

T-SIX: An Indexing System for XML Siblings

SungRan Cho
L3S, University of Hannover
scho@l3s.de

ABSTRACT

We present a system for efficient *indexed* querying of XML documents, enhanced with sibling operations. R-tree index proposed in [5] has a very high cost for the following-sibling and preceding-sibling axes. We develop a family of index structures, which we refer to as *transformed split-tree indexes*, to address this problem, in which (i) XML data is horizontally split by a simple, yet efficient criteria, (ii) the split value is associated with tree labeling, (iii) all data elements are transformed into new dimensions to avoid possible overlap between bounding boxes representing data elements in the split tree. The T-SIX system incorporates building transformed split-tree index for XML documents as well as query processing on all XPath axes to provide query answers.

1. INTRODUCTION

Efficient querying XML documents is an increasingly important issue considering the fact that XML becomes the de facto standard for data representation and exchange over the Web, and XML data in diverse data sources and applications is growing rapidly in size. Given the importance of XPath based query access, XML query evaluation engines need to be able to efficiently identify the elements along each location step in the XPath query. In this context, several index structures for XML documents have been proposed [4, 5, 7, 8, 10], in a way to efficiently querying XML documents.

As XML documents are modeled by a tree structure, a numbering scheme, labeling tree elements, allows for managing the hierarchy of XML data. For example, each element has the position, a pair of its beginning and end locations in a depth first search. In general, the numbering approach has the benefit of easily determining the ancestor-descendant relationship in a tree. In this respect, R-tree index using node's preorder and postorder, we refer to as *whole-tree indexes* (WI), has been proposed in [5]. Such index, however, does not consider issues related to the costs of the preceding-sibling and following-sibling axes.

In this paper, we develop index techniques to reduce the cost of XML siblings. It also addresses an issue of what efficient packing for XML tree data is. An efficient packing method for a tree

is not only to group together data elements which are close in a tree, but also to reduce dead space resulting in false positives (no data in indexed space). In the WI, packing method, taking a whole tree, may cover considerable dead space, which influences querying XML siblings. We design the *transformed split-tree index* to address the problem, in which (i) an XML tree is horizontally split by the simple, but efficient criteria, (ii) the split value is associated with tree labeling, (iii) all data elements are transformed into new dimensions to avoid possible overlap between bounding boxes representing data elements in the split tree. To take advantage of the semantics of the index structure, we develop novel index lookup algorithms for XPath axes for the transformed split-tree index.

We describe T-SIX, a system for indexed querying enhanced with XML sibling operations. T-SIX incorporates tree labeling and coordinate transformation for XML documents. T-SIX implements novel index lookup algorithms for querying, providing query answers for the entire set of XPath query axes.

This demonstration is organized as follows. Section 2 shows the architecture of T-SIX. Section 3 gives the structure of our index and describes functionality and features that T-SIX encompasses.

2. SYSTEM ARCHITECTURE

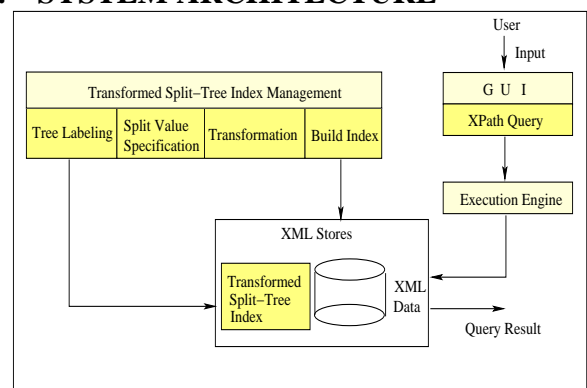


Figure 1: Architecture of T-SIX

T-SIX is a Java-based prototype. Its architecture is depicted in Figure 1. It consists of two main components: the *transformed split-tree index management* module and the *XPath querying* module. The first component implements various aspects of managing transformed split-tree index including tree labeling and coordinate transformation. It accepts the split value of a tree through a graphical interface and builds the index based on the value. The second

component implements a query processor on all XPath axes, that accepts queries and returns desired elements.

3. DEMONSTRATION OUTLINE

The system encompasses the following functionality and features that will be demonstrated: (a) facilitates browsing of different XML data sets, (b) facilitates browsing of element’s original and transformed positional numbers, (c) incorporates building transformed split-tree index for XML data, (d) incorporates novel index lookup algorithms in the transformed split-tree index for XPath query processing, (e) implements a flexible and interactive graphical interface and display of queries and query results, (f) supports flexible ways to input split value information for XML documents, and (g) supports adjusting page size parameter of the index in an interactive mode.

3.1 Transformed Split-Tree Index Management

Tree Labeling: We use an encoding scheme, L_n and R_n , for nodes in XML documents that has the same effect as preorder and postorder. L_n is the rank at which the node is encountered in a *left to right* depth first search (DFS) of the XML data tree, and R_n is the rank at which the node is encountered in a *right to left* DFS. In order to handle level sensitive matching, such as child and parent axes (matching nodes one level apart), and following-sibling and preceding-sibling axes (matching nodes with the same parent), the parent node’s L_n , written as PL_n is associated with each node. Thus each XML element node is labeled with three numbers: L_n , R_n , and PL_n . These numbers become coordinates in multi-dimensions. Users can view element’s positional information, L_n , R_n , and PL_n , through an interface. In Figure 2, an example XML database represents an *e-store* that contains information about items and clients.

Split Value Specification: XML documents can be selected and browsed in graphical form. XML element nodes are labeled with element tags or string values; edges are either between elements or between an element and a string value. Once XML document is selected, a user can specify the split value, which is associated with element’s L_n and R_n . In effect XML data tree is divided horizontally by the split value. T-SIX system provides conveniently visualize XML tree split. In Figure 2, for example, once a user submits the split value of 12, the *e-store* document is split by the value, in which *e-store*, *toys*, *CDs*, *pop* elements are cut in the XML tree and highlighted through a graphical interface. Split value information for XML documents can be dynamically modified on demand.

Transformation: While the separate packing reduces long thin boundary boxes, that may contain dead space (space which is indexed but does not have data), it causes the overlap between bounding boxes at each region of the tree. Due to overlap, multiple paths from the root downwards on the SI may need to be traversed, which results in increasing page accesses. Before building the index, T-SIX transforms coordinates of element nodes to avoid possible overlap. An element node $n=(L_n, R_n, PL_n)$ is transformed into $n'=(L_n', R_n', PL_n')$, such that

$$\begin{aligned} L_n' &= L_n \\ R_n' &= L_n + R_n \\ PL_n' &= PL_n \end{aligned}$$

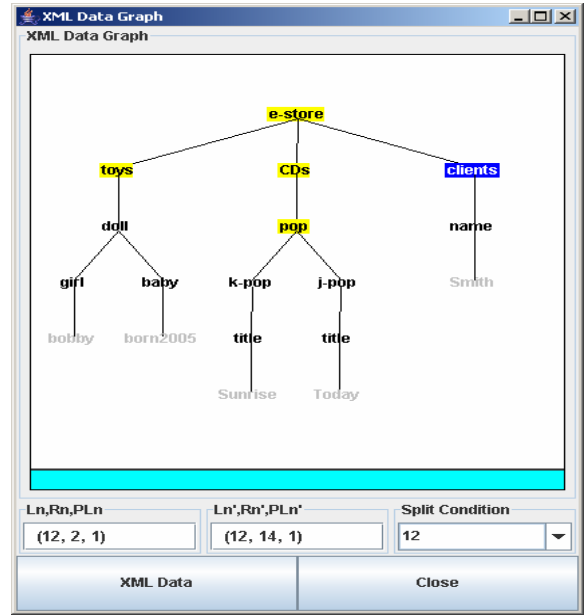


Figure 2: XML data

As a result, the original dimensions are extended with in the R_n direction with respect to L_n . Users can view the transformed coordinate information of elements.

Building Index: A transformed split-tree index is constructed on new dimensions (L_n' , R_n' , PL_n'). T-SIX supports for loading new documents to construct the index in the hierarchical structure. Transformed split-tree index contents as well as index operations can be efficiently visualized in the system. Users can input the page capacity through a graphical interface. Figure 3, for example, shows a transformed split-tree index over the split *e-store* dataset of Figure 2 with a page capacity of 2. The leaf pages in the index contain both leaf and non-leaf XML elements, and non-leaf index pages indicate page boundaries by the smallest and the largest values occurring in the page.

3.2 XPath Querying

Once an XML document to be queried is selected, users can specify XPath queries through a flexible graphical interface. After a user query is submitted, T-SIX uses the transformed split-tree index to return all relevant XML data. T-SIX system supports all XPath axes and provides various ways to conveniently visualize query results. In Figure 3, for example, a user issues a query, “retrieve all elements preceding the *clients* element”. As a result, elements returned as well as non-leaf pages scanned in the index are highlighted in the graphical interface. The XML data tree also highlights the query results for easy understanding of XPath query operations.

4. CONCLUSION

The T-SIX system provides XPath querying enhanced with sibling operations. It supports novel transformed split-tree indexing methods to facilitate query operations in XML documents and dynamic changes of split value information on the XML data tree. It accepts XPath queries and displays query results through a graphical user interface. Furthermore, it supports dynamic visualization of

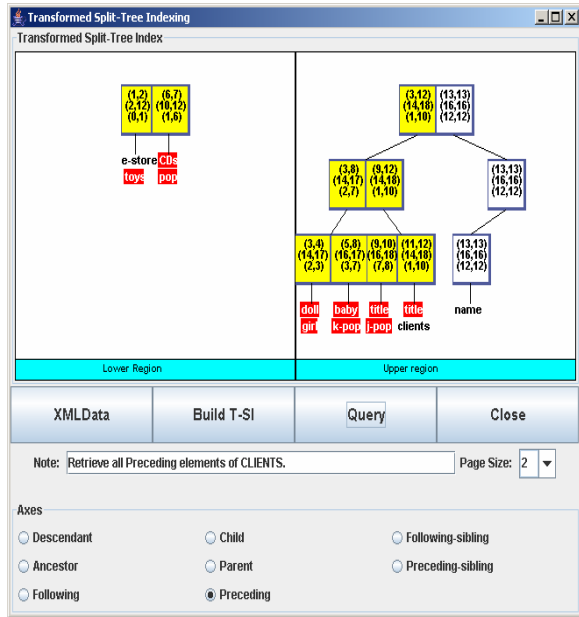


Figure 3: XPath querying

the index.

5. REFERENCES

- [1] M. Altinel and M. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proc. of VLDB*, Cairo, Egypt, 53–64, 2000.
- [2] J. Clark and S. DeRose. XML path language (XPath) version 1.0 w3c recommendation, Technical Report REC-xpath-19991116, World Wide Web Consortium, 1999.
- [3] E. Cohen, H. Kaplan and T. Milo. Labeling dynamic XML trees, In *Proc. of PODS*, 271–281, 2002.
- [4] B.F. Copper, N. Sample, M.J. Franklin, G.R. Hjaltason and M. Shadmon. A fast index for semistructured data, In *Proc. of VLDB*, Rome, Italy, 341–350, 2001.
- [5] T. Grust. Accelerating XPath location steps, In *Proc. of SIGMOD*, 2002.
- [6] A. Guttman. R-trees: a dynamic index structure for spatial searching, In *Proc. of SIGMOD*, 45–47, 1984.
- [7] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions, In *Proc. of VLDB*, Rome, Italy, 361–370, 2001.
- [8] T. Milo and D. Suciu. Index structure for path expressions, In *Proc. of ICDT*, Jerusalem, Israel, 271–295, 1999.
- [9] K.V. Ravikanth, D. Agrawal, A.E. Abbadi, A.K. Singh and T. Smith. Indexing hierarchical data, Univ. of California, CS-Tr-9514, 1995.
- [10] H. Wang, S. Park, W. Fan and P. Yu. ViST: a dynamic index method for querying XML data by tree structures, In *Proc. of SIGMOD*, San Diego, USA, 2003.
- [11] H.V. Jagadish, S. Al-Khalifa, A. Chapman, L.V.S. Lakshmanan, A. Nierman, S. Pappas, J.M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu, TIMBER: A native XML database, *VLDB Journal* 11(4): 274–291, 2002.
- [12] XQuery 1.0: An XML query language, W3C Working Draft, November 2002.
- [13] K. Deschler and E. Rundensteiner. MASS: A multi-axis storage structure for large XML documents, In *Proc. of CIKM*, Louisiana, USA, 2003.