INCREMENTAL HORIZONTAL FRAGMENTATION OF DATABASE CLASS OBJECTS

C.I. Ezeife*
School of Computer Science, University of Windsor
Windsor, Ontario, Canada N9B 3P4
cezeife@uwindsor.ca

Pinakpani Dey,
School of Computer Science, University of Windsor
Windsor, Ontario, Canada N9B 3P4

Key words: Object-orientd databases, Incremental horizontal fragmentation, Distribution

Abstract: Horizontal fragments of a class in an object-oriented database system contain subsets of the class extent or instance objects. These fragments are created with a set of system input data consisting of the application queries, their access frequencies, the object database schema with components - class inheritance and class composition hierarchies as well as instance objects of classes.
When these system input to the fragmentation process change enough to affect system performance, a re-fragmentation is usually done from scratch. This paper proposes an incremental re-fragmentation method that uses mostly the updated part of input data and previous fragments to define new fragments more quickly, saving system resources and making the data at distributed sites more available for network and web access.

1 Introduction

Database class objects (classes) can be horizontally fragmented with each horizontal fragment of a class allocated to a site where it is needed most, in a distributed environment. The benefit of such distribution is improved system performance, arising from faster query response time, lower data transmission cost and time, more efficient utilization of storage and lower data maintenance cost.

A class (C, also called class object) is made up of a set of instance objects (I), which have common attributes (A) and common methods (M). Every class has a class identifier (K). Instance objects of a class are also called the class extents and a class can be represented as an ordered relation C = (K, A, M, I). For example, a class *Person* can be represented as (Person, {a.ssnumber, a.name, a.age, a.address}, {m.ssnumber-of, m.whatname, m.daysold, m.newaddress}, $\{I_1, I_2, \ldots, I_{10}\}$). This indicates that class *Person* has attributes social security number, name, age and address as well as methods for manipulating these attributes. This class has ten instance objects $I_1, I_2, \ldots, I_{10}$. A class, *A*, can have subclasses *B*, which are specializations of this class definition. In this case, class A becomes the superclass of class B and class B can inherit all attributes and methods of its superclass A, but not all of its extents. Object-oriented database systems also support class composition hierarchies which allow an attribute of a class to have another database class as its domain or part of it. 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 object-oriented database schema is then defined as a set of its classes that capture features of encapsulation, inheritance, and class composition hierarchies. A method can simply access only attributes of its class.

Each horizontal fragment (C_h) of a class contains all attributes and methods of the class but only some instance objects or subsets of instance objects in the class (Ezeife and Barker, 1995; Ozsu and Valduriez, 1999). Input to the fragmentation process include application queries accessing database objects, their frequencies of access, the database schema specifying the classes in the object based system (class inheritance and composition hierarchies).

Overtime, new class instances are created, database schema may evolve, the pattern of application queries' access and their access frequencies may change, leading to a drop in system performance. Existing distributed database design work in the liter-

*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.

ature including (Ezeife and Barker, 1995; Navathe et al., 1995; Bellatreche et al., 1997; Ozsu and Valduriez, 1999) would normally re-fragment the object database from scratch by re-running the fragmentation schemes utilizing all original input data as well as the updated part of input data. The shortcomings of this approach include (1) overcommitment and utilization of system resources (memory and CPU time and network traffic) and (2) non-availability of fragmentation process to dynamic distribution as the entire system has to shut down to enable re-fragmentation and re-distribution.

This paper proposes an algorithm that incrementally fragments horizontally, database classes using the change in input data with the previous fragments at distributed sites. The changes to the four normal input to the fragmentation design process, which are used for incremental re-fragmentation are: (1) changes in application queries' access patterns, (2) changes in application queries' access frequencies, (3) changes in class inheritance and composition hierarchies, and (4) changes in class object instances. For each of these four kinds of input data, the proposed algorithm provides a solution for when there is an addition, deletion or change in the location of the particular input data. An example change in the input data of type (1) in the list above is having two new application queries q_4 and q_5 start accessing data fragments at site 2. This is treated as an addition of the two applications to queries accessing data at this site. An example change in location of application query is q_3 running at site 1 now runs at site 3. This type of change is treated as a delete of query q_3 from site 1 followed by an addition of this application to site 3.

1.1 Related Work

Recent work in object horizontal fragmentation include schemes OOHF in (Ezeife and Barker, 1995), scheme in (Bellatreche et al., 1997), schemes in (Savonnet et al., 1998). A method for performing dynamic horizontal object fragmentation was proposed in (Ezeife and Zheng, 1999). Distributed object-oriented systems include THOR (Liskov et al., 1996).

The OOHF algorithm in (Ezeife and Barker, 1995) starts by identifying owner and member classes from class inheritance and composition hierarchies. Every subclass of a class, *C* is class *C*'s owner class, while class *C* is a member class of its subclass. Similarly, every class *P* that is part of a class *D* is class *D*'s owner class, while class *D* is a member class of its contained classes. Secondly, the algorithm defines primary horizontal fragments of the class using predicates from queries accessing this class directly. Primary horizontal fragments are defined with complete and minimal minterm fragments (Ozsu and Valduriez, 1999). Thirdly, derived fragments of this class are defined using all primary fragments of all its owner classes from all 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 highest affinity count with, to obtain non-overlapping fragments of each class.

1.2 Contributions

This paper proposes a new algorithm that performs incremental horizontal fragmentation of class objects by using the change in input data with the previous fragments to obtain new sets of best fragments. The proposed incremental horizontal fragmentation scheme utilizes the technique of OOHF algorithm (Ezeife and Barker, 1995) and the principles of object and fragment affinities to undo or re-do the effects on fragments of an addition, deletion or change of an input to the fragmentation process. This approach cuts down on use of computer resources dedicated to re-fragmentation and would make the process of object re-fragmentation more easily applicable in a dynamic distribution environment and on the web.

1.3 Outline of the Paper

Section 2 presents the proposed incremental object horizontal fragmentation scheme. Section 3 demonstrates the working of the scheme with an example. Section 4 presents experimental evaluation of the proposed scheme in comparison with the static approach of OOHF, while section 5 presents conclusions.

2 Algorithm Incremental Object Horizontal Fragment-IOHF

Some definitions needed for the algorithm are first presented before the algorithm, Incremental Object Horizontal Fragmentation - IOHF is presented in this section.

2.1 Definitions

To present the definitions more clearly, assume that a class has the two temporary fragments (from initial application of minterm predicates), F_1:$\{I_1, I_2, I_5\}$, and F_2:$\{I_1, I_4, I_5\}$.

Definition 2.1 *Access frequency acco(I)* of an instance object (I) is the sum of the given access frequencies of all the application queries $\mathcal{Q}_i$ accessing the object through its class or subclasses. Thus, acco(I) = $\sum_i^q acc(q_i, I)$ for all q applications accessing the object. ■

This definition can be used to define the number of relevant and irrelevant accesses made to an object with respect to another fragment. Assume fragments F_1 and F_2 above are at sites 1 and 2 respectively. If a query accessing site 1 needs the instance object I_4, this object is shipped from site 2 to site 1 because it is a relevant access (needed access) with respect to fragment F_1. On the other hand, if a query running at site 1 needs only instance objects I_1 and I_4, it also processes through the unneeded (irrelevant access) objects, I_2 and I_5, which are part of fragment F_1. The affinity measure between any two fragments or between an object and a fragment is increased with increased relevant access count and decreased irrelevant access counts of all objects in the two fragments/units.

Definition 2.2 *Derived access frequency dacco(I)* of an instance object (I) is the sum of the access frequencies of all the application queries $\mathcal{Q}_i$ accessing all of this object's owner objects through all its subclasses. Thus, dacco(I) = $\sum_i^q acc(q_i, I)$ for all q applications accessing object's subclasses. ■

For example, with class City as a superclass of class School, if instance object I_1 of City has attribute name *Toronto*, which is accessed 20 times by a query, and instance object I_2 (York University in Toronto, owner object of City's I_1), is accessed 30 times by applications. While the total acco of City's I_1 is 50, its derived access, dacco is 30 in this case.

Definition 2.3 *Relevant accesses(F_i,F_j)* measures the amount of access made to local objects of fragments F_i and F_j. Thus, Relevant accesses(F_i,F_j) is the sum of access frequencies acco(I) for all $I \in (F_i \cap F_j)$. ■

For example, Relevant access(F_1, F_2)=acco(I_1) + acco(I_5).

Definition 2.4 *Irrelevant access(F_i,F_j)* are those made to instance objects which are in the fragment, F_i, or the fragment, F_j, but not in both. Irrelevant access(F_i,F_j) is the sum of access frequency for all $I \in ((F_i \cup F_j) - (F_i \cap F_j))$. ■

For example, Irrelevant access(F_i,F_j)=acco(I_2) + acco(I_4).

Definition 2.5 *Affinity between two Fragments F_i and F_j of the same class, faff(F_i,F_j)* is measured as the count of how frequently objects of these two fragments are needed together by applications. Thus, faff(F_i,F_j) = Relevant access(F_i,F_j) − Irrelevant access(F_i,F_j). ■

For example, faff(F_i,F_j)=(acco(I_1) + acco(I_5))-(acco(I_2) + acco(I_4))

Definition 2.6 *The Object Affinity* $affo(I_i, I_j)$, measures the affinity between two objects I_i and I_j as the sum of the access frequencies of the two objects for all queries that access both objects. Thus, $affo(I_i, I_j)$ is the sum for all q accessing both I_i and I_j of access frequency of q's access frequencies to I_i and I_j ■

For example, $affo(I_1, I_2)$= acco(I_1) + acco(I_2).

Definition 2.7 *Object-fragment affinity, ofaff(I_i,F)* is the sum of object affinities between this object I_i and all objects of the fragment F. Thus, ofaff(I_i,F) is the sum of affo(I_i,I_j), for all I_j in F not the same object as I_i. ■

For example, ofaff(I_1,F_1) = affo(I_1, I_2) + affo(I_1, I_5)= acco(I_1) + acco(I_2) + acco(I_1) + acco(I_5).

Definition 2.8 *Maximum Object-fragment affinity bond of a class, mofaffbond(I_i,C)* is the highest affinity between any object of a class and any fragment of the class at time of initial fragmentation. ■

For example, mofaffbond(I_1,C)= maximum(ofaff(I_1,F_1), ofaff(I_1,F_2)).

Definition 2.9 *Maximum fragment-fragment affinity bond of a class, mfaffbond(C)* is the highest affinity between any two fragments of a class at time of initial fragmentation. ■

For example, if F_1=$\{I_1,I_2\}$, F_2=$\{I_3,I_4\}$ and F_3=$\{I_5\}$ are fragments of a class C at time of initial fragmentation, mfaffbond(C)=maximum(faff(F_1, F_2), faff(F_1, F_3), faff(F_2, F_3)).

2.2 Algorithm IOHF

The main input to the algorithm are previous horizontal fragments at distributed sites, their minterm predicates, change in application queries' access pattern (represented as change in set of predicates accessing classes), new application queries' access frequencies, change in class hierarchy, change in class composition hierarchy, change in instance objects of classes. The result or output of the scheme is the updated set of horizontal fragments. A summary of parts of the descriptive algorithm is presented as Figures 1, 2 in two continuing parts.

There are basically three ways any of the input data can change, namely: old one can be deleted, new one can be added or existing input can change its position. For example, new queries might start accessing a class, some of the old queries accessing a class may no longer be used, or some queries might stop accessing one class but start accessing another class. Similar changes can be seen for most input data above. Thus, in the algorithm, we define a sequence of actions that will be performed on existing fragments to handle each of the four input types of query access pattern change, query access frequency change, database schema change (class inheritance and class

composition hierarchies), instance object change. For each of these four classes of input data, the algorithm then, presents the sequence of steps that will be performed to handle (1) adding this input to the system, (2) deleting this input from the system and (3) moving this input from one location to another. The step for moving the input from one location to another is generally treated as executing a delete from first location and followed by an execution of addition to the new location. The sequence of actions to take when handling a change in application access pattern is sequence 1 of Figure 1 (part I), while sequence 1.A handles addition of new applications, 1.B handles deletion and handling of relocation of applications constitutes execution of sequence 1B followed by sequence 1A. For handling a change in application access frequency, sequence 2 of Figure 2 (part II) is followed, while sequence 3 of the same figure handles a change in class inheritance and composition hierarchies. Basically, the iohf algorithm given in Figure 1(part I), says that if new applications accessing a particular class are added to the system, apply the minterm predicates from these new applications to existing fragments of this class to define new primary fragments of the class. Then, propagate the effects of these new queries to all superclasses (member classes) of this class by using the new primary fragments to define the new derived fragments of the member classes with each of member class' existing fragments. Using affinity rules 1 and 2, merge the set of new derived horizontal fragments with the set of new primary horizontal fragments of each class, before again using the same affinity rules to merge the new horizontal fragments with the set of existing horizontal fragments of each class. When application queries are deleted, iohf will again obtain the new primary (for the main class) and derived fragments (for its superclasses) from existing fragments. Then, the access frequencies due to the deleted queries are subtracted from the access frequencies of all objects in the new primary and derived fragments. The resultant access frequencies of the objects in the existing fragments are used to re-merge all sets of existing fragments with the fragments they have highest affinity with, by applying affinity rules 1 and 2. As shown in Figure 2(part II), when the application access frequency of an object changes, the change in the access frequency of the object is used to find the affinity measure between the object and all fragments of its class, and the object is re-assigned to the fragment it has highest affinity with provided this affinity due to change in access frequency is higher than the maximum affinity bond between any object and the class at the time of initial fragmentation. Then, the effect of re-assigning an object is carried through the class inheritance hierarchy by obtaining and merging new derived fragments of member classes. To handle change in class inheritance hierarchy, for each new subclass of a class, define new primary horizontal fragments. Then, define new derived fragments of this class now which are merged with its existing fragments. Similarly, for each new superclass of a class, define new derived horizontal fragments and merge the set of each class' new derived fragments with its set of existing fragments. The process for handling change in instance objects of classes is discussed next. When new instance objects of a class are added to the system, use the existing primary minterms to obtain new primary fragments of the class and use the existing derived minterms from all owner (subclasses) classes to obtain new derived fragments of the class. Then, select the existing primary fragment that matches each of the new primary or derived fragments and merge the new fragment with it. If no existing fragment matches the defining minterm of the new fragment, then, the new fragment becomes a new primary fragment. When objects are deleted from the system, iohf will remove the objects from the existing fragments of the class to generate new primary fragments. Then, derived fragments of all member classes are defined using existing fragments. After subtracting derived access frequencies of each deleted object from the access frequencies of the member objects in existing fragments, the affinity measures of existing fragments are re-calculated to merge each fragment with the one it has highest affinity measure with.

3 Example Application of Algorithm IOHF

For example, assume an original object database schema has a superclass, called *City* and class *City* has two subclasses *School* and *Hospital*. The sample database of the three classes is given in Table 1. The application queries for initial fragmentation of the data are given below as q_1 to q_3.

Query q_1: This query groups schools according to their type. Thus, simple predicates from this query are P_1: type = "University" and P_2: type = "Grade".

Query q_2: This query groups hospitals according to the number of beds. Simple predicates from this query are P_1:hobeds$>$150 and P_2:hobeds $\leq$ 150.

Query q_3:This query groups cities according to their population. Simple predicates from this query are P_1: cipop $>$ 60000 and P_2:cipop $\leq$ 60000.

The access frequencies of these class instances for all query applications accessing each object I_j of a class, acco(I_j) are derived as:

For objects of class City: acco(I_1)=10, acco(I_2)=5, acco(I_3)=5, acco(I_4)=10, acco(I_5)=5

For objects of class School: acco(I_1)=5, acco(I_2)=5, acco(I_3)=5, acco(I_4)=10, acco(I_5)=5, acco(I_6)=10

Algorithm 2.1 *(Algorithm (IOHF - Algorithm for Performing Incremental Horizontal Fragmentation of Classes - Part I))*

Algorithm IOHF
Input: Previous Horizontal Fragments
and their Minterm Predicates,
(input changes: change in Application
Queries Access Pattern ($\triangle AF$),
change in Application Queries Access
Frequencies ($\triangle AQ$),
change in class inheritance hierarchy
($\triangle CH$),
change in class composition hierarchy
($\triangle CCH$),
change in instance objects of classes
($\triangle IS$ fragments),
existing fragmentation input data
(AQ, AF, CH, CCH, IS)
Output: Updated Set of Horizontal
Fragments of Classes (F_N^H)
Begin
1. // To handle change in Application Access
Pattern, we do the following: //
A. For all New Application Queries Arriving
for a class at a site do
begin
1.a.1 Obtain new primary horizontal
fragments (F^T) of each fragment
of the class.
1.a.2 Obtain new derived horizontal
fragment (F_d^T) of all member classes
of this class using all of their instance
objects of each fragment.
1.a.3 Merge the new derived fragments, F_d^T
with the new primary fragments, F^H using
affinity rule 1, which merges the new derived
fragment, F_d^T, with the new horizontal,
F_k^H it has the highest affinity aff(F_i^T, F_k^H) with.
1.a.4 Merge the new horizontal fragments from
with existing old horizontal fragments
they have highest affinity with.
end // end of 1.A sequence //
B. For all New Application Queries Deleted for a class
from a site do
begin
1.b.1 Do the same as 1.a.1
1.b.2 Do the same as 1.a.2
1.b.3 By intersecting each new fragment, F_k^T (both
primary and derived) with old horizontal fragment
F_k^H obtain the fragment groups $F_k^{C_i} = F_k^T \cap F_k^H$.
1.b.4 For each object, after subtracting acco(I) or
dacco(I) of deleted queries, from its acco(I)
to remove the effect of deleted query, merge
each $F_i^{C_i}$ with the $F_m^{C_i}$ it has
highest affinity with, where $i \neq m$.
end // end of 1.B sequence //
end // of iohf sequences 1 only//

Figure 1: The Algorithm Incremental Horizontal Fragmentation (Part I)

Algorithm 2.2 *(Algorithm (IOHF - Performing Incremental Horizontal Fragmentation - Part II))*

Algorithm IOHF - Part II
Begin
2. // To handle change in Application Query Access
frequencies, we do the following: //
begin
2.1 Compute the affinity measure between every
instance object I, in a class and all of the class's
fragments, faff(I, F_i^H) using the change in
application access frequencies. Then, re-assign
I to the fragment, F_k^H with highest affinity
if this highest affinity is larger than maximum
affinity bond moaffbond(I, C) of the class.
Maximum affinity bond of a class is the highest
affinity between any object of a class and a
fragment at time of initial fragmentation.
2.2 Obtain derived horizontal fragments (F^d) of
of all member classes of this class using new
fragments from 2.1 above.
Merge derived fragments, F^d with each existing
fragments F^H they have highest affinity with.
end // end of sequence 2 //
3. // To handle change in class inheritance and composition
hierarchies, we do the following: //
A. For a new class added to a link in the hierarchy, do
begin
3.a.1 Make all its new superclasses/containing
classes member classes of this class and all its
new subclasses/contained classes, owner
classes of this class.
3.a.2 For each new owner class of the class, obtain
the derived horizontal fragment of the class, F_d^h.
3.a.3 Merge all of this class's old fragments F^h
with its new derived fragments by merging each
F_d^h with the fragment it has highest affinity
aff(F_i^T, F_k^H) with.
3.a.4 For each new member class of this class,
obtain the derived horizontal fragment, F_d^h.
3.a.5 Merge each derived fragment of the class, F_d^h,
with the old horizontal, F_k^H it has the highest
affinity aff(F_d^h, F_k^H) with.
end // end of 3.A sequence //
B. For a new class deleted from a link in the hierarchy,
begin
3.b.1 For each old member class of this class,
obtain the derived horizontal fragment based on
the class, F_d^h.
3.b.2 Merge each derived fragment of its owner
class, F_d^h, with its old horizontal, F_k^H
it has the lowest affinity aff(F_d^h, F_k^H) with.
end // end of 3.B sequence //
end // of iohf sequences 2 and 3 only//

Figure 2: The Algorithm Incremental Horizontal Fragmentation (Part II)

Table 1: Sample Instances of Object Database Classes

City	Objects with attributes {cityid, ciname, cipop, cistate}
City	I_1\{C01, Toronto, 80000, ON\} I_2\{C02, Windsor, 40000, ON\} I_3\{C03, Montreal, 50000, QB\} I_4\{C04, New York, 90000, NY\} I_5\{C05, Queens, 60000, NY\}
School	Objects with attributes {cityid, scid, scname, stupop, sctype}
School	I_1\{C01, S01, U of Toronto, 4500, Univ\} I_2\{C01, S02, York Univ, 4000, Univ\} I_3\{C02, S03, U of Windsor, 3000, Univ\} I_4\{C02, S04, ABC School, 500, Grade\} I_5\{C04, S05, NY University, 3500, Univ\} I_6\{C05, S06, M School, 400, Grade\}
Hospital	Objects with attributes {cityid, hoid, honame, hobeds}
Hospital	I_1\{C01, H01, Toronto Hospital, 200\} I_2\{C02, H02, Windsor Hospital, 300\} I_3\{C02, H03, WS Hospital, 150\} I_4\{C03, H04, Montreal Hospital, 100\} I_5\{C04, H05, NY Hospital, 300\} I_6\{C01, H06, Queens Hospital, 150\}

For objects of class Hospital: acco(I_1)=20, acco(I_2)=20, acco(I_3)=10, acco(I_4)=10, acco(I_5)=20, acco(I_6)=10

The initial horizontal fragments of these classes are obtained by applying the oohf algorithm of (Ezeife and Barker, 1995), which generates class horizontal fragments as a set of their complete and minimal minterm predicates (valid and non-redundant conjunction of simple predicates in their positive and negated forms). The minterm predicates generated for the three classes from relevant queries are:

Class City: Minterms: MM_1:population $>$ 60000, MM_2:population $\leq$ 60000. Class School: MM_1: type = University $\wedge$ type $\neq$ Grade, MM_2:type $\neq$ University $\wedge$ type = Grade.

Class Hospital:MM_1: Beds $>$ 150, $MM_2 \leq$ 150.

Applying these minterm predicates to object database will create the initial horizontal fragments of classes. Affinity rule 1 is used to merge derived fragments of each class with a primary fragment it has highest affinity with, while Affinity rule 2 is used to maintain non-overlapping fragments by keeping each replicated object in the one final fragment it has highest affinity with. The final initial fragments generated for these classes are:

City: f_1^o $\{I_1, I_2, I_4\}$, f_2^o $\{I_3, I_5\}$

School: f_1^o $\{I_1, I_2, I_3, I_5\}$, f_2^o $\{I_4, I_6\}$

Hospital: f_1^o $\{I_1, I_2, I_5\}$, f_2^o $\{I_3, I_4, I_6\}$

The process described above is the method used by oohf for generating horizontal fragments from scratch.

The method being proposed is incremental approach using an algorithm (iohf). The incremental approach does not involve all data (old and new) in the process, but uses only the new data (the change in the input data) with the initial fragments of classes to create the new best fragments. Continuing with the example, a change in application queries accessing the database is given by an addition of two more queries q_4 and q_5 accessing classes School and City respectively. The query q_4 groups schools with student (P_1) population $>$ 3000 in one fragment and the rest (P_2) population $\leq$ 3000, in another fragment. The query, q_5 groups city where P_1:state=ON, P_2:state=QB, P_3:state=NY. The sequence in the new algorithm IOHF to be applied here is the sequence 1.A of the algorithm given as Figure 1, which requires that we first obtain new primary fragments of the affected classes, *School* and *City* by applying the minterm predicates from the new queries, to each existing fragment of the class. For class School, new minterms are: MM_1 : $stupop$ $>$ 3000, MM_2 : $stupop \leq$ 3000. Applying these predicates to original fragments of School, produces the four primary fragments: f_1^{np} : $\{I_1, I_2, I_5\}$, f_2^{np} : $\{I_3\}$, f_3^{np} : $\{\}$, f_4^{np} : $\{I_4, I_6\}$. Step 2 of the algorithm requires generation of the new derived fragments of all member classes (superclasses, e.g., City) of this class (School), that are based on its newly formed primary fragments (arising from new applications). Thus, each new primary fragment of *School* creates a new derived fragment of member class *City*, to give $f_1^{nd}\{I_1, I_4\}$, f_2^{nd} : $\{I_2\}$, f_3^{nd} : $\{I_2, I_5\}$. Step 3: The six new primary horizontal fragments of class City are: f_1^{np} : $\{I_1, I_2\}$, f_2^{np} : $\{\}$, f_3^{np} : $\{I_4\}$, f_4^{np} : $\{\}$, f_5 : $\{I_3\}$, f_6^{np} : $\{I_5\}$. Each of class City's three new derived fragments from primary fragments of its subclass, School, will merge with any of these six primary fragments of City it has highest affinity count with. For example, faff(f_1^{nd}, f_2^{np}) = 0(for I_4) - 10(for I_1) - 10(for I_6) = -20. Thus, f_1^{nd} is merged with f_3^{np} of City to obtain a final horizontal fragment of City. All such merging will lead to the following fragments of City: f_1^h : $\{I_1, I_4\}$, f_2^h : $\{I_1, I_2\}$, f_3^h : $\{I_2, I_5\}$, f_4^h : $\{I_3\}$. Affinity rule 2 is applied to keep each overlapping object only with the fragment it has maximum object-fragment affinity ofaff(I,F) with. Both I_1 and I_2 are removed from fragment 2 above and the final horizontal fragments of class City are: f_1^h : $\{I_1, I_4\}$, f_2^h : $\{I_2, I_5\}$, f_3^h : $\{I_3\}$.

Table 2: CPU Times (msec) of IOHF and OOHF Algorithms

Changes in input	OOHF time	IOHF time	IOHF Gain
Add Queries(AQ)	9509.7	2602.7	6907.0
Delete Queries(DQ)	6005.2	4223.3	1781.8
Access Freq(AF)	9224.9	1738.7	7486.2
Add Classes(AC)	10130.6	8947.8	1182.8
Delete Classes(DC)	6630.0	1962.7	4467.4
Add Instance (AI)	9524.7	7880.5	1644.3
Delete Instance(DI)	9283.3	7725.7	1557.6

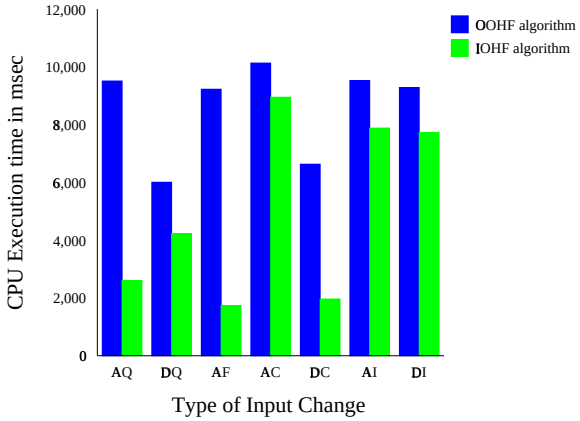

Figure 3: CPU Executions Times of OOHF and IOHF algorithms

4 Experimental Evaluation

Both the correctness and performance of the proposed iohf algorithm were verified experimentally using implementations of the iohf and oohf (Ezeife and Barker, 1995) algorithms in Visual C++, running on an IBM PC with processor speed of 733MHz and 128MB RAM with Windows 2000 operating system. Eight classes in the database are Country, State, City, Road, School, Hospital, Highway and Urbanroad. For all changes in input data, the results obtained by running the oohf from scratch are the same as the fragmentation results obtained by running incremental iohf algorithm on all classes, verifying correctness of the proposed scheme. Experiments show that in all cases, the iohf takes less CPU time than the oohf to refragment the classes. The number of instance objects in the classes for the result of the experiment shown in Table 2 ranges from only 2 countries to 288 zones. The graphical presentation of the same experimental data is shown as Figure 3.

5 Conclusions

The problem of fragmenting and distributing data objects in an object-oriented database system is complex owing to features of encapsulation, inheritance and class composition relationships that need to be captured. This paper handles a further important extension of incremental horizontal fragmentation of classes by considering the sequence of operations to perform on existing fragments when an input data is added, deleted or changed. The sequence of operations would generally use the new input data change to define new temporary fragments of the class which are then merged with existing fragments of the class based on affinity rules and whether the change is an addition, deletion or a move.

An example is used to demonstrate the working of the scheme and experiments show that substantial CPU time resources are saved by the use of incremental approach over the use of re-fragmentation from scratch. Future work should consider extending this technique to vertical and hybrid fragmentations.

REFERENCES

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

Ezeife, C. and Barker, K. (1995). A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System. *International Journal of Distributed and Parallel Databases*, 1:247–273.

Ezeife, C. and Zheng, J. (1999). Measuring the Performance of Database Object Horizontal Fragmentation Schemes. In *Proceedings of the 3rd IEEE international database engineering and Applications Symposium (IDEAS99)*. IEEE press.

Liskov, B., Adya, A., Day, M., Castro, M., Ghemawat, S., Gruber, R., Shrira, L., Myers, A., and Masheshwari, U. (1996). Safe and efficient sharing of persistent objects in THOR. In *proceedings of ACM SIGMOD International Conference on Management of Data*. Association of Computing Machinery.

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

Ozsu, M. and Valduriez, P. (1999). *Principles of Distributed Database Systems*. Prentice Hall, second edition.

Savonnet, M., Terrasse, M., and Yetongnon, K. (1998). 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.