2012 4th International Conference on Computer Engineering and Technology (ICCET 2012)
IPCSIT vol.40 (2012) © (2012) IACSIT Press, Singapore

Siblings Labeling Scheme for Updating XML Trees Dynamically

Hamdi A. Al-Jamimi, Ahmed Barradah and Salahadin Mohammed

King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

{aljamimi, g199968940, adam}@kfupm.edu.sa

Abstract. Labeling schemes in XML trees 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. In order to efficiently determine structural relationships among XML elements and to avoid re-labeling for updates, much research about labeling schemes has been conducted, recently. Although, the recent proposed XML dynamic labeling schemes avoid re-labeling, sometimes they need to do a lot of computations to specify unique labels for the inserted nodes. In this paper, we review and compare a number of Dewey-based labeling schemes that support XML data updates. Several significant criteria have been identified for the evaluation and comparison. Moreover, we propose a novel dynamic labeling scheme, called *Siblings Labeling, for the XML trees*. The Siblings Labeling scheme is similar to Dewey coding, however it adds additional divisions to the node label to support the relationship retrieval perfectly as well as update XML tree when insertion with a very minimal re-labeling.

Keywords: XML, labelling scheme, XML storage, XML Query processing, XML indexing

1. Introduction

The eXtensible Mark-up Language (XML) has emerged recently as a new standard for the exchanging and publishing of data over the Web [1]. To facilitate the XML queries, two main techniques have been proposed structural index and labelling schemes. In that way, documents that follow the XML standard can be viewed as trees. XML database systems often give each item in the document (node in the tree) a unique logical identifier (called a label) and use these labels for an efficient processing of queries. Thus, the labelling schemes can be utilized to optimize query retrieval, since the structural relationships of the nodes, such as Parent-Child (P-C), Ancestor-Descendant (A-D) as well as document order, can be efficiently established by comparing their labels. Moreover, the labelling schemes lie at the core of query processing for many XML database management systems [2, 3]. XML data on the Web are subjected to frequent updates. Designing labelling schemes for dynamic XML documents is an important problem that has received a lot of research attention. Existing dynamic labelling schemes, however, often sacrifice query performance and introduce additional labelling cost to facilitate arbitrary updates even when the documents are seldom updated [4].

In this paper, we have two main contributions summarized as follows: First, based on significant evaluation criteria (presented in Section 3), we discuss the core idea of several Dewey-based labeling schemes, evaluate and compare them. Second, we propose a novel labelling scheme, *sibling labelling that* utilizes the Dewey coding. The proposed labeling scheme is designed to efficiently support queries for both static and dynamic XML documents. Even though the proposed scheme may require very minimal re-labeling when updating the XML documents, the structural relationships (A-D, P-C, siblings) of the nodes can be retrieved easily without overhead computations.

The rest of the paper is organized as follows. Section 2 reviews related work and introduce briefly the idea of each surveyed labelling scheme. Section 3 presents the evaluation criteria, while Section 4 compares

the different labelling schemes against each other. We describe our proposed labelling scheme in details in Section 5. Then, Section 6 concludes the paper and introduces the possible future work.

2. Background and Related Work

In this section, we introduce briefly the core idea of several Dewey-based labelling schemes including: DeweyID [5], OrdPath [6], TJFast [7], dynamic Dewey (DDE) [4], compact dynamic Dewey (CDDE) [4], and Dewey with even numbers [8].

Dewey ID: Tatarinov et al. [5] introduced Dewey coding into XML query processing such that each node is associated with a vector of numbers that represents the node-ID path from the root to the node. Dewey ID labels the nth child of a node with an integer n, and this n should be concatenated to the prefix (its parent's label). In practice, Dewey ID uses UTF8 to process the delimiter. When a node is inserted, Dewey ID needs to re-label the sibling nodes after this inserted node and the descendants of these siblings.

OrdPath: ONeil et al. [6] introduced OrdPath, which is a dynamic labelling scheme variant of the Dewey order. The label of each node is determined by the Dewey order scheme except that it reserves even and negative integers for later insertions into an existing tree. Also it stores the label of each node as the compressed binary representation and the ancestor–descendant or parent–child structural relationship between two nodes is determined by the substring comparison. Because of the reserved even and negative integers, almost no node re-labeling occurred for new data insertions. When the sizes of the OrdPath codes overflow which means it must re-label all the existing nodes.

TJFast: One of the other enhancement and extension for Dewey labelling scheme was proposed by Lu et al. [5]. All the elements names along the path from the root to the element can be derived from the label of an element alone. Based on extended Dewey they design a novel holistic twig join algorithm, TJFast. To answer a twig query, TJFast only needs to access the labels of the leaf query nodes. This reduces disk access and supports an efficient evaluation of queries with wildcards in branching nodes. However, this scheme is not suitable for the update processing because the labels of whole nodes and the finite state transducer must be reconstructed after data insertions.

Dewey with even numbers: Theo Harder et al. [8] suggest an insert-friendly labeling scheme based on Dewey IDs. Each label is composed of divisions that reflect the element id as well as its parents' IDs. In the initial document labeling, only odd numbers are used with some gaps (they used a gap of 4) between the numbers to facilitate the insertion of new labels. In case there is no space between the elements for insertion, a new division is introduced with an even number which is not counted when computing the level of the element.

DDE: A DDE [4] is tailored for both static and dynamic XML documents. For static documents, the labels of DDE are the same as Dewey which yield compact size and high query performance. Unlike containment labels which have level fields, the level information is implicitly represented by the label. A DDE label implicitly stores the level information as the number of divisions in that label. This property will remain true after random insertions and deletions. When updates take place, DDE can completely avoid re-labeling and its label quality is most resilient to the number and order of insertions compared to the existing approaches. Fig.2 demonstrates processing insertions with DDE labels.

CDDE: CDDE [4] is designed to optimize the performance of DDE for insertions. The label format of CDDE is the same as DDE which is a sequence of divisions separated by '.'. Moreover, the initial labeling of CDDE is the same as DDE. Unlike DDE labels whose first divisions are restricted to be positive decimal numbers, the first division of a CDDE label can be either positive or negative. We refer to the CDDE labels with positive first divisions as positive CDDE labels and those with negative first divisions as negative CDDE labels. Fig.3 shows the insertions process with CDDE labels.

3. Evaluation Criteria

Since different labelling schemes have commonalities and viabilities, we attempt here to identify several evaluation criteria to evaluate and compare the various properties of each labelling scheme.

- *Storage Requirement*- reflects the amount of storage space required to store the numbers or the indexes. Also, it represents how efficiently a numbering scheme copes with the growth of the number of nodes. This is because in most cases the index/number size is a function of the number of nodes.
- *Supported Axes*- represents the types of relationships between any two nodes that can be derived from the numbering scheme
- *Efficiency of Relationship Retrieval*- represents the amount/type of calculation or comparisons required to identify the relationships between two nodes given their indexes.
- *Update Efficiency*- indicates how efficient a numbering scheme is in the face of updates (e.g. insertion, deletion); and whether or not it requires re-labeling of tree nodes.

4. Comparison and discussion

Storage Requirement

In general, the Dewey-based schemes have the same tactic when dealing with initial labeling or static tree. The label division's number of a particular node is equal to its level. These schemes differ when update XML tree after insertions. In Dewey-based schemes the length of the label representation becomes longer by frequent data insertions. For instance, in Dewey with even numbers labeling scheme the number of divisions in the labels can grow with the horizontal and vertical growth of the xml tree.

Supported Axes

The different studies labelling schemes support almost all the common axes such as P-C, A-D and sibling relationships and document order. That is, it is possible to determine if a given label A is the parent/ ancestor/ sibling/ preorder of another label B. In addition, it is possible whether label A precedes another label B in document order. Moreover, some of these schemes like DDE, CDDE can support another axes like Lowest Common Ancestor (LCA).

Efficiency of Relationship Retrieval

In the most surveyed schemes, such as Dewey ID, OrdPath, TJFast, and Even labeling schemes, only simple comparisons are required to retrieve the relationship between any two given indexes the same as when dealing with initial Dewey coding. In DDE and CDDE, verifying the relationship between given nodes require a lot of calculations.

Table 1: Evaluation Summary of the Dynamic XML Labeling Schemes

Scheme	Storage Requirement	Supported Axes	Efficiency of Relationship Retrieval	Update Efficiency
Dewey ID	Medium	Common axes	Simply retrieved	re-label the sibling nodes
DDE	Medium	common axes	Perform calculations	No re-labeling
CDDE	Medium	common axes	Perform more calculations	No re-labeling
OrdPath	High	A-D, P-C	Simply retrieved	Re-labeling
TJFast	Medium	A-D, P-C	Simply retrieved	Re-labeling
Even-labeling	High	Common axes	Directly	Scan and perform calculations

Update Efficiency

The need for XML tree updating arises when deleting existing nodes or inserting new nodes. The deletion of labels does not affect the order of other labels while the insertions without re-labelling nodes are not easy task. Essentially, the insertion could be leftmost, rightmost or between consecutive siblings. The insertion between consecutive nodes vary form scheme to another. Although in the Dewey-based schemes do not require re-labeling, they might require loading both the previous and following siblings' indexes and doing some computations. This kind of operations can be very costly especially when the indexes are huge. In Table 1, we summarize the evaluation of the studied dynamic XML labeling schemes according to the considered criteria (introduced in Section3).

5. New approach: Sibling-labeling scheme

One of the most important qualities of any re-labeling scheme is its behavior upon tree insertion or deletion. Most of the techniques in the literature either require massive tree re-labeling after intensive update operations or involve complex calculations to generate the new labels. Dewey encoding supports all the common axes as well as retrieves the relationship between given nodes easily form their labels. However, one huge drawback of Dewey labelling is that it certain update operations are very costly and might require the complete re-labelling of the whole tree (except the root). Other Dewey-based schemes, as discussed in Section 4, overcome this drawback and avoid re-labelling but retrieving the different axes is non-trivial task and needs to perform some computations that may negatively affect the performance.

Motivated by this, we suggest the sibling-labeling scheme that is based on Dewey encoding. Our technique avoids complex operations and requires at most the re-labeling of two adjacent nodes. Thereby, the proposed sibling-labeling scheme can reduce the drawbacks related to the Dewey-based schemes significantly. The basic idea is to add two divisions to the indexes in order to reflect the preorder numbers of the next and previous siblings. This is shown in Figure 1; the root node is labeled 0.0.1 which means that it is the first node in the level and does not have any siblings. The first child is labeled 0.2.1.1 where the last two divisions are the Dewey label of the node and the first two divisions are the preorder numbers of the previous and next siblings respectively. This way, only the second division of the previous node and the first division of the next node need to be updated in the worst case. This means the maximum number of nodes re-labeling required is 2. Figure 1 shows all the possible cases of insertion. In the first case, the node is inserted as the last child (right most) and only the second division of the previous node needs to be updated. Figure 1b shows the worst case scenario where the node is inserted between two existing nodes in which case only 2 re-labeling operations are required, the second division of the previous node and the first division of the next node which both point to the preorder of the newly inserted node.

In terms of update efficiency, the sibling-labeling is more efficient than the Dewey labeling scheme in the sense that it never requires the update of more than two nodes. The only situation where the Dewey labeling scheme requires no updates is during the right most insertion in which case a single update is needed by the sibling-labeling scheme. Of course this is if we exclude the single childe insertion because none of the techniques should require re-labeling in this case. In comparison to the even-labeling [8] the sibling-labelling performs very closely. This is due to the fact that in both techniques both the next and previous node indexes are read. As for the even-labeling, it might need to scan many divisions of these two indexes and then apply certain rules and sometimes perform certain operations to generate the new label. For example, if n_1= 1.9.5.7.5 and n_3= 1.9.5.7.16.5 then n_2 will have the label 1.9.5.7.11 when inserted between n_1 and n_3. This label is generated by scanning n_1 and n_3 and taking the median of the first differing divisions (5 and 16) to generate the odd number 11. This kind of operations can be very costly especially when the labels are huge. The sibling-labeling scheme on the other hand will scan at most the first two divisions of the previous and next indexes and perform the update operation on one of these divisions as mentioned earlier. As for the storage efficiency, the sibling-labelling is slightly inferior to the Dewy labelling because the sibling-label will always have two more divisions than the normal Dewy label. Compared to the even-labelling scheme, our sibling-labelling scheme can be more costly in case the tree has not undergone any updates. This is because

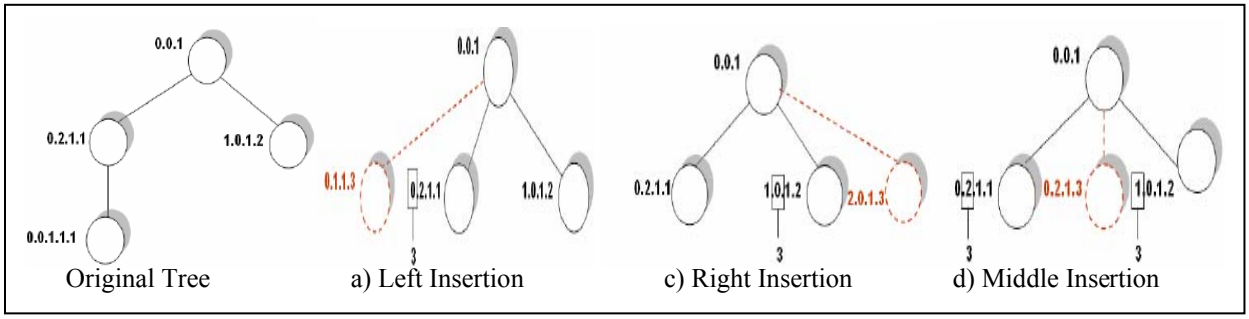

Fig.1:All the cases of insertion in the sibling-labeling.

in this case, the even-labeling is identical to the Dewy labeling in terms of storage requirements. After massive updates on the other hand, the sibling-labeling outperforms the even-labeling since the indexes in the even-labeling can grow (get inflated) both horizontally (insertion at an existing level) and vertically (as the height increases). For instance, if n nodes are to be inserted before the node d_1= 1.5.2.1 and d_1 is the first child, then with the even-labeling the nth node will have a label that consists of n+1 2s (1.5.2.2.2...n+1.[odd number]). Also, there are no restrictions on the number of even divisions that can be added to the label nor there is a restriction on how big the even number is. Tables 2 and Table 3 summarize the above discussion.

Table 2: Storage Requirements

Scheme	Initial tree	Initial tree
Dewey	Length(dewey_label)	Length(dewey_label)
Even-labeling	Length(dewey_label)	Length(dewey_label) + N
Sibling-labeling	Length(dewey_label) + 2	Length(dewey_label) + 2

Table 3: Update Efficiency

Scheme	Left insertion	Right insertion	Middle insertion
Dewey	Re-label all the following and their siblings	No re-labeling	Re-label all the following and their siblings
Even-labeling	Scanning and perform calculations on next sibling but no re-labeling	No re-labeling	Scanning and perform calculations on next & previous siblings but no re-labeling
Sibling-labeling	1	1	2

6. Conclusion and Future Work

In this work, we reviewed, evaluated and compared a number of dynamic labeling schemes that support XML data updates. Specifically, we considered deferent labelling schemes that based on Dewey encoding. These labelling schemes can be considered superior in terms of update efficiency; however, they are not prefect in terms of storage requirements. Additionally, we proposed a novel labeling scheme, sibling-labeling scheme which is based on the famous Dewey coding. By using our scheme two nodes at most can be re-labeled. Nonetheless, sibling-labeling is very perfect to support the common axes and retrieve the relationships between the given nodes straightforwardly. As future work, we plan to develop our own Dewey compression mechanism or test some of the existing compression techniques in order to reduce the label storage overhead.

7. Acknowledgments

The authors would like to thank KFUPM for supporting this research.

8. References

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible markup language (XML) 1.0 third edition W3C recommendation. 2000.

[2] A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, and J. Simon. XML path language (XPath) 2.0, W3C working draft 16. Technical ReportWD-xpath20-20020816, World Wide Web Consortium,2002.

[3] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simon. XQuery 1.0: An XML Query Language, W3C working draft 16. Technical Report WD-xquery-20020816, World Wide Web Consortium, 2002.

[4] Liang Xu, Tok Wang Ling, Huayu Wu, Zhifeng Bao. DDE: From Dewey to a Fully Dynamic XML Labeling Scheme. *In SIGMOD Conference*, 2009, pp719-730.

[5] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. *In Proc. Of SIGMOD*, 2002, pp. 204-215.

[6] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML node labels. *In Proceedings of the ACM SIGMOD*, 2004, pp. 903- 908.

[7] J. Lu, Ling, T.W., Chan, C., Chen, T. From region encoding to extended Dewey: on efficient processing of XML twig pattern matching,. *In: Proceedings of the VLDB*, 2005, pp. 193–204.

[8] T. H¨arder, M. Haustein, C. Mathis, M. Wagner. Node Labeling Schemes for Dynamic XML Documents Reconsidered. Data & Knowledge Engineering, 2006.