International Journal of Fuzzy Logic and Intelligent Systems, vol. 8, no. 2, June 2008 pp. 121-125

PSN: A Dynamic Numbering Scheme for W3C XQuery Update Facility

Dong Kweon Hong

Computer Engineering dept. of Keimyung University

abstract

It is essential to maintain hierarchical information properly for efficient XML query processing. Well known approach to represent hierarchical information of XML tree is assigning a specific node number to each node of XML tree. Insertion and deletion of XML node can occur at any position in a dynamic XML tree. A dynamic numbering scheme allows us to add nodes to or delete nodes from an XML tree without relabeling or with relabeling only a few existing nodes of XML tree while executing XML query efficiently. According to W3C XQuery update facility specifications a node can be added as first or last child of the existing node in XML tree. Generating new number for last child requires referencing the number of previous last child. Getting the number of last child is very costly with previous approaches. We have developed a new dynamic numbering scheme PSN which is very effective for insertion of a node as last child. Our approach reduces the time to find last child dramatically by removing sorting of children.

Key Word : dynamic XML data, dynamic numbering scheme, XML updates

1. Introduction

There exists many active researches to manage large XML data effectively and XML data management on SQL database have been shown very promising in many researches[1, 2, 3, 4]. It is needed to encode hierarchical information when we store XML data on relational databases where a relational data model is flat and it is based on bag. Several numbering schemes already have been proposed to encode XML node relationships in static and dynamic environments. Dewey Order is a well known approach in static XML environments where insertions or deletions of a part of XML tree are not allowed[3]. ORDPATH and DLN are dynamic numbering schemes that can add a node without relabeling other nodes in XML tree[5, 6, 7]. ORDPATH uses Dewey Order dot notation. It uses "careting" operation to make a new number by introducing another dot level when there is no proper number between two nodes. Since there is no proper number between (1.3.1) and (1.3.3) it makes a new node number (1.3.2.1) [5].

Conceptually DLN (Dynamic Level Numbering) is very similar to ORDPATH. DLN uses sub_value idea to make new numbers. When there is no proper number between two consecutive number m and n DLN creates a number by attaching a suffix to m. Since there is no proper number between node (1.2.1) and (1.2.2) it makes a new node number (1.2.1/1) [6]. Whenever a node is inserted to an XML tree both of ORDPATH and DLN search for neighbouring nodes of insertion position to create a proper number for the new node. It is very costly to find those neighbouring nodes in ORDPATH and DLN. The weakness of ORDPATH and DLN is mainly due to the following reasons.

(1) They were trying to maintain parent-child relationship and sibling order together with only one number.

(2) The number can be used to identify the relative sibling order but they are not enough for absolute sibling order in both approach. They can not identify whether a node is the first or the second child with the number. To find out the absolute sibling order all siblings need to be compared.

In this paper we propose a new dynamic numbering scheme PSN, P and S field Number scheme, which allows efficient insertion and deletion of parts of XML. With PSN approach S field of a few nodes could be updated when a new node is inserted to XML tree. Still its numbering process could be done with much less cost than previous approaches. PSN solves the problem of ORDPATH and DLN by separating parent-child and sibling relationships. It can identify the absolute sibling order without comparing the relative order of siblings.

2. PSN Dynamic Numbering Scheme

PSN number consists of P and S fields as in [Fig. 1]. P field maintains parent-child relationship by using Dewey Order dot notation and S field maintains sibling relationships by using natural numbers.

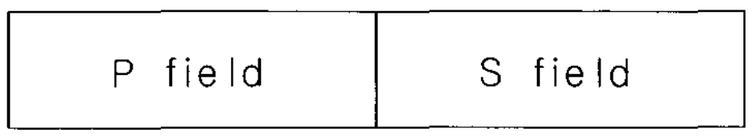


Fig.1 PSN number format

In PSN we maintain 3 additional metadata for each node. They are *pmax*, *sno*, and *eno* of each node. *sno* of a node represents the smallest S field value of its children. While *eno* represents the largest S field value. These values make us to find (n)th child of a node without looking at PSN numbers of all its children. Value of *pmax* is the largest last component value that has been assigned to P field value of its children. It helps us to generate a unique num-

Manuscript received Jan.14, 2008; revised May. 21, 2008.

ber for a new node without looking at PSN numbers of all its siblings.

Initial values of P field in XML tree are the same as Dewey Order and P field value of a node depends only on the P field value of its parent as in [Fig. 2]. The value of S field of a node is the same as the last component of P field value of it before any update operation of XML happens. After some operations of insertion and deletion of nodes the last component of P field and S field value of a node do not match.

Properties of a PSN number are as follows.

Property 1: The P field value of a node is unique in an XML tree and uses dewey order dot notation. The value of a, b, c can be any number in a node (a.b.c|d) and the value of c is not greater than *pmax* of its parent node.

Property 2: The S field value of a node ranges between *sno* and *eno* values of its parent node.

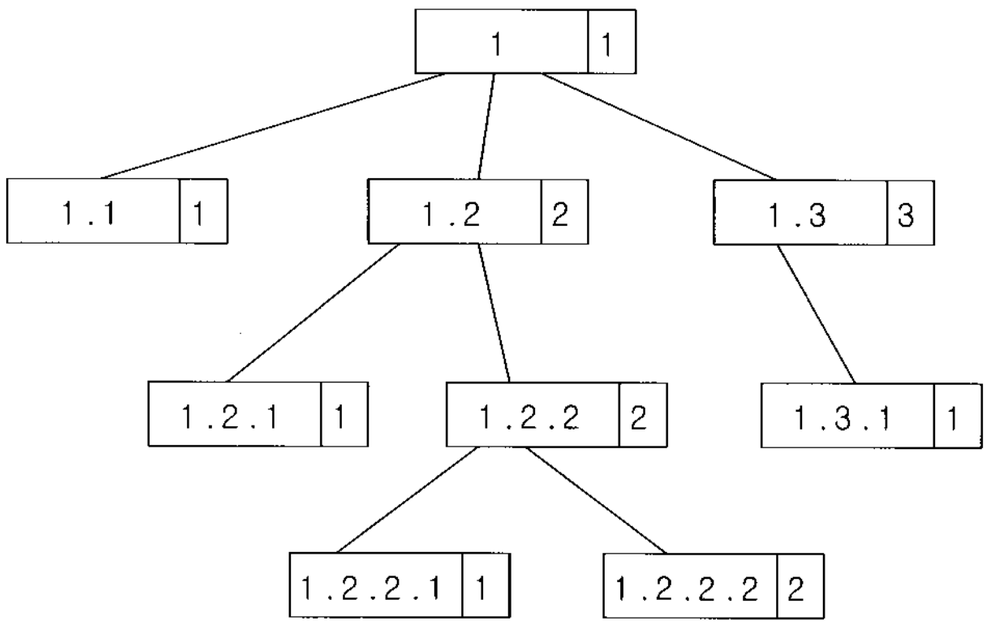


Fig. 2 Example of initial PSN representation

2.1 Insertion of an XML Node

New node can be added in any positions of XML tree with XQuery insert operations. We can classify the insertion of a node as 3 cases according to the insertion position of new node.

(1) Insertion of new node as a middle child
(2) Insertion of new node as a first child
(3) Insertion of new node as a last child

2.1.1 Insertion of Middle Child

Let's consider the situation where a new node is inserted as a second child of XML node (1.2|2) as in [Fig. 2]. This case can be happen when a node is inserted by using *insert before* or *insert after* operation of XQuery.

You can find the pseudo code for insertion procedure in [Fig .3] and you can see the result of node insertion in [Fig. 4]. The value of P field for new node would be (1.2.3) that represents parent-child relationships well and is unique number in the XML tree. The value of S field should be 2 which is greater than the first child by 1 to maintain absolute sibling relationships. S value of the following sibling node is incremented by 1 that make its PSN number changed to (1.2.2 |3) as a third child. Even though node number of the third child has been changed the numbers of its children never be changed because parent-child relationship is only dependent on P field value. Node (1.2.2.1|1) and node (1.2.2.2|2) in [Fig. 2] hadn't been changed even though its parent number has been changed from (1.2.2|2) to (1.2.2|3).

```
// Pseudo code for insert after the des_node
procedure insert_after(des_node)
{
  pare = find_parent_node (des_node);
  //find node (1.2|2)
  max = pare.pmax;
  Assign (1.2.max +1) as P field value of new node ;        //
p field (1.2|3)
  Assign S field value as 2; // (1.2.3|2)
  Increase S field values of  all succeeding
   sibling by 1;
  Increase pmax and eno of pare by 1;
}
```

Fig. 3 Insertion of middle child

The number of children *n* for each node can be easily calculated (n = eno - sno + 1) because each node maintains *sno* and *eno* for S value of its children. When the insertion position is less than n/2 in sibling order the S values of preceeding siblings need to be updated. Otherwise S values of succeeding siblings need to be changed. On average n/4 of siblings need to be changed their S values with our PSN approach.

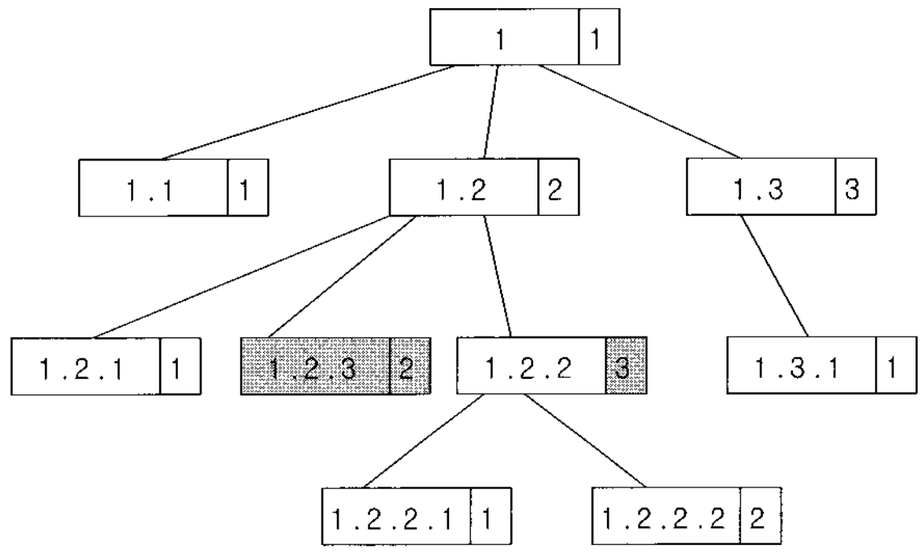


Fig. 4 PSN representation after insertion of second child

2.1.2 Insertion of First Child

A node can be added as a first child with "insert first" or "insert before" operations of XQuery. When a node is inserted in front of node (1.2.1 | 1) in [Fig. 4] the P value of new node would be (1.2.4) and S value of it would be 0 for sibling order. Thus PSN (1.2.4 | 0) maintains its parent-child relationship and sibling relationships correctly. The value of *pmax*, *sno* and *eno* of node (1.2|2) changed to 4, 0 and 3 respectively. The other nodes are never accessed

during the insertion processing. We can see the result of insertion in [Fig. 5].

2.1.3 Insertion of Last Child

A node can be added as a last child with "insert last" or "insert after" operations of XQuery. Most of insertions are likely to be in this category. Only new node and its parent node are modified while processing insertion of last child. For example when a node is added as a last child of node (1.2|2) the number of new node would be (1.2.x+1 | y+1) where x is *pmax* and y is *eno* of node (1.2| 2). And then the values of *pmax* and *eno* of node (1.2|2) are incremented by 1 to update node information. The other nodes are never updated during the insertion processing.

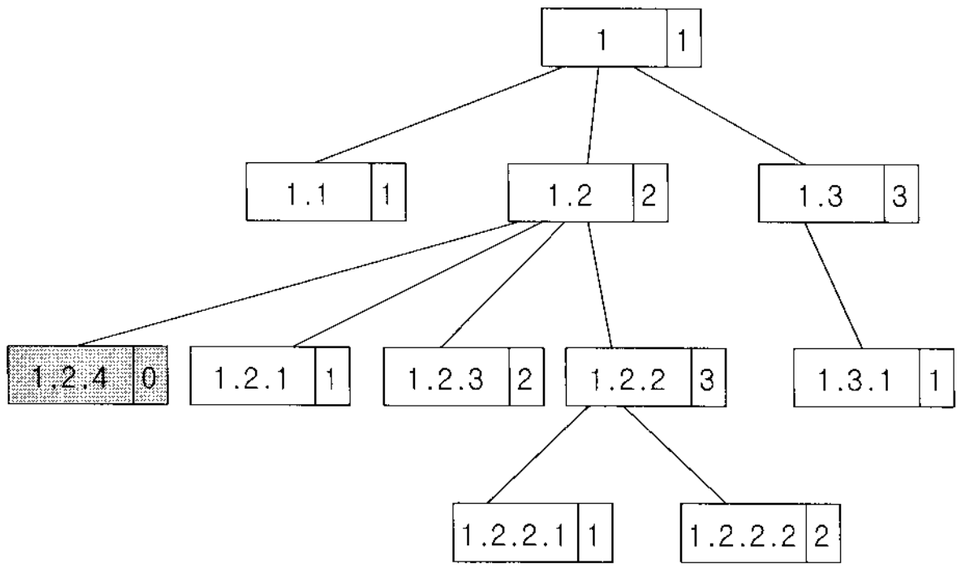


Fig. 5 After insertion of first child

2.2 XML Node Deletion

In dynamic XML environments nodes of XML tree can be deleted at any position. When a node is deleted from an XML tree all of its descendants are deleted also. It is easy to delete nodes in subtree because all nodes in subtree have the P field value which has the common prefix to their parent nodes. For example all descendants of node (1.3 | x) have the (1.3.* | x) node format. For simplicity we only consider XML node deletion in this paper.

Deletion of a node from an XML tree can also be classified 3 cases according to the position of the node as follows.

(1) deletion of middle child
(2) deletion of first child
(3) deletion of last child

2.2.1 Deletion of Middle Child

Let's delete a middle child node (1.2.1 | 1) from figure [Fig. 5]. After deletion S values of siblings that are in front of or in the back of deleted one need to be adjusted. When we delete a node that is at the left of n/2 position (n is the number of its siblings) in sibling order ones in front of it need to be adjusted.

```
// Pseudo code for deletion
procedure delete(node m)
{
  par = parent(m);
  if (position of m is at the left of n/2)
  {
          Increment S values of siblings in front of m.
          Increment sno of par;
  else
  {
          Decrement S values of siblings in the back of m;
          Decrement eno of par;
  }
}
```

Fig. 6 Pseudo code for deletion of node m

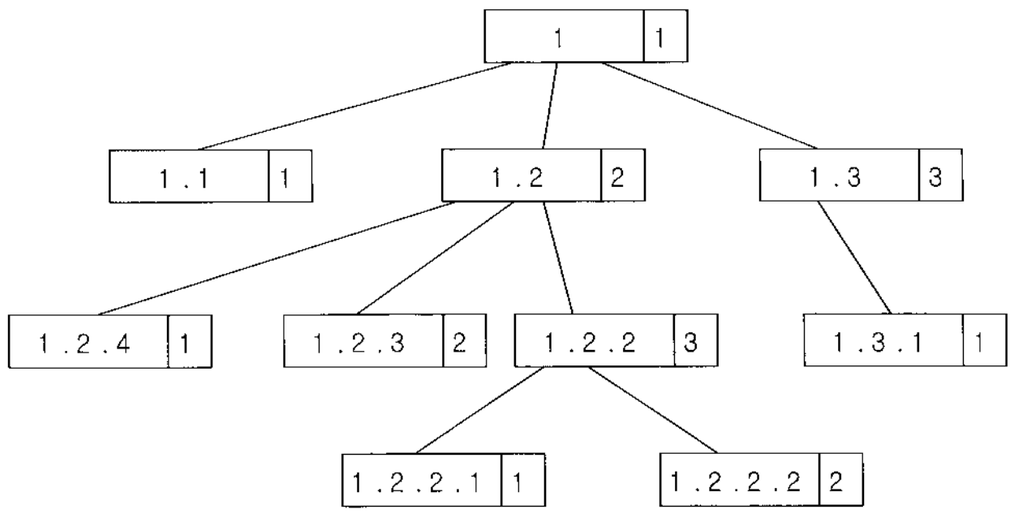


Fig. 7 After deletion of (1.2.1|1) from Fig. 5

After we delete node (1.2.1| 1) from [Fig. 5] the S values of nodes in front of deleted nodes are updated. At the same time we adjust *sno* values of node (1.2|2) to 1. When the deleted node has *n* siblings on average S values of n/4 nodes are updated while processing deletion.

2.2.2 Deletion of First and Last Child

When a first child is deleted *sno* value of its parent is incremented by 1. Meanwhile when a last child is deleted *eno* value of its parent is decremented by 1. The other nodes have never accessed.

3. Evaluation of PSN Scheme

PSN numbering scheme uses P field for parent-child relationship and S field for sibling relationship. Separation of sibling relationship from parent-child relationship enables us to eliminate restrictions of numbers and complicated computations. Evaluations of PSN is done by comparing it with ORDPATH. Comparisons of PSN to ORDPATH scheme are as follows.

1) ORDPATH uses positive odd numbers when it assigns intial numbers to an XML tree. It waste half of number space available. PSN uses any positive numbers.

2) When we insert a node as *i*th child to node (1.3) with ORDPATH we need to find (i-1)th and (i)th sibling to

get their numbers. Children of node (1.3) have node number such as (1.3.2.1), (1.3.2.2.1) and so on. In order to find children from descendants of node (1.3) we need to check whether their number follows the regular expression (1.3.even_no*.odd_no) for all descendants of node (1.3). And then we need to sort those children to find (i-1)th and (i)th children. Insertion time T of a node can be estimated as follows.

T(ORDPATH) = access_time(no_of_descendant) + sort_time(no_of_children)
(no_of_descendant >> no_of_children)

In PSN (i)th child of node (1.3|x) has the node format (1.3.y+1| z). Here y is the *pmax* of node (1.3|x) and z is *sno* + 1 or *eno* - 1 with *sno* and *eno* values of node (1.3|x). After the insertion of a node S fields of siblings in front of or in the back of the inserted node would be updated.

T(PSN) = access_time(no_of_children / 4)

3) Insertion as a last child is the most frequent operation among XML update operations. The node number of new node is dependents on the one of previous last child with ORDPATH. Locating the previous last child requires to scan all of its siblings. While The node number of new node is only dependent on the one of its parent with PSN. When it is inserted as a last child there is no need to adjust S field values of its siblings.

T(ORDPATH) = access_time(no_of_children)
T(PSN) = access_time(parent_node)

4) After continuous insertion and deletion operations the level of ORDPATH numbers are getting longer and longer as in [Fig. 8].(a). While in PSN the value of n is getting bigger and bigger even the actual count of siblings is much smaller as in [Fig. 8](b).

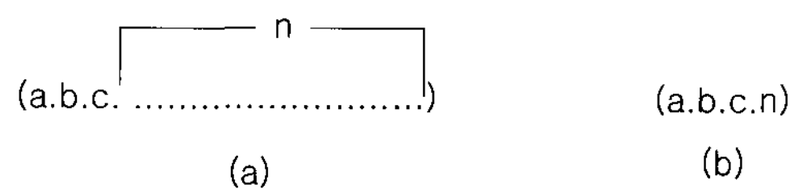


Fig. 8 ORDPATH and PSN numbers

In environments where insertion and deletion operations are occur very often we need to initialize ORDPATH numbers so that the level of a number and its node in the XML tree matches each other. Meanwhile we need to initialize PSN numbers so that the last component of a P field of a node matches the number of its children.

5) When a relational database is used for XML repositories node numbers are usually selected as a primary key. Numbers of adjacent siblings could be very different due to "careting" operation of ORDPATH. For example node (1.3.2.9) and (1.3.7) could be adjacent sibling in XML tree. But relational DBMS do not know their relationships without a special comparison functions for ORDPATH. Thus those numbers could not be in the same B-tree block. Meanwhile PSN is suitable for index clustering when it is used as a primary key because its numbers are easily ordered by comparing numbers component by component because those numbers have the same length of dot separated level. Numbers of siblings are apt to fit in the same B-tree index block because numbers of siblings have the same string lengths with PSN.

4. Conclusions

XML has the hierarchical structure and the order of each node in XML tree is meaningful. The hierarchical structure and the order among nodes are usually maintained by assigning proper numbers to nodes. Effective numbering schemes have been proposed for static XML in which a part of XML is never changed. Extension of XQuery to update facility requires a new numbering scheme for dynamic XML where a part of XML could be changed. Static numbering schemes are not suitable for dynamic XML any more.

ORDPATH is the representative dynamic numbering scheme that has been applied to MS SQL server for XML management. ORDPATH shows some inefficiencies in executing XQuery insert operations to generate new numbers for newly inserted nodes.

In this paper we have developed new dynamic numbering scheme PSN which shows many advantages over ORDPATH. With PSN we can insert a new node very efficiently by locating (i)th child of a node without accessing all descendants of it. There is no need to sort children to find (i)th child. In addition the level of PSN number and corresponding node level in XML tree matches each other which make it easy to compare numbers.

References

[1] J. Shanmugasumdaram et al, "Relational Databases for Querying XML document: Limitations and Opportunities" *in Proceedings of the 25th VLDB Conference,* 1999.

[2] I. Tatarinov, S. Viglas, K.Bayer, J. Shanmugasundaram, E. Shekita, C. Zhang, "Storing and Querying Ordered XML Using a Relational Database System" *in Proceedings of ACM SIGMOD* 2002.

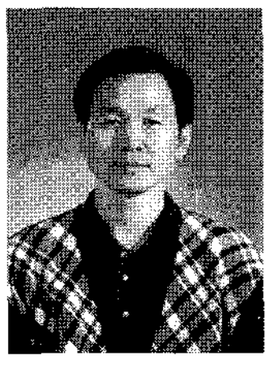
[3] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman, "On supporting Containment Queries in Relational Database Management Systems" *in Proceedings of ACM SIGMOD, May Santa Barbara, CA* 2001.

[4] D. Dehan, D. Toman. M. Consens, and M. Tamer Ozsu, "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding" *in Proceedings of ACM SIGMOD, San Diego CA,* 2003.

[5] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, "ORDPATHs: Insert-friendly XML node lables" *in Proceeding of ACM SIGMOD,* June Paris, France 2004.

[6] T. Bohme, E. Rahm, "Supporting Efficient Streaming and Insertion of XML data in RDBMS" *in Proceeding of CaiSE'04 Workshop,* Volume 3(DIWeb'04), pp70-81, 2004.

[7] T. Harder, M. Haustein, C. Mathis, M. Wagner, "Node labeling schemes for dynamic XML documents reconsidered" *Data and Knowledge Engineering* Volume 60, Issue 1, 2007.



Dong Kweon Hong
He received his MS and Ph.D degree from CIS department of University of Florida, Gainesville, USA in 1992 and 1995. From 1997 to present, he is a professor in Computer Engineering department of Keimyung University. His research interests are XML, databases and web technologies.

Phone : 053-580-5281
Fax : 053-580-5165
E-mail : dkhong@kmu.ac.kr