An Efficient XML Index Technique with Relative Position Coordinate

Tackgon Kim, Wooseang Kim
Dept. of Computer Science, Kwangwoon Universicy
Wolkye-dong, Nowon-gu, Seoul, Korea

Abstract: - Recently, a lot of index techniques for storing and querying XML document have been studied so far and many researches of them used coordinate-based methods. But update operation and query processing to express structural relations among elements, attributes and texts make a large burden. In this paper, we propose an efficient index technique with Relative Position Coordinate (RPC). It supports containment queries using SQL statements and data management operations. It does not cause serious performance degradations even if there are frequent update operations because it maintains relative relationship of XML tree. Overall, the performance could be improved by reduction of the number of times for traversing nodes.

Key-Words: - XML Index, Relative Position Coordinate, RPC, Containment Query

1 Introduction

The existing multimedia applications use for the limited area of the simple play for the images, audios, and videos. But multimedia applications have been expanded into services through the complex practical uses of multimedia resources with increases of the explosive internet popularization and various demands of internet users these days. XML[1], which is derived from SGML[2] through recommendations of W3C(World Wide Web Consortium), comes up as a standard language, which exchanges the data through internet and still keeps the interoperability, in recent years.

There are many applications with abundant representation ability in XML languages. It is the trend to use XML for expression of all the data in business application program. Therefore, there are steady efforts for the study to store and retrieve the XML documents in the database efficiently and it needs to provide any methods to preserve not only the contents also structural and positional information in the XML documents.

In this paper, we propose an index technique that expresses relative structural information with RPC using RDBMS. RPC method maintains relative relationship in each nodes in XML tree. We will show RPC method that can efficiently perform storing and querying XML document through the proposed method.

The RPC method uses relative positions for each node which is consisted of parent node's position, an order number as offset on sibling nodes, and identifier of leaf node's full-path. So each node of XML tree can be expressed relative relationship each other. And we will use update operators for efficient XML data management with relative position coordinate.

This paper is organized as follows. In chapter 2, we review related works about index techniques of XML documents. In chapter 3, we introduce an efficient index technique with Relative Position Coordinate. In chapter 4, we present update operations for proposed index technique, and in chapter 5, we explain query examples for containment queries. In chapter 6, we show the performance evaluation between the proposed technique and existing coordinate-based technique. Finally we make a conclusion in chapter 7.

2 Related Works

Index techniques for XML documents are described in a lot of researches. [3] explains position-based indexing and path-based indexing to access XML document by content, structure, or attributes.

In position-based indexing, queries are processed by manipulating the range of offsets of words, elements or attributes. GCL position-based model was proposed in [4]. GCL is based on a data structure called a concordance list, which consists of text intervals called extents. Each extent is described by a start position and length. It is possible to process the query to express containment relation, thanks to the description for the range of a position.

In path-based indexing, the location of words is expressed as structural elements and the paths in tree structures are used for the processing of query. In order to determine the position of a word within a document, it is necessary to construct an encoding of

the path of the element names from the root of the document to the leaf node containing the word. And then, for each word occurrence, the inverted list includes a representation of the path to that word.

In [5], data structure for indexing XML documents based on relative region coordinates is used. Region coordinates describe the location of content data in XML documents. They refer to start and end points of text sequences in XML documents. Region coordinates are adjusted by offsets relative to the corresponding region coordinate of the parent node in the index structure.

In [6][7], new technique based on bitmap indexing was introduced. XML documents are represented and indexed using a bitmap indexing technique. They define the similarity and popularity of the available operations in bitmap indices and propose a method for partitioning a XML document set. 2-dimensional bitmap index is extended to a 3-dimensional bitmap index, called BitCube. They define statistical measurements and correlation coefficient. BitCube proved eminent performance in performance assessment with systems such as existent XQEngine, XYZFind already through the fast search speed.

[8] explains about index graph that changes structure to find a fast route that is used often using data mining method. It has a problem that it must update the index graph every time the query processes fundamentally inaccurate index query.

3 Proposed Index Model with Relative Position Coordinate

In this section, we introduce our proposed index model, and how to process relative position coordinate for index.

3.1 Table schemas for index

We use the table schema in figure 1 in order to store positional and relationship of nodes.

Schemas for relative structural information consist of six tables: ElementType, ElementNode, Path, PathMap, Attribute, Text.

ElementType {etID, etName, level}
ElementNode {docID, etID, enID, parentID, level, offset}
Path {docID, pathID, etID}
PathMap {docID, pathID, enID}
Attribute {docID, pathID, attrName, attrValue}
Text {docID, pathID, textValue}

Fig. 1 Table schemas

The ElementType table indicates information about distinct element nodes in XML document with etID, etName, level fields. etID field is a unique ID of distinct element node, etName is element name, and level is a depth value of XML tree.

The ElementNode table keeps relative position information about each nodes. This table consists of docID, etID, enID, parented, level, offset fields. The docID field is a XML document identifier, etID is a ElementType table's etID value, enID is a element's identifier in a XML document, level is a depth value, and offset is a order number of sibling nodes.

The Path table stores information about full paths from root to leaf. Fields are docID, pathID which is path identifier, and etID which is a leaf element node's etID.

The PathMap table indicates information about how connect each node in ElementNode table by using Path table information. Fields are docID, pathID, enID.

The Attribute and Text tables store each identification and positional information about attribute and text contents of elements.

3.2 Description of index technique with relative position coordinate

To define proposed index technique, we consider an XML document D rooted at r. Let's denote P as a parent node of the tree and C_1, C_2, …, C_n are child nodes of P node.

Definition 1. (Relative Position Coordinate) Relative Position Coordinate method expresses an relative relationship of nodes in XML document. Address of the child node among sibling nodes C_1, C_2, …, C_n of an XML document are combinations of numbers (P, O, L), where P is parent node's enID, O is offset for child node, and L is node's depth in XML tree with root node's level is 0. So, one node has relative information which are relationship of parent node, order number of sibling nodes, and level.

Definition 2. (Offset) Offsets among sibling nodes are given to a sequence order like {0, 1, 2, …, n} as described in figure 2.

Example 1. In case we assume that ElementNode's identifier of parent node P is (1), and the number of child nodes n is (5). The offsets of child nodes C_1, C_2, C_3, C_4, C_5 are {0, 1, 2, 3, 4}. That is, the ElementNode

table's record of child nodes C_1, C_2, C_3, C_4, C_5 in figure 2 are (1,0,2), (1,0,3), (1,0,4), (1,0,5), (1,0,6).

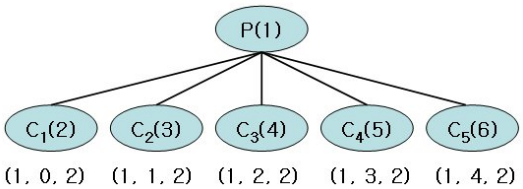

Fig. 2 An example tree and ElementNode table's value list

Definition 3. (Path Identifier, pathID) The pathID field which is a path identifier, is used to identify full path from root node to leaf node. In case we assume that one path from root to leaf with leaf node's level is d, is consisted of N_0 (root node), N_1, …, N_d. In PathMap table, we can use to determine node's relationship as follow.

$$(0 \le enID_{level} \le d) \; AND \; (enID_{pathID} = PathMap_{pathID}) \quad (1)$$

Example 2. In figure 2, parent node P's node identifier is (1), and if C_1, C_2 is same type of element, the pathID is expressed in figure 3 as follow.

node	pathID	etID	docID	enID list
C1	1	1	1	2, 1
C2	2	1	1	3, 1
C3	3	2	1	4, 1
C4	4	3	1	5, 1
C5	5	4	1	6, 1

Fig. 3 An example of pathID

4 Operations for Index Technique with Relative Position Coordinate

We consider update of XML documents with insert, update, remove activities. We use five update operations in figure 4 in order to manage of XML documents with extending the research of [9].

operation	description
InsertBefore (ref, content)	Insert a node before ref sibling node
InsertAfter (ref, content)	Insert a node after ref sibling node
Appen (ref, content)	Insert a node of ref node's child
Remove (ref)	Remove ref node and sub nodes
Update (ref, content)	Update a node's data

Fig. 4 Update operations

4.1 Access of nodes

Access of nodes by proposed index technique is for search of XML document that is processing relevant node ultimately. This paper examined method that can search various containment relations between nodes using simple query language through chapter 5.

4.2 Insertion of a node

Figure 5 and figure 6 show pseudo algorithm for changing index structure from the insertion of a node.

■ InsertBefore | InsertAfter (ref_node, content_node)
1. Get ref_node's information
   (docID, enID, parentID, level, offset)
2. Set inserted node's offset from ref_node
   $content\_node_{offset} = ref\_node_{offset} \pm 1$
3. Compare content_node and all sibling nodes
   If ($sibling\_node_{offset} \ge content\_node_{offset}$)
      $sibling\_node_{offset} = sibling\_node_{offset} + 1$
   Update ElementNode table.
4. Update Path, PathMap table as follows.
   Create a new path value for inserted node.
   Assign path value from ref_node's value except itself

Fig. 5 Pseudo algorithm for insertion operation between sibling nodes

■ Append (ref_node, content_node)
1. Get ref_node's information
   (docID, enID, parentID, level, offset)
2. Set inserted node's offset from ref_node
   $content\_node_{parentID} = ref\_node_{enID}$
3. Check child node existence
   If ref_nod has child node(s)
      $content\_node_{offset} = ref\_node_{NumberofSiblingNodes} + 1$
   else
      $Content\_node_{offset} = 0$
   Update ElementNode table.
4. Update Path, PathMap tables as follows
   Create a new path value for inserted node.
   Assign path value from ref_node's value except itself

Fig. 6 Pseudo algorithm for insertion operation as child node

Example 3. If C_6 node is inserted between C_2 and C_3 as figure 6, pseudo algorithm in figure 4 will assign enID and offset value for C_6 node. and the result is in figure 7 and figure 8 as follows.

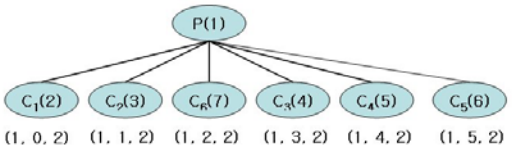

Fig. 7 ElementNode list after insert operation

node	pathID	etID	docID	etID list
C1	1	1	1	2, 1
C2	2	1	1	3, 1
C3	3	2	1	4, 1
C4	4	3	1	5, 1
C5	5	4	1	6, 1
C6	**6**	**5**	**1**	**7, 1**

Fig. 8 Path table after insert operation

4.3 Deletion of nodes

The deletion operation of nodes processes using pseudo algorithm in figure 9.

```
■ Remove (ref_node)
1. Get ref_node's information
   (docID, enID, parentID, level, offset)
2. Find sub nodes (including child, descendent)
   If (ref_node_pathID == subnode_pathID)
   and (ref_node_level < subnode_level)
      Remove sub nodes in the ElementNode, Path,
      PathMap tables
4. Remove ref_node from ElementNode table.
5. Update Path, PathMap tables.
```

Fig. 9 Pseudo algorithm for deletion operation

4.4 Update nodes

Because it is stored the field value as the node's information in table of relational database in proposed index technique, the existing value performs through process that replaces existent value by new value simply at update. The update operation can process with pseudo algorithm in figure 10.

```
■ Update (ref_node, content)
1. Get the ref_node's information
   (docID, pathID)
2. Get the ref_node's value on the Text or Attribute table
3. Get the ref_node's text value or attribute value
4. Change the value of ref_node by text of 'content'
5. Update Text table or Attribute table.
```

Fig. 10 Pseudo algorithm for update operation

5 Query Processing

In this section, we use containment relationship queries[10][11] to evaluate index technique based on relative position coordinate such as direct containment query, indirect containment query, perfect containment query, proximity query. Containment queries are based on containment relationships between elements, attributes, and their contents. These queries are path expressions, Boolean and proximity queries. We use examples to illustrate these queries and then show how to use the proposed index technique and the relational database schemas to process these containment queries.

5.1 Direct containment query

Definition 8. (Direct containment) Direct containment relationship query indicates the query that is consisted of direct containment relationships among elements, attributes, and texts. The symbol '/' indicates direct containment (i.e. parent-child relationship). This query is processed by followed condition.

$$ParentNode_{enID} = ChildNode_{parentID} \text{ AND } ParentNode_{pathID} = ChildNode_{pathID} \quad (2)$$

Example 4. Figure 11 represents the sample SQL statement for expressing direct containment relationship query where 'movie' for parent node, and 'title' for child node.

Query : *movie/title*

```
SELECT  enC.docID
FROM    ElementType etP, ElementNode enP, Path pP,
        ElementType etC, ElementNode enC, Path pC
WHERE   etP.etName = 'movie' and etP.etID = enP.etID and
        etC.etName = 'title' and etC.etID = enC.etID and
        enP.docID = enC.docID and enP.enID = enC.parentID
```

Fig. 11 Direct containment query

5.2 Indirect containment query

Definition 9. (Indirect containment) Indirect containment relationship query indicates the query that is consisted of indirect containment relationship among elements, attributes, and texts. The symbol '//' indicates indirect containment relation (i.e. ancestor-descendent relationship). The query is processed by followed condition.

$$AncestorNode_{pathID} = DescendentNode_{pathID} \quad (3)$$

Example 5. Figure 12 describes the sample SQL statement for searching indirect containment relationship query where 'movie' for ancestor node, and 'title' for descendent node.

Query : *movie//title*

```
SELECT  distinct (pmD.docID)
FROM    ElementType etA, ElementNode enA, Path pA,
        ElementType etD, ElementNode enD, Path pD,
        PathMap pmA, PathMap pmD
WHERE   etA.etName = 'movie' and etA.etID = enA.etID and
        etD.etName = 'title' and etD.etID = enD.etID and
        pmA.enID = enA.enID and pmA.docID = enA.docID and
        pmD.enID = enD.enID and pmD.docID = enD.docID and
        pmA.pathID = pmD.pathID and pmA.docID = pmD.docID
```

Fig. 12 Indirect containment query

5.3 Complete containment query

Definition 10. (Complete containment) Complete containment relationship query indicates the query

that is consisted of complete containment relationship among elements, attributes, and texts. It can be executed to compare the text value of a node and the parameter in the query condition

Example 6. Figure 13 shows the example of complete containment query. It selects all documents that have 'TOPGUN' as text value of a node.

Query : *//title='TOPGUN'*

```
SELECT   Path.docID
FROM     ElementType et, ElementNode en,
         Path p, PathMap pm, Text t
WHERE    et.etName='title' and et.etID = en.etID and
         en.etID = pm.etID and
         pm.pathID = p.pathID and pm.docID = p.docID
         t.textValue = 'TOPGUN' and
         t.pathID = p.pathID and t.docID = p.docID and
```

Fig. 13 Complete containment query

5.4 Proximity query

Definition 11. (Proximity) Proximity query indicates the query that is consisted of two terms and its distance k between them. In this paper, we apply the distance between two terms which is the number of elements between each elements has terms. This query is expressed by followed condition.

$$Distance\ (term1, term2) < k$$
$$Distance = (Ascendent_{level} - Descendent1_{level}) + (Ascendent_{level} - Descendent2_{level}) \quad (4)$$

Example 7. Figure 14 represents the sample SQL statement for expressing proximity query where distance between 'title' node and 'studio' node is less than '5'.

Query : *Distance ('title', 'studio') < 5*

```
SELECT   en2.docID
FROM     ElementType et1, ElementNode en1,
         ElementType et2, ElementNode en2
WHERE    et1.etName = 'title' and et1.etID = en1.etID and
         et2.etName = 'studio' and et2.etID = en2.etID and
         en1.docID = en2.docID and
         (2 * {
         SELECT    max (pm2.level)
         FROM      ElementType et1, ElementNode en1, PathMap pm1,
                   Elementtype et2, ElementNode en2, PathMap pm2,
         WHERE     et1.etName = 'title' and et1.etID = en1.etID and
                   et2.etName = 'studio' and et2.etID = en2.etID and
                   pm1.enID = en1.enID and pm1.docID = en1.docID and
                   pm2.enID = en2.enID and pm2.docID = en2.docID and
                   pm1.docID = pm2.docID and pm1.enID = pm2.enID
         } - en1.level + en2.level) < 5)
```

Fig. 14 Proximity query

6 Performance Evaluation

In this section, we show the comparison between conventional coordinate-based index techniques and proposed index technique. Proposed algorithm reducees the workload to perform the comparison and update operation when it retrieves and indicates data in the relational tables for these indices.

6.1 Comparative operation for containment query

In the conventional coordinate-based index technique, it uses two position values, wicha are relevant to start and end position, to represent the positional region of the node. In the method, it should compare the pairs of start point and end point of the two nodes to check the containment relationship between the nodes. In the proposed index technique, it can perform containment relationship query along the little amount of comparative operation because it compares just containment relationship of pathID between ancestor node and descendent node. Figure 15 shows the difference of comparison times for verifying the containment relationship between conventional coordinate-based index technique and proposed index technique.

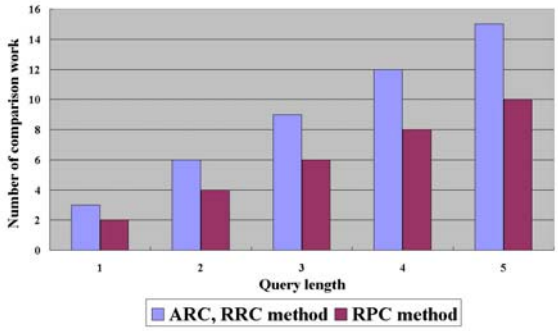

Fig. 15 Comparative operation for indirect containment query

6.2 The number of updated nodes

In the XML tree, according to growing the depth and the width, the number of nodes is to be increased in a geometrical progression. In the situation that update of an offset data affects other nodes by the insertion of a node. In the worst case, all of the position data should be reconstructed and it is caused lots of cost for the operation.

Let XML tree is balanced tree, depth is from 0 to k,

and number of sibling nodes is s, in case using coordinate-based index and proposed index, the accumulative number of the node of which offset data is changed from the insertion and update operations are as follows.

6.2.1 Insert operation on leaf nodes

Figure 16, figure 17 shows the comparison of the number of insert operation per node in accordance with increasing the number of node among the different index techniques. In case of RRC, ARC method, the number of node, which participate in update operation, increase at the ratio of geometric progression based on [5]. But RPC method is better than others because it only change sibling nodes.

$$\text{ARC method} = \frac{1}{2}\prod_{j=1}^{k} s_j \left(\sum_{m=1}^{k}\prod_{j=1}^{m} s_j + k + 2\right) \quad (5)$$

$$\text{RRC method} = \frac{1}{2}\prod_{j=1}^{k} s_j \left(\sum_{j=1}^{k} s_j + k + 2\right) \quad (6)$$

$$\text{RPC method} = s^{k-1} \times \sum_{i=1}^{s} i \quad (7)$$

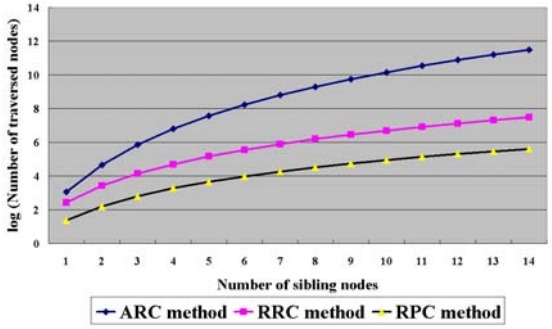

Fig. 16 Insert operation on leaf nodes according to increase the number of sibling nodes with depth is 5

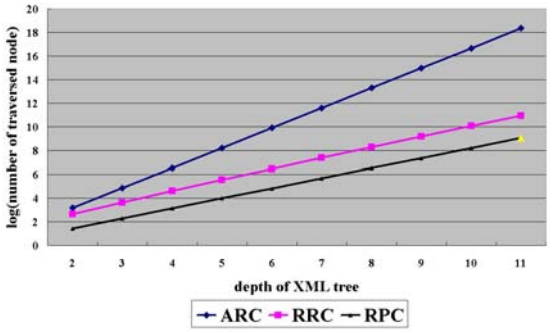

Fig. 17 Insert operation on leaf nodes according to increase the depth of XML tree with number of sibling nodes is 5

6.2.2 Update operation on leaf nodes

Figure 17 shows the comparison of the number of update operation per node in accordance with increasing the number of node among the different index techniques. In case of RRC, ARC methods are same with Insert operation. But RPC method only changes a text field value.

$$\text{ARC method} = \frac{1}{2}\prod_{j=1}^{k} s_j \left(\sum_{m=1}^{k}\prod_{j=1}^{m} s_j + k + 2\right) \quad (8)$$

$$\text{RRC method} = \frac{1}{2}\prod_{j=1}^{k} s_j \left(\sum_{j=1}^{k} s_j + k + 2\right) \quad (9)$$

$$\text{RPC method} = \text{constant} \quad (10)$$

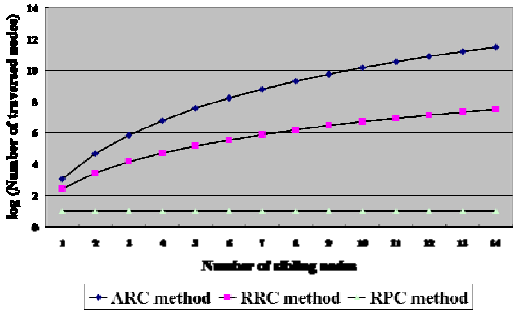

Fig. 18 Update operation on leaf nodes according to increase the number of sibling nodes with depth is 5

7 Conclusion

We proposed an index technique with relative position coordinate to express relative relationship between nodes in a XML document. It is an efficient index technique that simplifies the comparative objects applied to a search query as containment query with relational database.

In addition, we consider update operations to manage XML documents easily, and these operations which are insert and update operations, shows good performance.

References:

[1] T. Bray, et al, "Extensible Markup Language (XML) 1.0 (Second Edition)," http://www.w3.org/TR/2000/REC-xml-20001006

[2] ISO, "Information Processing - Text and Office System - Standard Generalized Markup Language (SGML)," ISO/IEC 8879, Oct. 15, 1986.

[3] R. Davis, T. Dao, J. Thom, J. Zobel, "Indexing documents for queries on structure, content and attributes", In International Symposisium on Digital Media Information Base (DMIB '97), Nov. 1997.

[4] C. Clarke, G. Cormack, F. Burkowski, "An algebra for structured text search and a framework for its implementation," The Computer Journal, 1995.
[5] D. Kha, M. Yoshikawa, S. Uemura, "An XML indexing structure with relative region coordinate," ICDE'2001, April 2001.
[6] J. Yoon, V. Rahgavan, and V. Chakilam, "BitCube: Clustering and Statistical Analysis for XML Documents", 13th International Conference on Scientific and Statistical Database Management, Virginia, July 2001.
[7] J. Yoon, V. Rahgavan, and V. Chakilam, "BitCube: A Three-Dimensional Bitmap Indexing for XML Documents", Journal of Intelligent Information System, Vol.17, pp.241-254, 2001
[8] C. Chung, J. Min, K. Shim, "APEX: An Adaptive Path Index for XML Data", In Proceedings of the ACM SIGMOD International Conference on the Management of Data, 2002:121-132
[9] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld, "Updating XML", ACM SIGMOD, Santa Barbara, California, USA, May, 2001, pp413-424.
[10] C. Cheng, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management system", ACM SIGMOD, 2001.
[11] C. Seu, S. Lee, H. Kim, "An Efficient Inverted Index Technique based on RDBMS for XML Documents", KICS:Database Vol 30, No 1, Feb., 2003.
[12] J. Song, W. Kim, "Extensible index technique for storing and retrieving XML documents", CIT '2004, pp280-287, Sep., 2004.