STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLIX**, Number 1, 2004

OPTIMAL CLASS FRAGMENTATION ORDERING IN OBJECT ORIENTED DATABASES

ADRIAN SERGIU DARABANT, ALINA CAMPAN, AND ANDREEA NAVROSCHI-SZASZ

Abstract. Distributed Object Oriented Databases require class fragmentation, performed either horizontally or vertically. Complex class relationships like aggregation and/or association are often represented as two-way references or object-links between classes. In order to obtain a good quality horizontal fragmentation, an optimal class processing order is needed. We present in this paper a new technique for establishing an order for class fragmentation. We improve fragmentation quality by capturing the semantic of input queries in the context of the aggregation hierarchy.

1. Introduction

Fragmentation is an important task that should be carried out in a Distributed Object Oriented Database (DOODB). The purpose of distributing a database is to increase query processing parallelism and to achieve high performance. Similar to the relational model, fragmentation in DOODB is performed horizontally and vertically. Horizontal fragmentation groups into fragments objects that are highly used together. 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. Vertical fragmentation breaks the logical structure of the class: *attributes* and *methods*, and distributes them across the fragments. The objective here is to group class attributes and methods that are frequently accessed together by queries. Each fragment contains, in this case, the same objects, but with different subsets of the attributes and methods [3].

Compared to the flat relational model, the object oriented paradigm introduces new issues into the fragmentation problem, due to its inherent complex nature. Complex object relationships like aggregation and association are part of this paradigm. They are often represented as two-way pointers or references, for performance reasons. This symmetric representation induces many cycles in

Received by the editors: 3/05/2004.
2000 *Mathematics Subject Classification.* C2.4, H2.4.
1998 *CR Categories and Descriptors.* H.2.4 [**DATABASE MANAGEMENT**]: Systems – *Distributed databases, Object-oriented databases* ; C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems – *Distributed databases* .

the association and aggregation graphs. We claim that the order of class fragmentation should take in account the links between classes. As long as there are cycles in the class graphs it is difficult to establish a fragmentation order between classes. In this paper we propose a method that properly eliminates cycles, by taking into consideration the semantic of class relationships and/or the strength of these relationships. We use query statistics in order to quantify the *strength* of links between entities. We propose an algorithmic approach that determines the proper order of fragmentation in an object database. Similar work has been conducted in [6], but only the inheritance relations are taken in account. The full semantic power of aggregation and associations is not captured however. Other research directions in object-oriented fragmentation do not address this problem at all [2, 5, 7, 8].

In section 2 we present the data model and basic concepts needed for problem definition. In section 3 we give an algorithm for establishing the optimal class fragmentation order in a context of an input set of queries. In section 4 we apply our algorithm to an example database and we conclude in section 5.

2. Data Model and Basic Concepts

We use an object-oriented model with the basic features described in the literature [1, 3]. 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. Class attributes/methods are classified as simple and complex. Simple attributes have primitive data types as their domain. Simple methods access only attributes of their class. Complex attributes have other classes as their domain. Complex methods access attributes and/or methods of other classes. Further, they can have as return value objects of a different class type.

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 [3].

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. We denote the fact that C_1 is a superclass of C_2 by $C_1 \prec C_2$. Association between an object and a class is materialized by the instantiation operation. An object O *is an instance of a class* C if C is the most specialized class associated with O in the inheritance hierarchy. An object O *is member of a class* C if O is instance of C or of one of subclasses of C.

An 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.

Aggregation and *association* are implemented as OID pointers. They are represented as a directed cyclic graph.

Definition 1. An ***entry point*** into a database is a meta-class instance bound to a known variable in the system.

An entry point allows navigation to all classes and class instances of its sub-tree (including itself). There are usually more entry points in an object database.

Definition 2. Given a complex hierarchy H, a ***path expression*** P is defined as $C_1.A_1...A_n$, $n \geq 1$ where: C_1 is an entry point in H, A_1 is an attribute of class C_1, A_i is an attribute of class C_i in H such that C_i is the domain of attribute A_{i-1} of class C_{i-1} $(1 \geq i \geq n)$.

Definition 3. 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 $q \in \{<, >, \leq, \geq, =, \neq, \}$.

Definition 4. We model the ***inheritance hierarchy*** as a directed acyclic graph Inh = (Class, Γ), where Class is the set of all classes in the database, $\Gamma = \{(C_1, C_2)|C_1, C_2 \in \text{Class }, C_1 \neq C_2, C_2 \text{ is superclass of } C_1\}$.

We give a working example of an inheritance hierarchy for a reduced university database in Fig. 1.

Let $Q = q_1, \ldots, q_t$ be the set of all queries in respect to which we want to perform the fragmentation.

Definition 5. We model the ***aggregation hierarchy*** as directed cyclic graph Agg = (Class, Λ), where Class is the set of all classes in the database, $\Lambda = \{(C_1, C_2)|C_1, C_2 \in \text{Class}, C_1 \neq C_2, C_2 \text{ aggregates or is associated to } C_1\}$. Each edge u receives a *weight*, denoted as *weigth(u)*, equal to the number of path expression from queries in Q traversing that edge. We say that a link is *stronger* as its weight is larger.

The aggregation hierarchy for our example is depicted in Figure 2.

3. Class Fragmentation Ordering

In the following paragraphs we give a representation of all relations between classes (inheritance, aggregation and association), we explain the reasons behind

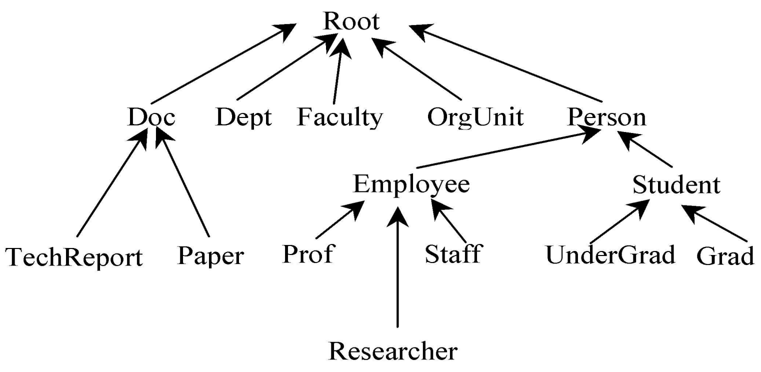

FIGURE 1. The database inheritance hierarchy

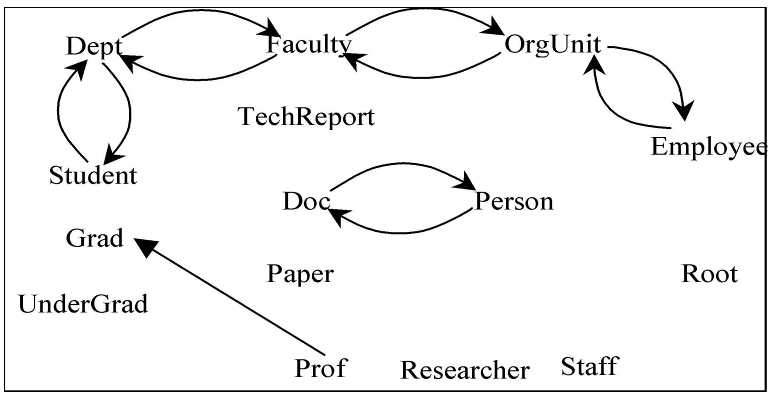

FIGURE 2. The database aggregation hierarchy

our modeling scheme. We propose an algorithm that takes as input an object database and returns the optimal order in which classes should be fragmented so that semantics of the queries is fully reflected in the resulting fragments. We prove that the proper order can always be found - i.e. the problem has always at least one solution.

3.1. **Relationship modeling.**

Definition 6. Let $Pred = \{p_1, \dots, p_q\}$ be the set of all simple predicates Q is defined on. Let $Pred(C) = \{p \in Pred | p$ imposes a condition to an attribute of class C or to an attribute of class C', where $C' \prec C\}$.

Given two classes C' and C, where $C' \prec C$, $Pred(C) \supseteq Pred(C')$. 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 [4].

Definition 7. The directed graph modeling all relations between classes is denoted as *CandidateRelGraph*=(NAClass,U) where NAClass is the set of non-abstract classes in Class. $(C_1, C_2) \in U$ if:

- Class C_1 is aggregated by class C_2
- $\exists C \in$Class, $C \prec C_2$ and C aggregates C_1; this means that aggregation is inherited by a class from its ancestor.
- $\exists C \in$Class, $C \prec C_1$ and C is aggregated by C_2; this means that the fact of being aggregated is inherited by a class from its ancestor.
- $\exists C_1', C_2' \in$Class, $C_1' \prec C_1, C_2' \prec C_2$ and C_1' is aggregated by C_2'; i.e. the fact of being aggregated by / aggregating a class is inherited.

Definition 8. Let *RelGraph*=(FClass,V) be the subgraph of *CandidateRelGraph*, where FClass=NAClass-$\{C|C \in$ NAClass, $Pred(C) = \emptyset$ and $\nexists$ a path in *CandidateRelGraph* between C and a class $C' \in$NAClass so that $Pred(C') \neq \emptyset\}$.

We don't keep abstract classes because clustering an empty set of objects has no meaning. By propagating aggregation relations through the inheritance hierarchy we eliminate the inheritance dimension from *RelGraph* and we keep in the same time the semantic of aggregation for each particular class. This will turn out to be a necessary and helpful decision for our method, as we will see. As a consequence, every pair (inheritance path, aggregation link) is transformed into a link with the weight of the aggregation link.

We start from the presumption that a fragmentation algorithm first performs fragmentation on the classes with query conditions on them. Then, derived fragmentation is performed on the aggregator or aggregated classes that are linked with the already fragmented classes. Definition 8 eliminates classes that do not meet any of these requirements because they cannot be fragmented.

Definition 9. We denote by $CAN(C)$ the node aggregation coefficient of class C, defined as follows:

$$CAN(C) = \sum_{u \in V, u=(C,\_)} weigth(u) + CAN(C'), C' \prec C. \tag{1}$$

An aggregation edge $C_1 \rightarrow C_2$ will be traversed by path expressions following the pattern $a.b.c \ldots C2.C1. \ldots d.e.f$. This means that the path expression navigates from instances of C_2 to one or more instances of C_1. It makes sense to first fragment class C_1 and then class C_2; when fragmenting C_2 we should take in account the fragmentation of C_1. We want to place in the same fragment of C_2 objects aggregating instances from a fragment of C_1. Objects of a fragment of C_2 should aggregate as much as possible objects from the same fragment of C_1.

When the aggregation graph has cycles they should be broken in order to be able to perform fragmentation. One edge needs to be ignored from each cycle. Our decision is to select the least traversed edge by path expressions. The CAN measure precisely quantifies the navigation frequency for edges adjacent to a node.

When class A aggregates a class C_1 it aggregates all subclasses of C_1. A refers instances of C_1 and of all its subclasses. This is why we transfer the CAN of C_1 to its subclasses.

The CAN coefficient of a node quantifies how strong the other classes aggregate the node. Intuitively, classes with large CAN values strongly influence the overall fragmentation of the entire database. We want to fragment classes in descending order of CANs, as much as possible. We conserve this way the strongest aggregation relationships and we perform derived fragmentation with respect to these relationships, together with primary fragmentation, in a single step.

3.2. **Algorithm FragOrder.**

```
Algorithm FragOrder is
Input:  RelGraph=(FClass,V); $CAN(C)$,where $C$ ∈FClass;
Output:  L=[$C_1, C_2, \ldots, C_n$], n=|FClass|; L gives the fragmentation order.
Var:
   LNC - CAN ordered (descending) list of classes having conditions;
   LNFC - the set of classes having no conditions imposed on them.
 Begin
   LNC:=[C|C∈FClass, $Pred(C) \neq \emptyset$, in CAN descending order];
   LNFC:=FClass-LNC; L:=∅;
   While FClass≠ ∅ do
      i:=1; continue:=true;
      While i≤|LNC| and continue=true do
         C:=LNC[i];
         If {u|u∈V,u=(A,C),A∈LNC}=∅ //C doesn't aggregate any class
            Call Remove(C,RelGraph,LNC,LNFC);
            continue:=false;
         Else //C aggregates classes from LNC
            Call BreakCycles(C,RelGraph);
            If {u|u∈V,u=(CA,C),CA∈LNC}=∅
               Call Remove(C,RelGraph,LNC,LNFC)
               continue:=false;
            End if;
         End if;
      End while;
   End while;
End.

Subalgorithm Remove(C,RelGraph,LNC,LNFC) is
 Begin
   LNC:=LNC-[C]; L:=L+[C];
   New:={U|U∈LNFC,(C,U)∈V} ∪
        {U|U∈LNFC,(U,C)∈V, ∄ path in RelGraph from CC∈LNC to U};
   LNC:=LNC+[C|C∈New]; Resort(LNC);
   LNFC:=LNFC-New;
   FClass:=FClass-C;
End;

Subalgorithm BreakCycles(C,RelGraph) is
Begin
  // We first break trivial cycles
  @ While C participates in a trivial cycle P
     V:=V-{u|u∈P,weigth(u)=min{weight(v),v∈P}}
  @ End While
  // Breaking non-trivial cycles
  @ While C participates in a cycle P
     V:=V-{u|u∈P,weigth(u)=min{weight(v),v∈P}}
  @ End While
End;
```

We use LNC - the list of classes with conditions either directly defined on them or induced by fragments of their aggregated classes. These are the classes that we can fragment at a given moment. We try to take out from LNC the most aggregated class (this class should be the first to fragment) - and add this class to L. A class C (with maximum CAN) can be taken out from LNC if it has no incident edges (C does not aggregate any of the remaining classes in LNC). If C aggregates classes from LNC then we break the trivial cycles C is involved in (if any), in order to free it. In each cycle we break the weakest aggregation edge. If C is not freed then we pass to the next class in LNC and we reiterate the same process. When we remove a class from LNC, we add that class to L and we add all its successors and all its predecessors (S) that do not aggregate classes from LNC (there cannot be any induced aggregation conditions from LNC on S).

3.3. **Correctness Issues.**

We prove that the algorithm terminates and that the problem always has (at least one) solution. First of all we prove that when no more nodes can be taken out from *RelGraph*, there is a cycle in *RelGraph*. Then we prove that, by breaking cycles, we free in each iteration one node.

Theorem 1. *Given an oriented graph $G = (X, \Gamma)$, $X = \{x_1, \ldots, x_n\}$, if $\forall x_i$, $|\Gamma^-(x_i)| > 0 \Rightarrow G$ contains at least a cycle.*

Proof: Let's presume by *reductio ad absurdum* that $\forall x_i, |\Gamma^-(x_i)| > 0$ and there are no cycles in G.
We take an $x_{i_1} \in X$. $|\Gamma^-(x_{i_1})| > 0$ means that there exists at least one $x_{i_2} \in X$ so that $(x_{i_2}, x_{i_1}) \in \Gamma$. Let Y be the set $\{x_{i_1}\}$. $|\Gamma^-(x_{i_2}) > 0$ means that there exists at least one $x_{i_3} \in X$ so that $(x_{i_3}, x_{i_2}) \in \Gamma$. If $x_{i_3} \in Y$ then there is a cycle. If $x_{i_3} \notin Y$, let $Y = Y \cup \{x_{i_2}\}$. By continuing this process, either we obtain a cycle if $x_{i_k} \in Y$, or we reach the situation where we detected no cycle and $Y = \{x_{i_1}, \ldots, x_{i_n}\}$. But $|\Gamma^-(x_{i_n})| > 0$, which means that there exists at least one $y \in X$ so that $(y, x_{i_n}) \in \Gamma$. Node y must be in Y, as there are no other nodes from which to choose. We have thus constructed a cycle, fact that contradicts our presumption.

Theorem 2. *Given an oriented graph $G = (X, \Gamma)$, $X = \{x_1, \ldots, x_n\}$, if $\forall x_i$, $|\Gamma^-(x_i)| > 0$, by breaking a finite number N of times an edge (x_{k_1}, x_{k_2}), $1 \le k \le N$, we obtain a new graph $G' = (X, \Gamma'), \Gamma' = \Gamma - \{(x_{k_1}, x_{k_2}), 1 \le k \le N\}$ so that exists $x_j \in X$ with $|\Gamma'^-(x_j)| = 0$.*

Proof: If $\forall x_i \in X, |\Gamma^-(x_i)| > 0$, then, according to Theorem 1, there is at least a cycle in G. Let $(x_{i_1}, x_{i_2}, \ldots, x_{i_m}), m \le n, x_{i_j} \neq x_{i_{j+1}}, 1 \le j < m, x_{i_m} = x_{i_1}$ be a cycle. We choose from it an edge to be eliminated, let it be $(x_{i_j}, x_{i_{j+1}}), 1 \le j < m$. This means that $|\Gamma^-(x_{i_{j+1}})|$ decreases by 1. As $|\Gamma^-(x_i)|$ is finite $\forall x_i \in X$, following the same procedure a finite number of times we reach the situation when for a node $x_j \in X, |\Gamma^-(x_j)| = 0$.

4. An example

Given the database in Figure 1 and Figure 2 having as entry points Doc, Person, Faculty, and a set of queries:
q1: This application retrieves all graduates having their supervisor in ProgrMeth or InfSyst OrgUnits.
q1 = (Grad, Faculty.Dept.Student, Grad.Supervisor.OrgUnit.Name in ("ProgrMeth", "InfSyst"));
q2: This application retrieves all undergraduates with scholarships from Comp. Sci.
q2 = (UnderGrad, Faculty.Dept.Student, UnderGrad.Dept.Name like "CS%" and UnderGrad.Grade between 7 and 10)
q3: This application retrieves all undergraduates older than 24 years from Math and Comp. Sci. depts.
q3 = (UnderGrad, Faculty.Dept.Student, (UnderGrad.Dept.Name like "Math%" or UnderGrad.Dept.Name like "CS%") and UnderGrad.Age()≥24)
q4: This application retrieves all researchers having published at least two papers.

q4 = (Researcher, Doc.Person,Researcher.count(Reasercher.doc)$\geq$2)
q5: This application retrieves all teachers from ProgrMeth and InfSyst OrgUnits with salary greater than 40000.
q5 = (Prof, Faculty.OrgUnit.Employee, Prof.OrgUnit.Name in ("ProgrMeth", "InfSyst") and Prof.salary$\geq$40000)
q6: This application retrieves all profs having published at IEEE or ACM
q6 = (Prof, Doc.Person., Prof.Paper.Publisher in ("IEEE" , "ACM") and Prof.Position="prof")
q7: This application retrieves all tech reports published after 1999.
q7 = (TechReport, Doc, TechReport.year>1999)
q8: This application retrieves all depts with students having grades less than 5.
q8 = (Set(Student.Dept), Person, Student.Grade<5)
q9: This application retrieves all employees with salaries greater than 35000.
q9 = (Employee, Person, Employee.salary>35000)
q10: This application retrieves all grads with at least one paper.
q10 = (Grad, Person, Grad.count(Grad.Paper)$\geq$1)
q11: This application retrieves all students from Comp. Sci. depts
q11 = (Student, Person,Student.Dept.Name like "CS%")
q12: This application retrieves all students from Math depts.
q12 = (Student, Person,Student.Dept.Name like "Math%")
q13: This application retrieves all staff with salaries greater than 12000.
q13 = (Staff, Person, Staff.salary>12000)

Aggregation Edges LineLabel → ColLabel	Dept – CAN=10	Grad – CAN=7	UnderGrad - CAN=7	Prof – CAN=6	Researcher– CAN=5	Staff– CAN=5	OrgUnit– CAN=3	TechReport– CAN=2	Paper– CAN=2	Faculty– CAN=0
Dept		2	5							3
Grad	3							2	2	
UnderGrad	3							2	2	
Prof		1					1	2	2	
Researcher							1	2	2	
Staff							1	2	2	
OrgUnit				2	0	0				1
TechReport		1		1	0					
Paper		1		1	0					
Faculty	0						0			

TABLE 1. RelGraph adjacency matrix.

the adjacency matrix for *RelGraph* is given in Table 1.

In our example all non-abstract classes have conditions imposed on them by the set of queries. The resulting class fragmentation order resulted by applying the algorithm is: Researcher→Staff→OrgUnit→Prof→Grad→Dept→UnderGrad →TechReport→Paper→Faculty.

5. Conclusions and Future Work

We claim that class fragmentation order is significant in distributed object orientated databases. We investigate in this paper a method for choosing this order in respect to a set of application queries given as input. Initial experiments show that the fragmentation order has an important impact in the fragmentation quality. We plan to develop measures for quantifying this quality improvement. We also aim to compare different, richer scenarios with and without fragmentation order involved and to study the limitations, if any, of this technique.

References

[1] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S. - The Object Oriented Database Manifesto, In Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases, 1989.

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

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

[4] Bellatreche,L., Karlapalem, K., Simonet, A. - Horizontal Class Partitioning in Object-Oriented Databases, In Lecture Notes in Computer Science, volume 1308, pp 58-67, Toulouse, France, 1997.

[5] Darabant, A.S, Campan, A. - Horizontal object fragmentation using clustering techniques, In Proc. of Computers and Communications, Oradea, Romania , 2004(to appear).

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

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

[8] Ravat, S. - La fragmentation d'un schema conceptuel oriente objet, In Ingenierie des systemes d'information (ISI), 4(2), pp 161-193, 1996.

Babes Bolyai University, Cluj Napoca,Romania
E-mail address: dadi@cs.ubbcluj.ro

Babes Bolyai University, Cluj Napoca, Romania
E-mail address: alina@cs.ubbcluj.ro

Babes Bolyai University, Cluj Napoca, Romania
E-mail address: deiush@cs.ubbcluj.ro