XML Storing and Processing Techniques

© Andrey Fomichev

Institute for System Programming of the Russian Academy of Sciences,
Moscow State University
fomichev@ispras.ru
Ph.D. Advisor S. D. Kuznetsov

Abstract

This paper gives an overview of the current research activities of the author in the area of XML data management. It sketches the following topics of interest of the author: XML data organization methods, query evaluation model for XQuery and physical optimization of XPath and XQuery queries. The paper presents author's current results in these areas and outlines the plan for future work.

1 Introduction

There is no doubt that XML has already gained ground as a widespread format for information exchange. With significant growth of amounts of XML data being transmitted industry needs systems dealing with huge XML documents in efficient way. To be successful such systems should have strong *physical layer*, which can serve as a basis for the full-featured native XML DBMSs that satisfies any user need.

Under the term of *physical layer* we understand the following: data representation in secondary and main memory, memory management, query evaluation facilities and physical query optimization (i.e. optimization, which depends on the knowledge about data and data structures). Summing up the experience of a number of research papers, industry needs and our own experience, we would like to outline the following requirements to physical layer:

- Support for large XML documents (much more than 1Gb);
- Efficient support for data updates;
- Efficient access to data by regular path expressions such as XPath [1] queries;
- Fast execution of queries formulated in high-level query languages such as XQuery [2], XSLT [3].

This paper describes the effort of the author in solving the problems discussed. The results presented were achieved during the work under the following projects: BizQuery [4] — virtual data integration system and Sedna [5] — native full-featured XML DBMS. Both systems were built from scratch with the goal to support XML storing and processing efficiently. The query language of both systems is XQuery.

The rest of the paper is organized as follows. Section 2 gives an outlook at related work. Section 3 presents our data representation for XML. Section 4 describes our query evaluation model. Section 5 sketches our work on physical query optimization. Section 6 draws some future plans and concludes the paper.

Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, St.-Petersburg, Russia, 2004

2 Related Work

We start with the description of related work concerning XML storage systems that concentrate on data organization for XML (which also includes the problems of efficient regular path queries processing). Then we outline what is done in the area of XQuery processing.

The problem of storing and processing XML documents efficiently has been admitted by the database community as a challenge and caused high research activity in this field. Historically, the first wave of research was adopting relational DBMSs for storing XML. The whole paper is not enough for detailed description of work that has been done, so we can only recommend a summary [6]. But the result of this research consists in principle constraints of pure relational DBMS to handle XML documents efficiently. Actually, XML documents are stored in relational systems either as atomic entities such as BLOBs or being decomposed into relations. The first way of storing cannot guaranty high performance of query evaluation because we need to extract the whole document from database. The second way leads to a great number of resource consuming joins to compose result.

Understanding drawbacks of using relational DBMSs for storing XML caused high activity in development of native XML DBMSs, which would not be straitened by any existing infrastructure. Not pretending to give the complete classification we would like to underline the essential characteristics of these systems. The first group consists of the systems that decompose XML documents at the node level like in case of using relational DBMSs, but make an accent on efficient reconstruction of XML documents (reconstruction is the inverse operation for decomposition). The key to this problem lies in efficient determination of parent-child and ancestor-descendent relationships between nodes. For that reason the notion of *numbering scheme* is introduced. The reconstruction of XML is performed by special join operations (structural joins or containment joins) with the help of the numbering scheme. Usually it is insufficient to have only a numbering scheme and such systems have a set of indexes to get quick access to nodes by name and to avoid tree traversal (because tree traversal leads to a number of structural joins). Most papers, which play around that idea, pay little attention to storage system and updates, but rather concentrate on efficient numbering scheme implementation and optimization of structural joins. An example of such systems is XISS [7].

Native XML systems, that make up the second group, work on placement of an XML document (which is essentially a tree) into a number of secondary memory blocks. In this case an XML document is represented as a number of nodes, which are somehow connected with each other by references, and the task is to distribute these nodes among the blocks to satisfy some requirements. For instance, the requirement may consist in minimizing the number of blocks used or in organizing blocks in a balanced tree, so any leaf of the XML tree can be accessed by reading a small fixed number of blocks (usually 2 or 3). A drawback of such approach is that it requires the resource consuming tree traversal operation for path queries, so some indexes should be introduced to speed up query execution. An example of such systems is Natix [8].

The third group of native XML DBMSs is the most promising from our point of view. Their main characteristic is that they use *descriptive schema* (or *data guide*, which is nearly the same) of XML document. *Descriptive schema* is defined as follows: every path of the document has exactly one path in the descriptive schema, and every path of the descriptive schema is a path of the document.

The earliest work on exploiting descriptive schema for XML data management, as far as we know, is the Lore project [9]. Their data guide was primarily used for query optimization. SphinX [10] system uses descriptive schema for organizing indexes on XML documents. We appreciate this work and think that our approach is closer to theirs than to any other. But they concentrate on indexing XML and do not discuss storage system and updates at all. One of the latest works on compressing XML [11] also takes into account the advantages of descriptive schema. Compressing skeleton that presents the structure part of an XML document they get a variant of data guide, which takes little memory and speeds up query execution.

But to the best of our knowledge there is no any native full-featured XML storage system built on the principles of the third group, which not only introduces indexes for XML, but also takes into account how XML is stored in secondary memory and how many I/O operations are performed for queries and updates. In our work we explore this idea and try to apply the descriptive schema to the XML storage organization.

In contrast with the XML storing methods, XML query processing is not very well elaborated. The developers have been concentrating on the support for full XQuery rather than on sophisticated methods of XQuery implementation. We would like to mark out only an effort made to bring the iterative query execution model to the XML world from the relational one. Several implementations of this model appeared nearly simultaneously, so it hard to say who was the first. We made it in [4].

3 Data Organization

Designing data organization, we would like it to be efficient for both queries and updates. As the result, the following main decisions were maid. First, we have developed a *descriptive schema driven storage strategy* which consists in clustering nodes of an XML document according to their positions in the descriptive schema of the document. Second, *direct* pointers are used to represent relationships between nodes of an XML document such as parent, child, and sibling relationships. Because of lack of space we do omit lots of details here and present main ideas only. More information can be found in [12], [13].

```
<library>
  <book>
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
  </book>
  <book>
    <title>An Introduction to Database
           Systems</title>
    <author>Date</author>
    <issue>
      <publisher>Addison-Wesley</publisher>
      <year>2004</year>
    </issue>
  </book>
  . . .
  <paper>
    <title>A Relational Model for
           Large Shared Data Banks</title>
    <author>Codd</author>
  </paper>
</library>
```

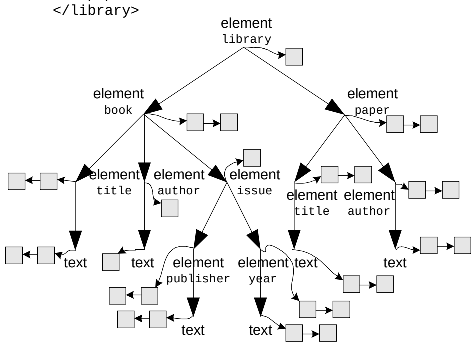

Figure 1. Data Organization

The overall principles of the data organization are illustrated in Figure 1. The central component is the descriptive schema that is presented as a tree of schema nodes. Each schema node is labeled with an XML node kind name (e.g. element, attribute, text, etc.) and has a pointer to data blocks where nodes corresponding to the schema node are stored. Some schema nodes depending on their node kinds are also labeled with names. Data blocks belonging to one schema node are linked via pointers into a bidirectional list.

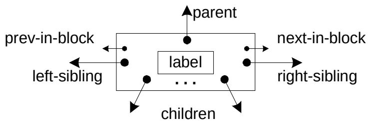

Figure 2. Common structure of node descriptor

The common structure of node descriptors for all node kinds is shown in Figure 2. The meaning of the `left-sibling`, `parent` and `right-sibling` pointers is straightforward. The `next-in-block` and `prev-in-block` pointers are used to link nodes within the block. `children` pointers are used for referencing the children nodes. These pointers are pointers to the first children by the descriptive schema, but not the pointers to 'all' children. This idea helps us to achieve the fixed size descriptors in the block. The `label` field contains a label of *numbering scheme*. Numbering scheme is used for operations based on notion of document order [2].

The data organization presented has the following advantages. First, Descriptive schema servers as an universal structure index for a wide class of XPath queries. Having the query `/library/book/title` we can simply evaluate this query on the descriptive schema and get access to blocks with data we need. Note that we read blocks that contains only the data we need and nothing more. As the result we minimize the number of blocks accessed. Second, direct pointer allow us passing from one node to its neighbours almost for free (if the neighbours are in memory buffers), which is important for effective XQuery implementation.

Besides the main idea of data representation given, there is a number of minor ideas and developments that we would like to emphasize. For complete description see [12], [13].

Not to restrict the size the documents being processed with the size of the virtual address space, we have developed our own layered virtual address space (LVAS). The size of the pointer in LVAS is 64 bits, so we can handle really huge documents.

To support updates efficiently we have made a number of design decisions. First, we have made the implementation of numbering scheme based on strings, which allows us to avoid XML tree reconstruction because of lack of free labels (we exploit the idea that for every two strings `str1` and `str2` such as `str1 < str2` there exist a string `str` for which `str1 < str < str2`). Second, we have achieved node descriptors to be of a fixed size. It simplifies management of free space in block. And third, we have introduced the indirection table for parent pointers to avoid mass updates.

4 Query Evaluation

In this section we would like to concentrate on XQuery specific tasks that have great influence on the query processing performance.

4.1 Suspended Element Constructors

Besides the well-known heavy operations like joins, sorting and grouping, XQuery has a specific resource consuming operation — XML element constructor. The construction of an XML element requires deep copy of its content that leads to essential overheads. The overheads grow significantly when a query consists of a number of nested element constructors. Understanding the importance of the problem, we propose *suspended element constructor*. The suspended element constructor does not perform deep copy of the content of the constructed element but rather stores a pointer to it. The copy is performed on demand when some operation gets into the content of the constructed element. Using suspended element constructor is effective when the result of the constructor is handled by operations that do not analyze the content of elements.

The research [14] of our colleagues allows us to claim that for a wide class of XQuery queries there will be no deep copies at all. Most XQuery queries can be rewritten in such a way that above the element constructors in the execution plan there will be no operations that analyze the content of elements.

4.2 Combining Lazy and Strict Semantics

In Section 2 we have mentioned that we adapted the iterative query execution model to XQuery language. The iterative model is highly suitable for query languages because it avoids unnecessary data materialization and deals with the intermediate results effectively. But keeping in mind that XQuery is a functional language, the iterative model can be regarded as an implementation of lazy semantics. On the other hand, it is generally accepted that computation efficiency of implementation of strict semantics for a programming language is higher comparing with implementation of lazy semantics for this language. As far as XQuery is considered as a general-purpose programming language [15] that can be used for expressing application logic, implementing lazy semantics only has bad impact on overall executor performance. To let the XQuery implementation be efficient for both query and application logic processing we combine these two evaluation models. We are working at the XQuery executor, which keeps track of amounts of data being processed and automatically switches from the lazy to strict modes and vice versa at run-time.

The query evaluation starts in the lazy mode having the execution plan constructed. The overheads of the lazy model strongly correlates with a number of function calls made during the evaluation process. The more function calls are made, the more copies of function bodies are performed. The goal is to find the tradeoff between the copying of function body and the materializing of intermediate results of function's operations. The mechanism is as follows. Every function call is a reason to switch to strict mode if the sizes of arguments are relatively small. Vice versa, the large input sequence for any physical operation in the strict mode is a subject to switch this operation to the lazy mode.

5 Physical Optimization

Data structures presented in the Section 3 gives ground for alternative ways of processing queries. Let us consider the following example: `/library/book[issue/year=2004]/title`. The first strategy of evaluation of this query is to select `/library/book` elements using the descriptive schema, then apply the predicate and the rest of the query using pointers in data. The second strategy is to execute query `/library/book/issue/year/text()` and then to apply the predicate (we select only those nodes, for which the text is equal to 2004), and at last, to apply `../../../title` to the result of the previous step. The idea is that we select blocks to which the predicate applies on the first step omitting blocks with book elements. Then we apply the predicate which potentially cuts off lots of data and then go up the XML hierarchy to obtain the final result.

Numbering scheme also adds a number of strategies for query evaluation. Let us consider the following query: `/*/book[author="Date"]/issue[year=2004]/publisher`. Besides the strategies given above we can use numbering scheme. First, we execute `/library/book/[author="Date"]` and `/library/book/issue[year=2004]` queries as was shown above. On the second step we filter the obtained elements `issue` with the help of numbering scheme by determining ancestor-descendant relationship between them and the selected `book` elements.

The examples of accessing to data given above demonstrate the richness of the strategy space for data representation described in Section 2. A priory, we cannot prove that one strategy is better than the other. So, the optimizer should make a decision based on statistics which strategy is the best one. The author is planning to work it through in the nearest future.

6 Conclusion and Future Work

In this paper we described three directions of the author's current work: XML data organization, XPath/XQuery query evaluation and physical optimization. The first direction is the basis for the rest ones and is very well elaborated and implemented. Query evaluation and physical optimization are the subjects for future work.

References

[1] XML Path Language (XPath) 2.0, W3C Working Draft, 12 November 2003, http://www.w3.org/TR/2003/WD-xpath20-20031112/.

[2] XQuery 1.0: An XML Query Language, W3C Working Draft, 12 November 2003, http://www.w3.org/TR/2003/WD-xquery-20031112/.

[3] XSL Transformations (XSLT) Version 2.0, W3C Working Draft, 12 November 2003, http://www.w3.org/TR/2003/WD-xslt20-20031112/.

[4] Antipin, K., Fomichev, A., Grinev, M., Kuznetsov, S., Novak, L., Pleshachkov, P., Rekouts M. and Shiryaev, D.: Efficient Virtual Data Integration Based on XML, Proceedings of ADBIS 2003

[5] Sedna native XML DBMS http://modis.ispras.ru/Development/sedna.htm

[6] Tian, F., DeWit, D., Chen, J., Zhang, C.: The Design and Performance Evaluation of Alternative XML Storage Strategies. SIGMOD Record 31(1): 5-10 (2002).

[7] Li, Q., Moon, B.: Indexing and Querying XML Data for Regular Path Expressions, Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.

[8] Fiebig, T., Helmer, S., Kanne, C.-C., Moerkotte, G., Neumann, J., Schiele, R., Westmann, T.: Anatomy of a native XML base management system, The VLDB Journal, Volume 11, Issue 4.

[9] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., Widom, J.: Lore: A Database Management System for Semistructured Data. SIGMOD Record, 26(3):54-66, September 1997.

[10] Leela, K., Haritsa, J.: SphinX: Schema-conscious XML Indexing, Technical Report, TR-2001-04, DSL/SERC, http://dsl.serc.iisc.ernet.in/pub/TR/TR-2001-04.pdf.

[11] Buneman, P., Grohe, M., Koch, C.: Path Queries on Compressed XML, Proceedings of the 29th VLDB Conference, Germany, 2003.

[12] Fomichev, A., Grinev, M., Kuznetsov, S.: Descriptive Schema Driven XML Storage, Submitted at ADBIS 2004.

[13] Grinev, M. et al: Sedna: A Native XML DBMS, Submitted at XIME-P 2004.

[14] Grinev, M., Pleshachkov, P.: Rewriting-based Optimization for XQuery Transformational Queries, Submitted at VLDB 2004. Available at www.ispras.ru/~grinev.

[15] Fernandez, M., Simeon, J.: Growing XQuery. ECOOP 2003: 405-430.