A Dynamic Labeling Scheme Based on Logical Operators: A Support for Order-Sensitive XML Updates

Taher Ahmed Ghaleb*, Salahadin Mohammed*
King Fahd University of Petroleum and Minerals (ICS), Dhahran 31261, Saudi Arabia

Abstract

Dynamic XML labeling schemes have demonstrated important usages in XML Database Management Systems. Several researches have been conducted to provide dynamic labeling schemes that can efficiently process queries with less label size and space overhead in addition to the capability of processing order-sensitive updates effectively and efficiently (i.e., without re-labeling). In this paper, we introduce a dynamic labeling scheme as an enhancement to our previous static one, XDAS. Dynamic XDAS is developed as a hybrid labeling scheme that combines the original XDAS with another labeling scheme called IBSL. Dynamic XDAS conveys all characteristics of the original XDAS in addition to the efficient treatment with update processes with no re-labeling, which is adapted from IBSL. Like the original XDAS, our experiments show that dynamic XDAS still can identify the A-D, P-C and sibling relationships using logical operators with efficient label size and storage space. Moreover, dynamic XDAS is capable for processing node/subtree updates efficiently with completely avoiding re-labeling or re-calculation, just like IBSL.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
Peer-review under responsibility of organizing committee of the 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015)

Keywords: XML; Dynamic Labeling Scheme; XML Query processing; Order-Sensitive; XML Update Processing; XML Tree

1. Introduction

Recently, extensive research has been conducted to manage XML data, accelerate and facilitate query processing, and provide efficient data storage. With static XML documents, many labeling schemes in the literature can process queries efficiently. However, the most important research topic is when we deal with dynamic XML documents.

* Corresponding author. Tel.: +966-13-860-1721
E-mail address: {g201106210, adam}@kfupm.edu.sa
When dealing with dynamic XML documents, existing labeling schemes they either re-label all or some of the existing nodes in the XML tree, or recalculate some values when inserting a new order-sensitive subtree or leaf node into the XML tree. On the other hand, some of them do neither re-labeling for any existing nodes nor recalculation of any values, but with the cost of label size and mode disk space overhead.

*Label* is a unique identifier assigned for each node in the XML tree. This label holds important information about XML tree nodes, such as level, order and sequence in a way the position as well as the relationship of these nodes with other nodes can be identified. Each node in the XML tree can be an ancestor, parent, descendant, child, or sibling of another node in the XML tree. The Ancestor-Descendant (A-D), Parent-Child (P-C) and sibling relationships are the most important relationships a labeling scheme should provide.

In this paper, we introduce dynamic XDAS by improving of the original XDAS labeling scheme introduced in1 and disclosed in2. This improvement makes XDAS capable for dealing with dynamic XML documents by extending it with an approach for processing XML data updates. This approach is inspired from another labeling scheme called IBSL3. Dynamic XDAS combines the strengths of the original XDAS, including bits-masking and logical matching, with the power of IBSL dynamicity for update processing.

The rest of the paper is structured as follows: section 2 reviews some work proposed in the literature related to labeling both: static and dynamic XML documents. Then, in section 3, our labeling scheme, dynamic XDAS, is elaborated in detail. Our experiments and analysis of the results is demonstrated in section 4. Finally, section 5 concludes the paper and suggests the possible future work.

2. Related Work

Existing XML labeling schemes are usually categorized as Interval based, Prefix based and Prime based. Interval based or Range based labeling schemes identify each node with a label that consists of start, end and level number according to the pre-order traversal of the XML tree4,5,6,7. Prefix based labeling schemes preserve information of ancestor labels in their descendant labels using delimiters3,8,9,10,11. Prime based labeling schemes assign unique prime numbers to the XML tree nodes12. Recently, some hybrid labeling schemes have been introduced to combine the advantages of two or more approaches13.

Interval- or Range- based labeling schemes generate labels of the form \((\text{StartNo}, \text{EndNo}, \text{Level})\); e.g., \((12, 25, 4)\), for each node in the XML tree. It uses arithmetic integer-comparison operations to determine the relationships between nodes. Nevertheless, sibling relationship cannot be identified from labels themselves. This labeling scheme is not applicable for dynamic XML documents since all nodes must be re-labeled in case of insertion of a new node or a subtree occurs. This limitation could be solved by using large interval size. However, this would cause wasting many values, which leads to increasing the space. In6, an interval-based labeling scheme was proposed with the structure of nested trees. This new approach supported XML updates with almost no node re-labeling. In addition, it supported the integer-list comparison operations instead of the integer comparison operations.

In Prefix-based labeling scheme, labels are of the form \((A1.A2.P.C)\), e.g. \((1.15.3.9)\), where A1 is the first ancestor, A2 is the second Ancestor, P is the direct parent and C is the child. To say that the node X is a descendant of the node Y if the label of Y is the prefix of the label of X. This kind of labeling schemes usually uses string-matching operations to determine the P-C, A-D and sibling relationships between any two nodes. The very well-known labeling schemes from this category is the Dewey ID9 and Extended Dewey10, which are considered to be static. This means that they requires re-labeling the whole tree whenever a node is inserted. To support dynamic Dewey ID, other approaches, like DDE14, have been proposed as extensions to the original Dewey to completely avoid node re-labeling.

O'Neil et al.11 introduced OrdPath, which uses the Dewey order but in different form. It uses only odd integers in the initial labeling, of the form \((1.15.3.9)\). It reserves even and negative integers for later insertions. It encodes labels in a binary representation. However, the problem occurs when the size of the codes overflow, which means it, must re-label all the existing nodes. The overflow problem15 was also suffered by LSDX16 and SCOOTER8 labeling schemes. Thus, these labeling schemes are not preferred when XML documents have deep trees.

Prime-based labeling scheme12 uses prime numbers for labeling XML tree nodes. It is also able to determine the relationships between any two nodes from their labels. This scheme is suitable for dynamic XML data because it does not re-label the existing nodes, but instead, it re-calculates some values, called SC values, when insertion occurs. This re-calculation wastes a lot of time for update queries.
Ko and Lee\(^{3}\), proposed IBSL “Improved Binary String Labeling”. Their labeling scheme uses bit-strings of the form (011.01.0101), with full support for update without re-labeling or re-calculation. On the other hand, this scheme does not take advantage of binary numbers to do logical-matching; instead, it uses string matching in order to identify the relationships between two nodes, like Dewey-based labeling schemes. This is because, when IBSL was designed, it concentrated on how to make it efficient for dynamic XML documents. However, space overhead and label size are not efficient.

Our previous work\(^{1}\) proposed the XDAS labeling scheme as a first scheme that uses logical operators for matching operations that identify the relationships between two given nodes. However, this scheme is preferred only for static XML data, because it needs re-labeling when update operation occurs. Because, when XDAS was firstly developed, it concentrated on how to efficiently accelerate query processing in less label size and less space overhead.

3. The Proposed Approach

Most of the existing schemes either have efficient querying time but require expensive tree re-labeling when order-sensitive update operations occur or perform complex calculations to generate labels for the new inserted nodes. Other labeling schemes overcome this drawback and avoid re-labeling or performing calculations but the performance is affected.

Motivated by this, we propose Dynamic XDAS, which is based on the original XDAS labeling scheme\(^{1}\). Dynamic XDAS employs the IBSL approach for update processing, and apply it to XDAS. Consequently, our scheme is efficient in querying time, label size and space requirements like the original XDAS. Moreover, it is efficient in update processing like IBSL.

Both, IBSL and XDAS, use binary digits (0 and 1) to represent node labels but in two different ways. IBSL generates labels according to the lexical order in a Dewey format as shown in Fig. 1a. On the other hand, XDAS generates labels according to the masking technique as shown in Fig. 1b where the first part of XDAS label is the level and the second part is the unique ID generated using bits-masking.

3.1. Applying IBSL order-sensitive update approach to XDAS labels

IBSL has an efficient and effective technique to process order-sensitive update operations. We have modified this technique and utilized it to work with XDAS properly. The modified approach has three main cases as follows:

3.1.1. Insert a node before the leftmost node

a) First Case: the leftmost label does not contain an update ID:

In this case, the label of the inserted node is that the label of leftmost node concatenated with the delimiter concatenated with 01. For example, in Fig. 2a the leftmost has the label 1,001, so the label of the inserted node is going to be 1,001.01.

b) Second Case: the leftmost node's label contains an update ID:

In this case, the label of the inserted node is that the last bit of the label of the leftmost node is changed to “0” and then concatenated with “1”. For example, in Fig. 2b, the leftmost has the label 1,001.01, and then the label of the inserted node is going to be 1,001.001.
3.1.2. Insert a node after the rightmost node

a) First Case: the rightmost node's label does not contain update ID:

In this case, the label of the inserted node is that the label of rightmost node concatenated with the delimiter concatenated with 11. For example, in Fig. 3a the rightmost has the label 1,101, so the label of the inserted node is going to be 1,101.11.

b) Second Case: the rightmost node's label contains update ID:

In this case, the label of the inserted node is that the label of rightmost node concatenated with “1”. For example, in Fig. 3b, the rightmost node has the label 1,101.11, and then the label of the inserted node is going to be 1,101.111.

3.1.3. Insert a node between any two nodes at any position

In this case, the label of the inserted node depends on the size of the labels of the two neighbor sibling nodes. Fig. 4 shows the different cases might happen whenever a new node to be inserted at any position, as follows:

a) If the size of the left sibling node's label is less than or equal to the size of the right sibling node's label, then the label of the inserted node depends on the label of the right sibling node, as follows:
   - If the label of the right sibling node does not have update ID, then the label of the inserted node is that the label of the right sibling node concatenated with the delimiter concatenated with 01. For example, in Fig. 4a the left sibling node has the label 1,010 and the right sibling node has the label 1,011, so the label of the inserted node is going to be 1,011.01.
   - If the label of the right sibling node has update ID, then the label of the inserted node is that the last bit of the right sibling node is changed to “0” and then concatenated with “1”. For example, in Fig. 4b the left sibling node has the label 1,010 and the right sibling node has the label 1,011.01; so the label of the inserted node is going to be 1,011.001.

b) If the size of the label of the left sibling node is larger than the size of the label of the right sibling node, then the label of the inserted node is that the label of left sibling node concatenated with “1”. For example, in Fig. 4c the left sibling node has the label 1,011.01 and the right sibling node has the label 1,011, so the label of the inserted node is going to be 1,011.011.
3.1.4. Insert a subtree at any position of the tree

In this case, the root of the inserted subtree is labeled according to the previous cases. The other nodes of the subtree are labeled according XDAS bits-masking technique. Fig. 5 shows an example of inserting a subtree into XML tree.

![Diagram](https://via.placeholder.com/150)

**Fig. 5. Insert a subtree at any position of the tree**

3.2. Order of Dynamic XDAS Labels

The order of the original XDAS labels is preserved in dynamic XDAS labels. The labels of the new inserted nodes have the same XDAS IDs concatenated with Update IDs. Therefore, the order of the labels that have the same XDAS ID but different Update IDs is defined as the following order:

- Firstly, labels that have Update IDs starting with bit “0”. Note, All Update IDs that start with the bit “0” are ordered lexically.
- Secondly, labels that do not have Update IDs.
- Finally, labels that have Update IDs starting with a bit “1”. Note, All Update IDs that start with the bit “1” are ordered lexically.

This means, labels (L,X,0...) < (L,X) < (L,X,1...).

E.g., (1,011.001) < (1,011.01) < (1,011.0101) < (1,011.011) < (1,011) < (1,011.11) < (1,011.1101) < (1,011.111).

4. Experimental Results

In\(^1\), several experiments were conducted in order to evaluate the original XDAS labeling scheme relatively with Dewey based and Range based labeling scheme. Those experiments focused on label size, disk space requirements and the processing time. On the other hand, some experiments were conducted in\(^3\) to evaluate IBSL labeling scheme in comparison with other labeling schemes, such as Dewey, Range, Prime, etc., and additionally it focused on time needed to update XML document.

In our experiments, we select the evaluation criteria used our previous experiments with the original XDAS except the querying time, which is substituted by in this paper by the time needed to update the XML tree. We focused in these experiments on three prefix-based labeling schemes, Dewey, dynamic XDAS and IBSL.

4.1. Experimental Setup

Labeling schemes used in this paper have been implemented in Visual C# 2010 in order to generate labels, calculate label sizes, and measure the space overhead of each. We store labels in the file system where each unique element in the XML document has a file that contains the labels of all occurrences of that element.

4.2. Datasets used

In our experiments, three real-world XML documents, which are commonly used by researchers, have been used because they have different characteristics presented in Table 1. All our experiments rely on three commonly
used real-world XML documents that are available online in\textsuperscript{17}. These datasets have different characteristics in terms of depth, fan-out, and total number of nodes shown in Table 1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Topic</th>
<th>Max/average fan-out</th>
<th>Max/average depth</th>
<th># of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>XMark</td>
<td>25,500/3,242</td>
<td>12/6</td>
<td>1,666,315</td>
</tr>
<tr>
<td>D2</td>
<td>Treebank</td>
<td>56384/1623</td>
<td>36/8</td>
<td>2,437,666</td>
</tr>
<tr>
<td>D3</td>
<td>DBLP</td>
<td>328,858/65,930</td>
<td>6/3</td>
<td>3,332,130</td>
</tr>
</tbody>
</table>

4.3. Performance Evaluation of Label Size

We have generated labels to each dataset using Dewey, IBLS and dynamic XDAS. We have observed that Dewey generated labels faster than the others did, while dynamic XDAS took few seconds more. IBLS is the slowest one among the three, where it took much time, significant number of minutes under our environment, to generate labels for the dataset D3 that has the biggest number of nodes and fan-outs. After that, we have automatically counted the maximum and average label size (length) as shown in Fig. 6 and Fig. 7.

4.4. Performance Evaluation of Space Overhead

Fig. 8 shows the disk space required for storing labels of the generated labels of the three datasets using the three labeling schemes. As shown in the figure, dynamic XDAS is more efficient than others in terms of space overhead, while IBLS’s space overhead is too high.
4.5. Performance Evaluation of Query Processing

With respect to query-processing, dynamic XDAS is indeed similar to the original XDAS since the improvement is done only in regards to supporting updates. This improvement does not affect the overall performance of the original XDAS, which was validated in\(^1\) to be more efficient that Dewey.

On the other hand, IBSL is a Dewey-based labeling scheme. This means (as demonstrated in\(^3\)) that its performance in terms if querying time is slightly similar to that with Dewey. Therefore, we do not present any results and performance analysis in terms of querying time that evaluates dynamic XDAS in comparison with IBSL. We let this kind of performance analysis for future work since it requires extensive experiments through different path and twig queries that enable fair evaluation of both labeling schemes.

4.6. Performance Evaluation of Updating Processing

We have conducted a simple but not trivial experiment, a bit similar to that in\(^3\), to evaluate the time needed to label the newly inserted nodes in the XML tree. The XML file DBLP, D3, has 845 books. So we did four tests as follows: insert a new book before book(1), insert a new book between book(1) and book(2), insert a new book after book(845), and insert a chapter for the existing book(20).

In dynamic XDAS, for the four insertion test cases, only the inserted element is labeled. Nevertheless, in the first three tests, labels for the new elements are assigned using the update technique inspired from IBSL. For the last insertion, a label is assigned using the masking technique inspired from the original XDAS.

As demonstrated in Fig. 9, time needed to update the XML document using our proposed dynamic XDAS seems to be equal to that by IBSL. Both dynamic XDAS and IBSL took in average around 6.5 milliseconds to process the order sensitive update operations with slight difference in some cases. On the other hand, Dewey and the original XDAS took much time because that they re-label all the existing 3,332,130 elements in addition to the new ones.
5. Conclusion

In this paper, we introduce an improved labeling scheme to our previous scheme called XDAS. This scheme resolves the limitation of the original XDAS, which restricts it to work efficiently with static XML documents. Dynamic do support dynamic XML updates by combining the characteristics of both: the original XDAS (including query processing through logical operators) and the IBSL scheme (including processing updates efficiently). Like the original XDAS, our proposed scheme, dynamic XDAS, recognize the A-D, P-C and sibling relationships efficiently with efficient label size and less disk requirements. On the other hand, like IBSL, dynamic XDAS efficiently processes node/subtree updates with totally avoiding of re-labeling any existing nodes or re-calculating any values.

In future, we hope to improve dynamic XDAS by supporting the feature of reusing of the deleted labels in updates with both deletions and insertions. Furthermore, extensive experiments is going to be accomplished in terms of update and query processing under various kinds of datasets other than the ones discussed in this paper.

Acknowledgment

The first author would like to sincerely thank and appreciate his home institution, Taiz University - Yemen, which dons him a scholarship to continue his graduate studies abroad.

References