan, Adrian Sergiu Darabant, Gabriela Serban
Department of Computer Science
University of Babes Bolyai University, Cluj Napoca
M. Kogalniceanu 1, Cluj-Napoca, Romania
email:{*alina, dadi, gabis*}*@cs.ubbcluj.ro*