Dynamic Containment Based Binary String Labeling Approach For XML Tree

Nyein Nyein Ei
University of Computer Studies, Mandalay
nyeinnyeinei@gmail.com

Abstract

The labeling scheme is used to label the nodes of an XML tree. Based on the labeling scheme, XML queries can be processed without accessing the original XML document. One more important point for the labeling scheme is to process updates when nodes are inserted into or deleted from the XML tree. In this paper, we propose a node labeling scheme that solves the weak points of the prefix, prime, interval and containment based schemes by supporting efficient insert processing. The weak points of previous schemes are re-labeling when subtree or node insert to XML tree, a large number of nodes that need to be relabeled in the case of an insertion of XML data, and inefficient processing of structural joins. To address this weak point, dynamic containment based binary string labeling approach for XML tree is proposed in this paper. This scheme uses containment based binary string for label dynamic XML tree and concatenation of binary string and character sequences for label insertion XML subtree.

1. Introduction

Extensible Markup Language (XML) nowadays is one of the most important standard media used for exchanging and representing data through the Internet. As for the labeling schemes, if XML is dynamic, how to efficiently update the labels of the labeling schemes is now becoming an important research topic [4, 5, 6]. This research can process the updates (inserts or deletes nodes) efficiently if the order of XML elements is not taken into consideration.

The method of assigning labels to the nodes of an XML tree is called a node labeling (or numbering) scheme. Based on the labels only, both ordered and un-ordered queries can be processed without accessing the original XML file. The core issue for XML query is to efficiently determine the following four basic relationships: ancestor-descendant (A-D), parent-child (P-C), sibling and ordering relationships. The existing node labeling schemes, i.e. containment, prefix and prime number schemes, are not efficient to determine all the four basic relationships. For instance, the containment scheme is very inefficient to determine the sibling relationship; it needs to search the parent of a node, and then decide whether another node is a child of this parent; the search of the parent needs a lot of parent-child relationship determinations which is very expensive. The prefix scheme is efficient to determine all the four basic relationships if the XML tree is shallow, however when the XML tree becomes deeper, the prefix scheme becomes not efficient because the labels of the prefix scheme become longer and the comparisons of node labels become expensive. The prime number scheme has very large label size and it employs the modular and division operations to determine the relationship which is expensive.

It is very important to maintain the document order when XML is updated; otherwise some semantics of XML will be lost and the ordered queries cannot be answered. It is very important to maintain the document order when XML is updated. All the existing node labeling schemes, i.e. containment, prefix and prime number schemes, have high update cost. The main contributions of this paper is the new scheme which does not need re-label any existing nodes and re-calculate any values when inserting an order-sensitive node into the XML tree.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes theory background, while Section 4 discusses proposed system architecture and then concludes the paper.

2. Related Works

To support efficient XML query processing, several node labeling schemes in the XML data tree were proposed. Several algorithms and index structures using these schemes for XML query processing have been studied.

Cohen et al presented algorithms to label the nodes of an XML tree which was subject to insertion and deletions of nodes. They labeled each node immediately when it was inserted and this label remained unchanged and from a pair of labels alone, they could decide whether one node is an ancestor of the other. They proved that their algorithm assign the shortest possible label which satisfy these requirements. They presented algorithms that used the nearly close nodes to assign shorter labels. They also proved that the length of their labels was close to the minimum possible [1].

OrdPath [7] is similar to DeweyID, but it only uses the odd numbers at the initial labeling. When an XML tree is updated, it uses the even number between two odd numbers to concatenate another odd. OrdPath wastes half of the total numbers. The query performance of OrdPath is worse since it needs more time to decide the prefix levels based on the even and odd numbers.

Wu et al. [10] proposed an approach to label XML trees with prime numbers shows Prime, in which the number above each node is the document order, the label is at the right side of each node, and the two numbers below each label are its parent_label and self_label. The root node is labeled with “1” (integer). Then based on a top-down approach, each node is given a unique prime number (self_label) and the label of each node is the product of its parent node’s label (parent_label) and its own self_label.

Agrawal et al [9] use a numbering scheme in which every node is assigned two variables: “start” and “end”. The insertion of a node may lead to a re-labeling of all the nodes of the tree. This problem may be alleviated if we increase the interval size with some values unused. However, it is not so easy to decide how large the interval size should be. Small interval size is easy to lead to re-labeling, while large interval size wastes a lot of values which causes the increase of storage.

They proposed the nested tree structure that eliminates the disadvantages and takes advantage of the previous node labeling schemes. The nested tree structure makes it possible to use the dynamic interval-based labeling scheme, which supports XML data updates with almost no node relabeling as well as efficient structural join processing. Experimental results show that their approach is efficient in handling updates with the interval-based labeling scheme and also significantly improves the performance of the structural join processing compared with recent methods. But their proposed algorithm could be more complex because of computation of space between start and end position. And their system is not fair to insert internal node in XML tree structure [11].

3. Theoretical Background

3.1. eXtensible Markup Language(XML)

XML is a set of rules for encoding documents in machine-readable form [12].The eXtensible Markup Language (XML) is a representation language as well as an exchange language. In the definition of XML, one element is allowed to refer to another; therefore theoretically an XML is a graph [13]. However for simplicity process queries over XML data those conform to an ordered tree-structured data model. With the tree model, data objects, e.g. elements, attributes, text data, etc., are modeled as the nodes of a tree, and relationships are modeled as the edges to connect the nodes of the tree. Without loss of generality, all queries are based on the ordered tree structured representation of XML data.

3.2. Labeling Schemes

Most node labeling schemes are based on the node-labeled data model. In a node-labeled data tree; there are two main objects, namely, nodes and edges. Nodes can be further classified into (1) Element Node (2) Attribute Node and (3) Value Node. Element Nodes correspond to the tags in the XML document, such as *book*, *title, and chapter* and so on. The Attribute Nodes and Value Nodes each correspond to attributes and data values in the XML document, such as the attribute '*id'* under the *book* elements and the value *'XML* Overview' between the *title* opening and closing tags respectively.

Labeling schemes have been developed to optimize query retrieval, since they provide a quick way to determine the type of relationships that are present among the nodes. According to [2], a labeling scheme for a document tree D is a decentralized structural summary of a specific set of tree relations in D. Each node in D is assigned a typically unique node label, so that any of these relations between the nodes in D can be inferred from their labels, without access to remote parts of D or to a global representation of the entire document tree. Edges in XML data trees represent structural relationships between data nodes. To answer XML queries, structural relationships, or more specifically reachability between any pair of nodes in XML data trees is compared. For example, in order to answer a path query 'A//B', given any pair of A-tagged node and B-tagged node, say (a, b), in a data tree, we need to determine whether there exists a path from a to b.

3.2.1. Subtree Labeling

This category is the simplest, where the label of a given document node v in D encodes the position and the extent of the subtree D_v of D that is rooted in v, by means of offsets in the sequence of nodes resulting from traversing (at least a part of) the document tree in a specific order [2]. While the exact representation of the subtrees varies accordingly, for the given nodes v, w in D, their Ascendant-Descendent and Parent-Child relationships are always determined by testing whether D_v contains D_w. The label of a node is usually concise in this group of labeling scheme. Nevertheless, performance degrades in an update intensive environment, as the labels usually need to be regenerated. The subtree labeling can be further broken down into three subclasses: interval encoding, containment encoding and region encoding. We will use containment encoding.

3.3. Binary String

In this section, we firstly introduce the definition of lexicographical order for binary strings each symbol of the binary string is stored with 1 bit [14]. A binary string is a sequence of bytes. We will define a binary string as a linear sequence of bipolar states. Zero and One (or One and Minus One) are the commonest form in which binary strings exist in computers. Presence or absence of anything could constitute a bipolar state and, as such, strings are widespread throughout real and imaginary spaces. In a computer, these values are stored as binary numbers, e.g. decimal number 5 will be stored as binary number 101.

4. Proposed System Architecture

In the system flow, the input data is XML document to be label and output is XML document with labeled nodes using lexicographical order with binary string. First, the system builds XML document as XML DOM tree structure. Second, the system labels to this XML tree with binary string using lexicographical order. The system label root node to leaf node go through down. If you want to insert leaf node or internal node or nested tree, the system labels to the nested tree or leaf or internal nodes as the second step. And then, the system concatenates the label of these inserted nodes with the Equation (1) character sequence. The system can avoid re-computation or relabeled the last labels in the tree by using this method.

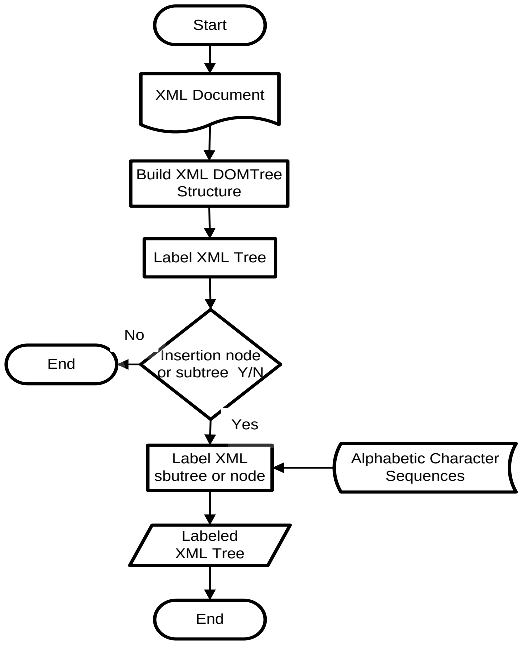
Start
XML Document
Build XML DOMTree Structure
Label XML Tree
Insertion node or subtree Y/N
No
End
Yes
Label XML sbutree or node
Alphabetic Character Sequences
Labeled XML Tree
End

Figure 1. System Flow of Labeling XML Tree

```
<books>
<book id="11210" category="fiction">
<author id="a1" sex="m">M. John</author>
<name>Computer Science 101</name>
</book>
<book id="11211">
<author>A. Mark</author>
<name>Applied Math 101</name>
<subject>Math</subject >
</book></books>
```

Figure 2. Example of XML document

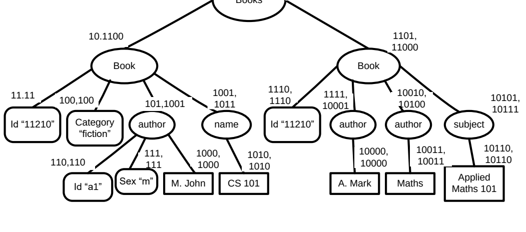

Figure 3. Example of labeling method

4.1. Insert processing

The XML data insertion can be processed by adding a subtree into the original XML data tree. For internal leaf node insertion, the system labels the node with right sibling's label concatenate the character from the following figure 4. For subtree insertion, the system labels this insert subtree from right sibling start label with lexicographical order.

```
Procedure Generating Label (T)
Input : XML document (D) with the root node(r) to be labeled
Output : XML Documents(D) with labeled nodes using lexicographical order for Binary String
1.  T is the tree
2.  Start s denotes the start of i^th node using lexicographical order for Binary String
3.  End e denotes the end of i^th node using lexicographical order for Binary String
4.  Begin
5.    For each node n of T do
6.        LV(I) = get the sequence to the concatenation of start and end in the lexicographical order
7.    End For
8.    If the nested tree exists to be insert
9.      Call the procedure InsertTree (ST, position)
10. End.
```

Figure 3. Proposed Node Labeling Algorithm

And then the whole inserted subtree labels with the concatenation from figure (4). The sequence can be expanded in both the directions left and right. By using this sequence the problem of re-labeling is avoided. The sequence will produce enough number of labels for any large size document. The combination of the upper and lower aliphatic characters is made efficiently. Also this proposed system will reduce the length of the labels. Using both lowercase and uppercase letters will produce a lot of label values with minimum storage. If the label size is reduced, the index size will automatically be reduced. Thus it will improve the performance of the query processing system. We propose an insert algorithm based on the nested tree structure to handle the above cases.

A;B;C; … Z : a; b; c; d; … ; z; Aa; … ; Zz; za: … ; zz; Azz; … ;Zzz; …

Figure 4. Alphabetic Character Sequences

We generate the above sequence of alphabetic character by combination of upper and lower alphabetic characters. The number to combination may be the following equation.

$$N = 52 + {}^{52}C_2 + {}^{78}C_3 + {}^{104}C_4 + ... + {}^{26n}C_n \qquad (1)$$

Where N = total number of combinations
n = number of alphabetic

The containment labeling scheme cannot support the dynamic update of XML data efficiently because it takes the sequential numbers as the labels of nodes and there is the interval property between the start and end positions for each node. When a new node is inserted, re-labeling of the existing nodes is indispensable. In order to solve this problem in Figure (6), it is possible to add one or more character with the label of these inserted subtrees for future XML data insertions.

```
<book>
<author>Ms. Smith</author>
<name>Computer Science 102</name>
</book>
```

Figure 5. Example of Insertion XML document

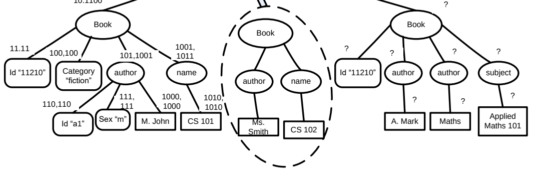

Figure 6. Insertion subtree problem

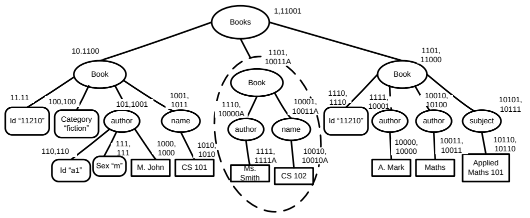

Figure 7. An example of nested tree structure

Procedure InsertTree(ST, position)

Input : XML document (D) with the root node(r) to be labeled

Output : XML Documents(D) with labeled nodes using lexicographical order for Binary String

```
1.  ST is the inserted subtree
2.  Position is the (start position, end position) pair of the node under which the subtree is inserted
3.  Start s denotes the start of ith node using lexicographical order for Binary String
4.  End e denotes the end of ith node using lexicographical order for Binary String
5.  Begin
6.    For each node n of ST do
7.      LV(I) = get the sequence to the concatenation of start, end and position in the lexicographical order
8.      Position is retrieved from the sequence in the Figure (4)
9.    End For
10. End.
```

Figure 8. Proposed Insertion Node or Subtree Labeling Algorithm

This system solve the weak point as a large number of nodes that need to be relabeled in the case of an insertion of XML data, huge space requirements for node labels, and inefficient processing of structural joins. The label of the existing tree does not need to re-label by joining the character and label of the inserted tree. Although XML tree is larger, node labels do not require the huge space due to our scheme uses only start and end position. In this system, the concatenation of the inserted tree is not adding any character to every node. Therefore, this system can get more efficient.

5. Conclusion and Future Works

When an order-sensitive node is inserted into the XML tree, the present node labeling schemes need to re-label the existing nodes or re-calculate some values to keep the document order which is costly in considering either the number of nodes for relabeling (re-calculation) or the time for re-labeling (re-calculation). To address this problem, we propose a node labeling scheme, which need not re-label any existing nodes and need not re-calculate any values when inserting order-sensitive nodes into the XML tree.

In the future, we will further study how to efficiently process the delimiters of this schemes and decrease the label size, as well keep the low label update cost. We will compare performance of our scheme with another scheme in the literature and analyst for our proposed system.

We will retrieve and query XML document stored in relational database. The system will translated XQuery queries into SQL statements. The retrieved results will reconstruct as XML hierarchical format and return to the user.

References

[1] E. Cohen, H. Kaplan and T. Milo, "Labeling dynamic XML trees", Proceedings of PODS.

[2] H. Su-Cheng, L Chien-Sing, "Node Labeling Schemes in XML Query Optimization: A Survey and Trends", Volume 26, Issue 2, 2009, pp. 88-100

[3] J. Paramasivam, T. Angamuthu, "An Enhanced Way of Labeling Nodes in Dynamic XML", European Journal of Scientific Research, ISSN 1450-216X Vol.55 No.3, 2011, pp.348-354

[4] J. Paramasivam, T. Angamuthu, "A New Scheme of Generating Persistent Labels for Dynamic XML Data", INTERNATIONAL JOURNAL OF COMPUTATIONAL COGNITION, Vol. 9, No. 1, March 2011

[5] J. Paramasivam, T. Angamuthu, "A New Method of Generating Index Label for Dynamic XML Data", Journal of Computer Science 7 (3): 421-426, 2011, ISSN 1549-3636

[6] L. Chang Qing and L. Tok Wang, "An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML"

[7] P. E. ONeil, et al., "ORDPATHs: insert-friendly XML node labels", International Proceedings of the ACM SIGMOD 2004, pp. 903–908.

[8] Q. Li, B. Moon, "Indexing and querying XML data for regular path expressions", International Proceedings of the VLDB 2001, pp. 361–370.

[9] R. Agrawal, A. Borgida, H. V. Jagadish, "Efficient Management of Transitive Relationships in Large Data and Knowledge Bases" SIGMOD Conference 1989: 253-262

[10] X. Wu, M. Lee, L. Hsu," A prime number labeling scheme for dynamic ordered XML trees", In: Proceedings of the ICDE 2004, pp. 66–78.

[11] Y. Jung-Hee Yun, C. Chin-Wan, "Dynamic interval-based labeling scheme for efficient XML query and update processing", The Journal of Systems and Software 81 (2008) 56–70.

[12] http : // en.wikipedia.org/wiki/XML

[13]http:// www. XML Query (XQuery) Requirements.htm

[14] http:// en.wikipedia.org / wiki / String_ (computer _ science)