IMBBTC: XML Document Indexing Model Based on Binary Tree Coding

Zhixin Hu

Nanchang Institute of science and Technology, Nanchang, 330108, China

Keywords: Binary Tree; Encoding; Indexing; trigeminal linked list; Query.

Abstract. In order to facilitate decision relation of nodes, support dynamic updates and improves the speed for XML data query, etc, this paper proposes a XML document indexing structure model based on binary tree encoding. The XML document tree uses trigeminal linked list of binary tree structure to encode nodes. The indexing model of binary sort tree was established, which uses elements of the leaf node as indexing terms, and combines with semantic information of nodes. This paper gives some corresponding algorithms, implements the prototype system of indexing model and its corresponding simulation experiments. Theoretical analysis and experimental results show that the indexing model not only supports update operation of nodes and facilitates decision relation of nodes, but also has advantage of short query time.

Introduction

Establishing the appropriate indexing structure combine with coding technology is one of the commonly used means [1-3]. In order to support and accelerate query processing, people put forward many coding scheme and design adaptive indexing structure. There are three main types of XML indexing: value, path and sequence indexing at present, such as a (k), Data Guide, XISS [4], and so on. It thinks elements and attributes as basic unit of query in XISS indexing, which idea is that separate the complicated paths into the simple paths, and needs join the simple paths. But the process mode is a time-consuming process. The SUPEX builds up indexing by using DTD information. A combined indexing is proposed, which has characteristics of small space overhead and high query efficiency, and can realize hybrid query based on content and text [5]. The above indexing structure is generally based on region code, prefix code and Dewey code. Researches of indexing based on binary tree code are few at present. From this perspective, this paper designs indexing of XML document based on binary tree coding [6], which takes binary tree as XML tree node code by using method of literature. In order to realize need of the XML document optimized query, an indexing of binary sort tree is established by taking leaf nodes as keywords in a particular order, which takes binary tree coding of trigeminal chain as nodes coding of XML tree, combine the document node semantics with characteristics of XML document, which can achieve keywords approximate query.

XML document Coding Schema Based on Binary Tree Traversal

Construction of XML Document Nodes Coding

Firstly, XML document shall be transformed into XML document tree, and then the XML document shall be transformed into binary tree. Each node of the XML document tree are to be numbered by n (n∈natural number, n represents the position information of node in XML tree) in the order top-to-down or down-to-top. Each node are stored by trigeminal linked list structure, namely, each node has left-child point, right-child point and parents point (in order to judge conveniently relationship between nodes and nodes). Construction of node code is achieved by two steps:

Step1: XML document shall be transformed into XML document tree (Using the left child – right brother method).

Step2: The binary tree with tree trigeminal: nodes of inverted the binary tree is by stored with the trigeminal chain, namely, each node of the binary tree includes three pointer fields: left child field, right field and parents pointer fields.

Decision of nodes relation

Property 1. (Decision of father and son / brother): Let v_i, v_j and v_k be any node in the binary tree, if $v_i = v_k->lchild$ and $v_j = v_k->lchild$, then and are brothers, v_i and v_k are father-son or v_j and v_k are father-son.

Property 2. (Decision of the grandfather and grandson): Let v_i, v_j and v_k be any node in the binary tree, if $v_k -> parent = v_j$ and $v_j -> parent = v_i$, then v_i and v_k are the grandfather and grandson.

Indexing Structure Model based on Binary Tree Coding

Thinking of Indexing Design

The element or node information is presented in XML document tree leaf nodes in accordance with characteristic of the XML document and XML document tree. Starting from the root node, it does not query along with a path until the leaf node. From this perspective, this paper constructs indexing structure table of XML document by using indexing technology, based on element leaf node for indexing key item, integrates with semantic information of nodes, and uses algorithm of binary sort trees. The model uses DFS (Depth First Search) thinking to travel XML document. Due to combining the nodes semantic information of the XML document and characteristic of binary sort tree, only some relevant leaf nodes need be traveled and visited. Therefore, by traveling some relevant node, indexing table of binary sort tree can be constructed. Taking the XML document of the Fig.1 for example, corresponding document tree is shown in Fig.2.

```
<? xml version = "1.0" encoding= "ISO-8859-1" ?>
<dblp>
   <paper>
      <title> AI </title><author> Bob </author>
    </paper>
   <paper>
    <title> XML </title><author> Mike </author>
   </paper>
  <paper>
     <title> DB </title><author> John </author>
  </paper>
</dblp>
```

Fig.1: DBLP.XML Document

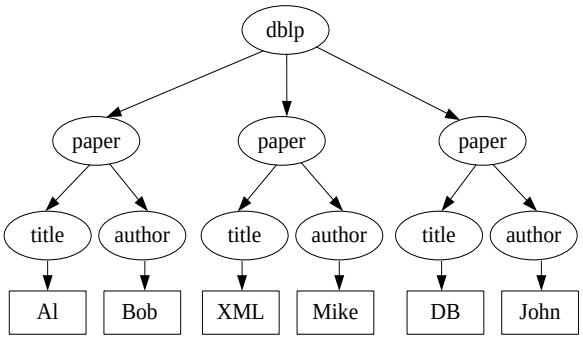

Fig. 2: XML Document Tree

The indexing table is built with the use of the leaf node of element node author as indexing keyword. The binary sort tree is built with depth-first traversal. Firstly, taking leaf node Bob as the root of the binary sort tree, and find the leaf Mike, and take the leaf node Mike compare with root node Bob(in alphabetical order), and take the node Mike as the right-child of the node Bob. In the same way, the XML tree is traveled with depth-first traversal so as to find the other node John, taking it compare with the root node, and find it lies to the right of the root node. Taking the node John compare with node Mike, which lies to the left of it. If there is other node author in the XML document, then follows the think to keep on it. And the ultimate the binary sort tree indexing is shown in Fig.3.

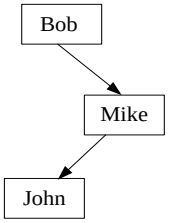

Fig. 3: Indexing Structure of Binary sort tree

If there are some the same element nodes in the XML document, we design the single-linked list or the array to store the element with the same value as author. In the same way, we can also take other element node (such as the node title, etc) as the keyword of index so as to query need of the other index.

Indexing Design and Construction of Query Algorithm

The model chooses some leaf node as indexing keyword, and builds indexing table of binary sort tree. The operation process is shown in algorithm 3. The algorithm of acquiring keyword is shown in algorithm 4.

```
Algorithm 3. Algorithm of building indexing of leaf nodes
Input : Document T and keyword: author ;
Output: Index  of binary sort tree consists of leaf nodes
01: SPXS(BitLeaf T, BitLeaf n_node, BitLeaf p , BitLeaf &q)
02: {  If (T==null) {   q=p; Return 0;}
03. Else If (EQ(T.item , author.item)){ q=T; Return 1;}
04.   Else If(LT(T.item, author.item))
05.     InsertPXS(T->rchlild , n_node , p, q) ;
06.      Else    InsertPXS(T->lchlild , n_node , p, q);
07. }
```

```
Algorithm 4.The algorithm of acquiring keyword.
Input: The document T and n_node.
Output:Return the next node of the author_list
01. GetName(BiLeaf n_node )
02. {  n_node= SPXS(Tree) ;
04. while(1)
05.   { n_node= GetName(author_list ) ;
06.    If(n_node= =0)       break;
07.       Else InsertPXS(T, n_node);
08.   }
09.}
```

Taking the ready for inserted node compared with nodes of the XML document in some order, in certain rule and searching appropriate location. Then taking the node inserts the ruled binary sort tree. The insertion operation is shown in algorithm 5.

```
Algorithm 5.Algorithm of keyword insertion.
Input: Document T and n_node
Output: Indexing table of binary sort tree.
01.  InsertPXS(Leaf &T, Leaf n_node)
02.  { if(SPXS(T, n_node, null , q)== 0)
04.   { if(p= =null)   T=n_node;
 05       else If ( LT(T.item, n_node.item))
06.                P->rclild=n_node ;
07.       else  p->lchild=n_node ;
08.    return 1; }
09.  else If(SPXS(T,n_node,null,q)= = 1)
11. { Add_next_list(T , n_node);   return 1;  }
12.     else  Return 0;
13.}
```

The Experiment Environment and Dataset

i) Data query algorithm. The process of data query is similar to build indexing structure of binary sorted tree. Firstly, comparing the node with the root of indexing table. Secondly, implementing the

next operation in accordance with above comparative result.

Because indexing of the model is a structure of binary sorting tree, average query length of the indexing is related to form of binary tree. When the binary sorting tree is in the worst, the binary tree is built by inserted operation of N nodes in turn. Average query length of the binary sorting tree is the equal of binary tree of single-branch, which its length is N. And for binary sorting tree of single-branch, its average query length is the same as singly linked list, its average query length is (N+1)/2. When the binary sorting tree is in the best, the binary tree is binary sorting tree of asymmetrical forms, which average query length is log (n). And, the time complexities of the binary sorting tree are O (logn) when a user performs inserts, queries, and deletes to the data.

ii) The query example of XML document data. Implementing data query by indexing, its query process is the same as indexing query. Next, the writer takes DBLP.XML of Fig.1 as an example (its XML document tree is shown in Fig.2) and finishes the querying of "John" information. Firstly, comparing node "John" to root " Bob"(by Alphabetical Order), comparative results show that node "John" is on the right of root " Bob", continue traversing at the right sub-tree until the node "Mike" is found. In the same way, comparing node "John" to node " Mike"(by Alphabetical Order), comparative results show that node "John" is on the left of node " Mike", continue traversing at the left sub-tree until all nodes of the indexing are traversed. After execution, the query result is displayed, and related information will be outputted. In addition, according to parent's pointers of node "John", its parent's information can be obtained, and some information of other nodes can also obtained.

The Simulation Experiment and Performance Analysis

The Experiment Environment and Dataset

This paper use DBLP (size: 131MB) as the dataset of experiment test; Hardware: CPU-Intel Pentium Dual 2.0G, RAM 2G; software development tools: JDK1.6 and MyEclipse7.0.

The Secondary XML Coding Rate

The experiment divides DBLP dataset into D1, D2, D3, D4, D5 and D6 documents by means of XML Len tools. The corresponding information of XML document is shown in table1.

Table. 1. Document Information

Document name	Total number of nodes	Number of encoding the two times
D1	725	114
D2	14318	2126
D3	22430	3024
D4	43968	5352
D5	132455	14273
D6	267590	25667

The information of the total number of XML document node and the secondary XML coding rate is shown in figure 4.

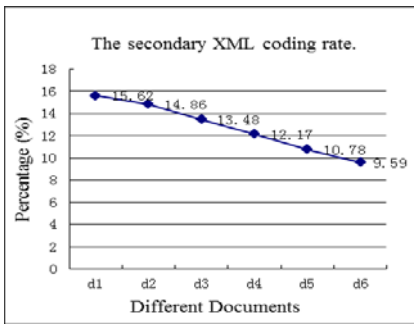

Fig. 4: The Secondary XML Coding Rate

Response Time Experiment

Table1 shows that the number of the secondary XML coding nodes is approximately proportional to the total number of nodes. Fig.4.shows the number of nodes of the XML document more much, but the secondary XML coding rate relatively less. The experiment selects Case1, Case2 and Case3 as test cases from DBLP dataset. The relevant information is shown in table 2.

Table 2. Test Cases

Dataset	Test case	Query path
DBLP	Case1	/dblp/improceedings/line
	Case2	/dblp/article/[author="Jim Gray"]/title
	Case3	/dblp/article/inproceedings/pages

In order to verify the performance of IMBBTC indexing structure, this paper compares IMBBTC indexing with a typical interval indexing XISS. Two kinds of indexing time results are respective 548s (XISS) and 507s (IMBBTC). The simulation experiment of prototype system of indexing model is built, and three groups of relevant test cases were test. Result of query response time is shown in Figure 5.

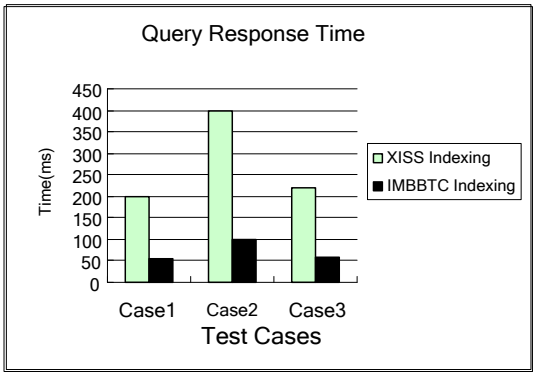

Fig.5: Query Response Time

Performance Analysis.

i) Data updating aspect: The dynamic index tree is order tree because it is constructed in binary sort algorithm. The dynamic index structure can meets the need of data updating of indexing file.

ii) Query speed aspect: The indexing structure is constructed by using leaf node as a keyword, in the certain order of leaf node. No need to traverse the entire XML document query, while to query by corresponding index. To query other node of information by the parent pointer, which can reduce the query operation of the independent nodes and save query processing time, while reduce query space overhead.

iii) Recall: It can improve the accuracy of query process by means of XML language markup information. For example, the above query case, the index structure use leaf nodes as index keyword, and build more indexing files when setting up indexing.

iv) The query response time: The indexing structure of the binary sort tree keeps nodes in order. The query process is similar to two search .Compared with the XISS index, the IBBTC indexing is shorter than the XISS indexing in query response time.

Conclusions

The XML document node is encoded by trigeminal chain structure of binary tree in this paper. The coding has not only short code length, only supports dynamic update operation and convenient in decision of nodes. The indexing model of XML document is constructed based on trigeminal linked list of binary tree. Combining the character of XML document with markup of semantic information of the document nodes, this paper establishes indexing structure model of binary sort tree, which are constructed by using elements of leaf node as indexing terms. The indexing model

can well support the dynamic updating of XML document, facilitate decision relation of nodes, and save query time.

Acknowledgements

The research work is supported by Supported by Program for New Century Excellent Talents in University of China under Grant no.NCET10-0787.

References

[1]Xiafeng Meng. XML Data management: the concept and technology [M]. Beijing:Tsinghua University press 2009, 10.

[2] lingbo Kong, Shiwei Tang, Dongqing Yang, etc. XML Data Indexing Technology [J]. Journal of Software, vol.16, (2005), p: 2063-2079.in press.

[3] Wei Wang, Haifeng Jiang, Hongjun Lu, Jeffrey Xu Yu. PBiTree Coding and Efficient Processing of Containment Join. In Proc. of the 19th International Conference on Data Engineering, India, 2003:251-257.

[4] Li Quanzhong, Bongki M. Indexing and Querying XML Data for Regular Path Expressions[C] //Proc. of the 27th International Conference on Very Large Databases. Roma, Italy: [s.n.], 2001:361-370.

[5] Gaosong Liu, Liyong Wan, Long Jun. Indexing Techniques Based on Inverted Table and B+Tree Combined Structure, Computer Engineering , vol.38,(2012),p:49-52.in press.

[6] Liyong Wan, Yin Chen. CSBTT: An XML Document Coding Schema Based on Binary Tree Traversal [J].Computer System Application, 2013, 22 (02):151-154.