USING FUZZY CLUSTERING FOR ADVANCED OODB HORIZONTAL FRAGMENTATION WITH FINE-GRAINED REPLICATION

Adrian Sergiu Darabant, Alina Campan
Babes Bolyai University, Faculty of Mathematics and Computer Science
1 M. Kogalniceanu, Cluj Napoca
Romania
dadi@cs.ubbcluj.ro, alina@cs.ubbcluj.ro

ABSTRACT
In this paper we present a new approach for horizontal object oriented database fragmentation combined with fine-grained object level replication in one step. We build our fragmentation/replication method using AI probabilistic clustering (fuzzy clustering). Fragmentation quality evaluation is provided using an evaluator function.

KEY WORDS
Object oriented databases, fragmentation, replication, and fuzzy techniques

1. Introduction

Designing an efficient DOODB system requires fragmentation and allocation techniques capable of handling the complex features of the object oriented data model. While the design phase could be adapted from the relational techniques, this solution is certainly not the most appropriate as the OO model is inherently more complex than the relational model. In order to cope with the increased complexity of the OO model, one can divide class features as follows: *simple attributes, complex attributes, simple methods*, *complex methods* [6].
Fragmentation methods for OODB environments, or flat data models have been generally considered in [2, 8]. Bellatreche et al. [8] propose a method that emphasizes the role of queries in the horizontal fragmentation. We have already discussed alternative AI clustering methods for horizontal fragmentation in OO models with simple and complex attributes/methods, without replication, in [9, 10, 11, 12]. Generally, *replication* is performed at *fragment level* in all allocation schemes. Synchronous and asynchronous replication are described in [14].

Contributions

In this paper we propose a method for *constructing class fragments* in an object oriented database with *object level replication* handled directly in the fragmentation step.

In OODBs, horizontal fragmentation can be carried in two steps: *primary* and *derived.* In our method, primary and derived horizontal fragmentation are performed in a single step, similar to [11, 12]. Primary fragmentation groups class instances according to a set of query conditions [9] imposed on the values of their simple attributes/methods. Derived fragmentation takes into account the class relationships (aggregation, association, complex methods). It groups instances of a class in fragments according to the fragmentation of the related classes. Generally, there are two approaches in derived fragmentation: *left order derived fragmentation (parent first)* and *right order derived fragmentation (child first)*. They differ in the order in which two related classes are fragmented. In the left order derived fragmentation, the referring (parent) class is fragmented first and determines a partitioning of the instance set of the referred (child) class. In the right order derived fragmentation, the referred class is fragmented first and determines the partitioning of the instances of the referring class.
Objects are modeled in a vector space. Each object is represented as a vector. We use distance functions for measuring the dissimilarity between any two objects of the same class. Objects are placed into fragments according to their dissimilarity in respect to a set of relevant user queries. Objects are grouped into overlapping fragments using a fuzzy clustering method [18].
For evaluating the results of our fragmentation method we propose an evaluation function that expresses how well the resulting fragmentation fits the input set of user applications.
The paper is organized as follows. The next section of this work presents the object data model and the constructs used in defining the object database and expressing queries. Section 3 introduces the vector space model we use to compare objects, methods for constructing the object vectors and similarity metrics over this vector space. Section 4 presents our fuzzy clustering fragmentation algorithm. In section 5 we present a fragmentation example over a class hierarchy and we evaluate the quality of our fragmentation scheme.

2. Data Model

We use an object-oriented model with the basic features described in the literature [19]. An OODB is a set of classes with all their instances and relationships (inheritance, aggregations, associations). Although we deal here, for simplicity, only with simple inheritance, moving to multiple inheritance would not affect the fragmentation algorithm in any way, as long as the inheritance conflicts are dealt with into the data model. 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.

An *entry point* into a database is a metaclass instance [9] bound to a known variable in the system. It allows navigation to all classes and class instances of its sub-tree. There are usually more entry points in an OODB. Given a *complex* hierarchy H, a *path expression* P, $C_1.A_1. \ldots A_n$, $n \geq 1$ denotes a path in the aggregation/association graph [9,11]. As presented in [9], a *query* is a tuple with the following structure q=(Target class, Range source, Qualification clause).

3. Vector Space Modeling

3.1 Primary Fragmentation Modeling

We denote by $Q=\{q_1, \ldots, q_t\}$ the set of all queries in respect to which we want to perform the fragmentation. Let $Pred=\{p_1, \ldots, p_q\}$ be the set of all atomic predicates Q is defined on. Let $Pred(C)=\{p \in Pred \mid p$ imposes a condition to an attribute of class C or of its parent$\}$. Given the predicate $p \equiv C_1.A_1. \ldots A_n \ \theta\ value$, $p \in Pred(C_n)$, where class C_i is the complex domain of A_{i-1}, i=2..n. Thus, given *two* classes C and C', where C' is subclass of C, $Pred(C') \supseteq Pred(C)$ [9].

We construct the *object condition matrix* for class C, $OCM(C) = \{a_{ij}, 1 \leq i \leq |Inst(C)|, 1 \leq j \leq |Pred(C)|\}$, where $Inst(C) = \{O_1, \ldots O_m\}$ is the set of all instances of class C (objects), $Pred(C) = \{p_1, \ldots, p_n\}$:

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

Each line i in OCM(C) is the *object condition vector* of O_i, where $O_i \in Inst(C)$.

3.2 Attribute Induced Derived Fragmentation Modeling

We have captured so far all characteristics of simple attributes and methods. We need to express the class relationships in our vector space model. We first model the aggregation and association relations.

Given two classes C_O (owner) and C_M (member), where C_M is the domain of an attribute of C_O, a path expression traversing this link navigates from instances of C_O to one or more instances of C_M. In the case of left derived fragmentation C_O will be fragmented first, followed by C_M. In the right derived fragmentation variant the order in which the two classes are fragmented is reversed. Each of the two strategies is suitable for different query evaluation strategies. For example, in reverse traversal query evaluation strategy, the right derived fragmentation variant gives the best results. We assume here, for space reasons, that right derived fragmentation method is used. However, both: the algorithm and the vector space model remain the same when considering left derived fragmentation order.

In right derived fragmentation method, when fragmenting C_O we should take in account the fragmentation of C_M [12]. Objects of a fragment of C_O should aggregate as much as possible objects from the same fragment of C_M.

Let $\{F_1, \ldots F_n\}$ be the fragments of C_M. We denote by $Agg(O_i, F_j)=\{O^m \mid O^m \in F_j,\ O_i$ *references* $O^m\}$ the set of objects in fragment F_j referred by O_i.

Given the set of fragments for C_M, we define the *attribute-link induced object condition vectors for derived fragmentation* as $ad_i = (ad_{i1}, ad_{i2}, \ldots, ad_{in})$, where each vector component is expressed by the following formula:

$$ad_{ij} = \text{sgn}\left(\left|Agg(O_i, F_j)\right|\right), \quad j = \overline{1,n} \qquad (2)$$

For an object $O_i \in Inst(C_O)$ and a fragment F_j of C_M, ad_{ij} is 1 if O_i is linked to at least one object of F_j and is 0 otherwise. Two objects are candidates to be placed in the same fragment of C_O in respect to F_j if they are both related in the same way to F_j.

3.3 Method Induced Derived Fragmentation Modeling

In the following paragraphs we model the class relationships induced by the presence of complex methods. Given a class with complex methods C (owner) that has to be fragmented, we need to take in account the fragmentation of classes referred by its complex methods. We express method reference dependencies in our vector space.

We denote by $MetComplex(C) = \{m_i \mid m_i$ *is a complex method of* $C\}$ – the set of all complex methods of class C.

Let $SetCRef(m,C)=\{C_R \mid C \neq C_R,\ C_R$ *is referred by method* $m \in MetComplex(C)\}$ be the set of classes referred by the complex method m of class C. For a given instance of a class C with complex methods we denote as:

$SetORef(m, O_i, C_R)=\{O'_r \in Inst(C_R) \mid C_R \in SetCRef(m,C),\ m \in MetComplex(C),\ O'_r$ *is referred by method* $m\}$ – the set of instances of class C_R, referred by the complex method m of class C, called by object O_i.

For each pair $(m_k, C_R) \in \{m_k \in MetComplex(C)\} x\ SetCRef(m_k, C)$ we quantify the way each instance of C refers - through complex methods - instances from fragments of C_R. Given a class C_R referred by a complex method m_k of class C, and the fragments $\{F_1, \ldots F_n\}$ of class C_R, we define the *method-link induced object condition vectors for derived fragmentation*. For each instance O_i of C let $md_i = (md_{i1}, md_{i2}, \ldots, md_{in})$ be the *method-link induced object condition vector*. Each vector component is defined by the following formula:

$$md_{ij} = \text{sgn}\left(\left| \begin{matrix} \{O_l \in Inst(C_R) \mid O_l \in F_j\} \cap \\ \cap\, SetORef(m_k, O_i, C_R) \end{matrix} \right|\right), \ j = \overline{1,n} \qquad (3)$$

Each md_{ij} evaluates to 1 when $O_i \in Inst(C)$ refers objects from fragment F_j of class C_R and 0 otherwise. We obtain one *method-link induced object condition vector* for each object O_i and each pair (m_k,C_R) in $\{m_k \in MetComplex(C)\} x\ SetCRef(m_k,C)$.

3.4 Derived Fragmentation Modeling

As the number of elements in $\{m_k \in MetComplex(C)\} x\ SetCRef(m_k,C)$ is usually large we need to use some heuristics in order to retain only the pairs with significant impact in the fragmentation. In order for a pair (m_k, C_R) to be kept it should satisfy the following combined restrictions: (a) The number of calls to the method m_k should be significant compared to the contribution brought by all method calls made by applications running on the database; (b) The number of instances of C_R referred by the method m_k should be significant compared to the number of instances of all classes generally referred by the applications. The above conditions are expressed in the following formula (*significance factor*):

$$Sig(m_k, C_R) = \frac{NrCalls(m_k)}{\sum_{m_l \in MetComplex(C)} NrCalls(m_l)} \times \\ \times \frac{\sum_{O_i \in Inst(C)} \left| SetORef(m_k, O_i, C_R) \right|}{\sum_{C_p \in SetCRef(m_k,C)} \sum_{O_r \in Inst(C)} SetORef(m_k, O_r, C_p)} \qquad (4)$$

In equation (4) the first factor gives the ratio between the number of calls to method m_k and the number of calls of all complex methods of class C. The second factor gives the ratio between the number of C_R instances referred by m_k and the number of all objects referred by m_k. In reality the actual method parameters would normally influence the set of objects referred by the method. Even more, the set of referred objects could be as well influenced by the internal state of the object. However, tracking all the possible combinations is computationally intractable, even in simple situations. The statistical heuristic proposed in (4) is still manageable and helps reducing the problem space dimensions.

We capture the semantic of both primary and derived fragmentation phases into one single step. We unify the object condition vector, the attribute-link and method-link induced object condition vectors for each object O_i of the class C, and we obtain the *extended object condition vector*. Each extended object condition vector quantifies all the information needed for fragmentation: the conditions imposed on the object's state and the relationships of the object with instances of related classes.

If the class C is related with classes $C_{A1}, C_{A2},\ldots, C_{Ap}$ by means of complex attributes, and with classes $C_{M1}, C_{M2}, \ldots, C_{Mr}$ by means of complex methods, then the *extended object condition vector* ae_i for object $O_i \in Inst(C)$ is obtained by appending the p attribute-link induced object condition vectors and the $mc=|\{m_k \in MetComplex(C)\} x\ SetCRef(m_k,C)|$ method-link object condition vectors to the object condition vector of O_i. However, as we have already mentioned above, we are using the *significance factor* to filter out non-relevant pairs (m_k, C_R) and vectors derived from them. The significance threshold is an input parameter for the fragmentation algorithm and its value is experimentally determined.

We denote by $EOCM(C)$ the extended object condition matrix for class C.

3.5 Dissimilarity (distance) between objects

The aim of our method is to group into a cluster those objects that are similar to one another. Distance between objects is computed using the following metrics:

$$d_E(ae_i, ae_j) = \sqrt{\sum_{k=1}^{n} \left(ae_{ik} - ae_{jk}\right)^2} \\ d_M(ae_i, ae_j) = \sum_{k=1}^{n} \left| ae_{ik} - ae_{jk} \right| \qquad (5)$$

We use d_E and d_M in (5) to measure how distant two objects are.

4. Fuzzy Clustering Based Fragmentation

Fuzzy c-means (FCM) is a method of clustering which allows one object to belong to one or more clusters. The algorithm we propose for horizontal fragmentation is described in the following:

```
Algorithm Fuzzy-c-meansFrag is
Input: Class C, Inst(C) to be fragmented, the distance function dist:Inst(C)xInst(C)→R, m=|Inst(C)|, 1<k≤m desired number of fragments, EOCM(C), z the fuzziness factor, ε_prob the probability matrix change threshold, MaxSteps maximum number of iterations, MinMembershipProb minimum membership probability, ε_centr threshold for centroid equality.
Output: The set of clusters F={$F_1$,…,$F_f$}, f ≤ k.
Var:
 U=[$u_{ij}$] the probability matrix, i=1..m, j=1..k.
Begin
 InitRandomProbMatrix($U^{(0)}$)
 // or InitGuidedProbMatrix(EOCM(C),$U^{(0)}$,k);
 f:=k; $F_j$:=∅, j=1..f; p:=1;
 Repeat
  //calculate the centroids $Centr^{(p)}=[c_j]$ with $U^{(p)}$
   For j:=1 to f do
          $c_j = \frac{\sum_{i=1}^{m} u_{ij}^z \cdot ae_i}{\sum_{i=1}^{m} u_{ij}^z}$                (a1)
   End For;
   RedIdentCentr($Centr^{(p)}$, $U^{(p)}$, f, ε_centr);
   For i:=1 to m do // update $U^{(p)}$ using $U^{(p-1)}$
     For j:=1 to f do
          $u_{ij} = \frac{1}{\sum_{l=1}^{f} \left( \frac{dist(ae_i, c_j)}{dist(ae_i, c_l)} \right)^{\frac{2}{z-1}}}$          (a2)
```

```
      End For;
    End For;
    p = p+1;
 Until (max{|u(p)ij- u(p-1)ij|}≤ε_prob) or
         (p ≥ MaxSteps);
 For i:=1 to m do
    For j:=1 to f do
      If u(p)ij ≥ MinMembershipProb then
        Fj = Fj∪{Oi};
    End For;
 End For;
End.
```

The algorithm generates the membership probability of each object of C to all clusters. It starts with an initial probability matrix $U^{(0)}$. An element u_{ij} of this matrix expresses the membership probability of object O_i to the cluster F_j. The sum of membership probabilities for an object to all clusters must be equal to one – the membership probability matrix is *standardized*. The probability matrix is optimized in an iterative manner. Each iteration starts by determining the centers of each fuzzy cluster, $Centr=\{c_1,\ldots, c_f\}$ – line (a1) in the algorithm. Next we adjust the membership probabilities to the clusters represented by the new centroids – line (a2) in the algorithm. This iterative process aims to minimize the following objective function:

$$J_z = \sum_{i=1}^{m}\sum_{j=1}^{f} u_{ij}^{z} \cdot dist(ea_i, c_j) \qquad (6)$$

where z is the fuzziness factor as described in [18]. The iterative process stops when changes in the probability matrix between two consecutive steps is insignificant (bellow a threshold value, ε_prob). The J function generally has saddle points and the generated probability series u_{ij} are not always convergent. To deal with this, we limit the number of iterations to *MaxSteps*.

At the end of the iterative process we build the horizontal fragments for the class C. We assign each object to all clusters with a membership probability exceeding *MinMembershipProb*. We chose $1/f$ (f is the number of resulting clusters) as threshold value.

If two centroids become equal in the iterative process that means that their clusters would contain similar objects (these are *degenerated clusters*). As they do not have a distinct semantic for the fragmentation, we merge all degenerated clusters with equal centroids. The resulting cluster will accumulate all objects of the source degenerated clusters by summing their membership probabilities. Degenerated clusters may appear when one initially requests more fragments than can be separated in our modeled vector space.

The initial selection of the probability matrix obviously influences the algorithm evolution. We propose two methods for initializing the probability matrix. The first one is implemented in the *InitRandomProbMatrix* procedure and generates a random standardized membership probability matrix. The second one is implemented in *InitGuidedProbMatrix* and generates the initial probability matrix as follows. It selects between the class instances the set of f most dissimilar ones (dissimilarity is measured as distance between objects). These objects are considered to be the initial centroids (*Centr*). Each object from *Centr* is assigned a membership probability of 1 to its cluster, and 0 to other clusters. For every other object we determine the set of the closest, equally distant centroids according to the distance function. Let *cnum* be the number of these centroids. We assign equal membership probabilities ($1/cnum$) to each corresponding cluster. We proceed in this manner in order to lower the risk of obtaining degenerated clusters.

5. Results and Evaluation

In this section we illustrate the experimental results obtained by applying our fragmentation scheme on a test object database. Given a set of queries, we first obtain the horizontal fragments for the classes in the database; afterwards we evaluate the quality and performance of the fragmentation results. The problem with the evaluation method is that it is difficult to quantify a fragmentation result without allocating the fragments to the nodes of a distributed system. On the other side, the allocation it's a very complex process on its own. As resolving the allocation problem in the general case is not a trivial task, we need a simplified allocation model, yet a valid one. We consider a distributed system running database applications (queries). All applications run with different frequencies on different nodes of the system. We chose to allocate each fragment to the node where it is most needed (accessed).

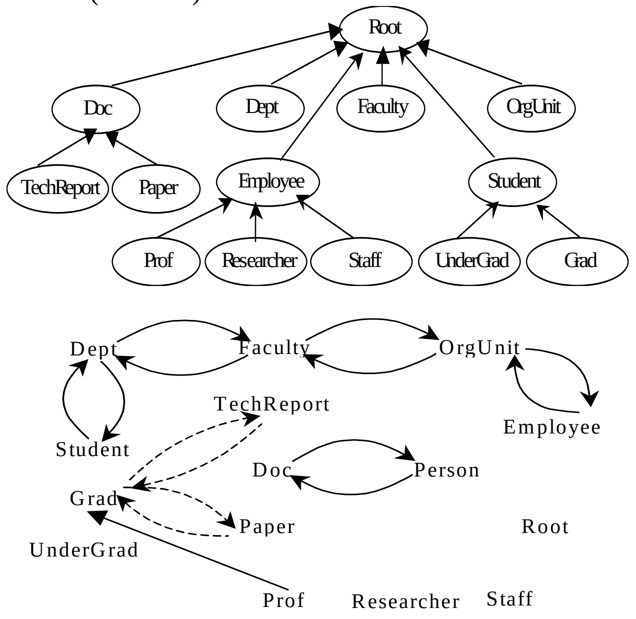

Figure 1 The database class hierarchy and aggregation/association graph

Our sample object database represents a reduced university database. The inheritance hierarchy and a trimmed down version of the aggregation/ association graph are shown in Figure 1. The links between Doc and Person should be inherited by all subclasses of Person and Doc. This is graphically represented in the figure by the dotted arrows. Similar inherited links are present for other classes in this graph (links not represented here). The motivation for aggregation/association inheritance is presented in [11]. We use the same set of 14 relevant queries for guiding the fragmentation process, as in [11].

For measuring the fragmentation quality we determine the cost of remote accesses combined with the cost of local irrelevant accesses to each fragment. Remote accesses are made by applications running on a given node and accessing objects that are not stored on that node. Local irrelevant accesses are given by local processing incurred when a query accesses a fragment. Each access to a fragment implies a scan to determine objects that satisfy a condition. Irrelevant local access measure the number of local accesses to objects that will not be returned by the query. Intuitively, we want that each fragment be as compact as possible and contain only objects accessed by queries running on the fragment's node.

We introduce the following measure for calculating the fragmentation quality:

$$FPE(C) = FEM + FER \tag{7}$$

$$FEM(C) = \sum_{t=1}^{T}\sum_{i=1}^{M} freq_{ts} * \left| F_i - \left(Acc_{Ct} - \bigcup_{j=1, F_j \in s}^{i-1} F_j \right) \right| \tag{8}$$

$$FER(C) = \sum_{t=1}^{T}\sum_{s=1}^{S} freq_{ts} x \left| Acc_{Ct} - \bigcup_{i=1, F_i \in s}^{M} F_i \right| x \\ x \left| FragCover\left(Acc_{Ct} - \bigcup_{i=1, F_i \in s}^{M} F_i \right) \right| \tag{9}$$

$Acc_{C,t}$ represents the set of objects of class C accessed by query t. $freq_{ts}$ is the frequency of query t running on site s. In (8) s is the site where F_i is located. M is the number of fragments for class C, T is the number of queries and S is the number of sites. The *FEM* term calculates the local irrelevant access cost for all fragments of a class. For a fragment and a query the irrelevant objects are those that a) are not accessed by the query or b) are accessed by the query but are replicas of objects from other fragments already considered in the evaluation. This comes from the natural fact that, when evaluating a query, will be referred only one replica of every object needed by that query. *FER* calculates the remote relevant access cost for all fragments of a class. The second factor expresses the number of remote accessed objects, for a given query t running on a site s. For a given query running on a given node, *FragCover* calculates an optimal covering scheme of the set of remote accessed objects, formed only with remote fragments, so that only one replica of each remote object is considered.

The FER term of the quality measure also reflects the fragmentation behavior in the presence of complex aggregation/association hierarchies. Minimizing the navigation degree from a fragment to its *neighbor* fragments (fragments of the related classes) helps reducing inter-node data transportation when evaluating queries. The *FragCover* factor includes, but is not limited to, the order of magnitude of the navigation degree for the fragments of a given class when evaluating queries over its fragments.

Globally, *FPE* measures how well fragments fit the object sets requested by queries. The fragmentation is better when the local irrelevant costs and the remote relevant access costs are smaller.

By applying the algorithm we obtain the following replication degrees on each class:

Class / Method	No of Instances	Euclid Guided	Euclid Random	Manhattan Guided	Manhattan Random
Dept	12	0%	0%	0%	0%
Grad	13	15%	7%	15%	15%
OrgUnit	5	20%	20%	20%	40%
Prof	20	0%	0%	0%	0%
Researcher	6	0%	20%	0%	0%
Staff	5	0%	0%	0%	0%
Undergrad	61	39%	31%	32%	9%
	Average	*12%*	*13%*	*11%*	*11%*

Table 1. Replication percentage for all classes.

Using the given query access frequency, the fragments above are allocated to 4 distributed sites. For space reasons we do not provide here the application frequencies. A similar example is presented in [11].

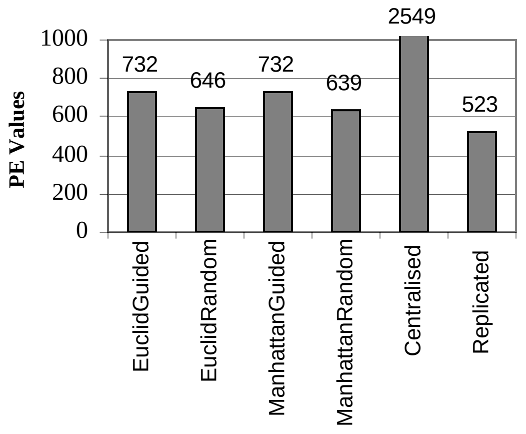

Figure 2 Comparative FPE values: fuzzy clustering, centralised, total replicated databases.

We qualitatively compare the results of our fragmentation method with a centralized and a full replicated database in Figure 3. The centralized version of the database is allocated to node S1, while in the replicated case each node holds a copy of the entire database.

As it can be seen, all the fuzzy fragmentation methods with any of the distance measures generally perform better than the centralised case.

There are two versions of results for each distance measure. One of them is the guided approach where the initial centroids are the most dissimilar objects of the class. This choice influences the initial probability matrix. The other approach for each measure is to take a random standardised initial membership probability matrix. As seen in the figure, the guided versions have slightly worse scores.

This is because the fuzzy behaviour of the algorithm is altered by the initial choice. On the other side, the random initial probability matrix yields fluctuant results and is impossible to know in advance if the obtained fragmentation is good or not. Without a reference average cost it is impossible to say if a FPE cost obtained by the random run is good or not. A good alternative is to first apply the guided method in order to obtain a good average result. Afterwards, the random fuzzy version of the algorithm could be run several times, as long as the obtained FPE values are improved. This approach follows the general idea that a fuzzy clustering algorithm must be repeatedly applied and the best result be retained. The results for *RandomEuclid* and *RandomManhattan* in Figure 2 are the improved results after running the random algorithms several times.

In Figure 3 we compare the results of the fuzzy horizontal fragmentation algorithm with those obtained by using the k-means clustering algorithm. The *Primary* algorithms do not take in consideration the complex class relationships. The complex versions of the k-means algorithm express and handle all inter-class relationships, but without replication [12]. We can see that the fuzzy fragmentation algorithm performs better in all its versions compared to the k-means clustering algorithm.

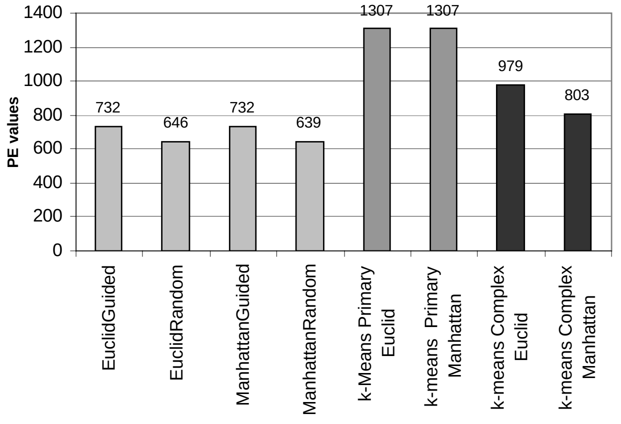

Figure 3 Fuzzy fragmentation vs k-means primary and k-means primary+derived fragmentations.

This is due to the fine replication obtained by assigning the same object to multiple fragments when needed. The most important improvement obtained by using the fuzzy clustering algorithm is the fact that we can quantify the degree of membership of one object to all clusters. When an object is needed in more than one cluster its membership degree will be high for all those fragments. This is an important achievement if we think that fragmentation is driven by user applications and users are not required to express queries with very well separated results so that fragmentation be an easy task.

6. Conclusions

We presented in this paper a new approach in horizontal distributed object oriented database fragmentation. Our fragmentation method uses the c-means fuzzy clustering method for grouping class instances into fragments. When fragmenting a class instance set, often an object is candidate to be placed in multiple fragments, due to the nature of the user applications accessing data. Traditional algorithms take a sharp decision in this case, by placing the conflicting objects in only one of the fragments. We claim that taking in consideration the fuzzy aspect of *object to fragment* membership and placing an object in multiple fragments, when needed, might help improve performance of the obtained fragmentation scheme. We compared our results to those obtained by applying the traditional k-means clustering algorithm – that generates non-overlapped clusters. The obtained fragmentation quality is better than the results of traditional non-intersecting fragmentation schemes.

As future work, we plan to improve the applicability of our combined fragmentation with fine replication scheme. The need to apply several times the fragmentation algorithm in order to improve the intermediary results could possible be eliminated by using a different heuristic for the initial probability matrix.

References:

[2] C.I. Ezeife & K. Barker, A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System, *International Journal of Distributed and Parallel Databases*, *3*(3), 1995, 247-272.

[6] C.I. Ezeife & K. Barker, Horizontal Class Fragmentation for Advanced-Object Modes in a Distributed Object-Based System, *Proc. of the 9th Int. Symposium on Computer and Information Sciences*, Turkey, 1994, 25-32.

[8] L. Bellatreche, K. Karlapalem & A. Simonet, Horizontal Class Partitioning in Object-Oriented Databases, *Lecture Notes in Computer Science*, *1308*, France, 1997, 58–67.

[9] A.S. Darabant & A. Campan, Hierarchical AI Clustering for Horizontal Object Fragmentation, *Proc of Int. Conf. of Computers and Communications*, Oradea, Romania, 2004, 117-122.

[11] A.S. Darabant, A.Campan & O. Cret, Hierarchical Clustering in Object Oriented Data Models with Complex Class Relationships, *Proc of 8th IEEE Int. Conf. on Intelligent Engineering Systems,* Romania, 2004, 307-312.

[12] A.S. Darabant, A. Campan & others, AI Clustering Techniques: A New Approach in Horizontal Fragmentation of Classes with Complex Attributes and Methods in Object Oriented Databases, *Proc of the Int. Conf. on Theory and Applications of Mathematics and Informatics*, Greece, 2004 (to appear).

[14] H. Edelstein, The Challenge of Replication, Parts 1 and 2, *DBMS: Database and Client-Server Solutions*, 1995.

[18] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms (*Plenum Press*, USA, 1981).

[19] E. Bertino & L. Martino, *Object-Oriented Database Systems; Concepts and Architectures* (Addison-Wesley, 1993).