Processing Queries over Distributed XML Databases

Guilherme Figueiredo[1,3], Vanessa Braganholo[2], Marta Mattoso[3]

[1] Brazilian Development Bank (BNDES), Brazil
[2] IC, Fluminense Federal University (UFF), Brazil
[3] COPPE, Federal University of Rio de Janeiro (UFRJ), Brazil
guilherme.coelho@bndes.gov.br, vanessa@ic.uff.br, marta@cos.ufrj.br

Abstract. The increasing volume of data stored as XML documents makes fragmentation techniques an alternative to the performance issues in query processing. Fragmented databases are feasible only if there is a transparent way to query the distributed database. Fragments allow for intra-query parallel processing and data reduction. This paper presents our methodology for XQuery query processing over distributed XML databases. The methodology comprises the steps of query decomposition, data localization, and global optimization. This methodology can be used in an XML database or in a system that publishes homogeneous views of semi-autonomous databases. An implementation has been done and experimental results can achieve performance improvements of up to 95% when compared to the centralized environment.

Categories and Subject Descriptors: H. Information Systems [**H.2. Database Management**]: H.2.4 Systems—*Distributed Databases*

Keywords: distribution, query processing, XML

1. INTRODUCTION

The increasing volume of stored XML data poses new challenges to efficient query processing. Queries on centralized databases may take a very long time to be processed, since large amounts of data need to be accessed. In most of the cases, indexes or even fine tuning are not enough to increase query performance.

In relational [Ozsu and Valduriez 1999] and object-oriented databases [Baião et al. 2004], fragmentation techniques have been successfully used to increase query performance. By fragmenting and distributing the data, queries can be sent to specific fragments, avoiding a complete scan over large portions of irrelevant data. Additionally intra-query parallelism can be achieved. The same ideas can be applied to XML. In fact, several systems have been proposed to process queries over distributed XML data [Suciu 2002; Aguilera et al. 2002; Re et al. 2004; Ives et al. 2002; Gertz and Bremer 2003; Kurita et al. 2007]. Some others focus on XML query processing over heterogeneous distributed systems [Gardarin et al. 2002; Baru et al. 1999; Lee et al. 2002; Kling et al. 2009]. In fact, most of the existing work on query processing focus on XML documents that have been distributed, but not fragmented [Gardarin et al. 2002; Baru et al. 1999; Lee et al. 2002; Abiteboul et al. 2008; Ferraz et al. 2010]. Only [Kling et al. 2009] deals with XML data fragmentation. However, their work does not provide a query processing methodology.

Defining data fragments is not enough. An efficient distributed query processing needs a query methodology that automatically translates a centralized query into an efficient distributed query execution plan. Thus, to be able to define a generic and automatic approach to process distributed XML queries, we need: fragmentation techniques to partition XML documents, fragmentation correctness

This work was conducted while Vanessa was at UFRJ. This research was partially supported by CNPq and FAPERJ.

rules, which are needed for query rewriting, and the corresponding formalism. When fragments and queries are represented in an algebraic form, the properties of the algebra can be used to replace references to the centralized database by references to its fragments in a given query. This can be achieved as long as both the fragments and the query are expressed in this same algebra. Also, a set of equivalence rules for the operations in the algebra needs to be provided. The algebraic properties allow us to formally prove that this query rewriting is correct. In this paper, we adopt the fragmentation technique of Andrade et al. [Andrade et al. 2006]. The choice of [Andrade et al. 2006] has several reasons. First, it provides correctness rules and an algebraic representation for the fragments. This is important since the query processing methodology needs to generate sub-queries. To ensure this is done correctly, an algebraic query representation is an important choice. The XML algebra used by [Andrade et al. 2006] is TLC (Tree Logical Class) [Paparizos et al. 2004]. This is also interesting because TLC is a set-based XML algebra. Being set-based allows us to mirror the techniques defined for the relational model when defining the query processing methodology, since the relational algebra is set-based. TLC provides lots of operations that are close to the relational ones: selection, projection, join, duplicate elimination, among others.

Thus, to process an XQuery over a fragmented database, the first step is to rewrite it in TLC and map global collections of XML documents to their equivalent union/join of fragments[1]. This way, the TLC query expression can be automatically rewritten to be expressed on the corresponding fragments instead of the centralized (virtual) database. However, a formal algebra is not enough. We need also to define a methodology with the steps needed to automatically generate an optimized execution plan, and a software architecture to execute this plan as distributed queries. This methodology is the main contribution of this paper. By generating plans with XQuery distributed sub-queries, they can be executed by the XQuery DBMS that manages the XML document locally. Thus, our solution is non-intrusive, i.e. it can be coupled to pre-existing XQuery engines. There is no need to change the DBMS. Local query engines are unaware of the global query processing controlled by the mediator.

Our non-intrusive approach to improve query performance over large repositories can be applied not only in fragmented databases, but also in semi-autonomous databases. 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 organization (ex: the total amount of sales per region in a given month), they must send individual queries to each of the databases, and then collect the results. However, when real time results are needed, this may not be the best alternative. 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 to distribute the query among each local branch library database. This would be completely transparent to the company directors, and real time results would be possible. From now on, we call this global (virtual) database a global view. Similarly, the local databases could be seen as local views [Abiteboul 1999].

Despite the existence of several fragmentation techniques, they lack a formal definition with the corresponding distributed query methodology to automatically and efficiently process distributed XML queries. To fulfill this gap, the contributions of this paper are threefold: (i) a comprehensive methodology to process queries over distributed XML databases; (ii) an architecture that implements the proposed approach in a non-intrusive way; (iii) an abstraction to perceive semi-autonomous databases as part of a single (virtual) distributed database.

It is important to note that there are several challenges when we consider distributed query processing XML data instead of relational data. The lack of a standard XML algebra, together with the lack of a standard definition for XML fragments makes an SGBD internal implementation not possible. Also, the TLC algebra operations, despite similar to the relational ones, are not equal, so adaptations

[1]Notice that TLC is unordered, and thus not all queries can be rewritten to TLC. In our approach, we deal only with the subset of queries that can be rewritten by TLC.

need to be performed. In this paper, we address all of these challenges.

This paper 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 some details of the implementation of our methodology to process distributed queries. Section 5 contains an experimental evaluation. Finally, we conclude on Section 7.

2. BACKGROUND

Query processing over distributed and fragmented databases presents several challenges. In a distributed environment, the DDBMS (Distributed 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 plan so it can be executed only in nodes that have data that can contribute to results. In [Ozsu and 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. 2004]. More advanced topics on distributed query processing, such as optimization and execution techniques, caching, architectures, etc., can be found in [Kossmann 2000].

An XML repository is a collection of XML documents (a set of data trees). An XML fragment is also a collection of XML documents. A collection may have a single document (SD), or multiple documents (MD). However, defining a fragment as a collection is not enough. 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 map the query on global collections to their fragments.

Several fragmentation techniques for XML repositories have been proposed in literature [Andrade et al. 2006; Ma and Schewe 2003; Bremer and Gertz 2003; Gertz and Bremer 2003]. We have adopted Andrade et al. proposal [Andrade et al. 2006], where horizontal, vertical and hybrid fragmentation are formally defined through TLC [Paparizos et al. 2004] algebra operations. When using TLC to decompose XQuery into TLC algebraic expressions, we can uniformly replace collections by the TLC expressions that represent the corresponding fragments. Besides the formal aspect of integrating algebraic expressions, the definitions of Andrade et al. can be applied to both SD and MD repositories [Yao et al. 2004]. Another interesting aspect of this work is that the fragmentation definitions are very similar to the ones proposed to the relational model [Ozsu and Valduriez 1999]. They also present correctness rules for each fragmentation type. Thus, adapting successful relational techniques to the XML model becomes an attractive and promising option, which we decided to explore.

3. A METHODOLOGY FOR DISTRIBUTED XML QUERY PROCESSING

This section presents our methodology to process XQuery queries over distributed XML databases. It assumes that global XML collections are fragmented and distributed across the network. The methodology involves the decomposition of the main query into sub-queries that will be executed in the remote sites containing fragments of the global collections. Our methodology is an adaptation of the four generic layers proposed by [Ozsu and Valduriez 1999] illustrated in Fig. 1.

Given an XQuery, we first analyze it and transform it into an algebraic query in TLC. This is done by the query decomposition layer. Then, the query tree is further analyzed to replace references of the global collection by its corresponding fragments (localization layer). The localization layer also reduces the query tree by removing irrelevant fragments (that is, fragments that do not contribute to the query result). Next global optimization is performed. At this point, sub-queries are generated and sent to the local sites. Each site performs local optimizations and executes the sub-query. Results are sent back to the central node, consolidated, and finally sent back to the user.

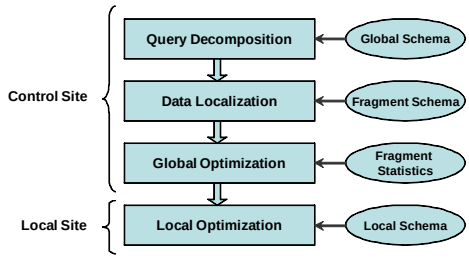


Fig. 1. Generic Layering Scheme for distributed query processing

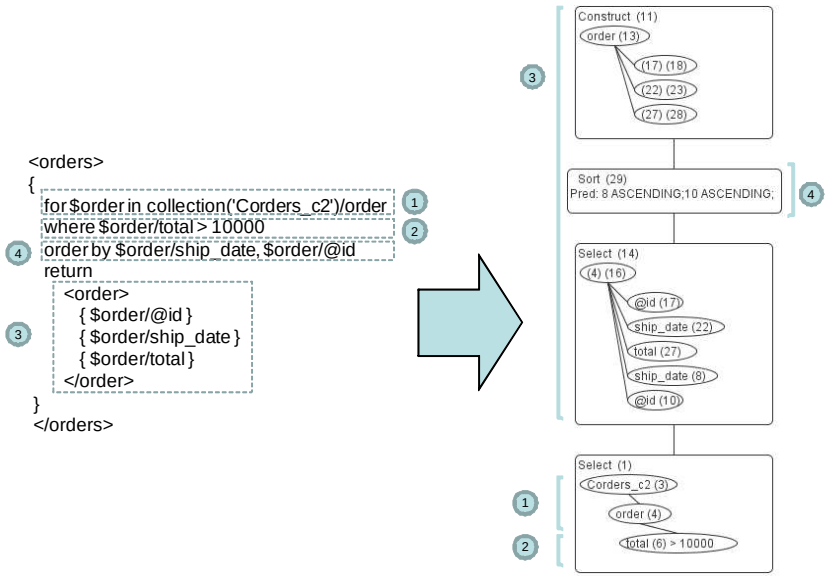


Fig. 2. XQuery query and its TLC representation

3.1 Query decomposition

The job of the first layer consists in translating 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. Information about data distribution is not used in this phase. This layer is very similar to XQuery processing in centralized environments. It has four main steps: syntactic validation; semantic validation; query simplification; and query translation to an algebraic tree.

The final product of the query decomposition layer is the algebraic tree of the query over the global collections. To do this, we use an algorithm that syntactically analyzes the query and generates the algebraic expressions, the pattern trees and the TLC logical classes based on the algorithm in [Paparizos et al. 2004]. Notice that this algorithm is publicly available as implemented in Timber [Wu et al. 2008] to translate XQuery to TLC. An example is shown in Fig. 2, which shows an XQuery and its corresponding representation on TLC. The query retrieves the *id*, *ship date* and *total* of orders with *total* greater than 10000 from a collection of orders extracted from [Yao et al. 2004]. Notice that the operations on the tree are processed bottom-up. *Select* and *Construct* operations have APTs as parameters. The APTs are graphically represented as a tree, and each node in the tree have an associated LCL (Logical Class Label [Paparizos et al. 2004] – the number in parentheses). In the bottom node, for instance, the logical class *order* has LCL 4. This LCL is used in the second *Select* operation, which receives a new LCL (16).

As for the query translation itself, the first thing that is done is the selection of *order*s that has *total* $>$ *10000* (steps 1 and 2 in the figure). Then, the elements that are involved in the "order by

clause" and construct operations are selected (step 3). The order by operation is performed (step 4), and the final result is constructed through a *construct* operation.

3.2 Data localization

The data localization layer receives as input the TLC query tree over global collections and has two main steps. The first consists in replacing the global collections on the algebraic query tree by the corresponding algebraic expression over the fragments that reconstructs the global collection. The second step tries to reduce the query tree by eliminating fragments that do not contribute to the query result. This reduction is based on the selection and projection predicates of the original query, in a similar way of what is done in [Ozsu and Valduriez 1999]. This layer is the main responsible for achieving benefits in query performance, since a query may have a better performance when irrelevant fragments are 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 stored in the catalog of the distributed database.

When fragments follow the correctness rules of [Andrade et al. 2006], we can use the reconstruction property to guarantee that there will be a TLC expression that is capable of reconstructing the global collection from its fragments. This allows for transparent and correct replacement of the global collection. This expression 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 selection predicates. In this way, the reconstruction can be made through the TLC union operation over these fragments [Andrade et al. 2006]. Let GC be a global collection and $HF_1, HF_2, ..., HF_n$ its horizontal fragments. According to [Andrade et al. 2006], $GC = HF_1 \cup HF_2 \cup ... \cup HF_n$, where $\cup$ denotes the union of two trees, resulting in a set of trees. Note that in this section we use the syntax of [Andrade et al. 2006] instead of that of [Paparizos et al. 2004]. A mapping between these two syntaxes is given in [Andrade et al. 2005].

Fig. 3 shows an example of localization of a TLC selection operation over a global collection composed by three horizontal fragments. Notice that the reference to the global collection (Corders_c2) was replaced by unions of its horizontal fragments Corders_c2_fh1, Corders_c2_fh2 and Corders_c2_fh3. These fragments are defined over the *total* element, such as Corders_c2_fh1 has orders with total ≤ 4000, Corders_c2_fh2 has orders with total among 4001 and 7999, and Corders_c2_fh3 has orders with total >8000.

After this we can move to the next step: reducing the query tree by eliminating irrelevant fragments. In the case of horizontal fragments, 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 definition. These fragments can be eliminated from the query tree. Formally, the elimination of fragments can be stated as follows (adapted from [Ozsu and Valduriez 1999]). Let GC be a global collection and H=$\{HF_1, HF_2, ..., HF_n\}$ the set of its horizontal fragments. Let p_i be the selection predicate over GC that defines fragment HF_i, such that $HF_i = \sigma_{p_i}(GC)$, where σ denotes a selection. Let p_q be the selection predicate of a query q over the global collection GC mapped to $HF_1 \cup HF_2 \cup ... \cup HF_n$. Let A be the algebraic tree representing query q over GC.

PROPOSITION 3.1. *A fragment HF_i can be eliminated from A if $\sigma_{p_q}(HF_i) = \emptyset$. This selection is empty if $\forall$ s in GC: $\neg(p_q(s) \wedge p_i(s))$, where $p(s)$ is TRUE if the sub-tree s satisfies the predicate p, and FALSE otherwise.*

PROOF SKETCH. By decomposing $\neg(p_q(s) \wedge p_i(s))$, we have $\neg p_q(s) \vee \neg p_i(s)$. By definition of

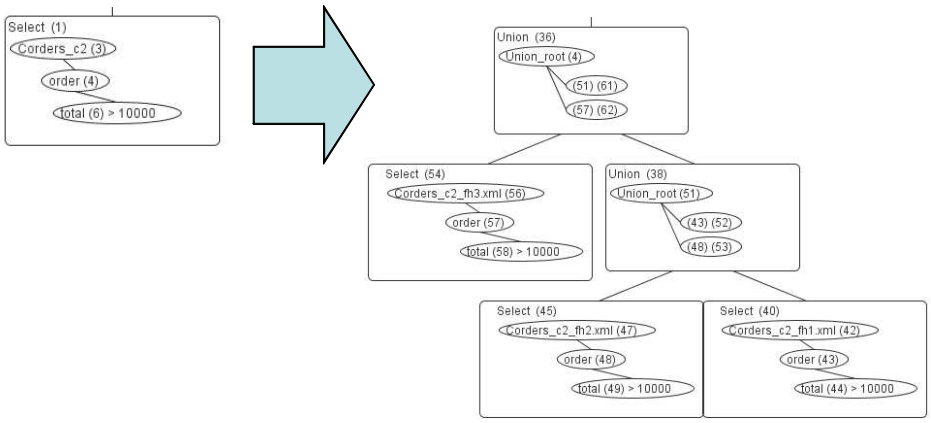


Fig. 3. Localization of a selection operation over a global collection horizontally fragmented

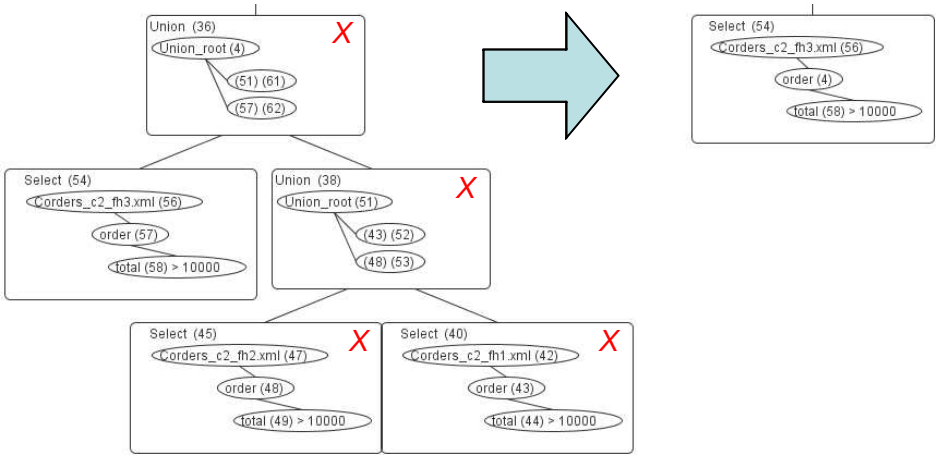


Fig. 4. Reduction of irrelevant operations to the final result

Horizontal Fragments, when $s \notin HF_i$, $p_i(s)$ is False. Then, $\neg p_i(s)$ is True and consequently, $\neg(p_q(s) \wedge p_i(s))$ is True. In this case, by definition, fragment HF_i is empty, and thus can not contribute to the query answer. For the case when $s \in HF_i$, we have $p_i(s)$ True, and consequently $\neg p_i(s)$ is False. Thus, the only way for $\neg p_q(s) \vee \neg p_i(s)$ is when $\neg p_q(s)$ is True. Thus, $p_q(s)$ must not hold, and the fragment can be discarded.

Fig. 4 shows the result of the reduction step of a query over a collection that is horizontally fragmented. The three fragments are formed by a selection criterion 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 Proposition 3.1. In the example, the query criterion is *total* $>$ *10000*. Clearly, fragments Corders_c2_fh1 and Corders_c2_fh2 can be eliminated, since they comprise documents with total $\leq$ 4000 and among 4001 and 7999, respectively. Thus, they cannot contribute to the result of the query. However, to do this automatically both the query predicate and the fragmentation definition predicate must be formally expressed under the same formalism. Then first order predicate calculus can be applied to perform the reductions. Other reductions due to join relationships can also be applied in the same way as in the relational algebra.

The reduction of a selection operation over a fragment in an algebraic plan of a horizontally fragmented database also implies the removal of the parent union operation, which should be replaced by its sibling (if any). Formally, when fragment C can be removed from $A \cup (B \cup C)$, the resulting

expression is $A \cup B$. This procedure is applied in all remove operations, thus resulting in a reduced query plan.

3.2.2 *Vertical fragmentation.* In vertical fragmentation, fragments have only projection operations. In this case, the reconstruction of the global collection can be done by joining the fragments [Andrade et al. 2006]. Let GC be a global collection, and $VF_1, VF_2, ..., VF_n$ its vertical fragments. According to [Andrade et al. 2006], $GC = VF_1 \bowtie VF_2 \bowtie ... \bowtie VF_n$, where $\bowtie$ denotes join of two trees. Note that $\bowtie$ is the join operation of TLC [Paparizos et al. 2004].

Vertical fragmentation requires additional work in the localization 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 cannot be discarded. It can be removed otherwise. This procedure is stated as follows. Let GC be a global collection and $H = \{VF_1, VF_2, ..., VF_n\}$ the set of its vertical fragments. Let $A = \{A_1, A_2, ..., A_m\}$ be the set of all possible subtrees in GC. By definition, each vertical fragment VF_i contains a projection (π) 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'.

PROPOSITION 3.2. *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 $\pi_P(VF_i) = \emptyset$. This projection is empty if the set of sub-trees P has no sub-tree in A'.*

PROOF SKETCH. By definition, $VF_i = \pi_{A'}(CG)$. Thus, when $\pi_P(VF_i) = \emptyset$, then $P \bigcap A' = \emptyset$. Consequently, VF_i does not contribute to answering q and can be eliminated from Q'.

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 depends 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. Let GC be a global collection and $YF_1, YF_2, ..., YF_n$ its hybrid fragments, 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. If 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.

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 Section 3.2.2, 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 Section 3.2.2. For each fragment YF_j defined over selection and projection operations, we must find its sibling fragments, that is, fragments that 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 Section 3.2.1. 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 that are 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 generating an algebraic plan of minimal cost by creating equivalent variations of the algebraic plan obtained in the Localization layer. These variations are generated by algebraic transformations. The minimal cost plan is obtained by using a cost function to choose the best among the alternative plans. The search for the optimal plan can be very expensive to the query processing, which in the end reduces the gains obtained by the optimizations of the algebraic plan. Thus, 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 paper, thus we briefly overview it.

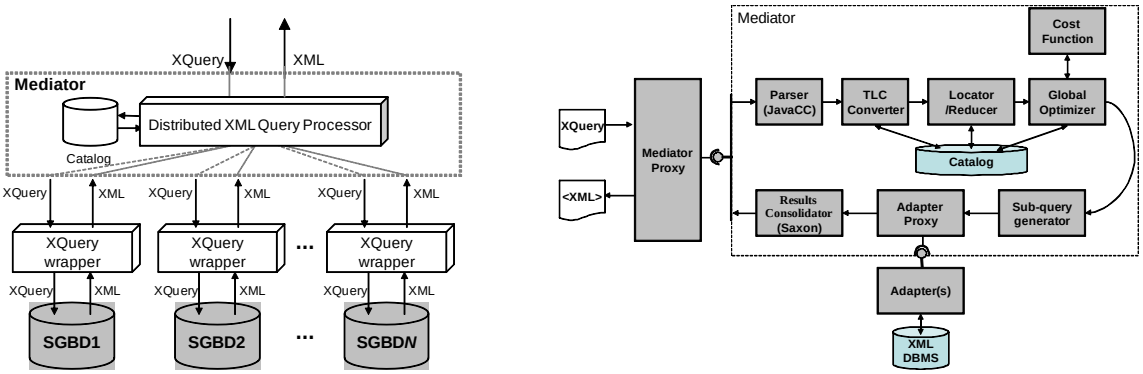
The task of globally optimizing a distributed query starts by generating query plans that are equivalent to the reduced query plan. This can be done by changing the order of the operations in the plan; replacing the localization of fragments (when there are replicas of fragments in different nodes of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

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 [Meier 2003; Fiebig et al. 2002].

4. AN ARCHITECTURE TO OUR METHODOLOGY

To evaluate the technical feasibility of the methodology proposed in Section 3, and to evaluate the performance of query processing in a distributed environment, we propose an architecture as shown



(a) Mediator and Adapters (b) Mediator Components

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

in Fig. 5(a). To compose a global view and also to serve as a unique access point to the system, we use a mediator [Wiederhold 1992]. 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. Our prototype implementation [Figueiredo et al. 2007b] was completely based on the architecture presented in Fig. 5(a). 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. The architecture we propose for the Mediator follows the basic query processing architecture presented in [Kossmann 2000], shown in Fig. 5(b). 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. To implement the parser, we have used the JavaCC with JJTree libraries. These libraries automatically construct 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 an 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. In practice, since we use first order predicates, our implementation mirrors the relational reduction algorithm proposed by [Ozsu and Valduriez 1999]. The Locator 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 of equivalent 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

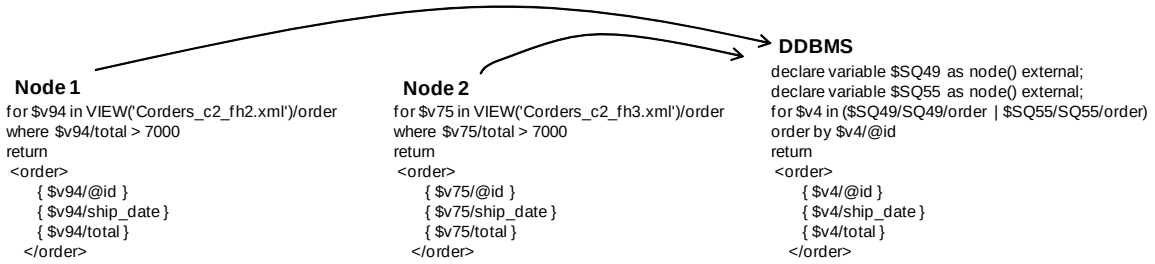


Fig. 6. Sub-queries sent to remote nodes and to the Mediator

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. The XQuery sub-queries are generated using the inverse XQuery/TLC algorithm used in the TLC Converter, 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, 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 6. We use the Saxon XQuery processor[2] 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.

Note that the remote sub-queries on Figure 6 use 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.

4.2 Catalog

The Catalog stores all information needed to process a distributed query. In particular, 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

[2]Saxon is a Java library to process XQuery queries over documents stored in the file system or in main memory.

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

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

a copy of a given fragment (when replication is used), and statistics about fragments (total number of nodes, selectivity characteristics, etc.).

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. We have implemented two types of adapters to execute XQuery sub-queries. One uses eXist [Meier 2003] and the other one uses Saxon. 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, such as eXist, or 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.

5. EXPERIMENTAL EVALUATION

The experimental environment was set up with three computers in a local network (Node 0, Node 1 and Node 2), with the servers totally dedicated to our tests. 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.

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 I and II. The databases we used in these scenarios are CLoja [Andrade et al. 2006] and COrders [Yao et al. 2004]. The queries were defined based on a benchmark [Yao et al. 2004], on related work [Andrade et al. 2006] and some were defined by us to include some queries that would benefit from the fragmentation and others that would not, in order to explore these two situations. Some of the queries perform aggregations, some others use the *order by*, that requires a post processing in the Mediator. The definition of the complete set of queries is available at [Figueiredo et al. 2007a]. Joins are required to reconstruct the documents of the CLoja collection, and Unions for the CStore.

The analysis of the results was done by comparing the average total time of query execution in all of the scenarios through a graph 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 ratio between its average

Table II. Definition of the experimental scenario 3

Scenario	Base	Fragments	Type	Size	Location
3	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 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\}} \bullet \sigma_{/Item/Secao="Perfumaria"} \rangle$	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 := $\langle C_{loja}, \Pi_{/Loja/Itens,\{\}} \bullet \sigma_{/Item/Secao \neq "Brinquedos" \wedge /Item/Secao \neq "Games" \wedge /Item/Secao \neq "Perfumaria" \wedge /Item/Secao \neq "Eletronicos" \wedge /Item/Secao \neq "CD" \wedge /Item/Secao \neq "DVD" \wedge /Item/Secao \neq "Livraria"} \rangle$	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

execution time in scenarios 1, 2 and 3, with the average time in scenario 0. Consequently, the average time in scenario 0 is always equal to 1 in our graphs.

Our first performance evaluation presented bad results that were caused by the communication overhead of Axis Web Services (we have used version 1.3). Web services require packing and unpacking data, and this had a big impact in the performance of the queries. In fact, when comparing the Web Services communication times with the eXist RPC API, we concluded that Web Services are up to 38 times slower. Our implementation, however, uses Web Services due to interoperability issues. Our next focus is on improving the performance of the prototype by using RPC. Thus, in this paper, we show the results without the packing and unpacking Web Services cost, but keeping the communication costs among sites. A more detailed discussion is available at [Figueiredo et al. 2007a].

Fig. 7 shows a performance comparison of queries over a vertically/hybrid fragmented SD collection (CLoja, which is described as CStore in [Andrade et al. 2006]), while Fig. 8 shows the results for queries over a horizontally fragmented MD collection (COrders, from [Yao et al. 2004]). The query names in both Figures follow a convention that aims at making the results interpretation easier. Each

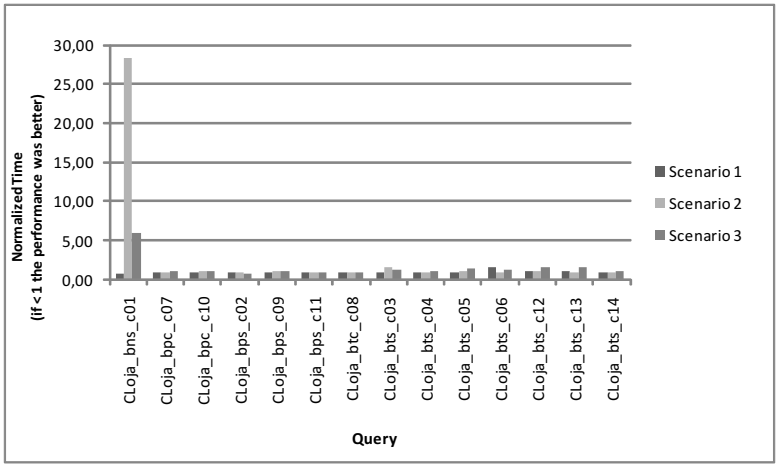


Fig. 7. Comparison of total execution time of queries over a SD collection in scenarios 0 (normalized), 1, 2 and 3

query has its name composed as follows: Collection_Benefit_cSeqNum.xq, where:

—Collection is the name of the collection (CLoja or COrders);

—Benefit is one of the following indicators: *bns* - no benefit from fragmentation; *bnc* - no benefit from fragmentation, with the additional complexity of result aggregation; *bps* - partial benefit from fragmentation; *bpc* - partial benefit from fragmentation, with the additional complexity of result aggregation; *bts* - total benefit from fragmentation; *btc* - total benefit from fragmentation, with the additional complexity of result aggregation.

—SeqNum is the number that identifies the query

For the queries executed over the SD collection (Fig. 7), there were practically no performance gains in the scenarios with fragmentation. Even for those queries that would benefit from fragmentation (those that query a single fragment), the execution time 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 was inferior in the distributed environment in most of the cases.

Some queries over the MD collection (Fig. 8) presented gains in the distributed environment. Some achieved reductions in the order of 95% when compared to the execution time in a centralized environment. The major gains were observed in queries that totally benefit from fragmentation and uses aggregation (COrders_btc_c14 and COrders_btc_c16). Queries with fragmentation benefit, but no aggregation obtained gains between 10 and 40%. For queries that benefit from the fragmentation, Scenario 3 achieved the best results. The average reduction in the execution time in Scenario 3 was 62% when compared with the centralized environment.

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 an MD collection, the database has to parse all documents to process a query, and this increases the query execution time. For this reason, an SD collection with hybrid fragments will also have performance gains in the distributed environment.

6. RELATED WORK

The hierarchical and semi-structured nature of XML poses some difficulties in dealing with distributed queries. Some work in literature [Suciu 2002; Re et al. 2004; Kling et al. 2009] deals with query pro-

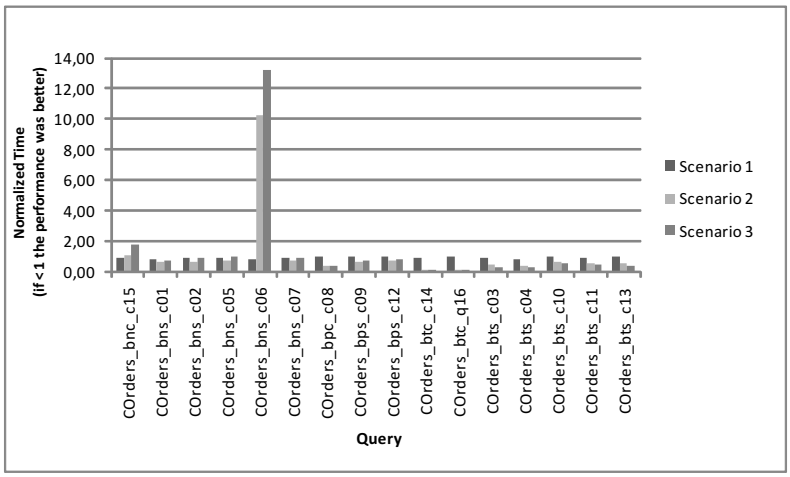


Fig. 8. Total query execution time on an MD collection in scenarios 0 (normalized), 1, 2 and 3

cessing over distributed XML data. Other works are based on integration of heterogeneous distributed databases using XML as a common standard between the heterogeneous data sources, and a mediator [Gardarin et al. 2002; Baru et al. 1999; Lee et al. 2002]. Such works do not deal with fragmented databases.

Gertz and Bremmer [Gertz and Bremer 2003] present a solution to 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, and reduces the query plan by eliminating irrelevant fragments. In addition we present a non intrusive architecture design. The index structures of [Gertz and Bremer 2003] could be used, complementarily, to improve the performance in our approach.

Kling, Ozsu and Daudjee [Kling et al. 2009] are also concerned with distributed query processing over homogeneous XML documents. They focus on data localization and pruning. However, they do not propose a query processing methodology, nor deal with hybrid fragmentation. The steps they comprise cover only part of the steps of our methodology. Specifically, they are able to generate the sub-queries over the fragmented database and prune irrelevant fragments out of the query plan. Optimization is not addressed. 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 efficiently used on top of pre-existing native XML DBMS applications.

7. FINAL REMARKS

This paper has shown a solution to transparent 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 formal reconstruction rules of the original XML collection from its fragments.

Our solution builds upon traditional and successful distributed query processing techniques for relational databases [Ozsu and Valduriez 1999]. We make an analogy of the relational model with the semi-structured model so we can take advantage of relational 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. 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. Using TLC we could easily move from XQuery to the pattern tree and back to XQuery. This allowed us to provide a non-intrusive solution, where sub-queries can be readily executed by any XQuery DBMS.

We have implemented prototypes of the Mediator and Adapters by using the eXist native XML database [Meier 2003]. The implementation was totally based on the rules and definitions shown in this paper, with the goal of showing the efficiency of our proposal. The prototype was implemented using only Open Source components. It was developed in Java, and thus is platform independent. Our implementation is very similar to ParGRES [Mattoso et al. 2005], which follows a database cluster approach to efficiently process OLAP SQL queries in parallel. ParGRES implements a methodology to process distributed queries over a virtually partitioned database. In fact, our implementation can be adapted to work with virtual adaptative fragmentation following the dynamic principles of ParGRES.

Experiments were conducted using the three types of fragmentation: horizontal, vertical and hybrid. The correctness of our approach builds on the algebraic properties [Paparizos et al. 2004] of fragmentation definitions, and reconstruction rules from [Andrade et al. 2006]. Our experiments have shown that our solution can achieve performance improvements of up to 95% when compared to the centralized environment, and average of 62% for queries that benefit from 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, up to 10 times slower in our experiments, because of the additional processing in the Mediator to process the intermediate results. The experiments have also shown that using Web Services in the interface of the components of our architecture compromised the query performance due to the time spent with packing and unpacking. On the other hand, using 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 a bookstore branches.

Our experiments also show that the distributed design of an 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. Replicating some collections without fragmentation can be part of the design. We intend to use these results to build a methodology to design distributed XML databases, similar to the existing ones for OO [Baião et al. 2004] and relational [Ozsu and Valduriez 1999] models.

Additionally, indexing approaches [Gertz and Bremer 2003] could be used to further improve performance of queries in our approach. Performance improvements are our next step.

REFERENCES

Abiteboul, S. On views and XML. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, Philadelphia, United States, pp. 1–9, 1999.

Abiteboul, S., Benjelloun, O., and Milo, T. The Active XML project: an overview. *The VLDB Journal* 17 (5): 1019–1040, 2008.

Aguilera, V., Cluet, S., Milo, T., Veltri, P., and Vodislav, D. Views in a large-scale XML repository. *The VLDB Journal* 11 (3): 238–255, 2002.

Andrade, A., Ruberg, G., BaiÃčo, F., Braganholo, V., and Mattoso, M. PartiX: processing XQuery queries over fragmented XML repositories. Technical Report ES-691, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 2005.

Andrade, A., Ruberg, G., Baião, F., Braganholo, V., and Mattoso, M. Efficiently processing XML queries over fragmented repositories with PartiX. In *International Workshop on Database Technologies for Handling XML Information on the Web*. Munich, Germany, pp. 150–163, 2006.

Baião, F., Mattoso, M., and Zaverucha, G. A distribution design methodology for object DBMS. *Distributed Parallel Databases* 16 (1): 45–90, 2004.

Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., and Chu, V. XML-based information mediation with MIX. *SIGMOD Record* 28 (2): 597–599, 1999.

Bremer, J. and Gertz, M. On distributing XML repositories. In *Proceedings of the Workshop on Web and Databases*. San Diego, United States, pp. 73–78, 2003.

Ferraz, C. A., Braganholo, V., and Mattoso, M. ARAXA: Storing and Managing Active XML documents. *Web Semantics: Science, Services and Agents on the World Wide Web* vol. 8, pp. 209–224, 2010.

Fiebig, T., Helmer, S., Kanne, C., Moerkotte, G., Neumann, J., Schiele, R., and Westmann, T. Anatomy of a native XML base management system. *The VLDB Journal* 11 (4): 292–314, 2002.

Figueiredo, G., Braganholo, V., and Mattoso, M. A methodology for query processing over distributed XML databases. Technical Report ES-710/07, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, 2007a.

Figueiredo, G., Braganholo, V., and Mattoso, M. Um mediador para o processamento de consultas sobre bases xml distribuídas. In *Proceedings of the Demos Session of the Brazilian Symposium on Databases*. João Pessoa, Brazil, pp. 21–26, 2007b.

Gardarin, G., Mensch, A., Dang-Ngoc, T., and Smit, L. Integrating heterogeneous data sources with XML and XQuery. In *Proceedings of the International Conference on Database and Expert Systems Applications*. IEEE Computer Society, Aix en Provence, France, pp. 839–846, 2002.

Gertz, M. and Bremer, J. Distributed XML repositories: Top-down design and transparent query processing. Technical Report T.R.CSE-2003-20, Department of Computer Science, 2003.

Ives, Z. G., Halevy, A., and Weld, D. An XML query engine for network-bound data. *The VLDB Journal* 11 (4): 380–402, 2002.

Jagadish, H. V., Lakshmanan, L. V. S., Srivastava, D., and Thompson, K. TAX: a tree algebra for XML. In *International Workshop on Database Programming Languages*. Frascati, Italy, pp. 149–164, 2001.

Kling, P., Ozsu, T., and Daudjee, K. Optimizing distributed XML queries through localization and pruning. Tech. Rep. CS-2009-13, University of Waterloo, 2009.

Kossmann, D. The state of the art in distributed query processing. *ACM Computing Surveys* 32 (4): 422–469, 2000.

Kurita, H., Hatano, K., Miyazaki, J., and Uemura, S. Efficient Query Processing for Large XML Data in Distributed Environments. In *Proceedings of the International Conference on Advanced Networking and Applications*. IEEE Computer Society, Washington, USA, pp. 317–322, 2007.

Lee, K., Min, J., and Park, K. A design and implementation of XML-Based mediation framework (XMF) for integration of internet information resources. In *Proceedings of the Hawaii International Conference on System Sciences*. Vol. 7. IEEE Computer Society, Big Island, Hawaii, United States, pp. 202–202, 2002.

Ma, H. and Schewe, K. Fragmentation of XML documents. In *Proceedings of the Brazilian Symposium on Databases*. Manaus, Brazil, pp. 200–214, 2003.

Mattoso, M., Zimbrão, G., Baião, F., Braganholo, V., Aveleda, A., Miranda, B., Almentero, B., and Costa, M. ParGRES: Middleware para Processamento Paralelo de Consultas OLAP em Clusters de Banco de Dados. In *Proceedings of the Demos Session of the Brazilian Symposium on Databases*. Uberlandia, Brazil, pp. 19–24, 2005.

Meier, W. eXist: an open source native XML database. *Web, Web-Services, and Database Systems* vol. LNCS 2593, pp. 169—183, 2003.

Ozsu, M. T. and Valduriez, P. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

Paparizos, S., Wu, Y., Lakshmanan, L. V. S., and Jagadish, H. V. Tree logical classes for efficient evaluation of XQuery. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. ACM, Paris, France, pp. 71–82, 2004.

Re, C., Brinkley, J., Hinshaw, K., and Suciu, D. Distributed XQuery. In *Proceedings of the Workshop on Information Integration on the Web*. Toronto, Canada, pp. 116–121, 2004.

Suciu, D. Distributed query evaluation on semistructured data. *ACM Transactions on Database Systems* 27 (1): 1–62, 2002.

Wiederhold, G. Mediators in the architecture of future information systems. *IEEE Computer* 25 (3): 38–49, 1992.

Wu, Y., Paparizos, S., and Jagadish, H. V. Querying XML in Timber. *IEEE Data Engineering Bulletin* 31 (4): 15–24, 2008.

Yao, B., Ozsu, M., and Khandelwal, N. XBench benchmark and performance testing of XML DBMSs. In *Proceedings of the IEEE International Conference on Data Engineering*. Boston, United States, pp. 621–632, 2004.