OrderBased Labeling Scheme for Dynamic XML Query Processing

Beakal Gizachew Assefa and Belgin Ergenc

Department of Computer Engineering, Izmir Institute of Technology
35430 Urla, Izmir-Turkey
{beakalassefa,belginergenc}@iyte.edu.tr

Abstract. Need for robust and high performance XML database systems increased due to growing XML data produced by today's applications. Like indexes in relational databases, XML labeling is the key to XML querying. Assigning unique labels to nodes of a dynamic XML tree in which the labels encode all structural relationships between the nodes is a challenging problem. Early labeling schemes designed for static XML document generate short labels; however, their performance degrades in update intensive environments due to the need for relabeling. On the other hand, dynamic labeling schemes achieve dynamicity at the cost of large label size or complexity which results in poor query performance. This paper presents OrderBased labeling scheme which is dynamic, simple and compact yet able to identify structural relationships among nodes. A set of performance tests show promising labeling, querying, update performance and optimum label size.

Keywords: XML Query Processing, Dynamic Labeling Scheme, OrderBased Labeling Scheme.

1 Introduction

The fact that XML has become the standard format for structuring, storing, and transmitting information has attracted many researchers in the area of XML query processing. XPath and XQuery are languages for retrieving both structural and full text search queries from XML documents [1 and 2]. XML labeling is the basis for structural query processing where the idea is to assign unique labels to the nodes of an XML document that form a tree structure. Label of each node is formed in a way to convey the position of the node in XML tree and its relationship with neighbor nodes. These relationships are Ancestor-Descendent (AD), Parent-Child (PC), Sibling and Ordering [2].

There are basically two approaches to store XML document. The first one is to shred the XML document to some database model. The XML document is mapped to the destination data model example, relational, object oriented, object relational, and hierarchical. The second approach is to use native XML Database (NXD) [27, 28, 29, 30 and 31]. Native XML database (NXD) is described as a database that has an XML document as its fundamental unit of storage and defines a model for an XML

G. Quirchmayr et al. (Eds.): CD-ARES 2012, LNCS 7465, pp. 287–301, 2012.

document, as opposed to the data in that document (its contents). It represents logical XML document model and stores and manipulates documents according to that model. Although XML labeling is widely used in NXD, it also plays a role in the shredding process.

Labeling schemes can be grouped under four main categories namely; Range based, Prefix based, Multiplication based, and Vector based. Range based labeling schemes label nodes by giving start and end position which indicate the range of labels of nodes in sub trees [3, 4, 5 and 23]. Prefix based labeling schemes concatenate the label of ancestors in each label using a delimiter [6, 7, 8, 9 and 10]. Multiplication based labeling schemes use multiplication of atomic numbers to label the nodes of an XML document [16 and 19]. Vector based labeling schemes are based on a mathematical concept of vector orders [17, 18 and 24]. Recently, it is common to see a hybrid labeling schemes which combine the advantages of two or more approaches [25] and [26].

A good labeling scheme should be concise in terms of size, efficient with regard to labeling and querying time, persistent in assuring unique labels, dynamic in that it should avoid relabeling of nodes in an update intensive environment, and be able to directly identify all structural relationships. Last but not least, a good labeling scheme should be conceptually easy to understand and simple to implement. Finding a labeling scheme fulfilling those properties is a challenging task. Generally speaking, labeling schemes that generate small size labels either do not provide sufficient information to identify all structural relationships among nodes or they are not dynamic [3, 4 and 5]. On the other hand, labeling schemes that are dynamic need more storage which results in decrease of query performance [6, 7, 8, 9, 10 and 20] or are not persistent in assuring unique labels [9 and 10].

This paper presents a novel dynamic labeling scheme based on combination of letters and numbers called OrderBased. Each label contains level, order of the node in the level and the order of its parent. Keeping the label of the existing nodes unaltered in case of updates and guaranteeing optimized label size are the main strengths of this approach. Label size and dynamicity is achieved without sacrificing simplicity in terms of implementation.

In performance evaluation OrderBased labeling scheme is compared with LSDX [9] and Com-D [10]. These labeling schemes are chosen because using combinations of letter and numbers, including the level information of a node in every label, and avoiding relabeling when update occurs are the common features and design goals of the three schemes. Storage requirement, labeling time, querying time, and update performance are measured. Results show that OrderBased labeling scheme is smaller in size and faster in labeling and query processing than LSDX labeling scheme. Although Com-D labeling scheme needs slightly less storage than OrderBased, its labeling, querying, and update performance is the worst due to compression and decompression overhead cost.

The paper is organized as follows: Section 2 presents a brief discussion of related work, section 3 presents OrderBased labeling scheme. Section 4 illustrates storage requirements, labeling time, querying, and update performance of OrderBased labeling scheme in comparison with LSDX and Com-D labeling schemes. Finally, section 5 gives conclusion and a glimpse of future works.

2 Related Work

Labeling schemes can be defined as a systematic way of assigning values or labels to the nodes of an XML tree in order to speed up querying. The problem of finding a labeling scheme that generates concise, persistent labels, supporting updates without the need of relabeling, and ease of understanding and implementation dates back to 1982[3]. In the pursuit of solving the labeling scheme problem, a number of approaches have been proposed. These labeling approaches can be grouped in four major categories: Range based, Prefix based, Multiplication based and Vector based.

Range based labels for a node X has a general form of <start-position, end-position>, where start-position and end-position are numbers such that for all nodes Y in the sub tree of X, start_position(Y)> start-position(X) and end-position (Y) < end-position(X). The variations among range based labeling schemes are due to the definition of start-position and end-position. For instance, [3, 4 and 5] define the start-position as pre-order traversal of a tree. Traversal Order, Dynamic Range Based labeling schemes take the end-position as post traversal order of a tree, Extended Traversal Order [4] consider it as the size of the sub tree which is greater than or equal to the total number of nodes in the sub tree. On the other hand, SL (Sector based Labeling scheme) [27] defines the ranges as angles. The sectors are allocated to nodes in such a way that the angle formed by a parent's sector at the origin completely encloses that of all its children. Range based labeling schemes generally produce concise labels and are fast in determining ancestor descendant relationships; however, except for Sector Based labeling scheme, they do not provide sufficient information to determine parent-child and sibling-previous/following relationships. In addition, even the dynamic labeling schemes do not avoid relabeling completely, they only support updates to some extent.

In Prefix based labeling schemes, node X is an ancestor of node Y if the label of node X is the prefix of node Y. The main advantage of Prefix based labeling approach is that all structural relationships can be determined by just looking at the labels. The main critics about prefix based labeling schemes is its large storage requirement. Simple Prefix labeling scheme [6] and Dewey ID [7] are not efficient for dynamic document since insertion needs relabeling of nodes. ORDPATH [8] supports updates without relabeling by reserving even and negative integer. However, after the reserved spaces are used up, relabeling is unavoidable. LSDX – Labeling Scheme for Dynamic XML documents [9] is a fully dynamic prefix labeling scheme. Nonetheless, it generates huge sized labels and does not guarantee unique labels. Com-D – Compact Labeling Scheme [10] reasonably reduces the size of labels through compression. However, compression while labeling and decompression while querying dramatically degrades its efficiency. Whereas LSDX avoids relabeling after updates at the cost of storage, Com-D achieves reasonably small storage requirement at the cost of labeling and querying time.

Multiplicative labeling schemes use atomic numbers to identify nodes. Relationships between nodes can be computed, based on some arithmetic properties of the node labels. The main limitations of this approach lies in its expensive computation and large size. Hence, it is unsuitable for labeling a large-scale XML document. Prime Number labeling scheme [19] and Unique Identifier labeling scheme [16] are examples of a multiplication based labeling schemes.

The other groups of labeling scheme that are seen in literature are based on vector order. A vector code is a binary tuple of the form (x, y) where x > 0. Given two vector codes A: (x1, y1) and B: (x2, y2), vector a precedes vector B in vector order if and only if $\frac{x1}{y1} \leq \frac{x2}{y2}$. If we want to add a new vector C between vector A and B, the vector code of C is computed as x1+x2 , y1 +y2).The vector order of A<B<C because $\frac{x1}{y1} \leq \frac{x1+x2}{y1+y2} \leq \frac{x2}{y2}$ holds true [17]. It is demonstrated that the vector based approach can be applied to both range based and prefix based labeling schemes [18]. DDE and CDDE are application of vector order approach to Dewey ID [24] whereas V-containment its application to range containment labeling scheme [24]. Vector based labeling schemes avoid relabeling in update intensive environment and can be applied to any other labeling schemes, however, there is always a computation overhead to determine relationship among nodes.

Recently it is common to see a hybrid labeling schemes which balances the weakness of one approach with the strength of another approach [25 and 26]. There are also labeling schemes that capitalize on the characteristics of data structures [22], or make use of the type information or DTD [21]. Moreover, Twig pattern matching algorithms has been researched for fast xml query processing [32, 33 and 34].

Generally, labeling schemes that generate small sized labels neither provide sufficient information to determine all structural relationships nor are efficient in a dynamic environment [3, 4 and 5]. On the other hand, labeling schemes that generate labels that provide enough structural relationship information and also support updates without relabeling either are large in size or are inefficient in query processing.

3 OrderBased Labeling Scheme

OrderBased labeling scheme presented in this paper is based on combination of letters and numbers. Each label contains level, order of the node in the level and the order of its parent. First part of the label is numeric and indicates the level information of a given node. The second part gives alphabetical order of the node relative to the left most node of the level. The last part is the order of the parent node. The order and the level information guarantee unique labels. The usage of characters enables it to generate a completely new order before and after the position of a given node, and also between two nodes without affecting existing order in case of insertions. For instance given two orders O1, and O2 where O1="abc" and O2= "bd", we can generate as many strings as we need which are between O1 and O2 in alphabetic order ("abcb", "abcd", "abce"..).

In OrderBased labeling scheme each label is a triple <level, order, parentorder>, where level is an integer that represents the distance of the node from the root node, order is a character that represents the level based horizontal distance of the node from the left most node at each level, parentorder is the parent's order of a given node .The level of the root node is 0, and the level of the children of the root node is 1. Likewise, the levels of other nodes can be computed as the distance of the node from the root node as seen in Fig 1.

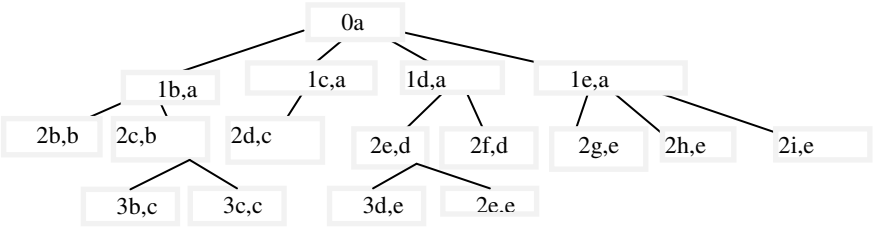

Fig. 1. OrderBased labeling scheme

An OrderBased label provides the information of the parent-child, and siblings-following/previous in a direct way, and ancestor-descendant relationships in recursive manner. For example in Fig 1, the node with label "1e, a" is the parent of the nodes with labels "2g, e", "2h,e", and "2i,e". This parent to child relationship is provided because the parent order of the three nodes is "e", and presumably their level is 1+1=2. Moreover, nodes that have the same level information and with the same parent order are siblings. However, to find the ancestors /descendants of a given node, first there is a need to move to the parent/children, and then the parent of the parent/children recursively till the intended level is reached.

3.1 Optimizing the Size

To address the problem of large storage size, OrderBased labeling scheme has a routine which optimizes the label size of every level. Small label sizes enhance query, update and labeling performances. Before labeling or making any insertions, the OrderBased labeling scheme computes the optimal number of characters needed to label the nodes at every level. To illustrate the need of optimizing the size, we will give a brief description of the size requirement in terms of number of characters.

Assume the total number of nodes at a given level is M. If we start labeling order of the first node in the level by 'b', the labeling continues with 'c', accordingly the orders of the 25th and 26th nodes will be 'z' and 'zb' respectively. Since there is a need of concatenating extra 'b' after reaching the letter 'z' in ever 26th node, the size of the order increases dramatically. If the total number of nodes at a given level M is not greater than 25, we can generate M unique one character length orders using alphabets from b to z. If M is between 26 and 50 inclusive, we use 25 single character alphabets and (M-25) double character length. For example If M= 10, 40, 66, and 90, then size requirement is then 1(10) =10, 1(25) + 2(40-25) =55, 1(25) + 2(25) + 3(66-50) = 123, and 1(25) +2(25) + 3(25) + 4(90-75) = 210 number of characters respectively.

The total size requirement for orders at a given level with a total number of nodes M can be generalized as,

$$25 * \sum_{i=1}^{w} i \; + \text{M mod } 25*(\text{w}+1) \tag{1}$$

where w=floor (M/25). In order to have an optimal size of orders, the OrderBased labeling first calculates the number of characters needed to label M number of nodes.

$$25^x = M$$

$$\log 25^x = \log M$$

$$X = Ceil(\frac{\log M}{\log 25})$$

The function Ceil returns the smallest integer that is greater than or equal to the given expression. For example, ceiling (1.45) =2, ceiling (9.8) =10.

By this approach the first child is labeled with X number of b's. For example if M is 625, X computed to be 2 , the order of the 1st ,2nd , 26th, 624th , and 625th is 'bb','bc','cb','zy', and 'zz' respectively. By this approach, the total size of orders for all nodes of a given level is

$$M * Ceil\left(\frac{log\ M}{log\ 25}\right) \quad (2)$$

Table 1. Analytical storage requirement

M	Optimized	Un- optimized
24	24	24
50	100	75
75	150	75
100	200	250
1000	3000	20500
2000	6000	81000
1000000	5000000	20000500000

Table 1 shows a comparison of the total number of characters needed to label the order of nodes using optimized and un-optimized approaches. For M<=25 both approaches need same storage requirement, while the number of nodes M is from 26 to 99, storage requirement for the un-optimized approaches is slightly smaller. Generally, for the number of nodes M>100, the storage requirement for the optimized approach is always smaller than the storage requirement of the un-optimized approach. The difference of the storage requirements for the two approaches considerably increases as the number of nodes M increases. This makes the optimized approach to be preferred to the un-optimized approach.

In OrderBased labeling scheme, optimizing the size is a prior operation before labeling and inserting a sub tree. The Determine-size routine seen in Fig 2, takes the XML tree to be labeled or inserted as input computes the number of nodes at every level, then returns a string array.

For example , if a given XML document has 500, 3000, 9000 , 1000000, and 2000000 number of nodes at 1st, 2nd ,3rd, 4th and 5th level respectively, the above routine returns Y, where Y[1]='bb', Y[2]='bbb', Y[3]='bbb', Y[4]='bbbbb', and Y[5]=' bbbbb'.

```
Determine-size (XML tree)
{
        String array Y[height of tree]
        Integer array X[height of tree]
        Determine the total number of nodes per each level
        Put them into an integer array X
        for ( i=0 to height of tree)
        {
                X[i] = Ceil(log X[i] / log 25)
                Y[i]=concatenate X[i] number of 'b'
        }
        Return Y
}
```

Fig. 2. Determine-size routine

3.2 Generating the Order of a Node

Rule 1

Label the order nodes of a given level starting by the concatenation of b's returned by the Determine-size routine. For the second, third, and forth node, increment the last character to 'c','d', and 'e' respectively. Accordingly for the rest of the nodes, increment the orders alphabetically.

For example if 'bbb' is the string returned for a given level, the order of the 1st, 2nd , 25th , 26th , an 15625th node are labeled as 'bbb', 'bbc',' bbz', 'bcb',and 'zzz' respectively.

3.3 Generating Orders for New Inserted Nodes

Rule 2

To insert a node before the first node of a given level, get the order of the node then count down to the preceding alphabet, if all characters are "b", insert "a" before the last "b".

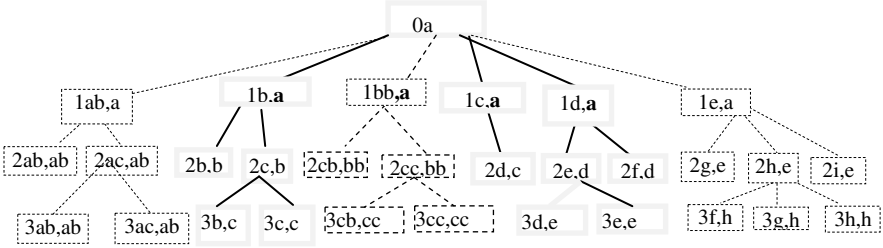

Fig. 3. Sub tree insertion before the first node of a given level, between two nodes and after the last node of a given level

Fig 3 shows how insertion before the first node of a given level is handled by OrderBased labeling scheme. Here Rule 2 is applied to insert a node before "1b,a". Because there is no node before it we add 'a' before 'b' then we will have "1ab,a". At the second level, there are two nodes to be inserted before "2b,b". Thus, applying Rule 2, the labels of the inserted nodes will be "2,ab,ab", and "2aab,ab". Similarly, the labels of the two nodes at level 3 will be "3ab,ab" and "3aab,ab". Insertions before the first node of a given level can be handled by applying Rule 2 without the need of relabeling.

Rule 3
To insert a node between two nodes, keep counting from the code standing before it so that the code for the new node will be greater than the code of its previous sibling and less than the code of its next sibling.

It can be seen from Fig 3 that, insertion between two nodes can be made without affecting the order of the existing nodes. Applying Rule 3 at the first level a unique label "1bb,a" is generated between "1b,a" and "1c,a". Likewise at level 3 and level "2cb,bb" , "2cc,bb", and "3cb,cc", "3cc,cc" respectively are unique labels generated between two nodes without the need or relabeling.

Rule 4
To insert a node after the last node of a level, increment the order of the last order alphabetically.

Fig 3 shows how insertion after the left most node of a tree is handled. Rule 4 states that insertion after the last node of a given node is handled by incrementing the order of the last node alphabetically. That is after "1d,a" is "1e,a", likewise, "2g,e" , "2h,e" and "3f,k","3g,k' are after "2i,h" and "3e,e" respectively.

Fig 3 demonstrates that inserting a sub tree at any arbitrary position does not need any relabeling of nodes. Rules 2, 3 and 4 guarantee unique labels are given to the newly inserted nodes or sub tree with regardless of the point of insertion. OrderBased labeling scheme is persistent in that it insures a uniqueness of labels in a dynamic environment.

4 Performance Evaluation

In this performance evaluation part of the study, OrderBased labeling scheme is compared with the LSDX and Com-D (Compressed LSDX) labeling schemes. These labeling schemes are chosen because they share main feature and design goals. Using combinations of letter and numbers, including the level information of a node in every label, and avoiding relabeling when update occurs are the common feature and design goals of the three schemes. Moreover, because three of them contain the information about the label of the parent node, they can be grouped under prefix based labeling scheme.

There are four sets of tests in this performance evaluation: the first set compares the storage requirement of three schemes. The second set analyzes labeling time. The third set examines the query performance and the last set investigates update performance.

4.1 Experimental Setting

The performance evaluation is conducted on an Intel(R) Core™2Duo CPU E8400 @3GHz.27 GHz and 2.00 GB of RAM Windows 7 Professional computer. All schemes are implemented using Visual Basic .net 2010. So as to avoid discrepancy, each querying and labeling time performance test is run 5 times and the average is taken.

A B+ tree is used to store the labels. In the non-leaf nodes of the B+ tree, only labels are stored. In addition to labels, the leaf nodes contain the name of nodes of the XML tree or attributes with their corresponding values [15 and 22].

4.2 Characteristics of Datasets

The datasets used in this performance evaluation are generated using xmlgen of the XMark: Benchmark Standard for XML Database Management [11]. The xmlgen produces XML documents modeling an auction website, a typical e-commerce application. It generates a well-formed, valid and meaningful XML data. Xmlgen is well known for its efficient and scalable generation of XML documents of several GBs.

Number and type of elements are chosen according to a template and parameterized with certain probability distributions. The words for text paragraphs are taken from Shakespeare's plays. The generator is deliberately designed to have only a single parameter: factor. The factor parameter determines the size of the document generated. It accepts float number from 0 to any number. Zero value for the factor generates the minimum document.

Table 2. Characteristics of datasets

Dataset	Factor	Size(MB)	No of Nodes	Max Fan-out
D05	**0.5**	**56.2**	**832911**	**12750**
D06	**0.6**	**68.2**	**1003441**	**15300**
D07	**0.7**	**79.7**	**1172640**	**17850**
D08	**0.8**	**90.7**	**1337383**	**20400**
D09	**0.9**	**102**	**1504685**	**22950**
D10	**1.0**	**113**	**1666315**	**25500**

By giving values from 0.5 to 1.0 to the factor parameter of the xmlgen, six datasets with size of 56.2 to 113 MB, with number of nodes ranging from 832,911 to 1666315 and maximum fan-out starting 12750 to 25,500 are generated. The characteristics of the datasets are seen in Table 2.

4.3 Storage Requirement

In this performance evaluation test set, the storage requirement for the three schemes is studied. For the six datasets introduced in the previous section, the sizes of labels in MB are shown in Fig 4.

The storage requirement of LSDX labels is the largest as compared to the rest of the two. This resulted from the fact that LSDX label size depends on fan-outs and the height of the tree. To illustrate: for the first 25 children the size of a LSDX label is 25 characters (letter b to z) plus the label of the all its ancestors. Since after every 25th children we reach at letter z, there is a need to concatenate b. This makes the label size to increase by one character. The storage requirement for LSDX labels depend on the fan-outs and the height of the tree (since each label contains the label of its ancestor nodes). The more the number of fan-outs and the taller the tree, the larger is the label size.

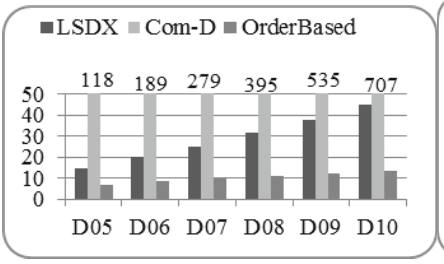

Fig. 4. Storage requirement(MB)

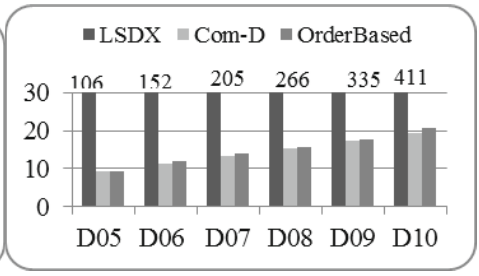

Fig. 5. Labeling time(seconds)

Com-D is a compressed version of LSDX. The compression is done by counting the number of times a letter is consecutively repeated. For example if the LSDX label of an XML node is abzzzzzzrr.dd, its equivalent Com-D label is ab6z2r.2d [10].

As it can be seen from Fig 6, for all the datasets used in this performance analysis, Com-D needs the least storage requirement. Com-D label size is from 91% to 95% smaller than LSDX label size. The figure also demonstrates that the storage requirement for OrderBased labels is from 91.11% to 94.94 % smaller than the storage requirement of LSDX labels. For dataset D05, OrderBased label size is the same as that of Com-D. However, for the rest of the datasets, the storage requirements are from 2.4% to 7.7% greater than the label size of Com-D.

Collision is one of the drawbacks of the LSDX and Com-D labeling scheme. For every dataset used in this performance evaluation, the two schemes give the same label for more than one XML nodes. Table 3 demonstrates the number of collisions detected while labeling using the LSDX and Com-D labeling schemes. For this reason, both LSDX and Com-D are impractical.

Table 3. Number of collisions detected

	D05	D06	D07	D08	D09	D10
Collision	57	43	34	13	30	86

In OrderBased labeling scheme, there is no collision. It avoids collision by keeping a global level based horizontal order and parent order. Both LSDX and Com-D are impractical due to the existence of collision. OrderBased is superior to the two labeling schemes for its persistence and optimal storage requirement.

4.4 Labeling Time

In this sub section, the time required to label a given XML document is studied. The time required for labeling that is seen on Fig 5 above is the average labeling time taken from five tests done on each dataset. The labels are generated by a depth first traversal for the three labeling schemes.

Fig 5 stipulates that for all the six datasets, LSDX is at 7.99 to 15.74 times faster than Com-D. With regard to labeling time, OrderBased labeling scheme is approximately 2.2 to 3.9 and 17.28 to 51.8 times faster than LSDX and Com-D labeling schemes respectively.

The labeling time performance hit of OrderBased over LSDX is due to LSDX's larger label size (Fig 4: the total label size of LSDX is more than 100 to 400 times larger than the total label size of OrderBased). Even though Com-D labels need the minimum storage requirement, it takes the longest labeling time. This decrease in labeling performance results from compression overhead.

The labeling time test set shows that OrderBased labeling scheme takes the least labeling time compared to LSDX and Com-D labeling schemes. This labeling time performance hit of OrderBased is because of the optimal label size. From this result it can be concluded that compression degrades labeling time performance more than large label size does.

4.5 Query

In this performance evaluation part, a query which returns all descendants of the root node is run. Finding descendant of a given node depends on the time required for Parent-Child, and Sibling-Order queries.

Given an ancestor finding its descendants is one of the structural queries found in XML querying. These types of queries are usually seen in XPath statements. The query for retrieving all descendant of a root node is equivalent to the XPath expression Site/*(since the root node of the data sets used in this performance evaluation is site).

For a reasonably small size and small number of nodes of a given XML data set, LSDX and OrderBased take nearly the same time. However, OrderBased executes faster as the data size and the number of nodes increase. In addition, both LSDX and OrderBased labeling scheme are incomparably faster than Com-D. This performance variation comes from decompressing overhead for Com-D. Com-D querying involves decompressing of each label. It can be seen from Fig 6 that decompressing degrades query performance than label size does.

OrderBased labeling scheme is superior to LSDX and Com-D with respect to querying time. Such a performance hit is due to its optimized size of labels.

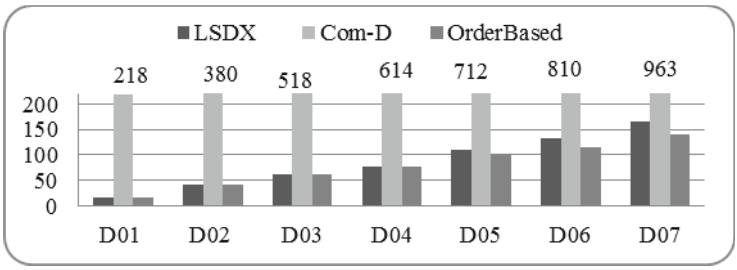

Fig. 6. Time required for retrieving all descendants of a given node

4.6 Updates

In this update performance evaluation of the study, the time needed to insert a sub tree, and delete a sub tree for the three schemes is analyzed. The most profound problem with most XML labeling schemes is that they are designed with an assumption of static document. Whenever a deletion or an insertion is done on the XML document, relabeling of all or part of the XML tree is inevitable. However, in real world applications, updating an XML document is an important and necessary operation.

Inserting a Sub Tree

In this performance evaluation part of the study, the time to insert a sub tree which is an XML by itself is seen. For this study, an XML dataset D01 of 11.3 MB is generated by giving 0.01 to the factor parameter of the xmlgen generator. Inserting D01 at different part of the XML tree produces same time. Thus, for convenience for all the datasets the D01 is inserted as the child of the root node.

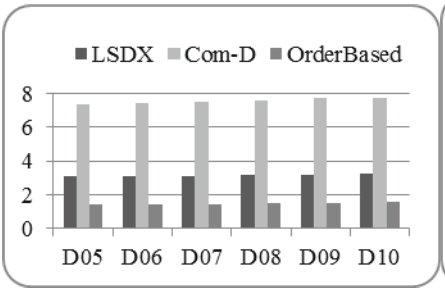

Fig. 7. Insertion time (sec)

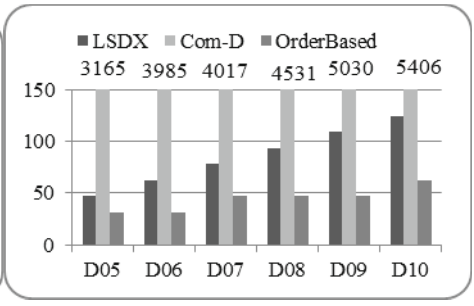

Fig. 8. Deletion time (ms)

Fig 7 shows that the time of insertion of DO1 to the six datasets is nearly constant irrespective of their size. Moreover, insertion time mainly depends on the size of the inserted sub tree.

Com-D takes more than two times and four times longer time than that of LSDX and OrderBased labeling schemes. These performances degrade is resulted from the time needed for compression, since all labels have to be compressed. Fig 7 illustrates

that OrderBased is superior to the rest of the schemes with respect to insertion time in that it is twice faster than LSDX and four times faster than Com-D. OrderBased insertion time performance hit is due to its reasonable small size.

Deleting a Sub Tree

In this part of the performance evaluation, the time needed to delete a sub tree is studied. All the three schemes avoid relabeling after deletion. The spaces and the labels deleted can be used for future insertions.

For the B+ tree used to store the labels of XML tree nodes, a mechanism of lazy deletion is employed. Lazy deletion does not rebalance the B+ tree on deletion. Avoiding rebalancing on deletion has been justified empirically [12, 13 and 14].

Delete site/closed_auctions: delete the node with name closed_auctions.

Fig 8 depicts that Com-D takes the longest time to delete in all the six datasets. This is because decompressing is necessary to determine whether the nodes are descendants of the deleted node. OrderBased labeling scheme deletion is 1.5 to 2.33 faster than LSDX.

4.7 Discussion on Results

In this performance evaluation study we have seen the storage requirement, labeling time, querying time, insertion time and deletion time for OrderBased, LSDX, and Com-D labeling schemes.

The first test set for storage requirement, LSDX labels need the largest storage requirement .Com-D labels need the least space. The storage requirement for OrderBased labels is nearly as good as the storage requirement for Com-D labels (2.34% to 7.7% greater than Com-D).

The second test set for labeling time requirement shows that OrderBased needs the least labeling time whereas Com-D takes the longest labeling time because of compression overhead. From this result it can be concluded that the larger the label size, the faster the labeling is. On the other hand, the compression reduces the label size; it degrades labeling time more than large label size does.

For querying performance, for small data sets, it seems LSDX and OrderBased take equal time. However, as the data size increases, it becomes clear that OrderBased needs the least time. Com-D has the least performance because of the need of decompression.

In the fourth test, update performance (insertion and deletion) time requirement is studied. With regard to insertion, OrderBased needs the least time. Again Com-D needs the longest time because of compression overhead. For deletion time requirement test, OrderBased needs the least time.

5 Conclusion

This paper pointed out the challenges of dynamic labeling scheme for XML documents. Large storage requirement, inefficient labeling or querying time and

complexity are challenges of dynamic labeling schemes. To address these problems, a fully dynamic labeling scheme called OrderBased labeling scheme is proposed. Performance evaluation studies show that OrderBased labeling scheme outperforms LSDX and Com-D with respect to labeling time, query performance, and update performance. It is also shown that the total label size for OrderBased labels from 91.1% to 91.95% smaller than label size of LSDX. Even though OrderBased label size is from 2.4% to 7.1% greater than that of Com-D, its efficient querying, labeling and update performance makes it preferable.

References

1. Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Simeon, J.: XQuery 1.0: An XML Query Language. W3C working draft (2001)
2. Clarke, J., DeRose, S.: XML Path Language (XPath) version 1.0. W3C Recommendation (1999)
3. Diets, P.F.: Maintaining Order in a Linked Lists. In: Proceedings of the ACM Symposium on Theory of Computing (1982)
4. Li, Q., Moon, B.: Indexing and Querying XML Data for Regular Path Expressions. In: Proceedings of the VLDB (2001)
5. Yun, J.H., Chung, C.-W.: Dynamic Interval-based Labeling Scheme for Efficient XML Query and Update Processing. The Journal of Systems and Software (2008)
6. Cohen, E., Kaplan, H., Milo, T.: Labeling Dynamic XML Trees. In: Proceedings of the ACM SIGMOD- SIGACT- SiGART (2002)
7. Tatarinov, I., Viglas, S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C.: Storing and Querying Ordered XML Using a Relational Database System. In: Proceedings of ACM SIGMOD (2002)
8. ONeil, P.E., et al.: ORDPATHs: Insert-Friendly XML Node Labels. In: Proceedings of the ACM SIGMOD (2004)
9. Duong, M., Zhang, Y.: LSDX: New Labeling Scheme for Dynamically Updating XML Data. In: Proceedings of 16th Australian Database Conference (2005)
10. Duong, M., Zhang, Y.: Dynamic Labelling Scheme for XML Data Processing. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1183–1199. Springer, Heidelberg (2008)
11. Schmidt, A., Waas, F., Kersten, M., Carey, J., Manolescu, I., Busse, R.: XMark: A Benchmark for XML Data Management. In: Proceedings of VLDB (2002)
12. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann (1993)
13. Mohan, C., Levine, F.: ARIES/IM, An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. SIGMOD Record 21(2), 371–380 (1992)
14. Olson, M.A., Bostic, K., Seltzer, M.I.: Berkeley DB. In: USENIX Annual, FREENIX Track, pp. 183–191 (1999)
15. Ying, L., Jun, M., Yuyin, S.: Applying Dewey Encoding to Construct XML Index of Path and Keyword Query. In: Proceedings of IEEE First International Workshop on Database Technology and Applications (2009)
16. Kha, D.D., Yoshikawa, M., Uemura, S.: A Structural Numbering Scheme for XML Data. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490, pp. 91–108. Springer, Heidelberg (2002)

17. Xu, L., Bao, Z., Ling, T.-W.: A Dynamic Labeling Scheme Using Vectors. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 130–140. Springer, Heidelberg (2007)
18. Xu, L., Ling, T.W., Wu, H.: Labeling dynamic XML Documents: An Order-centric Approach. IEEE Transactions on Knowledge and Data Engineering (2012)
19. Wu, X., Lee, M.L., Hsu, W.: A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In: Proceedings of the 20th Int. Conference on Data Engineering (2004)
20. Gabillon, A., Fansi, M.: A Persistent Labeling Scheme for XML and tree Database. In: Proceedings of ACI (2006)
21. Damien, K., Franky, F., William, L., Shui, M., Wong, R.K.: Dynamic Labeling Schemes for Ordered XML Based on Type Information. In: Seventeenth Australasian Database Conference Technology, vol. 49 (2006)
22. Silberstein, A., et al.: BOXes: Efficient Maintenance of Order-Based Labeling for Dynamic XML Data. In: Proceedings of International Conference on Data Engineering (ICDE). IEEE CS Press (2005)
23. Thonangi, R.: A Concise Labeling Scheme for XML Data. In: International Conference on Management of Data, COMAD 2006, Delhi, India (2006)
24. Xu, L., Ling, T.W., Wu, H., Bao, Z.: DDE: From Dewey to a Fully Dynamic XML Labeling Scheme. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, Providence, Rhode Island, USA, June 29-July 02 (2009)
25. Yun, J.H., Chung, C.W.: Dynamic Interval-Based labeling Scheme for Efficient XML Query and Update Processing. Journal of Systms and Sofware 81, 56–70 (2008)
26. Haw, S.C., Lee, C.S.: Extending Path Summary and Region Encoding for Efficient Structural Query Processing in Native XML Databases. Journal of Systems and Software (2009)
27. Fomichev, A., Grinev, M., Kuznetsov, S.: Sedna: A Native XML DBMS. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 272–281. Springer, Heidelberg (2006)
28. Jagadish, H.V., Al-Khalifa, S., Chapman, A., Lakshmanan, L.V.S., Nierman, A., Paparizos, S., Patel, J.M., Srivastava, D., Wiwatwattana, N., Wu, Y., Yu, C.: TIMBER: A Native XML Database. The VLDB Journal (December 2002)
29. Pal, S., Cseri, I., Seeliger, O., Rys, M., Schaller, G., Yu, W., Tomic, D., Baras, A., Brandon, B., Denis, C., Eugene, K.: XQuery Implementation in a Relational Database system. In: Proceedings of the 31st International Conference on VLDB (2005)
30. Nicola, M., Linden, V.D.: Native XML Support in DB2 Universal Database. In: Proceedings of the 31st International Conference on VLDB (2005)
31. Guangjun, X., Cheng, Q., Jarek, G., Calisto, Z.: Some Rewrite Optimizations of DB2 XQuery Navigation. In: CIKM 2008 (2008)
32. Lu, J., Ling, T.-W., Yu, T., Li, C., Ni, W.: Efficient Processing of Ordered XML Twig Pattern. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 300–309. Springer, Heidelberg (2005)
33. Jiang, Z., Luo, C., Hou, W.-C., Zhu, Q., Che, D.: Efficient Processing of XML Twig Pattern: A Novel One-Phase Holistic Solution. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 87–97. Springer, Heidelberg (2007)
34. Li, J., Wang, J.: Fast Matching of Twig Patterns. In: Bhowmick, S.S., Küng, J., Wagner, R. (eds.) DEXA 2008. LNCS, vol. 5181, pp. 523–536. Springer, Heidelberg (2008)