Indexing XML Data with ToXin

Flavio Rizzolo, Alberto Mendelzon
University of Toronto
Department of Computer Science
{flavio,mendel}@cs.toronto.edu

Abstract

Indexing schemes for semistructured data have been developed in recent years to optimize path query processing by summarizing path information. However, most of these schemes can only be applied to some query processing stages whereas others only support a limited class of queries. To overcome these limitations we developed *ToXin*[1], an indexing scheme for XML data that fully exploits the overall path structure of the database in all query processing stages. ToXin consists of two different types of structures: a *path index* that summarizes all paths in the database and can be used for both forward and backward navigation starting from any node, and a *value index* that supports predicates over values. ToXin synthesizes ideas from object-oriented path indexes and extends them to the semistructured realm of XML data. In this paper we present the ToXin architecture, describe its current implementation, and discuss comparative performance results.

1. Introduction

Semistructured databases [ABS99], unlike traditional databases, do not have a fixed schema known in advance and stored separately from the data. Broadly speaking, semistructured data is self-describing and can model heterogeneity more naturally than either relational or object-oriented data. Examples of such self-describing data are tagged documents such as XML [W3C00]. The data model proposed for this type of data consists of an edge-labeled graph in which nodes corresponds to objects or values and edges to elements or attributes. *Figure 1* shows a fragment of a semistructured database modeled as an edge-labeled graph. This data model carries both data (in the nodes) and schema information (in the edges), being naturally suitable to represent semistructured data.

Yet, semistructured data models pose new challenges in many areas. Let us consider query optimization of path queries for instance. Due to the lack of information about the schema, a naïve evaluation that scans the whole database in the search of those paths that satisfy a given query is prohibitively expensive. In addition, since navigating the graph is essentially pointer traversal and the objects may be scattered across the disk or even stored at different locations, some queries may require many disk accesses and cause significant performance degradation. Recent work has addressed the problem of efficiently evaluating path queries in this area [CCM96, FS98, MW97].

In order to optimize path query processing, several indexing schemes have been proposed in the past for object-oriented databases. Example of such schemes are Path Indexes [BK89, SB96] and Access Support Relations [KM92], which materialize frequently traversed paths in the database in order to support navigation along reference chains leading from one object to another. However, since these approaches are based on the paths found in the schema, it is not possible to use them for the typical schema-less XML documents.

Index structures for semistructured data have been developed in recent years in order to address this problem. These indexing schemes keep record of existing paths in the database summarizing path information. Examples of such indexes for semistructured data are Dataguides [GW97], 1 indexes and 2 indexes [MS99], T-indexes [MS99], and Reversed Dataguides [LS00].

Dataguides are a concise and accurate summary of all paths in the database that start from the root. Every label path - the string formed by the concatenation of the labels of the edges - in the database is described exactly once in the Dataguide. Dataguides reduce the portion of the database to be scanned for path queries and are useful for navigating the semistructured graph from the root. However, since they do not provide any information about the parent-child relationship between nodes in the database, they cannot be used for navigation from any arbitrary node. As a result, in the case of general path queries that require at some point backward navigation, e.g., *"find the year of publication of a given article"* in the database of *Figure 1*, we need to use additional index structures to optimize the query evaluation. A similar problem is faced by 1-indexes, which represent the same set of paths that Dataguides do, although using a different approach.

[1] ToXin was developed within the ToX (Toronto XML Engine) project at the University of Toronto [BBM+01]

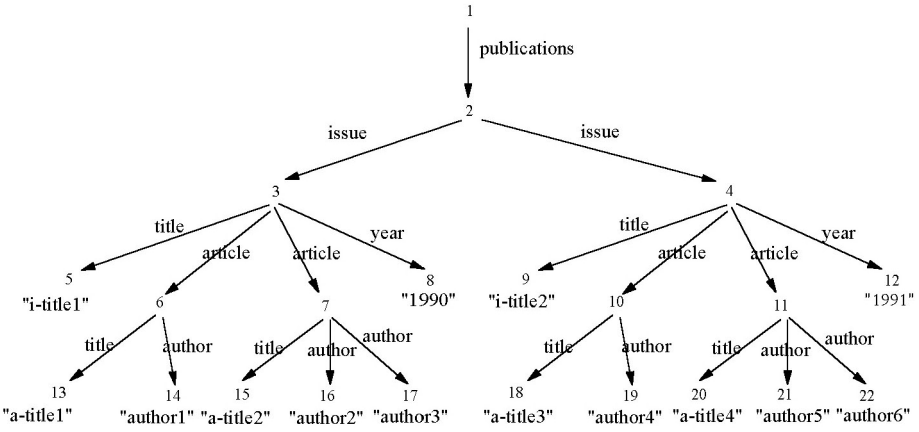

Figure 1: Fragment of a generic publication database

The Lore system [MAG+97] attempts to address this problem by using two additional index structures for forward and backward navigation, the Bindex and the Lindex [MW97]. These indexes, however, do not exploit the database structure as Dataguides do: the nodes, edges and values are stored regardless of the overall path structure of the database. Although using such indexes reduces the number of pointer traversal operations, the lack of path information causes extensive look-ups and joins over large data sets for both forward and backward navigation from a generic node. For instance, let us consider the query "*find the title of the issue where a given article was published*" in the database of *Figure 1*. To answer that query, not only would Bindex retrieve the parent-child pairs connected by "title" edges in "issue", which are relevant to the query, but also those connected by "title" edges in "article", which are not.

T-indexes are specialized path indexes. Rather than indexing all paths, T-indexes only summarize a limited class of paths specified by a given path template. 1-indexes and 2-indexes are special cases of T-indexes.

In order to optimize query processing for regular path queries we have developed *ToXin*, an indexing scheme that fully exploits the overall path structure of the database and can be used for both forward and backward navigation starting from any node. Our original goal when designing ToXin was to have an index that supports navigation of the XML graph to answer any regular path query. At the same time, we wanted to keep the size of the index linear (in the worst case) w.r.t. the size of the XML graph. In addition, we needed data structures to help locate nodes that not only satisfy regular path expressions but also predicates over values. As we explained above, previous index schemes developed for OODB and semistructured data satisfy only some of these requirements.

2. ToXin Overview

ToXin consists of two different types of index structures: the *Value Index* and the *Path Index*. The Path Index has two components: the *index tree*, which is a *Minimal Dataguide* [GW97], and a set of *instance functions*, one for each edge in the index tree. Since we are using an edge-labeled tree as data model (we do not consider the semantic of the IDRefs in the current prototype), we need the instance functions to keep track of the parent-child relationship between the pair of nodes that defines each XML element. The algorithm that constructs the index tree works as follows. First, it performs a depth-first traversal of the XML tree. For each edge visited in the XML tree it first checks whether the corresponding index edge has already been added and adds it if it was not. Then it updates the instance function for the current index edge by adding the pair (parent node, child node) of the current XML element.

The structure and the contents of the instance functions resemble those of Access Support Relations. ToXin instance functions are the equivalent of binary access support relations and the Minimal Dataguide plays a similar role to the object-oriented schema in the construction of the index. However, while the structure of Access Support Relations is static, the structure of the instance functions changes every time the index tree is updated.

Each instance function is stored in two redundant hash tables: a *forward instance table* for forward navigation and a *backward instance table* for backward navigation.

The Value Index, on the other hand, consists of a set of value relations that store the XML nodes and values corresponding to an index edge. A value relation is created for each edge in the index schema that corresponds to a set of XML nodes containing values. Each relation is implemented as B+-trees keyed on the values, which are

always strings. Value and Path Indexes combined can be used to answer regular path queries with predicates over values such as those expressed using *XPath* [W3C99].

Using XPath syntax, the hierarchical relationship between nodes are expressed by operators "/" and "//". The former specifies a parent-child relationship between two nodes whereas the later specifies an ancestor-descendant relationship between two nodes at any depth. The sections between brackets in an XPath expression are called *filter sections*. We consider in this paper XPath queries having only one filter section, but the discussion given here can be generalized to queries with any number of filters. A filter is a predicate that is applied to the nodes that match the path expression before it (i.e. the pre-filter section). For instance, the query *"find all the titles of articles from 1990"* can be expressed in XPath as //issue[year="1990"]/article/title. The evaluation of the rest of the pattern (i.e., post-filter section) continues only for those nodes that matches the pre-filter section and satisfies the filter. XPath also includes a wildcard operator "*", that matches any node at its location in the path expression, and a disjunction operator "|".

Let us see with an example how ToXin works. *Figure 2* shows the ToXin tree and the tables for the XML document from *Figure 1*. The VT boxes in *Figure 2* represent the value tables and the IT boxes the instance tables. In contrast to Dataguides and 1-indexes, which index only paths which start from the root, all paths in the database are represented in ToXin. For instance, not only do we find the paths that match x/publications/issue/y (represented by the reference chains in tables IT1 and IT2) but also those that match x/issue/y (represented by the table IT2). This way we can use ToXin for both forward and backward navigation starting from any node in the index.

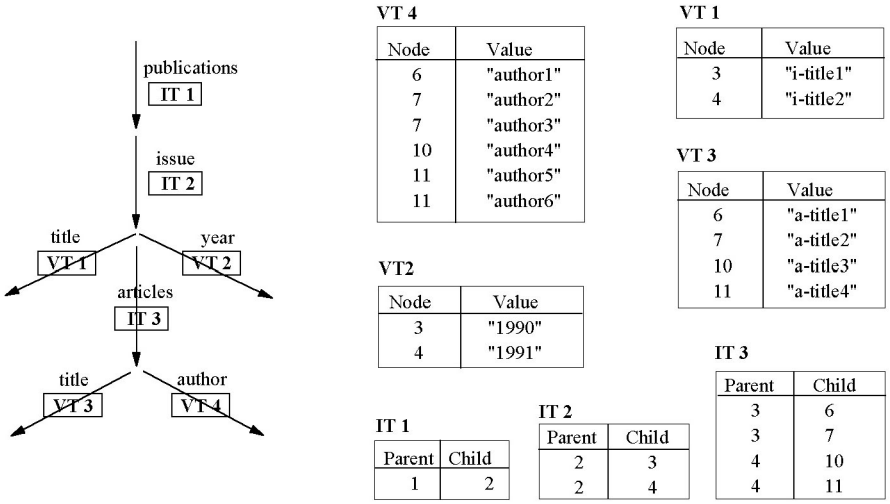

Figure 2: ToXin Tree and Tables

3. Experiments

We implemented the ToXin prototype in Java 2 and we used the IBM parser XML4J to create the DOM tree from the XML document. All the experiments presented herein were conducted on a Sun SPARCstation running Solaris 2.5. Both Index and DOM were kept in main memory in the experiments.

In order to study the tradeoffs of a number of indexing schemes we carried out a series of comparative performance experiments. We use query processing time as performance metric. According to our query decomposition strategy we define three stages in the query evaluation, each one corresponding to an XPath section. The first stage (called *pre-selection stage*) comprises the first navigation down the tree for the pre-filtering section. The second stage (called *selection stage*) consists of the value selection performed by the filter section. Finally, the third stage (called *post-selection stage*) spans the navigation up after the selection stage and the last navigation down. The experiments evaluate the effect of several parameters on the performance of the query evaluation using ToXin, Dataguides, and Dataguides + Value Index. The parameters can be classified into *data source-specific* and *query-specific*. The data source-specific parameters are document size, number of XML nodes and values, path complexity (degree of nesting), and average value size (short or long strings). Those query-specific are selectiveness

of the path constraints (queries expressed with or without // and * operators), size of the query answer (small or large), and number of elements selected in the filter section (small or large).

	DBLP		IMDB		Shakespeare Works	
	Doc 1	**Doc 2**	**Doc 1**	**Doc 2**	**Doc 1**	**Doc 2**
File size (Mb)	1.8	8.9	0.8	3.9	1.1	4.4
Index generation time (sec)	17.2	80.2	5.5	41.5	4.1	25.7
# of XML nodes	90040	405103	57854	293183	45776	181438
# of XML values	44027	190232	27084	137296	20437	81779
# of ToXin nodes	27	40	8	8	31	37
Avg. Value Size	Short	Short	Short	Short	Long	Long
Degree of Nesting	High	High	Low	Low	High	High

Table 3: Benchmark Parameters

To compare query performance across different documents we classify the queries with regards to the selectiveness of its path constraints and the sizes of the query answer and node selection in the filter section. *Table 4* shows the classification according with these criteria.

Query Type	Query	Characteristics
LL	q_1 = /dblp/conference/issues/issue/inproceedings[year = "1998"]/title q_2 = /movies/movie[genre = "Drama"]/title q_3 = /shakespeare/play/act/*/speech[speaker = "Mark Anthony"]/line	Large query answer Large filter selection
LL*	q_{1*}= //[year = "1998"]//title q_{2*} = //[genre = "Drama"]//title q_{3*} = //[speaker = "Mark Anthony"]//line	Large query answer Large filter selection Relaxed constraints
LS	q_4 = /dblp/conference[title = "VLDB"]/issues/issue/inproceedings/title q_5 = /shakespeare/play[title = "The Tragedy of Anthony and Cleopatra"] /act/*/speech/line	Large query answer Small filter selection
LS*	q_4' = //conference[title = "VLDB"]/*//title q_5' = //[title = "The Tragedy of Anthony and Cleopatra"]/act//line	Large query answer Small filter selection Relaxed constraints
SS	q_6 = /dblp/conference/issues/issue/inproceedings [author = "Serge Abiteboul"]/title q_7 = /movies/movie[year = "1950"]/title q_8 = /shakespeare/play[title = "The Tragedy of Anthony and Cleopatra"]/personae/persona	Small query answer Small filter selection
SS*	q_6' = //[author = "Serge Abiteboul"]//title q_7' = //[year = "1950"]//title q_8' = //[title = "The Tragedy of Anthony and Cleopatra"]//persona	Small query answer Small filter selection Relaxed constraints

Table 4: Query Classification

Our benchmark consists of four data sources: the conference papers from the DBLP database [Ley00], a sample of movies from the Internet Movies Database [IMDB00], the twenty Shakespeare plays from [Bos99], and four religious texts from [Bos98]. The choice of the document samples was aimed at determining the impact of nesting and average value size on the index performance. DBLP is a classical example of a bibliographical database containing deeply nested data. IMDB presents a flat structure typical of a straight-forward mapping from relational data. The values contained in both the DBLP and IMDB documents are short strings. The Shakespeare works and the religious texts, on the other hand, are exponents of text databases with longer string values than DBLP and IMDB. In order to save space, we have omitted here the religious texts results, but they can be found in [Riz01].

We use DBLP and IMDB to explore the impact of nesting. To study the effects of different value sizes, we use Shakespeare works. In addition, to test the index performance with different document sizes and similar path structures, we created two XML documents for each data source, one larger than the other one. *Table 3* shows some parameters of the benchmark.

Figure 5 summarizes the performance of LL, LL*, SS, and SS* queries on three different index schemas: ToXin, Dataguides, and Dataguides with Value Indexes (we provide in [Riz01] a more detailed report of the experiments).

In all of them ToXin outperforms the other two schemes, in some cases by one order of magnitude. The results suggest that the first stage (pre-selection) generally benefits the most by the use of a path index scheme. All query types, except SS in the Shakespeare works, have shown important performance improvements when using an index in the first stage. In some queries the performance of the third stage (post-selection) is also considerably improved by using the index. Those query types with large query answers benefit the most by the use of a path index during the third query processing stage. From the results we can also conclude that, when using an index, the closer to the root the filter section starts, the better the improvement in the performance of the third stage and the worse that of the first stage. Since Dataguides support only the first stage, when the filter section is close to the root, using only a Dataguide without additional structures can do little to improve the performance of the entire query evaluation.

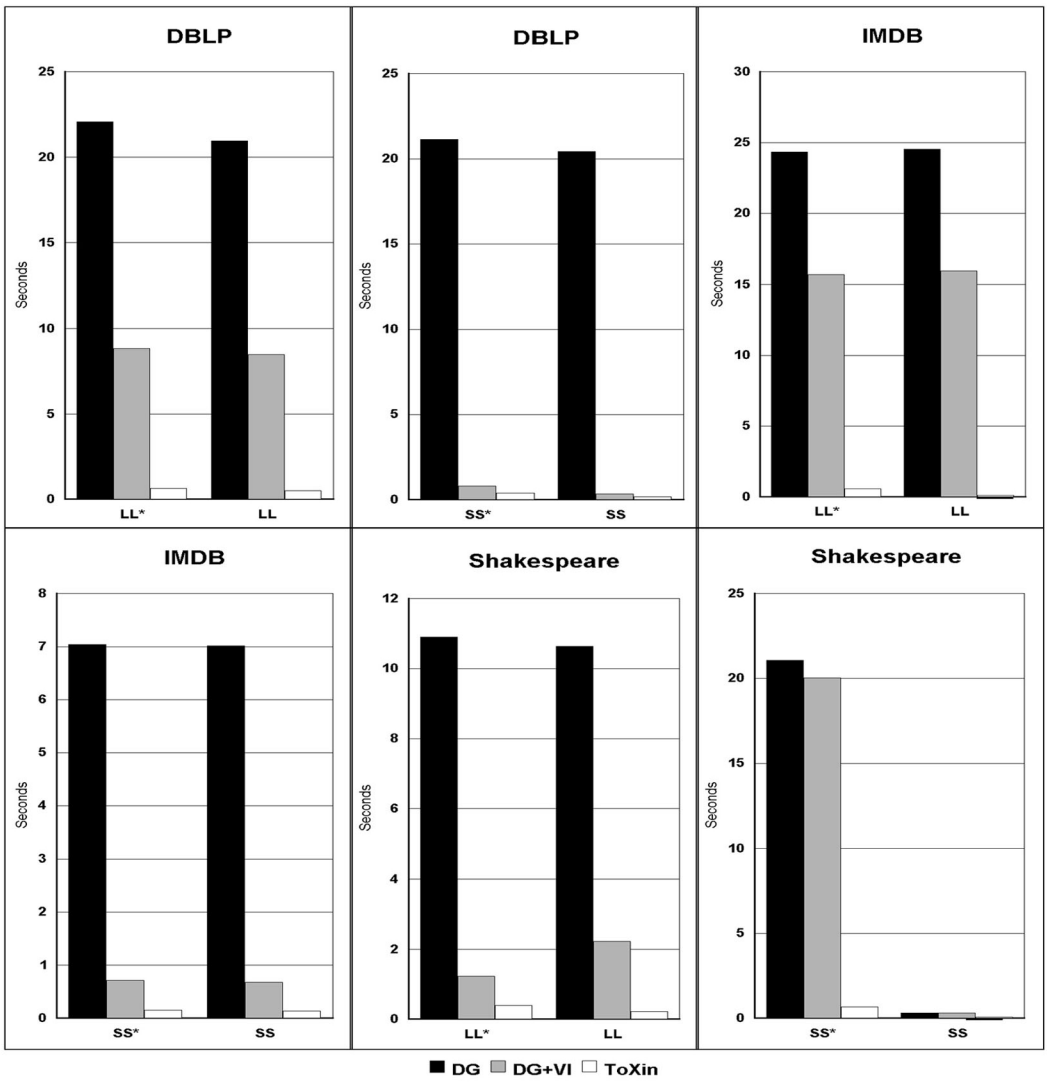

Figure 5: Summary of Performance Results on Selected Query Types

Regarding the results when using a value index, except for queries with small number of values selected in the filter section, the improvement is also considerable. Nevertheless, it is important to note that in none of the tested queries does the selection stage play an important role in the total performance of the query evaluation.

4. Conclusions

In this paper we presented *ToXin*, an indexing scheme that supports path queries over XML data. We discussed its architecture, related work and presented performance results. The motivation for this work was to overcome some limitations of current indexing proposals for semistructured data, such as the lack of support for all query processing stages (Dataguides, 1-indexes and 2-indexes), and the need for an explicit specification of the paths to index (T-

indexes). To that end, we combined ideas from Dataguides and access support relations in a way that allows us to use the index in all the query evaluation stages for any general path query.

The experimental results suggest that the query types that benefit the most by using ToXin are those with large query answers. In addition, the closer to the root the filter section starts, the wider the difference in performance between ToXin and other indexing schemes.

We are currently extending the work presented here in several ways. The main directions are: adding order to the index structure; implementing the ToXin graph, by extending the ToXin tree with the semantics of the IDRefs; making the index persistent; and investigating ways to extend ToXin so it can be used as an alternative to DOM for storing, querying and updating XML documents.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada and Bell University Laboratories.

References

[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann, 1999.

[BBM+01] D. Barbosa, A. Barta, A. Mendelzon, G. Mihaila, F. Rizzolo, P. Rodriguez-Gianolli. ToX: The Toronto XML Engine. In *Proc. of the Workshop on Information Integration on the Web*. 2001.

[BC00] A. Bonifati, S. Ceri. Comparative analysis of five XML query languages. *SIGMOD Record* 29(1): 68-79.

[BK89] E. Bertino and W. Kim. Indexing techniques for queries on nested objects. *IEEE Transactions on Knowledge and Data Engineering*, 1(2): 196-214, 1989.

[Bos98] Jon Bosak. Religious Texts in XML. *http://www.ibiblio.org/xml/examples/religion*. 1998.

[Bos99] Jon Bosak. Complete Plays of Shakespeare in XML. *http://www.ibiblio.org/xml/examples/shakespeare*. 1999.

[CCM96] V. Christophides, S. Cluet and G. Moerkotte. Evaluating queries with generalized path expressions. *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, 413-422, 1996.

[FK99] D. Florescu and D. Kossmann. Storing and querying XML data using and RDBMS. *IEEE Data Engineering Bulletin* 22(3): 27-34, 1999.

[FS98] M. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proc. of the IEEE Int'l Conf. on Data Engineering*, pages 14-23, 1998.

[GW97] R. Goldman and J. Widom. DataGuides: enabling query formulation and optimization in semistructured databases. In *Proc. of the Int'l Conf. on Very Large Databases*, pages 436-445, 1997.

[IMDB00] The Internet Movie Database. *http://www.imdb.com*. 2000.

[KM92] A. Kemper and G. Moerkotte. Access support relations: an indexing method for object bases. *Information Systems*, 17(2): 117-145, 1992.

[Ley00] Michael Ley. DBLP database web site. *http://www.informatik.uni-trier.de/~ley/db*. 2000.

[LS00] H. Liefke and D. Suciu. Xmill: an Efficient Compressor for XML Data. In *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, pages 153-164, 2000.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom. Lore: a database management system for semistructured data. SIGMOD Record 26(3): 54-66, 1997.

[MS99] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of the Int'l Conf. on Database Theory*, pages 277-295, 1999.

[MW97] J. McHugh and J. Widom. Query optimization for semistructured data. *Technical Report, Stanford University*, 1997.

[Riz01] Flavio Rizzolo. ToXin: an indexing scheme for XML data. Master's Thesis. Department of Computer Science, University of Toronto, 2001.

[SB96] B. Shidlowsky and E. Bertino. A graph-theoretic approach to indexing in object-oriented databases. *Proc. of the IEEE Int'l Conf. on Data Engineering*, pages 230-237, 1996.

[W3C00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, Eve Maler (Eds.). Extensible Markup Language (XML) 1.0 (Second Edition). In *http://www.w3.org/TR/REC-xml*.

[W3C98] W3C Recommendation. Document Object Model (DOM) Level 1 Specification. In *http://www.w3.org/TR/REC-DOM-Level-1*. 1998

[W3C99] W3C Recommendation. XML Path Language (Xpath) 1.0. In *http://www.w3.org/TR/xpath*. 1999.