論文 / 著書情報
Article / Book Information

Title	Storage Consumption of Variable-length XML LabelsUninfluenced by Insertions
Author	Akihiro Takahashi, Wenxin Liang, Haruo Yokota
Journal/Book name	Proceedings of the Second IEEE International Conference on Digital Information Management, , , pp. 571-573
Issue date	2007, 10
DOI	http://dx.doi.org/10.1109/ICDIM.2007.4444284
URL	http://www.ieee.org/index.html
Copyright	(c)2007 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Storage Consumption of Variable-length XML Labels Uninfluenced by Insertions

Akihiro Takahashi
Tokyo Institute of Technology
akihiro@de.cs.titech.ac.jp

Wenxin Liang
Japan Science and Technology Agency
Tokyo Institute of Technology
wxliang@de.cs.titech.ac.jp

Haruo Yokota
Tokyo Institute of Technology
yokota@cs.titech.ac.jp

Abstract

In recent years, the method of assigning labels to the nodes of an XML tree is getting more attraction. Various functions in an RDBMS can be easily utilized by storing the labeled XML documents into the RDB. However, in traditional labeling methods, a number of nodes need to be relabeled, when the XML documents are updated. To address this problem, we proposed DO-VLEI code combining VLEI code with the Dewey Order method. DO-VLEI code is effective to reduce the update cost, but the label size increases rapidly when handling large XML documents. To reduce the label size, we presented Compressed-bit-string DO-VLEI (C-DO-VLEI) code. However, it is difficult to handle the length of C-DO-VLEI because it is a variable-length code. In this paper, we propose two effective methods, VLEI-ABL and VLEI-EOL for handling the code length of C-DO-VLEI. We perform experiments to compare the storage consumption of the proposed methods with the previously known ORDPATH. The experimental results show that our methods considerably outperform the ORDPATH.

1 Introduction

Relational Database systems (RDBs) are now widely used for storing XML data, so that their management functions can be used for retrieval, updating, transactions, storage, etc. An XML document can be considered as a tree structure by treating each element between a start tag and end tag as a node [3], because the XML document forms a nested construction of elements enclosed by these tags.

The method of assigning labels to the nodes of an XML tree is called a labeling scheme. The labeling scheme makes it possible to determine the containment relationships between nodes and the positions of the nodes in the XML tree.

A number of labeling schemes based on the Dewey Order (DO) numbering method [9] have been proposed. The DO numbering method uses a delimiter to extend the label of a parent node and includes the code that represents the child's sibling order. However, the naive DO labeling scheme, which uses numerical values for the codes, has the problem of expensive update cost. When a new node is inserted into an intermediate position of the XML tree, a number of nodes need to be re-labeled.

To address this problem, we proposed Variable-Length-Endless-Insertable code (VLEI code) and DO-VLEI labeling method [6] combining the VLEI code with the DO numbering method. It uses the VLEI code [6] for representing the sibling orders, which enables unlimited insertion of new nodes without relabeling any nodes.

DO-VLEI code is effective to reduce the update cost. However, the label size increases rapidly when handling large XML documents. To reduce the label size, we presented Compressed-bit-string DO-VLEI (C-DO-VLEI) code which represents the label by using variable-length bit-strings instead of using character strings in DO-VLEI code. However, it is difficult to handle the length of the code.

In this paper, we propose two effective methods, VLEI-ABL and VLEI-EOL for handling the code length of C-DO-VLEI. We perform experiments to compare the storage consumption of the proposed methods with the previously known ORDPATH. The experimental results show that our methods considerably outperform the ORDPATH.

The rest of the paper is organized as follows. In Section 2, we introduce DO-VLEI and C-DO-VLEI labeling methods. Section 3 describes the ORDPATH. In Section 4, we describe the properties of C-DO-VLEI code. Section 5 discusses two methods for handling the variable-length code. In Section 6, we show the experimental evaluation of the storage consumptions comparing the proposed methods with the ORDPATH. Section 7 summarizes the paper and discusses future directions.

2 DO-VLEI

We first explain the VLEI code and the DO-VLEI code.

2.1 Definition of the DO-VLEI Code

Definition 1 (VLEI code) *Assume v is a bitstring comprising bits from $\{0,1\}$ whose initial bit is 1. v is a VLEI code, if it satisfies the following order:*

$$v \cdot 0 \cdot \{0|1\}^* < v < v \cdot 1 \cdot \{0|1\}^*$$

A new VLEI code between two arbitrary adjacent VLEI codes can be unrestrictedly generated [6][5].

In DO numbering method, the label for a child node is expressed by adding a delimiter and a sibling code that represents the sibling order. In the DO-VLEI labeling method, the sibling code is expressed by the VLEI code.

Definition 2 (DO-VLEI)

1. *The DO-VLEI label of a root node is assumed to be 1.*
2. *A sibling code C_{child} is a VLEI code in which the order between sibling nodes is expressed by the VLEI code. If the label of a parent node is L_{parent}, the DO-VLEI label of the child node comprises a delimiter ".", L_{parent}, and C_{child} as follows.*

   DO-VLEI = L_{parent} . C_{child}

2.2 Compressed-bit-string DO-VLEI

DO-VLEI code is composed by character strings. Thus, when it handles large XML documents, the size of DO-VLEI code may increase exponentially. Therefore, an effective method for reducing the label size becomes critical. In [7], we have proposed the Compressed- bit-string DO-VLEI (C-DO-VLEI) labeling method, in which it expresses both the DO-VLEI codes and the delimiters by bitstrings.

The Compressed-bit-string DO-VLEI(C-DO-VLEI) is defined as follows:

Definition 3 (Compressed-bit-string DO-VLEI) *Corresponding to DO-VLEI, ".1" is expressed as the bit string (11), the element "1" is expressed by bitstring (10) and the element "0" is expressed by the bitstring (0). This bitstring expression of DO-VLEI is called Compressed-bit-string DO-VLEI (C-DO-VLEI).*

3 Related Work

Similar to the DO-VLEI code method, some labeling methods that can update the XML document without relabeling have been proposed[8][4][12]. In the following subsection, we briefly introduce ORDPATH which is the comparison object in our experiments.

3.1 ORDPATH

The ORDPATH label of a child node is made from a sibling code, a delimiter, and the label of the parent node based on the DO numbering method.

In ORDPATH, only positive odd numbers are used for the initial labeling, because negative integers and even numbers are used for insertion operations. Besides, the end of a sibling code, which decides the sibling order, must be an odd number. For example, when a node is newly inserted between nodes"1.3.1" and "1.3.3", the label of the new node can be "1.3.2.1" , where the sibling code is "2.1".

3.1.1 Compressed-bit-string ORDPATH

ORDPATH is implemented by bitstring expressions using "0" and "1". In this paper, the binary representation of ORDPATH is called the Compressed-bit-string ORDPATH (C-ORDPATH). An integer in ORDPATH is translated into a bitstring by using a prefix schema. A pair of bitstrings L_i and O_i are used to express an integer. A label is a string of delimited integers, and i is the order of the integer within the string. L_i is a bitstring specifying the number of bits of O_i. L_i is specified by analysis of the tree using a prefix schema. O_i is treated as a binary number within a range set by L_i.

Table 1. Algorithm: CompareDO-VLEI

Algorithm CompareDO-VLEI(v, w)
! ! **Input**
! !! !! ! v, w :C-DO-VLEI code to which are to be compared
! !! !! !! ! (where length(v)≤length(w))
! ! **Output**
! !! !! ! v is greater than w

$l' = length(w) - length(v)$
$v' = v \cdot 1 \cdot \{0\}^{l'-1}$
if $v' \geq w$
return true
else
return false
endif

4 Properties of C-DO-VLEI

The C-DO-VLEI code keeps the property of DO, because we convert each element of DO-VLEI code into bitstrings.

Let "0" = v_0 ! $, ".1" = v_d ! $, "1" = v_1. If

$$1 \cdot v_0 \prec_v 1 \prec_v 1 \cdot v_d \prec_v 1 \cdot v_1 \quad (1)$$

then, the ascending order by using the comparison algorithm shown in Table 1 equals to the XML document order[1].

5 Handling Code Length

It is difficult to extract the C-DO-VLEI label from bitstring because the length of labels is variable. In order to derive the length of each label, we propose two methods. The one provides the length of each code at the head of the bitstring, which is called VLEI-ABL. The other adds a terminal symbol (EOL) to the end of each code, which is called VLEI-EOL.

5.1 VLEI-ABL

At first we generate the C-DO-VLEI code. After that we figure out the length of it, and add a part specifying the length at the head of it. We call that code VLEI-Add Bit Length (VLEI-ABL). In the VLEI-ABL, we use the variable-bitstring component L/O just as C-ORDPATH. The prefix schema of Table 2 is used for our experiments.

5.2 VLEI-EOL

The length of a variable-length code can be also handled by adding a terminal symbol, EOL (End of Label) to the end of the label. In this case, the elements representing the DO-VLEI code becomes: ".1","1", "0" and "EOL". Next, we discuss how to convert the four elements into bitstrings.

[1] See [11] for the prove.

Table 2. Prefix Schema for Length

L Bitstring	O length	O value range
0	3	[1, 7]
100	4	[8, 23]
101	6	[24, 87]
1100	8	[88, 343]
1101	12	[344, 4439]
11100	16	[4440, 69975]
11101	20	[69976, 1118551]
11110	24	[1118552, 17895767]
11111	31	[17895768, 2165379414]

Table 3. Summary of VLEI-ABL and VLEI-EOL

	vlei-ABL	vlei1	vlei2	vlei3	vlei4	vlei5	vlei6
0	0	00	0	0	0	00	00
1	10	11	10	110	11	1	1
.1	11	10	110	10	10	01	01
EOL		01	111	111	1111	0101	0110

At first we present the Huffman-coding based method. "EOL" has the lowest occurrence frequency because it appears only once in each label. ".1" occurs $(depth) - 1$ times in each label, because it is the only element that represents the delimiter. "0" and "1" occur at least once in each sibling number.

$$\text{“0”} > \text{“1”} <> \text{“.1”} > \text{“EOL”} \quad (2)$$

Inequality (2) shows the occurrence frequency of each element when the frequency of "0" is greater than "1". According to Inequality (2), the vlei 1–3 show in Table 3 can be generated based on the Huffman-coding.

On the other hand, since the occurrence frequency of "EOL" is apparently smaller than that of other symbols, we can define specific sign patterns as "EOL". The patterns of "EOL" used in vlei 4-6 are shown in Table 3. Since the "EOL" has the lowest occurrence frequency, the code length of vlei 4-6 can be decreased.

When XML documents are labeled by vlei1,3,4 , we can use the comparison algorithm of Table 1.

6 Experiment

We assign the labels to the nodes of XML documents by VLEI-ABL, VLEI-EOL and C-ORDPATH to compare their storage consumptions. Because the C-ORDPATH is also variable-length codes, we add the bit length information to the C-ORDPATH. We use 23 XML documents in the experiments that are taken from XML Data Repository [1] and generated from XMLGEN [2], which is a XML document generator for XMark[10].

6.1 Experimental Results

The total label size of each method in the initial state is measured. Table 4 shows the average ratio of the label size of the VLEI-ABL and VLEI-EOL comparing with that of the C-ORDPATH[2].

As we can see from Table 4, the label size using vlei4–6 and vlei-ABL is about 30% and 24% smaller than that using the C-ORDPATH. Therefore, we consider that the proposed C-DO-VLEI methods consume much lower storage space than the C-ORDPATH.

[2]Here the label size of C-ORDPATH is set to 1

Table 4. Average ratio of each method

vlei-ABL	vlei1	viel2	vlei3	vlei4	vlei5	vlei6
0.7638	0.8221	0.7289	0.7505	0.7015	0.6979	0.6911

7 Conclusions and Future Work

In this paper we have proposed two effective methods, VLEI-ABL, VLEI-EOL for handling the length of C-DO-VLEI code.

We have compared the storage consumption of the proposed methods with that of the C-ORDPATH. The experimental results show that our methods consume less storage space than the C-ORDPATH.

In future, it is necessary to do further experiments to evaluate the performance of the C-DO-VLEI code.

Acknowledgments

This work is partially supported by MEXT of the Japanese Government via Grant-in-Aid for Scientific Research #19024028, the JST of CREST and the TokyoTech 21st Century COE Program.

References

[1] XML Data Repository. `http://www.cs.washington.edu/research/xmldatasets/`.
[2] xmlgen. http://monetdb.cwi.nl/xml/downloads.html.
[3] T. Eda, Y. Amagasa, M. Yoshikawa, and S. Uemura. A robust node-labeling scheme fro xml trees. In *DBSJ Letters,No.1 in Vol.1,2002*, 2002.
[4] M. P. Haustein, T. Härder, C. Mathis, and M. Wagner. Deweyids - The Key to Fine-Grained Management of XML Documents. In *SBBD*, pages 85–99, 2005.
[5] K. Kobayashi, W. Liang, D. Kobayashi, A. Watanabe, and H. Yokota. Update conscious xml labeling methods using dedicated codes. Technical Report TR04-0006, Dept. of Comp. Sci., Tokyo Institute of Technology, 2004.
[6] K. Kobayashi, W. Liang, D. Kobayashi, A. Watanabe, and H. Yokota. VLEI code: An Efficient Labeling Method for Handling XML Documents in an RDB. In *ICDE*, 2005.
[7] S. Murakami, D. Kobayashi, and H. Yokota. The label size and query performance of xml storage using the do-vlei. In *Proc. of DEWS2006.*, 2006.
[8] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML Node Labels. In *Proc. of the ACM SIGMOD International Conference on Management of Data 2004*, pages 903–908. ACM Press New York, NY, USA, 2004.
[9] Online Computer Library Center. Introduction to the Dewey Decimal Classification. `http://www.oclc.org/oclc/fp/about/about_the_ddc.htm`.
[10] A. Schmidt, F. Waas, M. Kersten, and D. Florescu. The XML Benchmark Project. Technical report, Technical Report INS-R0103, `http://monetdb.cwi.nl/xml/index.html`, April 2001.
[11] A. Takahashi, W. Liang, and H. Yokota. Storage Consumption of Variable Length XML Labels Uninfluenced by Insertions. Technical report, IEICE Technical Report vol.107, No.131, DE2007-38, pp.97-102, 2007.
[12] W. H. Xiaodong Wu and, Mong Li Lee and. A Prime Number Labeling Schema for Dynamic Ordered XML Trees. In *ICDE*, pages 66–78, 2004.