Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System*

Georges Gardarin, Fei Sha, Zhao-Hui Tang

CNRS-PRiSM Laboratory
University of Versailles St-Quentin
78035 Versailles Cedex, FRANCE

Firstname.Lastname@prism.uvsq.fr

Abstract

IRO-DB is an object-oriented federated database system to access multiple data sources from an ODMG compliant C++ interface. The system encompasses several components, including local database adapters to homogenize local data sources, a remote object access component to query and transfer collections of objects from site to site, and a mediator to define integrated views, decompose and optimize queries, and combine results. This paper gives an overview of the IRO-DB architecture and describes in detail the cost evaluator currently under elaboration for the next version of the distributed query optimizer. The cost model is composed of a set of mathematical formulas with coefficients to estimate the cost of the search operators. The coefficients are deduced from a calibrating object-oriented database composed of linked collections of objects. A tuning application is run on each local site to adjust the cost formulas and fix the coefficients. We report on the tuning of O2 and ObjectStore. We show that the estimation is quite accurate for path traversals with the OO7 benchmark on top of ObjectStore.

1. Introduction

Recently several research projects have experimented the use of object-oriented techniques to facilitate the complex task of building federated database systems. The object-oriented paradigm brings new solutions in several dimensions, including the modeling of local data sources as objects with a well defined and published interface, the use of a semantically rich common object model to ease application integration, the development of standards to interoperate among objects, the use of object-oriented transaction models, etc. Well known research projects in object-oriented multidatabase systems are Pegasus [ADD91], DOMS [MHG92], Carnot [WSH93], Femus [ADS93], InterBase [BCD93], etc. A complete survey of object-oriented multidatabase systems can be found in [BE95].

The current paper focuses on IRO-DB (Interoperable Relational and Object-Oriented DataBases) [Gar94], an ESPRIT project currently developed in Europe. The novelty of the IRO-DB architecture is to use the ODMG'93 standard [Cat93] as a common object model supporting the OQL pivot language to federate various object-oriented and relational data sources. It is also clearly divided into three layers, thus facilitating the cooperative development of the project in several research centers. The local layer adapts local data sources to the ODMG standard ; the communication layer efficiently transfers OQL requests and the resulting collections of objects ; the interoperable layer provides schema integration tools and a global query evaluator. A first version of the system is currently operational interconnecting the Ingres relational system and the Matisse and Ontos object systems. O2 and ObjectStore are currently in the process of being connected to IRO-DB. A CIM application has been prototyped on top of the system [RFF95].

Query optimization is a major issue in federated database systems. In the current version of IRO-DB, the query optimizer applies simple heuristics to detach subqueries that are sent to the participating systems. A synthesis query runs on the query site to elaborate the final answer to the user. More precisely, given an OQL query Q, an algebraic tree T(Q) is elaborated and simply

* *This work is partially supported by IRO-DB ESPRIST project.*

Proceedings of the 22nd VLDB Conference
Mumbai(Bombay), India, 1996

transformed by moving down selections. More sophisticated transformations could be applied to find a better execution plan, but it is very difficult for the optimizer to estimate query costs as physical characteristics of participating DBMSs are unknown. However, due to the diversity of the participating systems, a cost-based query optimizer is required. For example, path traversals are very efficient in object system, but inefficient in relational system. Thus the need of a cost model is evident for a multidatabase system such as IRO-DB. If the home site where the global query is compiled has the knowledge of the response time of each subquery sent to the participant DBMSs, it can choose the best plan to reduce local execution time and the total communication cost and to favorite parallel execution of sub-queries.

In [DKS92], a logical cost model with cost coefficients is developed for relational systems. The coefficients represent on average how much CPU time, I/O time, and other overhead is involved in query and result processing A calibrating procedure is proposed to estimate the coefficients on relational DBMSs, including AllBase, DB2, Informix, and Oracle. Another novelty of the IRO-DB project is to extend this strategy to deduce the necessary information for object systems, such as O2 and ObjectStore. The extension requires the introduction of a path traversal operator, also called *pointer chasing*. Generic formula is designed to evaluate the cost of the pointer chasing operator. Cost parameters such as object size, collection size, projection size, and fan out are also introduced. Using the IRO-DB generic formulas, we calibrate two participating systems, namely ObjectStore and O2. We finally use the OO7 benchmark to validate our genetic cost model for path traversals.

Of course, the generic cost model is less precise than a proprietary one, which has knowledge of all the detailed execution algorithms implemented inside the DBMS. But still the approach of using a generic cost model can provide good results due to two reasons. First, low level operators in most commercial DBMSs are very similar, for example, scan, index scan, nested join, sort merge join, depth first pointer chasing, etc.. This makes possible to propose similar formulas with coefficients to estimate their costs. Second, as the coefficients are derived from testing a large amount of sample queries through the real configuration of participating DBMS, the generic cost model needs less system parameters and gives better estimation in many cases. As pointed out in [LVZ93], the objective of query optimization is not to find the best execution plan, but rather to avoid the bad ones. If the generic cost model can give an accurate estimation with less than 20% of errors in a heterogeneous system such as IRO-DB, it is reasonable to consider that this approach solves the technical problem of cost-based global query optimization.

This paper is organized as follows. In section 2, we present the IRO-DB system architecture. The architecture is novel because (i) it is based on the ODMG standard as pivot model, (ii) it integrates multiple data sources through a generic toolbox to write local database adapter scaleable in functionality, (iii) it gives object-oriented facilities to define integrated views on the client home site, (iv) it integrates an active dictionary with a generic cost model for global query optimization. In section 3, we present our generic cost model for object-oriented DBMSs. The cost model is novel because it is parameterized with operators and coefficients adequate for object systems. In section 4, we show the design of the synthetic calibrating database. It is an extension of that proposed in [DKS92] with object-oriented features (e.g., links). Section 5 gives the results of the calibrating procedure on ObjectStore and O2, which validate the approach. Section 6 further validates the approach through comparing the estimated and real costs of typical path traversal queries for the OO7 benchmark. Section 7 concludes the paper.

2. IRO-DB System Architecture

IRO-DB federates relational and object-oriented databases through an interchange object-oriented data model. The selected object model is the ODMG data model [Cat93]. OQL, the ODMG query language, is provided to query interoperable databases. It is the pivot language of the IRO-DB system. OML/C++, the ODMG manipulation language embedded in C++, is used to write applications.

2.1 Schema Architecture

The IRO-DB schema architecture is represented in Figure 1. Each local database of a local DBMS is described with a schema called a *local schema*. A local schema describes the subset of a database that a local system authorizes to access from entitled cooperating systems. Each local schema is defined using the data definition language of the corresponding local DBMS. An *export schema* is obtained by translating a local schema into the ODMG object-oriented data model. The benefit of using an upcoming standard is the expected availability of ODMG compliant interfaces for many commercial object DBMSs, which will simplify the mapping, as it is already the case for the O2 system [BFE95].

To produce an homogeneous view of multiple remote databases on a client site, an *integrated schema* has to be defined. An integrated schema is an integration of parts

of export schemas included in a federated database, with a unique semantic for all object classes and operations. It can be perceived as an integrated view of the federated database. Integrated classes are derived from *imported classes*, which are defined as a partial one-to-one copy of remote exported ones [RFF95]. The *schema integrator workbench* [FBH94] is an interactive tool to help the database administrator to define integrated classes and relationships. The specification of integrated classes consists of two parts : the interface specification and the mapping specification. The interface is specified in standard ODMG syntax (i.e., ODL). The mapping specifies the combination (e.g., aggregation, generalization, fusion) of imported classes to derive an integrated class. Three kinds of OQL queries are generated by the integrator workbench to define the mapping, the first to populate the integrated class extents, the second to define the attribute mappings and the third to generate its relationships.

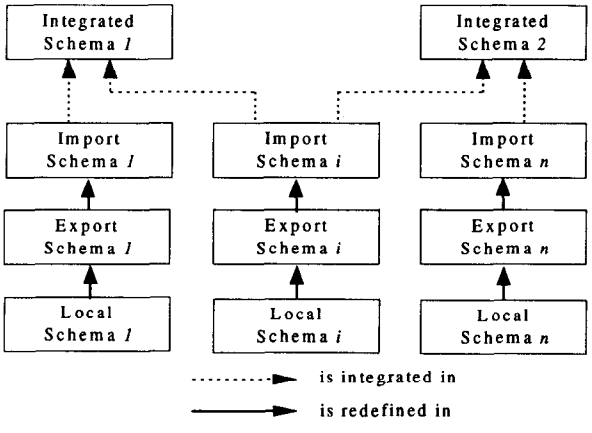

Figure 1 —The organization of schemas.

2.2 The Federated System Layers

The architecture of IRO-DB consists of three functional layers as represented in Figure 2. At *the interoperable layer*, the *integrator workbench* stands for specifying integrated views, which are generated as ODL/C++ class definitions. Integrated and export schemas are stored in a repository known as the *data dictionary*. Object manipulation facilities include the OML/C++ ODMG user interface and modules to decompose global queries in local ones and to control global transactions. More precisely, the *global query processor* maps queries referencing integrated views to subqueries referencing import schemas. It also performs query optimization and determines the best synthesis query to combine partial results on the home DBMS. The *global object manager* collects the results of local subqueries and performs the final combination of partial results on top of the home object-oriented DBMS (Ontos in the current implementation).

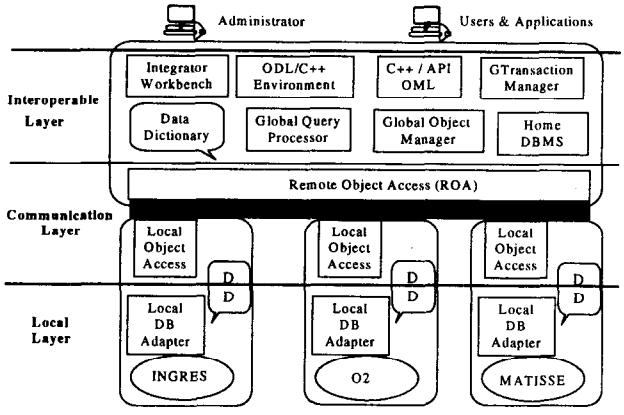

Figure 2 — The IRO-DB system architecture

The *communication layer* implements an object-oriented remote data access protocol (*Remote Object Access Protocol — ROA*) which is an extension of the SQL/Call Level Interface of the SQL Access Group to support object-oriented features. This layer provides services to establish connections with remote database servers, to submit OQL queries to these servers and to transfer results from the local layer to the interoperable layer. Results of queries are organized as collections of objects. Objects are identified through *global identifiers* composed of a server identifier, a class extent identifier and a marker inside the class extent, i.e., a bit string to locally retrieve the object within the class extent. In conformance with ODMG, sets, bags, arrays and lists are supported as collections. According to the OQL query type, a collection of object identifiers is generated first on the server. Then, the collection is transferred on demand from the global object manager using iterators. Identifiers are transferred by blocks to avoid message overhead. Imported objects are generated in virtual classes representing the classes of export schemas on the home site. The identifier of the original object is kept in the imported object, so that any operation or attribute applied to it can be shipped to the original object.

The *local layer* is composed of *local database adapters*. A local database adapter maps ODMG schemas and queries to local schemas and queries. It provides services to answer OQL queries on an abstraction of a local schema in term of an export schema specified in ODL. An export schema only describes locally implemented interfaces. Thus, only locally available functionalities are described in export schemas. This feature simplifies the implementation of local database adapters on raw data sources, as they have to provide only a restricted subset of the OQL query language to filter independent collection of objects. The

adapter can also support full functionalities of OQL on ODMG databases, as it is the case for example with the O2 system. Available data types with associated operations are described in the export schema stored in the *local data dictionary* (DD). The local data dictionary is similar in structure to the global one. A function is available to export schemas on demand from a remote site. Schemas are stamped with a version number to guarantee their consistent use throughout the network. In summary, the interoperable layer can only invoke locally available functionalities described in the local data dictionary. That means for example that methods and relationships can not be invoked if they are not described in export schemas, which is the case for relational systems.

2.3 Architecture of the Cost Evaluator

The global query processor [FFS95] decomposes end-user queries and chooses one of the best query execution plans based on cost estimations. The optimizer first accepts OQL queries referring integrated classes. Next, each integrated class is replaced by the query defining it in terms of imported classes. Integrated class attributes are also replaced by their definitions retrieved in the data dictionary. If there exists several layers of integrated views, the process is applied recursively. Finally, an object algebra tree is generated to represent the whole transformed query. Usual transformation rules are applied to generate equivalent trees. For each tree, local subqueries are detached to be sent to local sites and a synthesis query is generated to be run on the results by the home ODBMS. The role of the *cost evaluator* is to determine the presumed execution cost of each tree, in order to choose the best one using a search strategy.

The cost evaluator is based on a *cost model*. The cost model contains a set of formulas representing the cost of algebraic operators. In order to return to the optimizer the estimated cost of a tree, the cost evaluator consults the data dictionary, which gathers useful information from different databases. There are three kinds of information : the database import schemas, the statistics of collections including cardinality, selectivity, fan out, object size, method cost, etc., and the system configurations including the type of the participating DBMSs, network parameters, and system capabilities.

As the participating DBMSs are autonomous, they can not provide some detailed parameters that the cost model requires, such as the cost of a selection, a join or a path traversal. To compensate, they have to implement a specific application called the *cost tuner*. The cost tuner is designed to test sample queries for deriving values of the required coefficients. Once a new participant enters the federated system, the cost tuner invokes the calibrating procedure on this DBMS using predefined local transactions that have to be provided. The calibrating procedure instanciates a cost parameter table memorizing the local system cost model required parameters. Figure 3 introduces the architecture of the cost evaluator for deriving the query cost of a global OQL query. The cost model is a set of mathematical formulas computed on demand of the global query optimizer, using parameters stored in the data dictionary. The generic parameters are retrieved from the cost parameter tables of local sites on demand of the home site database administrator through specific queries. The data dictionary also contains, for each collection described in an import schema (IS), the number of objects, the average size of an object and the existence of indexes with their types (clustered or not). These information are important to select the adequate formulas for cost estimations. They can be refreshed from export schemas (ES) on demand of the home site database administrator.

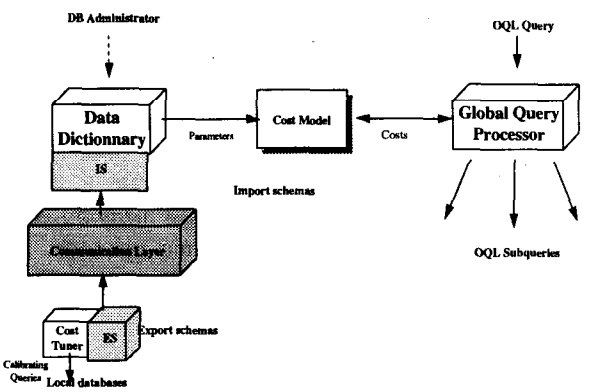

Figure 3 — Detailed architecture of the cost evaluator.

3. The Generic Cost Model

A cost model is a set of math formulas used to estimate the cost of an execution plan. Cost-based query optimizers select the most efficient execution plan among the alternatives based on the cost estimations. There are several major components of the cost : CPU cost, IO cost, and Communication cost. IRO-DB requires a generic cost model as participating DBMSs and systems are not known in detail. Thus, we do not separate CPU and IO costs, which are buried in global cost formula parameters. An execution plan consists of a set of basic operators ; the cost of each is estimated separately and their sum gives the total response time of executing the query. Communication cost can be reflected more precisely according to the network description ; this topic is out of the scope of this paper.

3.1 Generic Operators

One of the important aspects of OODBMS is object navigation [ZW86, KKS90, KGM91]. Each object has a unique identifier (OID) and objects from different classes can refer each other freely. OID can be either physical as in ObjectStore or logical as in Versant. No matter which OID technique is applied, an OODBMS provides an efficient way for decoding OIDs and charging the state of object into memory. This mechanism is the base of the navigation procedure in object database systems. Navigation among objects is often called *pointer chasing.*

To speed up the query execution, object-oriented DBMSs support classic indexes to class instances. Indexes are generally organized as B-trees containing attribute values with OID lists. Further, some systems like ObjectStore [Obj94] provide path indexes, which follow a sequence of objects linked by relationships. This accelerates the evaluation of path expressions.

Due to these different aspects of OODBMS, the execution model of OODBMS is richer than that of RDBMS. In a relational system, the most common operators are scan and join (both of them have different algorithms), whilst in an object system, the nary operator *pointer chasing* exists. Pointer chasing profits of pre-computed links, thus has very low CPU costs compared to traditional binary joins, but it may generate a lot of random disk accesses. Although there exist many different commercial DBMSs, the basic algebraic operators are always the same. Our model covers the following common operators :

- *unary operators including sequential scan, index scan and clustered index scan ;*
- *binary operators including nested join, index join and sort-merge join ;*
- *nary operator corresponding to pointer chasing.*

Pointer chasing in collections $C_1, C_2, \ldots C_n$ starts from a collection C_i and performs a depth first traversal of the tree following (multivalued) attribute pointers from C_i to C_j. The average number of pointers in one object of collection C_i to objects of collection C_j is supposed to be fan_{ij}. A filtering predicate is supposed to be applied in collection C_i with selectivity Sel_i.

3.2 The Cost Formulas

We use ||C|| to denote the cardinality of collection C, |C| the number of data pages of collection C, *Sel* the predicate selectivity of a collection, *proj* the size of all the projected attributes. For each operator, we determine the cost using a parametrized cost formula. The role of the tuning application is to settle the parameters when a new system joins the federation or more generally, on demand of the database administrator. Each of the cost formulas contains three major components : the first item is the initial cost of the operator, the second term is the cost of processing the predicate and the third item is the cost of processing the selected objects.

For evaluating the cost of a selection, the following cost parameters are introduced :

- *SS_0, IS_0, CIS_0 are the initial cost of sequential scan, index scan and clustered index scan ;*
- *SS_1 is the amortized I/O cost and CPU cost of processing each page of the collection (object fetch and condition checking) for sequential scan;*
- *IS_1, CIS_1 are costs of index lookup respectively in the case of index scan and clustered index scan ;*
- *SS_2, IS_2, CIS_2 represent the cost of processing result tuples respectively in the case of sequential scan, index scan and clustered index scan.*

The cost formulas of the unary operators are then categorized as follows :

- *Sequential scan*
  $$SS = SS_0 + SS_1 * |C| + SS_2 * \|C\| * Sel * proj$$
- *Index scan*
  $$IS = IS_0 + IS_1 + IS_2 * \|C\| * Sel * proj$$
- *Clustered index scan*
  $$CIS = CIS_0 + CIS_1 + CIS_2 * |C| * Sel * proj$$

These formulas are very similar to those given in [DKS92], except that we differentiate the number of objects and number of pages in a collection and we introduce the projection cost. The experiments reported in the sequel demonstrated the need for this differentiation in object systems. In the formula for sequential scan, the cost of object fetching and predicate evaluation is $SS1_{cpu} * \|C\| + SS1_{io} * |C|$. To merge I/O and CPU costs, we define SS_1 as I/O plus CPU cost per data page ; thus the amortized cost will be $SS_1 * |C|$. As the I/O cost is much higher than the CPU cost, this approximation works well when object size is not too small. Index traversal requires a predicate processing cost almost independent of the number of objects in a collection, whilst it is proportional to that number for sequential scan. With clustered index scan, resulting tuples are processed on a page basis, which explains the $|C|$ factor in the last formula.

To elaborate the cost formulas of the binary join operators, we introduce the following parameters :

- *Sort is the cost of sorting a collection and it can be zero if there is an index on the joining attribute ;*

- *SJ_2 is the cost of merging join objects;*
- *J_{12} is the selectivity of the join operation on C_1 and C_2;*
- *NJ_1 is the amortized IO and CPU cost to join tuples in one page of each collection;*
- *NJ_2, IJ_2 represent the cost of processing result tuples for nest join and index join;*

The join cost formulas are then categorized as follows :

- *Nested join (sequential scan on C2)*

$$NJ = SS(C_1) + SS_0(C_2) + |C_1| * |C_2| * NJ_1 + \|C_1\| * \|C_2\| * J_{12} * NJ_2 * proj$$

- *Index join (index scan on C2)*

$$IJ = SS(C_1) + IS_0(C_2) + \|C_1\| * IS_1 + \|C_1\| * \|C_2\| * J_{12} * IJ_2 * proj$$

- *Sort merge join*

$$SJ = Sort(C_1) + Sort(C_2) + SS(C_1) + SS(C_2) + \|C_1\| * \|C_2\| * J_{12} * SJ_2 * proj$$

The novelty of object systems is mainly the integration of navigation in the query evaluation process. Thus, as explained above, we introduce a pointer chasing operator to model pointer traversals. For pointer chasing evaluation, we introduce the following cost parameters :

- *PC_0 is the initial cost of pointer chasing, i.e., the cost of processing the operator and setting up the traversals.*
- *PC_1 is the amortized I/O cost and CPU cost of fetching an object by its OID and verifying the predicate ;*
- *PC_2 is the cost of processing a result tuple for pointer chasing.*
- *n is number of collection.*

Thus, the cost of pointer chasing is given by the complex formula :

$$PC = PC_0 + PC_1 * \|C_1\| * \left(1 + \sum_{i=1}^{n-1} \prod_{j=1}^{i} \left(fan_{j,j+1} * Sel_j\right)\right) + PC_2 * \|C_1\| * Sel_1 * proj * \prod_{i=1}^{n-1} \left(fan_{i,i+1} * Sel_{i+1}\right)$$

PC_0 denotes the initial cost of pointer chasing; $PC_1 * \|C_1\| * \left(1 + \sum_{i=1}^{n-1} \prod_{j=1}^{i} \left(fan_{j,j+1} * Sel_j\right)\right)$ gives the cost component of charging the states of objects into memory through their OIDs and evaluating the predicate; $PC_2 * \|C_1\| * Sel_1 * proj * \prod_{i=1}^{n-1} \left(fan_{i,i+1} * Sel_{i+1}\right)$ gives the cost of processing the selected objects from this pointer chasing operator.

3.3 The Cost of Methods

Object-oriented database systems also support operations to encapsulate data. IRO-DB supports remote operation invocation through the OQL query language. The operation code is a method in object systems or a stored procedure in relational systems. The operation interface is described in export schema. Since these functions are written by users, database system cannot provide correct formulas for estimating their costs. The only possibility to know the cost is that the database administrator documents their costs in the data dictionary, when describing the interface. Each time a function is invoked, the tuner compares the actual cost and the documented one and updates the dictionary if there is a significant difference between the two costs. Further work remains to be done to confirm the validity of the approach for methods.

4. The Calibrating Database

In order to derive the values of the coefficients used in our generic cost model presented in the previous section, each local DBMS must support a synthetic database. The IRO-DB cost tuner application queries it when required by the local database administrator to compute the cost parameters. The cost parameters are stored in the local cost parameter database. Through IRO-DB, this database is exported to home sites to give input parameters to the global cost evaluator, as explained at end of section 2.

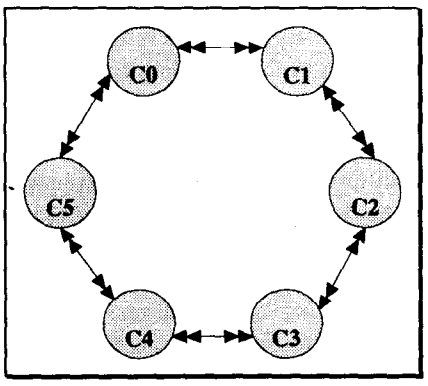

Figure 4 — The calibrating database schema.

As mentioned in [DKS92], to calibrate a given system, there are two major problems : one is that it is difficult to predict the way a given query is executed by the system ; the other problem is to eliminate the effect of data placement, pagination and other physical storage factors

on query execution. The first problem is not severe for object databases since, in actual commercial object-oriented DBMSs, implemented query optimizers are not yet sophisticated. Thus, it is possible to predict the generated execution plan. For fixing the second problem, we design a calibrating database whose attribute values guarantee certain distribution criteria. In this section, we present our calibrating database which is an object extension of the relational database of [DKS92].

The calibrating database is composed of six collections $C_0, C_1 ... C_5$. They are interconnected as shown in Figure 4. Each collection Ci has inverse link with its two neighbor collections. Apart from the attributes of OID types, each object of these collections has seven attributes with atomic values. Suppose the cardinality of a collection is 2^n, the seven attributes have the following characteristics :

- *A1 : integer [0,n], indexed, clustered.*
- *A2 : integer [0, 2^n-1], indexed, de facto clustered.*
- *A3 : integer [0,n], indexed, unclustered.*
- *A4 : integer [0,n], no index.*
- *A5 : integer [0, 2^n-1], indexed, unclustered*
- *A6 : integer [0, 2^n-1], no index.*
- *A7 : a string with a certain length to meet the object size.*

The values of these attributes are defined as follows. The i-th object of a collection C is a tuple composed of the following values :

- *A1[n,i]=x+1, such that $2^x \le i < 2^{x+1}$*
- *A2[n,i]=i*
- *A3[n,i]=n-x, such that i mod $2^{x+1}=2^x-1$*
- *A4[n,i]=A3[n,i]*
- *A5[n,i]=$2^{n-k}+j$ such that $i=2^{k-1}*(1+2*j)$ and A5[n,0]=0.*
- *A6[n,i]=A5[n,i].*

Table 1 shows the values of these integer attributes when n=4 (the cardinality of the collection equals to 16).

The reason for choosing the above function for A1 is to model those cases where all the objects are well clustered, i.e., just fit in continuous pages for a given value of A1. The rest of the design guarantees the uniform distribution of attributes for different kinds of queries including equality and range queries, and for different access methods using or not using indexes. Any selection on attribute A1 will result in clustered index scan. A2 represents the indice number inside the collection. The values in A3 and A4 guarantee that for any integer between 0 to n, tuples containing this value on attributes A3 and A4 are uniformly distributed among all the disk pages. A5 and A6 are designed for range queries. Any integer between 0 to n, the set of values [0, 2^i-1] are distributed uniformly inside the collection. Compared to the design in [DKS92], our functions for assigning values to atomic attributes are simpler and easier to calculate, and yet with the same effect in attribute value distribution.

A1	A2	A3	A4	A5	A6
0	0	4	4	0	0
1	1	3	3	8	8
2	2	4	4	4	4
2	3	2	2	9	9
3	4	4	4	2	2
3	5	3	3	10	10
3	6	4	4	5	5
3	7	1	1	11	11
4	8	4	4	1	1
4	9	3	3	12	12
4	10	4	4	6	6
4	11	2	2	13	13
4	12	4	4	3	3
4	13	3	3	14	14
4	14	4	4	7	7
4	15	0	0	15	15

Table 1 — Integer attribute values of a collection with 16 objects.

	Object Size	Cardinality	Disk Page Number
C0	252	8192	512
C1	504	8192	1024
C2	252	16384	1024
C3	504	16384	2048
C4	252	32768	2048
C5	504	32768	4096

Table 2 — Calibrating database configuration.

Objects are interconnected each other. The maximum fan out number between two neighbor collections is set to a fixed value. For practical calibrations, it is set to 4. Experiences have shown that changing this value does not significantly change the coefficients in the path traversal cost formulas, as exemplified below. Thus, each object refers to 4 objects in the next collection, and the links among objects are randomly generated. The object sizes and cardinalities of the collections are different. Table 2 gives the database configuration. The whole calibrating database holds about 60M bytes.

A large amount of sample queries are designed for the calibrating. These queries include simple selection queries on one collection, join queries involving two collections, and navigation queries involving several collections. Both equality and range predicates are included in the sample set. Figure 5 gives some of these queries expressed in OQL.

5. Calibrating ObjectStore and O2

In this section, we present some results of our experimentation on calibrating two commercial object-oriented DBMSs : ObjectStore and O2. The objective of this experimentation is not to compare performance, but to derive the coefficients values to settle our generic cost

```
select A1 from C5 where A1=n ;          select A1 from C4 where A1=n ;
select A1 from C3 where A1=n ;          select A1 from C1 where A1=n ;
select A2 from C5 where A2<n ;          select A5 from C5 where A5<n ;
select A6 from C5 where A6<n ;          select A4 from C5 where A4=n ;
select A1,A2 from C5 where A4=n ;
select A1,A2,A3,A4, from C5 where A4=n ;
select C4.A1, C5.A2 from C4, C5 where C4.A1=n & C4.A1=C5.A1 ;
select C4.A1, C5.A3 from C4, C5 where C4.A1=n & C4.R1=C5.A3 ;
select C4.A2, C5.A2 from C4, C5 where C4.A2<n & C4.A2=C5.A2 ;
select C4.A5, C5.A5 from C4, C5 where C4.A5<n & C4.A5=C5.A5 ;
select C4.A6, C5.A6 from C4, C5 where C4.A6<n & C4.A6=C5.A6 ;
select y.A1 from x in C0, y in x.C1 where x.A3 = n ;
select y.A1 from x in C0, y in x.C1 where x.A4 < n ;
select z.A1 from x in C0, y in x.C1, z in y.C2 where x.A3 = n ;
select z.A1 from x in C0, y in x.C1, z in y.C2 where x.A4 < n ;
select u.A1 from x in C0, y in x.C1, z in y.C2, u in z.C3 where x.A3 = n ;
select u.A1 from x in C0, y in x.C1, z in y.C2, u in z.C3 where x.A4 < n ;
...
```

Figure 5 — Some sample calibrating queries.

formulas. In our experimentation, ObjectStore (version 3.0.2) runs on Windows NT 3.5 using a Pentium 90 machine with 24 M of main memory, while O2 (version 4.2) runs on a SUN Sparc 4 with 32 M of memory.

Figure 6 shows the result of different scan operations. The legends of the curves for O2 and ObjectStore start with 'O2' and 'OS' respectively. All the curves with 'T' in the legend represent the theoretical cost, which is from our cost estimation.

Figure 6(a) shows the results of sequential scan. In our formula, the cost of scan and predicate evaluation is proportional to collection size and the cost for processing the result tuples is proportional to the number of result tuples. This matches quite well with the experimental results. Since the cardinality of C5 is twice as that of C3, and their object sizes are the same, we can see from Figure 6(a) that the inclination of OSC5 is almost twice as much as that of OSC3 ; the same tendency is observed in O2C5 and O2C3. Figure 6(b) shows the result of index scan. The cost of initialization, index access and predicate evaluation is independent of the collection while the cost for processing the result tuples is proportional to the number of result tuples. The response time is more sensitive to the collection cardinality than the actual size of the object. Figure 6(c) gives the experimental results of clustered index scan. Due to the fact that the result objects are clustered in the physical storage, the result processing cost is proportional to the number of disk pages of the collection rather than to the number of result tuples, as in the case of non clustered scans. Comparing all these experimental and theoretical curves, we observe that they match quite well. Figure 7 shows the response time of pointer chasing. The legend 'P4fan3' represents the curve for path length equals to 4 and fan out equals to 3. The same notation is used to name other curves.

Figures 7(a) and 7(c) give the response time of pointer chasing when path length varies from 4 to 5 and fan out from 3 to 4. From both figures for ObjectStore and O2, we discover that the response time of pointer chasing of path length 5 and fan out 3 is quite close to that of path length 4 and fan out 4. From the cost formula for pointer chasing, we can see that the major cost happens to be in the second item, which represents the cost of following object links. We calculate that the total number of traversed object links equals to n* 364 for P5fan3 and n*341 for P4fan4, where n is the number of qualified objects in the starting collection. This explains the curve similarity phenomenon. As the number of accessed objects in the starting collection of the path grows, the cost of the traversal increases proportionally. Figure 7(b) and 7(d) show the result of pointer chasing for different path lengths and fan outs. The number of qualified objects in the starting collection of the path remains to 32. The horizontal axes is the fan out and each curve represents the response time of different path lengths when fan out keeps on varying. We notice that the response time of pointer chasing is very sensitive to the fan out when the path length is long. TP4 and TP5 follow the experiment result and this again proves the correctness of the pointer chasing formula.

The above experimentation helps to derive the values of the cost formula parameters and proves the validity of our set of cost formulas as well. Thus, it is appropriate to tune our cost formulas with coefficients derived from our experimentation. The calculated values of these coefficients are given in the table 3. As explained in section 2, they constitute the results of the tuner application, which export them as the cost parameter database to be used by the global query optimizer.

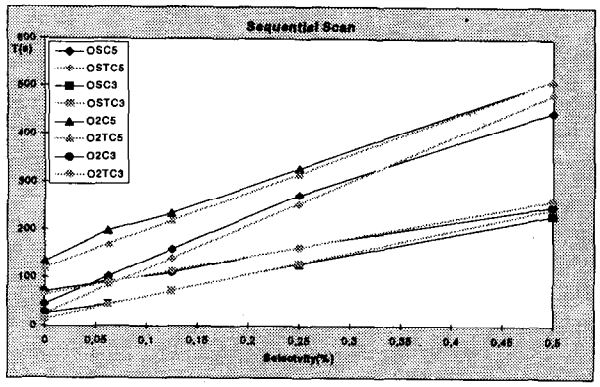

Figure 6(a) — Sequential scans.

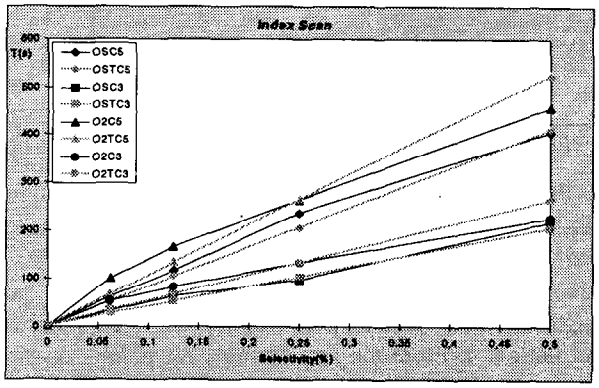

Figure 6(b) — Index scans.

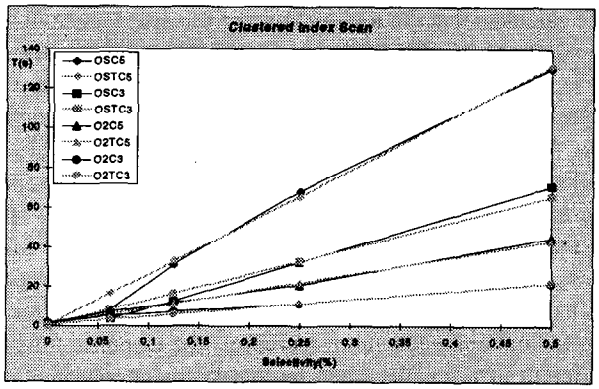

Figure 6(c) — Clustered index scans.

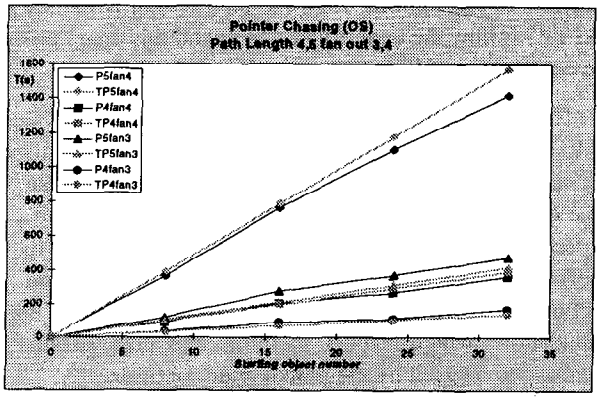

Figure 7(a) — Pointer chasing.

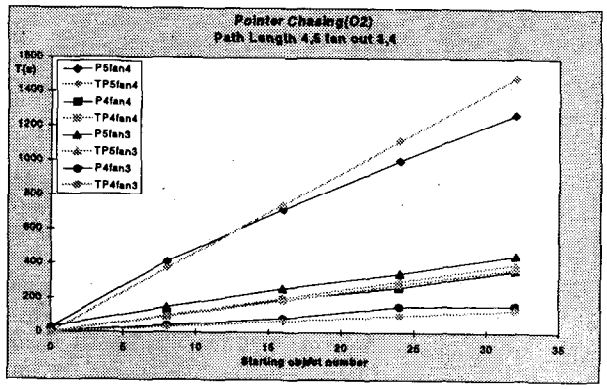

Figure 7(b) — Pointer chasing.

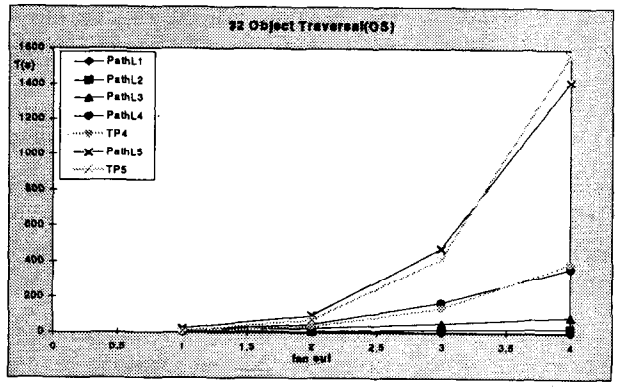

Figure 7(c) — Pointer chasing.

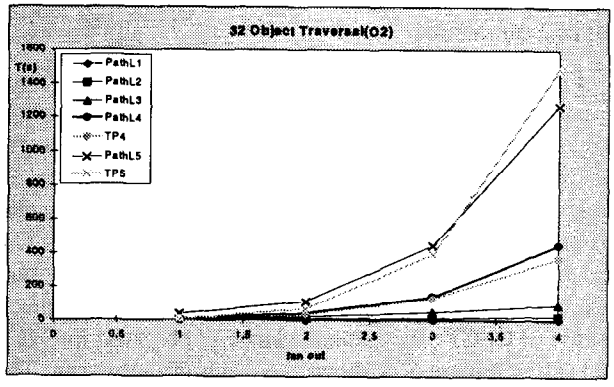

Figure 7(d) — Pointer chasing.

	Object Store	O2		Object Store	O2
SS0	0,2	0,3	**IS0**	0,6	0,7
SS1	0,018	0,027	**IS1**	0,08	0,08
SS2	0,024	0,024	**IS2**	0,025	0,032
CIS0	0,6	0,6	**PC0**	0,25	0,05
CIS1	0,08	0,07	**PC1**	0,008	0,007
CIS2	0,008	0,0026	**PC2**	0,028	0,027

Table 3 — The Cost coefficients of ObjectStore and O2.

6. Validation With The OO7 Benchmark

To further validate our approach, we use our generic cost model with the derived coefficients to estimate the cost of queries on the OO7 benchmark. The OO7 benchmark was designed by the Wisconsin database research group [CDN93]. In its schema, each *composite part* is an aggregate of a number of *atomic parts*, which could represent variables, statements or expressions in a CASE procedure. Over these composite parts, a *base assembly* hierarchy has been added to represent higher level design objects such as an ALU in a CAD application. A *complex assembly* is an aggregate of other assembly objects. A *module* describes a complete assembly hierarchy. Since the database schema is quite rich and objects are interconnected, the OO7 benchmark is very suitable for testing queries on object traversals. There are three different configurations proposed by OO7 : small, medium and large databases. In our experimentation, we choose the medium size, which is about 50 M. The benchmark is generated on ObjectStore for Windows NT platform.

Among all the collections in OO7, atomicPart is the largest since the relationships between collections from up to down in the schema graph are all one-to-many. For the validation, we generate queries on *atomicPart* for testing the unary operator such as sequential scan, index scan, clustered index scan. Since the binary join operators such as nested join and sort merge join are not implemented on ObjectStore, all the queries with path expressions are executed using pointer chasing. If the fan out between two neighbor collections is greater than 1, the depth-first-traversal method is applied [Obj94].

From Figures 8(a), 8(b), 8(c), we can see that the response times of queries involving only one collection match quite well with the estimated costs by our generic cost model. And for low selectivities, the difference between the estimated cost and the real cost is less than 5%. When the selectivity goes higher, the error increases also. But in the worst case as in the clustered index scan, the correctness can still reach about 80%. Figure 8(d) shows the result of executing a query with path expression *module.assembly.compositePart.atomicPart*. The fan out between any of the two neighbor collections is set to 3. The horizontal axes gives the number of objects in the starting collection *module*. The theoretical curve is calculated using the pointer chasing formula described in section 2 with the coefficients derived from calibrating procedure. Comparing these two curves, we can see that the proposed formula represents quite well the propriety of pointer chasing. Second, the coefficients derived from the calibrating database can be used for estimating the query cost of path traversals within the OO7 database with less than 10% of error in most cases.

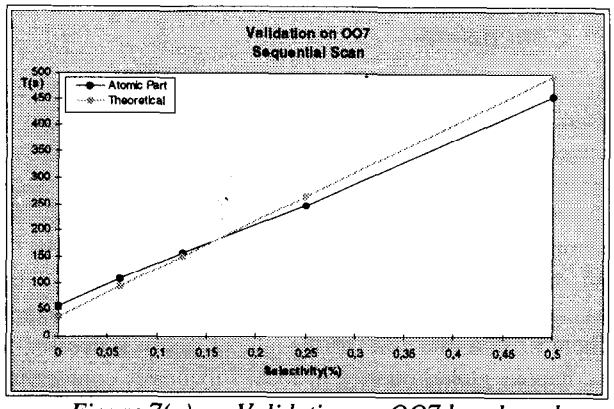

Figure 7(a) — Validation on OO7 benchmark.

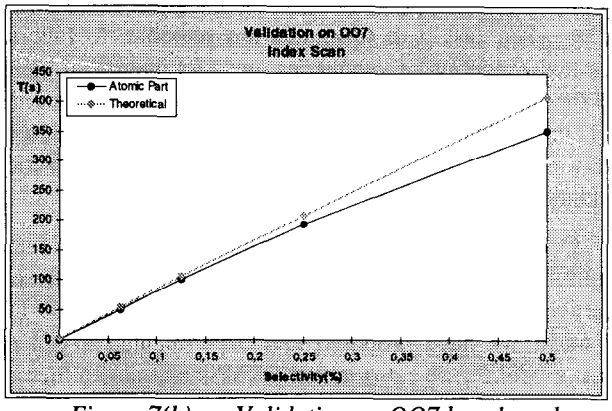

Figure 7(b) — Validation on OO7 benchmark.

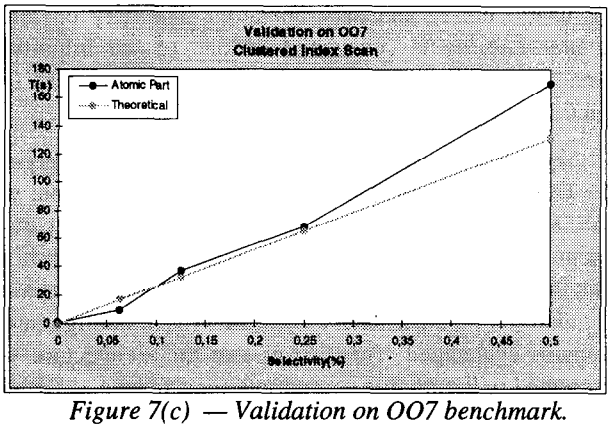

Figure 7(c) — Validation on OO7 benchmark.

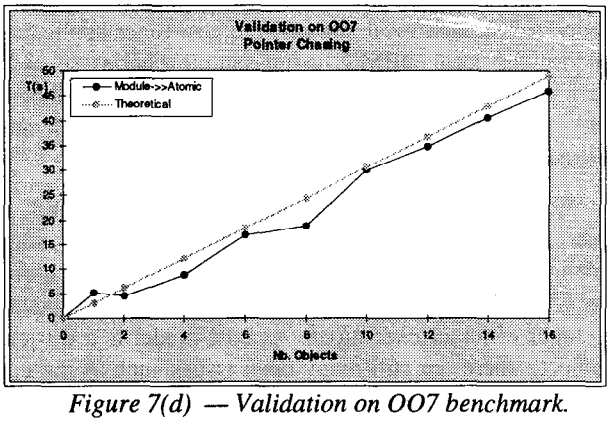

Figure 7(d) — Validation on OO7 benchmark.

7. Conclusion

IRO-DB is an object-oriented federated database system based on the ODMG standard. The global data model is expressive enough to capture the meaning of all local data sources. It is scaleable in functionality, so that local adapters can be developed with few functionalities — for example, without supporting methods and relationships — and enhanced later. The OQL subqueries sent to local systems only invoke functionalities described in export schemas. It is the role of the global query processor to decompose global queries in local subqueries conforming to the various export schemas. Global query optimization is important to improve performance, e.g., to avoid costly joins when fast path traversals are possible, to favorite parallel execution of subqueries, etc.. As integrated views of the federated databases can be easily defined using the schema integrator workbench, global queries may become very complex, which further requires good optimization techniques.

One of the major problems of global query optimization is that the cost functions of the different participating DBMSs are not available. In the IRO-DB context, we have proposed and experimented a generic cost model for object-oriented DBMSs and also a method for deriving the values of the cost coefficients for this model. Our generic cost model includes the cost formulas for unary, binary and nary operators. The nary operator *pointer chasing* is considered as one of the important aspects of object databases. It is generally applied for queries with path expression.

We partially validate our approach through querying the OO7 benchmark. Throughout the experimentation, we can see that our cost model provides quite accurate estimations for different kinds of queries. The results show that the coefficients derived from the calibrating databases can be used on other databases if the system configuration remains unchanged. This proves that the architecture we presented in Figure 3 is appropriate for a federated DBMS to have knowledge of the performance characteristics of each participating site. The calibrating database is generated at each local site and the cost tuner application queries the local DBMSs using sample queries so as to keep the values of coefficients valid in a cost parameter table exported to query sites through the federated database system.

We will continue our validation on different kinds of queries on O2 and ObjectStore, as well as on some relational databases. These results will help the global query optimizer to choose the optimal execution schedule to favor efficient executions. As cost model is a difficult subject, there are still many open issues such as the cost of methods, the cost formula for some special operators implemented in different DBMSs, etc. Today, in most commercial object-oriented DBMSs, the implemented query optimizer is not yet complicated ; thus it is possible to predict the selected execution plan of a given query. But once these participating query optimizers will become sophisticated, our approach could face a problem to guarantee that the subqueries sent to each site will be executed as supposed by the global query optimizer. The question is then to determine when the estimated cost will become a bad approximation. Further knowledge of the behavior of different DBMS optimizers will then be required to enhance the global query optimizer.

Acknowledgment The authors wish to thank Béatrice Finance, Véronique Smahi, Jérôme Fessy and Sofien Gannouni from the PRiSM laboratory; Regis Legoff from EDS; Wolfgang Klas, Peter Fankhauser, Ralph Busse and Gerald Huck from the GMD-IPSI Laboratory; Virginia Escuder from Ibermatica; and Antonis Ramfos from Intrasoft for their collaboration. Without them IRO-DB would not be such as it is now.

References

[ADD91] Ahmed R., DeSchedt P., Du W., Kent W., Ketabchi M., Litwin W., Rafii A., Shan M., "The Pegasus Heterogeneous Multidatabase System", IEEE Computer, 24:12, Dec. 1991, pp. 19-27.

[ADS93] Andersson M., Dupont Y., Spaccapietra S., Yétongnon K., Tresh M., Ye H., "The FEMUS approach in Building a Federated Multilingual Database System", 3rd International Workshop on RIDE-IMS, Vienna, Austria, April 1993.

[BCD89] Bancilhon F., Cluet S., Delobel C., "A query language for the O2 object-oriented database system", Proceedings of 2nd Intl. Workshop on Database Programming Languages, 1989.

[BCD93] Bukhres O.A., Chen J., Du W., Elmagarmid A.K., Pezzoli R., "InterBase: An Execution Environment for Heterogeneous Software Systems", IEEE Computer, 26:8, August 1993, pp. 57-69.

[BE95] Bukhres O.A., Elmagarmid A.K. Ed., Object-Oriented Multibase Systems, Prentice Hall, 1995.

[BFE95] Bancilhon F., Ferran G., "The ODMG Standard for Object Databases", Proceedings of the 4th International Conference on Database Systems for Advanced Applications (DASFAA'95), World Scientific Pub., Singapore, April 1995.

[BFO92] Bertino E., Foscoli P., "An analytical model of object-oriented query costs", Proceedings of Workshop on Persistent Object System, in Computing Series, Springer Verlag, 1992.

[Cat93] Cattell R.G. Ed., Object Databases : The ODMG-93 Standard, Morgan & Kaufman, 1993.

[CDN93] Carey M.J., DeWitte D.J., Naughton J.F., "The OO7 Benchmark", Sigmod Record 22(2) :12-21, 1994.

[DKS92] Du W., Krishnamurthy R., Shan M.C., "Query optimization in heterogeneous DBMS", Proceedings of 18th International Conference on Very Large Databases, Vancouver, 1992.

[DL87] Dwyer P., Larson J., "Some experiences with a distributed database tested system", Proceedings of the IEEE, 75(5), pp. 633--647, May 1987.

[FBH94] Fankhauser P., Busse R., Huck G., "IOM Design Specification", Technical report, IRO-DB Esprit Project (EP8629), IRO/SPEC/GMD/ FBH940629, GMD-IPSI, Darmstadt, Germany, July 1994.

[FFS95] Finance B., Fessy J., Smahi V., "Query Processing in IRO-DB", Conf. Bases de Données Avancées, INRIA Ed., Nancy, France, Sept. 1995.

[FLU94] Frohn J., Lausen G., Uphoff H., "Access to objects by path expressions and rules", Proceedings of 20th International Conference on Very Large Databases, 1994.

[Gar94] Gardarin G.et al., "IRO-DB : A Distributed System Federating Object and Relational Databases", In Object-Oriented Multibase Systems, O. Bukres and A. Elmagarmid Ed., Prentice Hall, 1995.

[GGT95] Gardarin G., Gruser J.R, Tang Z.H., "A Cost Model for Clustered Object-Oriented Databases", Proceedings of Int. Conference on Very Large Databases VLDB, Zurich, Switzerland, 1995.

[KGM91] T. Keller, G. Graefe, D. Maier, "Efficient Assembly of Complex Objects", Proceedings of ACM-SIGMOD International Conference on Management of Data, 148-157., 1991.

[KKS90] M. Kifer, W. Kim, Y. Sagiv, "Querying object-oriented databases", Proceedings of ACM-SIGMOD International Conference, 1990.

[LVZ93] R. Lanzelotte, P. Valduriez and M. Zait. On the effectiveness optimization search strategies for parallel execution spaces. In Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

[MHG92] Manola F., Heiler S., Georgakopoulos D., Hornick M., Brodie M., "Distributed Object Management", International Journal of Intelligent and Cooperative Information Systems, 1:1, March 1992, pp. 5-42.

[Obj94] ObjectStore Editor, "ObjectStore User Guide", ObjectStore Pub., Release 3.0, 1994.

[RFF95] Ramfos A., Fessy J., Finance B., Smahi V., "IRO-DB : a solution for Computer Integrated Manufacturing Applications", Proceedings of the Third International Conference on Cooperative Information System (CoopIS-95), Vienna, Austria, May, 1995..

[Sha86] Shapiro L., "Join Processing in Database Systems with Large Main Memories", ACM Transactions on Database Systems, Vol 11 n°3, p239-264, September 1986.

[Shi81] Shipman D., "The functional data model and the data language DAPLEX", IEEE Transactions on database systems, 6(1), pp. 140-173, March 1981.

[Swa89] Swami A., "A validated cost model for main memory databases", Performance Evaluation Review, May 1989.

[Wid95] Widom J., "Research Problems in Data Warehousing", Proc. of the 4th Int. Conf. on Information and Knowledge Management (CIKM), Nov. 1995.

[WSH93] Woelk D., Shen W., Huhns M., Cannata P., "Model Driven Enterprise Information in Carnot", 1st International Conference on Enterprise Integration Modeling, The MIT Press, Cambridge, Mass., 1992, pp. 301-309.

[ZW86] S. Zdonik and P. Wegner. Language and methodology for object-oriented database enviroments. In Proceeding of the Hawaii International Conference on System Sciences, 1986