Visually Exploring and Querying XML with BaseX

Christian Grün, Alexander Holupirek, Marc H. Scholl

Databases and Information Systems Group
University of Konstanz
<Firstname>.<Lastname>@ uni-konstanz.de

Abstract: XML documents are widely used as a generic container for textual contents. As they are increasingly growing in size, XML databases are emerging to efficiently store and query their contents. Besides, due to the hierarchic structure of XML documents, hierarchic visualizations are needed to facilitiate cognitive access to query results. BaseX is a simple database prototype, mapping XML documents to a table based tree encoding. An integrated treemap visualization and a query interface allow visual access to the documents and demonstrate the efficiency of the underlying data storage.

1 Introduction

XML has since long developed into the standard format for storing arbitrary text contents. Whereas many XML instances are rather small, more and more documents are available which exceed main memory constraints. All major database vendors have already integrated XML as a fixed datatype. Still, the versatility of the XML format leads to a general mistrust, concerning the efficiency of storing and querying. However, XML documents are nothing else than trees, and as such plenty of well-known and proven techniques exists to handle the data structure. Furthermore, the idea to store XML in relational tables, such as e.g. discussed in [FK99], generalized by the XPath Accelerator [Gr02] and realized in the MonetDB/XQuery database [Bo06], has proven to be surprisingly efficient both in terms of speed and memory usage.

With BaseX, we present a database prototype of a table based tree encoding of XML nodes [Gr06][1]. The current version of BaseX is disk-oriented, but can also be run in main memory for volatile document processing. A major focus is set on a memory saving representation of the original document, so the chosen encoding consists of just the essential node attributes which are needed to allow a complete and yet quick traversal of all XPath axes. In contrast to other approaches such as [Bo02], the table representation is schema oblivious, i.e., no DTD or XML Schema is required to encode a document. The integrated XPath processor applies query algorithms that are partly derived from the

[1] Homepage of BaseX: `http://www.inf.uni-konstanz.de/dbis/research/basex`

Staircase Join operator [Gr03]. Very similar algorithms are in fact used to build the treemap visualization and allow user interactions in interactive time.

Section 2 starts with an overview on the applied tree encoding and introduces some sample query algorithms. In Section 3, the treemap visualization and its on-the-fly construction and interaction techniques are presented. Section 4 summarizes our contribution and gives a brief outlook on future work.

2 Tree Encoding

XML documents can be represented as an ordered set of nodes. All nodes can have numerous attributes, such as a unique node ID, references to parent and children nodes, node kinds (element/text), tag names or text content, number of descendant nodes, etc. Some of these attributes are redundant, e.g., references to child nodes and the childrens' parent references. Hence, it is advisable to skip such information as long as two demands are still met:

- the original document can be reconstructed
- query efficiency is preserved

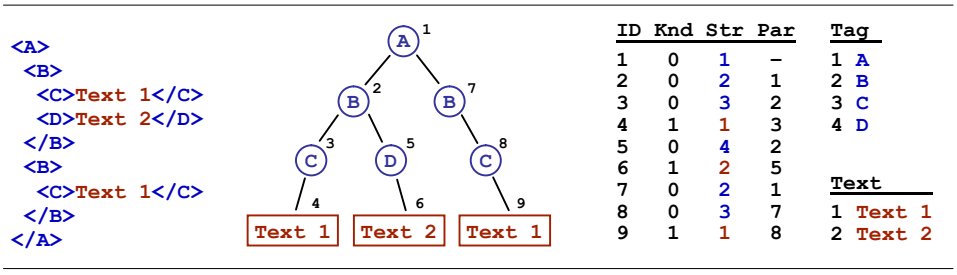

Figure 1: Mapping of an XML document.
left: original XML document, center: tree representation, right: table encoding

Storing three attributes actually suffices to restore XML instances and to process all XPath axes (see Fig. 1): a reference to the parent ID, the node kind and a string reference[2]. The node ID is implicitly given by the table position, and the string is further numerically encoded, pointing to a kind specific text index. Instead of the absolute parent ID, the distance to the parent is stored; this is especially beneficial for performing updates. Due to some characteristics of the stored table attributes, the resulting representation can be compressed by merging attributes together [Gr06]. Thus, each tuple is internally stored in just eight bytes[3]. XML attributes are represented in the table as well; they consist of a name and value combination. As the number of different attribute

[2] note that special XML features such as namespaces or processing instructions are disregarded in this paper for the sake of simplicity.
[3] tuple size grows to 16 bytes for namespace support and node IDs > 2^32.

names and the distance to the element (parent) node is small enough, they can also be stored together in mentioned eight bytes.

The fixed and compact size of the nodes allows for a very efficient and straightforward access in main memory as well as on disk. All table operations that sequentially traverse tuples additionally benefit from common prefetching strategies of hard disks. Node processing is shown here for the `descendant` axis: starting a sequential scan at node `x`, all following nodes are regarded as descendant nodes. As soon as the first node `y` is encountered with `Par(y) < ID(x)`, all descendant nodes have been found, and further node traversal is skipped. If descendants of multiple nodes are to be found, pruning and partitioning concepts are applied to preserve linear complexity of traversal (see [Gr03] for pruning, partitioning and skipping XML tuples based on the `Pre/Post` encoding).

To accelerate value and text based queries, indexes can be constructed for attribute values and text nodes. Queries are internally rewritten to first access the indexes before the remaining XPath is traversed in a backward manner. As an example, in the query `//address[@town="Washington"]` all attribute nodes matching the value "Washington" are assembled before the attribute name `town` and the parent `address` tag is evaluated. This often leads to a speedup of several orders of magnitude and is especially helpful for content based queries as they are offered in our visualization (see Fig. 2).

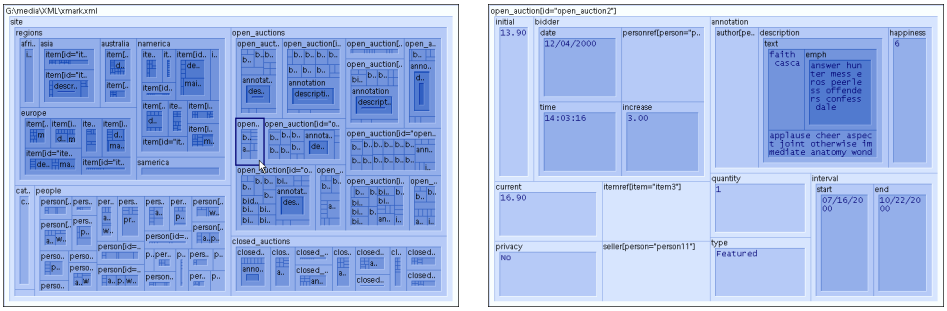

Figure 2: TreeMap Visualization. left: complete view with focused rectangle, right: zoomed view

3 TreeMap

The space-filling TreeMap was inititally introduced by [JS91]; its data structure mainly consists of a rectangle array. The implemented layout algorithms change the rectangle orientation as soon as more vertical than horizontal space is given and vice versa. Colors are used to visualize the tree depth of a node. Interestingly, the algorithms needed for hierarchic visualizations – such as the TreeMap – are very similar to the applied query algorithms for traversing XPath axes. A sequential scan of the node table allows a straightforward calculation of the rectangles to be painted. The calculation is restricted to the given screen space, i.e., as soon as rectangles become too small to be displayed, they are skipped. As rectangles are added sequentially, they have the same order as the node

table. This implies descendants to always be arranged after their parents. This property comes in handy if, for instance, the according rectangle beneath the current mouse position is searched to be highlighted (see Fig. 2, left side). It is sufficient to traverse the array in reverse order and to terminate at the first rectangle which contains the coordinates of the mouse cursor as this is always the smallest rectangle displayed.

Two general interactions are offered to change the currently shown nodes. If a rectangle is selected via double click, a new array with its descendant nodes is calculated and displayed (zooming, shown in Fig. 2, right side). Secondly, a text field is offered in which tag names, text nodes and attributes may be entered by the user. The input is interactively converted to XPath after each key events, and the result set of the executed query is highlighted in the visualization. Content queries clearly profit from activated indexes. Next, a filter operation allows to recalculate the TreeMap for the currently highlighted nodes, and a history function, as integrated in common internet browsers, allows to return to the previously shown views.

4 Main Contributions

BaseX is a database prototype, working on a compact table based representation of XML documents. The fixed node size is equally suitable for main and secondary storage. Intentionally supposed to demonstrate the efficiency of the underlying data structure, the TreeMap additionally allows a visual access to the represented documents. With the integrated query interface, users can filter query results in interactive response time. As the applied encoding has also been designed in respect to minimum costs for updates, major future work will now be the integration of update capabilities and their visual support, realized as drag & drop operations. Moreover, the integrated text indexes will be enhanced to support a broader range of full text requests.

Literaturverzeichnis

[Bo02] Bohannon, P. et al: From XML Schema to Relations: A Cost-Based Approach to XML Storage. In ICDE, 2002

[Bo06] Boncz, P. et al.: MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In Proc. of the 25st ACM SIGMOD Conference on Management of Data, 2006

[Gr02] Grust, T.: Accelerating XPath Location Steps. In Proc. of the 21st Int'l ACM SIGMOD Conference on Management of Data, 2002

[Gr03] Grust, T. et al.: Staircase Join: Teach A Relational DBMS to Watch its (Axis) Steps. In Proc. of the 29th Int'l Conference on Very Large Databases (VLDB), 2003

[Gr06] Grün, C. et al.: Pushing XPath Accelerator to its Limits. In Proc. of the First International Workshop on Performance and Evaluation of Data Management Systems, ExpDB 2006, in cooperation with ACM SIGMOD, 2006

[FK99] Florescu, D. and Kossmann, D.: Storing and Querying XML Data using an RDMBS. In IEEE Data Engineering Bulletin, Vol. 22, Number 3, p. 27-34, 1999

[JS91] Johnson, B. and Shneiderman, B.: Tree maps: A space-filling approach to the visualization of hierarchical information structures. In Proc. of the 2nd International IEEE Visualization Conference, pages 284–291.IEEE ComputerSociety, 1991