Cost-Based Query Optimization in a Heterogeneous Distributed Semi-Structured Environment

Tianxiao Liu
(Supervised by Dominique Laurent and Tuyet Tram Dang Ngoc)
ETIS Laboratory - University of Cergy-Pontoise, Cergy-Pontoise, France
Xcalia S.A., Paris, France
tianxiao.liu@u-cergy.fr
tianxiao.liu@xcalia.com

ABSTRACT

How to efficiently process queries in an heterogeneous and distributed data integration environment is an interesting and unsolved topic. Our research project proposes an approach for providing a generic cost framework for query optimization in an XML-based mediation system called XLive, which integrates heterogeneous data sources. Our approach relies on cost annotation on an XQuery logical representation called Tree Graph View (TGV). A generic language is used to define an XML-based uniform format for cost communication within the XLive system. This cost framework is suitable for various search strategies to choose the best execution plan for minimizing the execution cost.

1. INTRODUCTION

The architecture of mediation system has been proposed in [30] for solving the problem of integrating and querying heterogeneous data sources. The key aspect of such an architecture is that the mediator provides to users an uniform query interface and wrappers deal with the problem of data source heterogeneity. Currently, the semi-structured data model represented by XML format is considered as a standard data exchange model. The project XLive [8], a mediation system based on XML standard, has a mediator that can accept queries in the form of XQuery [29] and return answers. The wrappers give the XLive mediator an XML-based uniform access to heterogeneous data sources.

For a given user query, the mediator can generate various execution plans (referred to as "plan" in the remainder of this paper) to execute it, and these plans can differ widely in execution cost (execution time, price of costly connections, communication cost, etc.). An optimization procedure is thus necessary to determine the most efficient plan with the least cost. However, how to choose the best plan based on the cost is still an open issue. In relational or object-oriented databases, the cost of a plan can be estimated by using a cost model. This estimation is processed with database statistics and cost formulas for each operator appearing in the plan. But in a heterogeneous and distributed environment, the cost estimation is much more difficult, due to the lack of underlying databases statistics and accurate cost formulas.

In this paper, we propose a framework for cost-based query optimization in the XML-based XLive mediation system. This framework allows to take into account various cost models for different types of data sources with diverse autonomy degrees. These cost models are stored as annotations in an XQuery logical representation called Tree Graph View (TGV) [7][27]. Moreover, cost models are exchanged between different components of the XLive system.

We summarize different cost models for different types of data sources (relational, object oriented and semi-structured) and different autonomy degrees of these sources (proprietary, non-proprietary and autonomous). The overall cost estimation relies on the cost annotation stored in corresponding components TGV. This cost annotation derives from a generic annotation model that can annotate any component (i.e. one or a group of operators) of a TGV.

In order to perform the cost communication within the XLive system during query optimization, we define an XML-based language to express the cost information in a uniform, complete and generic manner. This language, which is generic enough to take into account any type of cost information, is the standard format for the exchange of cost information in XLive.

The reminder of the paper is organized as follows: Section 2.1 gives a synthesis of related work. In Section 3, we introduce XLive system with its TGV modeling of XQuery and we motivate our approach to cost-based optimization. Section 4 gives the detailed description of our proposed solution. We summarize and discuss directions for future work in Section 5.

2. RELATED WORK

2.1 Cost Models for Heterogeneous Autonomous Data Sources

We summarize different existing cost models for various types of data sources in Fig. 1. This summary is not only based on the type of data sources but also on the autonomy degree of these sources. In addition, this summary gives some relations between different works on cost-based query optimization. The cost models with the name "operation"

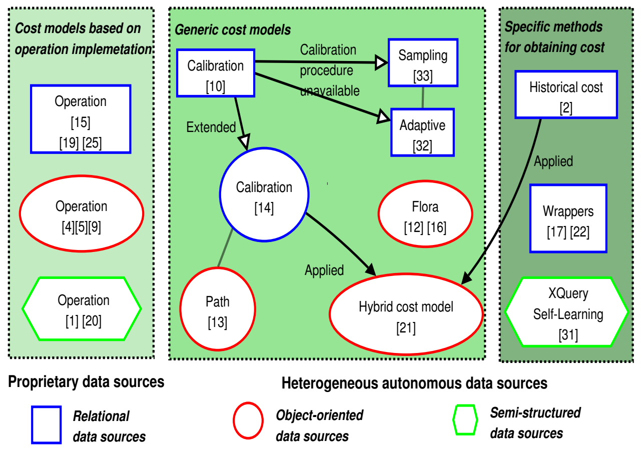


Figure 1: Cost models for heterogeneous sources

contain accurate cost formulas for calculating the execution cost of operators appearing in the plan. Generally, cost information such as source statistics is necessary for these cost models, because these statistics are used to derive the value of coefficients in cost formulas. It is often DBMS implementers who are able to give accurate cost formulas with indispensable sources statistics.

When the data sources are autonomous, cost formulas and source statistics are unavailable. For obtaining cost models we need some special methods that vary with the autonomy degree of data sources. For example, the method by *Calibration* [10] estimates the coefficients of a generic cost model for each type of relational data sources. This calibration needs to know access methods used by the source. This method is extended to object-oriented databases by [14]. If this calibration procedure can not be processed due to data source constraints, a sampling method proposed in [33] can derive a cost model for each type of query. The query classification in [33] is based on a set of common rules adopted by many DBMSs. When no implementation algorithm and cost information are available, we can use the method described in [2], in which cost estimation of new queries is based on the history of queries evaluated so far.

Specially, for semi-structured data sources, [1] proposed two techniques for estimating the selectivity of simple XML path expressions over complex large-scale XML data as would be handled by Internet-scale XML applications. [20] describes the cost-based query optimizer in a DBMS for XML-based data supporting an expressive query language. [31] proposed an approach to use a new statistical learning technique called "transform regression" instead of detailed analytical models to predict the overall cost for XML-based data sources.

2.2 Mediation System with Cost-Based Query Optimization

[21] proposed a mediation system called DISCO, based on global-as-view approach. The mediator DISCO distinguishes operations executed at data sources (wrappers) level and operations executed on the mediator level. The data model of DISCO is based on ODMG standard. Disco uses a cost-based optimization approach that combines a generic cost model with specific cost information exported by wrappers. The data source interface is specified using a subset of CORBA IDL extended with a cardinality section for data sources statistics and a cost formula sections for specific formulas. This subset defines a cost communication language suitable for object-oriented environment but is not generic.

Other mediation systems with cost-based query optimization include Garlic [17], Hermes [2], Ariadne [3], etc. However, none of the solutions mentioned above has addressed the problem of overall cost estimation in a *semi-structured* environment integrating heterogeneous data sources.

2.3 Search Strategy

An important aspect in cost-based query optimization is how to explore the set of alternative execution plans that constitute the *search space*. Generally, the mediator uses transformations rules for operations to generate new plans from an original one. As the number of rules is huge, this implies an exponential blow-up of the candidate plans. It is impossible to calculate the cost of all these candidate plans, because the cost computation and the subsequent comparisons will be even more costly than the execution of the plan. An efficient search strategy is necessary to reduce the size of the search space containing candidate execution plans. First of all, [18] describes 3 important rules used by most DBMSs for plan generation: (1) Transformation based on join order; (2) Avoid cross products join; (3) Avoid taking intermediate result as inner operand of each join. Furthermore, commercial systems use essentially *Dynamic Programming Algorithm* [24] for generate candidate plans. This algorithm is a dynamically pruning, exhaustive search algorithm, which can construct all alternative join trees (that satisfy the 3 rules) by iterating on the number of relations joined so far and delete trees that are known to be suboptimal.

3. BACKGROUND AND MOTIVATION

In this section we first introduce the XQuery processing in XLive mediation system with its TGV modeling. Then we give the objective of the PhD Work in this context.

3.1 XQuery processing in XLive

A user's XQuery submitted to the XLive mediator is first transformed into a canonical form. Then the canonized XQuery is modeled in an internal structure called TGV. We annotate the TGV with information on evaluation, such as the data source locations, sources functional capabilities of sources, etc. This *annotated* TGV is then transformed into an execution plan using a physical algebra. To this end, we have chosen the XAlgebra [6] that is an extension to XML of the relational algebra. Finally, the physical execution plan is evaluated and an XML result is produced, Fig.2 depicts the different steps of this processing.

3.2 Tree Graph View

TGV is a logical structure model implemented in the XLive mediator for XQuery processing, which can be manipulated, optimized and evaluated [27]. TGV takes into account the whole functionality of XQuery (collection, XPath, predicate, aggregate, conditional part, etc.) and uses an intuitive representation that provides a global view of the request in a mediation context. Each element in the TGV model has been defined formally using Abstract Data Type in [26] and has a graphical representation. In Fig. 3 (a), we give an example of XQuery which declares two FOR clauses ($a and

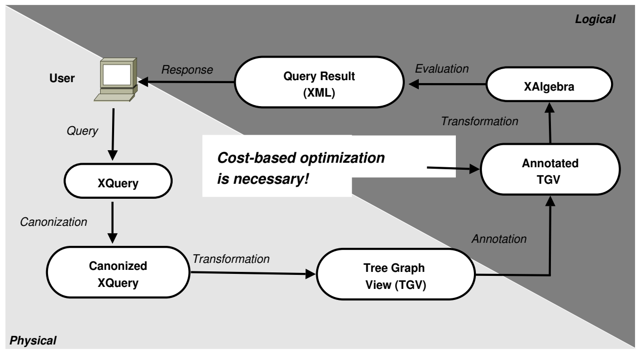
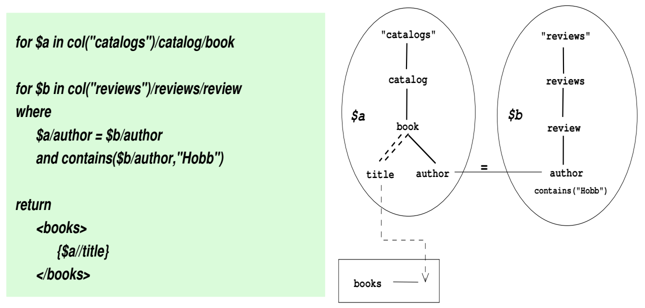


Figure 2: XQuery processing in the XLive mediation system

$b), a join constraint between authors and a contains function, then a return clause projects the title value of the first variable. This query is represented by a TGV in Fig. 3 (b). We can distinguish the two domain variables $a and $b of the XQuery, defining each nodes corresponding to the given XPaths. A join hyperlink links the two *author* nodes with an equality annotation. The *contains* function is linked to the $b "author" node, and a projection hyperlink links the node *title* to the ReturnTreePattern in projection purposes.

The subsets of elements of a TGV model can be annotated with various information. Precisely, for each arbitrary component (i.e. one or a group of operators of TGV), we add some additional information such as system performance information, source localization, etc. Our annotation model is generic and allows annotation of any type of information. The set of annotation based on the same annotation type is called an *annotated view*. There can be several annotated views for the same TGV, for example, algorithm annotated view, sources-localization annotated view, etc.



```
for $a in col("catalogs")/catalog/book

for $b in col("reviews")/reviews/review
where
    $a/author = $b/author
    and contains($b/author,"Hobb")

return
    <books>
        {$a//title}
    </books>
```

(a) An XQuery query

(b) TGV representation

Figure 3: An example of XQuery and its TGV representation

3.3 Objective of PhD Work

In Xlive, for a given XQuery, the mediator can generate different TGVs (execution plan) to execute it, and these plans differ widely in execution cost. Thus we need an cost-based optimizer to choose the best TGV to execute the XQuery. This optimizer should involve a generic cost framework that integrates different cost information at the mediator level and deals with the cost communication that needs to be done between different components of XLive on a standard XML-based format. Due to the lack of accurate cost models for autonomous XML data source (ref. Fig. 1), a specific method for constructing a generic cost model for native XML database is needed. Finally, we should prove that the generic cost framework can be combined with various search strategies and provide an optimization with high performance.

4. PROPOSED SOLUTION

In this section, first we describe our cost modeling for XQuery processing in XLive, with a generic language for cost communication, then we introduce TGV cost annotation and the procedure for the overall cost estimation at the mediator level.

4.1 XLive Mediator Cost Model Description

4.1.1 Generic cost model

The objective of this cost modeling is to integrate different types of cost information within XLive system. A cost model is generally designed for some type of data source (but there are also some methods that can be used for different types of sources, for example, the method by history [2]). It can contain some accurate cost formulas with coefficients' value derived from data sources statistics, or a specific method for deriving the cost formulas. A cost model may also have only a constant value for giving directly the execution cost of operators. For the cost model in XLive, we apply the principle *as accurate as possible*. For example, the method by *calibration* can normally provide more accurate cost models than the method based on *historical costs*, but it has a lower accuracy level than cost models based on operation implementation. That means if the cost models based on operation implementation are available, we use neither the method by *calibration* nor *history*. For each operation defined by TGV specification, we provide a cost model to estimate its execution cost.

4.1.2 Generic Cost Model for XML Data Source

In the Fig. 1, we notice that there are no specific methods that can provide a generic cost model for XML data source, in case the accurate operation-level cost model is unavailable for the mediator. XLive integrates some native XML databases such as eXist [11] and need estimate the cost of operations executed on these data sources. We propose a calibration method inspired from [10] to derive a generic cost model for these XML data sources.

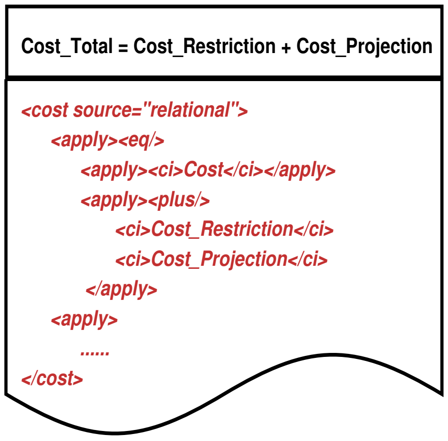
XML query operators are quite different from relational and object query operators because they generally include tree navigation. We first identify the operators implemented in native XML DBMS with their implementation algorithm. For example, for eXist database, typical operations include *Decision, Reconstruction, Node ID Comparison, Structural Join*, etc. For each operation, a generic cost formula is created with n unknown coefficients. Then we execute n times the operation and obtain the corresponding execution cost. All of this forms an equation system with n equation and n unknown coefficients. By solving this equation system, we can derive the value of the unknown coefficients and use it to the future cost estimation. Generally, the coefficients to calibrate can be the average CPU Time, I/O time and other overhead involved in query processing.

4.2 Generic Language for Cost Communication (GLCC)

To perform cost communication within our XLive system, we define a language to express the cost information in a uniform, complete and generic manner. This language fits to our XML environment, to avoid costly format converting. It considers every cost model type and allows wrappers to export their specific cost information. In our XLive context, this language is generic enough to express cost information of different parts of a TGV and is capable to express cost for various optimization goals, for example, response time, price, energy consummation, etc.

Our language extends the MathML language [28], which allows us to define all mathematical functions in XML form. MathML fits to cost communication within XLive due to its semi-structured nature. We use the *Content Markup* in MathML to provide explicit encoding for cost formulas. We just add some rules to MathML to define the grammar of our language. Furthermore, this grammar is extensible so that users can always define its own tags for any type of cost.

Cost formulas are represented in the form of equations set. Each equation corresponds to a cost function that may be defined by the source or by the mediator. Each component of TGV is annotated with an equation set in which the number of equations is undefined. One function in a set may use variables defined in other sets. We define some rules to ensure the consistency of the equations system. First, every variable should have somewhere a definition. Second, by reason of generality, there are no predefined variable names. For example, in the grammar, we do not define a name "time" for a cost variable because the cost metric can be a price unit. It is the user of the language who gives the specific significant names to variables. This gives a much more generic cost definition model compared to the language defined in [21]. Fig. 4 gives a simple example for the expression of a cost formula.


Cost_Total = Cost_Restriction + Cost_Projection

```
<cost source="relational">
    <apply><eq/>
        <apply><ci>Cost</ci></apply>
        <apply><plus/>
            <ci>Cost_Restriction</ci>
            <ci>Cost_Projection</ci>
         </apply>
    <apply>
        ......
</cost>
```

Figure 4: An example of cost formula and its representation on GLCC

4.3 Dynamic Cost Estimation Framework

Fig. 5 shows the role of our language in cost communication. After extracting cost information from data source, the wrapper exports that information using our language to the parser, which derives cost models that will be stored in the wrapper information repository. When the mediator needs to compute the execution cost of a plan (TGV), the wrapper information repository provides necessary cost information for operators executed on wrappers. We have a cache for storing historical execution cost of queries evaluated, which can be used to adjust the exported cost information from the wrapper. All these communications are processed in the form of our language. Our language completes the interface between different components of XLive.

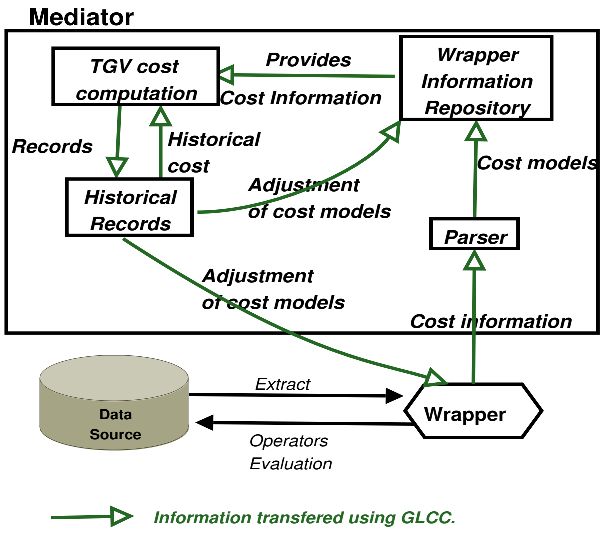


Figure 5: Dynamic cost estimation framework

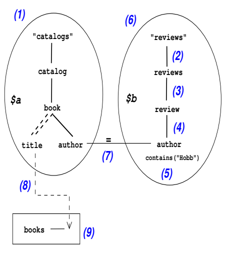
4.4 Overall Cost Estimation

4.4.1 TGV Cost Annotation

As mentioned in Section 3, the execution plan of XQuery within the query processing in XLive. The purpose of our query optimization is to find the *optimal* TGV with the least execution cost. For estimating the overall cost of a TGV, we annotate different components (one or a group of operators) of TGV. For an operator or a group of operators appearing in a TGV, the following cost information can be annotated:

- Localization: The operator(s) can be executed on the mediator or on the wrappers (data sources).
- Cost Model: Used to calculate the execution cost of the component.
- Other information: Contains supplementary information that is useful for cost estimation. For example, several operators' (such as join operator) implementation allows parallel execution between its related operators.

Fig. 6 gives an example for TGV cost annotation. In this example, different components of the TGV introduced in Fig. 3 (Ref. Section 3) are annotated. We can see for the operators executed on Source1 (S1), we have only the historical cost to use for estimate the total execution cost of all the these operators; in contrast, for each operator executed



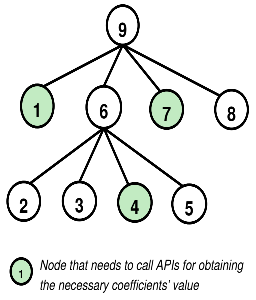
N°	Localization	Cost Model			Other information
		Type	Method	Cost formulas	
(1)	Wrapper	Relational	History	Cost_s1 = return value of API	Dynamic
(2)	Wrapper	XML		Cost_link_review = Cost_IO*card(reviews) Cost_IO = 0.05 reviews = 10000	
(3)	Wrapper	XML		Cost_link_review = Cost_IO*card(catalogs) card(catalogs) = 6000	
(4)	Wrapper	XML	Calibration	Cost_link_author = 2*card(catalogs)*sel sel = 0.2	
(5)	Wrapper	XML		Cost_restr_author = Cost_IO*sel	
(6)	Wrapper	XML		Cost_s2 = (2) + (3) + (4) + (5)	
(7)	Mediator	XML	Calibration	Cost_join = Cost_CPU*(card_s1*card_s2)	Parallel execution enable
(8)	Mediator	XML		Cost_proj = result_card * 0.02	
(9)	Mediator	XML		Cost_global = Max((1), (6)) + (7) + 2*(8)	

Figure 6: An example for TGV cost annotation

on Source2 (S2), we have a cost model for estimating its execution cost. For the join operator (numbered (7)) executed on the mediator, the operators linked to it can be executed in parallel.

4.4.2 *Cost Annotation Tree (CAT)*

We have seen how to annotate a TGV with cost information. Now we are concentrated on how to use this cost annotation for the overall cost estimation of a TGV. As illustrated in Fig. 6, the cost of an annotated component of TGV generally depends on the cost of other components. For example, for the cost formula annotated in (6), we see that it depends on the cost of (2), (3), (4) and (5). From the cost formulas annotated for each component of TGV, we obtain a *Cost Annotation Tree (CAT)*. In a CAT, each node represents a component of TGV annotated by cost information and this CAT describes the hierarchical relations between these different components. Fig. 7 (a) illustrates the CAT of the TGV annotated in Fig. 6.


9
1
6
7
8
2
3
4
5
1
Node that needs to call APIs for obtaining the necessary coefficients' value

(a) Cost Annotation Tree (CAT)

```
1  associateCost (node) {
2      node.analyzeCostModel ( );
3      if (node.hasSpecialMethod( )) {
4          node.callAPI( );
5      }
6      for (each child of node) {
7          associateCost(child);
8      }
9      node.configCostFormula( );
10     node.calculateCost( );
11 }
```

(b) Overall cost estimation algorithm

Figure 7: Overall cost estimation

4.4.3 *Overall cost estimation algorithm*

We now show how to use the CAT of a TGV to perform the overall cost estimation. We use the recursive breadth-first search algorithm of a tree for performing cost estimation of each node. For each node of CAT, we define a procedure called *associateCost* (Fig. 7 (b)) for operating the cost annotation of a node. This procedure first analyzes the cost annotation of the node and derives its cost model (line 2); If a specific cost method is found, it calls an API implemented by XLive for obtaining the necessary values of coefficients or cost formulas for computing the cost (line 3-5); if the cost of this node depends on the cost of its child nodes, it executes recursively the *associateCost* procedure on its child nodes (line 6-8). When these 3 steps are terminated, a procedure *configCostFormula* completes the cost formulas with obtained values of coefficients (line 9) and execution cost of this node will be calculated (line 10). By using this algorithm, we can obtain the overall cost of a TGV, which is the cost of the root of CAT.

5. SUMMARY AND FUTURE WORK

5.1 Summary: Compared to Related Work

In this paper, we described our cost framework for the overall cost estimation of candidate execution plans in an XML-based mediation system. The closest related work is DISCO system [21], which defines a generic cost model for an object-based mediation system. Compared to DISCO work and other mediation systems, we have the following contributions: First, to our knowledge, our cost framework is the first approach proposed for addressing the costing problem in XML-based mediation systems. Second, our cost communication language is completely generic to express any type of cost, which is an improvement compared to the language proposed in DISCO. Third, our cost framework is generic enough to fit to overall cost computation within various mediation systems.

5.2 Future Work Discussion

5.2.1 *Calibrating Cost of Native XML Data Sources*

As mentioned in Section 4.1.2, for autonomous native XML DBMS, we need to develop a calibration method to derive a generic cost model. We plan to take into account the operations that constitute execution plan in such DBMSs and construct the generic cost formulas according to the implemented algorithm of these operations. Thus, a proper classification of operations by database need to be done. And we need to prove that the proposed method can be used to accurately calibrate typical native XML databases.

5.2.2 *Search Strategy*

It has been shown in [27] that for processing a given XQuery, a number of candidate plans (i.e. TGV) can be generated using transformation rules that operate on TGVs. These rules have been defined for modifying the TGV without changing the result. The execution cost of a TGV can be computed by using our generic cost framework and thus we can compare the costs of these plans to choose the best one to execute the query. However, as the number of rules is huge, a search strategy is necessary to limit the number of candidate plans (ref. 2.3). We note in this respect that our cost framework should be proved generic enough to be applied to various search strategies such as exhaustive, iterative, simulated annealing, genetic, etc.

5.2.3 *Cost Framework Validation and Optimizer Performance*

We plan to validate our proposed cost framework by implementing the specification mentioned in Section 4 within XLive system. We first need to demonstrate the accuracy of the cost models, which means that the measured execution cost should correspond to the expected cost predicted by the cost framework; second, the cost framework needs to be proved efficient with various search strategies, by testing the optimization performance of the system, for example, the average reduced execution time for the queries.

Acknowledgment

This work is supported by *Xcalia S.A.* (France) and by ANR *PADAWAN* project.

6. REFERENCES

[1] A. Aboulnaga, A. Alameldeen, and J. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. *VLDB*, 2001.

[2] S. Adali, K. Candan, and Y. Papakonstantinou. Query Caching and Optimization in Distributed Mediator Systems. In *ACM SIGMOD*, 1996.

[3] J. Ambite and C. Knoblock. Flexible and scalable query planning in distributed and heterogeneous environments. In *the Fourth International Conference on Artificial Intelligence Planning Systems*, 1998.

[4] J. A. Blakeley, W. McKenna, and G. Graefe. Experiences Building the Open OODB Query Optimizer. In *ACM SIGMOD*, 1993.

[5] S. Cluet and C. Delobel. A General Framework for the Optimization of Object-Oriented Queries. In *ACM SIGMOD*, 1992.

[6] T. Dang-Ngoc and G. Gardarin. Federating Heterogeneous Data Sources with XML. In *Proc. of IASTED IKS Conf.*, 2003.

[7] T. Dang-Ngoc, G. Gardarin, and N. Travers. Tree Graph View: On Efficient Evaluation of XQuery in an XML Mediator. In *BDA*, 2004.

[8] T. Dang-Ngoc, C. Jamard., and N. Travers. XLive: An XML Light Integration Virtual Engine. In *Bases de Données Avancées (BDA)*, 2005.

[9] A. Dogac, C. Ozkan, B. Arpinar, T. Okay, and C. Evrendilek. *Advances in Object-Oriented Database Systems*. Springer-Verlag, 1994.

[10] W. Du, R. Krishnamurthy, and M. Shan. Query Optimization in a Heterogeneous DBMS. In *VLDB*, 1992.

[11] eXist. Open Source Native XML Database, 2007. http://exist.sourceforge.net/.

[12] D. Florescu. *Espace de Recherche pour l'Optimisation de Requêtes Objet*. PhD thesis, University of Paris IV, 1996.

[13] G. Gardarin, J. Gruser, and Z. Tang. Cost-based Selection of Path Expression Algorithms in Object-Oriented Databases. In *VLDB*, 1996.

[14] G. Gardarin, F. Sha, and Z. Tang. Calibrating the Query Optimizer Cost Model of IRO-DB. In *VLDB*, 1996.

[15] D. Gardy and C. Puech. On the Effects of Join Operations on Relation Sizes. *ACM Transactions on Database Systems (TODS)*, 1989.

[16] J. Gruser. *Modèle de Coût pour l'Optimisation de Requêtes Objet*. PhD thesis, University of Paris IV, 1996.

[17] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimization Queries Across Diverse Data Sources. In *VLDB*, 1997.

[18] Y. Ioannidis. Query Optimization. *ACM Computer surveys*, 28(1), Mar. 1996.

[19] L. F. Mackert and G. M. Lohman. R* Optimizer Validation and Performance Evaluation for Local Queries. In *ACM SIGMOD*, 1986.

[20] J. McHugh and J. Widom. Query Optimization for Semistructured Data. Technical report, Stanford University Database Group, 1999.

[21] H. Naacke, G. Gardarin, and A. Tomasic. Leveraging Mediator Cost Models with Heterogeneous Data Sources. In *ICDE*, 1998.

[22] M. Roth, F. Ozcan, and L. Haas. Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System. In *VLDB*, 1999.

[23] M. T. Roth and P. Schwarz. Don't Scrap It, Wrap it! A Wrapper Architecture for Legacy Data Sources. In *VLDB*, 1997.

[24] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. In *ACM-SIGMOD*, 1979.

[25] P. G. Selinger and M. E. Adiba. Access path selection in distributed database management systems. In *ICOD*, 1982.

[26] N. Travers. *Optimization Extensible dans un Médiateur de Données XML*. PhD thesis, University of Versailles, 2006.

[27] N. Travers, T. Dang-Ngoc, and T. Liu. TGV: An Efficient Model for XQuery Evaluation within an Interoperable System. *Int. Journal of Interoperability in Business Information Systems (IBIS)*, 3, 2006.

[28] W3C. Mathematical Markup Language (Mathml TM) Version 2.0, 2003. http://www.w3.org/TR/REC-MathML.

[29] W3C. An XML Query Language (XQuery 1.0), 2005. http://www.w3.org/TR/xquery/.

[30] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, Mar. 1992.

[31] N. Zhang, P. J. Haas, V. Josifovski, and C. Z. G. M. Lohman. Statistical Learning Techniques for Costing XML Queries. In *VLDB*, 2005.

[32] Q. Zhu. *Estimating Local Cost Parameters for Global Query Optimization in a Multidatabase System*. PhD thesis, University of Waterloo, 1995.

[33] Q. Zhu and P. Larson. Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems. *Distributed and Parallel Databases*, 1998.