IF : 3.62 | IC Value 70.36

Volume-5, Issue-5, May - 2016 • ISSN No 2277 - 8160

Research Paper

Statistics

Query Processing System and Xml Query Operation

Kanchan Kumari	Department of Statistic and Computer Applications, T. M. B. U. Bhagalpur.
Brajnandan Kumar	Department of Statistic and Computer Applications, T. M. B. U. Bhagalpur.
B.K. Das	Department of Statistic and Computer Applications, T. M. B. U. Bhagalpur.

ABSTRACT

XMLis also known for the excessive information redundancy in its representation. There are several steps involved in this method; however, we have focused particularly on XML data partitioning and dynamic relocation of partitioned XML data in our research work. Since the efficiency of query processing depends on both XML data size and its structure, these factors should be considered when XML data is partitioned. A general way to publish relational data as XML is to provide XML views over relational data, and allow business partners to query these views using an XML query language. In the present paper, we have described the operation in query processing over XML and different operational steps. In this paper, we also have presented XML through the process to create the query and all processes down by the XML operation steps and then obtained the output as result.

KEYWORDS :

1 INTRODUCTION:

XML languages, such as XQuery, XSLT and SQL/XML, employ XPath as the search and extraction language. XPath expressions often de¯ne complicated navigation, resulting in ex-pensive query processing, especially when ex-ecuted over large collections of documents. In this paper, we propose a framework for ex-ploiting materialized XPath views to expe-dite processing of XML queries. We explore a class of materialized XPath views, which may contain XML fragments, typed data values, full paths, node references or any combination thereof. We develop an XPath matching algo-rithm to determine when such views can be used to answer a user query containing XPath expressions.An *XML document* can be seen as a rooted, ordered, la-belled tree, where each node corresponds to an element or a value, and the edges represent (direct) element-subelement or element-value relationships. The ordering of sibling nodes (children of the same parent node) implicitly de nes a total order on the nodes in a tree, obtained by traversing the tree nodes in preorder. An XML database can be viewed as an XML document, once a dummy root node has been added to convert the forest into a tree.

A general and flexible way to publish relational data as XML is to create (possibly many) XML views of the underlying relational data. Each of these XML views can provide an alternative, application-specific view of the underlying relational data. Through these XML views, business partners (and other XML application developers) can access existing relational data as though it was in some industry-standard XML format. One simple solution is to materialize the entire XML view on request and return the resulting XML document. The main problem with this approach is that, in many cases, applications do not require the whole view to be materialized. For example, in an XML view of available items, a business partner may only be interested in a particular item. Materializing the availability of all the items would be wasteful in this case because it wouldThe query processing cost as well as storage cost of XML data is dependent on the data size, so that we should evenly distribute partitioned data among computation nodes. If the partitioned data can be distributed in such a way, query processing cost is distributed among the multiple computation nodes.
The database community utilizes inverted list filtering, since the problem is so similar to that addressed in structured information retrieval applications. In addition to inverted list filtering, XML query processing naturally includes navigational access to XML data. Such access is similar to *Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.* There are queries for which structural navigation followed by inverted list filtering is best. This suggests that a native XML

repository needs to support query plans that utilize these query processing approaches, and needs to be able to pipe intermediate results between the two. Finally, given that no one style of processing dominates, an XML query processor requires query optimization techniques and statistics to decide how to choose among the alternatives for any given query results which provides significant memory savings and al-lows to join larger relations at the cost of more intensive CPU usage and some loss in response time. We present experimental results showing that our optimizations give significant effect as compared with a straightforward implementation. We also demonstrate that the algorithm is superior to some other algorithms for set-containment joins.

2 FUNDAMENTAL XML & ELEMENTS, DTD:

An XML document is usually modeled as a rooted, ordered, labeled tree. Each node in the tree corresponds to an ele-ment or a value, and the edges represent immediate element-sub element or element-value structural relationships. When considering structural XML queries, one has to quickly de-termine parent-child and ancestor-descendant relations for a given set of tree nodes. This is achieved by using special la-beling schemes that capture such relationships between node pairs without having to traverse the path to each node. An important feature of a numbering scheme for multiversioned XML documents, is how the scheme adapts to changes. We thus considered various such schemes that are describe we discuss previous approaches on maintaining and querying multiversion XML documents.

Mikael Fernanduzet al.,(2000)XML serves dual functionalities as markup language and data format. It separates presentation and data thus offering independency and flexibility for content association. Due to this nature of flexibility, data interchanged between two very different systems can use XML as the data format. XML tree-like structure is intuitive, human readable, and easy to understand. With the help of XML schema or DTD, the type and attributes of each tag usable for certain XML document can be well defined. The eXtensible Markup Language (XML) is a hierarchial format for data representation and exchange. An XML document consists of nested XML elements starting with root element. Each element can have attributes and text values, in addition to nested sub-elements.

3 FULL TEXT SEARCH QUERY SEMANTICS:

Full-text search queries concerned in this paper are composed of keywords and four operations conjunction, disjunction, proximity and order. Conjunction and disjunction operations combine several (at least 2) sub-queries into a single query. Proximity and order operations are applied to a single query to produce another query.

4 QUERY PROCESSING ARCHITECTURE:

Jayavel Shanmugasundara et al.,(2001) present our high-level query-processing architecture. We begin by illustrating how XML views are created and queried in XPERANTO. As a starting point, XPERANTO automatically creates a *default XML view,* which is a low-level XML view of the underlying relational database. Users can then define their own views on top of the default view using XQuery. Moreover, views can be defined on top of views to achieve higher levels of abstraction. The main advantage of this approach is that a standard XML query language is used to create and query views. This is in contrast to approaches such as j.cheang et al.,(2000) and m.femanez et al., (1999) where a proprietary language is used to define the initial XML view of the underlying relational database. The default XML view for a simple purchase-order database. As shown, the database consists of three tables, one table to keep track of customer orders, a second table to keep track of the

items associated with an order, and a third table to keep track of the payments due for each order. Items and payments are related to orders by an order id (oid). In the default XML view, top-level elements correspond to tables with table names appearing as tags.

4.1 P2P Systems

- Peer-to-peer (P2P) systems adopt a completely decentralized approach to resource management. By distributing data storage, processing and bandwidth across all peers in the network, they can scale without the need for powerful servers. P2P systems have been successfully used for sharing computation, e.g. SETI@home [m.slopiro et al.,(2004)], communication [ICQ. http://www.icq.com/.] or data, e.g. Gnutella [Gnutella. http://www.gnutelliums.com/.] and Kaaza [JXTA. http://www.jxta.org/.]. The success of P2P systems is due to many potential benefits: scale-up to very large numbers of peers, dynamic self-organization, load balancing, parallel processing, and fault-tolerance through massive replication. Furthermore, they can be very useful in the context of mobile or pervasive computing. However, existing systems are limited to simple

❖ ***Some important theme that's through we are know how work p2p system and (my contribution)***

- **Processing system Autonomy**: processing system autonomous peer should be able to join or leave the system atany time without restriction. It should also be able to control the data it stores and which other peers can store its data, e.g. some other trusted peers. **Xml Query processing expressiveness**: the xml query processing language should allow the user to describethe desired data at the appropriate level of detail. The simplest form of query is key look-up which is only appropriate for finding files. Keyword search with ranking of results is appropriate for searching documents. But for more structured data, an SQL-like query language is necessary.

- **Efficiency power**: this through we are get the fundamental resorcesefficient use of the P2P system resources (bandwidth,computing power, storage) should result in lower cost and thus higher throughput of queries, i.e. a higher number of queries can be processed by the P2P system in a given time.

- **Working purity**: depend upon working purity and query processing and check the query by the system and we are get the output and refers to the user-perceived efficiency of the system, e.g.completeness of query results, data consistency, data availability, query response time, etc.

- **Problem analysis**: analysis areiniseal for query processing so, efficiency and quality of services should be provided despitethe occurrence of peers' failures. Given the dynamic nature of peers which may leave or fail at any time, the only solution is to rely on data replication.

System Security:security vital roll playin the query processing time the open nature of a P2P system makes security a major challengesince one cannot rely on trusted servers. Wrt. data management, the main security issue is access control which includes enforcing intellectual property rights on data contents.

5 VARIOUS XML OPERATION:

5.1 Multi-data source definition language

The purpose of MDL is to define the multi-data source schema of a peer starting from a set of data sources schemas. A multi-data source schema is a collection of data sources' schemas or multi-data sources. The following example shows the schema of a multi-data source owned by peer *Sg* denoted P_{Sg}. In this example the three peers P_{Bnp}, P_{Cio} and P_{Cl} are called remote peers.

5.2 Multi-data source retrieval language

We present briefly the Multi-data source Retrieval Language (MRL) and we give an example of a query using this language. An MRL query form is defined as follows:

Use (multi-)datasource1 name1 [,(multi-)datasourcejnamej] *
Allow $ <semantic variables>

(E)XQuery query Close
name1 [,namej] *

The clauses *Use*, *XQuery query* and *Close* are mandatory in MRL Queries, whereas the clause *Allow* is optional. The clause *Use* determines the scope of the query and connects to datasources for processing whilst the clause *Close* disconnects from data sources. The *name* is a given alias for either a data source or multi-data source and the clause *Allow* permits the declaration of semantic variables. Through these variables, the user declares his/her intention to access data, in a given query, that are semantically similar and differently named. The (E)XQuery query can be formulated like a query w.r.t. to XQuery language or as an EXQuery query that allows an active data source to be called.

Example 2 (One semantic variable): Select in$MSch_{sg}$the name of the branches in thetwo data sources *Bnp* and *Cio*.

Q: *use bnpb,cio callow*
$x=nom,name

for $a in document("MSch$_{sg}$")/bank/bank1/bank2, $b in $a/*/$x
return <result>$b/text()</result>
closeb,c

6 QUERY PROCESSING SYSTEM & MY CONTRIBUTION:

Query processing system (over xml query operation)there are six type operation perform in query processing system.

Step 1→client side query input

Client side query create by the then it required all information it is form of query and here input the instruction set of program and with set of rules.

Step 2→query validation and operation and progress translation

Query validation that through we are checked the all instruction syntax through the system and translate it.

Step 3→query optimizer and execution time

Query execution time is very impotent time that is query optimizer and execution plan successfully down.

Step 4→query code generate operation

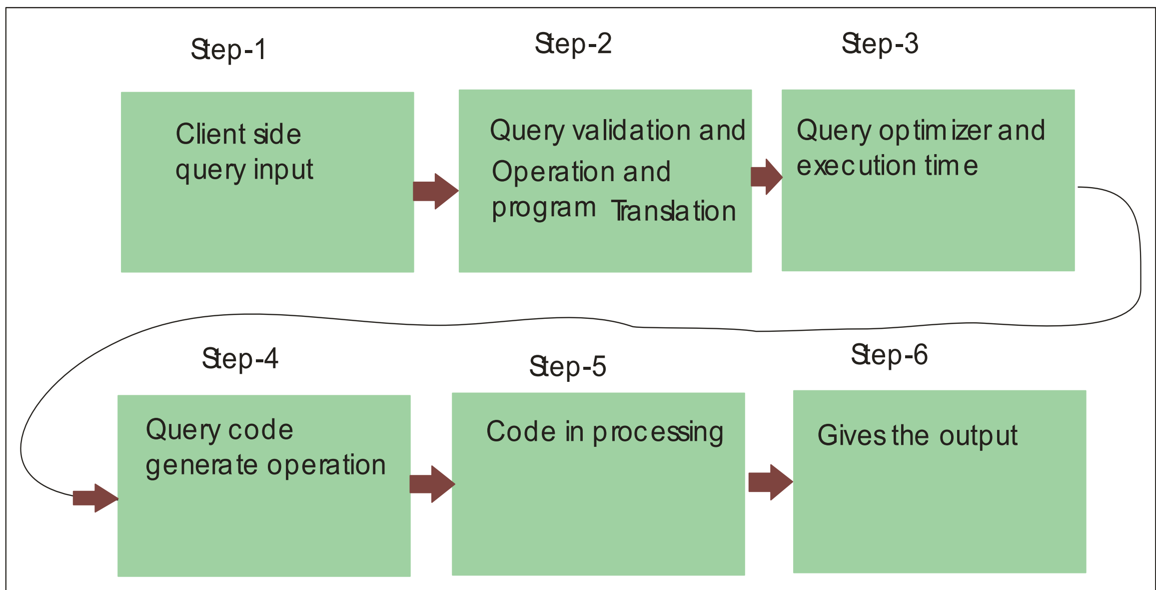
Codes aregeneratingin this operation and promote the next step easily.

Step 5 ➜ code in processing

In this process code ready to go in processing.

Step 6 ➜ gives the output

7 ATUAL OUTPUT/RESULT OF QUERY:



1.1 XML query operation steps

8 VIEW OF XML:

8.1 Multiple Views XML

Now we only considered using a single view and a single compensation root to construct compensation. However, a query can usingmultiple views. Similarly the same view could potentially be used multiple times, if the view extraction point mapped to multiple query nodes.We construct compensation plan using the fol-lowing four-step algorithm, which takes as input a set of compensation roots produced by matching one or more views into the query.

- Find an XPS node in the query tree, that is a low-est common ancestor (LCA) of all compensation roots.

For each compensation root q_i , construct an XPath expression Q_i that starts at the compensa-tion root and traverses upward to the LCA node. The Q_i includes local predicates of q_i if the cor-responding view contains *data* extraction.

- Optimize each Q_i, as if it was a compensation, using the algorithm of Section 4.2; construct an intersection of all Q_i expressions Construct the compensation expression with $\bigcap Q_i$ as the view and LCA as the compensation root.

Note that every view involved in the plan, has to store node references to facilitate upward traversal from the compensation roots to the LCA node

EXAMPLE 4.3 Consider *Q* = ==*order*[@*date* > "*Jan* 1*;* 2004" *and lineitem=@price* > 100]=*number*, and a view *V* = ==@¤ with *data; path*, and *reference*extractions. The view maps into the query in two different ways. The two compensation roots are @*date* and @*price*. Thus the LCA node is ==*order*.

In the second step we construct expressions Q_1 and Q_2which start at the compensation roots and nav-igate to ==*order*. Q_1 = *self* :: *attribute*(*date*)[*:* > "*Jan* 1*;* 2004"]=*parent* :: *order*. Q_2 = *self* :: *attribute*(*price*)[*:* >100]=*parent* ::*lineitem=parent* :: *order*. Both $Q_1$and Q_2contain local predicates onthe compensation roots, because *V* can answer these predicates directly using the *data* extraction.

The Q_1 and Q_2 are optimized into the expressions $P_1$and P_2, shown below, using the three types of ex-tractions stored in the view.

P_1= ¼*reference=parent*::¤(
¾*data*>"*Jan* 1*;*2004"^*path»==order=@date*(*V*))
P_2= ¼*reference=parent*::¤=*parent*::¤(
¾*data*>100^*path»==order=lineitem=@price*(*V*))

Finally, the compensation expression is computed using ==*order* (LCA) as the compensation root and Q_1 \ Q_2as the view. The resulting plan is:
¼*self*::¤=*n*
umber(P_1 \ P_2) 2

Note that the above algorithm provides only one of many ways to construct compensation from multiple materialized views. For some queries and some datasets it might make sense to apply a portion of the compensation before the structural join on the LCA. We are currently investigating cost-based optimization of compensation expressions.

9 VIEW COMPOSITION:

XML views with nested sub-elements are computed Jayavel Shanmugasundaram et al.,(2001) from flat relational tables. Navigational operations expressed as path expressions in XQuery queries traverse these nested structures to extract sub-elements and their attributes. Therefore, the query operators that traverse nested structures effectively invert the query operators that create them in a view. Navigational operations can thus be eliminated by undoing the construction of the corresponding elements. Our view composition module performs this query simplification.

Removing all XML navigation operations offers several performance benefits. The obvious benefit is that the construction of intermediate XML.

9.1. Composition Rules

These represent the functions that capture all the navigation operations in an XQuery query defines twelve composition rules that can be used to eliminate all occurrences of these navigational functions. These rules are complete in the sense that they specify how *all* occurrences of navigational functions can be eliminated. This is done by specifying a composition rule *for every possible input* to a navigational function, which specifies how the navigational function is to be removed for the given input. We now describe the composition rules in detail.

The *getTagName* function when applied to the *cr8Elem* can be reduced to the first argument of the *cr8Elem* function (which is the tag name of the created element). Rules 2-5 are defined in a similar manner. Rules 6 and 7 replace the *isElement* function by *true* or *false*, depending on whether the input is anelement or not. Rules 8 and 9 are defined similarly. Rule 10 composes the *unnest* function with the *aggXMLFrags* function by simply returning the input to *aggXMLFrags* without performing any aggregation. Rule 11 composes the *unnest* function with the *cr8XMLFragList* function by reducing the *unnest* function to a union of all the arguments of the *cr8XMLFragList* function. Rule 12 is defined similarly

9.2 Applying Composition Rules

We eliminate all navigational operations by repeated application of the composition rules. This step is complemented by a number of other query rewrite transformations that push down predicates, and remove unreferenced columns and operators.

10 COMPUTATION PUSHDOWN:

The goal in this phase of query processing is to push all data and memory intensive operations down to the relational engine as an efficient SQL query. We describe two query processing techniques that make this possible.

10.1. Query Decorrelation

we showed how complex expressions in XQuery are represented using correlations. However, it has been shown in earlier work that executing XML queries as correlated queries over a relational database leads to poor performanceShanmugasundaram, et. al., (2000)We thus present query decorrelation P. Seshadri, H. Pirahesh et al.,(1996) as a necessary step for efficient XML query execution.

11 QUERY PROCESSING TIME:

11.1 Query Time

We distinguish between ordered and unordered matches. The subsequence based methods need to run all possible orderings of an unordered twig query so as to find all un-ordered matches.

11.1.1 *Ordered Matches*

The execution time (note the logarith-mic scale) of Q_1–Q_5 for the three version ranges, respec-tively. The single and interval (5 or 20 consecutive) ver-sions were chosen randomly within the document's evolu-tion. The Snapshot approach could be implemented using either LCS or Twigstack. In these figures we report the Twigstack implementation (effectively, for each version, the snapshot of each element list is stored and accessed by the original Twigstack algorithm).

For a single version query, the Snapshot approach performs well. However, as the ver-sion range increases the performance quickly degrades; this is because results from each document version are first col-lected and then merged. While the Log-Based method pro-vided the minimal amount of storage space, its query run-times are too expensive regardless of version range. This is attributed to the overhead incurred when having to recreate the document for a given version.

TLCS is consistently worse than both the MVBT-Twigstack and MVBT-LCS approaches. This is because the MVBT-based approaches focus the algorithms to the nodes valid for the version(s) in the query. Instead, the TLCS has to parse the overall document sequence (including nodes that are not related to the query version(s)) before it creates the docu-ment sequence needed for the query. The TLCS process-ing time is unaffected by the size of the version range since the processing is eff ectively the same. In contrast, MVBT-Twigstack and MVBT-LCS processing increases as the ver-sion range increases since the number of nodes accessed by either MVBT-Twigstack or MVBT-LCS also increases.

Among the MVBT based approaches, the performance of MVBT-LCS is faster than MVBT-Twigstack. This obser-vation is similar to what has been reported for ordered twig queries in the non-versioned environment [14]. Among the various queries, MVBT-LCS shows the smaller runtime for queries Q_1 and Q_5; this is to be expected since these queries have the lowest selectivity.

The support of versions through the use of the MVBT has a relatively small overhead on the traditional query processing algorithms. This can be seen in Figure 5, when comparing the Snapshot approach with MVBT-Twigstack for a single version

query. The Snapshot approach used Twigstack on the stored version of the document, while MVBT-Twigstack has the overhead of using the MVBT. We observed the same when comparing LCS-TRIM with MVBT-LCS for a single version [3].

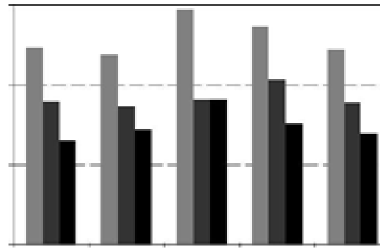


Figure 8: Unordered: Single Version

11.1.2 *Unordered Matches*

Figures 8-10 depict the execution times of $Q_1 - Q_5$ when unordered matches are desired. For simplicity the graphs show only the TLCS, MVBT-Twigstack and MVBT-LCS. The MVBT-Twigstack behavior is similar with the ordered case, since the Twigstack approach can easily do both or-dered and unordered matchings. Instead, the LCS based approaches create an ordered sequence for each twig query. As a result, all configurations of a query must be processed for finding the unordered matchings (more twigneed to processed per query). As expected the runtime of the LCS approaches increases

12 MULTIPLE QUERY PROCESSING OVER XML:

In this section, we present our high-level query-processing architecture. We begin by illustrating how XML views are created and queried in XPERANTO.

As a starting point, XPERANTO automatically creates a *default XML view,* which is a low-level XML view of the underlying relational database. Users can then define their own views on top of the default view using XQuery. Moreover, views can be defined on top of views to achieve higher levels of abstraction. The main advantage of this approach is that a standard XML query language is used to create and query views. This is in contrast to approaches where a proprietary language is used to define the initial XML view of the underlying relational database.
The default XML view for a simple purchase-order database. As shown, the database consists of three tables, one table to keep track of customer orders, a second table to keep track of the items associated with an order, and a third table to keep track of the payments due for each order. Items and payments are related to orders by an order id (oid).

13 CONCLUSION:

In this paper, we are present work p2p system. We then pro-pose a new p2p system work and first, this system have important theme that's through stating to ending process are satisfied and fully work doun and my second, work is xml query processing system (over xml query operation)and six steps operation are work perform and this steps through we get the After processing gives the actual output/result of query. So, it's very important for Peer-to-peer (P2P) systems adopt a completely decentralized approach to resource management. By distributing data storage, processing and bandwidth across all peers in the network, they can scale without the need for powerful servers and next Query processing system (over xml query operation)that's through easy to query processing system over XML.

REFERENCES:

1. M. Fernandez, A. Morishima, D. Suciu, "Efficient Evaluation of XML Middleware Queries", SIGMOD Conf., Santa Barbara, May 2001, pp. 103-114.
2. L. Lakshmanan, F. Sadri, I. Subramanian, "SchemaSQL – A Language for Interoperatability in Relational Multi-Database Systems", VLDB Conf., Mumbai, India, Sep. 1996, pp. 239-250.
3. J. Shanmugasundaram, et. al., "XPERANTO: Bridging Relational Technology and XML", IBM Research Report, June 2001.
4. J. Cheng, J. Xu, "XML and DB2", ICDE Conf., San Diego, March 2000, pp. 569-573.
5. M. Fernandez. W. Tan, D. Suciu, "SilkRoute: Trading Between Relations and XML", World Wide Web Conf., Toronto, Canada, Ma
6. Microsoft Corp. http://www.microsoft.com/XML.
7. R. Huebsch et al. Querying the Internet with PIER. VLDB Conf., 2003.
8. JXTA. http://www.
9. .A. Levy, A. Rajaraman, J. Ordille. Querying heterogeneous information sources using source descriptions. VLDB Conf.,1996.jxta.org/.
10. M. Shapiro. A simple framework for understanding consistency with partial replication. Technical Report, Microsoft Research, 2004.
11. J. Shanmugasundaram, et. al., "XPERANTO: Bridging Relational Technology and XML", IBM Research Report, June 2001.
12. P. Seshadri, H. Pirahesh, C. Leung, "Complex Query Decorrelation", ICDE Conf., New Orleans,
13. J. Shanmugasundaram, et. al., "Efficiently Publishing Relational Data as XML Documents", VLDB Conf., Cairo, Egypt, Sep. 2000, pp. 65-76.
14. .P. Seshadri, H. Pirahesh, C. Leung, "Complex Query Decorrelation", ICDE Conf., New Orleans, Louisiana, March 1996, pp. 450-458.