Measuring the Performance of Database Object Horizontal Fragmentation Schemes

C.I. Ezeife*
School of Computer Science
University of Windsor
cezeife@cs.uwindsor.ca
http://www.cs.uwindsor.ca/users/c/cezeife

Jian Zheng
School of Computer Science
University of Windsor

Abstract

A horizontal fragment of a database class in an object-oriented database system contains subsets of its instance objects (or class extents) reflecting the way applications are accessing database objects. Allocating well-defined fragments of classes to distributed sites has the advantage of minimizing transmission costs of data to remote sites as well as minimizing retrieval time of data needed locally. All algorithms so far proposed in the literature for defining horizontal fragments of database objects are based on information from earlier static requirements analysis. Thus, a re-fragmentation of the system is needed when application access and schema information have undergone sufficient changes. In this paper, we provide a technique for measuring the performance of object horizontal fragments placed at distributed sites. This work provides a platform for dynamic object horizontal fragmentation and for comparing object horizontal fragmentation schemes.

Keywords: Object-oriented databases, Horizontal fragmentation, Distribution, Performance analysis

1 Introduction

The importance of fragmentation of database entities and subsequent allocation to distributed sites (relations or classes) have been argued by many works including [NR89, OV91, EB95, NKR95, BSS96, EB98]. All distributed database design work so far in the literature are static because input to the design process, which include queries accessing database and their frequencies as well as the database schema, are obtained from an earlier full requirements analysis. The implication of the static design is that if fragmentation and allocation had been based on application and schema information which later changed, the distributed design procedure comprising first fragmentation followed by allocation may need to be re-done when the system performance drops lower than a certain limit or threshold. It is surprising that only very limited attention has been paide to the problem of measuring the performance of a horizontally distributed system given that horizontal fragmentation is more commonly used in the commercial sector. The only effort in this direction is the work by Chakravarthy *et al.* in [CMVN93] which presents an objective function for measuring the performance of vertical fragments of relations. This work was

*This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

extended to measure the performance of vertical fragments of objects in object-oriented database system [EB98]. The purpose of this paper is to present an algorithm for measuring the performance of a given object horizontally fragmented system. The result from the work can be used to determine (1) a system performance threshold, (2) system performance degradation due to changes in application and schema requirements, (3) when a re-design of the distribution process should be done, (4) the goodness of object horizontal fragmentation scheme and ultimately how to trigger and conduct dynamic re-fragmentation of objects.

1.1 Data Model

Object -oriented databases represent data entities as objects supporting features of encapsulation. Thus database objects have a set of attributes ($\mathcal{A}$) and a set of methods ($\mathcal{M}$) for accessing these attributes. Objects with common attributes and methods belong to the same class C with a class identifier K. Every object has an object identifier corresponding to its class identifier prefixed to a unique object number. All objects of a class form its instance objects ($\mathcal{I}$) or class extents. Thus, we can represent a class as an ordered relation C = (K,$\mathcal{A}$,$\mathcal{M}$,$\mathcal{I}$). For example, a class *Person* has class identifier Person. Its attributes are social security number, name, age, methods getname, getage etc. The instances of the class have names, John, Mary etc. Object-oriented database systems also support inheritance whereby a class B called subclass of class A can be defined as a specialization of class A. Class A is called the superclass of class B and class B can inherit all attributes and methods of its superclass A but not all of its extents. Thus if class B is a subclass of class A, A is represented as $C^A = (KA, A^A, M^A, I^A)$ while B can be represented as $C^B = (KB, A^A \cup A^B, M^A \cup M^B, I^B)$. For example, in a database with class *Person* and *Student*, *Student* is a subclass of class *Person*.

Thirdly, object-oriented database systems support class composition hierarchies. This means that an attribute of a class may have another database class as its domain or part of it. This type of attribute is called complex attribute while simple attributes have primitive data types (e.g., integer, character string) as their domain. For example, a class *Student* may have attributes, studentid, dept and amountowing. It can be seen that both studentid (a character string) and amountowing (real) are simple attributes while dept (another class with attributes "name" and "profs") is a complex attribute. If a class D is part of a class E, then attributes and methods of class E logically include attributes and methods of class D. The representation for class E is then $C^E = (KE, A^E \subseteq A^D, M^E \subseteq M^D, I^E)$ for all classes D, whose domain are assumed by any attribute of class E. With this representation, the object-oriented database schema can be defined as a set of its classes, capturing the features of encapsulation, inheritance and class composition hierarchies. A simple method is a method accessing only attributes of its class while a complex method is a method accessing attributes of another class. classes. For example, in the above definition of class E, all $M \cup M^E$ are simple methods while all $M \cup M^D$ are complex methods. Other method nesting hierarchies may also be supported if the object-oriented system allows a method of a class A that is neither an ancestor class nor a contained (part-of) class of another class B to be invoked by methods of class B. Each horizontal fragment (C_h) of a class contains all attributes and methods of the class but only some instance objects ($I' \subseteq I$) [EB95]. Vertical fragments of a class contain subsets of attributes and methods but all instance objects of the class, while hybrid fragments contain subsets of both attributes, methods and instances of the class.

1.2 Related Work

Ceri, Negri and Pelagatti [CNP82] define application queries in terms of boolean predicates which are collected into sets of minterms that form horizontal fragments of relations. Ceri et al. in [CSW83] further uses directed links drawn between relations where the relation at the tail of a link is called the owner of the link while that at the head is called the member. The link between the owner and member relations are used to capture the equijoin relationships in the schema. In horizontally fragmenting relations, primary fragmentation is performed on all owner relations while derived fragmentation is performed on all member relations of links according to a selection operation specified on its owner relation. Ozsu and Valduriez [OV91] give a formal definition of simple predicates as $P_j : A_i \ \theta \ Value$ (e.g. Studentgpa $\geq$ 4.0) and that given a set of simple predicates $P_{ri} = \{P_{i1}, P_{i2}, ..., p_{im}\}$, the set of minterm predicates $M_i = \{m_{i1}, m_{i2},...,m_{iz}\}$ is defined as $M_i = \{m_{ij} | m_{ij} = \wedge P^*_{ik}\}$, $P_{ik} \in P_i$, $1 \leq k \leq m$, $1 \leq j \leq z\}$ where $P^*_{ik} = P_{ik}$ or $P^*_{ik} = \neg P_{ik}$. For example, with P_1 as studentgpa $\geq$ 4.0 and P_2 as major ="Math", the defined minterms are $m_1 = P_1 \wedge P_2$, $m_2 = P_1 \wedge \neg P_2$, $m_3 = \neg P_1 \wedge P_2$, $m_4 = \neg P_1 \wedge \neg P_2$. They also propose an iterative algorithm (COM_MIN) that basically eliminates minterm predicates that are either redundant or meaningless.

Object-oriented database systems that support some form of distribution or based on client/server model include ITASCA [IS95], GOBLIN [KPB94], THOR [LAD+96] and for improved performance of such systems, techniques for proper fragmentation of objects are needed. Thus, Ezeife and Barker in [EB95] present a set of horizontal fragmentation algorithms (called object oriented horizontal fragmentation scheme OOHF), for four types of class models including those with both simple attributes and methods, simple attributes and complex methods, complex attributes and simple methods, and complex attributes and methods. Since the model with complex attributes and methods is the most encompassing, this algorithm is summarized next. The algorithm defines the link graph for all classes in the database using the inheritance hierarchy, class composition hierarchies and method nesting hierarchies. From the inheritance hierarchy, starting from the leaf classes (most specialized subclasses) which are at the head of links and called owners of the links, the appropriate links are connected to their immediate superclasses and the process continues untill the root class is encountered. From the class composition hierarchies, every domain class (or complex attribute) forms head of a link which is connected to all classes it is part-of. Thus, domain classes are owners of links while classes containing them are members of the links. Secondly, the algorithm now proceeds to define primary horizontal fragments of the class using predicates from queries accessing this class directly. Thirdly, derived fragments of this class are defined using all primary fragments of all its owner classes from all link graphs to accommodate inheritance, class composition hierarchies and possibly method nesting hierarchies. Finally, the sets of primary and derived fragments of the class are combined following the rule that every derived fragment should be merged with a primary fragment it has the highest affinity count with. However, if a class has no primary fragments, its derived fragments become its final fragments ensuring all instance objects are placed in some fragment. Savonnet and Yetongnon [SY95] start with class dependency graph (CDG) which represent the inheritance, class composition and method nesting hierarchies. From the CDG, they construct a set of partition trees and each class of the object database belongs to only one partition tree. The algorithm then horizontally fragments classes of each partition by fragmenting the root class of the tree and propagating this fragmentation through other descendant classes of the tree. Thus primary fragmentation is defined on root class of the partition tree but derived fragmentation is defined on other classes. Savonnet *et al.* [STY98] present Fragtique– a graph based partitioning method similar to the approach discussed above in [SY95] . Addition here is that methods used to partition roots of partition trees and the concept of partitioning methods into partial method for parallel execution are developed. Flow measures are also used to evaluate the quality of results of the fragmentation process and for allocating class fragments among sites.

Ravat and Zurfluh [RZ95] present a technique for horizontally fragmenting relation using predicate usage matrix. The Bond Energy algorithm [MSW72] is used to cluster the predicate affinity matrix and binary partition used to partition the predicates. The horizontal fragments are defined by ORing the predicates with the same attributes in each partition while those with different attributes are ANDed. An additional fragment which is the negation of disjunction of predicate terms is included. Bellatreche, Karlapalem and Simonet in [BKS97] define primary horizontal fragmentation on simple queries while derived horizontal fragmentation is defined on component queries. A simple query uses attribute_name $< operator >$ value where the domain of attribute_name is primitive type. A component predicate has the attribute name as a branch of class composition hierarchy called path. A component query uses component predicates. The primary horizontal fragmentation algorithm builds the predicate usage matrix for the class using all queries. The predicate affinity matrix is constructed to include the other non-numerical values: $\Rightarrow$ is the value if predicate p_i on the row implies predicate p_j on the column, $\Leftarrow$ means that predicate p_j on the column implies predicate p_i on the row while the value "*" means that the row predicate p_i is "similar" to the column predicate p_j because they are both jointly used with a third predicate p_l. An affinity graph with nodes as the predicates and edges as the affinity values is generated. From the affinity graph, they define clusters of predicates which correspond to graphs with cycles. Predicate terms are defined from sets of predicates and these correspond to horizontal fragments. Chakravarthy *et al.* in [CMVN93] presents an objective function called Partition Evaluator (PE) for evaluating the goodness of any vertical partitioning algorithm. The PE is used to measure the total costs of processing all applications at all distributed sites if each access to a data fragment amounts to a unit cost. It is assumed that there are no data redundancies and following following fragmentation, the fragments have been allocated to sites where they are most needed. The two components of PE are irrelevant local method access cost which minimizes the square error for a fixed number of fragments and assigns a penalty factor whenever irrelevant attributes are accessed in a particular fragment. The mean vector V_i for fragment i is given by $V_i = (1/n_i)\sum_{j=1}^{n_i} A_{ij}$ where n_i is the number of attributes in fragment i and A_{ij} is the attribute vector for the jth attribute in fragment i. The difference vector for attribute vector A_{ij} is $(A_{ij} - V_i)$ and the square error of fragment i is the sum of squares of the lengths of the difference vectors of all the attributes in fragment i given by $e_i^2 = \sum_{j=1}^{n_i}(A_{ij} - V_i^T)(A_{ij} - V_i)$. The square error for all m fragments generates is: $E_m^2 = \sum_{i=1}^{m} e_i^2$ and this equation simplifies to $E_m^2 = \sum_{i=1}^{m}\sum_{t=1}^{T}[q_t^2 * |s_i t| * (1 - |s_i t|)/n_i])$ where s_{it} is the set of attributes contained in fragment i that the transaction t accesses. It is empty if t does not need fragment i. A lower value of E_m^2 means a lower penalty cost of irrelevant local attribute access. The second component of the PE computes for a set of applications running on a fragment, the ratio of the number of remote attributes to be accessed to the total number of attributes in each of the remote fragments. This summed over all the fragments and applications to obtain $E_R^2 = \sum_{t=1}^{T} minimum(\sum_{k\neq i}[q_t^2 * |R_{itk}| * |R_{itk}|/n_{itki}])$ where q_t is the frequency of application t for t=1, 2,..., T. $R_i tk$ is the set of relevant attributes of fragment k remotely needed by application t while running on fragment i and $|R_i tk|$ is its cardinality. Others are n_{itk}^r, the total number of attributes in remote fragment k accessed by application t while running on fragment i Thus $PE = E_m^2 + E_R^2$. In [EB98], Ezeife and Barker extend the relational vertical Partition Evaluator (PE) to handle measuring the costs of local and remote accesses incurred by the object vertical fragments.

1.3 Contributions

This paper contributes by first presenting a modified objective function or partition evaluator (PE) suitable for measuring the irrelevant local object instance cost (E_M^2) and relevant remote object instance cost (E_R^2) for horizontally fragmented object classes which are placed at distributed sites. Since the methodologies for horizontal fragmentations differ, this work presents an algorithm that transforms only the input to the fragmentation schemes and thus is object horizontal fragmentation scheme independent. The algorithm inputs the set of horizontal fragments generated by the shemes as well. Two major components of the performance analyzer proposed are (1) an algorithm, object horizontal partition evaluator (OHPE) which counts the amount of local irrelevant access made to objects locally by applications as well as the amount of remote relevant accesses made to objects by applications. The overall performance penalty for the fratgments, partition evaluator is obtained as the sum of these two performance penalties. The second component of the performance analyzer is an algorithm that computes the input sets of all instance objects accessed by applications running at all sites on each database class. The access of objects of each class includes accesses through applications running directly on objects of this class, applications accessing these objects indirectly through their descendant classes, containing classes and complex method classes. This second algorithm, called GetAISet (for getting the application object instance sets) computes this essential input used by OHPE in (1) above.

The work presented here provides a basis for comparing object horizontal partitioning schemes and for initiating dynamic object horizontal fragmentation.

1.4 Outline of the Paper

The rest of the paper is organized as follows. Section 2 presents the architecture of the horizontal object distributed database design and the modified object horizontal partition evaluator (OHPE). Section 3 presents the input transformation algorithm (GetAISet). Section 4 presents an example with results while section 5 presents conclusions.

2 Distributed Object Horizontal Database Design Architecture

This section presents the components and architecture of distributed object horizontal database design which indicates the input, output, the algorithms and the flow of operations in the object horizontal performance analyzing algorithm. Section 2.1 discusses the components of the design while section 2.2 presents the modified objective function for measuring the performance of fragments produced by an object horizontal fragmentation scheme.

2.1 Components of the Design

Figure 1 shows the architecture of the distributed object horizontal database design. Three algorithms can be identified in this design namely:

1. The object horizontal fragmentation scheme used to produce the fragment of the database objects, e.g., OOHF [EB95], Bell's [BKS97].

2. The GetAISet algorithm, which is used to transform the available input data to obtain application object instance sets. The algorithm computes the application instance set $[AI]_{qc}$ for every application q at each site c. These application object instance sets form the main input to the third algorithm OHPE discussed next. For object-oriented databases, this is adjusted to account for access of a class instance object by applications running on it descendant classes, its containing classes and classes accessing it through complex methods.

3. OHPE is the object horizontal partition evaluator algorithm. Input to this algorithm is application object instance set from the horizontal distribution (fragmentation and allocation). OHPE measures the costs of local and remote accesses incurred by fragments placed at distributed sites. The

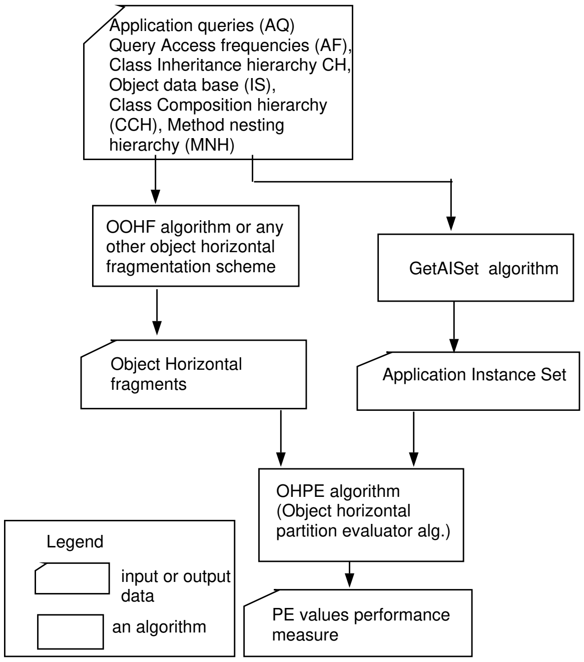

Figure 1: Distributed object horizontal database design architecture

PE value is taken as the sum of the two costs so that higher PE value means higher performance penalty and thus lower performance by the scheme.

The algorithms are shown in rectangles in the figure while the solid input and output data are shown in left-corner trimmed rectangles. Both algorithms (2) and (3) above are contributions of this paper. Section 2.2 discusses the algorithm proposed for object horizontal partition evaluator while the GetAISet algorithm is discussed in section 3.

2.2 Object Horizontal Partition Evaluator (OHPE)

Every application processing cost consists of its local processing and remote processing costs. It is assumed there are no data redundancies and following fragmentation, the fragments have been allocated to sites where they are most needed. Whereas the relational vertical partition evaluator computes the processing costs of distributed fragments using an application/attribute matrix which counts the number of times applications access attributes at distributed sites, the object vertical partition evaluator [EB98] computes the costs by beginning with an application/method matrix since by encapsulation, all accesses to attributes are via its methods. This paper computes the processing cost of horizontally distributed objects by beginning with application instance object sets. The challenge lies with obtaining these application instance object sets. Thus, partition evaluator PE is given as:
$PE = E_M^2 + E_R^2$, where $E_M^2 = \sum_{i=1}^{M} \sum_{t=1}^{T} [q_{tc}^2 * |S_{itc}|(1 - |S_{itc}|/n_i)]$ and M is the number of fragments, $|S_{itc}|$ is the set of instance objects contained in fragment F_i that transaction t accesses at site c. It is

empty if t does not need fragment F_i. The q_{tc} is the frequency of application t for t=1, 2, ..., T at site c and n_i is the total number of instance objects contained in fragment F_i. Relevant remote instance access cost of PE is given by:
$E_R^2 = \sum_{t=1}^{T} \min(\forall_i^M \sum_{\forall_{k \neq i}} q_{tc}^2 * |R_{itk}| * |R_{itk}|/n_{itk}^{rem})$ where $|R_{itk}|$ is the number of relevant instances in fragment k accessed remotely with respect to fragment i by transaction t at site c, n_{itk}^{rem} is the total number of instances that are in fragment k accessed remotely with respect to fragment i by transaction t.

For measuring the performance of a scheme, the object horizontal partitioning scheme is first used to compute horizontal fragments which are allocated at distributed sites (e.g., 3 sites). Then, both the application object instance sets and the fragments constitute input to the OHPE algorithm which measures the PE value for the system. Each PE value corresponds to the total penalty cost incurred by each scheme through both local irrelevant access costs and remote relevant access costs. A lower PE value means less performance penalty and thus, a better performance. The algorithm for computing the object horizontal partition evaluator OHPE is given as Figure 2.

3 The Input Computation algorithm (GetAISet)

From the components of the object horizontal database design architecture, the OHPE algorithm which is the object horizontal partition evaluator presented in section 2.2 above requires the input of application object instance set and this information may be too detailed to be available during fragmentation. This requires that the needed information be computed from the available data sets. This section presents an algorithm called GetAISet (get application object instance sets), which is responsible for defining the application instance sets given the set of user queries accessing class C_i and its descendant classes, all of the classes containing this class in the class composition hierarchies and all of the classes using their complex methods to access this class' methods from method nesting hierarchies. Application frequency matrix that contains the frequency of each application at each distributed site is also available. If we were to compute the application object instance matrix of a class, we would normally begin by defining for each query accessing the class as matrix rows and each instance object of class as matrix column, assigning 1 to matrix element (q,i) if application q accesses object i. However, since classes may have hundreds of instance objects, this approach is not very realistic. As an alternative, we can begin by computing application/predicate matrix. There are only a countable number of predicates accessing a class and the matrix has applications as rows and predicates as columns. A value of "1" is assigned to the matrix cell of application q and predicate p if application q uses predicate p and "0" otherwise.

The set of predicates P^{C_i}, accessing a class C_i is defined as the union of all predicates (P^{C_i}) from the queries accessing the class directly, predicates ($P_{des}^{C_i}$)of all queries of all the descendant of this class that access this class, and predicates ($P_{cont}^{C_i}$) of all its containing classes accessing the class, and

Algorithm 2.1 *(Algorithm (OHPE-Computes the object horizontal PE values)*

Algorithm OHPE
Input: $[AI]_{tc}$: Application object instance set of applic. t at site c
F^{C_i}: set of horizontal fragments of class C_i
Q^{C_i}: set of transaction or queries on class C_i
$AF_{tc}^{C_i}$ access frequencies of transactions on class C_i at site c
Output: E_M^2: Irrelevant Local Method Access Cost
E_R^2: Relevant Remote Access Cost, PE: partition evaluator value

```
begin
// Compute the E_M^2 component of PE //
E_M^2 = {} // Initialize //
    for i=1 to number of fragments
        begin
        for t=1 to number of transactions
        for c=1 to number of sites
            begin
            S_itc = [AI]_tc ∩ [Fragment]_i
            // S_itc is the set of instance objects in both //
            // applications instance set ot t at site c and Fragment i //
            n_i = [Fragment]_i // n_i is the number of objects in fragment i //
            E_M^2 = E_M^2 + (AF_tc)^2 * |S_itc| * (1 - |S_itc|/|n_i|)
            // Note that AF_tc is the application frequency of transaction t at site c //
        end // for c //
        end //for t //
    end // for i //
    // Next compute the E_R^2 component of PE. Initialize variables. //
    for t=1 to numbers of transactions
        begin
            for i=1 to number of fragments
            begin
            for k=1 to number of fragments
            begin
                if k ≠ i then
            begin
                R_itk = ([AI]_tc - [Fragment]_i) ∩ [Fragment]_k
                n_itk^rem = number of elements in [fragment]_k
                E_Rtemp^2 = E_Rtemp^2 + (AF_tc)^2 * |R_itk| * |R_itk|/|n_itk^rem|)
            end // end if //
            if i=1
            then E_Rmin^2 = E_Rtemp^2 else if E_Rtemp^2 < E_Rmin
            then E_Rmin^2 = E_Rtemp^2
            end // for k //            end // for i //
            E_R^2 = E_R^2 + E_Rmin^2
            end // for i //
            PE = E_M^2 + E_R^2
    end // OHPE //
```

Figure 2: The Object Horizontal Partition Evaluator Algorithm

predicates ($P^{C_i}_{cmeth}$) of all its complex method classes. Thus, the set of predicates for class C_i, P^{C_i} is re-defined as:

$P^{C_i} = P^{C_i} \cup P^{C_i}_{des} \cup P^{C_i}_{cont} \cup P^{C_i}_{cmeth}$

Similarly, all queries accounting for these predicates make up the queries accessing the class C_i and are defined as follows:

$Q^{C_i} = Q^{C_i} \cup Q^{C_i}_{des} \cup Q^{C_i}_{cont} \cup Q^{C_i}_{cmeth}$

The approach entails defining an application predicate matrix by specifying the application Q^{C_i} as row labels and the predicates P^{C_i} as column labels. A value of "1" is assigned to the cell (q, p) if query q accesses predicate p and "0" otherwise. Then, for each application q, and for each site c, the application/Instance set $[AI]_{qc}$, which is the set of instance objects accessed by application q, is computed as:

$I \in [AI]_{qc}\ iff\ I \in C_i \wedge P \in P^{C_i} \wedge q \in Q^{C_i} \wedge \exists(p|access(q,p) = 1) \wedge I \in P)$.

The algorithm GetAISet, accepts the queries accessing class C_i, the inheritance, class composition and method nesting hierarchies, as well as the queries accessing descendant, containing and complex method classes. Since queries are represented as a set of predicates, the predicates from the queries are ready to be used by the algorithm. Output from the algorithm are the set of instance objects accessed by each application and the formal algorithm is presented as Figure 3.

4 An Example

This section shows how to compute the performance of the object horizontal fragmentation schemes with the proposed object horizontal partition evaluator (OHPE) algorithm. The working of the scheme is demonstrated using the object database example horizontally fragmented by an object oriented horizontal fragmentation scheme (OOHF) in [EB95]. A comparison is provided with a case where data is not fragmented at all but replicated at different sites on one hand and the case where data is neither fragmented nor replicated but shipped to remote sites on request.

The sample object database represents a university database as in Figure 4 . The inheritance hierarchy is given as Figure 5.

The five queries running on the classes of the database and their predicates are given below.

q_1**:** This application groups grads according to their area of specialization which is determined by the name of their supervisor. The methods used are defined on class **Grad** and the predicates are:

$\{P_1$: supervisor = "Prof John West",P_2: supervisor = "Prof Mary Smith"$\}$

Algorithm 3.1 *(GetAISet - Computes the instance objects accessed by each application)*

Algorithm GetAISet
Input: queries on each class $\{Q_m^{C_i}\}$, queries on C_i's descendant classes $\{Q_{des}^{C_i}\}$
queries on C_i's containing classes $\{Q_{cont}^{C_i}\}$, queries on C_i's complex method classes $\{Q_{cmeth}^{C_i}\}$
access frequency of queries on C_i $\{AF_m^{C_i}\}$, instance objects of C_i $\{I^{C_i}\}$
Output: AISets: Application/Instance sets for applications on C_i
begin
for each q $\in Q^{C_i}$
    begin
        // Compute all predicates P^{C_i} from q as //
        $P_{C_i} = P_{C_i} \cup P^q$
    end //of for each q, Next include predicates of queries accessing descendant classes//
for each q $\in Q_{des}^{C_i}$
    begin
        if access(q, C_i) = 1 then
        begin
        $P_{C_i} = P_{C_i} \cup P^q$, $Q_{C_i} = Q_{C_i} \cup q$
        end // of if then//
    end // of for each, Next include predicates of queries accessing containing classes//
for each q $\in Q_{cont}^{C_i}$
    begin
        if access(q, C_i) = 1 then
        begin
        $P_{C_i} = P_{C_i} \cup P^q$, $Q_{C_i} = Q_{C_i} \cup q$
        end // of if then//
    end // of for each Next include predicates of queries accessing complex method classes//
for each q $\in Q_{cmeth}^{C_i}$
    begin
        if access(q, C_i) = 1 then
        begin
        $P_{C_i} = P_{C_i} \cup P^q$, $Q_{C_i} = Q_{C_i} \cup q$
        end // of if then//
    end // of for each Now compute application/instance sets //
for each q $\in Q^{C_i}$
    begin
    for each p $\in P^{C_i}$ And p $\in q$
    for each site c
        begin
        if access(q,p) at c = 1 then
        begin
            for each I $\in C_i$ at site c
            begin
            if $I \in p$ then $[AI]_q = [AI]_q \cup I$
            end //of for each I //
            end //of if access //
        end // of for each c and for each p//
end {GetAISet}

Figure 3: The Get Application/Instance Sets Algorithm

Person = {Person,{a.ssno,a.name,a.age,a.address},
{m.ssno-of,m.whatname,m.daysold,m.newaddr},
{ I_1 {Person1,John James,30,Winnipeg}
I_2 {Person2,Ted Man,16,Winnipeg}
I_3 {Person3,Mary Ross,21,Vancouver}
I_4 {Person4,Peter Eye,23,Toronto}
I_5 {Person5,Mary Smith,40,Toronto}
I_6 {Person6,John West,32,Vancouver}
I_7 {Person7,Jacky Brown,35,Winnipeg}
I_8 {Person8,Sean Dam,27,Toronto}
I_9 {Person9,Bill Jeans,43,Vancouver}
I_{10} {Person10,Mandu Nom,30,Winnipeg} } }
Prof = Person pointer $\odot$ {Prof,{a.empno,a.status,a.dept,a.salary,a.stude nt},
{m.empno,m.status-of,m.coursetaught,m.whatsalary},
{I_1 (person pointer5) $\odot$ {Prof1,asst prof,Computer Sc.,45000,studen ts pointers}
I_2 (person pointer6) $\odot$ {Prof2,assoc prof,Math,60000,students point ers}
I_3 (person pointer9) $\odot$ {Prof3,full prof,Math, 80000,students pointers}
I_4 (person pointer10) $\odot$ {Prof4,full prof,Math,82000,students pointer s} } }
Student = Person pointer $\odot$ {Student,{a.stuno,a.dept,a.feespd} ,
{m.stuno-of,m.dept-of,m.owing},
{ I_1 (person pointer1) $\odot$ {Student1,Math,Y}
I_2 (person pointer4) $\odot$ {Student2,Computer Sc.,N}
I_3 (person pointer2) $\odot$ {Student3,Stats,Y}
I_4 (person pointer3) $\odot$ {Student4,Computer Sc.,N}
I_5 (person pointer7) $\odot$ {Student5,Math,Y}
I_6 (person pointer8) $\odot$ {Student6,Stats,N} } }
Grad = Student pointer $\odot$ {Grad,{a.gradstuno,a.supervisor},
{m.gradstuno-of,m.whatprog}
{ I_1 (Student pointer1) $\odot$ {Grad1,John West}
I_2 (Student Pointer2) $\odot$ {Grad2,Mary Smith}
I_3 (Student Pointer5) $\odot$ {Grad3,Mary Smith} } }
UnderG = Student
I_1 (Student pointer3); I_2 (Student pointer4)
I_3 (Student pointer6)
Dept = {Dept,{a.code,a.name,a.profs,a.students},{m.whichdept,m.number-of-pro fs}},
{ I_1 {Dept1,{Computer Science,prof pointers,student pointers}
I_2 {Dept2,{Math,prof pointers,student pointers}
I_3 {Dept3,{Actuary Science,prof pointers,student pointers}
I_4 {Dept4,{Stats,prof pointers,student pointers} } }

Figure 4: The Complex Sample Object Database Schema

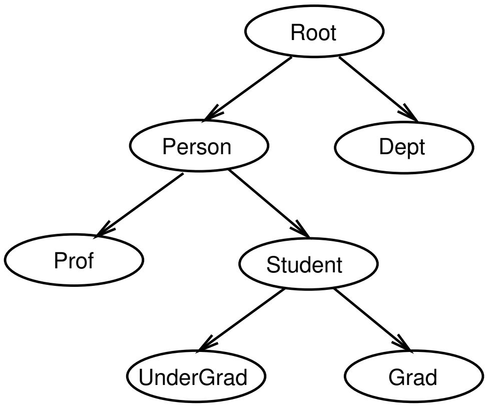

Figure 5: Complex Class Hierarchy

q_2**:** This application groups profs by their addresses. The methods used are defined on class **Prof** with the following predicates:

$$\{P_1\text{: address} = \text{“Winnipeg”}, P_2\text{: address} = \text{“Vancouver”}, P_2\text{: address} = \text{“Toronto”}\}$$

q_3**:** This application separates profs with salaries greater than or equal to \$60,000 from those with salaries less than \$60,000. The methods are in the class **Prof** with the following predicates:

$$\{P_4\text{:salary} \geq 60000, P_5\text{:salary} < 60000\}$$

q_4**:** Groups students by their departments. The methods are from class **Student** and the predicates used are:

$$\{P_1\text{:dept=“Math”}, P_2\text{:dept=“Computer Sc”}, P_3\text{:dept=“Stats”}\}$$

q_5**:** This application groups departments by their general area determined by their names. The methods are for the class **Dept** and the predicates used are:

$$\{P_1\text{:dept=“Math”}, P_2\text{:dept=“Computer Sc”}, P_3\text{:dept=“Stats”}\}$$

Using the given query access frequency and other input data, the object horizontal fragmentation scheme (OOHF) produces the following fragments for the classes in this database.

Class Person

$$F_1^h = \{I_1, I_7\},\ F_2^h = \{I_3, I_4\},\ F_3^h = \{I_2, I_8\},\ F_4^h = \{I_{10}\},\ F_5^h = \{I_6, I_9\},\ F_6^h = \{I_5\},$$

Class Student

Class	Site 1	Site 2	Site3
Person	F1={I1, I7} F5=(I6, I9} F4={I10}	F6={I5} F2={I3, I4}	F3={I2, I8}
Student	F1={I1, I5}	F2={I2, I4}	F3={I3, I6}
Prof	F1={I2, I3, I4}	F2={I1}	
Dept	F1={I2, I3}	F2={I1}	F3={I4}
Grad	F1={I1}	F2={I2, I3}	

Figure 6: Allocation of Fragments to Distributed Sites

	S1	S2	S3	Running directly on
q1	10	20	0	Grad
q2	20	30	15	Prof
q3	25	10	0	Prof
q4	15	25	10	Student
q5	20	20	20	Dept

Figure 7: Access Frequencies of queries at distributed sites

$$F_1^h = \{I_1, I_5\},\ F_2^h = \{I_2, I_4\},\ F_3^h = \{I_3, I_6\},$$

Class Prof

$$F_1^h = \{I_2, I_3, I_4\},\ F_2^h = \{I_1\},$$

Class Dept

$$F_1^h = \{I_2, I_3\},\ F_2^h = \{I_1\},\ F_3^h = \{I_4\},$$

Class Grad

$$F_1^h = \{I_1\},\ F_2^h = \{I_2, I_3\},$$

The fragments above are allocated to three distributed sites using the simple allocation scheme that places fragments at sites where they are most needed by queries, while Figure 6 shows the placement of above fragments to sites and Figure 7 shows the access frequencies of queries at the three sites.

From the application information gathered at distributed sites, the algorithm GetAISet computes the Application/Predicate matrix and the Application/Instance sets as follows:
Class Person

Class	Application	AIsets		
		Site 1	Site 2	Site 3
Person	q1 q2 q3 q4	{I1} {I6, I9, I10} {I6, I9, I10} {I1, I7}	{I4, I7} {I5} {I5} {I3, I4}	 {I2, I8}
Student	q1 q4	{I1} {I1, I5}	{I2, I5} {I2, I4}	 {I3, I6}
Prof	q2, q3	{I2, I3, I4}	{I1}	
Grad	q1	{I1}	{I2, I3}	
Dept	q5	{I2, I3}	{I1}	{I4}

Figure 8: Application/Instance Sets at all Sites

Set of predicates $P^{person} = P^{person} \cup P^{prof} \cup P^{student} \cup P^{grad}$

$P^{person} = \{P_1^{prof}, P_2^{prof}, P_3^{prof}, P_4^{prof}, P_5^{prof}, P_1^{student}, P_2^{student}, P_3^{student}, P_1^{Grad}, P_2^{Grad}\}$

Set of queries $Q^{person} = Q^{person} \cup Q^{prof} \cup Q^{student} \cup Q^{grad}$

$Q^{person} = \{q_1, q_2, q_3, q_4\}$

The computed application instance set for all classes and for each query accessing the class at the three different sites is given as Figure 8.

The local irrelevant access costs, remote relevant access costs and the PE values obtained after executing the OHPE algorithm are shown in the table below:

```
CASE1 (with fragmentation)
Class        Local irrelevant  Remote relevant   PE value
Person           250               0               250
Student          250               0               250
Prof             0                 0               0
Grad             0                 0               0
Dept             0                 0               0

CASE2 ( no fragmentation, full copy resides on every site)
Class        Local irrelevant  Remote relevant   PE value
Person           11834             0             11834
Student          1816              0             1816
Prof             1538              0             1538
```

Grad	300	0	300
Dept	1000	0	1000

CASE 3 (No fragmentation, No replication)

For full copy data reside in site 1

Class	Local irrelevant	Remote relevant	PE value
Person	2603	140	2743
Student	1366	217	1583
Prof	769	250	1019
Grad	267	33	300
Dept	400	200	600

The comparative PE values for the three cases is given in the table below and this clearly shows how better the fragmented system performs against the other non-fragmented systems for all classes.

Comparative PE values of three approaches

Class	OOHF method	Non-fragment, Replicated data	Non-fragmented, Non-replicated data
Person	250	11834	2743
Student	250	1816	1583
Prof	0	1538	1019
Grad	0	300	300
Dept	0	1000	600

A graph showing the comparative performance of the three approaches is given in Figure 9. Lower PE values mean better performance.

5 Conclusions

Availability of a technique to measure the performance of distributed object database systems will greatly increase user confidence in the systems and provide a basis for determining when a re-design should be performed. A re-design of the distribution technique may be needed when due to changes in application access pattern or object database schema, system performance has dropped below an acceptable threshold. This paper contributes by presenting a set of algorithms that would measure the performance of a horizontally distributed object database system by measuring the amount of local irrelevant data access as well as remote relevant data access made by each fragment of each class, the sum of which gives the single partition evaluator (PE) value.

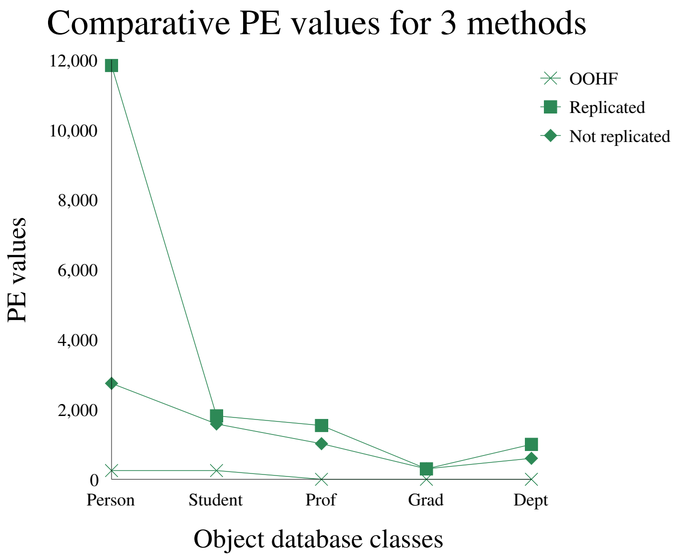

Figure 9: Comparative PE values of three methods

The performance measuring algorithms presented here are an extension of the PE value measure suggested by [CMVN93] for vertically fragmented relations. The work discussed here presents a significant contribution for object horizontally fragmented data and form a basis for future work on much needed dynamic fragmentation of objects. Future work should consider extending this work to accommodate dynamic object fragmentation.

References

[BKS97] L. Bellatreche, K. Karlapalem, and A. Simonet. Horizontal Class Partitioning Design in Object-Oriented databases. In *Lecture Notes in Computer Science volume 1308 DEXA*, pages 58–67, Toulouse-France, September 1997.

[BSS96] L. Bellatreche, A. Simonet, and M. Simonet. Vertical fragmentation in Distributed Object Database Systems with Complex Attributes and Methods. In *In Proc. of 7th Intl. Conf. and Workshop on Database and Expert Systems and applications DEXA 96*, September, 1996.

[CMVN93] S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. B. Navathe. An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis. *Distributed and Parallel Databases*, 2(1):183–207, 1993.

[CNP82] S. Ceri, M. Negri, and G. Pelagatti. Horizontal Data Partitioning in Database Design. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 1982.

[CSW83] S. Ceri, S.Navathe, and G. Wiederhold. Distributed design of logical database schemas. *IEEE Transactions on Software Engeneering*, 9(4), 1983.

[EB95] C.I. Ezeife and Ken Barker. A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System. *International Journal of Distributed and Parallel Databases*, 1:247–273, 1995. Kluwer Academic Publishers.

[EB98] C. I. Ezeife and Ken Barker. Distributed Object Based Design: Vertical Fragmentation of Classes. *Journal of Distributed and Parallel Database Systems*, 6(4):327 – 360, 1998.

[IS95] IBEX Corporation. *distributed object database management system*, 1995. Technique Report Technique Summary Release 2.3.5.

[KPB94] M.L. Kersten, S. Plomp, and C.A Van Den Berg. Object Storage Management in Goblin. In M. Tamer Ozsu, U. Dayal, and P. Valduriez, editors, *Distributed Object Management*. Morgan Kaufmann Publishers, 1994.

[LAD+96] Barbara Liskov, A. Adya, M. Day, M. Castro, and M. Ghemawat. Safe and efficient sharing of persistent objects in THOR. In *proceedings of ACM SIGMOD International Conference on Management of Data*. Association of Computing Machinery, 1996.

[MSW72] W.T. McCormick, P.J. Schweitzer, and T.W. White. Problem Decomposition and Data Reorganization by a Clustering Technique. *Operations Research*, 20(5), 1972.

[NKR95] S.B. Navathe, K. Karlapalem, and M. Ra. A Mixed Fragmentation Methodology for Initial Distributed Database Design. *Journal of Computer and Software Engineering*, 3(4), June 1995.

[NR89] S.B. Navathe and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. In *Proceedings of the ACM SIGMOD*. ACM, 1989.

[OV91] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[RZ95] F. Ravat and G. Zurfluh. Issues in the Fragmentation of object oriented database. In *Proceedings of the Basque International Workshop on Information Technology*, 1995.

[STY98] M. Savonnet, M. Terrasse, and K. Yetongnon. Fragtique: A Methodology for Distributing Object Oriented Databases. In *Proceedings of the 9th International Conference of Computing and Information ICCI'98 Winnipeg*, pages 148–159, 1998.

[SY95] M. Savonnet and K. Yetongnon. A qualitative approach for class fragmentation in distributed object oriented databases. In *Proceedings of the 5th Annual Workshop on Info. Technologies and systems WIT'S 95*, 1995.