An Efficient Encoding and Labeling for Dynamic XML Data

Jun-Ki Min[1], Jihyun Lee[2], and Chin-Wan Chung[2]

[1] Korea University of Education and Technology, Korea
jkmin@kut.ac.kr
[2] Korea Advanced Institute of Science and Technoloy, Korea
{hyunlee,chungcw}@islab.kaist.ac.kr

Abstract. In order to efficiently determine structural relationships among XML elements and to avoid re-labeling for updates, much research about labeling schemes has been conducted, recently. However, a harmonic support of efficient query processing and updating has not been achieved. In this paper, we propose an efficient XML encoding and labeling scheme, called EXEL, which is a variant of the region numbering scheme using bit strings. In order to generate the ordinal and insert-friendly bit strings in EXEL, a novel binary encoding method is devised. Also, we devise a labeling scheme for a newly inserted node which incurs no re-labeling of pre-existing labels. These encoding and inserting methods are the bases of efficient query processing and the complete avoidance of re-labeling for updates. Moreover, EXEL supports all structural relationships in XPath and the relationships can be checked by SQL statements supported by an RDBMS. Finally, the experimental results show that EXEL provides fairly reasonable query processing performance while completely avoiding re-labeling for updates.

Keywords: Dynamic XML, Labeling and Update.

1 Introduction

Due to its flexibility and a self-describing nature, XML [2] is considered as the *de facto* standard for data representation and exchange in the Internet. In order to search the irregularly structured XML data, path expressions are commonly used in XML query languages, such as XPath [4] and XQuery [14].

Basically, XML data comprises hierarchically nested collections of elements, where each element is bounded by a start tag and an end tag that describe the semantics of the element. Generally, an XML data is represented as a tree such as DOM [12]. The tree of XML data is implicitly ordered according to the visiting sequence of the depth first traversal of the element nodes. This order is called the *document order*.

Given a tree of XML data, the path information and the structural relationships of nodes should be efficiently evaluated. Diverse approaches such as path index approaches [7,3] and the reverse arithmetic encoding [10] provide help for obtaining the list of nodes which are reached by a certain path.

In order to facilitate the determination of structural relationships of nodes (e.g., the ancestor-descendent relationship), various labeling methods such as region numbering scheme [17,9] and prefix based scheme [15] have been proposed. In addition, structural modifications to the XML data can occur. For example, insertions of nodes change the structure of a tree of XML data, and the assigned labels may need to be changed. Thus, many researches [1,5,16,11,8] have been conducted in order to provide an efficient way to handle labels for updating XML data. However, they still cannot entirely remove re-labeling for insertions.

Our Contribution. In this paper, we devise a novel XML encoding and labeling scheme, called EXEL (Efficient XML Encoding and Labeling). EXEL is effective to compute the structural relationships as well as to support the incremental update. The contributions of the paper are as follows:

– **Devise a novel binary encoding:** we devise a novel binary encoding method to generate bit strings which are ordinal and insert-friendly. We extend the region numbering scheme using the bit strings instead of decimal values. The efficient query processing and complete avoidance of re-labeling are based on our binary encoding method.
– **Completely remove re-labeling for updates: we devise a labeling scheme** for a newly inserted node. In our scheme, re-labeling of pre-existing labels for insertion can be completely avoided.
– **Support full axes:** EXEL supports all structural relationships in XPath and the relationships [1] can be checked by SQL statements supported by an RDBMS.

The remainder of the paper is organized as follows. In Section 2, we review various XML labeling schemes. We describe the details of EXEL in Section 3 and present an update method of EXEL in Section 4. Section 5 contains the results of our experiments. Finally, in Section 6, we summarize our work.

2 Related Work

In the region numbering scheme [17,9], each node in a tree of XML data is assigned a region consisting of a pair of start and end values which are determined by the positions of the start tag and the end tag of the node, respectively. Even though all structural relationships represented in XPath can be determined efficiently using <start, end, level>, an insertion of a node incurs re-labeling of its following and ancestor nodes. [9,1] have tried to solve the re-labeling problem by extending a region and using float-point values. However, the re-labeling problem can not be avoided for frequent insertions after all.

In the prefix labeling scheme [15,5,11], each node in a tree of the XML data has a string label which is the concatenation of the parent's label and its own identifier. The structural relationships among nodes can be determined by a

[1] In XPath, there are 13 axes. In this paper, we do not consider namespace, and attribute axes since they are not structural relationships.

string function to extract a prefix of a string and string comparison operations. These function and operators degrades a query performance. Dewey labeling scheme [15] and Binary labeling scheme [5] do not require re-labeling for appending leaf nodes. However, they cannot avoid the re-labeling for insertions between two sibling nodes and an insertion of a node between parent and child nodes. Recently proposed ORDPATH [11] is tolerant for insertions. ORDPATH follows a labeling principle similar to the Dewey labeling scheme. In order to avoid re-labeling, it uses only odd numbers for initial labels. When an insertion occurs, it uses an even number between two odd numbers and concatenates an odd number. Although ORDPATH is more bearable for insertions than other approaches, they cannot also avoid re-labeling for an insertion between parent and child nodes.

The prime number labeling scheme [16] uses an inherent feature of the prime number which has only one and itself as its common divisors. The label of a node is a product of its parent node's label and its self-label (i.e., a unique prime number). For the order sensitive query, the prime number labeling scheme uses the simultaneous congruence (SC) values. Even though re-labeling for nodes can be avoided for insertions, the SC values should be re-calculated, and the re-calculation consumes much time. Also, an insertion between parent and child nodes incurs the re-labeling.

In addition, a dynamic quaternary encoding, QED [8], that can be applied to different labeling schemes, has been proposed. In QED, the label size increases by two bits for inserting a node. In contrast, the label size increases by one bit for the insertion in our scheme.

3 Efficient XML Encoding and Labeling (EXEL)

In this section, we present a novel binary encoding method for labeling XML data and an enhanced encoding method to reduce label length. We use the bit strings in the region numbering scheme instead of decimal values for the efficient query processing and the complete elimination of re-labeling for updates.

3.1 Binary Encoding in EXEL

The original region numbering scheme uses decimal values for labels which are sensitive of updates. Therefore, we propose a novel efficient XML encoding and labeling method, called EXEL. It uses bit strings which are ordinal as well as insert-friendly. The bit strings for labeling are generated by the following binary encoding method:

(1) The first bit string $b(1) = 1$.
(2) Given the i^{th} bit string $b(i)$, if $b(i)$ contains 0 bit then $b(i+1) = b(i) + 10$. Otherwise, $b(i+1) = b(i)0^k1$, where k is the length of $b(i)$.

Definition 1. Lexicographical order ($<$)
(i) 0 is lexicographically smaller than 1 ($0 < 1$).
(ii) if two bit strings a and b are the same(=), a is lexicographically equal to b.
(iii) Given bit strings a, b, a' and b', $ab < a'b'$, if only if $a < a'$ or $a = a'$ and $b < b'$ or $a = a'$ and b is null, where $length(a) = length(a')$.

Bit strings generated by the above binary encoding method have the lexicographical orders presented in Definition 1. For example, 1<101<111<1110001. Also, according to the above generating rule, the bit string always ends with 1. Thus, our encoding scheme satisfies the following property.

Property 1. Given bit strings $s_1 1$ and $s_2 1$ generated by the above binary encoding method, if $s_1 1 < s_2 1$, then $s_1 < s_2$ in the lexicographical order.

Theorem 1 presents the space requirement of our binary encoding scheme.

Theorem 1. *In order to encode N ordinal values, the binary encoding of EXEL needs $2^{\lceil log_2 log_2 N+1 \rceil} - 1$ bits, which is about $2 log_2 N - 1$ bits.*

Proof. (i) 1-bit string (i.e., 1) can represent only 1 value. (ii) 3-bit string (i.e., 101, 111) can represent 2 values. (iii) 7-bit string (i.e., 1110001,...,1111111) can represent 2^3 values. (iv) consequently, by the mathematical induction on k, $(2^k - 1)$-bit string can represent $2^{2^{k-1}-1}$ values.

Let $N = 2^0 + 2^1 + ... + 2^{2^{k-1}-1} = \sum_{i=1}^{k} 2^{2^{i-1}-1}$.
By the mathematical induction, $N = \sum_{i=1}^{k} 2^{2^{i-1}-1} < 2 * 2^{2^{k-1}-1} = 2^{2^{k-1}}$.
Therefore $k = \lceil log_2 log_2 N+1 \rceil$. Consequently, $2^k - 1 = 2^{\lceil log_2 log_2 N+1 \rceil} - 1 \approx 2 log_2 N - 1$.■

3.2 Enhancement of Binary Encoding

In the binary encoding method of EXEL, k-bit string has superfluous $(k - 1)/2$ bits consisting of only 1 in order to guarantee the lexicographical order among variable length bit strings. For example, the 7-bit strings from 1110001 to 1111111 can represent only $2^3 = 8$ ordinal values since the first three bits are 111 and the last bit is 1. In order to remove the superfluous part, we devise another binary encoding method with a predefined length of a bit string. The predefined length is obtained from the total number of ordinal values which would be encoded. For labeling a tree of XML data, it is determined by the total number of nodes. The bit string with a predefined length is generated by the following rule:

Let N be the total number of values.
(1) The first bit string $b(1) = 0^{log_2 N}1$.
(2) Given i^{th} bit string $b(i)$, $b(i+1) = b(i) + 10$.

According to the above rule, a bit string ends with 1 like the original binary encoding method, and it satisfies Property 1.

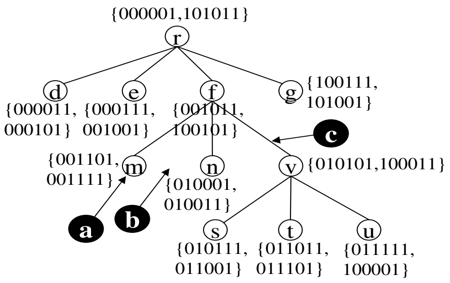


Fig. 1. An example of a tree labeled by EXEL

3.3 Region Labeling in EXEL

In EXEL, we extend the region numbering scheme using ordinal bit strings rather than decimal values. The start and end values are ordinal bit strings generated by the binary encoding of EXEL. Fig 1 shows an example of a tree labeled by EXEL using the binary encoding with a predefined length.

EXEL uses the parent information to determine the parent and child relationships in order to support efficient updates although the original region numbering scheme uses the level information. The use of the parent information also results in an improvement of the query performance. For the simplicity, the parent information (i.e., the start value of the parent's label) is not presented in Fig 1.

The following theorem presents the space requirement of EXEL using the binary encoding with a predefined length.

Theorem 2. *Given the total number of nodes N, the binary encoding using a predefined length in EXEL needs $log_2 2N + 1$ bits for each bit string. Also, the region labeling in EXEL requires $3(log_2 2N + 1)$ for start, end, and parent information.*

3.4 Query Processing

EXEL supports all XPath axes (i.e., ancestor, descendent, parent, child, following, preceding, following-sibling, and preceding-sibling) as the same way in the region numbering scheme because EXEL is based on the original region numbering scheme. For example, in Fig 1, f is an ancestor of t since $s_f(= 001011) < s_t(= 011011)$ and $e_t(= 011101) < e_f(= 100101)$, where (s_x,e_x) is the region of a node x.

For many years, intensive research on storing and managing XML data as the relational data have been conducted. By using an RDBMS, we can utilize the stable repository as well as the efficient query optimizer and the executor. Therefore, we store XML data into relational tables. For a node x in a tree of XML data, the relational table stores its region (s_x, e_x) and its parent's information ps_x. All above conditions for the corresponding axes in the XPath can be expressed by simple SQL statements supported by an RDBMS. For example, the SQL statement to find all descendents of a node f in Fig 1 is SELECT * FROM NODE WHERE 001011 < start AND end < 100101.

4 Update

In this section, we present the update behaviors of EXEL. Since the deletion does not incur the re-labeling of nodes, we present the algorithm for the insertion.

4.1 Labeling for Update

The algorithm *MakeNewBitString* makes a new bit string between two pre-existing bit strings. This algorithm can be applied to the original binary encoding and the binary encoding with a predefined length. $\bigoplus$ denotes the concatenation of two bit strings.

```
Algorithm MakeNewBitString( $leftB$, $rightB$ )
begin
1.    if length($leftB$) > length($rightB$) then $newB$ := $leftB \bigoplus 1$;
2.    else $newB$ := ($rightB$ with the last bit changed to 0)$\bigoplus 1$;
3.    $newB$ := $newB \bigoplus 1$;
4.    return $newB$;
end
```

For example, when we insert two bit strings successively between 101 and 111, the first one is 1101 (101<1101<111) and second one is 11011 (1101<11011<111). For bit strings generated by our binary encoding method, the lexicographical order has a property as follows.

Property 2. Given bit strings $s_1 1$ and $s_2 1$ generated by the binary encoding method of EXEL, if $s_1 1 < s_2 1$, then $s_1 1 < s_2 01$ and $s_1 11 < s_2 1$.

For example, let $s_1 1 = 000011$ and $s_2 1 = 000101$, then $000011 < 0001001$ and $0000111 < 000101$. Through the above property, we can explain that a new bit string generated by the algorithm *MakeNewBitString* preserves the lexicographical order among pre-existing bit strings.

Theorem 3. *The bit string generated by the algorithm MakeNewBitString preserves the lexicographical order.*

Proof. If length($leftB$) > length($rightB$), then $leftB < newB(=leftB \bigoplus 1)$ (by Definition 1) and $newB < rightB$ (by Property2). Otherwise, given $leftB = s_1 1$ and $rightB = s_2 1$, $newB(=s_2 0 \bigoplus 1) < rightB(=s_2 1)$ (by Definition 1), and $leftB(=s_1 1) < newB(=s_2 01)$ (by Property 2) ■

4.2 Update Processing

There are three kinds of insertions in XML data according to the positions in which nodes are inserted; inserting a child of a leaf node, inserting a sibling and inserting a parent.

```
Algorithm InsertChildOf( $cur$ )
begin
1.    $s_{new}$ := MakeNewBitString($s_{cur}$, $e_{cur}$);
2.    $e_{new}$ := MakeNewBitString($s_{new}$, $e_{cur}$);
3.    $ps_{new}$ := $s_{cur}$;
4.    Insert new node with $s_{new}$, $e_{new}$, and $ps_{new}$;
end
```

The algorithm *InsertChildOf* inserts a node as a child of a leaf node *cur*. In EXEL, a region of a child node is contained in that of its parent node. Thus, a region of a inserted node *new*, (s_{new}, e_{new}) should satisfy $s_{cur} < s_{new} < e_{new} < e_{cur}$. The algorithm *MakeNewBitString* is used to make the start and end values of a node *new*. Additionally, the parent information of *new*, ps_{new} will be the start value of *cur* (i.e., s_{cur}). For example, in Fig 1, a node a is inserted into a child of a leaf node m. The region of a, (s_a, e_a) should satisfy $s_m < s_a < e_a < e_m$. Thus, s_a=0011101, e_a=00111011, and ps_a=001101. Algorithms of inserting a sibling and a parent (e.g., in Fig 1, inserting b and c, respectively) are similar to that of inserting a child. We omit them for want of space.

The insertion of a parent incurs the increase of levels of its all descendants in the original region numbering. However, EXEL keeps the parent information instead of the level. Even if a node is inserted as an ancestor, the parents of the descendent are still unchanged except the child of the inserted node. Consequently, EXEL guarantees the complete avoidance of re-labeling for insertions in any positions even between parent and child nodes.

Even in case of a subtree insertion, the labeling can be efficiently handled. We first apply our labeling method to the subtree. Second, we generate a new bit-string x according to the inserting point (p, q) using the algorithm *MakeNewBitString*, then truncate the last bit (i.e., '1') of x. Let the truncated bit-string be x'. We complete labeling for the subtree by attaching x' as a prefix into the labels of the subtree's nodes. Since the prefixes of subtree's nodes are equal, lexicographical orders among labels of subtree's nodes are preserved. Note that, in the lexicographical order, $p \leqslant x' < q$. Thus, by Definition 1-(iii), labels of subtree's nodes (whose prefixes are x') are greater than p and smaller than q. Therefore, the generated labels still keep the lexicographical order among the pre-existing labels.

5 Experiments

We empirically compared the performance of EXEL with the those of region numbering scheme, the prefix labeling schemes (i.e., ORDPATH and QED-PREFIX), and the prime numbering scheme using synthetic data as well as real-life XML data sets.

5.1 Experimental Environment

The experiments were performed on an Intel Pentium 3GHz with 1GB memory, running Window XP. The XML data sets were stored on an RDBMS,

Table 1. XML Data Set

Data	Name	Size(MB)	# of nodes
XMark	XM1	1	33152
	XM50	50	1390697
	XM115	115	3231322
Shakespeare	S7	7.7	328778

Table 2. Database size

Labeling Scheme	DB Size(MB)
EXEL2 (EXEL1)	147(163)
Region Numbering	119
QRDPATH	140
QED-PREFIX	161
Prime	146

Table 3. Query Set

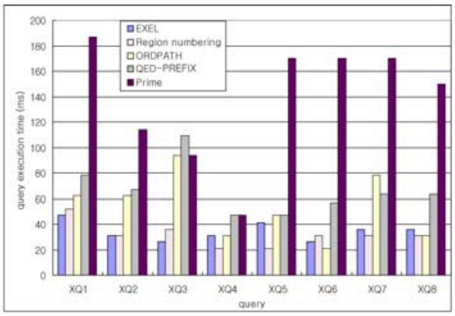
Name	Query Definition	Name	Query Definition
XQ1	//bidder/ancestor::open_auction	SQ1	//SPEAKER/ancestor::ACT
XQ2	//parent//city	SQ2	//ACT//LINE
XQ3	//name/parent::person	SQ3	//SCENE/parent::ACT
XQ4	//person/name	SQ4	//ACT/TITLE
XQ5	//category/following::person	SQ5	//ACT[3]/following::SPEECH
XQ6	//buyer/preceding::category	SQ6	//TITLE/preceding::ACT
XQ7	//person[2]/following-sibling::person	SQ7	//ACT[6]/following-sibling::ACT
XQ8	//item[46]/preceding-sibling::item	SQ8	//SPEECH[40]/preceding-sibling::SPEECH

PostgreSQL 8.1. We have implemented two versions of EXEL; EXEL1 and EXEL2 using the original binary encoding and the binary encoding with a predefined length, respectively. The storage spaces according to the encoding methods have been observed. In order to show the efficiency of EXEL for the query processing and the update processing, we compared EXEL with other representative labeling methods; the region numbering scheme, ORDPATH, QED-PREFIX and the prime numbering scheme. In our experiments, QED was applied to prefix labeling scheme, and we call it QED-PREFIX. In database, we store each element name, its label, and its parent's label according to the labeling schemes. We evaluated EXEL using XMark data [13] and Shakespeare data [6]. The characteristics of the data sets are summarized in Table 1.

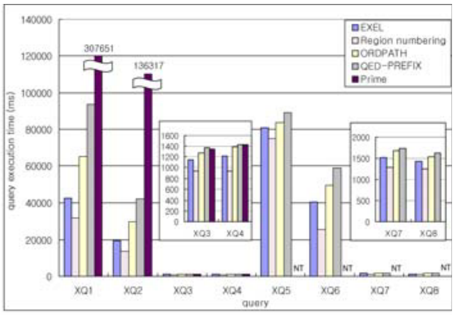
The queries used in our experiments are described in Table 3. The first character in a query name indicates the data set on which the query is executed: 'X' denotes XMark, and 'S' is for Shakespeare. We evaluated the query performance for all axes in XPath. The number in a query name denotes the type of a query according to the axis contained in the query (i.e., 1 for ancestor, 2 for descendent, 3 for parent, 4 for child, 5 for following, 6 for preceding, 7 for following-sibling, and 8 for preceding-sibling). All experiments were repeated 10 times and we used the average of the processing times except the minimum and maximum values.

5.2 Experimental Results

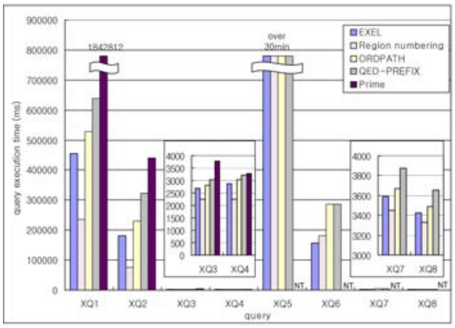
Query Performance. In our experiments, a given XPath query was transformed to the corresponding SQL statement according to labeling schemes, and the SQL statements were executed on a database. A query parsing time and a query translation time are similar and very small for all labeling schemes. Therefore, in this paper, we present only SQL execution time. The query execution time over various sized data sets are shown in Fig 2. The notation NT means NOT MEASURED due to excessive processing time.



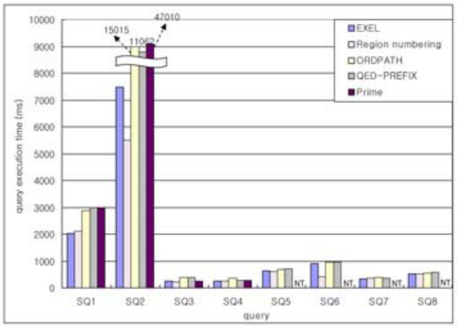
(a) XM1



(b) XM50



(c) XM115



(d) S7

Fig. 2. Query execution time

First of all, ORDPATH and QED-PREFIX should additionally use a string function to extract prefix of a string to compute structural relationships. The use of the string function degrades their performance. In Fig 2(b), the performance of EXEL is about 1.5 times better than that of ORDPATH and 2 times than QED-PREFIX for the query XQ1. As the data size increases, differences of processing time increases.

For queries with ancestor (i.e.,XQ1), descendent (i.e.,XQ2), following(i.e.,X5) and preceding (i.e.,XQ6) relationships, the region numbering scheme is better than EXEL since the label size of the region numbering scheme is smaller than that of EXEL. However, the performance of EXEL is close to that of the region numbering scheme compared with other labeling schemes.

Additionally, for finding a parent (i.e., XQ3), children (i.e., XQ4) and siblings (i.e., XQ7 and XQ8), the difference of the query performance among the labeling schemes is not considerable due to the use of the parent information. However, EXEL is still better than ORDPATH and QED-PREFIX.

The prime numbering scheme shows bad performance for ancestor and descendent relationships since it uses the *mod* operation which is more expensive

Table 4. The performance of inserting

Inserting a child						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
EXEL		31	0		16	0
ORDPATH	XM1	31	0	XM50	32	0
QED-PREFIX		31	0		5	0
Region numbering		156	11205		15203	468736
EXEL		15	0		15	0
ORDPATH	XM11	5	0	XM115	15	0
QED-PREFIX		16	0		16	0
Region numbering		3485	109368		27203	1089457

Inserting a sibling						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
EXEL		47	0		688	0
ORDPATH	XM1	47	0	XM50	922	0
QED-PREFIX		63	0		906	0
Region numbering		235	17039		21890	717133
EXEL		187	0		1469	0
ORDPATH	XM11	282	0	XM115	3359	0
QED-PREFIX		234	0		4922	0
Region numbering		5484	167772		38750	1666222

Inserting a parent						
Labeling Scheme	Data	Time(ms)	Re-labeling	Data	Time(ms)	Re-labeling
EXEL		31	1		1656	1
ORDPATH	XM1	63	3344	XM50	5734	141572
QED-PREFIX		63	3344		6350	141572
Region numbering		125	11429		19141	469031
EXEL		453	1		3703	1
ORDPATH	XM11	609	32667	XM115	10266	330135
QED-PREFIX		834	32667		13016	330135
Region numbering		3938	109612		31187	1089687

than the comparison operations for integers and bit strings. For order sensitive queries (i.e., query type 5, 6, 7 and 8), the performance is poorer than those of other labeling schemes since SC-values should be used to compute the document order of a node. Moreover, it takes very long time to compute SC values by an algorithm in [16] even for small data. Thus, we could not measure the query time for the order sensitive queries for over 1MB data.

Consequently, EXEL is superior to the prefix labeling scheme and the prime numbering scheme over all cases. Also, EXEL is comparable with the region numbering scheme than others. This is achieved by the binary encoding scheme generating the ordinal bit strings which can be effectively adopted to the region numbering scheme.

Upate Performance. We evaluated the performance of three kinds of insertions; inserting a child node of a leaf node, inserting a next sibling node of a node, and inserting a parent node. In our experiments, we excluded the prime numbering scheme since it requires very expensive re-calculations of SC values even for small data. The influence of inserting a subtree on pre-existing labels is the same as that of inserting a node. Therefore, we omitted the experiment of inserting a subtree. In order to evaluate update performance, we randomly selected a node (a leaf node for inserting a child) for each kind of insertion and inserted a node as its child, next sibling, or parent. For fair comparisons, we used the same node for all labeling schemes. Table 4 shows the performance of inserting a node.

In the region numbering scheme, the re-labeling was inevitable for all kinds of insertions. For inserting a child to a leaf node, other labeling schemes did not require re-labeling after the insertion. For inserting a next sibling node, in EXEL and ORDPATH, re-labeling of nodes is not incurred. However, in order to generate a label for a newly inserted node, they need to know the label of the next sibling. The performance of EXEL to find the following-sibling of a node is better than those of other labeling schemes, so the time spent to insert a sibling node in EXEL is smaller than those in others. For inserting a node between parent and child nodes, ORDPATH should re-assign labels for the child and its all descendents. EXEL keeps the parent information which is invariant for insertions of ancestors except a parent. Therefore, in EXEL, only one update was incurred. EXEL needs the labels of the previous and next sibling nodes to generate a new label for an inserted node. However, the time to find labels of siblings is much smaller than the time for re-labeling.

In summary, EXEL achieves the complete removal of re-labeling for insertions. Therefore, EXEL can save much time for updates. Since the time measure smaller than 100ms is unstable and less significant, the comparison of execution time over 100ms shows that the update performance of EXEL is 2.3 times on the average and up to 3.8 times better than those of ORDPATH, with the performance gap increasing as the size of XML data gets larger.

Storage Space. Table 2 shows the size of the databases where XM50 is stored using each labeling scheme. EXEL2 reduces the space requirement effectively. EXEL2 requires an additional scan of data to count the number of nodes before labeling. However, the time for the preprocessing is much smaller than the total storing time. Although EXEL2 uses three binary coding values (i.e., start, end, and parent's start), the space requirement is only slightly larger than ORDPATH and Prime numbering scheme. However, the query performance of EXEL is better than them as shown through the experiment results. Moreover, the database size of QED-PREFIX is bigger than EXEL2. EXEL needs a larger space than the region numbering scheme due to the use of the insert-friendly bit string. However, the significant improvement of the update performance according to the use of the bit strings compensates for the space overhead.

6 Conclusion

We propose EXEL, an efficient XML encoding and labeling method which supports efficient query processing and updates. A novel binary encoding method used in EXEL generates ordinal and insert-friendly bit strings. EXEL is a variant of the region numbering scheme using bit strings generated by the novel binary encoding method. EXEL supports all axes in XPath, and the conditions to compute the structural relationships can be simply expressed by SQL statements of an RDBMS. Furthermore, we proposed a labeling method for a newly inserted node, so EXEL removes the re-labeling overhead entirely unlike other existing labeling schemes. The experimental results show that EXEL provides fairly reasonable query performance. Also, the update performance of EXEL is better than those of existing labeling schemes, with performance gap increasing as the size of XML data gets larger.

Acknowledgements. This research was supported by the Ministry of Information and Communication, Korea, under the College Information Technology Research Center Support Program, grant number IITA-2006-C1090-0603-0031.

References

1. T. Amagasa and M. Yoshikawa. QRS: A Robust Numbering Scheme for XML documents. In *Proc. of ICDE 2003*, pages 705–707, 2003.
2. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation, http://www.w3.org/TR/REC-xml, 2004.
3. C.-W. Chung, J.-K. Min, and K.-S. Shim. APEX: An Adaptive Path Index for XML Data. In *Proc. of ACM SIGMOD 2002*, pages 121–132, 2002.
4. J. Clark and S. DeRose. XML Path Language(XPath) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xpath, 1999.
5. E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In *Proc. of PODS 2002*, pages 271–281, 2002.
6. R. Cover. The XML Cover Pages. http://www.oasis-open.org/cover/xml.html, 2001.
7. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. of VLDB 1997*, pages 436–445, 1997.
8. C. Li and T. W. Ling. QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates. In *Proc. of ACM CIKM 2005*, pages 501–508, 2005.
9. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Expressions. In *Proc. of VLDB 2001*, pages 367–370, 2001.
10. J.-K. Min, M.-J. Park, and C.-W. Chung. XPRESS: A Queriable Compression for XML Data. In *Proc. of ACM SIGMOD 2003*, pages 122–133, 2003.
11. P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *Proc. of ACM SIGMOD 2004*, pages 903–4908, 2004.
12. R. W. Philippe Le Hegaret and L. Wood. XML Path Language(XPath) Version 1.0. http://www.w3.org/DOM, 2005.
13. A. Schmidt, F. Waas, M. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: A Benchmark for XML Data Management. In *Proc. of VLDB*, pages 974–985, 2002.
14. D. C. Scott Boag, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. W3C Recommendation, http://www.w3.org/TR/xquery/, 2005.
15. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *Proc. of ACM SIGMOD 2002*, pages 204–215, 2002.
16. X. Wu, M. L. Lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In *Proc. of ICDE 2004*, pages 66–78, 2004.
17. C. Zhang, J. Naughton, D. Dewitt, Q. Luo, and G. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *Proc. of ACM SIGMOD 2001*, pages 425–436, 2001.