One Practical Range-Based Numbering Scheme for XML Document

December 8, 2003

Abstract

It takes constant time to determine the ancestor relation between two nodes in an XML document when the document is indexed with range-based labeling scheme. This constant time procedure makes range-based labeling scheme very attractive. However, relabeling is unavoidable when arbitrary insertions are allowed, and relabeling can be very time-consuming. Relabeling problem limits the use of range-based labeling scheme in naive XML database management system, and document processing systems where insertions and deletions are frequent.

In this paper, we present one practical algorithm to extend range labeling scheme (extended preorder) for indexing XML documents. Under this new scheme, arbitrary insertions can be accommodated without relabeling. The new method can be viewed as a combination of range-based labeling and prefix-based labeling.

Range allocation methods to improve the performance of the indexing method are also discussed in this paper.

Keywords: XML, Indexing, Path-expression Query, Range-based Labeling

1 Introduction

XML is becoming a standard format for data exchange and document presentation [1, 2] over the Internet. XML documents can be viewed as ordered labeled trees (the parse tree of the XML document). To facilitate XML document processing or query processing in XML databases, each item (node) in an XML document is given a unique label (identifier) for efficient processing, especially path-expression queries used in both XML database queries and XML document transformation. Queries over XML documents usually involve specifying patterns of selection on multiple elements satisfying certain structural relations specified by the axis in XPath [3], among these relations, ancestor relation is the most important one.

An XML document itself does not provide a unique label for its items (elements, attributes, etc.). Tags are used to markup the items, but tags are not unique within a document. Two labeling systems
are currently in use [5], for each item in an XML document, it is assigned:

1. a persistent label (id) by the processor that does not change over time, and
2. a structural (which might change as the document evolves, for example extended preorder labeling proposed in [4] and Index Fabric in [10]) label for indexing.

These two labels serve for different purposes: persistent label is used for uniquely specifying a node, which corresponds to the node test in XPath; while structural label is used for traversing the document (the relation among the nodes in an document), which corresponds to the axis in XPath.

But having two labels for each node is awkward. Various schemes [4, 5] were proposed to integrate persistent label and structural label into one: persistent structural label. This means the structural label assigned for each node does not change as the document evolves (update operations may change the structure of the document).

For example, one persistent structural label was proposed in [5], to integrate the structural index which is not subject to change for arbitrary future insertions. The scheme proposed in this paper also falls into this category, but more straightforward for implementation.

For real applications, it was proposed in [9] that floating point numbers instead of integers are used to represent the range. But the limitations with the existing solutions are they make relabeling less frequent, but still unavoidable.

We are considering two kinds of updates: insertion and deletion at the leaf level. It should be noted that insertions at arbitrary level are not allowed as those operations are much more difficult to handle.

The remaining of this paper is organized as follows. We start by the introduction of one range-based labeling scheme: extended preorder labeling in section 2. Then we talk about extending the algorithm presented in [4] to allow arbitrary insertion in section 3. In section 4, we talk about range allocation method to improve the performance of this labeling scheme. Conclusion remarks are given in section 5.

2 Extended Preorder Labeling: Pros and Cons

Several methods were proposed to index XML data, which can be classified into two broad categories: range-based labeling schemes and prefix-based labeling schemes.

One example of range-based labeling scheme is extended preorder labeling proposed in [4]. Under this scheme, each node x in an XML document is labeled with a pair (order, size), where order denotes preorder and size denotes the size of the subtree rooted at x. The preorder labeling is extended in the sense that the size(x) is upper bound on the numbers of nodes of the subtree rooted at x. For two nodes u and v in the tree, with labels (order(u), size(u)) and (order(v), size(v)) respectively, u
is an ancestor of \( v \) if and only if \( \text{order}(u) < \text{order}(v) \leq (\text{order}(u) + \text{size}(u)) \). Similar ideas were presented in [8], range \((l, r)\) instead of \((\text{order}, \text{size})\) is used to label each node in the document. The equivalent condition to determine the ancestor relation becomes whether or not \( u \) is an ancestor of \( v \) if and only if \([l_v, r_v] \subset [l_u, r_u]\). It is easy to see that these two schemes are essentially the same, but extended preorder labeling takes less extra storage as \( \text{size} \) is smaller than \( l \), but more computation, we need to compute the the right boundary. In the whole paper, we will treat the extended preorder label \((l, s)\) and range label \([l..r]\) the same, where \( r = l + s \).

One disadvantage of the range-based scheme is that it is difficult to accommodate arbitrary insertions. Relabeling is unavoidable even though gaps can be reserved for future insertions, as gaps can be filled by a sequence of insertions. Although we can optimize the range allocation method to make this relabeling infrequent, very short sequences do exist for which relabeling is unavoidable [5].

Although XML documents are viewed as ordered labeled trees, the order among the siblings of one element in the document is not important. As a general rule of relational database, the physical order of the data should not reflect the logical order of the elements. Applying the same idea to XML data and to make the problem easier to handle, we make the following assumptions:

1. The insertion of a new node is appended as the last child of a specific element. When the order between the sibling is significant, we can add an extra element for the element (internally for the system) to specify the order between the siblings.

2. The insertions are restricted to the leaf level (The parent need to be inserted before its children).

Under these two conditions, to get the conditions to necessitate the relabeling, we consider insertion of a node \( y \) as the child of node \( x \) with label \((p_x, s_x)\), whose children are \( c_1, c_2, \ldots c_n \) with labels \((p_1, s_1), (p_2, s_2), \ldots (p_n, s_n)\) from the left to the right, respectively.

As the new node \( y \) will be appended as the last child of \( x \), the label of \( y \) only depends on the label of the last child (right-most) of \( x \), which is \((p_n, s_n)\), and the label of \( x \) itself, which is \((p_x, s_x)\).

It is not hard to verify that the label of \( y \), \((p_y, s_y)\) must satisfying the following conditions:

1. \( p_y > p_n + s_n \): as the node appears to the right of \( c_n \);

2. \( p_y + s_y \leq s_x\): the \( y \) is one child of \( x \), so the range of \( x \) will cover the range of \( y \).

So whenever a new node needs to be inserted, the order label of the node will be fixed by the value of \( p_n + s_n + 1 \), and \((p_n + s_n + 1 + s_y) > (p_x + s_x)\) to make sure the above two conditions are satisfied.

Take the tree view of a segment of XML document in Figure. 1 as an example, when a new node \( x \) is inserted as a child of \( B \), the range available is \([8..12]\), so it is possible to pick 8, 9, 10, 11 or 12 as the order label, but we will pick 8 as we assume that insertions at node \( B \) will be appended after \( x \). So have the smallest order label will leave more range available for future insertions. The size label of
this new node $x$ should be bounded by 8, as the range of this node should be covered by the range of its parent $B[2..12]$.

Similarly, when $x$ is inserted at $C$, the order label is 26, as the range of its right most sibling $H$ is $[23..25]$; the size is bounded by 2, as its range should be covered by its parent $C[13..28]$.

The next question is that if relabeling is unavoidable in the original scheme, what are the necessary changes? The solution is that our label need to “grow”: the idea from the pre-fix labeling.

Let’s look at the properties of the prefix-based labeling scheme first. We know in prefix labeling scheme, for two nodes $u$ and $v$ which are labeled with $L(u)$ and $L(v)$ respectively, $u$ is an ancestor of $v$ if and only if $L(u)$ is a proper prefix of $L(v)$.

One quick example used in [5] for prefix labeling is described as follows:

1. The root is labeled with empty string $\epsilon$;

2. Similarly, for any node with label $L(v)$, we label its first child with $L(v)0$, the second child with $L(v)10$, and the $i$-th child with $l(v)i0$.

Clearly, the above scheme is a correct labeling scheme: for any pairs of nodes $u$ and $v$, $L(v)$ is a prefix of $L(u)$ iff $v$ is an ancestor of $u$. One advantage of using prefix labeling is that arbitrary insertions can be accommodated at the leaf level, relabeling becomes unnecessary.

It should also be noted that $\Omega(n)$ is the lower bound for both range-based labeling and prefix-based labeling schemes in the worst case [5].

For extended preorder labeling, size of node $v$ is an upper bound for the number of descendents rooted at node $v$, and for prefix labeling, the level of a node in an XML tree is defined as the length of the label. In the next section, these two definition will be generalized to accommodate the new labeling scheme.

Let’s look at the differences between range-based labeling and prefix-based labeling. In prefix-based labeling, we determine whether or not one node is the ancestor of the other by determining the labeling of the node is a proper prefix of the other.
3 Extending Range-based Numbering Scheme

We know that extended preorder labeling scheme works perfectly except in the case of insertions, when there is no range available. As discussed in section 2, it is difficult for the existing range-based labeling scheme to accommodate arbitrary insertions. But it is easy for prefix-based labeling scheme to support arbitrary insertions, as the label for the newly inserted node could be generated by taking the label of its parent as the prefix and appending a short id to distinguish it from its siblings. We borrow this idea to handle the situation when relabeling is unavoidable as discussed in section 2.

We will show how this problem can be remedied by adding a new level to the label.

In this new scheme, each node $v$ is labeled with an integer pair prefixed with a sequence of integers:

$$p_1 \ldots p_n(o, s)$$

We call $n$ as the level of the label for node $v$. As will be explained later, $s$ (stands for size) is the number of descendants whose label are at the same level. In the above label $p_n = o$, and $o$ is from $p_1 \ldots p_{i-1}(o_p, s)$, the label of $v$’s nearest ancestor one level up.

**Theorem 1** For any two nodes $u$ and $v$ in an XML document with label $o_1 \ldots (o_n, s_n)$ and $o_1 \ldots (o_m, s_m)$, we claim that $u$ is ancestor of $v$ iff $o_1 \ldots (o_n, s_n)$ is a proper prefix of $o_1 \ldots (o_m, s_m)$ or $o_1 \ldots o_{n-1} = o_1 \ldots o_{m-1}$, and $o_{n-1} < o_{m-1} < o_{n-1} + s_n$.

Let’s consider the scenario when node $x$ is inserted as a child of node $p$. ($p$ is labeled with $(p_1 \ldots p_n(o, s))$, it may have children or not):

1. Based on Condition (1), range is available. The inserted node will be labeled the same way as by the extended preorder labeling (the only difference with the scheme proposed by [4] is that our scheme uses variable-length encoding).

2. Based on Condition (1), no range is available. The inserted node will be labeled with $p_1 \ldots p_n(o_{n+1}, s_{n+1})$. (How to pick the proper $(o_{n+1}, s_{n+1})$ will be discussed in section 4.) When we have this new label, we use the prefix $p_1 \ldots p_n$ to record the ancestor information the same way as the prefix labeling, and use $(o_{n+1}, s_{n+1})$ to create a new pseudo-root of a new subtree, making insertions to this newly created node the same way as insertions handled by extended preorder labeling.

It should be noted that the label generated by the second branch will take the label of its parent as prefix, thus the length of of the new label is longer than that of its parent. In theory, the length of the label is not a constant, it is a small constant in practice as long as the range can be predicated smartly. Also, the program will not enter the second branch very frequently. Each time we enter the second branch, it means that range is not available for the newly inserted node.

Let’s illustrate the algorithm by the following sequence of insertions to get the above XML data segment:
A, B, C, D, E, F, G.

When D is inserted as child of A to the document, range is available as node C with label (4,0) is the right most descendent, but the tree rooted at A can have label up to 8 (calculated by order + size).

But when node F is inserted to the document as child of C, no range is available (size of C is 0, so it may not take any nodes as children in the original extended preorder labeling). We will use the order of C as the prefix, and append a range in the next level.

In Figure 1., we know nodes F and G can not be inserted as the children of node C without relabeling the whole tree in the original extended preorder labeling presented in [4], as the size of C is 0, which means the maximum number of children G has is 0.

Several observations about the above numbering scheme (which are different from the extended preorder labeling scheme presented in [4]) are listed as follows:

1. Arbitrary insertions are allowed;
2. Nodes at the same level do not necessarily have the same length of encoding;
3. Left-right information can only be derived from the labeling when two labels are at the same level.

But as we discussed before, the order between the siblings are insignificant for most applications, this will not be a problem for most applications.

It is easy to see that this new numbering scheme combines the advantages of range-based numbering scheme (test the ancestor relation in constant time) and prefix-based numbering scheme (allow arbitrary insertions).

It should be noted that size here is different from the size used in the original paper in [4]. The size here refers to the upper limit of the descendents that are numbered in the same level (this node can have descendents in the next level). Take Figure 1 as an example: the size of node A is 7, so 7 is the upper bound of the number of descendents at the same level (nodes B, C, D, E, and the reserved range for one child of B, one child of D and one child of A between D and E, which are totaled to 7, but these do not include node F and G).

It is also easy to see that prefix-based method can be viewed as a special case of our method: Inserted node will get \((c, 0)\) prefixed with its parent’s label, where \(c\) is used to denote that this node
is the $c$-th child of its parent. This means we need to go into the second the branch whenever a node is inserted (range is never available).

Another advantage of using the above numbering scheme is that it allows deletion of a node at any place, not necessarily at the leaf level.

Insertion of a new internal node can be handled as long as all the children of the newly inserted nodes are at the same level. We could just use the starting label of the left most child to get the label information, and the information from the right most child to get the size information.

4 Various Performance Issues

In this section, we talk about several factors that affect the performance of this new labeling scheme.

4.1 Schemes to Accommodate Deletions

When deletion operations are allowed, the above labeling scheme works fine except some ranges may become useless (wasting of range space). To handle this problem, we could add a list of ranges available to each node information.

Still take the data segment in Figure. 1 as an example, when $F$ and the nodes in its subtree (if any) are all deleted from the tree, the range of [14..17] will become available for any new children of $C$, so for each node, will will keep a recycling list of these available ranges. Whenever there is a new insertion, the recycling list is examined first to decide whether or not ranges are available. If there are suitable ranges available from the recycling list, the range will be used first.

It should be noted that if deletion is not very frequent, it is not necessary to recycle the ranges.

4.2 Range allocation methods and Performance

Range allocation plays an important role for practical use of the above scheme. As illustrated in the previous section, if we restrict the size of the inserted node is 1, then the resulting labeling of the tree will become a prefix labeling based on the new labeling scheme.

One important requirement for the range allocation algorithm is that it must be simple. Naively, we could use the parent, and right most sibling to get the new range parameters: order and size. At the minimum, we need to get the order based on the right most sibling (if any) or the parent (if there is no sibling) to the order information. The size is restricted by the gap between the order and the parent node’s range.

Based on the minimum order and the maximum size, a suitable order and size should be assigned for the newly inserted node. But what is suitable? One one hand, range should not be wasted, a large range should not assigned to one node having only very few descendents; one the other hand, range should not make the length of the labeling growing fast.
Several practical techniques could be used are:

1. The gap for a node which is closer to the root element in the document should have more gap space;
2. DTD could be used to calculate the minimum size of the subtree rooted at a specific node;
3. Document statistics could be used for predicting the size of the subtree rooted at a specific node;
4. Experimental tuning for specific databases.

Further experiments are needed to improve the efficiency of range allocation methods. But DTD and document statistics have been shown to be very promising [5].

5 Conclusion and Future Work

We presented a new labeling scheme to extend the extended preorder indexing method used in XISS [4] to accommodate arbitrary insertions. The new scheme borrows the idea from prefix-based numbering scheme. Based on the DTD and statistics about the document, we could make the variable length encoding method to achieve the same performance as the fixed encoding while allows arbitrary insertion in a document.

Several potential applications of this methods are:

1. Naive XML database: It would be interesting to integrate this work with the existing system developed by Li and Moon [4]. It would also be interesting to investigate how to use the statistics about the document efficiently.

2. DOM: DOM provides a complete parse tree of an XML document for efficient document transformation. But navigation of the parse tree is not from a single node to another single node in most applications, it is from a set of nodes to another set of node, instead. Range-based labeling which allows arbitrary update operations will be a very good alternative to the current method used for most DOM parsers.


References


