A Methodology for Query Processing over Distributed XML Databases

Guilherme Figueiredo[1], Vanessa Braganholo[2], Marta Mattoso[1]

[1]Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ, Brazil

[2]Departamento de Ciência da Computação – IM/UFRJ, Brazil

{g.coelho, marta}@cos.ufrj.br, braganholo@dcc.ufrj.br

Technical Report ES-710/07[1]

Abstract. *The constant increase in the volume of data stored as native XML documents makes fragmentation techniques an important alternative to the performance issues in query processing over these data. Fragmented databases are feasible only if there is a transparent way to query the distributed database, without the need of knowing the fragmentation details and where each fragment is located. This paper presents our methodology for XQuery query processing over distributed XML databases, which consists on the steps of query decomposition, including the query's TLC algebra representation; data localization; global optimization; global query execution and final result assembly. This methodology can be used in an XML database that allows fragmentation and also in a system that publishes an integrated view of semi-autonomous and homogeneous XML databases. We propose an architecture based on a Mediator with Adaptors (wrappers) attached to remote databases. The Mediator publishes a global XML view of the distributed data, which can be queried by users in a transparent way. A Mediator and two Adapters prototypes have been implemented and experiments were executed, where we could analyze the performance improvements and impacts of different queries over distributed XML databases.*

1. Introduction

The increasing volume of stored XML data poses new chalenges to efficient query processing. Queries posed over centralized databases may take very long times to be answered, since large ammounts of data need to be accessed. In most of the cases, indexes are not enough to increase query performance.

A solution to this problem may be to distribute and fragment data accros the network. In fact, lots of systems to process queries over distributed data have been proposed [GUPTA et al. 2000, AGUILERA et al. 2002, IVES et al. 2002, SUCIU 2002, GERTZ,BREMER 2003, RE et al. 2004, SILVEIRA,HEUSER 2005]. Some others focus on query processing over heterogeneous distributed systems [BARU et al. 1999,

[1] This document presents a revised version of the Technical Report ES-710/07, previously published in 2007.

GARDARIN et al. 2002, LEE et al. 2002]. None of them, however, deal with XML data fragmentation.

In relational [ÖZSU,VALDURIEZ 1999] and object-oriented databases [BAIÃO et al. 2004], fragmentation techniques have been sucessfuly used to increase query performance in distributed databases. By fragmenting and distributing the data, queries can be sent to specific fragments, avoiding a complete scan over large portions of irrelevant data. This is the direction we take in this paper to solve the performance problem of XML queries over large repositories.

Several fragmentation techniques for XML data have been proposed in literature [BREMER,GERTZ 2003, MA,SCHEWE 2003, GERTZ,BREMER 2003, ANDRADE et al. 2006, ANDRADE 2006]. All of them focus on fragments definitions, but only Gertz and Bremmer [GERTZ,BREMER 2003] deal with query processing over the fragmented data. However, their approach is not generic – it completely depends on the index structures created by their approach. Thus, we still lack of a generic and non-intrusive methodology for distributed XML query processing.

In this paper, we adopt the fragmentation technique of Andrade et al. [2006] and define a methodology for query processing over fragmented databases. Andrade's approach was adopted due to several reasons. First, it uses an XML algebra to represent the fragments. Thus we can use the algebra properties to process queries over the fragments. Second, it has experimental results that show that fragmentation can also be applied successfully to increase the performance of XML queries.

To be able to define a generic and automatic approach to process distributed XML queries, we need some formalism. When fragments and queries can be represented in an algebraic form, the properties of the algebra, together with the rewrite rules for fragments [ANDRADE et al. 2005, ANDRADE et al. 2006] can be used to replace references to the centralized database by references to its fragments in a given query. The algebraic properties allow us to formally prove that this query rewriting is correct. Thus, to process a query over a fragmented repository, the first step is to rewrite it so it references the fragments instead of the centralized (virtual) database.

However, an algebra is not enough. We need also to define a methodology with the steps needed to automatically execute distributed XQuery queries. This methodology is the main contribution of this report.

Before going further, it is important to notice that our approach to improve the performance of queries over large repositories can be applied not only in fragmented databases, but also in semi-autonomous databases. In some real-world cases, semi-autonomous databases can be considered a fragment of a global (virtual) database, as we explain below.

The dynamism of the real world makes companies to grow faster than its technological infrastructure. This causes lots of decentralized databases that store information about the company to appear. As an example, consider a bookstore that has several branches. Each of them may have a local (semi-autonomous) database to store local orders, among other information. However, when the company directors need to have a global view of the company (ex: the total amount of sales of each branch library in a given month), (s)he must send individual queries to each of the databases, and then collect the results. A possible alternative to the problem would be to replicate each local database in a centralized server, and then pose queries to this server. However, when up to date results are needed, this may not be the best choice. The solution we propose to this problem is to look at these local databases as horizontal fragments of a global company database. In this way, our approach could be used to process a query over the global (virtual) database and distribute the query among each local branch library database. This would be completely transparent to the company directors, and up to date results are always guaranteed. From now on, we call this global (virtual) database *global view.* Similarly, the local databases could be seen as *local views.* In fact, they may be real views in some cases as we will see later on. In such scenarios, both the global view and the local views can be considered XML views [ABITEBOUL 1999]. Thus, XQuery can be used to query them.

The solution above requires the XML views of the branch libraries to be completely homogeneous, since integration of heterogeneous data sources is out of the scope of our work. However, data appearing in such views may be stored in different ways, using different data models. The only requirement is that the XML view obtained from this data be homogeneous. The XML views can be automatically obtained and maintained by using some of the existing approaches (SHANMUGASUNDARAM et al., 2000; FERNÁNDEZ et al., 2002; BRAGANHOLO et al., 2004; CAREY et al., 2000), depending on the data model in which the source data is stored.

This report is structured as follows. Section 2 presents an overview of distributed query processing and existing work on this area. The methodology we propose to process distributed queries is presented in Section 3. Section 4 shows details of the implementation of our methodology and a prototype of a Mediator to process distributed queries. Section 5 contains an experimental evaluation. Finally, we conclude on Section 6.

2. Querying Distributed Databases

Query processing over distributed and fragmented databases is more challenging than doing so in a centralized environment. In a distributed environment, the DBMS needs to know where each node is located, as well as parameters such as communication costs and current load of each node, to be used by the query optimizer. Fragmentation further adds complexity related to reducing the query so it can be executed only in nodes that have relevant data to that query answer. In [ÖZSU,VALDURIEZ 1999], we can find a very good reference about distributed databases, and also a methodology for distributed query processing in relational databases. The general ideas of this methodology can be applied to other data models [BAIÃO et al. 2000, BAIÃO et al. 2004]. More advanced topics on distributed query processing, such as optimization techniques, execution techniques, dynamic replication in the distributed environment, caching, architectures, etc., can be found in [KOSSMAN 2000].

In general, to process a distributed query we need to transform a high-level query over the global (centralized) view of the distributed data into one or more sub-queries of lower level (to be executed over the nodes in the distributed environment). We now present a summary of fragmentation and query processing techniques in the relational model, object-oriented model and semi-structured model.

Ozsü and Valduriez [1999] present a methodology to process distributed queries over the relational model. The methodology consists in several layers: query decomposition; data localization; global optimization; and local optimization, as shown in Figure 1.

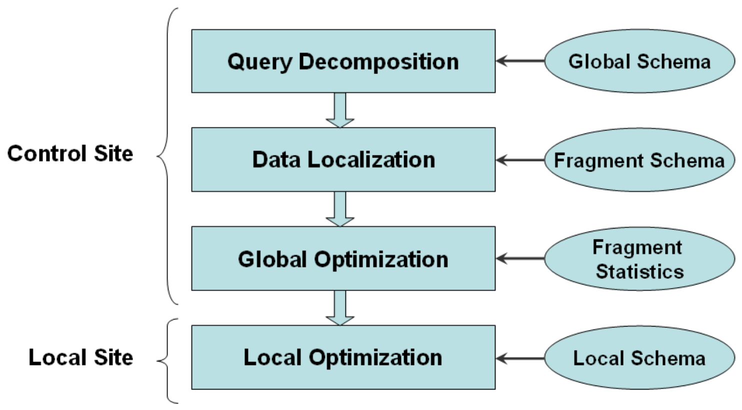


Figure 1. Generic Layering Scheme for distributed query processing [ÖZSU,VALDURIEZ 1999]

The first layer decomposes the query in an algebraic query over global relations. In this process, syntactic and semantic analyses are performed over the submitted query. The query is simplified and rewritten in an algebraic form. Information about data distribution are not used in this layer.

The data localization layer has as goal to locate query data by using information about data distribution (fragmentation and allocation of fragments among the nodes). This layer determines the fragments that are involved in the query, and replaces the references to the global view by references to fragments in the algebraic query. This replacement can be performed automatically when the fragmentation design follows correction rules [ÖZSU,VALDURIEZ 1999] such as the reconstruction rule (the global database can be reconstructed from its fragments).

The global optimization layer tries to find a near to optimal strategy (or plan) to execute the global query. The optimization is performed, in general, by minimizing a cost function. The cost function is usually a combination of CPU, communication and I/O costs. To databases distributed in slow networks, the communication cost must be the most important factor in the cost function.

After global optimization of the global query, sub-queries are generated and sent to the remote sites. Each sub-query executed in a given site is further optimized using the local site schema in the local optimization layer. The results of the remote sub-queries are processed by the DBMS according to the adopted strategy of final result composition.

Further information about the layering scheme for distributed query processing can be found in [ÖZSU,VALDURIEZ 1999].

2.1. Distributed Queries on the Semi-structured Model

The semi-structured data model was defined for data that cannot be represented by a strict schema [BUNEMAN 1997, ABITEBOUL et al. 1999]. This model allows data to be partially structured. This means that components may be missing in some data items, components may have different types in different items, and data collections can be heterogeneous. In the semi-structured data model, a database is modeled as a labeled rooted graph. In this graph, nodes represent objects and have an associated identifier (*oid*). XML [W3C 2006] is essentially a syntax to represent semi-structured data.

An XML repository may be of two distinct types: Single Document (SD) or Multiple Documents (MD) [YAO et al. 2002]. In SD repositories, all of the information is stored in a single XML document (with an associated DTD or XML Schema). MD repositories, on the other hand, also require a schema definition. However, the repository is composed of multiple instances (XML documents) of this schema, constituting what is called a *collection* of XML documents.

The hierarchical and semi-structured nature of XML poses some difficulties in dealing with distributed queries. Some work in literature [SUCIU 2002, GERTZ,BREMER 2003, RE et al. 2004, SILVEIRA,HEUSER 2005] deal with query processing over distributed XML models.

One of the first work on distributed query processing over the semi-structured model is presented in [SUCIU 2002]. The author analyzes the problem for two kinds of queries over XML repositories: path expressions and *select-where* queries. Suciu analyzes the efficient evaluation of distributed queries by minimizing the communication times and also the data volume transferred between remote nodes. The approach, however, does not support join operations between remote bases, and also do not deal with fragmented repositories.

In [GERTZ,BREMER 2003], Gertz and Bremmer present a complete solution to distributed XML query processing. However, their solution is based on join algorithms that need to be implemented in the XML repositories. Additionally, they do not present a methodology for query processing. Their algorithms focus on using their index structures to efficiently process distributed queries. Our approach is more generic in the sense that it uses an algebraic representation for both fragment definition and query decomposition. In addition we present a non intrusive architecture design. The index structures of [GERTZ,BREMER 2003] could be used, complementarily, to improve the performance in our approach. To be best of our knowledge, there is no work in literature that proposes a methodology and execution model to process queries over a distributed XML database. We also contribute by showing how it can be used on top of pre-existing native XML DBMS applications.

Re et al. [2004] propose an extension to XQuery to allow sub-queries over remote sites to be declared directly in the extended XQuery syntax. The goal of this approach is to remotely pre-select the document that will be used in the query, avoiding the need of transferring the entire document to the server that executes the query. A disadvantage of this approach is that the user needs to know the structure of the remote sites, which reduces its applicability in practice.

In literature, we can also find work on integration of heterogeneous distributed databases using XML as a common standard between the heterogeneous data sources, and a mediator [BARU et al. 1999, GARDARIN et al. 2002, LEE et al. 2002]. Such works do not deal with fragmented databases, since their focus is on dealing with the heterogeneous schemas attached to the Mediator. The remote databases are accessed through adapters that provide an XML view of the data and support queries over these views.

2.2. Fragmentation Techniques in Native XML Databases

To work with distributed XML queries, we need a formal definition of XML database fragmentation. This definition must allow us to reconstruct the global collection from its fragments (by using reconstruction rules). These rules are needed to decompose the distributed query. Several fragmentation techniques for XML repositories have been proposed in literature [BREMER,GERTZ 2003, MA,SCHEWE 2003, ANDRADE et al. 2006].

Ma et al [2003] proposes three types of fragmentation: horizontal, which groups elements of an XML document according to a selection predicate; vertical, whici restructures the XML document; and *split*, which breaks an XML document in a set of new XML documents. These fragmentation definitions are not accompanied by an XML algebra that supports the correct reconstruction of the original document. This prevents the use of this technique to automatically decompose algebraic queries over distributed XML documents and the consequent result composition.

In [BREMER,GERTZ 2003], the authors propose a new approach to the distribution design of an XML database. This design comprehends both fragmentation and allocation of fragments. However, the approach considers only SD repositories. Additionally, there is no formal distinction between horizontal and vertical fragments, which are combined in a hybrid type of fragment. The fragments are defined in a language derived from XPath, and their definitions are stored in a metadata repository. This metadata repository is used in the distributed query processing, and is applied in all of the nodes of the distributed environment.

In [ANDRADE et al. 2005, ANDRADE et al. 2006], the authors formally define horizontal, vertical and hybrid fragmentation. Fragments are defined through TLC [PAPARIZOS et al. 2004] algebra operations, which allows queries to be decomposed over the fragments when one uses TLC to represent XQuery queries. A horizontal fragment is created by using a selection operation over the original XML document(s); a vertical fragment uses a projection operation over the original document(s). A hybrid fragment uses both selection and projection operations to create the fragment. Besides the formal aspect of integrating an algebra to XQuery queries, the definitions of Andrade et al. can be applied to both SD and MD repositories. Another interesting aspect of this work is that the fragmentation definitions are very similar to the ones proposed to the relational model [ÖZSU,VALDURIEZ 1999]. Thus, adapting the well succeeded techniques of the relational model to the XML model becomes an attractive and promising option. Furthermore, the authors present correction rules for each fragmentation type, which is essential to query decomposition.

Using the analysis of the XML fragmentation techniques we have made, we have decided to adopt the technique defined in [ANDRADE et al. 2006]. This choice was made for several reasons. First, it comprises both SD and MD collections. Second, it has formal correction and reconstruction rules using the TCL algebra, which is essential to distributed query processing.

2.3. XML Query Algebras

Work done in both relational and object-oriented models shows us that using an algebra is essential to query processing and optimization. XML documents have a very flexible structure. This allows semi-structured documents, but on the other hand, makes the creation of an XML algebra more complex. Despite there is no standard XML algebra, there are several work on literature on this subject [FERNÁNDEZ et al. 2000, JAGADISH et al. 2001, FRASINCAR et al. 2002, ZHANG et al. 2002, CHEN et al. 2003, PAPARIZOS et al. 2004].

The Tree Logical Class (TLC) algebra [PAPARIZOS et al. 2004], also implemented in the native database Timber [JAGADISH et al. 2002], is an evolution of TAX [JAGADISH et al. 2001]. It allows access to heterogeneous sets of trees through annotated pattern trees (APTs). APTs extend the concept of pattern tree by allowing its use with heterogeneous sets. Also, the concept of Logical Classes is used to label nodes of the output trees according to the pattern tree.

Besides the concepts of annotated pattern trees and logical classes mentioned above, TLC defines the following algebraic operations: filter, join, selection, projection, elimination of duplicates, aggregation functions and construction.

3. A methodology for Distributed Query Processing of XQuery queries

This section presents our methodology to process XQuery queries over distributed XML databases. It involves the decomposition of the main query into sub-queries that will be executed in the remote sites containing fragments of the global collections. Over the main query, we apply the layers of decomposition, localization, global optimization, creation of sub-queries and their execution on the XML remote databases.

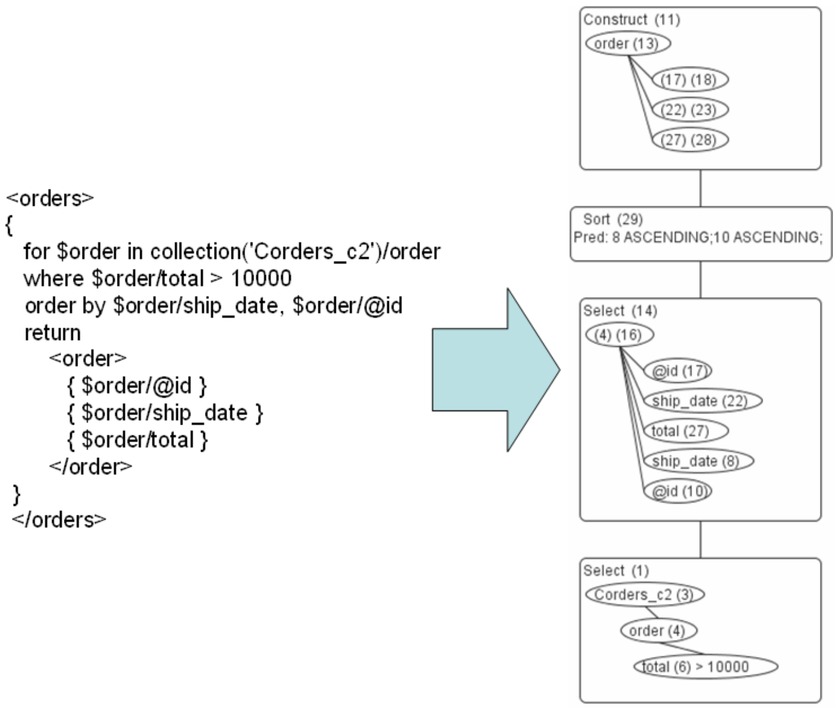
This methodology can be applied in a database that allows fragmentation as well as in a system that provides an integrated view of homogeneous semi-autonomous databases (since it does not support heterogeneous schemas). In both cases, a Catalog stores information about the distributed database (global schemas, fragments schema, fragments definitions, information about fragments allocation, as well as statistics about the fragments and remote nodes).

Our methodology is an adaptation of the four generic layers proposed by [ÖZSU,VALDURIEZ 1999] illustrated in Figure 1: query decomposition; data localization; global optimization; and local optimization. We describe each of these layers in the context of distributed XQuery queries in the next sections.

3.1. Query Decomposition

The first layer consists on decomposing the XQuery query into a TLC algebraic expression over the global collections. In this layer, we use the catalog of global collections to validate the collections referenced in the query. However, information about data distribution is not used in this phase. This layer is very similar to XQuery processing over centralized environments. It has four main steps: syntactic validation; semantic validation; query simplification; and query translation to an algebra expression.

The final product of the query decomposition layer is the algebraic representation of the query over the global collections. To do this, we use an algorithm that syntactically analyses the query and generates the algebraic expressions, the patterns trees and the TLC logical classes. We show an example in Figure 2.



```
<orders>
{
  for $order in collection('Corders_c2')/order
  where $order/total > 10000
  order by $order/ship_date, $order/@id
  return
     <order>
       { $order/@id }
       { $order/ship_date }
       { $order/total }
     </order>
}
</orders>
```

Figure 2. XQuery query and its TLC representation

3.2. Data Localization

The data localization layer has two main steps: replacing references to the global collections in the algebraic query plan by references to local fragments; and reducing these fragments according to selection and projection predicates of the original query, in a similar way of what is done in [ÖZSU,VALDURIEZ 1999]. This layer is the main responsible for benefits in query performance, since a query may have a better performance when irrelevant fragments can be discarded (by reducing the queried data volume).

To replace references to global collections in the algebraic query, we need to know the fragments and their predicates: selection, in the case of horizontal fragments; projection, in the case of vertical fragments; or both, in the case of hybrid fragments. This information is store in the catalog of the distributed database.

With this knowledge in hands, we now need to know how to make the replacements. When the fragments follow the correction rules of [ANDRADE et al. 2006], we can use the reconstruction property to guarantee that there will be a TLC operation that is capable of reconstructing the global collection from its fragments. This operation will be defined according to the fragmentation type: union for horizontal fragments; join for vertical fragments (and both for hybrid fragments). In the next sections we present localization rules for the different types of fragmentation.

3.2.1. Horizontal Fragmentation

Horizontal Fragmentation implies, by definition, that all fragments of the global collection are defined only by using selection predicates. In this way, the reconstruction can be made through the TLC union operation over these fragments [ANDRADE et al. 2006].

Definition 1: Let *GC* be a global collection and HF_1, HF_2, ..., HF_n its horizontal fragments. According to [ANDRADE et al. 2006], $GC = HF_1 \cup HF_2 \cup ... \cup HF_n$.

Figure 3 shows an example of localization of a TLC selection operation over a global collection composed by three horizontal fragments.

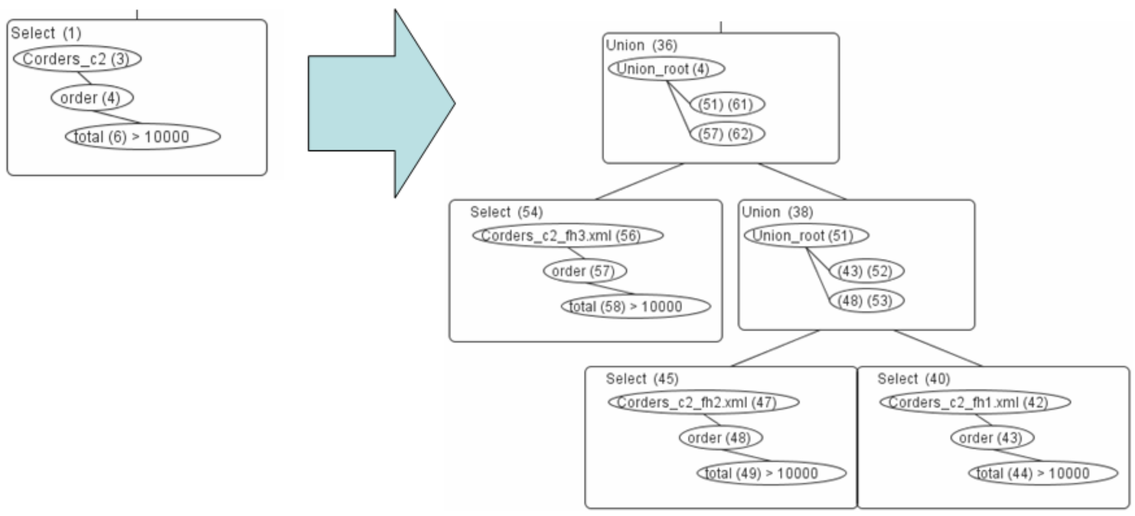


Figure 3. Localization of a selection operation over a global collection horizontally fragmented

After replacing the global collection by its reconstruction expression (that is, by the union of its fragments), we pass to the next step: reducing irrelevant fragments. In the case of horizontal fragments, which have selection predicates, the reduction consists in an analysis of the global selection operation (that is now being applied to the fragments) to identify the ones that contradict the selection predicates of the fragments. These fragments can be eliminated from the query. Formally, the elimination of fragments can be defined by the following rule:

Definition 2: Let *GC* be a global collection and $HF_1, HF_2, ..., HF_n$ its horizontal fragments. Let *pi* be the selection predicate that defines fragment HF_i. Let *pq* be the selection predicate of a query *q* over the global collection *GC*. Let *A* be the algebraic tree representing query *q* over *GC*. A fragment HF_i can be eliminated from *A* if $\sigma pq(HF_i) = \varnothing$. This selection is empty if the following rule holds: $\sigma pq\ (HF_j) = \varnothing$ if $\forall s$ in *GC*: $\neg(pq(s) \wedge pi(s))$, where *p(s)* denotes that the sub-tree *s* satisfies the predicate *p*.

Figure 4 shows the result of the reduction step of a query over a collection that is horizontally fragmented. The three fragments are formed by selection criteria over an attribute of the collection that is being queried. Since the query uses this same attribute, we can verify the compatibility among the query selection criteria and the fragments selection criteria, and then eliminate fragments that produce empty answer sets, according to the rule above.

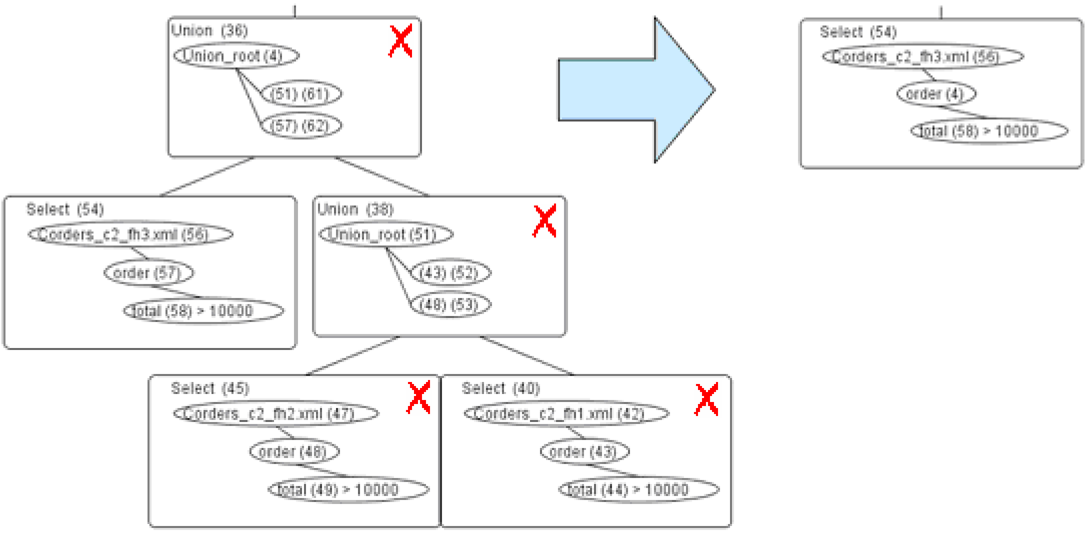


Figure 4. Reduction of irrelevant operations to the final result

The reduction of a selection operation over a fragment in an algebraic plan of a purely horizontally fragmented database also implies in removing the parent union operation, that should be replaced by its sibling (if any). Formally, if $GC = A \cup (B \cup C)$, and fragment C can be eliminated, then the resulting expression is $A \cup B$. This procedure is applied in all remove operations, thus resulting in a reduced query plan. In the same way as in the localization of a global collection, where the operation over the global collection is replaced by the union of its horizontal fragments, a special attention must be given to the LCLs of the APTs of the reduced operations, so that there is not lost reference between algebraic operations.

3.2.2. Vertical Fragmentation

In vertical fragmentation, fragments will have only projection operations. In this case, the reconstruction of the global collection can be done through joins of the fragments [ANDRADE et al. 2006].

Definition 3: Let GC be a global collection, and VF_1, VF_2, ..., VF_n its vertical fragments. According to [ANDRADE et al. 2006], $GC = VF_1 \bowtie VF_2 \bowtie \dots \bowtie VF_n$.

Vertical fragmentation requires an additional care in the location layer. Since vertical fragments do not have all elements of the global collection (they are distributed over the fragments), it is necessary to prune the APTs of the selection operations applied to fragments in order to eliminate elements that do not belong to the vertical fragment. This procedure is not necessary in cases of horizontal fragmentation, since the schemas of horizontal fragments are homogeneous.

After replacing references to the global collection by a join of its vertical fragments, we need also to reduce irrelevant fragments of the query plan. In the case of vertical fragmentation, this process consists on analyzing the sub-trees of the algebraic query tree. When we use vertical fragmentation we need to check all the algebraic operations (not only selections as in horizontal fragmentation). In case a subtree of the node representing a vertical fragment in the algebra query tree is needed in any other operation (result construction, join, order by, etc.), then this fragment can not be discarded. It can be removed otherwise. This procedure is formally defined as follows:

Definition 4: Let *GC* be a global collection and VF_1, VF_2, ..., VF_n its vertical fragments. *GC* has a set of sub-trees $A = \{A_1, A_2, ..., A_n\}$. By definition, each vertical fragment VF_i contains a projection such that $VF_i = \Pi_{A'}(GC)$, where $A' \subseteq A$. Let *Q* be an algebraic tree that represents a query *q* over *GC*, and *Q'* the result produced by the localization layer on *Q*. Let *P* be the set of sub-trees ($P \subseteq A$) used by operations in *Q'*. A fragment VF_i can be eliminated from *Q',* if *Q'* does not use as an operand any sub-tree contained in *A'*, such that $\Pi_P(VF_i) = \varnothing$. This projection is empty when the following rule holds: $\Pi_P(VF_i) = \varnothing$ if the set of sub-trees *P* has no sub-tree in *A*'.

3.2.3. Hybrid Fragmentation

Hybrid fragmentation is characterized by fragments composed of selection and/or projection operations. It must contain at least one fragment defined using both operations (selection and projection).

The reconstruction of the global collection in face of a hybrid fragmentation is done using union and joins applied over the fragments. The reconstruction rule of the global collection is constructed depending on how the operations are used in the fragments. A hybrid fragmentation can be reconstructed by a join applied over unions, or by a union applied over joins, depending on the type of hybrid fragmentation.

A hybrid fragmentation is of type *primary horizontal* when it is created by a horizontal fragmentation followed by a vertical fragmentation of these horizontal fragments. Similarly, a hybrid fragmentation is of type *primary vertical* when it is created by a vertical fragmentation followed by a horizontal fragmentation of these vertical fragments. Determining the type of hybrid fragmentation is essential to identify which reconstruction rule should be used. It can be determined as follows:

Definition 5: Let *GC* be a global collection and YF_1, YF_2, ..., YF_n its hibrid fragmetns, ordered in the sequence in which they were created. If YF_1 is defined using only a selection operation, then the hybrid fragmentation is of type *primary horizontal*. If YF_1 is defined only by a projection operation, then the fragmentation is of type *primary vertical*. Otherwise, we must read the definitions of the sibling fragment YF_2. This fragment will have a selection or projection operation that is identical to the same operation in YF_1. Is this identical operation is a selection, than the hybrid fragmentation is of type *primary horizontal*, otherwise, it is *primary vertical*.

After we identify the type of hybrid fragmentation, we can create the reconstruction function for the global collection. The following rule shows how we can create this function for a primary vertical fragmentation. The rule for primary horizontal fragmentation can be easily obtained by adaptation in this rule.

Definition 6: Let *GC* be a global collection and YF_1, YF_2, ..., YF_n its primary vertical hybrid fragments. This type of fragmentation is defined initially by a vertical fragmentation. According to Definition 3, the root of the global collection reconstruction function will be a join operation. If there are fragments YF_i defined only by the projection operation, such fragments can be treated as vertical fragments and added to the reconstruction function by using the rules of Definition 3. For each fragment YF_j defined over selection and projection operations, we must find its sibling fragments, that is, fragments the have projection operations identical to the one in YF_j. These sibling fragments are a result of a horizontal fragmentation applied over a vertical fragment, and can be united according to Definition 1. In this way, we can create the reconstruction function of a hybrid fragmentation using as basis the rules for horizontal and vertical fragmentation.

After constructing the algebraic expression using the reconstruction rules, we can apply the same principles applied to horizontal and vertical fragmentation for reducing irrelevant hybrid fragments. In this way, the reduction of a hybrid fragment can occur due to selection and or projection predicates criteria.

3.3. Global Optimization

The global optimization layer is responsible for obtaining an algebraic plan of minimal cost by creating equivalent variations of the algebraic plan obtained in the Localization layer. These variations are obtained through algebraic transformations. The minimal cost plan is obtained by using a cost function to choose the best among the alternative plans. The seek for the optimal plan can be, besides very difficult and sometimes impossible, very expensive to the query processing, which in the end reduces the gains obtained by the optimizations of the algebraic plan. Consequently, this layer should also use algebraic optimization heuristics and techniques for minimization of cost functions that have the lowest possible processing cost. This layer is out of the scope of this report. Thus we give readers only an overview of this layer.

The task of globally optimizing a distributed query starts by the generation of query plans that are equivalent to localized query plan. This can be done by changing the order of the operations in the plan; replacing the localization of fragments (when there are replicas of fragments in different nodes of the system); etc.

For each equivalent plan produced in this step, we calculate the estimated cost by using a cost function. The plan with lowest cost is chosen to execute the distributed query. This cost function should use and estimate parameters to calculate the cost of each plan operation, so it can obtain the total cost of the plan. Among the parameters, we can cite the data volume processed by the operation, the cost of disk access, the cost to data transfer in the network, volume estimations and histograms of the database, estimation of data volume returned by a given operation, etc.

The Global Optimization layer has as a result a near-to-optimal execution plan. Each operation in the optimized algebraic plan must know the site where it should be executed. This site can be a remote site or the distributed DBMS. This algebraic plan will be used to assemble sub-queries in XQuery that will execute all of the operations designated to a given site.

3.4. Local Optimization

Local optimization is performed by the DBMS that stores the XML fragments in each of the remote nodes. Any database capable of processing XQuery queries can be used, since the sub-queries are generated in XQuery. Details about XQuery local optimization in native XML DBMS are out of the scope of this work. Details about query processing in native XML databases can be found in [SCHONING 2001, FIEBIG et al. 2002, JAGADISH et al. 2002, MEIER 2002].

4. Implementation of the Methodology

To evaluate the technical viability of our methodology, and to evaluate the performance of query processing in a distributed environment, we propose an architecture based on the methodology we propose in Section 3. To compose a global view and also to serve as a unique access point to the system, we propose the use of a *mediator* [WIEDERHOLD 1992] as shown in Figure 5. The mediator is responsible for processing distributed XML queries, thus making localization and fragmentation issues completely transparent to users. Queries submitted over the global view are decomposed in sub-queries that are executed over the fragments in the remote sites. The results of each sub-query return to the mediator where the final result is built.

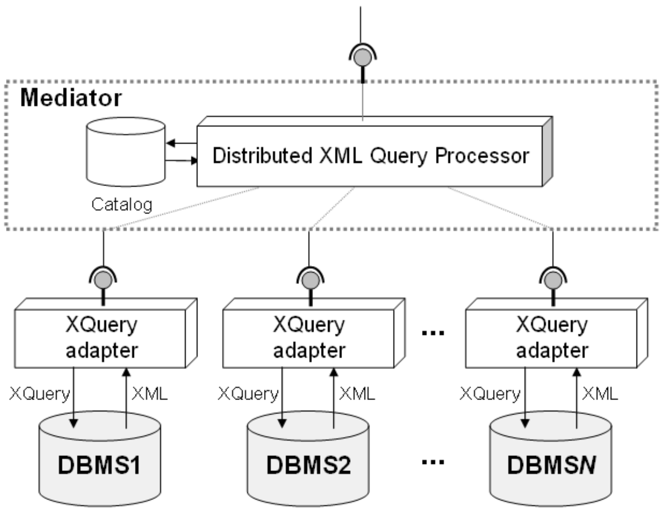


Figure 5. Mediator-Adapters architecture to execute XML queries over distributed databases

Our prototype implementation was completely based on the architecture presented in Figure 5. Its main components are explained in the next sections.

4.1. Mediator

The Mediator is the main component of the architecture, since it is responsible for processing the distributed query. It is also responsible for the decomposition and localization layers of our methodology.

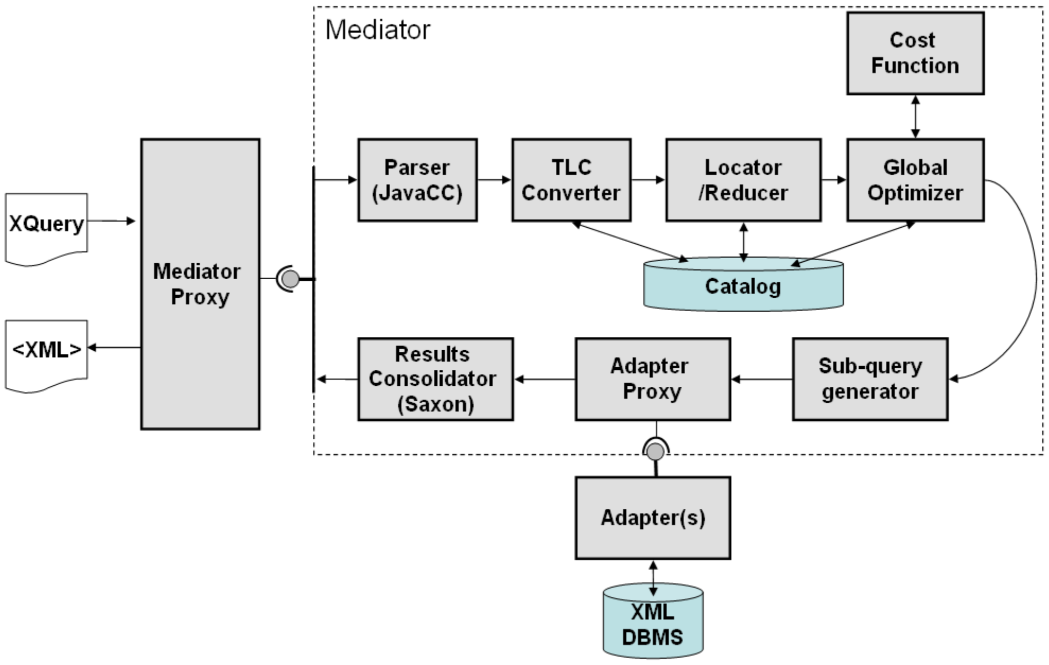


Figure 6. Blocks diagram of the Mediator components

The architecture we propose for the Mediator follows the basic query processing architecture presented in [KOSSMAN 2000], as shown in Figure 6. This architecture has several modules. Each of them is responsible for a step of the distributed query processing, as we describe below.

Parser: the parser is responsible for syntactically validating the XQuery query submitted by the user. The query parsing is the first step in the query decomposition layer, as mentioned in Section 3.1. To implement the parser, we have used the *JavaCC* with *JJTree* libraries [SUN MICROSYSTEMS 2006]. These libraries automatically constructs a parser using as input the grammar of the language that will be accepted by the parser. Besides performing syntactical validation, the parser transforms a textual query into an internal representation that will be used in the next processing step.

TLC Converter: this module transforms a XQuery query into na equivalent TLC algebraic representation. It implements the XQuery/TCL conversion algorithm proposed in [JAGADISH et al. 2001].

Locator/Reducer: this module is responsible for the data localization layer of our methodology. It performs the following activities: (i) replacement of references of global collections by references to fragments of these collections; (ii) reduction of irrelevant fragments (see Section 3.2). This module uses Catalog data to locate global collections.

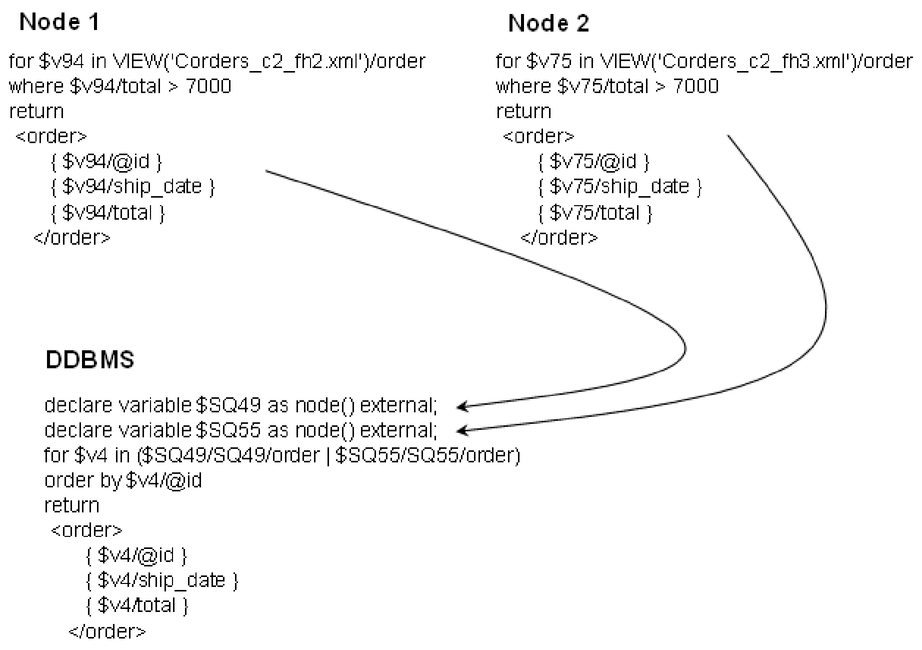
Global Optimizer: the global optimizer is responsible for the global optimization layer of our methodology. In our prototype implementation, we developed an optimizer that produces a set equivalent of algebraic plans from replicas of fragments existing in the distributed environment to find (by using a cost function) the lowest cost plan. Other approaches and heuristics could have been used to optimize the global query plan, as mentioned in Section 3.3.

Cost Function: this is the module responsible for calculating the cost of a given algebraic plan. To do so, it uses statistical data of each fragment, selectivity parameters, disk read weight, communication weight, etc. In the prototype implementation, our cost function uses only communication weight and an estimation of the total number of nodes of a fragment. The cost is calculated bottom-up. We perform an estimation of the data volume that will be processed by each operation in the plan, and also the data volume that will be transferred between the operations. Operations that are executed in the same node have no communication cost, but on the other hand, cannot be executed in parallel, which could compensate the communication costs in some cases.

Sub-query generator: this module generates the sub-queries of the optimized algebraic plan. Each sub-query in algebra is translated to XQuery and sent to the appropriate Adapter. A sub-query can also stay in the Mediator for composing the final result. The XQuery sub-queries are generated using the inverse XQuery/TLC algorithm used in the TLC Conversor, thus generating XQuery from TLC expressions.

Adapter Proxy: the Adapter proxy allows the communication between the Mediator and the Adapters. This is done by using protocols for Web Service calls. The proxy allows us to define the address of the Adaptor that will be invoked, making the communication process transparent to the rest of the Mediator.

Results Consolidator: this module generates the final result of a query. In our implementation, the final result composition is done through the execution of a local XQuery query over the results sent by the Adapters, as shown in Figure 7. We use the Saxon XQuery processor [SAXONICA LIMITED 2006] to execute the query in memory, without having to store the results sent by the Adapters (this would slow down the query execution). This was done to simplify the implementation of the prototype. Another alternative would be the Mediator to physically execute the algebraic operations of result composition. In this way, it could use streaming processing techniques, which would certainly improve the performance of queries over large volumes of data. When there is only one sub-query, there is no need for result composition -- the final result will be the result of this sub-query.



Node 1

```
for $v94 in VIEW('Corders_c2_fh2.xml')/order
where $v94/total > 7000
return
 <order>
     { $v94/@id }
     { $v94/ship_date }
     { $v94/total }
   </order>
```

Node 2

```
for $v75 in VIEW('Corders_c2_fh3.xml')/order
where $v75/total > 7000
return
 <order>
     { $v75/@id }
     { $v75/ship_date }
     { $v75/total }
   </order>
```

DDBMS

```
declare variable $SQ49 as node() external;
declare variable $SQ55 as node() external;
for $v4 in ($SQ49/SQ49/order | $SQ55/SQ55/order)
order by $v4/@id
return
 <order>
     { $v4/@id }
     { $v4/ship_date }
     { $v4/total }
   </order>
```

Figure 7. Sub-queries sent to remote nodes and to the Mediator

Note that the remote sub-queries on Figure 7 uses the *VIEW* expression to refer to a fragment. This sub-query will be processed by an Adaptor in the remote database, where this VIEW expression will be replaced by the correct XQuery syntax that represents the local address of the document or collection queried.

Another detail in this example is the execution of the order by operation only in the Mediator (not on the remote sub-queries). Since we use the Saxon API to execute the results consolidation sub-query in the Mediator, we have no control over the algorithm that is used in the union operation performed over the fragments. If we were executing our methodology in a native XML database, for instance, we could execute the ordering operation in the remote sub-queries and make the Mediator to unite the fragments using a *merge* algorithm, thus taking advantage of the pre-order of the results.

Mediator Proxy: in the same way of the Adapter Proxy, the Mediator Proxy allows a client to communicate with the Mediator through the implementation of communication protocols e together with the configuration of specific attributes in the Mediator. It is important to notice that, despite they are different proxies, the interface they implement is exactly the same. The Mediator proxy was implemented only to make the development of client applications easier.

4.2. Catalog

The Catalog stores all information needed to process a distributed query. In special, it serves the layers of data localization by providing the name and schema of global views, the fragments that compose the global view, the definition of each fragment, the address of each remote Adapter that has a copy of a given fragment, statistics about fragments.(total number of nodes, selectivity characteristics, etc.). The Catalog was implemented as a set of Java objects that can be serialized and deserialized in an XML document for manual edition.

One of the most important information stored in the Catalog is the definition of the fragments of the global collections. Using the relationship between the selection and projection criteria that forms the fragments, the Mediator is capable of assembling a reconstruction operation to reconstruct the global view from its remote fragments. The relationship of the fragment predicates also allows the Mediator to reduce the algebraic query plan by removing the operations over the irrelevant fragments for a given query.

4.3. Adaptors

The Adaptors are responsible for the execution of the sub-queries in the XML DBMS attached to them through a library or communication protocol. The result of each sub-query is sent to the Mediator for final result composition.

We have implemented two types of adapters to execute XQuery sub-queries. One uses eXist [EXIST DEVTEAM 2006] (a native XML DBMS) and the other one uses Saxon [SAXONICA LIMITED 2006] (a Java library to process XQuery queries over documents stored in the file system or in memory). By using the Web Service interface published by the Adapters, the nature of the XML database is transparent to the Mediator. It can be a native XML DBMS, as eXist, and also an XQuery processor API such as Saxon. This transparency to the Mediator allows one to implement different adapters to different XML databases, or even to relational or OO databases that publish XML views corresponding to the fragments needed by the Mediator.

The implementation of an adapter is relatively simple, since the XQuery query sent by the Mediator is almost read for execution. The only responsibility of the Adapter, besides implementing the communication interface with the Mediator, is to update the location of the queries XML document or collection to its address in the local database (or disk). To do so, the Adapter has a configuration file that contains the mapping of the document name (fragment) with its complete address in the local server. After performing this mapping, the Adaptor can execute the query using the interface or API to the database it is connected to. This resource allows the Mediator not to worry with storage details on the adapters. It delegates this responsibility. The blocks diagram of an Adaptor components is shown in Figure 8.

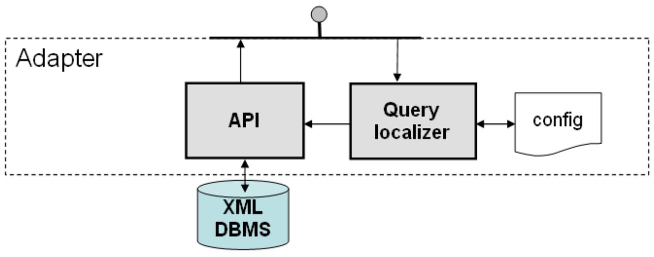


Figure 8. Blocks diagram of Adaptor components

Over the implementation of the Mediator and Adapters, we executed local tests to evaluate the implementation modules. After these initial tests, we started the planning of a series of experiments to be executed in laboratory and analyzed the results. We show them in the next section.

5. Experimental Evaluation

Our experiments were executed in laboratory. The environment was set up with three computers in a local network (Node 0, Node 1 and Node 2). The computers had all the same configuration: 1.8 GHz Dual Core Pentium with RAM memory of 1 GB running Windows XP. We have installed eXist Adapters in all of them. Additionally, one of the nodes (Node 0) played the role of the Mediator. The servers of the distributed environment were totally dedicated to our tests. We have programmed a simple application to collect the experimental results. They are analyzed in this section

We have defined four scenarios for our experiments: scenario 0, centralized with no fragmentation; scenario 1, distributed with no fragmentation; scenario 2, distributed with little fragmentation; scenario 3, distributed and heavily fragmented. They are described in details in Tables 1 and 2.

Table 1. Definition of the experimental scenarios 0, 1 and 2

Scenario 0				
Base	**Fragments**	**Type**	**Size**	**Location**
CLoja	CLoja_c0	SD	4,71 MB	Node 0
COrders	COrders_c0	MD	10,1 MB	Node 0
Scenario 1				
Base	**Fragments**	**Type**	**Size**	**Location**
CLoja	CLoja_c1	SD	4,71 MB	Node 1
COrders	COrders_c1	MD	10,1 MB	Node 1
Scenario 2				
Base	**Fragments**	**Type**	**Size**	**Location**
CLoja	CLoja_c2_fv1 := $\left\langle C_{loja}, \Pi_{/Loja, \{/Loja/Itens\}} \right\rangle$	SD	4 KB	Node 1
	CLoja_c2_fv2 := $\left\langle C_{loja}, \Pi_{/Loja/Itens, \{\ \}} \right\rangle$	SD	4,71 MB	Node 2
COrders	COrders_c2_fh1 := $\left\langle C_{orders}, \sigma_{/order/total \leq 4000} \right\rangle$	MD	3,23 MB	Node 0
	COrders_c2_fh2 := $\left\langle C_{orders}, \sigma_{/order/total > 4000 \wedge /order/total < 8000} \right\rangle$	MD	3,70 MB	Node 1
	COrders_c2_fh3 := $\left\langle C_{orders}, \sigma_{/order/total \geq 8000} \right\rangle$	MD	3,16 MB	Node 2

The queries were defined based on a benchmark [YAO et al. 2002], on related work [ANDRADE et al. 2006] and some were defined by us to include some queries that would be beneficiated with the fragmentation and others that wouldn't, in order to have experimental results in these two situations. The definition of the complete set of queries is available at Appendix A.

The analysis of the results was done by comparing the average total time of query execution in all of the scenarios. We have also analyzed the average total time of query execution without the communication time between the Mediator and the Adapters. Finally, for each query in each scenario, we analyzed the average total compilation time at the Mediator, so that we could individually verify the cost of query decomposition.

The comparison of the average total time of query execution in the scenarios we assembled was done through a graphic that shows the query execution time normalized by the execution time of the same query in scenario 0. Thus, we calculate for each query, the reason between its average execution time in scenarios 1, 2 and 2, with the average time in scenario 0. Consequently, the average time in scenario 0, is always equal to 1 in our graphics. Figure 9 shows a performance comparison of queries over a vertically/hybrid fragmented SD collection (*CStore*), while Figure 10 shows the results for queries over a horizontally fragmented MD collection (*COrders*).

Table 2. Definition of the experimental scenario 3

Scenario 3				
Base	**Fragments**	**Type**	**Size**	**Location**
CLoja	CLoja_c3_fy1 := $\langle C_{loja}, \Pi_{/Loja,\{/Loja/Itens\}} \rangle$	SD	4 KB	Node 1
	CLoja_c3_fy2 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="Brinquedos"} \rangle$	SD	1 MB	Node 0
	CLoja_c3_fy3 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="Games"} \rangle$	SD	284 KB	Node 0
	CLoja_c3_fy4 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="Perfumarid"} \rangle$	SD	97 KB	Node 1
	CLoja_c3_fy5 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="Eletronicos"} \rangle$	SD	727 KB	Node 1
	CLoja_c3_fy6 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="CD"} \rangle$	SD	907 KB	Node 2
	CLoja_c3_fy7 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="DVD"} \rangle$	SD	477 KB	Node 2
	CLoja_c3_fy8 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{/Item/Secao="Livraria"} \rangle$	SD	896 KB	Node 2
	CLoja_c3_fy9 := $\left\langle C_{loja}, \Pi_{/Loja/Itens,\{\ \}} \bullet \sigma_{\substack{/Item/Secao \neq "Brinquedos" \\ \wedge /Item/Secao \neq "Games" \\ \wedge /Item/Secao \neq "Perfumaria" \\ \wedge /Item/Secao \neq "Eletronicos" \\ \wedge /Item/Secao \neq "CD" \\ \wedge /Item/Secao \neq "DVD" \\ \wedge /Item/Secao \neq "Livraria"}} \right\rangle$	SD	648 KB	Node 2
COrders	COrders_c3_fh1 := $\langle C_{orders}, \sigma_{/order/total \leq 2000} \rangle$	MD	1,36 MB	Node 0
	COrders_c3_fh2 := $\langle C_{orders}, \sigma_{/order/total > 2000 \wedge /order/total \leq 4000} \rangle$	MD	1,86 MB	Node 0
	COrders_c3_fh3 := $\langle C_{orders}, \sigma_{/order/total > 4000 \wedge /order/total \leq 6000} \rangle$	MD	1,87 MB	Node 1
	COrders_c3_fh4 := $\langle C_{orders}, \sigma_{/order/total > 6000 \wedge /order/total \leq 8000} \rangle$	MD	1,83 MB	Node 1
	COrders_c3_fh5 := $\langle C_{orders}, \sigma_{/order/total > 8000 \wedge /order/total \leq 10000} \rangle$	MD	1,87 MB	Node 2
	COrders_c3_fh6 := $\langle C_{orders}, \sigma_{/order/total \geq 10000} \rangle$	MD	1,29 MB	Node 2

For the queries executed over the SD collection (Figure 9), there was no performance gains because of distribution and fragmentation. Even for those queries that would benefit from fragmentation (those that query a single fragment), the times introduced by the distributed architecture such as communication times between nodes and compilation time in the Mediator were significant when compared to the total query time in the centralized environment (scenario 0). Because of this, their performance were inferior in the distributed environment.

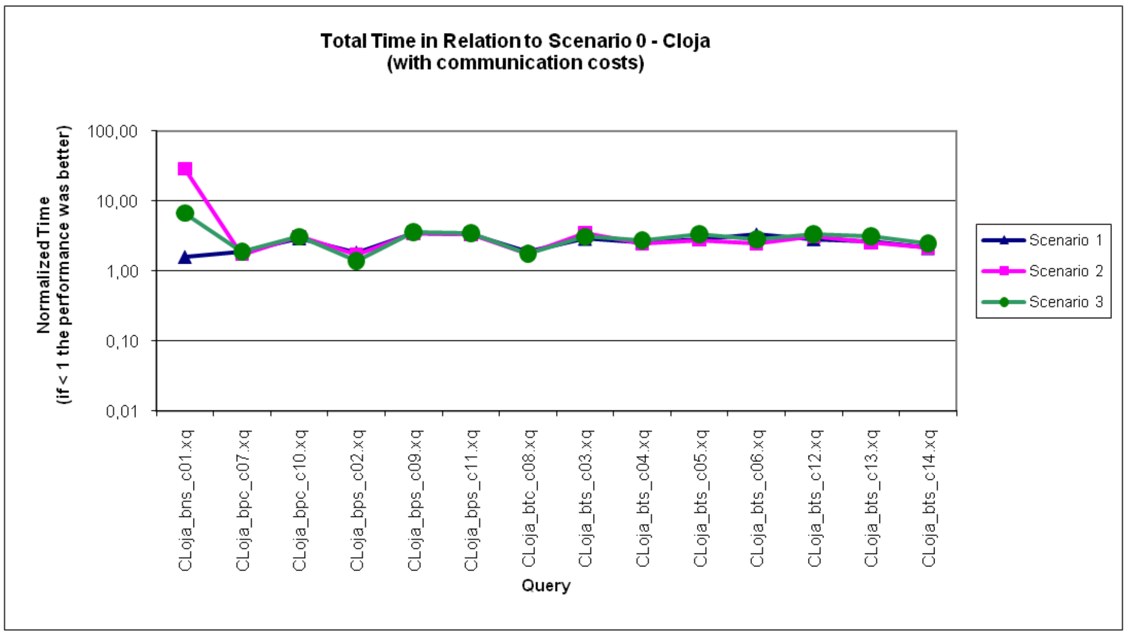


Figure 9. Comparison of Total Execution Time of queries over CLoja - scenarios 0 (normalized), 1, 2 e 3.

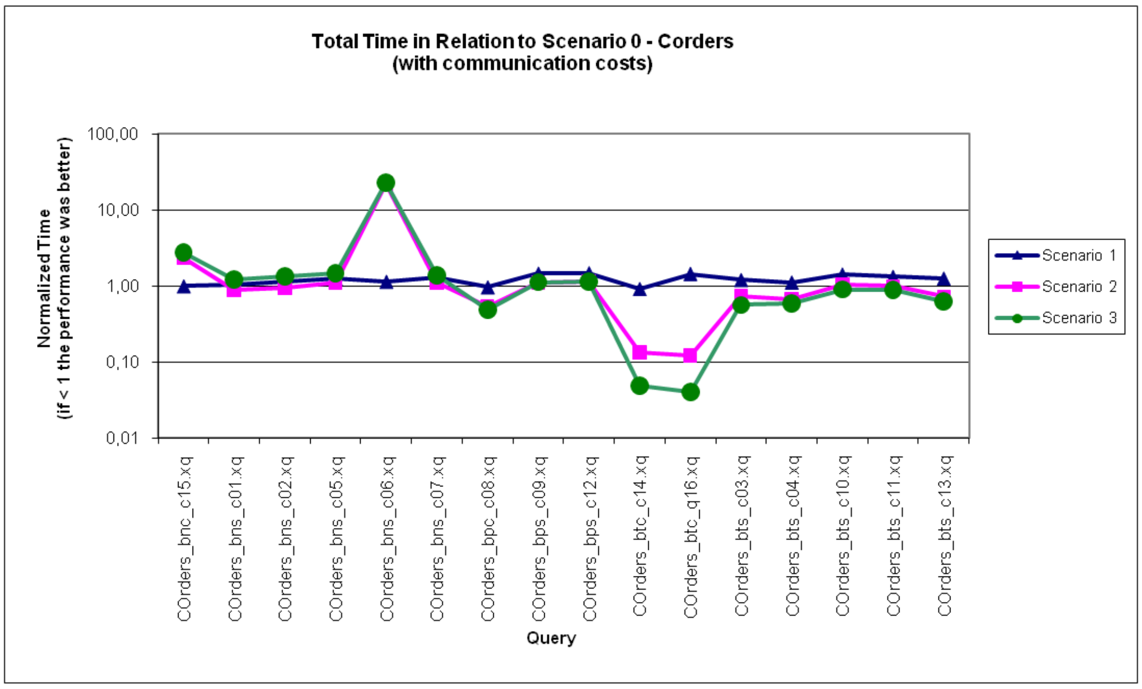


Figure 10. Comparison of Total Execution Time of queries over COrders - scenarios 0 (normalized), 1, 2 e 3.

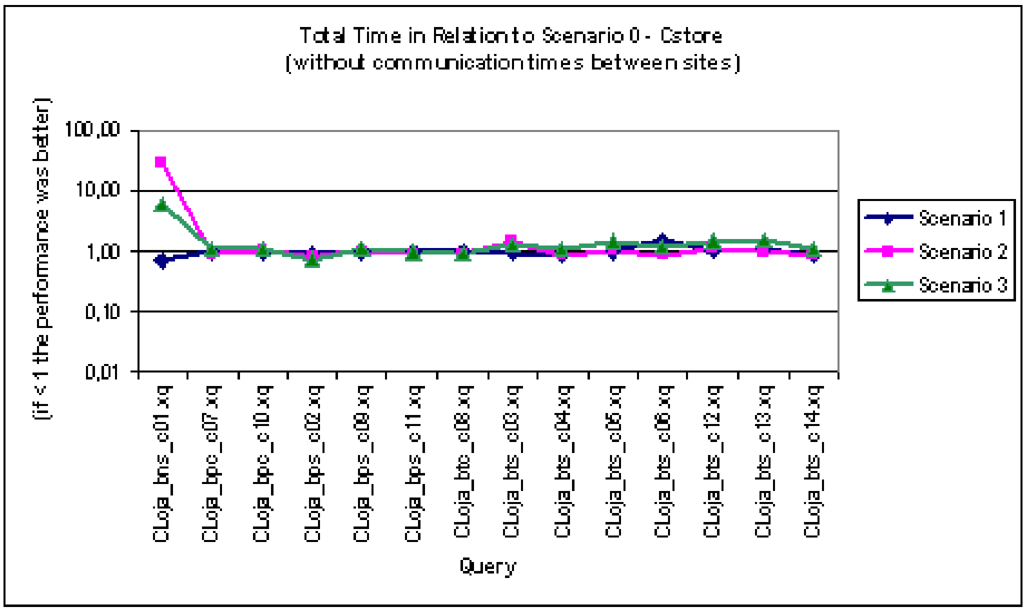


Figure 11. Comparison of total execution time of queries over a SD collection in scenarios 0 (normalized), 1, 2 and 3

Some queries over the MD collection (Figure 12) presented gains in the distributed environment. Some achieved reductions in the order of 95% when compared to the execution time in centralized environment. The major gains were observed in queries that totally benefit from fragmentation and have results aggregation. Queries with fragmentation benefit, but no aggregation functions obtained gains between 10 and 40%.

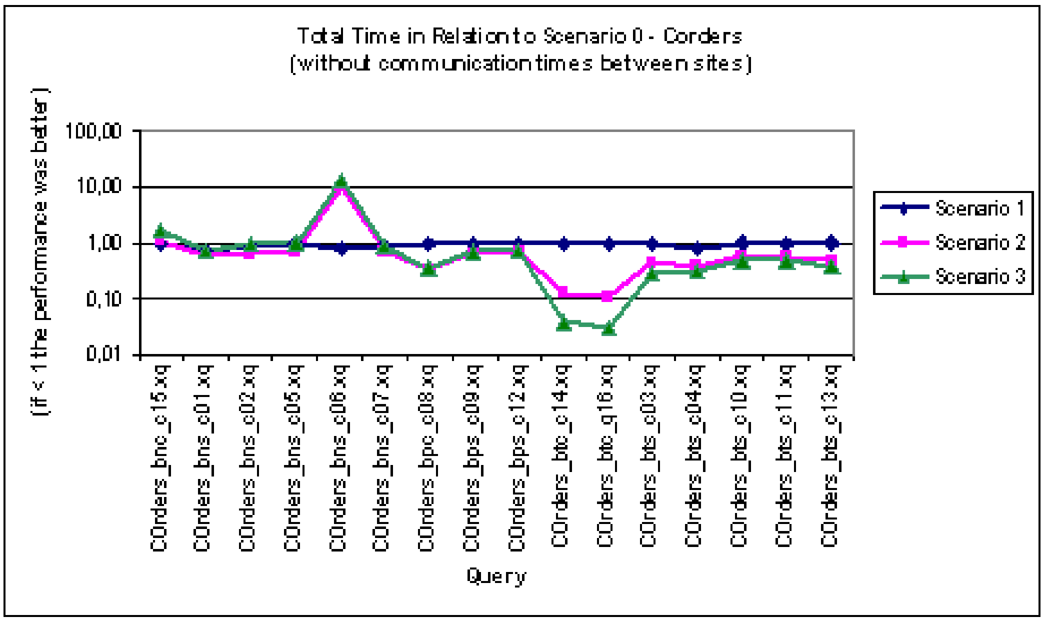


Figure 12. Comparison of total execution time of queries over a MD collection in scenarios 0 (normalized), 1, 2 and 3

During the analysis of our results, we noticed that the time spent with the communication between the nodes and the Mediator, and between the Mediator and the client, was high when compared to the total execution time of the queries. To evaluate the performance of the distributed queries without the interference of the communication costs, we have recalculated the query execution times by removing the communication costs. The results are presented in Figure 11 and Figure 12.

With these results, we conclude that the performance problems found in the queries over the SD collection were related to the communication cost in most of the times, especially because of the use of web services as the interface technology which increases the communication overhead. However, even when we exclude the communication costs, these queries still show no performance gains. When we compare the results obtained by the SD and MD collections, we can easily see that the fragmentation of the MD collection *COrders* presented better results. The (centralized) query processing over eXist in the SD collection was more efficient than over the MD collection. This explains the better results of the fragmentation of the MD collection. In a MD collection, the database has to parse all documents to process a query, and this increases the query execution time. For this reason, a SD collection with hybrid fragments will also have this disadvantage.

Based on our results, we can conclude that the fragmentation of an XML database is possible by using a XQuery decomposition system as proposed in our work. The results show that it is possible to reduce query execution times up to 95%, depending on the type of fragmentation, on the query and on the queried data volume. However, the fragmentation of na XML database needs to be carefully planned. It can highly improve the performance of queries that benefit from fragmentation, but it also can significantly reduce the performance of queries that do not benefit from fragmentation. The more fragmented a base is, the more severe these behaviors will be. For this reason, we need a methodology to design fragmented XML databases, similar to the existing ones for OO [BAIÃO et al. 2004] and relational [ÖZSU,VALDURIEZ 1999] models. With this, the fragmentation process of an XML base would be easier, or even automatic (based on the history of queries over the database).

6. Final Remarks

This report has shown a solution to query processing over distributed and fragmented native XML databases. Our goal was achieved by using the TLC algebra [PAPARIZOS et al. 2004] to process distributed queries, as well as a definition of XML fragmentation [ANDRADE et al. 2006] that provides us formal reconstruction rules of the original XML document from its fragments.

Our solution is based on distributed query processing techniques for relational databases [ÖZSU,VALDURIEZ 1999]. We make an analogy of the relational model with the semi-structured model so we can take advantage of the techniques in our approach. We propose the use of a Mediator with Adapters architecture [WIEDERHOLD 1992] for the distributed databases. The Mediator is responsible for processing the distributed query by implementing the layers: query decomposition, data localization, global optimization, generation of sub-queries to be sent to the Adapters, and consolidation of the final result. The Adapters are responsible for executing the sub-queries sent by the Mediator over the fragments. From the fragments definitions (stored in the Catalog), we could define rules to reduce the global query to assure that only relevant fragments would be accessed. This improves the performance of distributed queries.

We have implemented prototypes of the Mediator and Adapters by using the eXist native XML database [EXIST DEVTEAM 2006]. The implementation was totally based on the rules and definitions shown in this report, with the goal of proving the viability of our proposal. Several experiments were conducted using the three types of fragmentation: horizontal, vertical and hybrid.

Our experiments have shown that our solution can achieve performance improvements of up to 95% when compared to the centralized environment, for queries that benefit form the fragmentation. This reduction in query processing time was obtained due to the reduction of irrelevant fragments done by the Mediator, and also due to the intra-query parallelism of the distributed environment. Queries that do not benefit from fragmentation presented inferior processing time when compared to the centralized environment, because of the additional processing in the Mediator. The experiments have also shown that the use of Web Services in the interface of the components of our architecture compromised the queries performance due to the time spent with communication between nodes. On the other hand, the use of Web Services allows more interoperability between the architecture components. In this way, nodes can be heterogeneous, as in the example of integration of semi-autonomous databases of the branches of a bookstore.

References

ABITEBOUL, S. (1999) "On views and XML", In: PODS, p. 1-9. ACM Press, Philadelphia, Pennsylvania, United States.

ABITEBOUL, S., BUNEMAN, P., SUCIU, D. (1999) "Data on the Web: From Relations to Semistructured Data and XML", Morgan Kaufmann Publishers, San Francisco, California, USA.

AGUILERA, V., CLUET, S., MILO, T., VELTRI, P., VODISLAV, D. (2002) "Views in a Large Scale XML Repository", *The VLDB Journal*, v. 11, 3, p. 238-255.

ANDRADE, A., 2006, *PARTIX: Projeto de fragmentação de dados XML*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

ANDRADE, A., RUBERG, G., BAIÃO, F., BRAGANHOLO, V., MATTOSO, M. (2005) "PartiX: processing XQuery queries over fragmented XML repositories", In: Technical Report ES-691. COPPE/UFRJ, Rio de Janeiro, RJ.

ANDRADE, A., RUBERG, G., BAIÃO, F., BRAGANHOLO, V., MATTOSO, M. (2006) "Efficiently processing XML queries over fragmented repositories with PartiX", In: DATAX, p. 150-163, Munich, Germany.

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G. (2004) "A Distribution Design Methodology for Object DBMS", In: Distributed and Parallel Databases, v. 16, p. 45-90. Kluwer Academic Publishers, Hingham, MA, USA.

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G. (2000) "Horizontal Fragmentation in Object DBMS: New Issues and Performance Evaluation", In: IPCCC, p. 108-114. IEEE CS Press, Phoenix, AZ, USA.

BARU, C., GUPTA, A., LUDAESHER, B., MARCIANO, R., PAPAKONSTANTINOU, Y., PAVEL, V., CHU, V. (1999) "XML-Based Information Mediation with MIX", In: SIGMOD, p. 597-599. ACM Press.

BREMER, J.-M., GERTZ, M. (2003) "On Distributing XML Repositories", In: WebDB, p. 73-78, San Diego, California.

BUNEMAN, P. (1997) "Semistructured Data", In: PODS, p. 117-121. ACM Press, Tucson, Arizona.

CHEN, Z., JAGADISH, H. V., LAKSHMANAN, L. V. S., PAPARIZOS, S. (2003) "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery", In: VLDB, p. 237-248, Berlin, Germany.

EXIST DEVTEAM, 2006, "eXist: Open Source Native XML Database", v. 1.1. Disponível em http://exist.sourceforge.net/.

FERNÁNDEZ, M., SIMÉON, J., WADLER, P. (2000) "An Algebra for XML Query", In: FSTTCS, p. 11-45. Springer-Verlag, London, UK.

FIEBIG, T., HELMER, S., KANNE, C., MOERKOTTE, G., NEUMANN, J., SCHIELE, R., WESTMANN, T. (2002) "Anatomy of a native XML base management system", *The VLDB Journal*, v. 11, 4, p. 292-314.

FRASINCAR, F., HOUBEN, G.-J., PAU, C. (2002) "XAL: an algebra for XML query optimization", In: Australian Database Conference, p. 49-56. Australian Computer Society, Inc., Melbourne, Victoria, Australia.

GARDARIN, G., MENSCH, A., DANG-NGOC, T.-T., SMIT, L. (2002) "Integrating Heterogeneous Data Sources with XML and XQuery", In: DEXA, p. 839-846. IEEE Computer Society.

GERTZ, M., BREMER, J.-M. (2003) "Distributed XML Repositories: Top-down Design and Transparent Query Processing". *Department of Computer Science*.

GUPTA, N., HARITSA, J., RAMANATH, M. (2000) "Distributed Query Processing on the Web", In: ICDE, p. 1-20. IEEE Computer Society.

IVES, Z. G., HALEVY, A. Y., WELD, D. S. (2002) "An XML query engine for network-bound data", *The VLDB Journal*, v. 11, 4, p. 380-402.

JAGADISH, H. V., AL-KHALIFA, S., CHAPMAN, A., LAKSHMANAN, L. V. S., NIERMAN, A., PAPARIZOS, S., PATEL, J., SRIVASTAVA, D., WU, Y. (2002) "TIMBER: A native XML database", *VLDB Journal*, v. 11, 4, p. 274-291.

JAGADISH, H. V., LAKSHMANAN, L. V. S., SRIVASTAVA, D., THOMPSON, K. (2001) "TAX: A Tree Algebra for XML", In: DBPL, p. 149-164.

KOSSMAN, D. (2000) "The State of the Art in Distributed Query Processing", In: ACM Computing Surveys, v. 32, p. 422-469.

LEE, K., MIN, J., PARK, K., LEE, K. (2002) "A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources", In: Hawaii International Conference on System Sciences, v. 7, p. 202-211. IEEE Computer Society.

MA, H., SCHEWE, K.-D. (2003) "Fragmentation of XML documents", In: XVIII Simpósio Brasileiro de Banco de Dados, p. 200-214, Manaus, AM, Brasil.

MEIER, W. (2002) "eXist: An Open Source Native XML Database", In: Web, Web-Services, and Database Systems, v. 2593, p. 169-183. Springer, Erfurt, Germany.

ÖZSU, M. T., VALDURIEZ, P. (1999) "Principles of Distributed Database Systems". 2 ed., Prentice Hall

PAPARIZOS, S., WU, Y., LAKSHMANAN, L. V. S., JAGADISH, H. V. (2004) "Tree Logical Classes for Efficient Evaluation of XQuery", In: SIGMOD, p. 71-82. ACM.

RE, C., BRINKLEY, J., HINSHAW, K. P., SUCIU, D. (2004) "Distributed XQuery", In: IIWeb, p. 116-121, Toronto, Canada.

SAXONICA LIMITED, 2006, "Open Source SAXON XSLT Processor, v.8.8". Disponível em http://saxon.sourceforge.net/.

SCHONING, H. (2001) "Tamino - A DBMS designed for XML", In: ICDE, p. 149-154. IEEE Computer Society, Washington, DC, USA.

SILVEIRA, F. V., HEUSER, C. (2005) "Decomposição de Consultas sobre Múltiplas Fontes XML", In: I Escola Regional de Banco de Dados, Porto Alegre, RS, Brasil.

SUCIU, D. (2002) "Distributed Query Evaluation on Semistructured Data", *ACM TODS*, v. 27, 1, p. 1-62.

SUN MICROSYSTEMS, 2006, "Java Compiler Compiler 4.0 (JavaCC)". Disponível em https://javacc.dev.java.net/.

W3C, W. W. W. C. (2006), "Extensible Markup Language (XML) 1.0". Disponível em: http://www.w3.org/TR/REC-xml/, acessado em 20/01/2007.

WIEDERHOLD, G. (1992) "Mediators in the Architecture of Future Information Systems", In Michael N.Huhns and Munindar P.Singh, *Readings in Agents*Morgan Kaufmann

YAO, B., ÖZSU, M. T., KEENLEYSIDE, J. (2002) "XBench: A Family of Benchmarks for XML DBMSs", In: EEXTT.

ZHANG, X., PIELECH, B., RUNDESNTEINER, E. (2002) "Honey, I shrunk the XQuery!: an XML algebra optimization approach", In: WIDM, p. 15-22. ACM Press, New York, NY, USA.

Appendix A – Queries used in the experimental evaluation

CLoja_bns_c01.xq	`<results>`
	` {`
	`   for $x in collection('Cloja_c?.xml')/Loja`
	`   for $a in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`   where $a/Secao = "CD"`
	`   return`
	`      <loja>`
	`        { $a/Nome }`
	`        { $x/Secoes }`
	`      </loja>`
	`  }`
	`</results>`

CLoja_bps_c02.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Lancamento = "T"`
	`    order by $x/Codigo`
	`    return`
	`        <lancamento_t>`
	`            { $x }`
	`        </lancamento_t>`
	`  }`
	`</results>`

CLoja_bts_c03.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "CD"`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`        </output>`
	`  }`
	`</results>`

CLoja_bts_c04.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "Livraria"`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`        </output>`
	`  }`
	`</results>`

CLoja_bts_c05.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "CD"`
	`          and $x/Lancamento = "T"`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`        </output>`
	`  }`
	`</results>`

CLoja_bts_c06.xq

```
<results>
 {
   for $x in collection('Cloja_c?.xml')/Loja/Itens/Item
   where $x/Secao = "Perfumaria"
   return
       <output>
          { $x/Nome }
          { $x/Preco }
       </output>
 }
</results>
```

CLoja_bpc_c07.xq

```
<results>
 {
   for $x in collection('Cloja_c?.xml')/Loja/Itens/Item
   where count($x/Caracteristica) >= 4
   order by $x/Codigo
   return
       <output>
          { $x }
       </output>
 }
</results>
```

CLoja_btc_c08.xq

```
<results>
 {
   for $x in collection('Cloja_c?.xml')/Loja/Itens/Item
   where $x/Secao = "CD"
        and count($x/Caracteristica) >= 4
   return
       <output>
          { $x }
       </output>
 }
</results>
```

CLoja_bps_c09.xq

```
<results>
 {
   for $x in
collection('Cloja_c?.xml')/Loja/Funcionarios/Funcionario
   return
       <output>
          { $x }
       </output>
 }
</results>
```

CLoja_bpc_c10.xq

```
<results>
 {
   for $x in collection('Cloja_c?.xml')/Loja/Funcionarios
   return
    <TotalPagamento>
        { sum($x/Funcionario/Salario) }
    </TotalPagamento>
 }
</results>
```

CLoja_bps_c11.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja`
	`    return`
	`        <loja>`
	`            { $x/Funcionarios }`
	`        </loja>`
	`  }`
	`</results>`

CLoja_bts_c12.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "Brinquedos"`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`            { $x/Preco }`
	`        </output>`
	`  }`
	`</results>`

CLoja_bts_c13.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "Brinquedos"`
	`       and $x/Preco > 50`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`            { $x/Preco }`
	`        </output>`
	`  }`
	`</results>`

CLoja_bts_c14.xq	`<results>`
	`  {`
	`    for $x in collection('Cloja_c?.xml')/Loja/Itens/Item`
	`    where $x/Secao = "Perfumaria"`
	`       and $x/Preco > 40`
	`    return`
	`        <output>`
	`            { $x/Nome }`
	`            { $x/Preco }`
	`        </output>`
	`  }`
	`</results>`

COrders_bns_c01.xq	`<results>`
	`  {`
	`    for $order in collection('Corders_c?.xml')/order`
	`    where $order/@id = "1"`
	`    return`
	`       <order>`
	`        { $order }`
	`       </order>`
	`  }`
	`</results>`

COrders_bns_c02.xq	`<results>`
	`  {`
	`  `
	`    for $a in collection('Corders_c?.xml')/order`
	`    where $a/@id = "3"`
	`    return`
	`        <items>`
	`           { $a//order_line/item_id }`
	`        </items>`
	`  }`
	`</results>`

COrders_bts_c03.xq	`<results>`
	`  {`
	`  `
	`    for $a in collection('Corders_c?.xml')/order`
	`    where $a/total > 11000`
	`    order by $a/ship_type, $a/@id`
	`    return`
	`    <Output>`
	`        {$a/@id}`
	`        {$a/order_date}`
	`        {$a/ship_type}`
	`    </Output>`
	`  }`
	`</results>`

COrders_bts_c04.xq	`<results>`
	`  {`
	`  `
	`    for $a in collection('Corders_c?.xml')/order`
	`    where $a/total > 11000.0`
	`    order by $a/total descending, $a/@id`
	`    return`
	`    <Output>`
	`        {$a/@id}`
	`        {$a/order_date}`
	`        {$a/total}`
	`    </Output>`
	`  }`
	`</results>`

COrders_bns_c05.xq	`<results>`
	`  {`
	
	`    for $a in collection('Corders_c?.xml')/order`
	`    where $a/@id = "5"`
	`    return`
	`     <Output>`
	`        {$a/order_lines}`
	`    </Output>`
	`  }`
	`</results>`

COrders_bns_c06.xq	`<results>`
	`  {`
	
	`    for $a in collection('Corders_c?.xml')/order`
	`    where count($a/order_lines/order_line) = 1`
	`    order by $a/@id`
	`    return`
	`     <Output>`
	`        {$a/@id}`
	`    </Output>`
	`  }`
	`</results>`

COrders_bns_c07.xq	`<results>`
	`  {`
	
	`    for $a in collection('Corders_c?.xml')/order`
	`    where $a/@id = "6"`
	`    return`
	`     <Output>`
	`        {$a}`
	`    </Output>`
	`  }`
	`</results>`

COrders_bpc_c08.xq	`<results>`
	`  {`
	`    for $order in collection('Corders_c?.xml')/order`
	`    let $l := $order/order_lines/order_line`
	`    where $order/total > 7000`
	` and count($l) >= 5`
	`    order by $order/ship_date, $order/@id`
	`    return`
	`        <order>`
	`          { $order/@id }`
	`          { $order/ship_date }`
	`          { $order/total }`
	`          <total_items>`
	`              { count($l) }`
	`          </total_items>`
	`        </order>`
	`  }`
	`</results>`

COrders_bps_c09.xq	<results>
	  {
	    for $order in collection('Corders_c?.xml')/order
	    where $order/total > 7000
	    order by $order/ship_date, $order/@id
	    return
	        <order>
	          { $order/@id }
	          { $order/ship_date }
	          { $order/total }
	        </order>
	  }
	</results>

COrders_bts_c10.xq	<results>
	  {
	    for $order in collection('Corders_c?.xml')/order
	    where $order/total > 10000
	    order by $order/ship_date, $order/@id
	    return
	        <order>
	          { $order/@id }
	          { $order/ship_date }
	          { $order/total }
	        </order>
	  }
	</results>

COrders_bts_c11.xq	<results>
	 {
	    for $order in collection('Corders_c?.xml')/order
	    where $order/total > 10000
	    order by $order/@id
	    return
	        <order>
	          { $order/@id }
	          { $order/ship_date }
	          { $order/total }
	        </order>
	  }
	</results>

COrders_bps_c12.xq	<results>
	 {
	    for $order in collection('Corders_c?.xml')/order
	    where $order/total > 7000
	    order by $order/@id
	    return
	        <order>
	          { $order/@id }
	          { $order/ship_date }
	          { $order/total }
	        </order>
	  }
	</results>

COrders_bts_c13.xq

```
<results>
 {
    for $order in collection('Corders_c?.xml')/order
    where $order/total > 7000
 and $order/total < 8000
    order by $order/@id
    return
        <order>
          { $order/@id }
          { $order/ship_date }
          { $order/total }
        </order>
  }
</results>
```

COrders_btc_c14.xq

```
<results>
  {
    for $order in collection('Corders_c?.xml')/order
    let $l := $order/order_lines/order_line
    where $order/total < 2000
 and count($l) >= 5
    order by $order/ship_date, $order/@id
    return
        <order>
          { $order/@id }
          { $order/ship_date }
          { $order/total }
          <total_items>
              { count($l) }
          </total_items>
        </order>
  }
</results>
```

COrders_bnc_c15.xq

```
<results>
  {
    for $order in collection('Corders_c?.xml')/order
    let $l := $order/order_lines/order_line
    where count($l) >= 5
    order by $order/ship_date, $order/@id
    return
        <order>
          { $order/@id }
          { $order/ship_date }
          { $order/total }
          <total_items>
              { count($l) }
          </total_items>
        </order>
  }
</results>
```

COrders_btc_q16.xq

```
<results>
  {
    for $order in collection('Corders_c?.xml')/order
    let $l := $order/order_lines/order_line
    where $order/total > 11000
 and count($l) >= 5
    order by $order/ship_date, $order/@id
    return
        <order>
          { $order/@id }
          { $order/ship_date }
          { $order/total }
          <total_items>
              { count($l) }
          </total_items>
        </order>
  }
</results>
```