

On Data Fragmentation and Allocation in Distributed Object Oriented Databases

A. Koreïchi B. Le Cun

Avril 97

Abstract

The objectif of object oriented databases is to respond to the needs of new applications as engineering and multimedia which manipulate complex data. Furthermore, most of these applications need to execute in a distributed environment, making necessary the distribution of data on different sites. This must guarantee a minimum cost of inter-site communication and minimum access to irrelevant data. Data distribution has largely been studied in the relational model, but the complex structure of objects and their relationships make it difficult for object oriented databases.

In this paper, both fragmentation and allocation problems are tackled. Hybrid fragments are defined by applying horizontal fragmentation followed by vertical fragmentation to each database class. Resulting fragments are disjoint regarding to data but overlapping regarding to methods. An optimisation model for a nonreplicated data allocation in the form of a non linear integer zero-one programming problem is also developed; the objective function proposed aims to minimize communication and storage costs.

keywords : object oriented databases, link graph, data fragmentation, data allocation.

1 Introduction

It is largely accepted nowadays that the relational model doesn't offer all the modelisation capabilities needed by advanced applications as geographic information systems and artificial intelligence. These applications require managing complex, possibly multimedia, objects. The relational model due to its simplicity isn't able to reply to these requirements.

The object oriented database management systems offer many possibilities of structuration to model complex data: an object may be hierarchically composed of sub-objects, several objects may share the same sub-object, an object may appear as attribute value of another object; also, the class hierarchy introduces another dimension of object relationships: an object of a class may appear in all its superclasses. It appears that the most important contribution of the object model is its capability of reflecting reality as accurately as possible through the objects and their relationships.

Data distribution has largely been studied in the relational model, but the complex structure of objects and their relationships make it more difficult for object oriented databases. Data distribution is carried out by first fragmenting data and then allocating the obtained fragments to the network sites. Fragmentation allows grouping data items used together by applications so as to minimize the I/O ratio. The allocation of fragments has the primary goal of minimizing the number of remote accesses which are performed by applications.

As proposed in [KNM94], the fragmentation methods used for the relational model may be extended to the object model. However, as classes are closely related to each other, much more importance must be accorded to derived horizontal fragmentation. In a

relational database, if the instances of a relation $R1$ are referenced by those of a relation $R2$, then $R1$ is fragmented based on the predicates of $R2$. Fragmenting $R2$ if it is referenced by more than one relation is however more difficult. In such a case, must $R2$ be fragmented based on the properties of one, some or of all the referencing relations?

This problem is much more complex when dealing with Object oriented databases because relationships are not only more numerous but also of different types. Indeed, as mentioned above, the principle contribution of the object model is its ability to reflect reality as accurately as possible through the objects and their relationships. The facilities offered by the object model in moving from the real-world to the model leads the designer to a fine "modularization", which creates in our sense a great number of classes and relationships and consequently complicates the fragmentation process.

The allocation problem is also more complex in distributed object oriented databases because it must not only deal with data allocation but with method allocation too. As a method manipulates only a subset of the attributes and the objects of a class, it must naturally reside at the site that contains the data it manipulates.

2 Fragmentation: state of the art

Since the beginning of the 80's, many researchers have been interested in distributed database design. Technological and organizational reasons justify this tendency: distributed databases eliminate many limitations of centralized databases, and naturally correspond to the decentralized structure of several organizations. A distributed database can be defined as a collection of data, which logically belong to the same system but are distributed over the sites of a computer network [CP84].

The design of a distributed database is a complex task because it requires the comprehension and the resolution of several related subproblems as data fragmentation (horizontal, vertical, hybrid), data allocation (with or without redundancy), optimisation and allocation of operations (request transformation, selection of the best execution strategy, allocation of operations to sites). The subproblems we consider are data fragmentation and allocation.

The design of a distributed database relies on the definition of the global conceptual schema. The global conceptual schema specifies all the data contained in the distributed database as if it was centralized. It will contain global relations if the chosen model is relational. Each global relation is divided into disjoint portions called fragments. The correspondance between the global relations and fragments is defined in the fragmentation schema. The allocation schema defines the allocation site of each fragment.

2.1 Relational fragmentation methods

There exists three fragmentation types: vertical, horizontal and hybrid. Vertical fragmentation consists of subdividing a relation into subrelations which are projections of the original relation according to a subset of attributes. The horizontal fragmentation divides a relation into subsets of tuples based on selection operations. The hybrid fragmentation consists of dividing a relation horizontally, and then splitting vertically each of the obtained horizontal fragments or vice-versa.

2.1.1 Vertical fragmentation

Vertical fragmentation is used in order to increase transaction performance. The more the obtained fragments are close to transaction requirements, the more the system is efficient. The ideal case occurs when each transaction matches exactly a fragment, i.e it needs only this fragment. If some attributes are always used together, the fragmentation process is

trivial. But in reality, applications are rarely faced with such trivial cases. For relations having tens of attributes, it is necessary to develop systematic approaches for vertical partitioning. If a relation has m attributes, it can be partitioned following $B(m)$ different ways where $B(m)$ is the m^{th} Bell number which is almost of order m^m [HN79]. Since the beginning of the 80's, many works have adressed the database vertical partitioning problem.

1. *Hoffer and Severance (1975)*

Hoffer and Severance [HS75] have developed the attribute affinity concept. This metric measures the frequency of accessing simultaneously a couple of attributes. The attributes having high affinity are grouped together by using the Bond Energy Algorithm developed by Mc Cormick and al. [MSW72].

2. *Hammer and Niamir (1979)*

Hammer and Niamir [HN79] have proposed a heuristic where the input is a set of blocks corresponding each one to an attribute. This is the initial candidate partition. At each step of the search, several modifications of this partition are generated and then submitted to a cost evaluator. If one of the modified partitions have a cost which is less than the one of the current partition, then this modified partition becomes the current candidate partition and the search continues until no modification is possible. Modifying a partition may be obtained in two different ways: by grouping two blocs or by regrouping an attribute i.e by removing it from one bloc and inserting it into another one.

3. *Navathe, Ceri and al. (1984)*

This work [NCWD84] is the extension of those of Hoffer and Severance. The authors use an attribute affinity matrix that they order by using the Bond Energy Algorithm as proposed in [HS75]. However, determining the vertical fragments is done automatically, whereas it was let to the subjectif judgment of the designer in [HS75]. There are two steps in the partioning algorithm. In the first step, the fragmentation is obtained by applying iteratively a binary partitioning algorithm. At this step , no cost factor is considered. At the second step, estimations of cost factors reflecting the physical environment, are included in order to optimise the initial fragments. The algorithm complexity is $O(n^2 \log n)$.

4. *Cornell and Yu (1986)*

Cornell and Yu [CY86] proposed a vertical partitioning algorithm which minimizes the number of disk accesses. The algorithm is based on integer linear programming methods. The partitioning of a relation requires the knowledge of several parameters concerning the relation (length, selectivity and number of attributes) and transaction types and behaviour (their frequency and the attributes they access)

5. *Ceri, Pernici and al. (1989)*

The authors propose two tools for vertical fragmentation: "DIVIDE" and "CONQUER" [CPW89]. The tool "DIVIDE" performs only data fragmentation and allocation. it implements the partitioning algorithm proposed in [NCWD84]. The tool "CONQUER", in addition to data fragmentation and allocation, ensures the optimisation and allocation of operations.

6. *Navathe and Ra (1989)*

Navathe and Ra proposed in 1989 a graphical technique of partitioning [NR89]. The attribute affinity matrix is considered as a complete graph where nodes represent attributes and the edges' weights represent the affinity values. The algorithm, by

successively adding edges, generates all the fragments in one iteration by considering a cycle as a fragment. The algorithm has a complexity of $O(n^2)$ and has the advantage of not using an objective function.

7. *Lin, Orłowska and al. (1993)*

Lin and al. [LOZ93] extend the work of [NR89] on graphical partitioning. The input to the algorithm is the affinity graph. They propose searching a subgraph of at least two nodes for which affinity values are greater than those of each incident edge.

8. *Chakravarthy, Muthuraj and al. (1993)*

The authors [CMVN94] have developed a partition evaluator which evaluates the partition quality by using two costs: the access cost to the irrelevant local attributes (present on the execution site of the transaction but not used by the transaction), and the access cost to the irrelevant remote attributes (not present on the execution site of the transaction but necessary for its execution).

2.1.2 Horizontal fragmentation

Horizontal fragmentation of a relation consists of partitioning the set of its instances (tuples) into disjoint subsets. Each subset, called fragment, has the attributes of the original relation and satisfies a predicate which is a boolean expression of the form $A_i \theta value$, where A_i is an attribute and θ a comparison operator. Each fragment is allocated to a site of the distributed database.

In order to carry out data distribution, the user must have a good knowledge of the logical schema (relations and relationships) and of the potential of use of the database at the different sites [CNW83]. These informations are used for defining relevant fragments and for allocating these fragments to sites efficiently. The user has also to know how the fragmentation of a relation can be propagated to the other relations via links. Thus, two types of horizontal fragments can be defined: primary and derived. A primary horizontal fragment depends on the properties of its own attributes. A derived horizontal fragment depends on the properties of another relation.

The primary horizontal fragmentation In the literature, fragmenting primarily a relation can be done in two different ways.

- *Predicate construction*: in this method [CP84], the set P of simple predicates describing the applications accessing a given relation is first defined. P must be complete and minimal. Completeness means that two tuples belonging to the same fragment have the same probability of being accessed by any transaction. Minimality guarantees that all the predicates are relevant. Given P , the set M of minterm predicates is then constructed. A minterm predicate is the conjunction of all the predicates of P , taken in natural or negation form, provided that the obtained expression is meaningful. A primary horizontal fragment is obtained by associating to each minterm predicate the set of tuples which verify it.
- *Adaptation of vertical fragmentation methods to horizontal fragmentation*: this method is based on the predicate affinity concept [ZO94]. Predicates with high affinity are grouped into horizontal fragments.

In order to fragment horizontally a relation by using the binary partitioning algorithm, the predicate usage matrix PUM is first constructed. Each line represents a transaction and each column a simple predicate. $PUM[i, j] = 1$ if the transaction i uses the predicate j and 0 otherwise. By using the process defined in [NCWD84] for fragmenting vertically relations, a predicate affinity matrix is constructed and

then ordered. The last step consists of constructing the horizontal fragments and thus defining the selection predicates used for the fragmentation. The predicate construction consists of linking the simple predicates having the same attribute by the "OR" operator, and those having distinct attributes by the "AND" operator. Last, a residual fragment is defined by constructing the negation of the disjunction of the different fragmentation predicates [ZO94].

The derived fragmentation Relations in a database schema are not independent. This dependency is specified in the relational model by referential integrity constraints. In [CP84], the relationships are modeled explicitly by links. The relation at the tail of the link is called the owner of the link and the relation at the head is called the member. The link between the owner and the member of a link is defined as an equi-join. As joins are very costly (especially in distributed databases), it is important to take into consideration inter-relation links while fragmenting horizontally relations. This is done by derived horizontal fragmentation.

A derived horizontal fragmentation is defined on a member relation according to the selection predicates of the owner relation. Derived horizontal fragments of a member relation R linked to an owner relation S are defined by:

$$R_i = R \bowtie S_i, \quad 1 \leq i \leq w$$

where w is the maximum number of fragments that may be defined on R and S_i the I^{th} fragment of S .

2.2 Objects and fragmentation

[KNM94] is to our knowledge the first paper which analyses the problem of object distribution and proposes general algorithms. It gives an overview of the different problems which must be solved in the distribution design. As general solution, it proposes the extension of the relational fragmentation methods. Other works present algorithms for fragmenting object oriented databases. We will describe briefly the approach followed in each of the algorithms we know.

1. Ezeife and Barker (1994-1995)

[EB94, EB95] have established a taxonomy based on the attribute types (simple or complex) and method types (simple or complex) and have extracted four class models: class model constituted of simple attributes and simple methods, class model constituted of complex attributes and simple methods, class attributes constituted of simple attributes and complex methods and last class model constituted of complex attributes and complex methods. Thanks to this classification, they took into consideration, in their algorithms, all the characteristics of the object orientation. [EB94, EB95] have proposed algorithms for vertical and horizontal fragmentation for each of the class models mentioned above.

• Horizontal fragmentation

The proposed algorithm uses the principle of minterm predicates. Both of the inheritance and aggregation hierarchies are preserved.

The principle consists of producing in a first step primary horizontal fragments for each of the database classes. The result of this fragmentation is then used to induce derived fragmentation via inheritance, composition and method call links. Finally, the algorithm merges primary and derived fragments of each class: a primary fragment is merged with the derived fragment with which it has the highest affinity. The affinity between a primary and a derived fragment is defined as the frequency at which both are used simultaneously by applications.

- **Vertical fragmentation**

Vertical fragmentation aims to fragment a class such as attributes and methods frequently used together are grouped together. The general approach consists of first grouping the methods frequently used together in each class [EB94]. Grouping techniques similar to those used for attribute grouping in relational are used: method usage matrix, applications' frequencies, method affinity matrix. Each method group is then extended to incorporate the attributes used by the group methods. If an attribute is referenced by methods which belong to different groups, it is assigned to the one with which it has the highest affinity.

2. *Savonnet (1995)*

The aim of this approach is to take into account the semantic information of the class relationships [Sav95]. The idea is that strongly related objects are more subject to common use than the others. The class dependency graph is thus partitioned into a set of partition trees called partition forest. Each class of the global conceptual schema appears in a unique partition tree. A partition tree is constructed by choosing iteratively a root node and by deleting from the dependency graph the connected subgraph. At each step of the construction, the node with the highest weight is selected as a root. The input to the fragmentation process is the partition forest. The root is fragmented primarily, and derived fragmentation is applied to each non root class.

3. *Nachouki and Briand (1995)*

[NB95] have proposed to group methods by using affinity measures between them and by using the Bond Energy Algorithm. From a method group, they deduce which objects, which attributes and which predicates are used, and consequently define hybrid fragments.

4. *Ravat (1996)*

Ravat [Rav96] extend relational fragmentation methods to the object model. Each class of the global conceptual schema is fragmented horizontally and vertically to produce hybrid fragments. All the classes are fragmented primarily except strong component classes (non shared classes) which are fragmented by using derived fragmentation.

3 Proposed work

This section first describes some preliminary concepts. We then give some propositions related to object fragmentation and allocation.

3.1 Preliminary definitions

Two strategies have been identified for designing distributed databases: the Top-Down approach and the Bottom-Up approach. In the Top-Down approach, the design of distributed databases is made up of three steps : conceptual design, distribution design and technical design. We are interested by the distribution design step which aims to divide the global conceptual schema produced at the conceptual design phase into local conceptual schemas, by distributing the entities over the sites of the distributed system.

In the following, we first describes some object oriented basic concepts. We then define the notion of link graph as input of the distribution process. Transaction modeling is last described.

A. The object data model

The data in an object based system consists of a set of encapsulated objects. The concept of object represents an encapsulation of the attributes that describe data and the methods that manipulate them. A unique identifier is associated to each object. Objects with common attributes and methods belong to the same class, and every class has a unique identifier. Inheritance allows reuse and incremental redefinition of classes in terms of existing ones. Parent classes are called superclasses while classes that inherit attributes and methods from them are called subclasses. Composition relationships allow the representation of composite objects which include other objects as part of them. These are called component objects. Due to the encapsulation concept, the access to objects can be performed only by method invocations. The set of objects that belong to a class represents its extension. In the inheritance hierarchy, objects are stored in the most specialized class.

B. The link graph

The global conceptual schema is composed of a set of classes and inter-class links. The existence of a link arises due to some real world relationship which exists between two classes. Two types of links characterize the object model: the inheritance links and the composition links. In order to fragment horizontally database classes, it is important to note how they are connected to each other, especially with joins. Composition links capture such relationships. However, some composition links are not explicitly represented in the global conceptual schema although they semantically exist. Such links are implicitly captured by the inheritance links. Indeed, if a class C_i is a subclass of a class C_j which has a component class C_k , then C_i inherits all the attributes of C_j including the complex attribute (C_k). However, in the global conceptual schema, the composition link ($C_i \rightarrow C_k$) is not represented.

We explicit such informations in a "Link Graph" that we construct from the global conceptual schema and which constitutes the input to our distribution process. The nodes in the link graph represent classes and the edges composition relationships. Given a class C_j and the set of all its subclasses $SUB(C_j)$, if C_j has an attribute of type C_k , then there exists an edge from each class belonging to $C_j \cup SUB(C_j)$ to C_k . Indeed, as mentioned above, subclasses of C_j inherit both simple and complex attributes of C_j . An example of link graph is given in figure 1.

C. Transaction modeling

A user query accessing database objects is defined as a sequence of method invocations on an object or a set of objects of classes. A user query Q_k is represented by $\{M^{i_1j}, M^{i_2k}, \dots, M^{i_nl}\}$, where each M in a user query refers to an invocation of a method of a class object. It is assumed that the user has a good notion of the important transactions that will run against the database, and of the methods involved in each transaction. We suppose that each transaction has known execution frequencies for each of the sites where it may originate, and that data volumes transmitted between transactions' methods (parameters and results) can easily be estimated.

3.2 Class fragmentation

Experience in the relational model showed that a simple horizontal or vertical fragmentation of a database schema is not sufficient to satisfy the requirements of user applications [OV91]. Thus, our objectif is to define hybrid fragments, by applying horizontal fragmentation followed by vertical fragmentation to each database class.

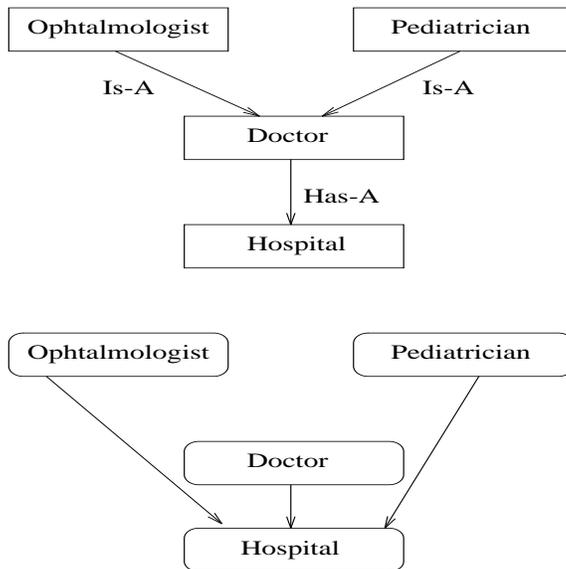


Figure 1: A Link Graph Example

A. Horizontal fragmentation

Input to the fragmentation process consists of the database classes and their relationships captured in the link graph, and the set of user requests. The output expected from this fragmentation is the set of horizontal fragments for all classes of the database.

Fragmenting horizontally a class can be done based on the applications running only on this class (primary horizontal fragmentation), or on the applications running on a referencing class (derived horizontal fragmentation).

Primary horizontal fragmentation Oszu and Valduriez described in [OV91] the steps involved in fragmenting primarily relations. These steps still hold for object oriented databases and fragmenting horizontally a database class C can be carried out by :

- Determining the most important user queries accessing the class C ,
- Selecting the methods of C used in each of the important transactions (a transaction is a set of methods),
- Analyzing semantically each of the selected methods to determine the simple predicates it involves.

These simple predicates guide the horizontal fragmentation process. Given a class $C(A_1, A_2, \dots, A_n)$ where A_i is an attribute defined on domain D_i , a simple predicate P_j defined on C has the form : $A_i \theta value$ where $\theta \in \{=, \neq, <, \leq, >, \geq\}$ and $value$ is chosen from D_i . As for relations, these simple predicates will not be used as such in the fragmentation process, because user requests include more complicated predicates which are combinations of simple predicates.

So, given the set P of simple predicates, a set of minterm predicates is constructed. A primary horizontal fragment is obtained by associating to each minterm predicate the set of object instances which verify it.

Derived horizontal fragmentation Classes which constitute the object database are not independent from each other, and relationships cannot be ignored while fragmenting a class horizontally. Generally, the object base information needed by the fragmentation process are of two types: the class lattice showing superclass-subclass relationship and

the aggregation graph which captures the attribute link between any two classes in the database. We have captured these informations in the link graph by reproducing composition links described in the aggregation graph and by expliciting those which were implicitly described by the inheritance links. The class at the tail of a link is called the owner of the link and the class at the head is called the member [CNW83]. A derived horizontal fragmentation is defined on the member class of a link according to a selection operation on the owner class. The link between the owner and the member class is defined as an explicit equi-join. Given a link L where $owner(L) = S$ and $member(L) = R$, the derived horizontal fragments of R are defined as:

$$R_i = R \uparrow S_i, 1 \leq i \leq w$$

where w is the number of fragments that will be defined on R , and \uparrow is the object pointer join operator introduced in [EB95]. The object pointer join \uparrow between the fragment S_i of the owner class S and the member class R returns the set of objects in the member class which are pointed to by objects in the fragment of the owner class S_i .

Fragmenting a member class based on the predicates of the owner class favours join queries. However, although these queries are important, there are still other queries or updates based on the predicates of the member class. In order to benefit all queries and updates, the predicates on the member class should be considered in its fragmentation process. As proposed in [ZO94], we first fragment a member class primarily and second the fragments of the member class are further fragmented through pointer joins with the fragments of owner classes.

If R_1, R_2, \dots, R_r and S_1, S_2, \dots, S_s are respectively primary fragments of the member class R and the owner class S defined above, the derived horizontal fragments of R are then defined as

$$R_{ij} = R_i \uparrow S_j, 1 \leq i \leq r, 1 \leq j \leq s$$

A most important problem arises when a class is member of more than one link. The designer must determine which links will be used for propagating fragmentation. In order not to penalize any of the applications accessing the class, we fragment the member class based on its predicates and on those of all the owner classes [ZO94].

For example, suppose there is another link L' where $owner(L') = T$ and $member(L') = R$, and T_1, T_2, \dots, T_t are the primary horizontal fragments of T . In this case, R_{ij} could further be fragmented as

$$R_{ijk} = R_{ij} \uparrow T_k, 1 \leq i \leq r, 1 \leq j \leq s, 1 \leq k \leq t$$

This fragmentation method has the advantage of taking into consideration all the requests accessing the class and not penalising any of them.

B. Vertical fragmentation

The objectif of vertical fragmentation is to break a class into a set of fragments so that all attributes and methods of the class most frequently used together are grouped together. Since every method in the object accesses a set of attributes, we first group attributes with high affinity. Each of the generated groups is then extended by incorporating methods accessing the attributes it contains.

As in relational, grouping the attributes of a class C frequently used together relies on the construction of three matrixes: the Attribute Usage Matrix (AUM), the Attribute Affinity Matrix (AAM) and the Clustered Attribute Affinity Matrix ($CAAM$). In the AUM , each line represents a class method (class methods include inherited methods), and each

column an attribute (including inherited attributes). $AUM[i, j] = 1$ if the method M_i uses the attribute A_j and 0 otherwise. Given the AUM and the access frequency of methods from the sites of the network, the AAM is constructed. $AAM[i, j]$ measures the bond between the attributes A_i and A_j . The AAM is then ordered by using the Bond Energy Algorithm [MSW72], producing the $CAAM$. The binary partitioning algorithm proposed in [NCWD84] can then be applied to split the set of attributes into nonoverlapping subsets that are accessed for the most part by distinct subsets of applications.

Extending attribute groups with methods give rise to the following problem: some methods may manipulate attributes belonging to two or more groups. The question is to which attribute group assigning such methods. The solution we propose is to duplicate these methods elsewhere they are needed. Since programs are typically very small in size compared to data, this is a reasonable solution. Vertical fragments we propose are thus disjoint regarding to attributes but overlapping regarding to methods.

As mentioned above, experience in relational databases have proved that horizontal or vertical fragmentation of a database schema is not sufficient to satisfy the requirements of user applications. The fragmentation we adopt consists of applying horizontal fragmentation to classes, and then vertical fragmentation to each horizontal fragment producing hybrid fragments.

3.3 An allocation model

Given a set S of n sites $\{S_1, S_2, \dots, S_n\}$ communicating via a network and a set F of k fragments $\{F_1, F_2, \dots, F_k\}$ communicating by method calls, the allocation problem may be formally described by a function from the set of fragments to the set of sites, $\Pi : F \rightarrow S$. If fragments are not replicated, there exists n^k possible allocations. A performance criterion used for comparing the n^k allocations is a function $f : \Pi \rightarrow R$ which associates a cost to each allocation. An optimal allocation is the one which minimizes the cost function. In general, the optimisation process aims to minimize the transactions' response time which depends on the I/O ratio and the communication delays. The expression of the cost function must be sufficiently simple in order to be easily evaluated and eventually bounded. However, taking into account all the network characteristics, especially modern networks, to develop general models yields generally to non linear functions and consequently complicates the problem. The allocation process must also take into account some imposed constraints such as disk capacity. One may impose for example that the total size of fragments assigned to a site must not exceed the disk capacity.

The distributed database system we consider is assumed to have a set of nodes connected to each other by means of a communication network. Computer hardware (terminal, minicomputer, workstation, personal computer ...) is located at each node. The hardware need not be identical at each node. This implies that processing and storage capacity may differ from one site to another. The nodes of the network can communicate at a certain cost per unit of data transmitted. Users of the system have access to fragments that can be stored at any of the nodes. Each fragment has a unique identifier, and contains a collection of data and methods. Two fragments F_i and F_j communicate if and only if there exists at least one method of F_i invoking a method of F_j and/or there exists at least one method of F_j invoking a method of F_i . This is a consequence of object encapsulation. Transactions in the database are of two types: read-only queries and update queries. Each query may consist of a series of method calls to extract the data item values and present them to the person sending the request. Similarly, each update could consist of a sequence of methods designed to extract the data item values and write them back into the appropriate database after updating them.

The model we describe in this section helps in making the following decision: Which

fragment could be allocated to which location ? In making this decision the following objectives and constraints are taken into account:

1. Minimize communication costs
2. Minimize storage costs
3. Satisfy the storage capacity at each location

A. The parameters

NF	= number of fragments,
NS	= number of sites,
i	= fragment index, $i \in \{1, \dots, NF\}$
l	= site index, $l \in \{1, \dots, NS\}$
r_i	= method r of fragment i ,
$f(r_i, s_j)$	= frequency of invoking method s of fragment j by method r of fragment i ,
$DT(r_i, s_j)$	= quantity of data transmitted between method r of fragment i and method s of fragment j (parameters+results),
$DTF(i, j)$	= quantity of data transmitted between fragment i and fragment j ,
MS_i	= set of methods of fragment i ,
FS_i	= size of fragment i ,
d_{lq}	= transmission cost per unit of data between sites l and q ,
ST_l	= cost of storing one unit of data at site l ,
Q_l	= storage capacity at node l ,
C_{il}	= cost of storing fragment i at site l , = $FS_i \times ST_l$
C_{iljq}	= communication cost between fragment i located at site l and fragment j located at site q ,

B. The decision variables

We define the following decision variables to formulate the problem:

$$x_{il} = \begin{cases} 1 & \text{if fragment } i \text{ is allocated to site } l, \\ 0 & \text{otherwise.} \end{cases}$$

C. The objective function

The objective function to be minimized consists of the sum of communication costs and storage costs.

$$Z_p = \text{Min} \sum_{l,q} \sum_{i,j} C_{iljq} x_{il} x_{jq} + \sum_i \sum_l C_{il} x_{il}$$

C_{iljq} is given by the formula :

$$C_{iljq} = d_{lq} \times DTF(i, j)$$

where

$$DTF(i, j) = \sum_{r_i \in MS_i} \sum_{s_j \in MS_j} f(r_i, s_j)DT(r_i, s_j) \\ + \sum_{s_j \in MS_j} \sum_{r_i \in MS_i} f(s_j, r_i)DT(s_j, r_i)$$

D. The constraints

$$\sum_{l=1}^{NS} x_{il} = 1 \quad \forall i = 1, \dots, NF \quad (1)$$

$$\sum_{i=1}^{NF} FS_i \times x_{il} \leq Q_l \quad \forall l = 1, \dots, NS \quad (2)$$

$$x_{il} \in \{0, 1\} \quad (3)$$

Constraint (1) states that each fragment is allocated to one site. Constraint (2) ensures that the total size of fragments allocated to one site doesn't exceed the disk capacity of that site. Constraint (3) is the binary constraint on the decision variable.

3.4 Discussion

The allocation problem in its generality is NP-Complete. It can be solved by using exact methods or heuristic methods.

Exact methods are based on the exploration of all the possible solutions. Although they result in obtaining optimal solutions, they are very costly and are inexploitable for large problems. An example is the Branch and Bound method [Sak84].

Heuristic methods yield to suboptimal solutions. The quality of a heuristic is measured uniquely by observing the results it gives, eventually by comparing it with the optimal solution when it is possible to determine it. In this category, several algorithms can be enumerated as greedy algorithms [CLR90], iterative algorithms, genetic algorithms [Ree93], simulated annealing [Dow93] and Tabu search [GL93]. Theoretically, these methods do not offer any performance guarantee, however practical experience shows that they generate solutions which are generally close to the optimum, and in reasonable delays.

The model we have proposed in the previous section is designated in the literature as "the Quadratic Generalized Affection Problem", which is known to be NP-Complete. If the size of the problem is large (which is typically the case of database allocation problem), exact methods are without issue and one has to use heuristic methods.

4 Conclusion and future work

The advantages of using object oriented concepts for tackling the design of complex applications is widely accepted. Although these applications are distributed in nature, little research has been directed to the problems of fragmenting and allocating object classes. Section 2 gives an overview of the existing fragmentation methods for both relational and object models.

The input to the distribution process is the link graph constructed by using the informations available in the global conceptual schema. The graph nodes represent database classes while edges represent composition relationships. Informations hidden by the inheritance links are explicitated through composition links too. This is described in section

3.1. Fragmenting database classes is tackled in section 3.2. Both horizontal and vertical fragmentations are applied. Special interest is granted to derived horizontal fragmentation because of the great number of edges to which a class may participate. Generated fragments are hybrid, disjoint regarding to data and overlapping regarding to methods. In section 3.3, nonreplicated data allocation is formulated as a quadratic generalized assignment problem which is known to be NP-Complete. NP-Complete problems can generally be solved by using exact or heuristic methods. Exact methods give optimal solutions but are adapted to little size problems. This unfortunately is not the case of database allocation problem for which heuristic methods must be investigated. Section 3.4 gives an overview of such methods. Our objectif in short term is to investigate heuristic methods in order to determine which one will be the most appropriate to our problem. In medium term, we aim to extend our model with replication mechanisms.

References

- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. 1990.
- [CMVN94] S. Chakraverthy, R. Muthuraj, R. Varadarajan, and S. Navathe. An objective function for vertically partitioning relations in distributed databases and its analysis. In *Distributed and parallel databases*, pages 183–207. Kluwer Academic Publishers, 1994.
- [CNW83] S. Ceri, S. Navathe, and G. Wiederhold. Distribution design of logical database schemas. *IEEE Transactions on Software Engineering*, SE-9(4):487–503, July 1983.
- [CP84] S. Ceri and G. Pelagatti. *Distributed databases: principles and systems*. McGraw-Hill, 1984.
- [CPW89] S. Ceri, B. Pernici, and G. Wiederhold. Optimisation problems and solution methods in the design of data distribution. *Information Systems*, 14(3):261–272, 1989.
- [CY86] D. Cornell and P. Yu. A vertical partitioning algorithm for relational databases. Technical Report RC 11762, IBM Research Center, December 1986.
- [Dow93] K. A. Dowsland. *Simulated Annealing*, chapter 2, pages 20–69. Blackwell, 1993.
- [EB94] C.I. Ezeife and K. Barker. Vertical class fragmentation in a distributed based system. Technical Report 94-03, University of manitoba, 1994.
- [EB95] C.I. Ezeife and K. Barke. A comprehensive approach to horizontal class fragmentation in distributed object based system. In *Distributed and parallel databases*, volume 3, pages 247–272. Kluwer Academic Publishers, 1995.
- [GL93] F. Glover and M. Laguna. *Tabu Search*, chapter 3, pages 70–150. Blackwell, 1993.
- [HN79] M. Hammer and B. Niamir. A heuristique approach to attribute partitioning. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 93–101, Boston, May 1979.
- [HS75] J. Hoffer and D. Severance. The use of cluster analysis in physical database design. In *Proceedings of the First International Conference on Very Large Databases*, pages 69–86, Framingham, August 1975.
- [KNM94] K. Karlapalem, S.B. Navathe, and M.M.A. Morsi. Issues in distribution design of object oriented databases. In T. Ozsu, U. Dayal, and P. Valduriez, editors, *Distributed object management*, pages 148–164. Morgan Kauffman Publishers, 1994.
- [LOZ93] X. Lin, M. Orłowska, and Y. Zhang. A graph based cluster approach for vertical partitioning in database design. *Data and Knowledge Engineering*, 11:151–169, 1993.
- [MSW72] W. McCormick, P. Schweitzer, and T. White. Problem decomposition and data reorganisation by a clustering technique. *Operations Research*, 20(4):741–751, July 1972.

- [NB95] J. Nachouki and H. Briand. A mixed approach for distributed object-oriented databases design. In *ISCIS X The tenth international symposium on computer and informatin science*, pages 209–216. 1995.
- [NCWD84] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical partitioning algorithms for database design. *ACM Transactions on Database Systems*, 9(4):680–710, December 1984.
- [NR89] S. Navathe and M. Ra. Vertical parttioning for database design: A graphical algorithm. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 440–450, Portland, Oregon, June 1989.
- [OV91] T. Ozsu and P. Valduriez. *Principles of Distributed Databases*. Prentice-hall edition, 1991.
- [Rav96] F. Ravat. *OD3: contribution méthodologique à la conception de bases de données orientées objet réparties*. PhD thesis, Institut de Recherche en Informatique de Toulouse (IRIT), 1996.
- [Ree93] C. R. Reeves. *Genetic Algorithms*, chapter 4, pages 151–196. Blackwell, 1993.
- [Sak84] M. Sakarovitch. *Optimisation Combinatoire*. Hermann, 1984.
- [Sav95] M. Savonnet. *Fragmentation et distribution de classes d’objets dans les bases de données orientées objet*. PhD thesis, Université de Bourgogne, 1995.
- [ZO94] Y. Zhang and M. Orłowska. On fragmentation approaches for distributed database design. *Information Sciences*, 1(3):117–132, 1994.