Proceedings of the 3rd International Conference on Computing and Informatics, ICOCI 2011,8-9 June, 2011 Bandung, Indonesia | *Paper No. 062*

A FRACTIONAL NUMBER BASED LABELING SCHEME FOR DYNAMIC XML UPDATING

Meghdad Mirabi[1], Hamidah Ibrahim[2], Leila Fathi[3],Ali Mamat[4], and Nur Izura Udzir[5]

[1]*Universiti Putra Malaysia, Malaysia, meghdad.mirabi@gmail.com*
[2]*Universiti Putra Malaysia, Malaysia, hamidah@fsktm.upm.edu.my*
[3]*Universiti Putra Malaysia, Malaysia, fathi_leila67@yahoo.com*
[4]*Universiti Putra Malaysia, Malaysia, ali@fsktm.upm.edu.my*
[5]*Universiti Putra Malaysia, Malaysia, izura@fsktm.upm.edu.my*

ABSTRACT. Recently, XML query processing based on labeling schemes has been proposed. Based on labeling schemes, the structural relationship between XML nodes can be determined quickly without the need of accessing the XML document. However, labeling schemes have to re-label the pre-existing nodes or re-calculate the label values when a new node is inserted into the XML document during the update process. In this paper, we propose a novel labeling scheme based on fractional numbers. The key feature of fractional numbers is that infinite number of fractional numbers can be inserted between any two unequal fractional numbers. Therefore, the problem of re-labeling the pre-existing nodes during the XML updating can be solved if the XML nodes are label by the fractional numbers.

Keywords: Dynamic Labeling Scheme, Fractional Number, XML Updating

INTRODUCTION

Recently, XML as a de facto standard has obtained a popularity for representation and exchanging the data over the Internet (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2008). With the growing popularity of XML, a large range of XML documents appeared on the web. In order to manage these documents, it is required to store and query the XML data efficiently. Several query languages like XPath (Clark & DeRose, 1999) and XQuery (Boag et al., 2007) are designed to process XML data. These query languages are based on regular path expressions to query XML data. The path expression locates nodes within the XML tree. In order to query XML data efficiently, the structural relationships between nodes have to determine quickly without the need of accessing the XML documents. Several researches have been proposed to label the XML tree nodes in such a way that the structural relationships between any two nodes can be determined directly (Amagasa, Yoshikawa, & Uemura, 2003; C. Li & Ling, 2005; Q. Li & Moon, 2001; O'Neil et al., 2004; Silberstein, He, Yi, & Yang, 2005; Tatarinov et al., 2002; Wu, Lee, & Hsu, 2004; Zhang, Naughton, DeWitt, Luo, & Lohman, 2001).

In general, labeling schemes can be categorized into two groups: static labeling schemes (Q. Li & Moon, 2001; Tatarinov et al., 2002; Zhang, Naughton, DeWitt, Luo, & Lohman, 2001) and dynamic labeling schemes (Amagasa, Yoshikawa, & Uemura, 2003; C. Li & Ling, 2005; O'Neil et al., 2004; Silberstein, He, Yi, & Yang, 2005; Wu, Lee, & Hsu, 2004). Static labeling schemes are adequate where XML documents are not updated while dynamic labeling schemes are more adequate where XML documents can be updated. The advantage of using static labeling schemes is that they need small memory space. However, inserting a new node to the XML tree may require re-labeling a large number of pre-existing nodes. In

dynamic labeling schemes, re-labeling the pre-existing nodes is avoided or at least smaller than static labeling schemes but the length of labels increases dramatically when new nodes are inserted to XML tree.

In this paper, we propose a novel XML labeling scheme based on fractional numbers. Our proposed labeling scheme is able to remove the need of re-labeling during XML updating process.

FRACTIONAL NUMBER BASED LABELING SCHEME

In order to easily understand the fractional number generation algorithm illustrated in Figure 1, we first give an example to illustrate how fractional numbers are assigned to a set of ordinal decimal numbers. Table 1 shows fractional numbers assigned to 20 ordinal decimal numbers. We choose 20 as an example but our proposed method can assign fractional numbers for any set of ordinal decimal numbers.

Table 1. Fractional Numbers assigned to 20 Ordinal Decimal Numbers

Decimal Number	Fractional Number	Decimal Number	Fractional Number	Decimal Number	Fractional Number	Decimal Number	Fractional Number
1	1/32	6	1/4	11	1/2	16	¾
2	1/16	7	9/32	12	17/32	17	25/32
3	1/8	8	5/16	13	9/16	18	13/16
4	5/32	9	3/8	14	5/8	19	7/8
5	3/16	10	7/16	15	11/16	20	15/16

The following steps illustrates the details how to assign fractional numbers to a set of ordinal decimal numbers.

Step 1: In order to assign the fractional numbers of 20 ordinal decimal numbers, we assume there is one more number before 1 which is 0 and one more number after 20 which is 21.

Step 2: We firstly assign the middle fractional number between (0, 1) to the middle decimal number between 0 and 21. The middle fractional number between (0, 1) is ½ where it is calculated with [(0 + 1) / 2] and the middle decimal number between 0 and 21 is 11 where it is calculated with 0 + [(21 - 0) / 2].

Step 3: Next, we calculate the middle decimal number between 0 and 11, and between 11 and 21. The middle decimal number between 0 and 11 is 6 (0 + [(11 - 0) / 2]) and the middle decimal number between 11 and 21 is 16 (11 + [(21 - 11) / 2].

Step 4: Next, we assign the middle fractional number between (0, ½) which is ¼ to 6 and the middle fractional number between (½, 1) which is ¾ to 16.

Step 5: Next, we assign the middle fractional number between (0, ¼) to the middle decimal number between 0 and 6, the middle fractional number between (¼, ½) to the middle decimal number between 6 and 11, the middle fractional number between (½, ¾) to the middle decimal number between 11 and 16, and the middle fractional number between (¾, 1) to the middle decimal number between 16 and 21. In this way, fractional numbers can be assigned to a set of ordinal decimal numbers.

The algorithm illustrated in Figure 1 is proposed to generate the fractional numbers for a set of ordinal decimal numbers between 1 and *N*.

FNG-Algorithm (*N*)
Input: A Positive decimal Number *N*
Output: A set of fractional numbers for decimal numbers 1 to *N*

1. Suppose there is a number before the first number which is 0 and a number after the last number which is *N* + 1;
2. Define an array *fractionalNumberArray*1[0, *N* + 1] to store numerators and an array *fractionalNumberArray*2[0, *N* + 1] to store denominators;
   // the size of each array is *N* + 2 and initially the arrays are empty;

3. *fractionalNumberArray*1[0] = 0;
4. *fractionalNumberArray*2[0] = 1;
5. *fractionalNumberArray*1[*N* + 1] = 1;
6. *fractionalNumberArray*2[*N* + 1] = 1;
7. EBFN (*fractionalNumberArray*1, *fractionalNumberArray2*, 0, *N* + 1);
8. Discard the 0^{th} and $(N + 1)^{th}$ elements of *fractionalNumberArray*1 and *fractionalNumberArray*2;

EBFN (*fractionalNumberArray*1, *fractionalNumberArray*2, *leftPosition*, *rightPosition*)
// EBNF is a recursive procedure;
middlePosition = *leftPosition* + **round**((*rightPosition* - *leftPosition) /* 2);
if ((*leftPosition* + 1) < *rightPosition*) **then**
Begin
*fractionalNumberArray*1[*middlePosition*] = (*fractionalNumberArray*1[*leftPosition*] * *fractionalNumberArray*2[*rightPosition*]) + (*fractionalNumberArray*1[*rightPosition*] * *fractionalNumberArray*2[*leftPosition*]);
*fractionalNumberArray*2[*middlePosition*] = 2 * (*fractionalNumberArray*2[*leftPosition*] * *fractionalNumberArray*2[*rightPosition*]);
//gcd return greatest common denominator;
gcdValue = gcd (*fractionalNumberArray*1[*middlePosition*], *fractionalNumberArray*2[*middlePosition*]);
*fractionalNumberArray*1[*middlePosition*] /= gcdValue;
*fractionalNumberArray*2[*middlePosition*] /= gcdValue;
EBFN (*fractionalNumberArray*1, *fractionalNumberArray*2, *leftPosition*, *middlePosition*);
EBFN (*fractionalNumberArray*1, *fractionalNumberArray*2, *middlePosition, rightPosition*);
End

Figure 1. Fractional Number Generation Algorithm

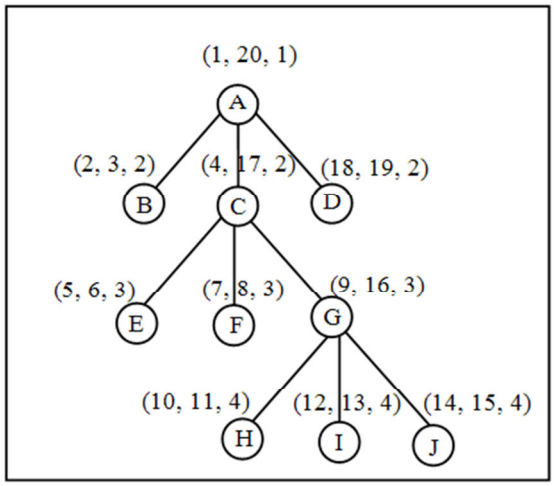

Figure 2. Region Number Labeling Scheme

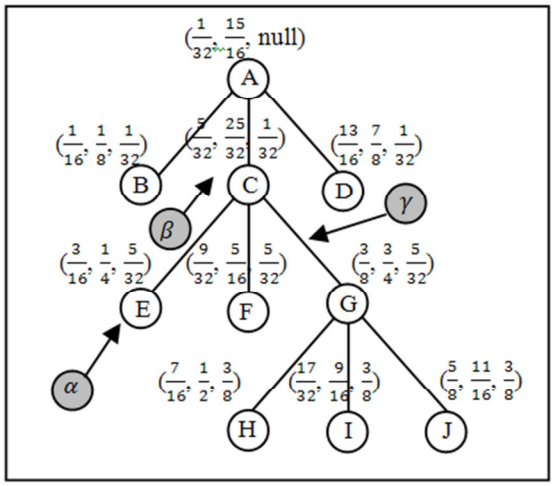

Figure 3. Fractional Number Labeling Scheme

Our proposed scheme which is based on fractional number is applied to the region number labeling scheme with the intention to avoid re-labeling the pre-existing nodes during XML data updating. Therefore, in this way, we are able to keep the order of XML nodes and determine the structural relationships between two arbitrary nodes.

As shown in Figure 2, the level value is added to each node label in order to determine the Parent-Child (P-C) and the sibling relationship between nodes in the region number labeling scheme. However, such information is sensitive in the dynamic XML environment because the level must be modified when a node is inserted into or deleted from the XML tree as a parent (an ancestor) node. Thus, in our proposed labeling scheme, the parent's start value is added to each node label instead of level value. The parent's start value needs more storage space than level value but it is not changed when internal nodes are inserted into or deleted from the XML tree. In addition, using the parent's start value instead of level value eliminates the comparison process of start and end values in order to determine the Parent-Child and the sibling relationships. Hence, the performance of XML query processing can be improved. An example of XML tree labeled by fractional number is illustrated in Figure 3.

When a leaf node or a sub-tree is deleted, re-labeling the pre-existing nodes is not required. In the case of internal node deletion, only parent's start values of its children must

be changed by its parent start value while start and end values for any nodes do not need to be modified. The problem of XML data updating is in insertion. In the following, we present the process of node insertion at different positions of the XML tree.

Generating an Inserted Fractional Number

The MakeNewFractionalNumber algorithm shown in Figure 4 generates a new fractional numbers between two pre-existing fractional numbers.

MakeNewFractionalNumber $\left(\frac{LFN1}{LFN2}, \frac{RFN1}{RFN2}\right)$

Input: $\frac{LFN1}{LFN2} < \frac{RFN1}{RFN2}$

Output: $\frac{MFN1}{MFN2}$ such that $\frac{LFN1}{LFN2} < \frac{MFN1}{MFN2} < \frac{RFN1}{RFN2}$

Case 1: insert a fractional number before the first fractional number

1. $\frac{MFN1}{MFN2} = \frac{RFN1}{2 \times RFN2}$;

Case 2: insert a fractional number after the last fractional number

2. $\frac{MFN1}{MFN2} = \frac{LFN1+LFN2}{2 \times LFN2}$;

Case 2: insert a fractional number between two fractional numbers

3. $MFN1 = (LNF1 \times RNF2) + (LNF2 \times RNF1)$;
4. $MFN2 = 2 \times (LNF2 \times RNF2)$;
5. /*gcd returns Greatest Common Denominator*/
6. gcdValue = gcd(MNF1, MNF2);
7. $MNF1 = \frac{MNF1}{\text{gcdValue}}$;
8. $MNF2 = \frac{MNF2}{\text{gcdValue}}$;

Figure 4. MakeNewFractionalNumber Algorithm

The Process of Insertion

There are three kinds of insertions in XML tree according to the positions in which nodes should be inserted: insertion a node as a child of a leaf node, insertion a node as a sibling node, and insertion a node as a parent node. The algorithm illustrated in Figure 5 is devised to insert a node as a child of the leaf node *targetNode.*

$$Algorithm\ InsertChildOf\left(\frac{start1_{targetNode}}{start2_{targetNode}}, \frac{end1_{targetNode}}{end2_{targetNode}}, \frac{parentStart1_{targetNode}}{parentStart2_{targetNode}}\right)\{$$

$$\frac{start1_{new}}{start2_{new}} = MakeNewFractionalNumber\left(\frac{start1_{targetNode}}{start2_{targetNode}}, \frac{end1_{targetNode}}{end2_{targetNode}}\right);$$

$$\frac{end1_{new}}{end2_{new}} = MakeNewFractionalNumber\left(\frac{start1_{new}}{start2_{new}}, \frac{end1_{targetNode}}{end2_{targetNode}}\right);$$

$$\frac{parentStart1_{new}}{parentStart2_{new}} = \frac{start1_{targetNode}}{start2_{targetNode}};$$

$$Insert\ a\ new\ node\ as\ the\ child\ of\ targetNode\ with\ label\left(\frac{start1_{new}}{start2_{new}}, \frac{end1_{new}}{end2_{new}}, \frac{parentStart1_{new}}{parentStart2_{new}}\right)$$

$$\}$$

Figure 5. InsertChildOf Algorithm

In our propose scheme, we assigned to each node label its parent's start value instead of level value. Therefore, the following rules should be satisfied for the new inserted node:

- $\frac{start1_{targetNode}}{start2_{targetNode}} < \frac{start1_{new}}{start2_{new}} < \frac{end1_{new}}{end2_{new}} < \frac{end1_{targetNode}}{end2_{targetNode}}$.
- $\frac{parentStart1_{new}}{parentStart2_{new}} = \frac{start1_{targetNode}}{start2_{targetNode}}$.

For example, in Figure 3, node α is to be inserted as a child of the leaf node *E*. Therefore, we have:

- $\frac{start1_{\alpha}}{start2_{\alpha}} = MakeNewFractionalNumber\left(\frac{3}{16}, \frac{1}{4}\right) = \frac{7}{32}$
- $\frac{end1_{\alpha}}{end2_{\alpha}} = MakeNewFractionalNumber\left(\frac{7}{32}, \frac{1}{4}\right) = \frac{15}{64}$
- $\frac{parentStart1_{\alpha}}{parentStart2_{\alpha}} = \frac{3}{16}$

The algorithm illustrated in Figure 6 is proposed to insert a new node as the next sibling of the *targetNode*.

$$Algorithm\ InsertSiblingAfter\left(\frac{start1_{targetNode}}{start2_{targetNode}}, \frac{end1_{targetNode}}{end2_{targetNode}}, \frac{parentStart1_{targetNode}}{parentStart2_{targetNode}}\right)\{$$
$$\frac{next1}{next2} = \frac{start1}{start2}\ of\ the\ nearest\ following-sibling\ of\ targetNode;$$
$$if\ there\ is\ not\ any\ following-sibling\ of\ targetNode\ then$$
$$\frac{next1}{next2} = \frac{end1}{end2}\ of\ targetNode's\ parent;$$
$$\frac{start1_{new}}{start2_{new}} = MakeNewFractionalNumber\left(\frac{end1_{targetNode}}{end2_{targetNode}}, \frac{next1}{next2}\right);$$
$$\frac{end1_{new}}{end2_{new}} = MakeNewFractionalNumber\left(\frac{start1_{new}}{start2_{new}}, \frac{next1}{next2}\right);$$
$$\frac{parentStart1_{new}}{parentStart2_{new}} = \frac{parentStart1_{targetNode}}{parentStart2_{targetNode}};$$
$$Insert\ a\ new\ node\ as\ the\ sibling\ of\ targetNode\ with\ label\ \left(\frac{start1_{new}}{start2_{new}}, \frac{end1_{new}}{end2_{new}}, \frac{parentStart1_{new}}{parentStart2_{new}}\right)$$
$$\}$$

Figure 6. InsertSiblingAfter Algorithm

For example, in Figure 3, node β is to be inserted after node *B*. Therefore, we have:

- $\frac{start1_{\beta}}{start2_{\beta}} = MakeNewFractionalNumber\left(\frac{1}{8}, \frac{5}{32}\right) = \frac{9}{64}$
- $\frac{end1_{\beta}}{end2_{\beta}} = MakeNewFractionalNumber\left(\frac{9}{64}, \frac{5}{32}\right) = \frac{19}{128}$
- $\frac{parentStart1_{\beta}}{parentStart2_{\beta}} = \frac{1}{32}$

The process of inserting a new node before a node is similar to the process of inserting a new node after. Therefore, the explanation on this process is omitted here.

Insertion a node as a child node or sibling node has been supported by other dynamic labeling schemes but the advantage of our proposed scheme is in the case of parent node insertion. Our proposed scheme is able to handle parent node insertion without re-labeling the pre-existing nodes. The algorithm illustrated in Figure 7 is designed to insert a new node as a parent of *targetNode*. The position of new parent node of *targetNode* (*new* node) is between the previous and next sibling nodes of *targetNode*. Therefore, start and end values of *new* node are between the end value of the previous sibling node of *targetNode* and the start value of the next sibling node of *targetNode.* In case that the preceding-sibling (the following-sibling) of the *targetNode* does not exist, the start (end) value of parent's *targetNode* can be used.

$$Algorithm\ InsertParentOf\left(\frac{start1_{targetNode}}{start2_{targetNode}}, \frac{end1_{targetNode}}{end2_{targetNode}}, \frac{parentStart1_{targetNode}}{parentStart2_{targetNode}}\right)\{$$
$$\frac{previous1}{previous2} = \frac{end1}{end2}\ of\ the\ nearest\ preceding - sibling\ of\ targetNode;$$
$$if\ there\ is\ not\ any\ preceding - sibling\ of\ targetNode\ then$$
$$\frac{previous1}{previous2} = \frac{start1}{start2}\ of\ parent's\ targetNode;$$
$$\frac{next1}{next2} = \frac{start1}{start2}\ of\ the\ nearest\ following - sibling\ of\ targetNode;$$
$$if\ there\ is\ not\ any\ following - sibling\ of\ targetNode\ then$$
$$\frac{next1}{next2} = \frac{end1}{end2}\ of\ parent's\ targetNode;$$
$$\frac{start1_{new}}{start2_{new}} = MakeNewFractionalNumber\left(\frac{previous1}{previous2}, \frac{start1_{targetNode}}{start2_{targetNode}}\right);$$
$$\frac{end1_{new}}{end2_{new}} = MakeNewFractionalNumber\left(\frac{end1_{targetNode}}{end2_{targetNode}}, \frac{next1}{next2}\right);$$
$$\frac{parentStart1_{new}}{parentStart2_{new}} = \frac{parentStart1_{targetNode}}{parentStart2_{targetNode}};$$
$$\frac{parentStart1_{targetNode}}{parentStart2_{targetNode}} = \frac{start1_{new}}{start2_{new}};$$
$$Insert\ a\ new\ node\ as\ the\ parent\ of\ targetNode\ with\ label\ \left(\frac{start1_{new}}{start2_{new}}, \frac{end1_{new}}{end2_{new}}, \frac{parentStart1_{new}}{parentStart2_{new}}\right);$$
$$\}$$

Figure 7. InsertParent Algorithm

For example, in Figure 3, node γ is to be inserted as the parent of node *G*. Therefore, we have:

- $\frac{start1_{\gamma}}{start2_{\gamma}} = MakeNewFractionalNumber\left(\frac{5}{16}, \frac{3}{8}\right) = \frac{11}{32}$
- $\frac{end1_{\gamma}}{end2_{\gamma}} = MakeNewFractionalNumber\left(\frac{3}{4}, \frac{25}{32}\right) = \frac{49}{64}$
- $\frac{parentStart1_{\gamma}}{parentStart2_{\gamma}} = \frac{5}{32}$

In the region number labeling scheme, level value is added into node label to find the Parent-Child relationship between two nodes. Therefore, after inserting a node as a parent (an ancestor) node, the level value of all the descendants should be updated. But, in our proposed scheme, parent's start value is kept instead of level value. Therefore, if a node is inserted as a parent (an ancestor) node, the parent's start values of all descendants are still unchanged except the parent's start value of the child of the inserted node. Consequently, our proposed scheme avoids the need of re-labeling for the three kinds of insertions as presented in this section.

CONCLUSION AND FUTURE WORKS

In this paper, we propose a novel XML tree labeling scheme based on fractional number. Our proposed scheme is able to avoid the need of re-labeling the pre-existing node in XML tree during the update process. When a node is inserted, the new fractional number is the middle value of the two neighbor fractional numbers. Therefore, it makes the updating process very easy. As a future study, we intend to evaluate our proposed scheme with different dynamic labeling schemes using different XML datasets.

REFERENCES

Amagasa, T., Yoshikawa, M., & Uemura, S. (2003). QRS: A Robust Numbering Scheme for XML Documents. *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*, 705-707. Bangalore, India.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., & Siméon, J. (2007). XQuery 1.0: An XML Query Language. Retrieved from htrp://www.w3.org/TR/xquery/

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2008). Extensible Markup Language (XML) 1.0 (5th Edition) W3C Recommendation. Retrieved from http://www.w3.org/TR/REC-xml/

Clark, J., & DeRose, S. (1999). XML Path Language (XPath) Version 1.0. Retrieved from http://www.w3.org/TR/xpath/

Li, C., & Ling, T. W. (2005). QED: A Novel Quaternary Encoding to Completely Avoid Re-Labeling in XML Updates. *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, 501-508. Bremen, Germany.

Li, Q., & Moon, B. (2001). Indexing and Querying XML Data for Regular Path Expressions. *Proceedings of the 27th International Conference on Very Large Data Bases*, 361-370. Roma, Italy.

O'Neil, P., O'Neil, E., Pal, S., Cseri, I., Schaller, G., & Westbury, N. (2004). ORDPATHs: Insert-Friendly XML Node Labels. *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 903-908. Paris, France.

Silberstein, A., He, H., Yi, K., & Yang, J. (2005). BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data. *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, 285-296. Tokyo, Japan.

Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., & Zhang, C. (2002). Storing and Querying Odered XML Using a Relational Database System. *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 204-215. Madison, Wisconsin.

Wu, X., Lee, M. L., & Hsu, W. (2004). A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, 66-78. Boston, USA.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., & Lohman, G. (2001). On Supporting Containment Queries in Relational Database Management Systems. *ACM SIGMOD Record Journal, 30*(2), 425-436.