Semi-supervised learning techniques: k-means clustering in OODB Fragmentation

Adrian Sergiu Darabant
Faculty of Mathematics and Computer Science
Babes Bolyai University – Cluj Napoca
1 Kogalniceanu, Cluj Napoca, 3400
Romania
dadi@cs.ubbcluj.ro

Alina Campan
Faculty of Mathematics and Computer Science
Babes Bolyai University – Cluj Napoca
1 Kogalniceanu, Cluj Napoca, 3400
Romania
alina@cs.ubbcluj.ro

Abstract – **Vertical and horizontal fragmentation are central issues in the design process of Distributed Object Based Systems. A good fragmentation scheme followed by an optimal allocation could greatly enhance performance in such systems, as data transfer between distributed sites is minimized. In this paper we present a horizontal fragmentation approach that uses the k-means AI clustering method for partitioning object instances into fragments. Our new method applies to existing databases, where statistics are already present. We model fragmentation input data in a vector space and give different object similarity measures together with their geometrical interpretations. We provide quality and performance evaluations using a partition evaluator function.**

I. INTRODUCTION

Distributed Object Oriented databases aim to minimize performance costs that are incurred by inter-processor communication and data transfers by regrouping related objects into clusters to reduce the number of unnecessary accesses to irrelevant data and by dividing query executions over multiple processors of a network in order to achieve maximum parallelism.

The distribution design of an Object Oriented Database (OODB) should handle data partitioning into a cohesive set of fragments, their assignment to local processing sites and the evaluation and fine-tuning for system performance. Minimizing data transfer has been already considered by almost all distribution techniques [1], either supporting complex characteristics of the object oriented model, or just flat data models. Recently these issues have been considered in ([4], [5], [6], [7], [2]). For fragmenting a class it is possible to use two basic techniques: vertical fragmentation and horizontal fragmentation. In an Object Oriented (OO) environment, horizontal fragmentation distributes class instances into fragments. Each object has the same structure and a different state or content. Thus, a horizontal fragment of a class contains a subset of the whole class extension. Horizontal fragmentation is usually subdivided in primary and derived fragmentation. On the other hand, the vertical fragmentation breaks the logical structure of the class: *attributes and methods*, and distributes them across the fragments. Each fragment contains, in this case, the same objects, but with different subsets of the attributes and methods [9].

We focus in this paper on horizontal object oriented fragmentation by using alternative methods to cluster objects into fragments. The OODB environment supports object oriented data models that are inherently more complex than the relational model. Features like encapsulation, inheritance, class aggregation hierarchy, and association relations complicate the definition of the horizontal class fragmentation. Different approaches have been identified in solving issues regarding fragmentation, which, to a large extent, aim to extend and develop the relational fragmentation and allocation techniques to OODBs. Research papers in the OO area claim that relational fragmentation methods can be applied to the object oriented data model to some extent [8], while others state that starting from relational fragmentation techniques brings a handicap, which is difficult to cover.

Related Work

Karlapalem identifies several issues and criteria for fragmenting distributed OODB horizontally and vertically [1]. Bellatreche et al. [10], Ezeife and Barker [2], Savonnet et. al. [11] propose algorithms for horizontal fragmentation of object classes. Vertical fragmentation is addressed by Bellatreche et. al. [12], Ezeife and Barker [4], Malinowski [13]. Mixed fragmentation is considered in [14] which starts from experimental results and propose a set of heuristics for mixed fragmentation. Ravat [7] proposes a mixed partitioning design approach using the Bond Energy Algorithm (BEA) for both horizontal and vertical object fragmentation. To group attributes, BEA uses attribute affinity measures, which represent the number of times two attributes are accessed together by a method. Predicate affinity measures are used in horizontal partitioning in order to evaluate the distance between two predicates. Ezeife uses minterm predicates identified in queries and object affinity measures with respect to these predicates to achieve primary and derived horizontal fragmentation. She defines fragmentation taxonomy that differentiates between classes with simple attributes/methods and complex attributes/methods. Savonnet et. al. present an OO distribution design methodology based on class dependency graph. They identify groups of co-referenced classes with respect to a set of methods for which parallel execution can be optimized. These groups of classes are partitioned together in fragments. Ghandeharizadeh and Wilhite [15] represent a database as a graph of objects. This graph is broken into subgraphs, which are allocated to nodes by means of a greedy object placement algorithm so that workload in the system meets imposed requirements. Some research papers present evaluation methodologies for fragmentation and/or allocation quality and system performance.

Contributions

We propose new techniques for horizontal fragmentation in object-oriented databases with simple attributes and methods. They rely on AI non-supervised clustering techniques for partitioning classes into sets of similar instance objects, rather than following the traditional minimal predicate set method. We consider the *k-means centroid based* clustering method [3]. Although this is a well-known clustering technique, it has not been used yet in object-database fragmentation, to our knowledge.

The algorithm groups objects together by their similarity with respect to a set of user queries with conditions imposed on data. Similarity (dissimilarity) between objects is defined in a vector space model and is computed using different metrics. As a result, we cluster objects that are highly used together by queries.

In order to improve fragmentation quality, we propose several methods for choosing initial cluster centroids, according to queries semantic.

The paper is organized as follows. The next section presents the object data model and the constructs used in defining the object database and expressing queries. It also introduces the vector space model we use to compare objects, methods for constructing the object characteristic vectors and similarity metrics over this vector space. Section 3 presents our fragmentation algorithm. In section 4 we present a complete fragmentation example over a class hierarchy and we evaluate the quality of our fragmentation schemes by using a variant of the Partition Evaluator [17].

II. DATA MODEL

We use an object-oriented model with the basic features described in the literature [9][16].

Object-oriented databases represent data entities as objects supporting features like inheritance, encapsulation, polymorphism, etc. Objects with common attributes and methods are grouped into classes. A class 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 simple attributes and simple methods. Simple attributes have primitive data types as their domain. Simple methods access only attributes of their class. Every object in the database is uniquely identified by an OID.

Each class can be seen in turn as a class object. Class objects are grouped together in metaclasses [9].

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 i.e. a class can have at most one superclass, 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 the subclasses of C. An OODB 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.

Basic Concepts

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

Given a complex hierarchy H, a *path expression* P is defined as $C_1.A_1. \ldots A_n$, n≥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} ($1 \leq i \leq n$). In our case path expressions always have a length of one and are entry points.

In general a *query* is a tuple with the following structure q=(Target class, Range source, Qualification clause), where:

- *Target class* – (query operand) specifies the root of the class hierarchy over which the query returns its object instances.
- *Range source* – a path expression specifying the source hierarchy.
- *Qualification clause* – logical expression over the class attributes in conjunctive normal form. The logical expression is constructed using simple predicates: *attribute* θ *value* where $\theta \in \{<,>, \leq, \geq, =, \neq\}$.

Let $Q=\{q_1, \ldots, q_t\}$ be 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 simple predicates Q is defined on. Let $Pred(C)=\{p \in Pred \mid p$ impose a condition to an attribute of class C or on the parent class of $C\}$.

Given two classes C and C', where C' is subclass of C, $Pred(C') \supseteq Pred(C)$. 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. We model class predicates this way in order to capture on subclasses the semantic of queries defined on superclasses. For example, given the hierarchy in Fig. 2, a condition "student.grade>5" imposed on Student should normally be reflected on all instances of Grad students as well (graduates are also students).

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, $Pred(C) = \{p_1, \ldots, p_n\}$:

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

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

We obtain from $OCM(C)$ the characteristic vectors for all instances of C. The characteristic vector for object O_i is $w_i = (w_{i1}, w_{i2}, \ldots, w_{in})$, where

$$w_{ij} = \frac{\sum\limits_{l=1..n,\, a_{lj} = a_{ij}} a_{lj}}{m} \quad (2)$$

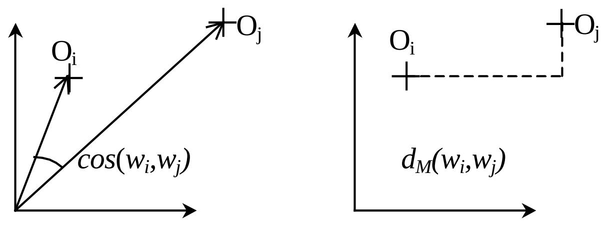

Fig. 1. Geometrical interpretation for the cosine and manhattan similiarities

each w_{ij} is the ratio between the number of objects in C respecting the predicate $p_j \in Pred(C)$ in the same way as O_i and the number of objects in C. We denote the characteristic vector matrix as *CVM*(*C*).

Over the set of characteristic vectors associated to all *C*'s instances we define several *pseudo-metrics*:

$$cos(w_i, w_j) = \frac{\sum_{k=1}^{n} w_{ik} \times w_{jk}}{\sqrt{\sum_{k=1}^{n} (w_{ik})^2} \times \sqrt{\sum_{k=1}^{n} (w_{jk})^2}} \quad (3)$$

$$d_M(w_i, w_j) = \sum_{k=1}^{n} \left| w_{ik} - w_{jk} \right| \quad (4)$$

D_M is the Manhattan distance as defined in [3]. This distance can be calculated on characteristic vectors, as well as on object-condition vectors. Cosine distance can only be applied to characteristic vectors. Given two objects O_i and O_j, we define two similarity measures between them as follows:

$$sim_{\cos}(O_i, O_j) = \cos(w_i, w_j) \quad (5)$$

$$sim_M(O_i, O_j) = 1 - \frac{d_M(w_i, w_j)}{|Inst(C)|} \quad (6)$$

According to the cosine similarity, two objects are more similar as the angle between their characteristic vectors is smaller, i.e. tends to zero. We should note that all characteristic vectors have positive coordinates by definition. As the angle between two vectors is smaller, their components tend to become similar. Translated in object-conditions terms this means that the two compared objects respect the condition set in about the same way, and they should be clustered together. According to the Manhattan (city block) distance/measure two objects are more similar as the component difference between the two vectors is smaller.

III. THE K-MEANS CLUSTERING FRAGMENTATION

The classical k-means algorithm takes the input parameter *k* and partitions a set of *m* objects into *k* clusters so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster's *center of gravity* (*centroid*). First, the k-means algorithm randomly selects *k* of the objects, each of which initially represent a cluster mean or center. For each of the remaining objects, an object is assigned to the cluster to which is the most similar, based on the distance between the object and the cluster centroid. It then computes the new centroid for each cluster and redistributes all objects according to the new centroids. This process iterates until the criterion function converges. The criterion tries to make the resulting *k* clusters as compact and separate as possible.

Our version of the algorithm improves several aspects of the original algorithm with regard to the semantic of object fragmentation. First of all, we implement a variant where we choose as initial centroids the most representative objects in respect with fragmentation predicates, rather than choosing them arbitrarily. At each iteration, if an object should be placed in any of several clusters (same similarity with the centroid), we choose the cluster to which the object has maximum similarity with. We also choose as criterion function the degree of compactness/homogeneity *H* of all clusters. For a given cluster F, this value is the difference between the maximum and minimum similarity of all pairs of objects in F.

$$H(F) = MAXSIM - MINSIM$$
$$MAXSIM = \max\{sim(a,b) \mid (a,b) \in FxF, a \neq b\} \quad (7)$$
$$MINSIM = \min\{sim(a,b) \mid (a,b) \in FxF, a \neq b\}$$

```
Algorithm k-meansFrag is
Input: Class $C$, Inst($C$) to be fragmented, the similarity function $sim:Inst(C)xInst(C)\rightarrow[0,1]$, m=|Inst($C$)|, 1<k≤ m desired number of fragments, $OCM(C)$, $CVM(C)$.
Output: The set of clusters F={$F_1$,…,$F_f$}, where f ≤ k.
Begin
    Centr={$c_1$,…,$c_k$}=InitialCentroids(
                            Inst($C$),$OCM(C)$,$CVM(C)$,k
                            );
    F={$F_i$|$F_i$={$c_i$}, $c_i \in$Centr, i=1..k}; F'=∅;
    // initial object allocation to clusters
    For all objects $O_i$ do
        $F_{candidates}$={$argmax_{centr}$(sim($O_i$,$c_l$),l=1..k))};
        $F_{u*}$ =$argmax_{sim}$(sim($O_i$,$f_c$),$f_c \in F_{candidates}$);
        $F_{u*}$= $F_{u*} \cup${$O_i$};
    End For;
    While F'<>F and H(F)<threshold_value do
        For all $F_i \in$F recalculate centroid $c_i$;
        F'=F;
        For all objects $O_i$ do
            $F_{candidates}$={$argmax_{centr}$(sim($O_i$,$c_l$),l=1..k))};(i)
            $F_{u*}$ =$argmax_{sim}$(sim($O_i$,$F_c$),$F_c \in F_{candidates}$);    (ii)
        $F'_v=F'_v-\{O_i\}$, where $O_i \in F'_v$;
        $F'_{u*} = F'_{u*} \cup \{O_i\}$;
        $F'=F'-\{F'_l | F'_l=\emptyset\}$; // eliminate empty clusters
      End For;
    End While;
End.
```

```
Function InitialCentroids(Inst(C),OCM(C),CVM(C),k) is
Begin
    Centr=∅; n=|Pred(C)|;
    For i=1 to k do
        $c_i=\text{argmin}[d_M(OCM(O_j),u_i)]$, $O_j \notin$ Centr, $i \le n$;   (iii)
        $c_i=\text{argmin}(sim(O_j,Centr))$, $O_j \notin$ Centr, $i>n$;      (iv)
        $Centr=Centr \cup \{c_i\}$;
    End for;
    Return Centr;
End Function;
```

Function *InitialCentroids* chooses the initial centroids as described above. In line (iii) u_i is the identity vector of degree *i*, which has 1 only on the ith position and 0 on the other positions. Each u_i represents the corresponding predicate from *Pred*(*C*). Line (iii) chooses as centroid the closest object to u_i, i.e. the most representative object for that predicate. We note that we can choose this way as many centroids as the number of predicates in *Pred*(*C*). If we need more clusters than |*Pred*(*C*)|, we choose as their initial centroids the objects most dissimilar to the already chosen centroids (line (iv)). We try this way to minimize the impact of "losing" clusters in the following iterations. This occurs when all objects in a cluster relocate to other clusters because the initial centroid is not semantically representative to our set of predicates.

We use in lines (i) and (ii) the similarity of an object O_i with a cluster F_c, defined as:

$$sim(O_i, F_c) = \frac{\sum_{a \in F_c} sim(O_i, a)}{|F_c|} \qquad (\mathbf{8})$$

IV. RESULTS AND EVALUATION

In this section we illustrate the experimental results obtained by applying our fragmentation schemes 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. For evaluation we use a variant of the Partition Evaluator as proposed by Chakravarthy in [17] for vertical relational fragmentation.

The sample object database represents a reduced university database. The inheritance hierarchy is given in Fig. 2. The queries running on the classes of the database are given bellow.

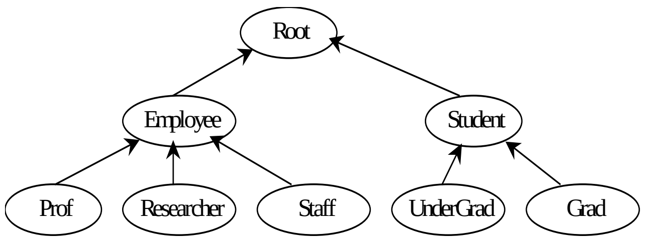

Fig. 2. The database class hierarchy

q_1: This application retrieves all lecturers and teaching assistants.
q_1 = (Prof, Prof, Prof.position in ("lecturer", "teaching assistant"))

q_2: This application retrieves all professors and assistant professors.
q_2 = (Prof, Prof, Prof.position="professor" or Prof.position="assistant proffesor")

q_3: This application retrieves all researchers older than 30 years.
q_3 = (Researcher, Researcher, Researcher.age≥30)

q_4: This application retrieves all researchers having published at least two papers.
q_4 = (Researcher, Researcher, Researcher.count(Reasercher.doc)≥2)

q_5: This application retrieves all graduates with grades less than 4 enrolled at the Computer Science departments.
q_5 = (Grad, Grad, Grad.grade≤4 and Grad.dept like "CS*")

q_6: This application retrieves all graduates older than 30.
q_6 = (Grad, Grad, Grad.age≥30)

We only give here the *Grad* and *Researcher* instances for space and simplicity reasons.

Grad = {{a.Dept, a.Name, a.SSN, a.Born, a.Grade}, {m.age},
G1 {CSR, Bercea Mihai, 1801229203220, 29/12/1980, 8.76}
G2 {CSR, Bleza Ovidiu, 2850912244171, 03/09/1971, 3.00}
G3 {M, Caciula Anamaria, 2790429080061, 29/04/1979, 9.07}
G4 {SD, Catana Florin, 2850912244353, 01/01/1970, 3.00}
G5 {PC, Cerba Dan, 2850912244296, 24/02/1973, 7.00}
G6 {MI, Cigher Simona, 2800807125829, 07/08/1980, 9.06}
G7 {MI, Cindrea Ioana, 2800924060021, 24/09/1980, 3.00}
G8 {CSR, Cioara Danut, 1760829054671, 29/08/1976, 8.60}
G9 {MI, Cosma Maria, 2810320060017, 20/03/1981, 3.00}
G10 {CSR, Damian Mircea, 1750616323929, 22/09/1976, 3.00}
G11 {CSM, Dani Iosif, 1761203120669, 03/12/1976, 3.00}
G12 {CSM, Darvas Laszlo, 1810413055099, 13/04/1981, 3.00}
G13 {CSR, Duhanes Dan, 2850912244193, 01/01/1970, 3.00} }
Researcher = {{a.OrgUnit, a.Name, a.SSN, a.Born, a.Doc}, {m.age},
R1 {Algebra, Morar Oana, 2651005123456, 05/10/1973, {T4,T18,T32}}
R2 {InfSyst, Cobarzan Claudiu, 1470120123456, 20/01/1979, {}}
R3 {ProgrMeth, Grebla Horia, 1720501123456, 01/05/1979, {T19,P20}}
R4 {InfSyst, Sterca Adrian, 1511123123456, 23/11/1971, {P14,P31,P32}}
R5 {Calculus, Tofan Daniel, 1560404123456, 04/04/1976, {T16,P32}}

R6 {InfSyst, Zeng Ioan, 1560331123456, 31/03/1972, {T17}} }

The fragments obtained for *Grad* using algorithm k-meansFrag and cosine as similarity measure are: $F_1 = \{G_5\}$, $F_2 = \{G_9, G_7, G_{12}, G_{11}, G_{10}\}$, $F_3 = \{G_6, G_3, G_1, G_8\}$, $F_4 = \{G_4, G_{13}, G_2\}$

The fragments obtained for *Researchers* using algorithm k-meansFrag and cosine as similarity measure are: $F_1 = \{R_6, R_2\}$, $F_2 = \{R_1, R_3, R_5, R_4\}$

Using the given query access frequency and other input data, the fragments above are allocated to four distributed sites. We use a simple allocation scheme that assigns fragments to the sites where they are most needed. Query frequency is presented in TABLE 1.

TABLE 1. Access Frequencies of queries at distributed sites

freq(q,s)	S1	S2	S3	S4	Class
q1	10	20	5	20	Prof
q2	0	10	5	25	Prof
q3	20	10	15	10	Researcher
q4	15	10	25	20	Researcher
q5	25	20	0	20	Grad
q6	30	25	20	10	Grad

First we qualitatively compare the cosine k-means (best centroid choice) fragmentation with a fully replicated database and a centralized database allocated on one of the sites.

TABLE 2. Allocation of Fragments to Distributed Sites

Class	S1	S2	S3	S4
Grad	F_3,F_4	F_2		F_1
Prof	F_3		F_1	F_2
Researcher		F_1,F_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 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.

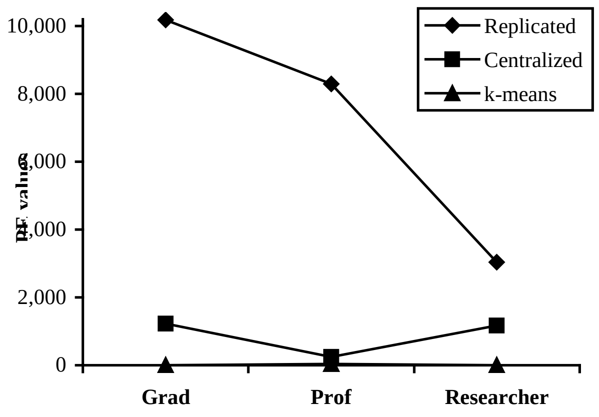

Fig. 4. Comparative PE for k-means, full replication and centralized case.

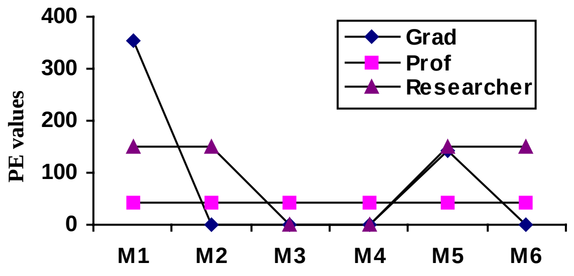

Fig. 3. Comparative quality measures for each class.

TABLE 3. Fragmentation methods legend

M1	k-means cosine – random centroid choice (RCC)
M2	k-means cosine – best centroid choice (BCC)
M3	k-means Manhattan on object-conditions RCC
M4	k-means Manhattan on object-conditions BCC
M5	k-means Manhattan on charact. vectors RCC
M6	k-means Manhattan on charact. vectors BCC

$$PE(C) = EM^2 + ER^2 \qquad (9)$$

where:

$$EM^2(C) = \sum_{i=1}^{M}\sum_{t=1}^{T} freq_{ts}^2 * |Acc_{it}| * \left(1 - \frac{|Acc_{it}|}{|F_i|}\right) \qquad (10)$$

$$ER^2(C) = \sum_{t=1}^{T} \min\left\{\sum_{s=1}^{S}\sum_{i=1}^{M} freq_{ts}^2 * |Acc_{it}| * \frac{|Acc_{it}|}{|F_i|}\right\} \qquad (11)$$

In (10) *s* is the site where F_i is located, while in (11) *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 q_t from the fragment F_i. The smaller *PE* is, better fragmentation quality we have.

The test results show that our fragmentation performs better in terms of *PE* than the centralized and fully replicated cases.

Our variant for initial centroid choice leads to better

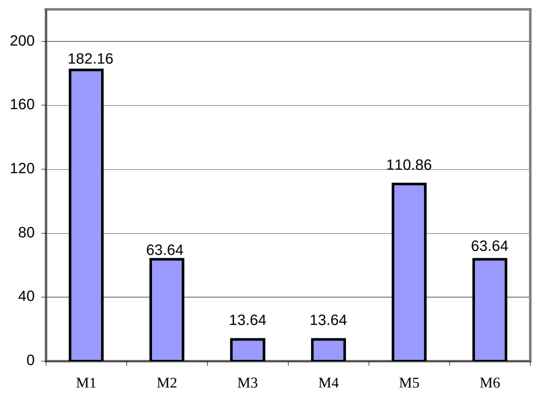

Fig. 5. Comparative PE values for our fragmentation methods.

fragmentation than the random centroid choice (M2, M4, M6 are better than M1, M3, respectively M5) as shown in Fig. 4 and Fig. 5.

When it comes to similarity measures, both cosine and Manhattan distinguish objects that do not respect predicates in the same way, but the differentiation refinement has different granularity for each method. As a consequence, resulting fragments are not always similar for the same input data. Also, the experiments show that no measure behaves optimally in all cases/classes. For example, there are particular data distributions, with perfectly separable clusters, where cosine measure is not capable of distinguishing any clusters. We have identified these particular cases and we are investigating solutions for handling them. Fig. 5 presents the comparative global PE values for all our methods. Globally the Manhattan measure applied on object-conditions outperforms the other measures.

V. CONCLUSIONS AND FUTURE WORK

In this work we prove that AI clustering methods can be effectively used in object-oriented fragmentation and we aim to extend the proposed approach to class models with complex aggregation (association) hierarchies and complex methods.

Currently, we are investigating new similarity measures with improved discriminative power. We are also working on alternative evaluation techniques for fragmentation quality. We also think that we can use our clustering methods to help solving dynamic fragmentation - by capturing the semantic of potential future query changes into the initial fragmentation, so that the fragments can be adapted to those changes with smaller costs.

VI. REFERENCES

[1] K. Karlapalem, S. B. Navathe and M. M. A. Morsi, “Issues in distribution design of object-oriented databases”, in *M. Tamer Ozsu, U. Dayal, P. Valduriez, editors, Distributed Object Management*, Morgan Kaufmann Publishers, 1994, pp 148-164.

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

[3] J. Han, and M. Kamber, *Data Mining: Concepts and Techniques*, The Morgan Kaufmann Series in Data Management Systems, 2000.

[4] C. I. Ezeife and K. Barker, “Vertical Class Fragmentation in a Distributed Object Based System”, TechnicalReport 94-03, University of Manitoba, Canada, 1994.

[5] K. Karlapalem and Q. Li, “Partitioning Schemes for Object-Oriented Databases”, in *Proceedings of the Fifth International Workshop on Research Issues in Data Engineering-Distributed Object Management*, Taiwan, 1995, pp 42–49.

[6] K. Karlapalem, Q. Li and S. Vieweg, “Method Induced Partitioning Schemes in Object-Oriented Databases”, in *Proceedings of the 16th Int. Conf. on Distributed Computing System* (ICDCS’96), Hong Kong, 1996, pp 377–384.

[7] S. Ravat, “La fragmentation d’un schema conceptuel oriente objet”, in *Ingenierie des systemes d’information* (ISI), 4(2), 1996, pp 161–193.

[8] C. I. Ezeife and K. Barker, “Horizontal Class Fragmentation for Advanced-Object Modes in a Distributed Object-Based System”, in *Proceedings of the 9th International Symposium on Computer and Information Sciences*, Antalya, Turkey, 1994, pp 25-32.

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

[10] L. Bellatreche, K. Karlapalem and A. Simonet, “Horizontal Class Partitioning in Object-Oriented Databases”, in *Lecture Notes in Computer Science*, volume 1308, Toulouse, France, 1997, pp 58–67.

[11] M. Savonnet et. al., “Using Structural Schema Information as Heuristics for Horizontal Fragmentation of Object Classes in Distributed OODB”, in *Proc IX Int. Conf. on Parallel and Distributed Computing Systems*, France, 1996, pp 732-737.

[12] L. Bellatreche et. al., “Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods”, in *Proc. of the 7th Int. Workshop on Database and Expert Systems Applications*, 1996, pp 15-21.

[13] E. Malinowski, *Fragmentation Techniques for Distributed Object-Oriented Databases*, Thesis, Univ. of Florida, 1996.

[14] F. Baiao and M. Mattoso, “A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases”, in *Proc. Of the 9th Int. Conf. on Computing Information*, Canada, 1998, pp 141-148.

[15] S. Ghandeharizadeh and D. Wilhite, “Placement of Objects in Parallel Object-Based Systems”, Technical Report 94-589, Department of Computer Science – University of Southern California, 1994.

[16] M. Atkinson et. al., “The Object Oriented Database Manifesto”, in *Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases*, 1989.

[17] S. Chakravarthy, J. Muthuraj, R. Varadarajan and S. B. Navathe, “An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis”, in *Distributed and Parallel Databases*, 2(1), 1993, pp 183-207.