Effective Schema-Based XML Query Optimization Techniques

Guoren Wang and Mengchi Liu
School of Computer Science
Carleton University, Canada
{wanggr, mengchi}@scs.carleton.ca

Jeffrey Xu Yu
Department of SEEM
CUHK, Hong Kong, China
yu@se.cuhk.edu.hk

Bing Sun, Ge Yu, and Jianhua Lv
Department of Computer Science
Northeastern University, China
{sunbing,yuge,dbgroup}@mail.neu.edu.cn

Hongjun Lu
Department of Computer Science
HKUST, Hong Kong, China
luhj@cs.ust.hk

Abstract

Use of path expressions is a common feature in most XML query languages, and many evaluation methods for path expression queries have been proposed recently. However, there are few researches on the issue of optimizing regular path expression queries. In this paper, two kinds of path expression optimization principles are proposed, named path shortening *and* path complementing*, respectively. The* path shortening *principle reduces the querying cost by shortening the path expressions with the knowledge of XML schema. While the* path complementing *principle substitutes the user queries with the equivalent lower-cost path expressions. The experimental results show that these two techniques can largely improve the performance of path expression query processing.*

1 Introduction

Although XML is usually used as an information exchange standard, storing, indexing and querying XML data are still important issues and recently have become research hotspots in the database community. To retrieve XML data from databases, many query languages have been proposed, such as *XQuery* [4], *XPath* [2], *XML-QL* [3], XML-RL [7], and *Lorel* [1]. Because the use of regular path expressions (RPE) is a common feature of these languages, query rewriting and optimization for RPE is becoming a research hotspot and a few research results have been published recently [9, 12].

Most implemented XML systems are based on relational DBs. In the relational way [5, 12], XML data is mapped into tables of a relational/object-relational database system and queries on XML data are translated into SQL statements. One of the ways to evaluate RPE in relational XML management systems is rewriting an RPE to some SPEs (Simple Path Expression) according to the schema information and statistic information on XML documents, and then computing the SPEs respectively by translating the SPEs into SQL statements and executing them. Take *VXMLR*[12] for example, RPE queries containing '//' and '*' are rewritten to SPEs, which will be translated into SQL statements for a later processing in accordance with relational database schema. In the *Lore* system[1], there are three strategies to compute SPE queries: *top-down*, *bottom-up* and *hybrid*. In the *top-down* way, an XML document tree (DOM tree) is navigated from its root to get proper results; and in the *bottom-up* way, an XML document tree is traversed from its leaves to the root with the help of value indices; finally, in the *hybrid* way, a long path is divided into several short paths, each of which can be evaluated in either the *top-down* way or the *bottom-up* way, and then the results of these short paths are joined together to get proper results.

As an improvement of the *hybrid* way, the *extent join* algorithm [9] has been paid a lot of attention recently. This paper introduces two optimization strategies based on the *extent join* algorithm: *path shortening* Strategy and *path complementing* Strategy. These strategies can also be used for other path evaluation algorithms besides the *extent join* algorithm. The *path shortening* Strategy shortens the path to reduce the number of join operations so that the query performance can be improved. The *path complementing* Strategy computes the original path expression query through some other low-cost path expressions.

2 Background

An XML document can be represented as a rooted tree, $T_d = (V_d, E_d, L_d)$, called an XML data tree, where V_d

is a set of nodes, each of which represents either an *element* or an *attribute*; L_d is a set of node labels representing *tags* of the nodes, each of which is associated with a label; E_d is a set of edges, each of which represents either a parent-child relationship between two element nodes or an element-attribute relationship between an element and an attribute. In the following, we use r_d to represent the root of an XML data tree.

Figure 1 shows an XML data tree for an XML document in the XML benchmark project, *XMark*. Here, an *ellipse* represents an element node, whereas a *triangle* represents an attribute node. The numbers with prefix "&" marked on the nodes are the node identifiers. Note, &1 is the root node of the XML data tree whose label is `site`.

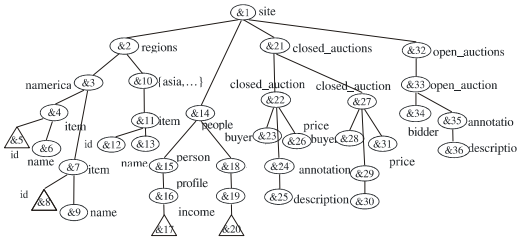

Figure 1. An XML Data Tree

An XML data tree is a set of *paths*. Given a node, a *data path* is a sequence of node identifiers from the root to the node, and a *label path* is a sequence of labels from the root to the node. For example, &1.&2.&3.&4.&6 and &1.&21.&22.&24.&25 are two data paths from the root to the nodes &6 and &25. The corresponding label paths are `/site/regions/namerica/item/name` and `/site/close_auctions/close_auction/annotation/description`, respectively.

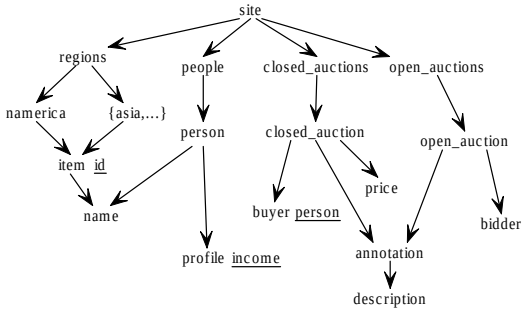

Figure 2. An XML Schema Graph

An XML schema graph is a rooted directed graph, $G_t = (V_t, E_t)$, where V_t is a set of labelled nodes and E_t is a set of edges. We use r_t to denote the root of the XML schema graph. It is important to note that an XML schema graph specifies the possible structures of XML data trees. The label structure of an XML data tree shall be a subgraph of the corresponding XML schema graph. In other words, the label paths imposed by the XML schema graph put restrictions on the possible label paths in the corresponding XML data trees. Figure 2 shows the XML schema graph for the XML data tree in Figure 1. In Figure 2, the root node is the node with a label `site`. The node labels with underline, such as `income` are for attributes.

Given an XML schema graph and an XML database consisting of XML data trees that conform to the XML schema graph, let p be a label path of an XML schema graph $p = /l_1/l_2/\cdots/l_k$, where l_1 is the label of the root node of the XML schema graph, and l_k the ending label of the label path. An extent of the label path p, denoted $ext(p)$, is a set of nodes (or node identifiers) of any data path $/\&n_1/\&n_2/\cdots/\&n_k$ in an XML data tree such that their corresponding label path matches p. In addition, an extent of a label, l_j, denoted $\overline{ext}(l_j)$, is a set of nodes (or node identifiers) that have label l.

Consider a label path of the XML schema graph in Figure 2 p = `/site/regions/namerica/item/name`. The extent of p, $ext(p)$, includes &6 in the XML data tree in Figure 1, because there is a data path &1.&2.&3.&4.&6 such that the corresponding label path of the data path matches p. On the other hand, $\overline{ext}(\texttt{name}) = \{\&6, \&9, \&13\}$.

$$
\begin{array}{l}
PathExpression ::= CONNECTOR\ PathSteps \\
\quad | \ PathSteps\ CONNECTOR\ PathSteps \\
PathSteps ::= Label \\
\quad | \ Label\ '|'\ PathSteps \\
\quad | \ (PathSteps) \\
\quad | \ '*' \\
CONNECTOR ::= '/' \ | \ '//'
\end{array}
$$

Figure 3. BNF Definition of Path Expression

Figure 3 shows the BNF definition of path expressions, as a simplified version of XPath. In Figure 3, *Label* represents node labels. A symbol '*' is introduced as a wildcard for an arbitrary node label in the XML schema graph. The *CONNECTOR* represents the connector of paths. We mainly discuss two widely used connectors: '/' and '//', where the former represents a parent-child relationship, and the latter represents an ancestor-descendant. Given an XML schema graph, G_t, and an XML database consisting of XML data trees that conform to the XML schema. A path expression query, Q, is a path expression (Figure 3). A path expression query, Q, is valid if it at least match a label path, $p = l_1/l_2/\cdots/l_k$, in G_t. A directed query graph $G_q = (V_q, E_q)$ is defined as a rooted subgraph of G_t such that G_q only includes the label paths of G_t that the query Q matches. The root of G_q is the root of G_t and the no-child nodes are the nodes with label l_k. The result of the query,

Q, denoted $ext(Q)$, is $\cup_i\ ext(p_i)$ where p_i is a label path from the root to the no-child node with label l_k in G_q.

3 Simple but Effective Optimization Techniques: Path Shortening and Path Complementing

The path shortening technique is based on the following observation.

Observation 1. *Given an XML schema graph G_t, and let $Q_i = //l_k$ be a valid path expression query, where l_k is a label in the XML schema graph. Then, $ext(Q_i) = \overline{ext}(l_k)$.*

As shown in Figure 1, given a path expression query, `//closed_auction`, $ext($`//closed_auction`$)$ is equal to $\overline{ext}($`closed_auction`$)$ which includes $\{\&22, \&27\}$. It is because that all appearances of the label `closed_auction`, that appear in the XML schema graph (Figure 2), are all included in the corresponding query graph for the query `//closed_auction`. Observation 1 leads to the utilization of an XML indexing mechanism, called XML extent index [9], which indexes all nodes that have the same label, l, e.g., $\overline{ext}(l)$.

Technique 1. (Simple Path Shortening) *Given an XML schema graph G_t and a path expression query $Q = \cdots /l_k$, a corresponding query graph G_q can be constructed ($G_q \subseteq G_t$). All no-child nodes in G_q are labeled with l_k. This query Q can be processed using $\overline{ext}(l_k)$, if and only if l_k does not appear in $G_t - G_q$, that is, l_k only appears in the intersection of two graphs, G_t and G_q.*

As shown in Figure 1, given a path expression query Q_1=`/site/closed_auctions/closed_auction`, $ext(Q_1)$ is equal to $\overline{ext}($`closed_auction`$)$ which includes $\{\&22, \&27\}$. Technique 1 utilizes the XML extent index. As another example, consider the path expression query Q_2 = `/site/closed_auctions/closed_auction/annotation/description`, to access the XML data tree in Figure 1 that conforms to the XML schema graph in Figure 2. The result of Q_2 shall include all nodes that have `description` as theirs label along the label path as specified in Q_2, $ext(Q_2) = \{\&25, \&30\}$. As noted, in this case, $ext(Q_2) \neq \overline{ext}($`description`$)$ because $\overline{ext}($`description`$) = \{\&25, \&30, \&36\}$. The reason is that the label `description` does not only appear in the corresponding query graph. There exists a label path in the XML schema graph, `/site/open_auctions/open_auction/annotation/description`, which leads to the label `description` and is not included in the corresponding query graph. In order to fully utilize the XML extent index, we propose an effective path shortening technique.

Technique 2. (Effective Path Shortening) *Let G_t be an XML schema graph and $Q = \cdots /l_k$ be an arbitrary valid path expression query. A corresponding query graph G_q can be constructed ($G_q \subseteq G_t$) as follows. Let Q_u be a set of labels, l_u, that such that, (i) l_u only appears in G_q and does not appear in $G_t - G_q$, (ii) all data paths, that satsify the query Q, must traverse at least one node with label l_u, and (iii) the label path $l_1/\cdots/l_u$ is unique. We call those labels unique labels. The effective path shortening technique logically divides a given query, Q, into two parts, Q_1/Q_2, where the end label in Q_1 is a unique label, say l_u. Therefore, Q_1 can be processed using the XML extent index, for $\overline{ext}(l_u)$, followed by extent join to process the second part of path Q_2. If multiple unique labels appear in Q, then the last unique label shall be used as the end label of Q_1.*

Reconsider Q_2 = `/site/closed_auctions/closed_auction/annotation/description`, which cannot be processed using the simple path shortening technique. In this case, we find that there exist three unique labels: `site`, `closed_auctions` and `closed_auction`. The `closed_auction` is the last in the path. Here, Q_2 can be rewritten as two parts Q_{2_a}/Q_{2_b} where Q_{2_a} = `/site/closed_auctions/closed_auction` and Q_{2_b} = `annotation/description`. The path of Q_{2_a} can be processed using the XML extent index, because $ext(Q_{2_a}) = \overline{ext}($`closed_auction`$)$.

The effective path shortening technique is a generalization of the simple path shortening technique. The effective path shortening technique improves the query performance by optimizing the path expression. Next, we introduce the *path complementing* technique, which rewrites a complex and high cost path expression, and produces an equivalent simple and low cost path expression. The technique is based on the following observation.

Observation 2. *Given the XML schema graph, G_t, (Figure 2), and a valid path expression query, Q_3 = `/site/regions/*/item/name`. This query cannot be processed using simple path shortening technique, because there exits a label path in G_t, `/site/people/person/name`. The query can be processed as $\overline{ext}($`name`$)$ - $ext($`/site/people/person/name`$)$.*

Technique 3. (Path Complementing) *Let G_t be an XML schema graph and $Q = \cdots /l_k$ be an arbitrary valid path expression query. A corresponding query graph G_q can be constructed ($G_q \subseteq G_t$) as follows. Let G'_q be a query graph including paths $/l_d/\cdots/l_k$ that are not included in G_q, where l_d is the label of the root node of G_t. Then, $ext(Q) = \overline{ext}(l_k) - ext(Q')$, where Q' represents the query that corresponds to G'_q.*

For a given path expression query, Q, there are many choices to process Q using either the effective path shortening technique, or the path complementing technique, or both. For instance, reconsider Q_3 = `/site/regions/*/item/name` using the XML schema graph (Figure 2). Using the path complement technique, Q can be processed using $\overline{ext}$(`name`) - ext(`/site/people/person/name`). Furthermore, the path expression `/site/people/person/name` can be further processed using the effective path shortening technique, because `person` is a unique label. That is, we can process `/site/people/person` using the XML extent index followed by an extent join.

Now we show how to use these two RPE optimization techniques in query processing procedure. The selectivity of path expression and cost estimation are not the focuses of this paper, so the details of these issues are ignored. The general steps of querying and optimizing path expression queries are shown as follows.

(1) Rewriting path step '*'. With the XML schema graph, label paths containing path step '*' are rewritten to the unions of all possible label paths only containing connectors '/' and '//'. (2) Rewriting connector '//'. With the XML schema graph, label paths containing connector '//' are rewritten to the unions of all possible label paths only containing connects '/'. (3) Complementary path selection. With the XML schema graph, the complementary paths of each label path are found and their costs are estimated. Check if the cost of complementary paths is lower than that of the original path. If does, the complementary approach is chosen. Otherwise, the original path is chosen. (4) *Path shortening*. With the given XML schema information, every label path is shorten by techniques 1 and 2 described before. (5) Index selection and query execution plan generation. Select correct indexes and transform the path expressions into query execution plans. (6) Query plan execution. Executing the query plan including indexes and joins.

4 Performance Evaluation

The hardware platform of the benchmark is a PC with a 993MHz CPU and 386MB memory, and the software platform is XBase system [8]. The entire benchmark program is written in C++ and Inada 2.0 (an object oriented persistent programming language), and it is compiled using MS VC++ 6.0. All the benchmarks are based on four data sets: two benchmark data sets (XMark and XMach-1) and two real dataset (Shakes and DBLP).

The partial schema of XMark [11] is shown in Figure 2. It has 20 queries that cover a lot of operations such as exact match, ordered access, casting, regular path expressions etc. The data scale factor used in our experiments is 1.0, and the size of the document is about 100M bytes. XMach-1 [10] is used to simulate applications on the web. The data set contains many documents having different DTDs. A special document is designed in the benchmark used as a directory to record the information of the documents. There are 8 query operations and 3 update operations in XMach-1. This paper does not analyze the performance of update operations. DBLP is a real data set, the XML document set of DBLP web site. The feature of this data set is that the number of the documents is very large, while the size of each of them is very small. The data set can be gotten on Internet at `ftp://ftp.informatic.uni-trier.de/pub/users/Ley/bib/records.tar.gz`. 8 queries are defined on the *DBLP* data set [6]. Shakes is another real data set, s a set of XML documents of operas of Shakespear marked by Bosak. There are 8 queries [6]. This data set can be gotten on Internet at `http://metalab.unc.edu/bosak/xml/eg/shakes200.zip`.

Figures 4, 5, 6 and 7 show the performance improvement of the *path shortening* strategy on benchmark data sets XMark, XMach-1, DBLP and Shakes, respectively, comparing to the *extent join* algorithm. Obviously, *path shortening* has better performance than *extent join* in all queries. It seems that the performance improvement on XMark is not very much. This is because the queries in XMark is XQuery queries other than path expression queries, there thus are time-consuming post-processing operations to do after path expression computing. Even through, *path shortening* do a much better job than *extent join*. Unlike real data sets, simulation data sets specially designed for XML database benchmarks have more complete types of queries, so the query performances on XMark and XMach-1 are studied first. Because of different query features, the performance improvement of *path shortening* is different.

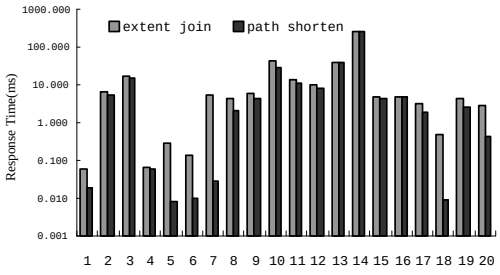

Figure 4. Path Shortening Query Performance (XMark)

(1) Performance of some queries is greatly improved, such as Q5, Q6, Q7, Q18 and Q20 in XMark and Q2, Q3, Q4, Q5, Q6 and Q7 in XMach-1. Path expressions of these queries can be shortened to very short paths, even to one

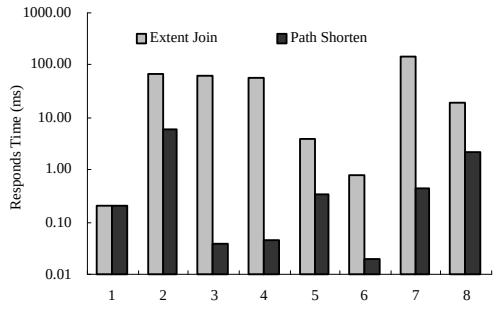

Figure 5. Path Shortening Query Performance (XMach-1)

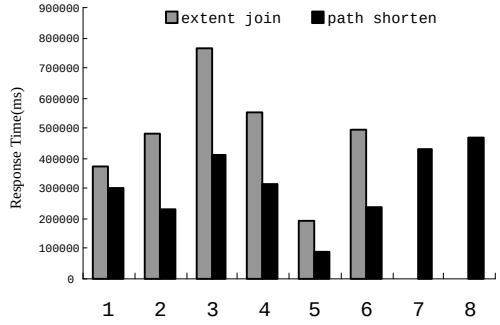

Figure 6. Path Shortening Query Performance (DBLP)

step paths. Another feature of these queries is that they have no predicates or only have predicates at the end of their paths. In these queries, *path shortening* is 10 to 200 times faster than *extent join*.

(2) Performance of some other queries is slightly improved, such as Q13, Q14, Q15 and Q16 in XMark and Q1 in XMach-1. In these queries, the performance improvement of *path shortening* is 0.3% to 8%. There are various reasons. Some queries consume much time to compute special operations other than compute the path expressions. For example, in XMark, Q13 has a complicated reconstruction operation and Q14 has a full-text search operation. Some path expressions are very long themselves but parts that can be cut off are very limited. For example, Q15 and Q16 can only be cut off 3 steps respectively while their original lengths are 12 and 14. Some other path expressions can hardly be shortened any more, such as Q1 in XMach-1.

(3) Performance improvement of the remainder queries is between the above two cases. For example, Q1, Q2, Q3,

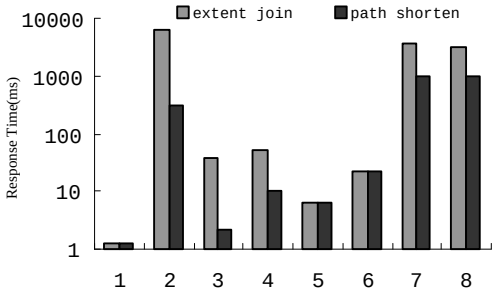

Figure 7. Path Shortening Query Performance (Shakes)

Q4, Q8, Q9, Q10, Q11, Q12 and Q17 in XMark and Q8 in XMach-1. Their path expressions can be slightly shortened or also have some high-cost operations such joins on values, ordered access, dereference etc, the performance improvement is not clearly seen. Most queries belongs to this type, and their performance that can be promoted by *path shortening* ranges from 10% to 400%.

The performance of queries on DBLP and Shakes are shown in Figures 6 and 7, respectively, in which Q7 and Q8 of DBLP using *extent join* cost too much time and cannot get the results. Except for a few queries such as Q1, Q5 and Q6 in Shakes, most queries can benefit from *path shortening*, some of which are improved greatly. These queries can also be divided into the types above. Unlike the simulation data sets, the real data sets are not very complicated, so they are more suitable to use the *path shortening* strategy.

Another valuable conclusion is that the data sets containing large quantity of small documents such as XMach-1, DBLP and Shakes benefit more from the *path shortening* strategy than one large document data set XMark. This is because, in the data set with numerous small documents, the set of joins is very large and even one step path may contain a lot of joins, the performance would be greatly improved when the paths are shortened even by only one step. Since the queries of the benchmark are all path expression queries, according to Observation 1, all the queries can be shortened at least by one step using the *path shortening* strategy.

Because cost evaluation is necessary for the *path complementing* strategy, whether or not this strategy can be used for a query depends not only on path expressions and XML schemas but also on the statistic information of XML documents. Thus, only some queries can be computed use their equivalent complementary paths. We use three queries based on XMark shown in Table 1. Figure 8 gives the average performance of the *path complementing* strategy. In some queries, the *path complementing* strategy can improve the performance by more than 20% to 200% on the basis of

Table 1. Path Complementing Queries

Query	Original Path Expression	Complementary Expression
Q1	`/site/regions/(asia|africa|europe |australia|samerica)/item`	`/site/regions/namerica/item`
Q2	`/site/regions/(asia|africa|europe |australia|samerica|namerica)/item/name`	`/site/people/profile/name`
Q3	`/site/(closed_auctions/closed_auctions|open_auctions /open_auction)/annotation/description`	`/site/category/description`

the *path shortening* strategy.

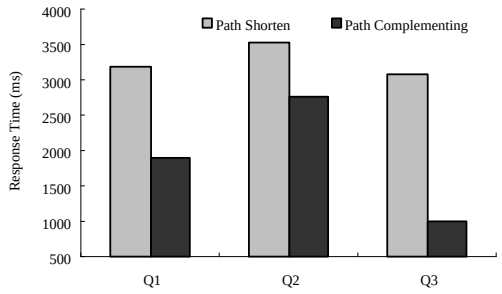

Figure 8. Path Complementing Query Performance

5 Conclusion

This paper proposed two optimizing strategies to improve the performance of path expressions: the *path shortening* strategy and the *path complementing* strategy. The *path shortening* strategy shortens path expression to reduce the cost of the query, while the *path complementing* strategy uses equivalent path expression with the least cost to compute the original path expression. The experimental results of simulation data and real data show that these two strategies are efficient and effective, and they can greatly improve the performance of path expression queries. The performance of 80 percent of the queries in the four benchmarks can be improved by 20% to 400%.

Acknowledgement. Guoren Wang's research is partially supported by the Teaching and Research Award Programme for Outstanding Young Teachers in Post-Secondary Institutions by the Ministry of Education, China (TRAPOYT) and National Natural Science Foundation of China under grant No. 60273079. Mengchi Liu's research is partially supported by National Science and Engineering Research Council of Canada.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. *Int'l Journal on Digital Libraries*, 1(1):68–88, 1997.

[2] J. Cark and S. DeRose. Xml path language (xpath), ver. 1.0. tech. report rec-xpath-19991116, w3c. Technical report, November 1999.

[3] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for xml. In *Proceedings of the Eighth International World Wide Web Conference*, Toronto, Canada, May 1999.

[4] P. Fankhauser. Xquery formal semantics: State and challenges. *SIGMOD Record*, 30(3):14–19, 2001.

[5] D. Florescu and D. Kossmann. Storing and querying xml data using an rdmbs. *IEEE Data Engineering Bulletin*, 22(1):27–34, 1999.

[6] H. Jiang, H. Lu, W. Wang, and J. Yu. Path materialization revisited: An efficient storage model for xml data. In *Proceedings of Thirteenth Australasian Database Conference*, Melbourne, Victoria, 2002.

[7] M. Liu and T. Ling. Towards declarative xml querying. In *Proceedings of The 3rd International Conference on Web Information Systems Engineering(WISE'02)*, Singapore, December 2002.

[8] H. Lu, G. Wang, G. Yu, Y. Bao, J. Lv, and Y. Yu. Xbase: Making your gigabyte disk queriable. In *Proc. of the 2002 ACM SIGMOD Conference*, Wisconsin, USA, 2002.

[9] J. Lv, G. Wang, J. X. Yu, G. Yu, H. Lu, and B. Sun. Performance evaluation of a dom-based xml database: Storage, indexing and query optimization. In *Proceedings of the 3rd International Conference On Web-Age Information Management(WAIM2002)*, pages 13–24, Beijing, China, August 2002.

[10] E. Rahm and T. Bohme. Xmach-1: A multi-user benchmark for xml data management. In *Proc. of 1st VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT2002)*, Hong Kong, China, 2002.

[11] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002.

[12] A. Zhou, H. Lu, S. Zheng, Y. Liang, L. Zhang, W. Ji, and Z. Tian. Vxmlr: A visual xml-relational database system. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB2001)*, pages 719–720, Roma, Italy, September 2001.