동적 순서 XML 트리에서 레이블링 기법을 이용한 효율적인 수정처리

이강우*

An Efficient Updates Processing Using Labeling Scheme In Dynamic Ordered XML Trees

Kang-woo Lee*

요 약

동적인 XML 문서에서 빈번히 발생하는 갱신에 대한 고려를 하지 않는 레이블링 기법들은 XML 문서 트리의 갱신이 일어날 때 마다 변화된 레이블 정보를 반영하기 위해서 전체 XML 트리를 재탐색하여 전체 노드의 레이블을 다시 계산하는 리레이블링 과정이 필요하다. 이러한 리레이블링은 갱신이 빈번히 일어나는 동적인 XML 문서에서는 비용이 상당히 크다는 단점이 있다. 이런 단점을 해결하기 위해 리레이블링 과정이 필요 없는 레이블링 기법으로 프라임 넘버 레이블링 기법이 제안되었다. 그러나 프라임 넘버 레이블링 기법은 문서가 갱신될 때 XML 문서 트리의 노드 간 형제순서를 갱신하는 문제는 고려하지 않고 있다. 이러한 형제순서의 갱신과정은 XML 문서 트리의 많은 부분을 재탐색하고 재기록 하여야 하므로 많은 비용이 필요하게 된다. 따라서 본 논문에서는 XML 문서 트리의 리레이블링과 재계산이 필요 없이 형제순서를 유지할 수 있는 형제순서 값을 갖는 프라임 넘버 레이블링 기법을 제안한다.

ABSTRACT

Labeling schemes which don't consider about frequent update in dynamic XML documents need relabeling process to reflect the changed label information whenever the tree of XML document is update. There is disadvantage of considerable expenses in the dynamic XML document which can occurs frequent update. To solve this problem, we suggest prime number labeling scheme that doesn't need relabeling process. However the prime number labeling scheme does not consider that it needs to update the sibling order of nodes in the XML tree of document. This update process needs much costs because the most of the XML tree of document has to be relabeling and recalculation. In this paper, we propose the prime number labeling scheme with sibling order value that can maintain the sibling order without relabeling or recalculation the XML tree of documents.

키워드

중요 Prime Numbering Scheme, Dynamic XML, Labeling, XML Trees

* 한라대학교 컴퓨터공학과

접수일자 2008. 06. 09

Ⅰ. Introduction

The growing number of XML repositories on the World Wide Web has provided the development of systems that can store and query XML data efficiently. XPath and XQuery are two main XML query languages that express the structure of XML documents as linear paths or twig patterns[1,2,3].

There are two main techniques to facilitate the XML queries, viz. structural index and labeling scheme. The structural index approaches, such as data-guide in the Lore System and representative objects, can help to traverse through the hierarchy of XML, but this traverse is costly. The labeling scheme approaches, such as containment scheme, prefix scheme and prime scheme, require smaller storage space, yet they can efficiently determine the ancestor-descendent and parent-child relationships between any two elements of the XML. In this paper, we focus on the labeling schemes[1].

If the XML is static, the existing labeling schemes can efficiently process different queries. However if the XML is dynamic, how to efficiently update the labels of the labeling schemes becomes to an important research topic[4].

As we know, the elements in the XML are intrinsically ordered, which is referred to as document order, i.e. the element sequence in the XML document. The relative order of two paragraphs in the XML document is important because the order may influence the semantics, thus the standard XML query language require the output of queries to be in document order by default. Hence it is very important to maintain the document order when the XML is updated[5,6].

Some researches have been done to maintain the document order in XML updating. However the update costs of these approaches are still expensive[7,8]. Therefore in this paper we focus on how to dramatically decrease the update cost.

The rest of the paper is organized as follows. Section 2 reviews the related work and gives the motivation of this paper. We propose our improved the prime numbering scheme with sibling-order value in Section 3. The most part of this paper is Section 3, in which we show that the scheme proposed in this paper need not relabel any existing nodes and need not re-calculate any values when updating an ordered node. The experimental results are illustrated in Section 4, and we conclude in Section 5.

Ⅱ. Related works

In this section, we present three families of labeling schemes, namely containment, prefix and prime.

2.1 Containment labeling scheme

The label of a node is assigned three values: [*start, end, level*] (see Fig. 1(a).)[9]. For any two nodes u and v, u is an ancestor of v *iff* $u.start < v.start$ and $v.end < u.end$. In other words, the interval of v is contained in the interval of u. Node u is a parent of and v iff u is an ancestor of v and $v.level - u.level = 1$.

While containment labeling schemes are effective in supporting XML query processing, they cannot handle dynamic updates. Insertion or deletion of nodes into a labeled XML tree may result in a total relabeling of the XML tree. This problem may be alleviated somewhere by reserving enough space for anticipated insertions[10]. However, it is hard to predict the actual space requirements. Thus, relabeling after updates is inevitable for containment labeling schemes which are not suitable for labeling XML documents in update-intensive applications.

2.2 Prefix labeling scheme

The label of a node is that its parent's label concatenations its own label(*self_label*)(see Fig. 1(b).)[4]. For any nodes u and v, u is an ancestor of v iff *label(u)* is a prefix of *label(v)*. Node u is a parent of node v iff *label(v)* has no prefix when removing *label(u)* from the left side of *label(v)*.

DeweyID[10] labels the n^{th} child of a node with an integer n, and this n should be concatenated to its parent's label and the delimiter (e.g. '.') to form the complete label of

this child node. When a node is inserted, DeweyID needs to relabel the sibling nodes after this inserted node and the descendants of these siblings to maintain the document order.

To date, none of the existing prefix labeling schemes can support updates for ordered XML data at low cost. When a new node is inserted into an ordered XML tree, all the existing prefix labeling schemes require a relabeling of the tree.

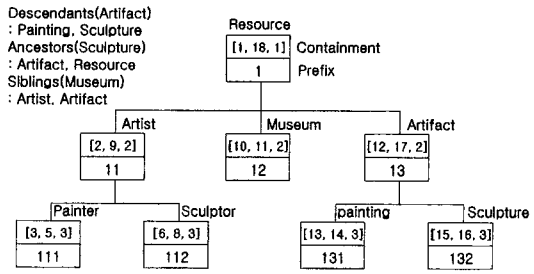

그림 1. 레이블링 기법 : a) 범위기반 b) 전위기반
Fig. 1. Labeling Schemes : a) Containment b) Prefix

2.3 Prime labeling scheme

Wu et al[4] use Prime numbers to label XML trees. The root node is labeled with "1"(integer). Based on the depth first approach, each node is given a unique prime number (*self_label*) and the label of each node is the product of its parent node's label (*parent_label*) and its own *self_label*. For any two nodes u and v, u is an ancestor of *v* iff *label(v)* mod *label(u)*=0. Node *u* is a parent of node *v* iff *label(v)*/*self_label(v)* = *label(u)*.

If an integer A has a prime factor which is not a prime factor of another integer B, then B is not divisible by A. We observe that in XML trees, if a node has a descendant which is not a descendant of another node B, then A cannot be a descendant of node B. Therefore, we can easily determine the ancestor-descendant relationship by using the divisible property of prime numbers. Fig. 2. illustrates the basic prime labeling scheme.

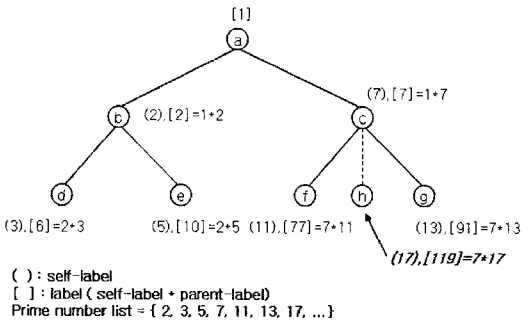

그림 2. 프라임 넘버 레이블링
Fig 2. Prime number labeling

Prime labeling scheme uses the SC(Simultaneous Congruence) values in Chinese Remainder Theorem to determine the document order, i.e. SC mod *self_label* = *document order*. When the document order is changed, Prime only needs to recalculate the SC values instead of relabeling, but the recalculation is much more time consuming[10].

Although prime labeling scheme supports order-sensitive updates without relabeling the existing nodes, it needs to recalculate the SC values based on the new ordering of nodes. The re-calculation is very time consuming.

Ⅲ. Prime labeling scheme with sibling-order value

In the section, we describe the proposed labeling scheme that exploits sibling-order value.

3.1 Sibling-order value

The most important feature of proposed labeling scheme is that we compare labels based on the lexicographical order. The lexicographical order is a way to order words(or strings). we can see their order in a phone book or an English language dictionary. Here is a formal definition of lexicographical order. If we have two words $x=x_1x_2x_3...x_n$ and $y=y_1y_2y_3...y_m$, where x_i's any y_i's stand for letters of these words, n is the length of x and m is the length of y, then

we say that:

x is equal to y (x = y) when

x and y have the same length and all corresponding letters match (example ababa = ababa),

x precedes y (x ≪ y) when

either x is a proper prefix of y (i.e., each letter of x is equal to the corresponding letter of y, and x is strictly shorter than y; abc ≪ abcaaa, but not ab ≪ ab, because ab is not a proper prefix of ab even though ab is a prefix of ab)

or x is not a prefix of y and the first from the left letter on which x and y differ is smaller for x than it is for y (example: acds ≪ acdz, c ≪ d).

Given two binary strings '0011' and '01', '0011' ≪ '01' lexicographically because the comparison is from left to right, and the 2nd bit of '0011' is '0', while the 2nd bit of '01' is '1'. Another example, '01' ≪ '0101' because '01' is a prefix of '0101'.

3.2 Determination of sibling order value

3.2.1 Algorithm for determination of sibling order value

The sibling order value is to determine the SO value of a new node by using the SO value of the previous node prior to the inserted position when the new node is put into the XML document tree. There are three kinds of the SO value in accordance with the inserted position.

[Case 1] Insert a node before the first sibling node(see Fig. 3.).

The Sibling Order(SO) value of the inserted node is that the last bit of the first SO value is changed to "01".

[Case 2] Insert a node at any position among two nodes - left node and right node(see Fig. 4.).

If the size of left node greater than or equal to the size of right node, the SO value of the inserted node is that SO value of left node concatenates "1". Otherwise, the SO value of the inserted node is that we change the last bit of the SO value of the right node into "01".

```
Function DetermineSiblingOrder_Pre(SOR)
// SOR : SO value of the right node
// SON : SO value of the inserted node
begin
  SON = replace(lastbit(SOR), "01");
  //replace() : change the lastbit(SOR) to "01"
  //lastbit() : fetch the last bit of SO value
end
End Function
```

그림 3. DetermineSiblingOrder_Pre 알고리즘
Fig. 3. DetermineSiblingOrder_Pre Algorithm

[Case 3] Insert a node after the last sibling node(see Fig. 5.).

The Sibling Order(SO) value of the inserted node is that we appended the SO value of left node "1".

3.2.2 Prime Numbering method with sibling order value

The scheme for determination of parent-child relationships in the XLM document tree is same to the prime numbering method. The labeling scheme is to add the SO value to the existing label for determining the sibling order. Therefore, the structure of labeling can be represent with [*self_label*, *SO*]. In case b2 is inserted between b1 and b3, *self_label* is 17 (in figure 5, a given prime number is followed by a next prime number in prime number list) and 119 is multiplied by parent's *self_label*(7). The SO value is "01" for SO_L, "0101" for SO_R and "01011" because of $size(SO_L) < size(SO_R)$. Thus when b2 is inserted, the label is [119, 01001]. The determination for parent-child relationships can be understood by figuring out that ⓐ and b2 are in a parent-child relation because the *self_label* of ⓐ, 7, divided by the *self_label* of b2, '119', equals '0'. The sibling order of b1, b2, and b3 is *01 ≪ 01001 ≪ 0101*.

```
Function DetermineSiblingOrder_In(SO_L, SO_R)
 // SO_L : SO value of left node
begin
 if size(SO_L) ≥ size(SO_R) then
  // size() : bit number of SO value
  SO_N = SO_L & "1";
  // & : concatenation operator
 else
  SO_N = replace(lastbit(SO_R), "01");
 end if
end
End Function
```

그림 4. DetermineSiblingOrder_In 알고리즘
Fig. 4. DetermineSiblingOrder_In Algorithm

```
Function DetermineSiblingOrder_Post(SO_L)
begin
 SO_N = SO_L & "1";
end
End Function
```

그림 5. DetermineSiblingOrder_Post 알고리즘
Fig. 5. DetermineSiblingOrder_Post Algorithm

The self_label value is used for the determination of ancestor-descendent relationships while the SO value is used for the determination of the sibling order. The following illustrates the insertion and deletion algorithm for a new node in the XML document tree which is represented with these labels.

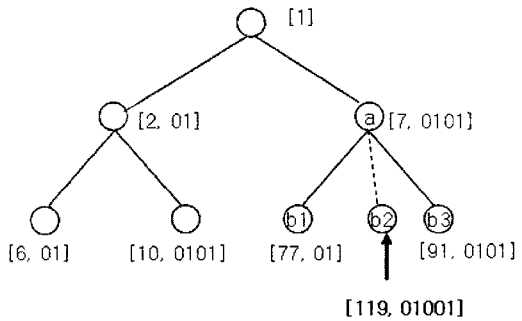

그림 6. 형제순서 값을 갖는 프라임 넘버링 기법
Fig. 6. Prime numbering scheme with sibling order value

If a node is required to delete, the corresponding node can be complete simply by deleting it without recalculating or relabeling the order information like an existing scheme.

Ⅳ. Experiments

Intel Pentium 2.0GHz for cpu, 2Gbytes for memory and Window XP for operating system are required for the experiment environment. The exercising language is Java(JDK1.6.0). SAX 2.0.2(SAX2r3) is used for SML Parser, and Access is for DBMS. The experimental data is Shakespeare 2.0 that the Shakespeare's plays are described through XML and accessible in Web. The experimental data are stored in database with a form of a XML tree that is given labels by SAX parser in terms of stack calculation.

The elements in the Shakespeare's play(D8) are order-sensitive. Here we study the update performance of the Hamlet XML file in D8. The Update performance of other XML files is similar. Hamlet has 5 acts, and we test the following six cases: inserting an act before act[1], inserting an act between act[1] and act[2], ..., inserting an act between act[4] and act[5], and inserting an act after act[5]. Fig. 8. shows the number of nodes for relabeling when applying different schemes. Prefix labeling scheme have the same number of nodes to relabel in all the six cases. The Hamlet XML file has totally 6,636 nodes, but prefix labeling scheme need to relabel 6595 nodes when inserting an act before act[1]. The prime scheme has the number of nodes necessary to recalculate the SC value[4]. The SC value is classified from three labels and experimented. Three labels should be grouped. Because classification of more than four labels may lead a large number that cannot be stored with 64bit in Java. If three labels are grouped, the number of nodes necessary to recalculate in the prime scheme is 1/3 that of nodes necessary to relabel in prefix labeling scheme.

```
/*
struct {
      int label;
      char so[20];
}N; // label of node
Prev_Sibling(N) : return previous sibling node of
            the node N
Next_Sibling(N) : return the next sibling node of
            the node N
get_Label( ) : return the new label
*/
Function Insertion(N)
 begin
  N.label = getLabel();
  if (Prev_Sibling(N) is not exist) then
    next_node = Next_Sibling(N);
    N.SO=DetermineSiblingOrder_Pre
                        (next_node.SO);
  else if (Next_Sibling(N) is not exist) then
    prev_node = Prev_Sibling(N);
    N.SO =DetermineSiblingOrder_Post
                        (prev_node.so);
  else
    prev_node = Prev_Sibling(N);
    next_node = Next_Sibling(N);
    N.SO=DetermineSiblingOrder_In(prev_node.so,
                        next_node.so);
  end if
  end if
end
End Function
```

그림 7. 삽입 알고리즘
Fig. 7. Insertion Algorithm

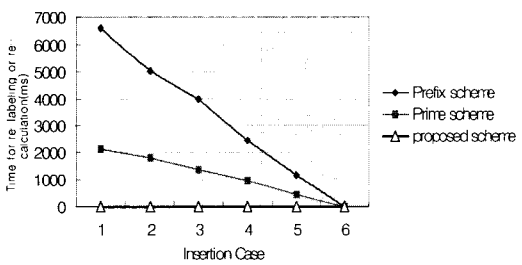

그림 8. 리레이블링과 재계산이 필요한 노드의 수
Fig. 8. Number of nodes or values for relabeling or recalculation

Fig. 9. shows the time for relabeling and recalculating the SC value. The prime scheme (right axis) requires more time than prefix labeling scheme (left axis). But the scheme introduced in this paper no more requires any nodes for relabeling or recalculating in all of the six cases.

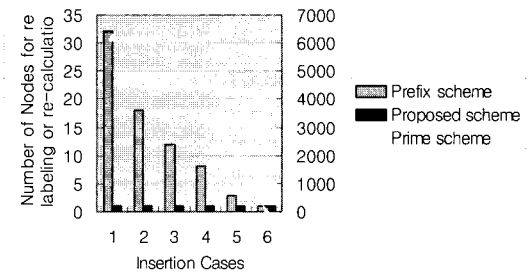

그림 9. 리레이블링과 재계산에 필요한 시간
Fig. 9. Processing time for relabeling or recalculation

V. Conclusion

The objective for designing labeling schemes for XML trees is to allow quick determination of the relationships among the element nodes without actually accessing the XML files. Motivated by the need to efficiently support queries and updates in ordered XML trees, we have developed a prime number labeling scheme with sibling order value.

If an order-sensitive node is inserted in XML document, updating or recalculating of labeling is required to sustain a document order like the existing labeling scheme. In oder to solve such a problem, this paper suggests that if an order-sensitive node is inserted into XML document tree, the prime numbering method with sibling order value is an alternative without relabeling or recalculating of labeling. That is, the sibling order value is given not by using SC table for reducing the maintenance cost of SC table, and problems of prime numbering method are solved by the sibling order value. The efficiency of the method proposed in this paper is demonstrated through experimenting both the existing and the proposed scheme.

As topics for future researches, we may focus on how to reduce the size of label and how to improve the existing query processing scheme through using the method proposed in this paper.

References

[1] V. Christophides, D. Plexousakis, M. Scholl and S. Tourtounis, On Labeling Schemes for the Semantic Web, In WWW, 2003.

[2] W3C Working Draft. XML Path Language (XPath)2.0, November 2002.

[3] D. Chamberlin et. al, SQuery 1.0 : An XML Query Language, W3C Working Draft, 2001.

[4] X. Wu, M. L. lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In Proc. of ICDE, pp66-78, 2004.

[5] S. C. Haw, G. S. V. Radha Krishma Rao. Query Optimization Techniques for XML Databases. IJIT Vol.2 No. 2, pp97-104, 2005.

[6] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, Relational Databases for Querying XML Documents: Limitations and Opportunities, Proc. of the 25th VLDB Conf., 1999.

[7] P. Buneman, S. Davidson, W. Fan, C. Hara, W. C. Tan, Keys for XML, In WWW, 2001.

[8] W. M. Shui, F. Lam, D. K. Fisher, R. K. Wong, Querying and Maintaining Ordered XML Data using Relational Databases, 16th Australasian Database Conf., 2005.

[9] Q. Li, B. Moon, Indexing and Querying XML Data for Regular Path Expressions, Proc. of the 27th VLDB Conf., Roma, Italy, 2001.

[10] Changqing Li, Tok Wang Ling, Min Hu, Efficient Processing of Updates in Dynamic XML Data, http: \\comp.nus.edu.sg.

[11] V. Christophides, G. Karvounarakis, D. Plexousakis, Optimizing Semantic Web Queries using Labeling Schemes, proceedings of WWW2003.

[12] Arthur M. Keller, Jeffrey D. Ullman, A Version Numbering Scheme with a Useful Lexicographical Order, http: \\cs.stanford.edu.

[13] Knuth, D.E., The Art of Computing Programming, Vpl. 3, Sorting and Searching(2nd Edition).

저자소개

이강우(Kang-Woo Lee)

1997년 2월 홍익대학교 전자계산학과(이학박사)

2002년~현재 한라대학교 컴퓨터공학과 교수

※관심분야 : 데이터베이스, 유비쿼터스컴퓨팅