Journal of Computer Science 6 (12): 1535-1540, 2010
ISSN 1549-3636

Controlling Label Size Increment of Efficient XML Encoding and Labeling Scheme in Dynamic XML Update

Meghdad Mirabi, Hamidah Ibrahim, Ali Mamat, Nur Izura Udzir and Leila Fathi
Department of Computer Science,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 Serdang, Selangor, Malaysia

Abstract: Problem statement: In order to facilitate XML query processing, labeling schemes are used to determine the structural relationships between XML nodes. However, labeling schemes have to reliable the existing nodes or recalculate the label values when a new node is inserted into the XML document during XML update process. EXEL as a labeling scheme is able to remove relabeling for existing nodes during XML update process. Also, it is able to compute the structural relationship between nodes effectively. However, for the case of skewed insertions where nodes are always inserted at a fixed place, the label size of EXEL scheme increases very fast. **Approach:** This study discussed how to control the increment of label size for the EXEL scheme. In addition, EXEL does not consider the process of deleting labels. We also study how to reuse the deleted labels for future label insertions. **Results:** We proposed an algorithm which is able to control the label size increment. **Conclusion:** It required less storage size to store the inserted binary bit string and thus can improve query performance.

Key words: Bit string, reuse of deleted label, skewed insertion, XML relabeling

INTRODUCTION

XML (Bray *et al*., 2006) has been proposed as a de facto standard to represent and exchange the data on the Internet. Generally, XML documents can be represented as an XML tree or XML graph. Elements in XML document can be labeled based on the structure of XML document to facilitate XML query processing. In order to improve the XML query processing time, the structural relationships between XML nodes must be determined. In other words, XML query processing requires the information of the structural relationships among XML nodes. The basic structural relationships are Parent-Child (P-C) and Ancestor-Descendant (A-D) and the core operation of XML query processing is to find all occurrences of structural relationships in an XML document. However, labeling schemes have to re-label the existing nodes or recalculate the label values when a new node is inserted into the XML document in dynamic XML update process. Recently, more researches are focused on how to update the labels when nodes are inserted into the XML tree (Min *et al*., 2007; 2009; Wu *et al*., 2004; Amagasa *et al*., 2003; O'Neil *et al*., 2004; Li and Ling, 2005; Li *et al*., 2006a; 2006b; 2008; Li and Moon, 2001; Yun and Chung, 2008; Ko and Lee, 2006; 2010). However, how to process the deleted labels is a new challenge in dynamic XML update (Li *et al*., 2006b; 2008; Yun and Chung, 2008; Ko and Lee, 2006; 2010).

In dynamic XML updating process, one of the important issues is the label update cost in inserting and deleting a node into or from the XML tree. Thus, the maintenance of the XML document order is very important when update is performed. Several researches have been suggested to solve the problem of relabeling the existing nodes in dynamic update process of XML (Min *et al*., 2007; 2009; Wu *et al*., 2004; Amagasa *et al*., 2003; O'Neil *et al*., 2004; Li and Ling, 2005; Li *et al*., 2006a; 2008; 2006b; Li and Moon, 2001; Yun and Chung, 2008; Ko and Lee, 2006; 2010).

Efficient XML Encoding and Labeling (EXEL) (Min *et al*., 2007; 2009) as an insert-friendly order-based bit string labeling scheme is able to remove relabeling for existing nodes during XML update process. Also, it is able to compute the structural relationship between nodes effectively. Thus, we can use EXEL to make a label for a new inserted node in XML tree without violating the ordering of the indexed and encoded nodes of XML tree. However, for the case of skewed insertion where nodes are always inserted at

Corresponding Author: Meghdad Mirabi, Department of Computer Science, Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 Serdang, Selangor, Malaysia

a fixed place, the label size of EXEL scheme increases very fast. This study discusses how to control the increment of label size for the EXEL scheme. In addition, EXEL does not consider the process of deleting labels. We also study how to reuse the deleted labels for future label insertions to control the label size increment and improve the query performance.

Related works: Wu *et al*. (2004) have proposed a scalable prime based labeling scheme which uses the property of prime number to label the XML nodes. Each node is labeled by an integer which can only be divided exactly by its own ancestor label in XML tree. The structural relationship between nodes in this scheme depends on whether the label of a descendant node is divisible by the label of an ancestor or not. Prime number labeling scheme uses the Simultaneous Congruence (SC) values in Chinese reminder theorem to decide the document order. However, prime needs to recalculate the SC values based on the new ordering of the nodes.

To solve the relabeling problem of region number labeling scheme, Amagasa *et al*. (2003) have extended a region by using float-point values for the start value and end value of intervals. However, this solution is unable to remove the relabeling in the case of frequent insertions.

ORDPATH as a prefix insert-friendly XML node label scheme (O'Neil *et al*., 2004) is able to insert nodes at any position of XML documents. ORDPATH is similar to the Dewey labeling scheme. It only uses odd numbers during initialization of labels. Even and negative numbers are reserved for later insertion into XML tree. Also, due to compressed binary representation of node labels in ORDPATH, the structural relationship between two nodes is determined by the substring comparison. It does not almost need to re-label existing nodes during node insertion process but binary representation length of labels is large and becomes longer by data insertion frequently. In addition, ORDPATH has the problem of skewed insertion.

QED (Li and Ling, 2005; Li *et al*., 2008) and CDBS (Li *et al*., 2006a) remove the need of relabeling the nodes when the XML document is updated. In addition, they can be applied to different labeling schemes which need to maintain the order. Most important feature of QED and CDBS is that the labels are compared based on lexicographical order rather than numerical order. The main problem of CDBS is overflow. If the numbers of inserted nodes are large, the length field size is not enough for new label. However, relabeling all of existing nodes is required. Even by increasing the size of length field, it still cannot able to remove relabeling completely and it will waste storage space. This problem is called overflow problem. In addition, in contrast with QED which the last 2 bits of the neighbor label must be modified, only the last 1 bit needs to be modified in CDBS. Thus, update cost of CDBS is smaller than update cost of QED.

In addition, deleted labels can be reused during insertion operation of nodes to decrease the storage size cost and improve XML query processing performance. Li *et al*. (2006b) have proposed an algorithm to reuse deleted labels and control the increment of storage size when nodes are inserted into and deleted from XML document frequently. Relabeling the existing nodes is not required in the proposed algorithm. The QED (Li and Ling, 2005; Li *et al*., 2008) labeling approach is unable to guarantee inserting the labels with smallest size when some labels are deleted. Li *et al*. (2006b) have suggested a reuse algorithm to modify QED labeling approach to find the smallest label lexicographically between two labels.

In order to overcome the overflow problem of CDBS, Compact Dynamic Quaternary String (CDQS) (Li *et al*., 2008) encoding approach is devised which is able to remove relabeling in updating the leaf node completely. CDQS also can be applied into different labeling schemes like CDBS and QED. In addition, Li *et al*. (2008) have proposed some techniques to update interval nodes efficiently but it is not able to completely remove the relabeling in interval node updates. In addition, to reuse all CDQS deleted labels, a new algorithm is devised. One solution to label the XML document which can remove relabeling in updating process is to leave some unused values for future insertion (O'Neil *et al*., 2004; Li and Moon, 2001). However, when the unused values are used up later, they have to re-label the existing nodes, the proposed algorithms (Li and Ling, 2005; Li *et al*., 2006a; 2006b; 2008) do not need to leave some unused values for future insertion.

The interval property of node labels in region number labeling scheme causes relabeling the existing nodes in XML update. Although reserving space for future node insertion is as a solution to avoid relabeling in existing nodes, when a large number of data is inserted relabeling is required (O'Neil *et al*., 2004; Li and Moon, 2001). If we can process a large XML insertion with a small space, we are able to solve the problem of relabeling in existing node of XML tree. Yun and Chung (2008) have devised the Nested Tree Structure according to this motivation to avoid relabeling for interval based number labeling schemes in updating process. In this approach, each XML

element is considered as an XML data update unit and is expressed by a sub-tree. In order to label the sub-tree, each node in the sub-tree is labeled by new number and then the sub-tree is labeled as a leaf node of the XML tree. Thus, the structural relationship between a node in the inserted sub-tree and other nodes which are not in the inserted sub-tree is determined by comparing the label of sub-tree and the label of other nodes. In order to obtain the structural relationship between nodes in the inserted sub-tree, the new labels which are marked in the sub-tree are used. This approach is called Nested Tree Structure because if the data insertion occurs in the previous inserted sub-tree, a new sub-tree is formed in the inserted sub-tree. As the data insertions like this occur continually, the structure of the whole of tree is nested by sub-trees. In addition, Yun and Chung (2008) have proposed an algorithm to release nested trees in the process of sub-tree deletion as much as possible.

IBSL as a binary string based prefix scheme (Ko and Lee, 2006; 2010) is able to remove relabeling and recalculation in XML updating process. Also, in order to handle reusability of the deleted labels, Ko and Lee (2010) have proposed an algorithm to decrease the cost of storage size when the large number of labels are inserted and deleted without worrying about degradation of query performance. An algorithm for inserting a sibling as well as sub-tree into XML tree without the need to relabeling the nodes are proposed in (Ko and Lee, 2006; 2010) but the proposed algorithm does not support inserting a node as a parent into XML tree like (Min *et al*., 2007; 2009; Li and Ling, 2005; Li *et al*., 2006a; 2006b; 2008).

EXEL (Min *et al*., 2007; 2009) removes relabeling the nodes for updating. EXEL is able to insert a sibling or a parent as well as a child into the XML tree without the need to relabeling the nodes. EXEL can save time in update operations because of complete avoidance of relabeling the XML tree. However, the problem of EXEL is the increment of label size when node insertions and deletions are performed frequently.

MATERIALS AND METHODS

Here we present the EXEL scheme and show how update to the XML can be done without relabeling the existing nodes and then our proposed algorithm is presented to control the label size increment of the EXEL scheme for the case of skewed insertions and process of reusability of the deleted labels.

EXEL labeling and encoding scheme: According to (Min *et al*., 2007; 2009), EXEL as an insert friendly bit string order based labeling scheme is able to remove relabeling for existing nodes during XML update process. Also, it is able to compute the structural relationship between nodes effectively. Thus, we use EXEL to make a label for a new inserted node in XML data without any violating on the ordering of the existing nodes of XML data. EXEL uses bit string to encode the XML data. This bit string is ordinal as well as insert friendly. The definition of lexicographical order (<) of bit string is defined as follows:

Lexicographical Order (<):

- 0 is smaller than 1 (0<1) lexicographically
- Bit string a is equal to bit string b lexicographically, if a and b are the same (a = b)
- For bit strings α_1, α_2, b_1 and b_2, $\alpha_1 b_1 < \alpha_2 b_2$, iff ($\alpha_1 < \alpha_2$) or ($\alpha_1 = \alpha_2$ and $b_1 < b_2$) or ($\alpha_1 = \alpha_2$ and b_1 is null (empty string)), where length (α_1) = length (α_2)

According to the above definition, for each bit string s which ends with '0', the largest bit string among bit strings which are smaller than s lexicographically is the s's longest prefix p (i.e., s = p0). However, we cannot generate any bit string which is greater than the prefix p and smaller than s. For example, there is not any bit string which can be inserted between '1110' and its longest prefix bit string '111'. Thus, if the last bits of any two consecutive bit strings are '1', we can insert a new one between the bit strings without any changes on them.

The key idea to remove relabeling during updating process of a node in the XML tree is property 1.

Property 1: For two bit strings a1 and b1, if a1<b1 lexicographically, then a < b lexicographically.

The algorithm of generating the bit string for nodes is shown in Fig. 1 which is the enhanced binary encoding algorithm in (Min *et al*., 2007; 2009). This algorithm obeys the property 1.

In order to encode N ordinal values, the bit string generation algorithm needs $[(\log_2^N)+1]$ bits for each bit string. Thus, the total size for encoding N value is N $[(\log_2^N)+1]$ for example, in order to encode 18 values, the size of the longest bit string is 6 and the total size is 6×18 = 108. Table 1 shows the enhanced bit string encoding of 18 numbers based on bit string generation algorithm.

Let N be the total number of nodes of XML tree,
For first bit string b (1), $b(1) = (0^{\log_2^N})1$.
For (i+1) th bit string b (i+1), b (i+1) = b (i) + 10.

Fig. 1: Algorithm of bit string generation

```
MakeNewBitString (LeftB, RightB)
Begin
  If length(LeftB) > length(RightB) then
    newB:= LeftB concatenate with 1;
  Else
    newB:= (RightB with last bit changed to 0)
            concatenate with 1;
  Return newB;
End
```

Fig. 2: MakeNewBitString algorithm

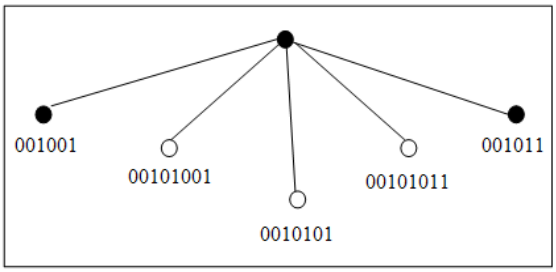

Fig. 3: Insertion of a new node between two existing nodes

Table 1: Enhanced binary encoding scheme in EXEL for 18 numbers

Decimal number	Bit string
1	000001
2	000011
3	000101
4	000111
5	001001
6	001011
7	001101
8	001111
9	010001
10	010011
11	010101
12	010111
13	011001
14	011011
15	011101
16	011111
17	100001
18	100011

EXEL uses MakeNewBitString algorithm which is shown in Fig. 2 to make a new bit string between two preexisting bit strings.

Example 1: As shown in Fig. 3, assume that we use MakeNewBitString algorithm to make a binary bit string between two existing nodes with binary bit strings “001001” and “001011”, the inserted binary string is “0010101”. We cannot find any other binary bit strings which are ended with “1”, are between “001001” and “001011” lexicographically with the small size.

```
ModifiedMakeNewBitString (LeftB, RightB)
Begin
  temp1:= LeftB +10;
  temp2:= RightB – 10;
  if (LeftB < temp1) and (temp1 < RightB) then
          newB:= temp1;
  else if (LeftB < temp2) and (temp2 < RightB) then
          newB:= temp2;
  else
    If length(LeftB) > length(RightB) then
      newB:= LeftB concatenate with 1;
    else
      newB:= (RightB with last bit changed to 0)
                concatenate with 1;
  Return newB;
End
```

Fig. 4: ModifiedMakeNewBitString algorithm

If we want to generate a binary bit string between “001001” and “0010101”, the binary bit string generated by the algorithm is “00101001”. Also, the binary bit string between “0010101” and “001011” generated by the algorithm is “00101011”.

According to the example 1, it is observed that for each binary bit string insertion, the size of bit string increases 1 bit. In other words, the label size increases linearly (O(N)) using MakeNewBitString algorithm.

The proposed algorithm: The MakeNewBitString algorithm generates a new binary bit string by increasing 1 bit in the label size. In order to control the label size increment, we modify the MakeNewBitString algorithm. The proposed algorithm which is shown in Fig. 4 is a modified version of the MakeNewBitString algorithm.

Example 2: Assume that we want to insert a new node between two nodes which are labeled with “001001” and “001011” as shown in Fig. 5. If we use the proposed algorithm to generate a binary bit string between “001001” and “001011”, the inserted binary bit string is “0010101” similar to the previous algorithm. We cannot find any other binary bit strings which are ended with “1” and are between “001001” and “001011” lexicographically with the small size. However, if we want to insert a binary bit string between “001001” and “0010101”, the binary bit string generated by the proposed algorithm is “0010011”. The result is different with the previous one which is “00101001”. It saves 1 bit in the label size and can reduce label size.

Based on Example 2, it is observed that the size of inserted binary bit string increases 1 bit in worst case for each binary bit string insertion in the proposed algorithm. In other words, the label size increases linearly (O(N)) in worst case while it increases always linearly (O(N)) in the previous algorithm.

In addition, the proposed algorithm is able to control the size of new inserted label in EXEL in the update environment with both insertions and deletions frequently. Example 3 shows this fact.

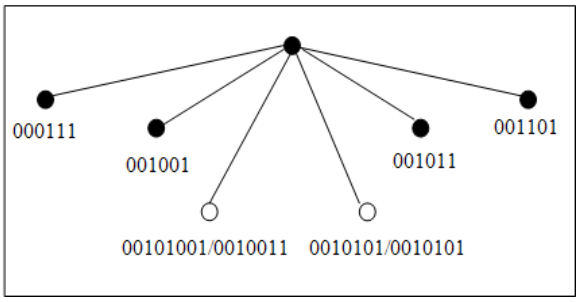

Fig. 5: Insertion of a new node between two existing nodes

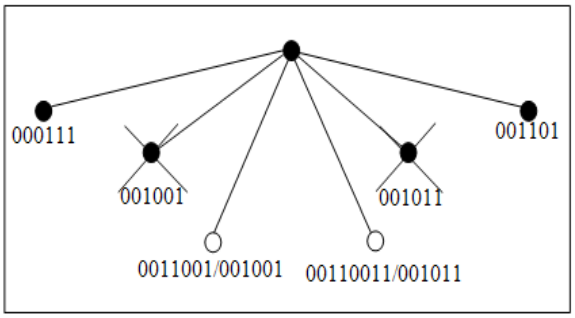

Fig. 6: Deleting and Inserting nodes frequently

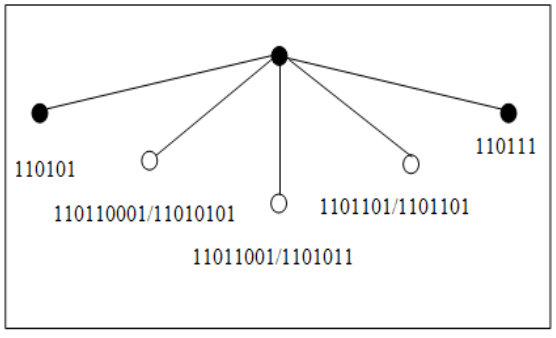

Fig. 7: The behavior of skewed insertions

Example 3: Assume that the two binary bit strings "001001" and "001011" are deleted. Now, we want to generate two new binary bit strings between "000111" and "001101" as shown in Fig. 6. If we use the MakeNewBitString algorithm to generate two new binary bit strings, the new binary bit strings will be "0011001" and "00110011" while the two new binary bit strings generated by the proposed algorithm will be "001001" and "001011". Thus, the proposed algorithm is able to reuse deleted nodes for future binary bit string insertion as well as control the label size.

Another problem of the MakeNewBitString algorithm is its behavior in skewed insertions. In skewed insertions, nodes are always inserted at a fixed place. Thus, the label size increases 1 bit for each insertion using the MakeNewBitString algorithm. We can control label size increment using the proposed algorithm. Example 4 presents the behavior of the proposed algorithm in the skewed insertions.

Example 4: Assume that we want to insert three binary bit strings between "110101" and "110111" as shown in Fig. 7. If we use the MakeNewBitString algorithm to generate new binary bit strings, "1101101", "11011001" and "110110001" are the three new binary bit strings while if we use the proposed algorithm to generate new bit strings, "1101101", "1101011" and "11010101" are the binary bit strings.

Based on example 4, it is observed that label size increment in the proposed algorithm for skewed insertions is equal or less than the inserted label size in the MakeNewBitString algorithm.

RESULTS AND DISCUSSION

The comparison of the proposed algorithm with the MakeNewBitString algorithm in terms of label size increment is shown in Table 2.

Based on Table 2, it is observed that the size of the inserted binary bit string increases 1 bit in worst case for each binary bit string insertion in the proposed algorithm. In other words, the label size increases linearly (O(N)) in worst case while it increases always linearly (O(N)) in the previous algorithm.

Table 2: The comparison of the proposed algorithm with the MakeNewBitString algorithm

	The proposed algorithm		MakeNewBitString algorithm
	Worst case	Best case	Worst case and Best case
a	O(N)	O(1)	O(N)
b	O(N)	O(1)	O(N)
c	O(N)	O(1)	O(N)

a: Insertion of new node between two existing nodes; b: Deletion and insertion of nodes frequently; c: The behavior of skewed insertions

CONCLUSION

This study modifies the MakeNewBitString algorithm of EXEL encoding and labeling scheme in order to control the label size increment of inserted nodes in XML update for skewed insertions. In addition, the MakeNewBitString algorithm is unable to reuse the deleted labels for future insertion while the proposed algorithm can reuse the deleted labels. However, for each label insertion, the size of new binary bit string is increased by 1 bit in the MakeNewBitString algorithm while in the proposed algorithm the size of inserted label is increased by 1 bit in the worst case. As a result, the proposed algorithm requires less storage size than the MakeNewBitString algorithm to store the inserted binary bit string and thus can improve query performance.

As a future study, we intend to evaluate the behavior of the proposed algorithm and compare it with the MakeNewBitString algorithm using different XML dataset for dynamic XML updating process.

REFERENCES

Amagasa, T., M. Yoshikawa and S. Uemura, 2003. QRS: A robust numbering scheme for XML documents. Proceeding of the 19th International Conference on Data Engineering, March 5-8, IEEE Computer Society, Bangalore, India, pp: 705-707. DOI: 10.1109/ICDE.2003.1260842

Bray, T., J. Paoli, C.M. Sperberg-McQueen, E. Maler and F. Yergeau *et al*., 2006. Extensible markup language (XML) 1.1. W3C. http://www.w3.org/TR/xml11/

Ko, H.K. and S.K. Lee, 2006. An efficient scheme to completely avoid re-labeling in XML updates. Lecture Notes Comput. Sci., 4255: 259-264. DOI: 10.1007/11912873_27

Ko, H.K. and S.K. Lee, 2010. A binary string approach for updates in dynamic ordered XML data. IEEE Trans. Knowl. Data Eng., 22: 602-607. DOI: 10.1109/TKDE.2009.87

Li, C. and T.W. Ling, 2005. QED: A novel quaternary encoding to completely avoid Re-labeling in XML updates. Proceeding of the 14th ACM International Conference on Information and Knowledge Management, Oct. 31-Nov. 5, ACM Press, Bremen, Germany, pp: 501-508. http://portal.acm.org/citation.cfm?id=1099554.1099692

Li, C., T.W. Ling and M. Hu, 2006a. Reuse or never reuse the deleted labels in XML query processing based on labeling schemes. Lecture Notes Comput. Sci., 3882: 659-673. DOI: 10.1007/11733836_46

Li, C., T.W. Ling and M. Hu, 2006b. Efficient processing of updates in dynamic XML data. Proceeding of the 22nd International Conference on Data Engineering. Apr. 3-7, IEEE Computer Society, Atlanta, Georgia, USA., pp: 13-13. DOI: 10.1109/ICDE.2006.58

Li, C., T.W. Ling and M. Hu, 2008. Efficient updates in dynamic XML data: From binary string to quaternary string. VLDB J., 17: 573-601. DOI: 10.1007/s00778-006-0021-2

Li, Q. and B. Moon, 2001. Indexing and querying XML data for regular path expressions. Proceeding of the 27th International Conference on Very Large Data Bases, Sept. 11-14, IEEE Press, Roma, Italy, pp: 361-370. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.862&rep=rep1&type=pdf

Min, J.K., J. Lee and C.W. Chung, 2007. An efficient encoding and labeling for dynamic XML data. Lecture Notes Comput. Sci., 4443: 715-726. DOI: 10.1007/978-3-540-71703-4_60

Min, J.K., J. Lee and C.W. Chung, 2009. An efficient XML encoding and labeling method for query processing and updating on dynamic XML data. J. Syst. Software, 82: 503-515. DOI: 10.1016/j.jss.2008.08.014

O'Neil, P., E. O'Neil, S. Pal, I. Cseri, G. Schaller and N. Westbury, 2004. ORDPATHs: insert-friendly XML node labels. Proceeding of the 2004 ACM SIGMOD International Conference on Management of Data, June 13-18, ACM Press, Paris, France, pp: 903-908. http://portal.acm.org/citation.cfm?id=1007568.1007686

Wu, X., M.L. Lee and W. Hsu, 2004. A prime number labeling scheme for dynamic ordered XML Trees. Proceeding of the 20th International Conference on Data Engineering, Mar. 30-Apr. 2, IEEE Computer Society, Boston, USA., pp: 66-78. DOI: 10.1109/ICDE.2004.1319985

Yun, J.H. and C.W. Chung, 2008. Dynamic interval-based labeling scheme for efficient XML query and update processing. J. Syst. Software, 81: 56-70. DOI: 10.1016/j.jss.2007.05.034